ABSTRACTION LAYER FOR IMPLEMENTATION OF EXTENSIONS IN PROGRAMMABLE NETWORKS

Collaborative project co-funded by the European Commission within the Seventh Framework Programme

Grant agreement no: 317880
Project acronym: ALIEN
Project full title: "Abstraction Layer for Implementation of Extensions in programmable Networks"
Project start date: 01/10/12
Project duration: 24 months

Deliverable D5.3
Experimental-driven research - Appendix I (Datapath Protocol Programming experiment results)

Version 7.9

Due date: 30/10/2014
Submission date: 12/11/2014
Editor: Damian Parniewicz (PSNC), Maider Huarte (EHU)
Author list: Damian Parniewicz, Łukasz Ogrodowczyk, Bartosz Belter, Artur Binczewski (PSNC).

Dissemination Level

☒	**PU:**	Public
☐	**PP:**	Restricted to other programme participants (including the Commission Services)
☐	**RE:**	Restricted to a group specified by the consortium (including the Commission Services)
☐	**CO**	Confidential, only for members of the consortium (including the Commission Services)

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Table of Contents

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Figure summary

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

Executive Summary

This document complements deliverable D5.3 “Experimental-driven research” with details about testing scenario steps and all materials like logs, screenshots, packet dumps collected during performing of experimentation tests. We highly encourage to read first D5.3 which contains overview of the experiment, goals to be achieved, experiment environment and also discussion about experiment validation.

<THIS PAGE IS INTENTIONALLY LEFT BLANK>

1 Datapath protocol programming

1.1 Testing scenarios

Datapath protocol programming testing is composed of four phases:

- Step 1: Loading a whole protocol description
- Step 2: Setting network configuration and processing network frames
- Step 3: Updating part of the protocol description
- Step 4: Setting network configuration after protocol update and processing network frames

1.1.1 Step 1: Loading a whole protocol description

In the first phase, the PAD client is passing a complete P4 language description to the SDN network, containing the structure of protocol headers, protocols parsers and actions (see below).

```
header ethernet {
fields {
   dst_addr : 48; // width in bits
src_addr : 48;
ethertype : 16;
}
}

header ictp {   // Information Centric Transport Protocol
fields {
nid : 32;   // Network content ID (ICN-ID)
csn : 32;   // Chunk Sequence Number
}
```

```
}

parser start {  // where parser should start
ethernet;     // this is a first header to be parsed
}

parser ethernet {
switch(ethertype) {  // header field based lookup
case 0x9100: ictp;  // what is next header
}
}

action push_ictp {
// Inserting ICTP header with offset equal to Ethernet header size
add_header(ictp, sizeof(ethernet));
}

action pop_ictp {
// Removing ICTP header from offset equal to Ethernet header size
remove_header(ictp, sizeof(ethernet));
}
```

The schema contains description for two protocols: Ethernet and ICTP (Information Centric Transport Protocol) defined by [CCNx]/[CONET]. The ICTP header represent a clean slate solution (see Figure 1-1) in which we don't use current IP protocol stack and we are building new protocols directly on Ethernet framing. The P4 language description of data plane protocols contains structure of headers, parsers and actions which can be performed on ICTP packet (two defined actions are strictly related to ICTP protocol).

