The Design and Implementation of the NDN Protocol Stack for RIOT-OS

Wentao Shang, Alex Afanasyev, Lixia Zhang

UCLA

Presented by Alex Afanasyev

December 8, 2016

Workshop on Information Centric Networking Solutions for Real World Applications (ICNSRA)
ICN/NDN “Edge” for IoT

- Forget about hassle with managing IP addresses
- Bring IoT semantics to the network layer
  - Name the “things” and operations on “things”
    - “temperature in the room”, “humidity on the second floor”
    - “blood pressure”, “body temperature”
    - “max/min/avg pH of soil in specific point of US soil grid”
  - Focus on data associated with things, not devices
    - status information or actuation commands
  - Secure data directly

W. Shang et. al, "Named Data Networking of Things,” in proc. of IoTDI’2016
IoT at the Edge

• Ultra low cost, longevity
  – constrained battery, low-power networking, limited memory, low CPU
  – SAMR21-PRO: 32-bit ARM, 48 MHz, 32KB RAM, 256KB flash

• RIOT-OS: multi-platform light-weight OS
  – https://www.riot-os.org/
  – full C and partial C++ programming environment
  – micro-kernel for multi-threading, priority scheduling, interrupt handling, IPC
  – standard build tools (gcc, make)
  – simulator for testing on Linux PCs
  – gaining a lot of momentum
NDN-RIOT: NDN For RIOT-OS

- Optimized for IoT apps
- Memory efficient packet encoding & decoding
- Data-centric security support
- Basic stateful NDN packet forwarding
- Support for 802.15.4 and Ethernet
- Application API

A few basic examples

Open source, contributions welcome
NDN-RIOT Architecture

NDN app \(\xrightarrow{IPC}\) NDN-RIOT module \(\xrightarrow{IPC}\) Net Device Driver

Threads

RIOT-OS Core

Hardware

CPU
Timer
NIC
Peripherals

ICNSRA’16, Washington, D.C.
Memory-Optimized Packet Decoding

• Shared memory block structure to move packets
  – avoid memory copy in most cases
• On-demand packet field extraction
  – avoid memory for decoded meta data
Security Support

• ECDSA
  – secp256r1 curve with 64-byte signatures
  – deterministic signing (RFC 6979) given lack of good entropy on many current devices
    • keys need to be generated outside the device

• no RSA
  – too much overhead and too expensive to produce signatures

• HMAC
  – RIOT-OS built-in APIs
Packet Forwarding

• PIT
  – exact match for interest
  – “any” prefix match for data (all interests that are prefix of the data)

• FIB
  – longest prefix match for interest names
  – static compile-time prefix registration
  – IPC-based run-time prefix registration (for local apps)

• CS
  – “any” match for interests (a data for which interest is a prefix)
  – compile-time adjustable size (~24KB default settings)
  – FIFO policy

• Work in progress
  – Extendable / adaptive interest forwarding strategy
  – Support for basic Interest selectors
  – Extend dynamic prefix registration and maintenance
L2 Communication

• Run directly over layer 2 interfaces
  – IEEE 802.15.4
    • send packets to FF:FF (broadcast)
  – Ethernet (e.g., debugging on native platform)
    • send packets to FF:FF:FF:FF:FF:FF:FF (broadcast)

• Simple hop-by-hop fragmentation if needed

```
+-----------------+-----------------+-----------------+
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |
+-----------------+-----------------+-----------------+
|1|0|M|   SEQ   |         Identification        |
+-----------------+-----------------+-----------------+
|0|1|2|3|4|5|6|7|8|9|0|1|2|3|
```

ICNSRA’16, Washington, D.C.
Application API

```
ndn_app_create

ndn_app_register_prefix

ndn_app_schedule

ndn_app_run

ndn_app_destroy
```

NDN-RIOT Thread

ICNSRA’16, Washington, D.C.
The NDN protocol is implemented as a kernel thread. SYSTEM device driver threads 100s of KB memory and low-power CPU.

Currently implemented features:

- HMAC-SHA256 data signing and verification
- Memory efficient packet encoding & decoding
- Basic packet forwarding logic (PIT, FIB, CS)
- Support to the constrained IoT devices with 100s of KB memory and low-power CPU.

