CICN
Community
Information-Centric Networking

Tutorial at ACM SIGCOMM ICN, Berlin, Germany
26th of September 2017
Tutorial agenda

• Project overview
• Vector Packet Processing
• vICN: automation of virtual ICN network deployment
• The consumer/producer socket API with applications to HTTP
CICN project overview

• CCNx Internet documents are specified at the ICNRG define the architecture.

• The rest is just software development, testing and experimentation.

• Focus on VPP and application development:
  • Vector Packet Processing as the Universal Data Plane for vRouting and vSwitching
  • vICN automation of virtual networks deployment
  • The Consumer/Producer Socket API and HTTP
What is FD.io (pronounced “fido”)?
FD.io: The Universal Dataplane

- Project at Linux Foundation
  - Multi-party
  - Multi-project
- Software Dataplane
  - High throughput
  - Low Latency
  - Feature Rich
  - Resource Efficient
  - Bare Metal/VM/Container
  - Multiplatform

Fd.io Scope:
- Network IO - NIC/vNIC <-> cores/threads
- Packet Processing – Classify/Transform/Prioritize/Forward/Terminate
- Dataplane Management Agents - ControlPlane
Fd.io in the overall stack

Application Layer/App Server

Orchestration

Network Controller

Data Plane Services
- Dataplane Management Agent
- Packet Processing
- Network IO

Operation System

Hardware

vICN

Linux
Multiparty: Broad Membership

Service Providers
- AT&T
- Comcast

Network Vendors
- Cisco
- Ericsson
- Huawei
- ZTE
- MetaSwitch
- Brocade

Chip Vendors
- Intel
- Cavium Networks

Integrators
- Red Hat
- Inocybe
Multiparty: Broad Contribution
Code Activity

• In the period since its inception, fd.io has more commits than OVS and DPDK combined, and more contributors than OVS

<table>
<thead>
<tr>
<th></th>
<th>Fd.io</th>
<th>OVS</th>
<th>DPDK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commits</td>
<td>6283</td>
<td>2395</td>
<td>3289</td>
</tr>
<tr>
<td>Contributors</td>
<td>163</td>
<td>146</td>
<td>245</td>
</tr>
<tr>
<td>Organizations</td>
<td>42</td>
<td>52</td>
<td>78</td>
</tr>
</tbody>
</table>

![Bar charts showing commits, contributors, and organizations for Fd.io, OVS, and DPDK from 2016-02-11 to 2017-04-03]
Multiproject: Fd.io Projects

Dataplane Management Agent
- vICN
- hc2vpp
- Honeycomb

Packet Processing
- ICNET
- ONE
- TLDK
- CICN
- odp4vpp
- VPP Sandbox
- VPP

Testing/Support
- CSIT
- puppet-fdio
- trex

Network IO
- deb_dpdk
- rpm_dpdk
Fd.io Integrations

Integration work done at 🌟 OPNFV

Control Plane
- LISP Mapping Protocol
- Netconf/Yang

Data Plane
- VPP
- Fd.io ML2 Agent
- Honeycomb
- VBD app
- GBP app

Openstack
- Neutron
  - ODL Plugin
  - Fd.io Plugin
Continuous Quality, Performance, Usability

Built into the development process – patch by patch

Submit → Automated Verify → Code Review → Merge → Publish Artifacts

Build/Unit Testing
120 Tests/Patch

Build binary packaging for
Ubuntu 14.04
Ubuntu 16.04
Centos 7

Automated Style Checking
Unit test:
IPFIX
BFD
Classifier
DHCP
FIB
GRE
IPv4
IPv4-IRB
IPv4 multi-VRF
IPv6
IPv Multicast
L2 FIB
L2 Bridge Domain
MPLS
SNAT
SPAN
VXLAN

System Functional Testing
252 Tests/Patch

DHCP – Client and Proxy
GRE Overlay Tunnels
L2BD Ethernet Switching
L2 Cross Connect Ethernet Switching
LISP Overlay Tunnels
IPv4-in-IPv6 Softwire Tunnels
Cop Address Security
IPSec
IPv6 Routing – NS/ND, RA, ICMPv6
uRPF Security
Tap Interface
Telemetry – IPFIX and Span
VRF Routed Forwarding
IPv4 Routing
QoS Policer Metering
VLAN Tag Translation
VXLAN Overlay Tunnels