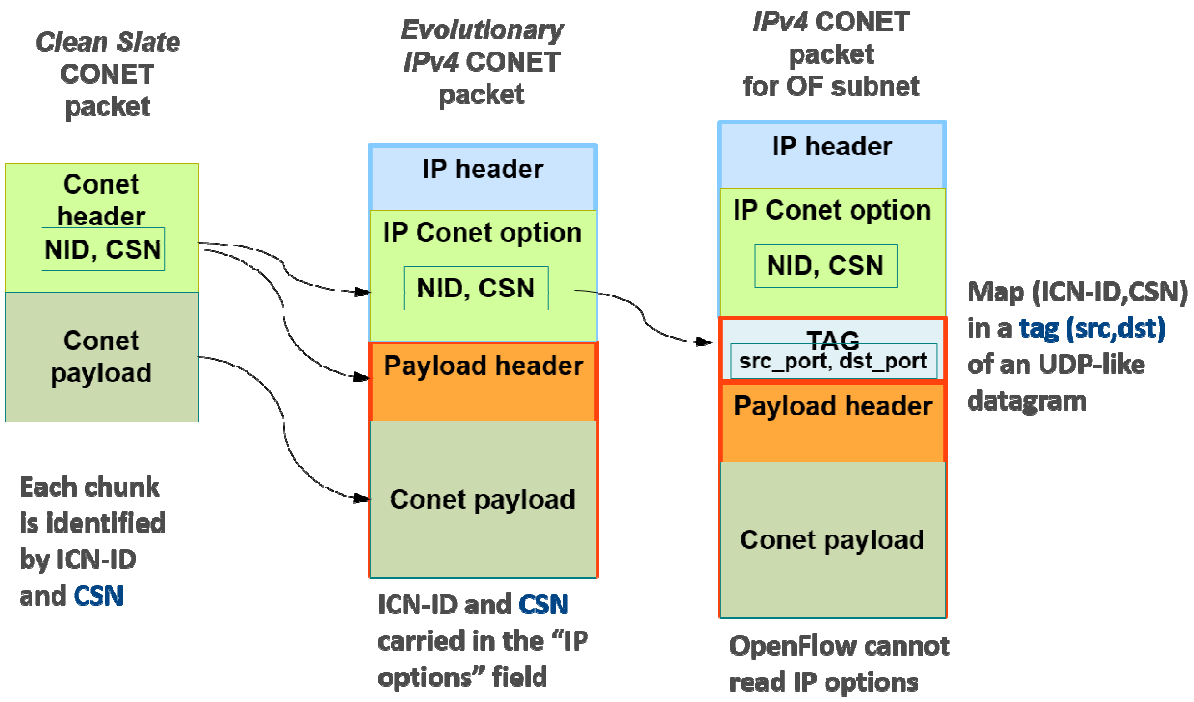

Figure 1-1 CCNx/CONET packets

Figure 1-2 presents the structure and order of headers expressed in a description passed by PAD client to code generator.

Ethernet header			ICTP header	
dst_addr	src_addr	ethertype	nid	csn
6 octets	6 octets	2 octets	4 octets	4 octets

Figure 1-2 Structure of Ethernet and ICTP headers defined by P4 schema

When the protocol description is translated into code files and compiled with ROFL and xDPd, then ROFL-based controller as well all xDPd/GNU-Linux software switches are restarted and SDN network is ready to work.

1.1.2 Step 2: Setting network configuration and processing network frames

In the second phase, ROFL-based controller populates both software switches with flow entries containing the generated actions:

- for xDPd switch 1: push-ictp, set-field-nid, set-field-csn
- for xDPd switch 2: pop-ictp

In this way, the first software switch adds ICTP header directly after Ethernet header to all frames received on interface 1 (generated by TAP client 1) and set values of ICTP header fields. The modified frames are transmitted by interface 2 to the second software switch. On the second software switch ICTP headers are removed and frames are transmitted by interface 2 to the TAP client 2. The whole frames processing is presented in Figure 1-3.

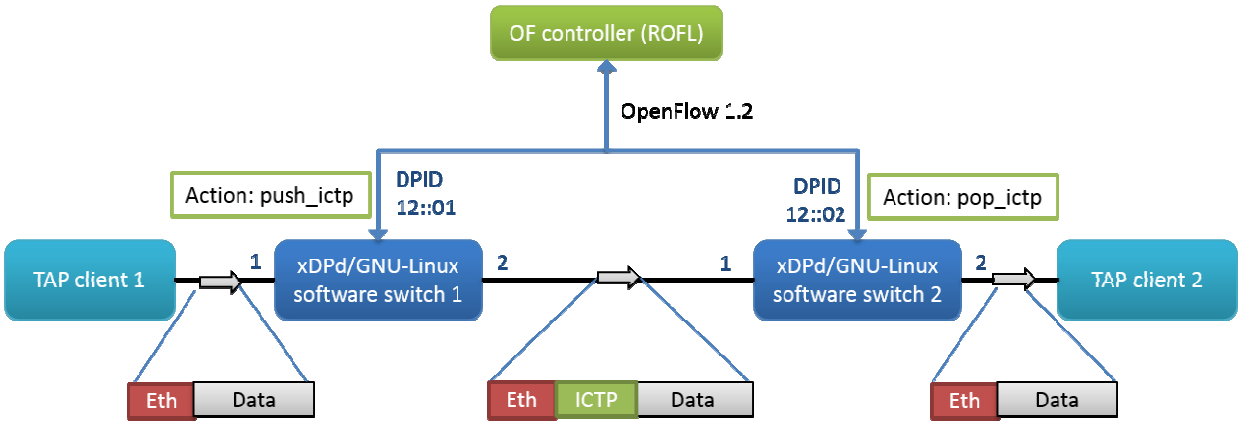

Figure 1-3 OpenFlow controller sets actions related to newly defined ICTP protocol

