Named Data Networking of Things: NDN-RIOT, NDN-PI

Alex Afanasyev
Florida International University

IoT over ICN Tutorial @ ACM ICN 2017
September 26, 2017
Berlin, Germany
Complexity and Semantic Mismatch for IP/IoT

• **App:** “Living room frontal view feed”

• **Network:**
  • Request stream (HTTP/CoAP)
  • Connect to camera (TCP/IP)

• +
  • Lookup mapping “Living room” -> camera URI
  • Connect to AlexHome.com (cloud?) service
  • DNS lookup IP of AlexHome.com service
  • DHCP to assign IP addresses to all devices
Named Data Networking of Things

• App: “Living room frontal view feed”
  • /AlexHome.com/LivingRoom/VideoFeed/FrontView/mp4/_frame=12/_chunk=20

• Network:
  • Use the name to send request to my camera responsible for Living’s room front view
  • OR retrieve data from caches
  • +
    • Cameras provision with “identity name” that defines what they are and what data names they produce
    • Can announce name prefixes or respond to local broadcasts
ICN/NDN “Edge” for IoT

- Bring IoT semantics to the network layer
- Name the “things” and operations on “things”
  - “Living room frontal view feed”, “CO level in kitchen”
  - “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
- Secure data directly

W. Shang et al., “Named Data Networking of Things,” in proc. of IoTDI’2016
Smarter IoT with Low-cost Devices

• Hardware: ultra low cost, longevity
  • Constrained battery, low-power networking, limited memory, low CPU
  • ~ 32-bit ARM, 48 MHz, 32KB RAM, 256KB flash

• Application getting smarter and more powerful
  • Need integration with public Internet and cloud service without requiring gateways
  • Need for data-centric security, local trust management
  • Need auto-discovery and auto-configuration

• Named Data Networking
  • common protocol for all applications and network environments

NDN-RIOT Architecture Highlights

- Support for NDN packet format for limited MTU links
- Support of data-centric security, including ECDSA and HMAC signatures, AES encryption
- Replaceable forwarding strategies
- Support of transmission (+fragmentation) over IEEE 802.15.4 and Ethernet
- Simple application API
- A few basic examples

Open source, contributions welcome
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
  • micro-ecc library (https://github.com/kmackay/micro-ecc)
  • 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

- Replaceable forwarding strategies

- Work in progress
  - 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

![Fragmentation Diagram]
Application API

ndn_app_create

ndn_app_schedule

ndn_app_register_prefix

ndn_app_put_data

ndn_app_express_interest

ndn_app_run

ndn_app_destroy

NDN-RIOT Thread

IoT over ICN Tutorial @ ACM ICN 2017
The source code of this work is available at
http://www.riot-os.org/

Currently the code is only tested in emulated
environments. The next step is to try it out
on a real IoT device.

This demo application shows two RIOT-OS nodes
connected through the NDN-Demo router
and an Ubuntu host NDN server.

The code is also available as a simple
NDN consumer on RIOT-OS

The current implementation does not have
all the advanced features of the
NDN architecture.

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

The NDN protocol is implemented as a kernel thread.

Support for Ethernet and 802.15.4

Currently implemented features:
- Basic packet forwarding logic (PIT, FIB, CS)
- HMAC-SHA256 data signing and verification
- Send configuration commands (e.g., add faces,
  100s of KB memory and low-power CPU.

Named Data Networking (NDN) has shown
great potential in supporting network applica-
tions in the IoT environments [1].
Getting Started with NDN-RIOT Examples

- **Downloading**
  - mkdir riot
  - cd riot
  - git clone https://github.com/named-data-iot/RIOT
  - git clone https://github.com/named-data-iot/ndn-riot
  - git clone https://github.com/named-data-iot/ndn-riot-examples

- **Compiling an example**
  - cd ndn-riot-examples/<APP>
  - For host architecture (for debugging)
    - make
  - For a specific RIOT board
    - make BOARD=samr21-xpro
    - make flash BOARD=samr21-xpro # to flash firmware
    - make term BOARD=samr21-xpro # to access board via serial interface

ndn-benchmark
ndn-consumer
ndn-ping
ndn-producer
ndn-rtt
ndn-template
### Stack Performance Numbers

<table>
<thead>
<tr>
<th>Operation</th>
<th>IoTLab-M3 (time)</th>
<th>IoTLab-M3 (power)</th>
<th>SAMR21-XPRO (time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get Data Content</td>
<td>4 µs</td>
<td>4 µs</td>
<td>291 µs</td>
</tr>
<tr>
<td>Get Data Name</td>
<td>2 µs</td>
<td>29 nJ</td>
<td>12 µs</td>
</tr>
<tr>
<td>Verify Data (ECDSA)</td>
<td>3 µs</td>
<td>294,225 µs</td>
<td>451,215 µs</td>
</tr>
<tr>
<td>Create Data (ECDSA)</td>
<td>4 µs</td>
<td>269,344 µs</td>
<td>451,215 µs</td>
</tr>
<tr>
<td>Create/Verify Data (HMAC)</td>
<td>2 µs</td>
<td>1,333 µs</td>
<td>16,000 nJ</td>
</tr>
<tr>
<td>Get Interest Name</td>
<td>2 µs</td>
<td>24 nJ</td>
<td>1,908 µs</td>
</tr>
<tr>
<td>Create Interest</td>
<td>25 µs</td>
<td>221 nJ</td>
<td>1,908 µs</td>
</tr>
<tr>
<td>Append to Name</td>
<td>29 µs</td>
<td>251 nJ</td>
<td>1,908 µs</td>
</tr>
<tr>
<td>Get Name component</td>
<td>28 µs</td>
<td>69 nJ</td>
<td>1,908 µs</td>
</tr>
<tr>
<td>Get Name size</td>
<td>11 µs</td>
<td>92 nJ</td>
<td>1,908 µs</td>
</tr>
<tr>
<td>URI to Name</td>
<td>13 µs</td>
<td>1,366 nJ</td>
<td>1,908 µs</td>
</tr>
</tbody>
</table>

A sensing app can create, sign, and transmit one data packet, every minute for half a year on a single battery charge.
IEEE 802.15.4
MTU: 102 bytes
Fixed data rate: 250 Kbps

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

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

100 bytes
196 bytes
Other IoT-Related NDN Efforts

• NDN-BMS: encryption-based access control

• NDN-ACE: authorization framework for actuation apps

• NDN on Arduino: minimal app for Arduino
  • https://github.com/ndncomm/ndn-btle
  • https://redmine.named-data.net/projects/ndn-embedded/wiki
Options to bring NDN stack to Raspberry Pi 1/2/3

• If running actual Ubuntu repo
  • Can you PPA repository to install pre-compiled binaries
    • sudo apt-get install software-properties-common
    • sudo add-apt-repository ppa:named-data/ppa
    • sudo apt-get update
    • sudo apt-get install nfd

• If running Raspbian
  • Install from source on Pi itself (could be slow)
  • Install using cross-compilation
    • Try https://gitlab.com/named-data/docker-raspberry-pi-ndn-cross-compiler docker app
    • It is still in progress and needs feedback