Performance Testing
144 Tests/Patch, 841 Tests

L2 Cross Connect
L2 Bridging
IPv4 Routing
IPv6 Routing
IPv4 Scale – 20k,200k,2M FIB Entries
IPv4 Scale - 20k,200k,2M FIB Entries
VM with vhost-user
PHYS-VPP-VM-VPP-PHYS
L2 Cross Connect/Bridge
VXLAN w/L2 Bridge Domain
IPv4 Routing
COP – IPv4/IPv6 whiteless
iACL – ingress IPv4/IPv6 ACLs
LISP – IPv4-o-IPv6/IPv6-o-IPv4
VXLAN
QoS Policer
L2 Cross over
L2 Bridging

Usability
Merge-by-merge:
apt installable deb packaging
yum installable rpm packaging
autogenerated code documentation
autogenerated cli documentation
Per release:
autogenerated testing reports
report perf improvements
Puppet modules
Training/Tutorial videos
Hands-on-usecase documentation

Run on real hardware in fd.io Performance Lab
Merge-by-merge packaging feeds
Downstream consumer CI pipelines
Universal Dataplane: Infrastructure

Bare Metal
- Server
- FD.io
- Kernel/Hypervisor

Cloud/NFVi
- Server
- VM
- VM
- VM
- FD.io
- Kernel/Hypervisor

Container Infra
- Server
- Con
- Con
- Con
- FD.io
- Kernel
Universal Dataplane: VNFs

FD.io based VNFs

Server

VM

FD.io

VM

FD.io

FD.io

Kernel/Hypervisor

FD.io based VNFs

Server

Con

FD.io

Con

FD.io

FD.io

Kernel/Hypervisor
Universal Dataplane: Embedded

- Embedded Device
  - Device
    - Intel
    - ARM
    - Power
  - FD.io
  - Kernel/Hypervisor
  - Hw Accel

- SmartNic
  - Server
    - Kernel/Hypervisor
    - SmartNic
    - FD.io
    - Hw Accel
Universal Dataplane: CICN Example

- Physical CICN router
  - Device
    - Intel
    - ARM
    - FD.io
    - Kernel/Hypervisor
    - Hw Accel
- CICN in a VM
  - Server
    - VM
      - FD.io
    - VM
      - FD.io
    - Kernel/Hypervisor
    - FD.io
- CICN in a Container
  - Server
    - docker
      - FD.io
    - LXC
      - FD.io
    - Kernel/Hypervisor
Universal Dataplane: communication/API

**Consumer/Producer Socket API**
- Segmentation/Naming
- Manifest management
- Reassembly
- Flow and Congestion Control

**Existing drivers for links**
- DPDK
- AF-PACKET
- MEMIF (SHARED MEMORY)
What is Vector Packet Processing?

An open-source software that provides out-of-the-box production quality switch/router functionality running under commodity CPUs

- High Throughput
  - 14+ Mpps per core
- Multiplatform
  - Intel, ARM, Power
- Feature rich
  - L2, L3, L4, local and remote programmability
- Modular and Extensible
  - Through plugins
Why VPP?

• NFV goals
  • Software flexibility without giving up to hardware level performance

• What about existing solutions?
  • Linux Kernel
    • Too slow for high throughput
    • Evolve slowly
  • Click
    • In principle similar to VPP, no V(ector)
CICN distribution

- Core libraries
  - Consumer/Producer Socket API, CCNx libs, PARC C libraries

- Server and Router
  - VPP cicn plugin for Ubuntu 16, CentOS 7
  - HTTP video server, Apache Traffic Server Plugin coming soon

- Client
  - Metis Forwarder
  - VIPER MPEG-DASH video player
  - Android 7/8, MacOS X 10.12, iOS 10/11, Ubuntu 16, CentOS 7
  - Soon Apple Store and Google Play

- vICN
  - intent-based networking
  - model driven programmable framework
  - monitoring and streaming for BigData support (PNDA.io)
Opportunities to Contribute

- Forwarding strategies
- Mobility management
- Hardware Accelerators
- vICN, configuration/management/control
- Consumer/Producer Socket API
- Reliable Transport
- Instrumentation tools
- HTTP integration

We invite you to Participate in fd.io

- Get the Code, Build the Code, Run the Code, install from binaries
- from binary packages
- Read/Watch the Tutorials
- Join the Mailing Lists
- Join the IRC Channels
- Explore the wiki
- Join fd.io as a member
- https://wiki.fd.io/view/cicn
- https://wiki.fd.io/view/vicn
- https://fd.io/
Vector Packet Processing for ICN

Alberto Compagno
How does VPP work?