The Ethernet frames are generated by the TAP client 1 Python program presented in listing below:

```
from pytun import TunTapDevice, IFF_TAP
from dpkt.ethernet import Ethernet

tap0 = TunTapDevice(name="tap0", flags=IFF_TAP)
tap0.run()

frame = Ethernet(dst = "\x01\x02\x03\x04\x05\x06",
src = "\x0A\x0B\x0C\x0D\x0E\x0F",
type = 0x800,
data = "\x61"*40)

PREAMBLE = "\x11\x22\x33\x44"
tap0.write(PREAMBLE + str(frame))
```

The program generates Ethernet frames with ethertype equal ‘0x800’ (it is arbitrary value) and contains 40 bytes of data. During the test we are sniffing how that frame is changing depending on which interface were listening.

1.1.3 Step 3: Updating part of the protocol description

In the third phase, the PAD client is removing ICTP header description and adding a new description of this header which is, right now, composed of more fields. Besides two 32-bits fields, the ICTP header includes one 1-byte field, one 2-bytes field and 1-byte padding expressed in the schema as __skip__ (__skip__ field is a special field, added by us to P4 syntax,

which doesn't generate OpenFlow extensions, matches and actions). The change of ICTP header is presented in Figure 1-4.

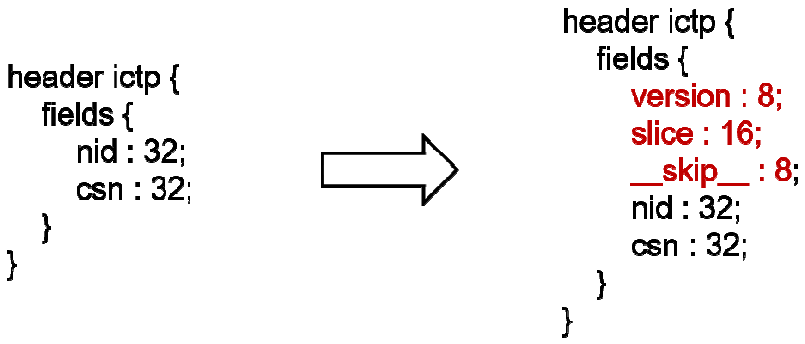

Figure 1-4 Change of the ICTP protocol header structure

Figure 1-5 presents the structure and order of headers expressed in a new description passed by PAD client to code generator.

Ethernet header			ICTP header				
dst_addr	src_addr	ethertype	version	slice	pad	nid	csn
6 octets	6 octets	2 octets	1 octet	2 octets	1 octed	4 octets	4 octets

Figure 1-5 A new structure of Ethernet and ICTP headers by P4 schema

We must emphasise that other parts of protocol descriptions remains the same. Finally, both controller and software switches must be recompiled and restarted.

1.1.4 Step 4: Setting network configuration after protocol update and processing network frames

In the fourth phase, we are performing the same actions as in the second phase:

- send the same flow entries to both software switches,
- generate the same Ethernet frames by TAP client 1,
- sniff packets on interfaces.

1.2 Experiment results

1.2.1 Step 1: Loading a whole protocol description

Our SDN network is modified by calling pad-client.py script which uses PAD API to upload protocol information to network nodes and controller (see listing below).

```
def upload_ictp_v1():

ethernet_hrd = """
header ethernet {
fields {
dst_addr : 48;
src_addr : 48;
ethertype : 16;
}
}
parser start {
ethernet;
}
parser ethernet {
switch(ethertype) {
case 0x9100: ictp;
}
  }
"""

ictp_hdr = """
header ictp {
fields {
nid : 32;
csn : 32;
}
}
"""

push_ictp = """
action push_ictp {
add_header(ictp, sizeof(ethernet));
}
"""

pop_ictp = """
action pop_ictp {
```

```
remove_header(ictp, sizeof(ethernet));
}
"""

pad.add_protocol("Ethernet", ethernet_hrd)
pad.add_protocol("ICTP", ictp_hdr)
pad.add_function("push_ictp", push_ictp)
pad.add_function("pop_ictp", pop_ictp)

of_extensions_ids = pad.commit_configuration()

print "OpenFlow experimental identifiers:"
pprint.pprint(of_extensions_ids)
```

