CCN-lite, PiCN, PyCN-lite – a growing family

CCN-lite 2011–2016: C, NDN+CCNx+, hackish, no tests

CCN-lite since 2016: still C! In very good shape (integration tests),
has moved closer to RIOT

PiCN since 2017: Python, finally.
Solid (=heavy?) software engineering, see the following slides

PyCN-lite since 2018: Python, but lite and hackish.
see the following slides, too

A) PiCN – Python ICN

https://github.com/cn-uofbasel/PiCN

ICNRG interim meeting, UCL
March 18, 2018

Chris Scherb
Claudio Marxer
(Christian Tschudin)

PiCN Project Goals

Prototyping-friendly, extensible **library** and a set of **tools**.

+ Multi packet format
+ NDN/CCN-core logic and data structs
+ Advanced: NFN, FLIC, security protocols, ...

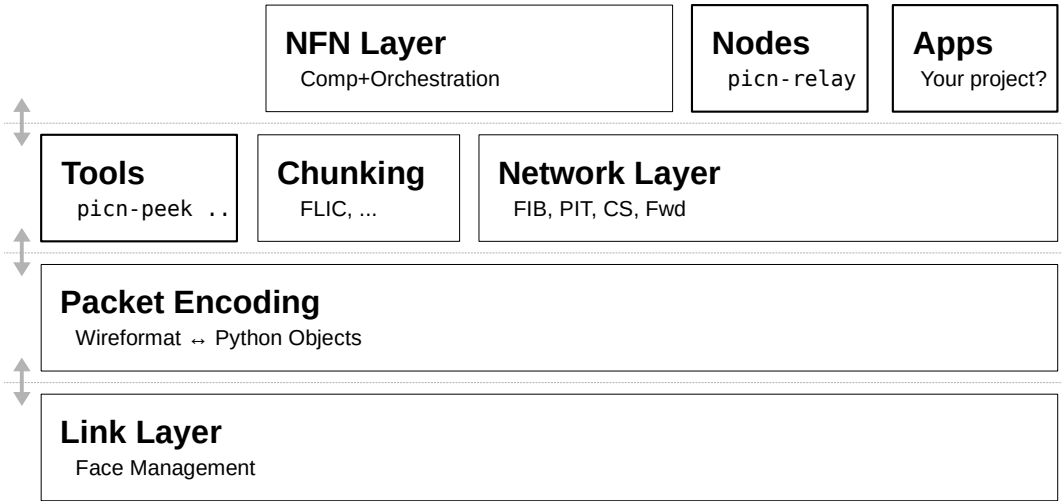
– Python 3.6+
– BSD-3-Clause

– Layered/stacked architecture
– Modular structure within each layer

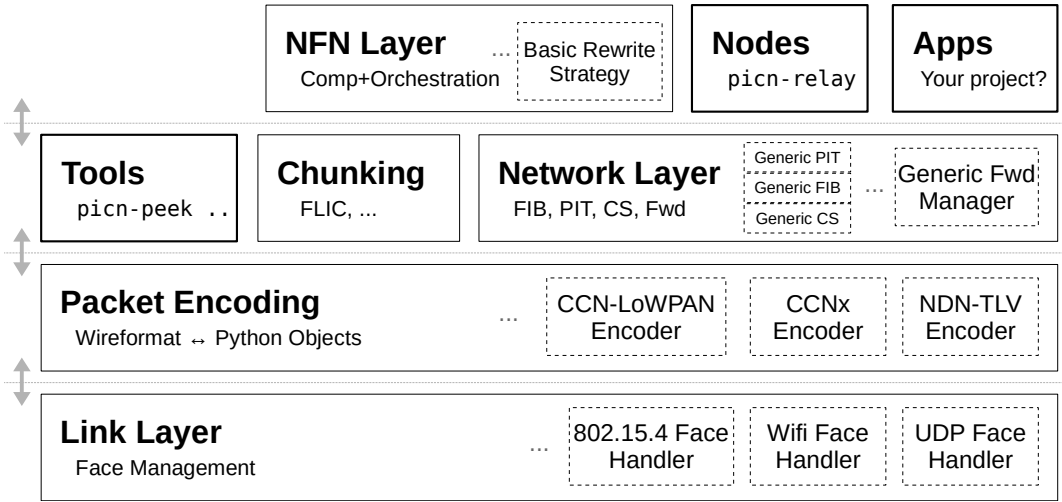
– Well-defined interfaces...
  .. between layers.
  .. for modules.

  ⇒ Plug-in custom modules or replace entire layers.