- VPP is a ‘packet processing graph’
  - Nodes are
    - Small
    - Loosely coupled
  - VPP processes vectors of packets
    - Passed from node to node
How does VPP work?

- Each node has its vector(s)
- Packets are “passed” from vector to vector
How does VPP work?

• Three types of nodes
  • Input
  • Internal
  • Process
How does VPP work?

**Input nodes**
- Read packets from RX buffer
- Create the packet vector

**Internal nodes**
- Process packets
- Called from other nodes
- Can be leaf (drop or TX)
How does VPP work?

Process nodes
- Not part of the processing graph
- Run in background
- React to timer/event
Extend VPP with plugins?

Plugins are first class citizen

They can:
• Add nodes
• Add api
• Rearrange the graph
How does VPP accelerate packet processing?
Accelerating packet processing

• Kernel bypass

• Code Design (Multi-loop, Branch prediction, Function flattening, Lock-free structures, Numa aware)

• Reduce cache misses
Reduce cache misses – Why?

• 14 Mpps on 3.5GHz CPU = 250 cycles/packet

• Cache hit:
  • ~2-30 cycles

• Cache miss (main memory)
  • ~140 cycles
Reduce cache misses
Reduce cache misses – I-cache

Let’s compare scalar processing with vector processing
Scalar Packet Processing

• Process one packet at a time
Scalar Packet Processing

• Process one packet at a time
Scalar Packet Processing

• Process one packet at a time
Scalar Packet Processing

• Process one packet at a time
Scalar Packet Processing

- Process one packet at a time
Scalar Packet Processing

- Process one packet at a time

Many I-cache misses per packet
Vector Packet Processing

- Every node process the full packet vector
Vector Packet Processing

- Every node process the full packet vector
Vector Packet Processing

• Every node process the full packet vector
Vector Packet Processing

- Every node process the full packet vector
Vector Packet Processing

- Every node processes the full packet vector

Processing the full vector amortizes the cost of the first L-cache miss... at the cost of increasing D-cache misses
Reduce cache miss – D-cache

VPP pre-fetches data into D-cache

Expensive
Reduce cache miss – D-cache

Example: Processing packet 1 & 2

Might have a cache miss for packet 1 & 2

VPP node pseudocode

while packets in vector
  while 4 or more packets
    PREFETCH #3 and #4
    PROCESS #1 and #2
  while any packets
    <as above but single packet>
Reduce cache miss – D-cache

Example: Processing packet 3 & 4

The cost of the first D-cache miss is amortized by the subsequent D-cache hits.
Hands on VPP!
VPP documentation

• Wiki

https://wiki.fd.io/view/VPP

• Doxygen

https://docs.fd.io/vpp/17.04/
Download VPP (v17.04)

- Clone the source code from git

  git clone https://gerrit.fd.io/r/vpp

- Or install it from .deb pkg (rpm for Centos available too)

  ... see wiki
Configure and Start VPP

• VPP configuration file
  
  # emacs /etc/vpp/startup.conf

• Start vpp
  
  # sudo vpp -c /etc/vpp/startup.conf
VPP Command Line Interface

• To start a shell:

    # vppctl

• To run one command:

    # vppctl <command>
VPP Command Line Interface

• A bunch of useful commands:
  • ?
  • show
  • set
Create your own plugin
Outline

- VPP structures
- Design & Implement your node(s)
- Insert your node(s) in the vlib_graph
- Compile and install your plugin
VPP structures

The vector of packets is called FRAME.

Each element is called VECTOR.

A vector is an index to a vlib_buffer_t.