Script pad-client.py has generated the following screen logs telling what files of both ROFL and xDPd were modified and what OpenFlow identifiers of experimental matches and actions were chosen for a new header:

```
pad$ python pad.py
  File generated: /home/geysers-wp5/rofl-core/src/rofl/common/endianess_other.h
  File generated: /home/geysers-wp5/rofl-core/src/rofl/common/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_action.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_action.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/ictp_actions.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/ictp_actions.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet_actions_autogenerated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/ictp_matches.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/ictp_matches.cc
```

```
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/openflow_experimental.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/coxmatch.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match_auto
   generated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match_auto
   generated.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_packet_mat
   ches.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_packet_mat
   ches.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet_autogenerated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/fictpframe.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/fictpframe.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/Makefile.am
  File              generated:              /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/packetclassifier.h
  File              generated:              /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/static_pktclassifier.h
  File              generated:              /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/static_pktclassifier.cc
  File              generated:              /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/pipeline-imp/packet.cc
  File                         generated:                         /home/geysers-
   wp5/xdpd/src/xdpd/openflow/openflow12/of12_translation_utils.cc
OpenFlow experimental identifiers:
{'ictp': [{'experimental_id': 27, 'field': 'nid'},
          {'experimental_id': 28, 'field': 'csn'},
          {'action': 'pop_ictp', 'experimental_id': 3},
          {'action': 'pop_ictp', 'experimental_id': 4}]}
```

Files were distributed to software switches and compiled manually. In the future, we would like to use one of the tools for automate administration (e.g.: Cuisine, Chef, Ansible, etc) to do the code file distribution, compilation and software restart.

The whole compilation and installation time of both ROFL and xDPd took around 3.5 minutes (assigned resources inside a virtual machine: 1 core from Intel Xeon 5600 and 4GB RAM).

1.2.2 Step 2: Setting network configuration and processing network frames

1) xDPd switch 1 - Adding ICTP header to network frames

Method, presented in listing below, was called when xDPd software switch 1 has connected to ROFL-based controller. In the method, a flow entry was created and sent to the xDPd software switch 1.

```
void padswitch::handle_dpath1_open(cofdpt *dpt) {

rofl::cflowentryfe(dpt->get_version());
fe.set_command(OFPFC_ADD);
fe.set_table_id(0);

  // setting matches
fe.match.set_eth_type(farpv4frame::ARPV4_ETHER);
fe.match.insert(rofl::coxmatch_ofb_eth_type(0x800));

  // setting actions
fe.instructions.next() = cofinst_apply_actions(dpt->get_version());
fe.instructions.back().actions.next()=cofaction_push_ictp(dpt->get_version(),
                                                          0x9100);
fe.instructions.back().actions.next()=cofaction_set_field(dpt->get_version(),
rofl::coxmatch_ofx_ictp_nid(0x11));
fe.instructions.back().actions.next()=cofaction_set_field(dpt->get_version(),
rofl::coxmatch_ofx_ictp_csn(0x2233));
fe.instructions.back().actions.next()=cofaction_output(dpt->get_version(),
                                                       2);
  // sending flow entry to xdpd/gnu-linux software switch
send_flow_mod_message(dpt, fe);
}
```

When flow entry was installed in xDPd software switch 1, the following logs were generated where we can see an OpenFlow table dump with flow entry inside:

```
[rofl-pipeline] OpenFlow switch instance (0x73f0f30)
[rofl-pipeline] ========================
[rofl-pipeline] Name: dp0
[rofl-pipeline] OpenFlow version: 3
[rofl-pipeline] OpenFlow datapathid: 256
[rofl-pipeline]
```

```
[rofl-pipeline] Dumping table # 0 (0x7f21c2abf010). Default action: 0. # of
   entries: 1
[rofl-pipeline]                [0] Entry (0x73eca50), prior. 2048 #hits 1
   Matches:{[ETH_TYPE:0x800], }
Inst->>  APPLY,
                       APP.ACTIONs:<PUSH_ICTP>,<SET_ICTP_NID:
   0x11000000>,<SET_ICTP_CSN: 0x33220000>,<OUTPUT port: 2>,
[rofl-pipeline]          [*] No more entries...
[rofl-pipeline] --End of pipeline tables--
[rofl-pipeline] Dumping group table. # of group entries: 0.
[rofl-pipeline]          [*] No entries
[rofl-pipeline]
[rofl-pipeline] --End of group table—
```