PiCN Architecture



PiCN Architecture



PiCN Software Status

Lines: 6000+
Tests: 229

Nodes & Tools

- ✓ `picn-relay`
- ✓ `picn-peek`
- ✓ `picn-mgmt`
- ✓ `picn-setup`
- (✓) Next-gen NFN
- ✗ FLIC repo

Security

- ✗ PKI, Signing, Verification
- ... this is a playground!

Link Layer

- ✓ UDP
- ✗ WebSocket
- ✗ IEEE 802.11, 802.15.4

Packet Encoding

- (✓) NDN + NDNLPv2
- ✗ CCNx
- ✗ CCN-LoWPAN

Network Layer

- ✓ Basic forwarding logic
- ✗ Improved forwarding strategies

B) PyCN-lite – MicroPython ICN

ICNRG interim meeting, UCL
March 18, 2018

Christian Tschudin

PyCN-lite Project Goals

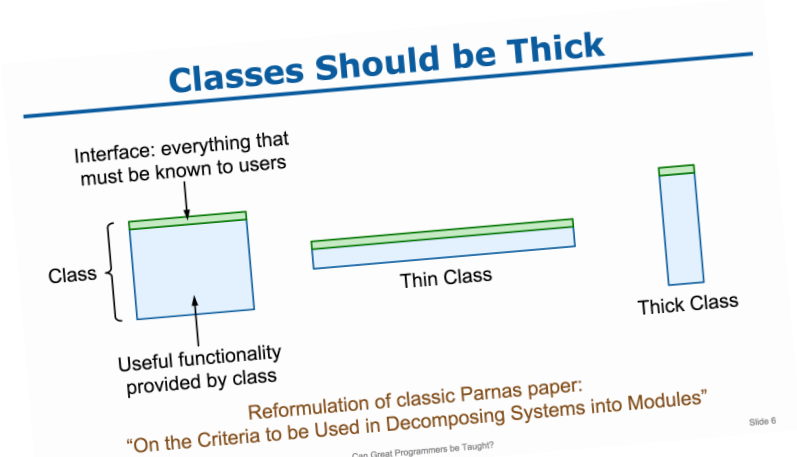
LITE, LITE, LITE! IoT-friendly despite Python

- Fast develop-debug cycles for
  higher-layer experiments (RPC, FLIC)
- Avoid bleeding edge (Python3.6 in PiCN):
  PyCN-lite uses MicroPython and plain Python3
- Avoid classitis: “OS-style” code:
  better few+deep classes than many+thin ones ...

PyCN-lite Architecture – comparison to PiCN

Software architecture? Let the classes speak. Class count:

- PiCN (NFN, tests, only NDN) > 150
- PyCN-lite (no NFN, no tests) < 40



(John Ousterhout, Stanford)

PyCN-lite on a constraint IoT device (NodeMcu)

ICN-enabled IoT devices

PyCN-lite forwarder

ESP8266, price < USD 5

uplink to the NDN testbed **(WiFi, IP)**

- NodeMcu/ESP8266: 96KB RAM, 28KB left for apps
- dual WiFi ! uplink, and access point
- MicroPython (subset of Python3)
- sufficient RAM to run a NDN fwd (no CS) **and a repo**

PyCN-lite Software Status

Lines: 3500+
Tests: 0

Nodes & Tools

- ✓ `pkt dump`
- ✓ `fetch (peek)`
- ✓ `repo`
- ✓ `fwd`
- ✓ `FLIC`

Link Layer

- ✓ UDP
- ✗ raw 802.11, 802.15.4

Packet Encoding

- ✓ NDN
- ✓ CCNx
- ✓ CBOR
- ✗ CCN-LoWPAN
- ✓ sexpr for marshalling
  (currently for NDN and CBOR)

Transport

- ✓ bidirectional streaming RPC

Contributions, testing, feedback, questions are highly welcome!

- https://github.com/cn-uofbasel/CCN-lite
- https://github.com/cn-uofbasel/PiCN
- https://github.com/cn-uofbasel/PyCN-lite

Backup Slide: RPC (work in progress)

Q: Where is the "publish" method?

It's crucial/urgent/essential/important to provide **publish()** functionality, even before forwarding (think "IoT device wanting to persist its measurements")

- Layer the `publish()` method on top of a generic RPC (instead of point-solution)
- Get inspired by service definitions in gRPC:

```
service ICNnode {
 rpc lookup(stream Interest) returns (stream Data);
 rpc traverseFLIC(Name)      returns (stream Data); # edge computing!
 rpc publish(stream Data)    returns (stream Retcode);
 rpc resolve(NFNexpr)        returns (stream Manifest);
}
```

- All packets (will be) encrypted: setup/handshake copied from HIP (RFC 7401)