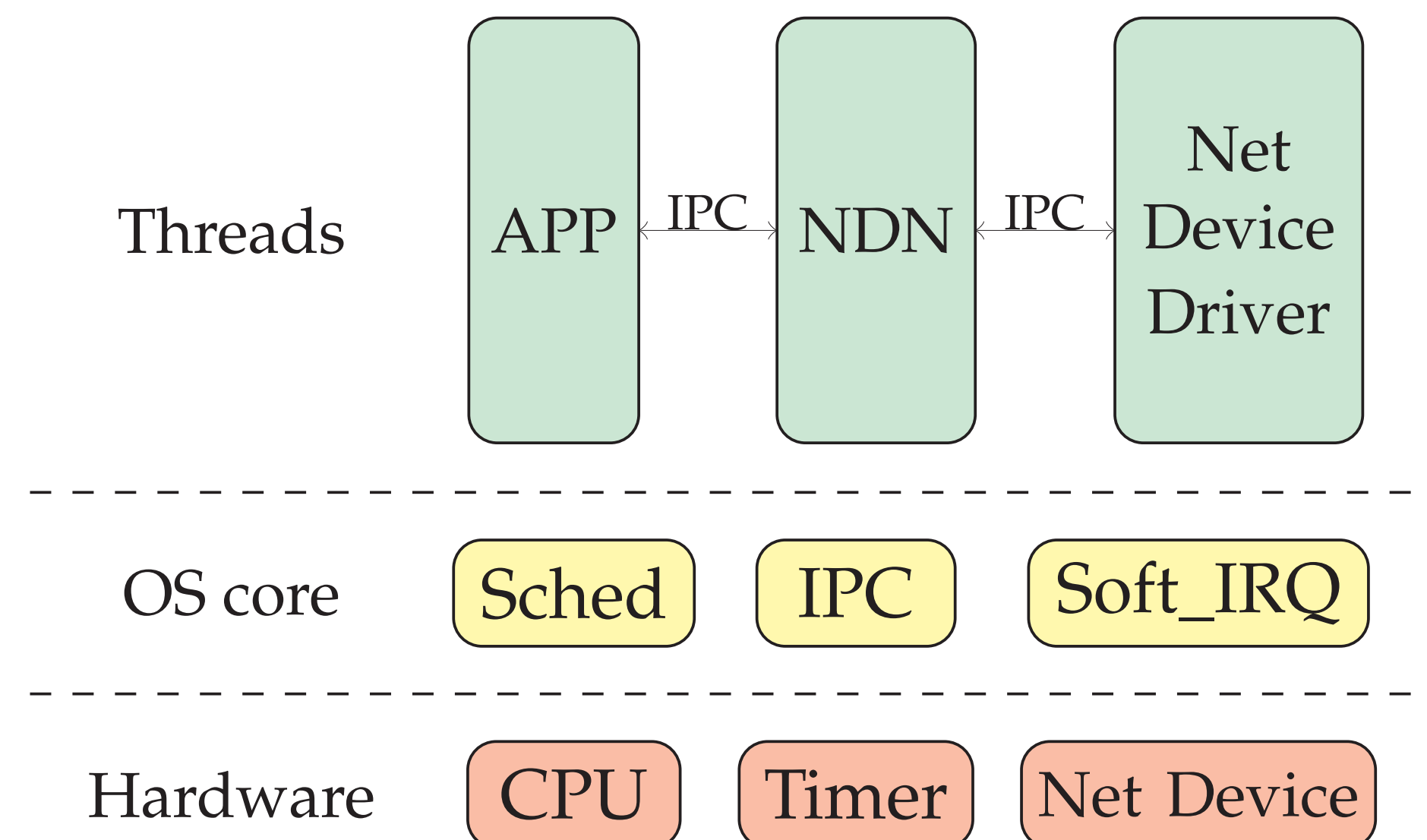


MOTIVATION

- 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].

SYSTEM DESIGN



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

REFERENCES

- [1] W. Shang, A. Bannis, T. Liang, Z. Wang, Y. Yu, A. Afanasyev, J. Thompson, J. Burke, B. Zhang, and L. Zhang. Named Data Networking of Things. In *Proceedings of 1st IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI'2016)*. (Invited paper).
- [2] RIOT - The friendly Operating System for the Internet of Things. <http://www.riot-os.org/>.

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

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_nonce, ndn_interest_get_lifetime
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(data, &content);
    // do something with content...
    return NDN_APP_STOP;
}

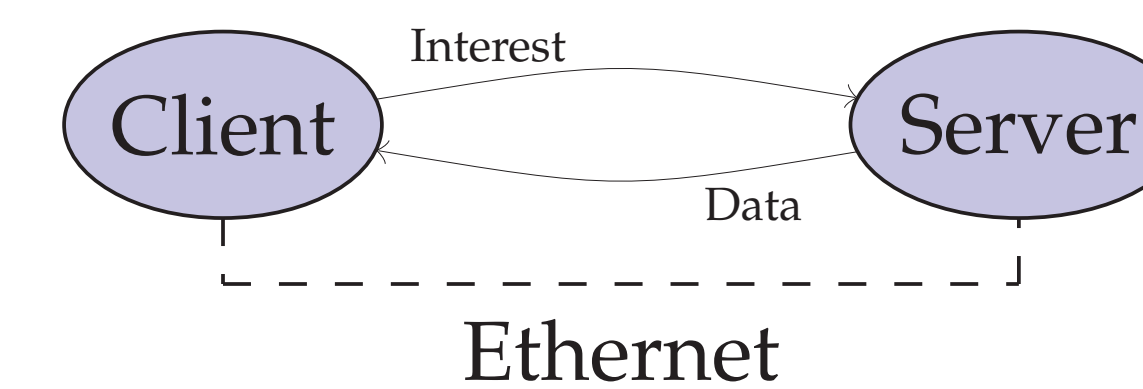
static int send_interest(void* context)
{
    const char* uri = (const char*)context;
    ndn_shared_block_t* sn = ndn_name_from_uri(uri, strlen(uri));
    ndn_shared_block_t* sin = ndn_name_append_uint16(&sn->block, 0);
    ndn_app_express_interest(handle, &sin->block, NULL, 1000,
        on_data, on_timeout);
    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, 1000000);
    ndn_app_run(handle);
    ndn_app_destroy(handle);
}