When flow entry has been installed then network frames started to be switched to the second interface of the xDPd switch. xDPd has generated the following logs:

```
[rofl-pipeline] Packet[0x236dc50] entering switch [dp0] pipeline (1.X)
Packet   matches   [PORT_IN:1,   ETH_SRC:0xa0b0c0d0e0f,   ETH_DST:0x10203040506,
   ETH_TYPE:0x800, IP_PROTO:97, IP_ECN:0x1, IP_DSCP:0x18,  IPV4_SRC:0x61616161,
   IPV4_DST:0x61616161, ]
[rofl-pipeline] Packet[0x236dc50] matched at table: 0, entry: 0x73eca50
[eth2.700] OUTPUT packet(0x236dc50)
Packet   matches   [PORT_IN:1,   ETH_SRC:0xa0b0c0d0e0f,   ETH_DST:0x10203040506,
   ETH_TYPE:0x9100,          IPV4_SRC:0x61616161,          IPV4_DST:0x61616161,
   ICTP_NID:285212672,
```

Finally, we have captured frames on both interfaces of the switch. Frames entering xDPd without ICTP header on the first interface:

```
PadSwitch1:~$ sudotcpdump -i tap0 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 65535 bytes
05:44:15.165821 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161                           aaaaaa
05:44:17.166520 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161
```

Frames leaving xDPd switch with ICTP header included on the second interface:

```
PadSwitch1:~$ sudotcpdump -i eth2.700 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2.700, link-type EN10MB (Ethernet), capture size 65535 bytes
05:41:29.030754   0a:0b:0c:0d:0e:0f   (oui   Unknown)   Unknown   SSAP   0x22   >
   01:02:03:04:05:06 (oui Unknown) Unknown DSAP 0x32 Information,   send seq 0,
   rcvseq 0, Flags [Command], length 44
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 1100  ................
       0x0010:  0000 3322 0000 6161 6161 6161 6161 6161  ..3"..aaaaaaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161       aaaaaaaaaaaaaa
05:41:31.033187   0a:0b:0c:0d:0e:0f   (ouiUnknown)   Unknown   SSAP   0x22   >
   01:02:03:04:05:06  (ouiUnknown)  Unknown  DSAP  0x32  Information,  sendseq  0,
   rcvseq 0, Flags [Command], length 44
       0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 1100  ................
       0x0010:  0000 3322 0000 6161 6161 6161 6161 6161  ..3"..aaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161       aaaaaaaaaaaaaa
```

2) xDPd switch 2 - Removing ICTP header from network frames

The method, presented in the following listing, was called when xDPd software switch 2 has connected to ROFL-based controller. In the method, a flow entry was created and sent to the xDPd software switch 2.

```
void padswitch::handle_dpath2_open(cofdpt *dpt) {

rofl::cflowentryfe(dpt->get_version());
fe.set_command(OFPFC_ADD);
fe.set_table_id(0);

// setting matches
fe.match.set_eth_type(farpv4frame::ARPV4_ETHER);
fe.match.insert(rofl::coxmatch_ofb_eth_type(0x9100));
fe.match.insert(rofl::coxmatch_ofx_ictp_nid(0x11));

// setting actions
fe.instructions.next() = cofinst_apply_actions(dpt->get_version());
fe.instructions.back().actions.next()=cofaction_pop_ictp(dpt->get_version(),
                                                         0x800);
fe.instructions.back().actions.next()=cofaction_output(dpt->get_version(),
2);

// sending flow entry to xdpd/gnu-linux software switch
send_flow_mod_message(dpt, fe);
}
```

When flow entry was installed in xDPd software switch 2, the following logs were generated where we can see an OpenFlow table dump with flow entry inside:

```
[rofl-pipeline] OpenFlow switch instance (0x5cb4eb0)
[rofl-pipeline] ========================
[rofl-pipeline] Name: dp0
[rofl-pipeline] OpenFlow version: 3
[rofl-pipeline] OpenFlow datapathid: 256
[rofl-pipeline]
[rofl-pipeline] Dumping table # 0 (0x7f23c1038010). Default action: 0. # of
   entries: 1
[rofl-pipeline]                 [0] Entry (0x5cb4660), prior. 2048 #hits 2
   Matches:{[ETH_TYPE:0x9100], [ICTP_NID:285212672]}
Inst->>  APPLY,
                      APP.ACTIONs:<POP_ICTP>,<OUTPUT port: 2>,
[rofl-pipeline]         [*] No more entries...
[rofl-pipeline] --End of pipeline tables--
[rofl-pipeline] Dumping group table. # of group entries: 0.
[rofl-pipeline]         [*] No entries
[rofl-pipeline]
[rofl-pipeline] --End of group table—
```