Named Data Networking (NDN) has shown great potential in supporting network applications. The goal of this project is to bring NDN protocol to constrained IoT devices with Named Data Networking of Things.

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

This demo application shows two RIOT-OS nodes in a emulated network environment on a Ubuntu 15.10 machine. NDN packets are sent over Ethernet interfaces in a emulated network environment on a Ubuntu 15.10 machine. NDN packets are sent over Ethernet interfaces running NDN-Ping client and servers respectively in a emulated network environment on a Ubuntu 15.10 machine. The next step is to try it out on a real IoT device.

This work has been supported by the National Science Foundation under grants CNS-1239385 and CNS-1618439. It is currently released under LGPL v2.1, the same license used by RIOT-OS itself.

The following code snippets are included in the demo application:

```c
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(data, &content);
    // do something with content...
    return NDN_APP_STOP;
}

static int send_interest(void* context)
{
    const char* uri = (const char*)context;
    ndn_shared_block_t* sn = ndn_name_from_uri(uri, strlen(uri));
    ndn_shared_block_t* sin = ndn_name_append_uint16(&sn->block, 0);
    ndn_shared_block_release(sn);
    ndn_app_express_interest(handle, &sin->block, NULL, 1000,
                             on_data, on_timeout);
    ndn_shared_block_release(sin);
    return NDN_APP_CONTINUE;
}

static void run_client(const char* uri)
{
    handle = ndn_app_create();
    ndn_app_schedule(handle, send_interest, (void*)uri, 10000000);
    ndn_app_run(handle);
    ndn_app_destroy(handle);
}
```
Basic Evaluations of the Stack

• Memory overhead on several different platforms for a simple client/producer app
• Processing speed
• Processing overhead
Memory Requirements for Functions

- ndn_app_put_data
- ndn_app_register_prefix
- ndn_app_express_interest
- ndn_app_schedule
- ndn_app_run
- ndn_data_verify_signature
- ndn_data_get_name
- ndn_data_get_content
- ndn_data_create
- ndn_interest_create
- ndn_interest_get_name
- ndn_name_get_size_from_block
- ndn_name_get_component_from_block
- ndn_name_from_uri
- ndn_name_append
- ndn_name_get_component_from_block

ICNSRA'16, Washington, D.C.
Overall Memory Requirements

<table>
<thead>
<tr>
<th></th>
<th>Flash (ARMv6-M)</th>
<th>Flash (ARMv7-M)</th>
<th>RAM (ARMv6-M)</th>
<th>RAM (ARMv7-M)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consumer</strong></td>
<td>35,492</td>
<td>34,092</td>
<td>11,400</td>
<td>11,400</td>
</tr>
<tr>
<td><strong>Producer</strong></td>
<td>35,404</td>
<td>33,992</td>
<td>11,400</td>
<td>11,400</td>
</tr>
</tbody>
</table>

ICNSRA’16, Washington, D.C.
Stack Performance Numbers

ICNSRA’16, Washington, D.C.
Stack Processing Delay

IEEE 802.15.4
MTU: 102 bytes
Fixed data rate: 250 Kbps

Fetching from remote node (generated data)
Fetching from remote cache
Fetching from local cache

0 ms  50 ms  100 ms  150 ms  200 ms  250 ms  300 ms  350 ms

100 bytes
196 bytes
Work in Progress

• Energy consumption evaluation / optimizations
• Advanced forwarding strategy support
  Data discovery
• Nearby data discovery
• Pub-sub API on top of Interest/Data exchange
Use Cases and Other IoT-Related NDN Efforts

• **NDN-BMS**: encryption-based access control

• **NDN-ACE**: authorization framework for actuation apps

• **NDN-IoT**: toolkit for NDN dev on Raspberry Pi
  – [https://github.com/remap/ndn-pi](https://github.com/remap/ndn-pi)

• **NDN on Arduino**: minimal app for Arduino
  – [https://github.com/ndncomm/ndn-btle](https://github.com/ndncomm/ndn-btle)