Simple NDN consumer on RIOT-OS
```

DEMO 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.



Emulated testbed

```
riot@riot-dev:~/RIOT/examples/ndn_ping
server (pid=2): send data to NDN thread, name=/a/%X0%04%06%F8/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/b%01y%3B
server (pid=2): send data to NDN thread, name=/a/b%01y%3B/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%87%28%K7
server (pid=2): send data to NDN thread, name=/a/%87%28%K7/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%D0%04%2C%F0
server (pid=2): send data to NDN thread, name=/a/%D0%04%2C%F0/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%F2%7F%02
server (pid=2): send data to NDN thread, name=/a/%F2%7F%02/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%1D%0E%DC
server (pid=2): send data to NDN thread, name=/a/%1D%0E%DC/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%FE%F3%06
server (pid=2): send data to NDN thread, name=/a/%FE%F3%06/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%5C%12%9D%5B
server (pid=2): send data to NDN thread, name=/a/%5C%12%9D%5B/%00
server (pid=2): return to the app
server (pid=2): Interest received, name=/a/%80%DC%84%7F
```

NDN-Ping server

```
riot@riot-dev:~/RIOT/examples/ndn_ping
client (pid=2): data received, name=/a/%D0%04%2C%F0/%00
client (pid=2): content=1524541085
client (pid=2): in sched callback, count=23
client (pid=2): express interest, name=/a/%F2%7F%02
client (pid=2): data received, name=/a/%F2%7F%02/%00
client (pid=2): content=4056830695
client (pid=2): in sched callback, count=24
client (pid=2): express interest, name=/a/%1D%0E%DC
client (pid=2): schedule next interest in 2 sec
client (pid=2): data received, name=/a/%1D%0E%DC/%00
client (pid=2): content=138113537
client (pid=2): in sched callback, count=25
client (pid=2): express interest, name=/a/%FE%F3%06
client (pid=2): schedule next interest in 2 sec
client (pid=2): data received, name=/a/%FE%F3%06/%00
client (pid=2): content=407257839
client (pid=2): in sched callback, count=26
client (pid=2): express interest, name=/a/%5C%12%9D%5B
client (pid=2): schedule next interest in 2 sec
client (pid=2): data received, name=/a/%5C%12%9D%5B/%00
client (pid=2): content=1233608294
client (pid=2): in sched callback, count=27
client (pid=2): express interest, name=/a/%80%DC%84%7F
```

NDN-Ping client

```
13:24:49.906719 46:09:59:b4:60:1f (out Unknown) Unknown SSAP 0x14 > Broadcast SN
A Unnumbered, 07, Flags [Response], length 23
0x0000: ffff ffff ffff 4609 59b4 601f 0123 0515
0x0010: 0700 0801 6108 04fe f367 a60a 0405 9768
0x0020: 1f0b 0203 08
13:24:49.907115 4e:ef:5b:bd:85:d6 (out Unknown) Unknown SSAP 0x40 > Broadcast IP
Unnumbered, 07, Flags [Command], length 66
0x0000: ffff ffff ffff 4eef 5b5d 85d6 0123 0640
0x0010: 070c 0801 6108 04fe f367 a608 0100 1403
0x0020: 1801 0015 0466 5e87 4910 031b 0104 1720
0x0030: a2b5 5d15 9981 891b 575b 1381 675b 5614
0x0040: 65c 35db c196 8a77 c440 1e96 08cc bcae
13:24:51.907684 46:09:59:b4:60:1f (out Unknown) Unknown SSAP 0x14 > Broadcast SN
A Unnumbered, 07, Flags [Response], length 23
0x0000: ffff ffff ffff 4609 59b4 601f 0123 0515
0x0010: 0700 0801 6108 04fc 129d 5b0a 04c1 781d
0x0020: 1f0b 0203 08
13:24:51.908041 4e:ef:5b:bd:85:d6 (out Unknown) Unknown SSAP 0x40 > Broadcast IP
Unnumbered, 07, Flags [Command], length 66
0x0000: ffff ffff ffff 4eef 5b5d 85d6 0123 0640
0x0010: 070c 0801 6108 04fc 129d 5b08 0100 1403
0x0020: 1801 0015 0466 5e87 4910 031b 0104 1720
0x0030: 6a8b 39c2 7a9f 4870 7905 7eac 0516 c043
0x0040: b1ce 2daf e036 00d7 9e88 3ee7 43e1 daaa
```

Tcpdump output of network packets

text	data	bss	dec	hex	file name
39636	228	11204	51068	c77c	ndn_ping.elf

Code size & static memory usage (compiled for SAM R21 IoT board)

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.