When flow entry has been installed then network frames started to be switched to the second interface of the xDPd switch. xDPd has generated the following logs:

```
[rofl-pipeline] Packet[0x9e9450] entering switch [dp0] pipeline (1.X)
Packet matches [PORT_IN:1, ETH_SRC:0xa0b0c0d0e0f, ETH_DST:0x10203040506,
   ETH_TYPE:0x9100, ICTP_NID:285212672, ICTP_CSN:2016423936, ]
[rofl-pipeline] Packet[0x9e9450] matched at table: 0, entry: 0x5cb4660
[tap0] OUTPUT packet(0x9e9450)
Packet matches [PORT_IN:1, ETH_SRC:0xa0b0c0d0e0f, ETH_DST:0x10203040506,
   ETH_TYPE:0x800, ICTP_NID:285212672, ICTP_CSN:2016423936, ]
```

Finally, we have captured frames on both interfaces of the switch. Frames entering xDPd with ICTP header on the first interface:

```
PadSwitch2:~$ sudotcpdump -i eth1.700 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.700, link-type EN10MB (Ethernet), capture size 65535 bytes
06:20:28.955882  0a:0b:0c:0d:0e:0f  (oui  Unknown)  Unknown  SSAP  0x22  >
   01:02:03:04:05:06 (oui Unknown) Unknown DSAP 0x32 Information,  send seq 0,
   rcvseq 0, Flags [Command], length 44
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 1100  ................
       0x0010:  0000 3322 0000 6161 6161 6161 6161 6161  ..3"..aaaaaaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
```

```
        0x0030:  6161 6161 6161 6161 6161 6161 6161       aaaaaaaaaaaaaa
06:20:30.958392   0a:0b:0c:0d:0e:0f   (ouiUnknown)   Unknown   SSAP   0x22   >
   01:02:03:04:05:06  (ouiUnknown)  Unknown  DSAP  0x32  Information,  sendseq  0,
   rcvseq 0, Flags [Command], length 44
        0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 1100  ................
        0x0010:  0000 3322 0000 6161 6161 6161 6161 6161  ..3"..aaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
        0x0030:  6161 6161 6161 6161 6161 6161 6161       aaaaaaaaaaaaaa
```

Frames leaving xDPd switch without ICTP header on the second interface:

```
PadSwitch2:~$ sudotcpdump -i tap0 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 65535 bytes
06:10:25.449658 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
        0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
        0x0030:  6161 6161 6161                           aaaaaa
06:10:27.499382 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
        0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
        0x0030:  6161 6161 6161                           aaaaaa
```

1.2.3 Step 3: Updating a part of the protocol description

After some time, our SDN network is modified once again by pad-client.py script which uses PAD API to reload ICTP protocol information stored in network nodes and controller (see listing below).

```
def upload_ictp_v2():
pad.remove_protocol("ICTP")  # the old ICTP definition is removed

ictp_hdr = """
    header ictp {
      fields {
        version : 8;
        slice : 16;
        __skip__ : 8;
nid : 32;
csn : 32;
      }
    }
  """
```

```
pad.add_protocol("ICTP", ictp_hdr)    # apply a new information about ICTP

of_extensions_ids = pad.commit_configuration()

  print "OpenFlow experimental identifiers:"
pprint.pprint(of_extensions_ids)
```

Script pad-client.py has generated once again the following screen logs telling what files of both ROFL and xDPd were modified and what OpenFlow identifiers of experimental matches and actions were chosen for ICTP header:

```
pad$ python example.py
  File generated: /home/geysers-wp5/rofl-core/src/rofl/common/endianess_other.h
  File generated: /home/geysers-wp5/rofl-core/src/rofl/common/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/cofaction.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/cofaction.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_action.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_action.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/ictp_actions.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/ictp_actions.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/actions/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet_actions_autogenerated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/ictp_matches.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/ictp_matches.cc
```

```
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/experimental/matches/Makefile.am
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/openflow_experimental.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/openflow/coxmatch.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match_auto
   generated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_match_auto
   generated.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_packet_mat
   ches.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/openflow/openflow1x/pipeline/of1x_packet_mat
   ches.c
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/datapath/pipeline/platform/packet_autogenerated.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/fictpframe.h
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/fictpframe.cc
  File                    generated:                    /home/geysers-wp5/rofl-
   core/src/rofl/common/protocols/Makefile.am
  File             generated:             /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/packetclassifier.h
  File             generated:             /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/static_pktclassifier.h
  File             generated:             /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/io/packet_classifiers/static_pktclassifier.cc
  File             generated:             /home/geysers-wp5/xdpd/src/xdpd/fwd-
   modules/gnu_linux/src/pipeline-imp/packet.cc
  File                        generated:                        /home/geysers-
   wp5/xdpd/src/xdpd/openflow/openflow12/of12_translation_utils.cc
OpenFlow experimental identifiers:
{'ictp': [{'experimental_id': 27, 'field': 'version'},
          {'experimental_id': 28, 'field': 'slice'},
          {'experimental_id': 29, 'field': 'nid'},
          {'experimental_id': 30, 'field': 'csn'},
          {'action': 'pop_ictp', 'experimental_id': 3},
          {'action': 'pop_ictp', 'experimental_id': 4}]}
```