Memory holding vlib_buffer_t objects.

vlib_buffer_t

....

u8 data[0];

Pointer to packet data (DMA memory)
Outline

• VPP structures
• Design & Implement your node(s)
• Inserting your node(s) in the vlib_graph
• Compiling and installing your plugin
Design & Implement your node(s)

• Your node should follow VPP style
  • Multi-loop, Branch prediction, Function flattening, Lock-free structures

• A node must implement a processing function that
  • “Moves vectors” from your node’s frame to the next node’s frame
  • Processes packets as YOU want

• Add whatever else you need
  • Supporting Functions, macros, variables, etc.. (C code)
Register your node(s) to VPP

- Each node must be registered to VPP through VLIB_REGISTER_NODE macro

```c
#define VLIB_REGISTER_NODE( x,
                           ...
)                     
```

### VLIB_REGISTER_NODE Macro Definition

```c
typedef struct _vlib_node_registration
{
    /* Vector processing function for this node. */
    vlib_node_function_t *function;

    /* Node name. */
    char *name;

    /* Name of sibling (if applicable). */
    char *sibling_of;

    /* Node index filled in by registration. */
    u32 index;

    /* Type of this node. */
    vlib_node_type_t type;

    /* Error strings indexed by error code for this node. */
    char **error_strings;
} _vlib_node_registration_t;
```

Definition at line 143 of file node.h.
Example: Cicn plugin

```c
VLIB_REGISTER_NODE (icnfwd_node) =
{
  .function = icnfwd_node_fn,
  .name = "icnfwd",
  .vector_size = sizeof (u32),
  .runtime_data_bytes = sizeof (icnfwd_runtime_t),
  .format_trace = icnfwd_format_trace,
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = ARRAY_LEN (icnfwd_error_strings),
  .error_strings = icnfwd_error_strings,
  .n_next_nodes = ICNFWD_N_NEXT,
  .next_nodes = {
    [ICNFWD_NEXT_LOOKUP] = "ip4-lookup",
    [ICNFWD_NEXT_ERROR_DROP] = "error-drop",
  },
},
```

Let’s take a look to icnfwd_node_fn
icnfwd node

from
n_left_from
to_next
n_left_to_next
next_index
next0=?
Hold the actual next node id
icnfwd node

b0

from

n_left_from

next0=

bi0=

to_next

n_left_to_next

icnfwd

ip4-lookup

e-error

vlib_buffer_t objects

next_index

next0=?
icnfwd node

- vlib_buffer_t objects
- icnfwd
- bi0
- from
- n_left_from
- to_next
- ip4-lookup
- error
- n_left_to_next
- next_index
- next0=?
Wrong speculation

b0

n_left_from

icnfwd

ip4-lookup

to_next

vlib_buffer_t objects

error

n_left_to_next

next_index

next0=?

n_left_to_next

next0=?
Example: Cicn plugin

VLIB_REGISTER_NODE (icnfwd_node) =
{
    .function = icnfwd_node_fn,
    .name = "icnfwd",
    .vector_size = sizeof (u32),
    .runtime_data_bytes = sizeof (icnfwd_runtime_t),
    .format_trace = icnfwd_format_trace,
    .type = VLIB_NODE_TYPE_INTERNAL,
    .n_errors = ARRAY_LEN (icnfwd_error_strings),
    .error_strings = icnfwd_error_strings,
    .n_next_nodes = ICNFWD_N_NEXT,
    .next_nodes = {
        [ICNFWD_NEXT_LOOKUP] = "ip4-lookup",
        [ICNFWD_NEXT_ERROR_DROP] = "error-drop",
    },
};
Other important macros

• **VPP_INIT_FUNCTION**
  • Function that is called during VPP initialization

• **VPP_REGISTER_PLUGIN**
  • Required to guarantee that your plugin is actually a VPP plugin
    …and not a library copied by mistake in /usr/lib/vpp_plugins
Outline

• VPP structures
• Design & Implement your node(s)
• Insert your node(s) in the vlib_graph
• Compile and install your plugin
Insert your node to VPP graph

1. direct all the packets from one interface
   • vnet_hw_interface_rx_redirect_to_node (vnet_main, hw_if_index, my_graph_node.index /* redirect to my_graph_node */);