Currently, code generator is quite dump is regenerating all files instead of those which really had to be changed. After file distribution, compilation and software restarting we were able to move to next phase of the testing.

1.2.4 Step 4: Setting network configuration after protocol update and processing network frames

1) xDPd switch 1 - Adding ICTP header to network frames

The method, presented in the following listing, was called when xDPd software switch 1 has connected to ROFL-based controller. In the method, a flow entry was created and sent to the xDPd software switch 1.

```
void padswitch::handle_dpath1_open(cofdpt *dpt)
{
rofl::cflowentryfe(dpt->get_version());
fe.set_command(OFPFC_ADD);
fe.set_table_id(0);
fe.match.set_eth_type(farpv4frame::ARPV4_ETHER);

fe.match.insert(rofl::coxmatch_ofb_eth_type(0x800));

fe.instructions.next() = cofinst_apply_actions(dpt->get_version());
fe.instructions.back().actions.next() = cofaction_push_ictp(
dpt->get_version(), 0x9100);
fe.instructions.back().actions.next() = cofaction_set_field(
dpt->get_version(), rofl::coxmatch_ofx_ictp_version(0x02));
fe.instructions.back().actions.next() = cofaction_set_field(
dpt->get_version(), rofl::coxmatch_ofx_ictp_slice(0x4455));
fe.instructions.back().actions.next() = cofaction_set_field(
dpt->get_version(), rofl::coxmatch_ofx_ictp_nid(0x11));
fe.instructions.back().actions.next() = cofaction_set_field(
dpt->get_version(), rofl::coxmatch_ofx_ictp_csn(0x2233));
fe.instructions.back().actions.next() = cofaction_output(
dpt->get_version(), 2);

send_flow_mod_message(dpt, fe);
}
```

When flow entry was installed in xDPd software switch 1, the following logs were generated where we can see an OpenFlow table dump with flow entry inside:

```
[rofl-pipeline] OpenFlow switch instance (0x7a96f50)
[rofl-pipeline] ========================
[rofl-pipeline] Name: dp0
[rofl-pipeline] OpenFlow version: 3
```

```
[rofl-pipeline] OpenFlow datapathid: 256
[rofl-pipeline]
[rofl-pipeline] Dumping table # 0 (0x7fc61cd3d010). Default action: 0. # of
   entries: 1
[rofl-pipeline]                [0] Entry (0x7a927a0), prior. 2048 #hits 1
   Matches:{[ETH_TYPE:0x800], }
Inst->>  APPLY,
                      APP.ACTIONs:<PUSH_ICTP>,<SET_ICTP_VERSION:
   0x2>,<SET_ICTP_SLICE:  0x5544>,<SET_ICTP_NID:  0x11000000>,  <SET_ICTP_CSN:
   0x33220000>,<OUTPUT port: 2>,
[rofl-pipeline]         [*] No more entries...
[rofl-pipeline] --End of pipeline tables--
[rofl-pipeline] Dumping group table. # of group entries: 0.
[rofl-pipeline]         [*] No entries
[rofl-pipeline]
[rofl-pipeline] --End of group table–
```

When flow entry has been installed then network frames started to be switched to the second interface of the xDPd switch. xDPd has generated the following logs:

```
[rofl-pipeline] Packet[0x28b67f0] entering switch [dp0] pipeline (1.X)
Packet  matches  [PORT_IN:1,  ETH_SRC:0xa0b0c0d0e0f,  ETH_DST:0x10203040506,
   ETH_TYPE:0x800, IP_PROTO:97, IP_ECN:0x1, IP_DSCP:0x18,  IPV4_SRC:0x61616161,
   IPV4_DST:0x61616161, ]
[rofl-pipeline] Packet[0x28b67f0] matched at table: 0, entry: 0x7a927a0
[eth2.700] OUTPUT packet(0x28b67f0)
Packet  matches  [PORT_IN:1,  ETH_SRC:0xa0b0c0d0e0f,  ETH_DST:0x10203040506,
   ETH_TYPE:0x9100, IPV4_SRC:0x61616161, IPV4_DST:0x61616161,  ICTP_VERSION:2,
   ICTP_SLICE:21828, ICTP_NID:285212672, ICTP_CSN:857866240, ]
```