2. capture packets with a particular ethertype
   • ethernet_register_input_type (vm, ETHERNET_TYPE_CDP, cdp_input_node.index);

3. for-us packet for new protocol on top of IP
   • ip4_register_protocol (IP_PROTOCOL_GRE, gre_input_node.index);
Insert your node to VPP graph

4. ip-for-us packet sent to a specific UDP port
   • `udp_register_dst_port (vm, UDP_DST_PORT_vxlan, vxlan_input_node.index, 1 /* is_ip4 */);`

5. direct all packets from one ip prefix
   • Create your own Data Path Object (i.e. result of a FIB lookup)
Outline

• VPP structures
• Design & Implement your node(s)
• Insert your node(s) in the vlib_graph
• Compile and install your plugin
Compiling your plugin

• VPP provides Automake/Autoconf examples
  • Install vpp-dev and move to /usr/share/doc/vpp/examples

• Adapting Makefile.am and sample.am is trivial

• Compile cicc-plugin:

```bash
$ cd cicc-plugin
$ autoreconf -i -f
$ mkdir -p build
$ cd build
$ ../configure --with-plugin-toolkit
OR, to omit UT code
$ ../configure --with-plugin-toolkit --without-cicc-test
$ make
$ sudo make install
```
vICN: configuration, management and control of an virtual ICN network

Marcel Enguehard
ACM ICN Conference – CICN tutorial
September 26th 2017
What is vICN

- Unified framework for network deployment, management and monitoring
- Integrates all the tools of the CICN fd.io suite
- Provides an API to easily bootstrap ICN deployments and get meaningful telemetry out of it
vICN at a glance

User intent:
- I want to deploy and monitor a network model

Infrastructure description:
- I have some infrastructure you can use

Resource description:
- I know how things work
Example vICN topology

- Cons
- Relay
- Prod
- consumer
- forwarder
  - veth
- producer
  - forwarder
  - veth
- Linux Containers
- Bridge
- veth
- veth
- veth
- veth
- veth

Data plane

Control plane
vlCN resources

• Virtual representation of deployment element
• Node, forwarder, application, link, etc.
• Described by attributes
Example resource: forwarder

• Represents an ICN forwarder
• Attributes:
  • node
  • cache_size
  • cache_policy (e.g., LRU)
  • log_file
  • etc.
Resource hierarchy

- Resource
  - Node
    - LXC Container
    - Physical node
  - Channel
  - Interface
    - ...
How does it work?

• Intent based-framework

• Object-based model

• State reconciliation between model and deployment

```python
cons = LxcContainer()
prod = LxcContainer()
link = Link(src=cons, dst=prod)
```
vICN functionalities

• Multithreaded deployment of network models
• SDN controller for IPv4, IPv6, and ICN
• Wireless links emulation
• Connection of real devices
• Built-in monitoring through Python model
Our example deployment
Network model deployment
Network model declaration

- JSON file containing list of resources
- Resources complemented with “key” attributes
- Intent-based declaration: descriptive approach (not imperative)
Physical resources

{  "type": "Physical",  "name": "server",  "hostname": "localhost"},

{  "type": "LxcImage",  "name": "cicn-image",  "node": "server",  "image": "ubuntu1604-cicnsuite-rc3"}
Nodes

References to previous resources
Links

```json
{
    "type": "Link",
    "src_node": "cons",
    "dst_node": "core1",
    "groups": [ "virtual" ]
},
{
    "type": "Link",
    "src_node": "core1",
    "dst_node": "core2",
    "groups": [ "virtual" ]
},
{
    "type": "Link",
    "src_node": "core2",
    "dst_node": "prod",
    "groups": [ "virtual" ]
}
```
IP networking on topology

CentralIP is similar to an SDN controller that assigns addresses and sets up the routing in the network:

CentralIP = (Ipv4Assignment | Ipv6Assignment) > IPRoutes
ICN forwarders

```
{
  "type": "MetisForwarder",
  "cache_size": 0,
  "node": "cons"
},
{
  "type": "MetisForwarder",
  "cache_size": 2000,
  "node": "core1"
},
{
  "type": "MetisForwarder",
  "cache_size": 0,
  "node": "core2"
},
{
  "type": "MetisForwarder",
  "cache_size": 0,
  "node": "prod"
},
{
  "type": "WebServer",
  "prefixes": [
    "/webserver"
  ],
  "node": "prod"
},
{
  "type": "CentralICN",
  "groups": [ "virtual" ],
  "face_protocol": "udp4"
}
```

Like CentralIP ether, udp4, udp6, tcp4, tcp6
GUI

{
    "type": "GUI",
    "groups": ["virtual"]
},

![Network view diagram](image-url)
Launching vicn

cicn@cicn-VirtualBox:~/vicn$ sudo vicn/bin/vicn.py -s examples/tutorial/tutorial06-acm-icn17.json

[...]
2017-09-21 17:48:15,023 - vicn.core.task - INFO - Scheduling task <Task[apy] partial<_task_resource_update>> for resource <UUID MetisForwarder-MPDRB>
2017-09-21 17:48:15,024 - vicn.core.resource_mgr - INFO - Resource <UUID MetisForwarder-MPDRB> is marked as CLEAN (99/104)
2017-09-21 17:48:15,146 - vicn.core.task - INFO - Scheduling task <Task[apy] partial<_task_resource_update>> for resource <UUID MetisForwarder-NC33W>
2017-09-21 17:48:15,148 - vicn.core.resource_mgr - INFO - Resource <UUID MetisForwarder-NC33W> is marked as CLEAN (100/104)
Traffic creation

Producer setup:

• producer-test
  producer-test -D ccnx:/webserver

• Webserver
  http-server -p $server_folder -l http://webserver

Consumer setup

• consumer-test
  consumer-test -D ccnx:/webserver

• iget
  iget http://webserver/$filename
Traffic visualization on the GUI
Network teardown

cicn@cicn-VirtualBox:~/vicn$ sudo ./scripts/topo_cleanup.sh examples/tutorial/tutorial06-acm-icn17.json

wifi_emulator: no process found
lte_emulator: no process found
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
Removing bridge...
Removing interface...
Removing stale routes
VPP in vICN

- Objective: learn to setup vICN to use your Intel interfaces
- VPP running in container
- Uses DPDK and ZC-forwarding
Setup

cons -- VPP -- prod

core1

enp0s9

core2

enp0s10

DPDK-compatible interfaces
Identifying the DPDK interfaces

Compare:

```
sudo lshw -c network -businfo
```

with [http://dpdk.org/doc/nics](http://dpdk.org/doc/nics)
Declaring the DPDK Interfaces

```json
{
  "type": "DpdkDevice",
  "name": "core1-dpdk1",
  "mac_address": "08:00:27:44:9a:38",
  "node": "core1",
  "device_name": "enp0s9",
  "pci_address": "0000:00:09.0"
},
{
  "type": "DpdkDevice",
  "name": "core2-dpdk1",
  "mac_address": "08:00:27:18:42:f2",
  "node": "core2",
  "device_name": "enp0s10",
  "pci_address": "0000:00:0a.0"
}
```

```
cicn@cicn-VirtualBox:~$ sudo lshw -c network -businfo
Bus info  Device   Class                Description
---------- ---------- -------------------------------
pci@0000:00:03.0 enp0s3  network          82540EM Gigabit Ethernet Controller
pci@0000:00:08.0 enp0s8  network          82540EM Gigabit Ethernet Controller
pci@0000:00:09.0 enp0s9  network          82545EM Gigabit Ethernet Controller (Copper)
pci@0000:00:0a.0 enp0s10 network          82545EM Gigabit Ethernet Controller (Copper)
```
Changes to resources

```
{  
  "type": "Link",
  "src_node": "core1",
  "dst_node": "core2",
  "groups": [ "virtual" ]
},

{  
  "type": "MetisForwarder",
  "cache_size": 2000,
  "node": "core1"
},

{  
  "type": "MetisForwarder",
  "cache_size": 0,
  "node": "core2"
},

{  
  "type": "PhyLink",
  "src": "core1-dpdk1",
  "dst": "core2-dpdk1",
  "groups": [ "virtual" ]
},

{  
  "type": "VPP",
  "node": "core1",
  "name": "vpp_core1"
},

{  
  "type": "CICNPlugin",
  "node": "core1",
  "name": "vpp-fwd"
},

{  
  "type": "VPP",
  "node": "core2",
  "name": "vpp_core2"
},

{  
  "type": "CICNPlugin",
  "node": "core1",
  "name": "vpp-fwd"
},

```
What is vICN actually doing?

- **VPP-ready host**
  - Install (if necessary) the DPDK driver and load it in the host kernel
  - Change driver for DPDK-compatible devices
  - Change number of hugepages for VPP

- **VPP-ready container**
  - Create a privileged container by changing its apparmor profile
  - Add DPDK-enabled interfaces to the container
What is vICN actually doing? (cont’d)

- **Start VPP on the container**
  - Create configuration file for VPP in the container
  - Start VPP
  - Set up IP forwarding

- **Start CICN plugin in VPP**
  - Enable CICN plugin
  - Set up ICN faces and routes
Launching vicn

cicn@cicn-VirtualBox:~/vicn$ sudo vicn/bin/vicn.py -s examples/tutorial/tutorial06-acm-icn17-vpp.json

[...]  
2017-09-21 17:48:15,023 - vicn.core.task - INFO - Scheduling task <Task[apy] partial<_task_resource_update>> for resource <UUID MetisForwarder-MPDRB>  
2017-09-21 17:48:15,024 - vicn.core.resource_mgr - INFO - Resource <UUID MetisForwarder-MPDRB> is marked as CLEAN (99/104)  
2017-09-21 17:48:15,146 - vicn.core.task - INFO - Scheduling task <Task[apy] partial<_task_resource_update>> for resource <UUID MetisForwarder-NC33W>  
2017-09-21 17:48:15,148 - vicn.core.resource_mgr - INFO - Resource <UUID MetisForwarder-NC33W> is marked as CLEAN (100/104)
Traffic creation

**Producer setup:**
- **producer-test**
  
  `producer-test -D ccnx:/webserver`

- **Webserver**
  
  `http-server -p $server_folder -l http://webserver`

**Consumer setup**
- **consumer-test**
  
  `consumer-test -D ccnx:/webserver`

- **iget**
  
  `iget http://webserver/$filename`
Traffic visualization on the GUI
Toward a new Python API

Use python objects instead of static JSON file

```python
cons = LxcContainer()
prod = LxcContainer()
link = Link(src=cons, dst=prod)
```
More on vICN

- Demonstration session: new dynamic python API
- Thursday 10:50am: vICN paper presentation
Available tutorials

In `examples/tutorial/`:

- `tutorial01.json` → Simple topology
- `tutorial02-dumbell` → VPP
- `tutorial03-hetnets.json` → Wireless emulators
- `tutorial06-acm-icn17*` → Today’s tutorial (soon)

[https://wiki.fd.io/view/Vicn#Tutorials_overview]
References

vICN wiki: https://wiki.fd.io/View/Vicn


vICN code: git clone -b vicn/master https://gerrit.fd.io/r/cicn vicn
Libicnet: transport layer library for ICN

Mauro Sardara
What is Libicnet?

• Library implementing a transport layer and exposing **socket API** to applications willing to communicate through an ICN protocol stack

• Relieves applications from the task of managing layer 4 problems, such as **segmentation and congestion control**

• Enhances the **separation** between Application Data Unit (ADU) and Protocol Data Unit (PDU) processing
Core Elements

• ProducerSocket
  • ADU Segmentation and Naming → Layer 4 PDU (ICN Content Object)
  • L4 PDU Signature
  • L4 PDU Publication

• ConsumerSocket
  • Congestion control
  • L4 PDU Fetching
  • Signature verification
  • L4 PDU reassembly → ADU
ProducerSocket

Produce(Produce(Name))

Seguequentiation + Naming
Signature
Publication

ADU
ProducerSocket

ProducerSocket

Segmentation + Naming

PDU = ICN Content Object

Signature

Publication

PDU = ICN

Application

ICN Name + Signature

PDU = ADU
• Signature
  • The application has to provide the library with the information for signing the content objects
  • Signing every content object is computationally expensive: we provide support for manifest
ConsumerSocket

ConsumerSocket

- Congestion Control
- Signature verification
- PDU Reassembly

Application

Consume(Name)
ConsumerSocket

ConsumerSocket

Congestion Control

Signature verification

PDU Reassembly

Application

Interest

Content Object

Consume(Name)
ConsumerSocket

- Congestion Control
- Signature verification
- PDU Reassembly

Interest → ConsumerSocket → Content Object

ConsumerSocket → Consume(Name) → Application
ConsumerSocket

ConsumerSocket

Congestion Control
Signature verification
PDU Reassembly

Interest
Content Object

Consume(Name)

Application

ADU
ConsumerSocket

• Congestion Control
  • Application can choose among a set of algorithms: VEGAS, RAAQM\(^1\), FIXED_WINDOW
  • Extension with new algorithms possible

• Signature
  • The application has to provide the library with the information for verifying the signature of the received content objects
  • As the producer case, verifying every content object is expensive: we verify just the manifest signature

\(^1\) G. Carofiglio et al. "Multipath congestion control in content-centric networks,” 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPs)
Hands on Libicnet!
Where to find Libicnet?

• Wiki page
  • https://wiki.fd.io/view/Libicnet

• Code
  • https://git.fd.io/cicn/log/?h=libicnet/master
Hello World Applications

• We will see how building two trivial applications against Libicnet:
  • Hello world Producer
    • It will produce a content of a certain size
  • Hello world Consumer
    • It will pull the content published by the producer
Topology

- **Hello World Consumer**
- **Metis Forwarder**
- **Hello World Producer**
Hello World Producer

#include <icnet/icnet_transport_socket_producer.h>

…
Name n("ccnx://helloworld");

ProducerSocket p_(n);

std::string content(10000, 'A');

p_.produce(n, (uint8_t*)content.data(), content.size());

p_.attach();

p_.serveForever();
Hello World Consumer

```cpp
#include <icnet/icnet_transport_socket_consumer.h>

... Consumer c_(Name(), TransportProtocolAlgorithms::RAAQM);
c_.setSocketOption(GeneralTransportOptions::INTEREST_LIFETIME, 1001);
c_.setSocketOption(GeneralTransportOptions::MAX_INTEREST_RETX, 25);

c_.setSocketOption(ConsumerCallbacksOptions::CONTENT_RETRIEVED,
                  (ConsumerContentCallback) std::bind(&processContent,
                  std::placeholders::_1,
                  std::placeholders::_2));

Name name("ccnx://helloworld");

c_.consume(name);
```

Congestion control algorithm

Callback called after whole ADU will be pulled and reassembled

Content Pull + Signature Verification + Reassembly
Callbacks

- The application can register into the library a set of callback allowing to directly handle events during the download/publication.

```c
typedef enum {
    INTEREST_OUTPUT = 401,
    INTEREST_RETRANSMISSION = 402,
    INTEREST_EXPIRED = 403,
    INTEREST_SATISFIED = 404,
    CONTENT_OBJECT_INPUT = 411,
    MANIFEST_INPUT = 412,
    CONTENT_OBJECT_TO_VERIFY = 413,
    CONTENT_RETRIEVED = 414,
} ConsumerCallbacksOptions;

typedef enum {
    INTEREST_INPUT = 501,
    INTEREST_DROP = 502,
    CACHE_HIT = 506,
    CACHE_MISS = 508,
    NEW_CONTENT_OBJECT = 509,
    CONTENT_OBJECT_SIGN = 513,
    CONTENT_OBJECT_READY = 510,
    CONTENT_OBJECT_OUTPUT = 511,
} ProducerCallbacksOptions;
```
Advanced Example: HTTP support

HTTP Client

libicnet
locator = getlocator()
name = resourceName()

send(locator)                       recv(name)

Send Buffer
Producer Socket

Recv Buffer
Consumer Socket

HTTP request reply

Network (ICN)

HTTP Server

libicnet
locator = getlocator()
name = resourceName()

send(name)                       recv(locator)

Send Buffer
Producer Socket

Recv Buffer
Consumer Socket

HTTP reply request

Network (ICN)
Thank You!