Finally, we have captured frames on both interfaces of the switch. Frames entering xDPd without ICTP header on the first interface:

```
PadSwitch1:~$ sudotcpdump -i tap0 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 65535 bytes
08:38:09.881782 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161                           aaaaaa
08:38:11.884284 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
```

```
        0x0030:  6161 6161 6161
```

Frames leaving xDPd switch with ICTP header included on the second interface:

```
PadSwitch2:~$ sudotcpdump -i eth2.700 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth2.700, link-type EN10MB (Ethernet), capture size 65535 bytes
08:41:59.087290  0a:0b:0c:0d:0e:0f  (oui  Unknown)  >  01:02:03:04:05:06  (oui
   Unknown), ethertype 802.1Q-9100 (0x9100), length 66:
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 0255  ...............U
       0x0010:  4400 1100 0000 3322 0000 6161 6161 6161  D.....3"..aaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0040:  6161                                     aa
08:42:01.089773  0a:0b:0c:0d:0e:0f  (oui  Unknown)  >  01:02:03:04:05:06  (oui
   Unknown), ethertype 802.1Q-9100 (0x9100), length 66:
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 0255  ...............U
       0x0010:  4400 1100 0000 3322 0000 6161 6161 6161  D.....3"..aaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0040:  6161
```

2) xDPd switch 2 - Removing ICTP header from network frames

The method, presented in the following listing, was called when xDPd software switch 2 has connected to ROFL-based controller. In the method, a flow entry was created and sent to the xDPd software switch 2.

```
void padswitch::handle_dpath2_open(cofdpt *dpt)
{
rofl::cflowentryfe(dpt->get_version());
fe.set_command(OFPFC_ADD);
fe.set_table_id(0);
fe.match.set_eth_type(farpv4frame::ARPV4_ETHER);

fe.match.insert(rofl::coxmatch_ofb_eth_type(0x9100));
fe.match.insert(rofl::coxmatch_ofx_ictp_nid(0x11));
fe.match.insert(rofl::coxmatch_ofx_ictp_slice(0x4455));

fe.instructions.next() = cofinst_apply_actions(dpt->get_version());
fe.instructions.back().actions.next()=cofaction_pop_ictp(dpt->get_version(),
                                                         0x800);
fe.instructions.back().actions.next()=cofaction_output(dpt->get_version(),
                                                       2);

send_flow_mod_message(dpt, fe);
```

```
}
```

When flow entry was installed in xDPd software switch 2, the following logs were generated where we can see an OpenFlow table dump with flow entry inside:

```
[rofl-pipeline] OpenFlow switch instance (0x7a59eb0)
[rofl-pipeline] ========================
[rofl-pipeline] Name: dp0
[rofl-pipeline] OpenFlow version: 3
[rofl-pipeline] OpenFlow datapathid: 256
[rofl-pipeline]
[rofl-pipeline] Dumping table # 0 (0x7fa76d84e010). Default action: 0. # of
   entries: 1
[rofl-pipeline]                 [0] Entry (0x7a59660), prior. 2048 #hits 3
   Matches:{[ETH_TYPE:0x9100], [ICTP_SLICE:21828] [ICTP_NID:285212672]}
Inst->>  APPLY,
                      APP.ACTIONS:<POP_ICTP>,<OUTPUT port: 2>,
[rofl-pipeline]         [*] No more entries...
[rofl-pipeline] --End of pipeline tables--
[rofl-pipeline] Dumping group table. # of group entries: 0.
[rofl-pipeline]         [*] No entries
[rofl-pipeline]
[rofl-pipeline] --End of group table—
```

When flow entry has been installed then network frames started to be switched to the second interface of the xDPd switch. xDPd has generated the following logs:

```
[rofl-pipeline] Packet[0x2806150] entering switch [dp0] pipeline (1.X)
Packet matches [PORT_IN:1, ETH_SRC:0xa0b0c0d0e0f, ETH_DST:0x10203040506,
   ETH_TYPE:0x9100, ICTP_VERSION:2, ICTP_SLICE:21828, ICTP_NID:285212672,
   ICTP_CSN:857866240, ]
[rofl-pipeline] Packet[0x2806150] matched at table: 0, entry: 0x7a59660
[tap0] OUTPUT packet(0x2806150)
Packet matches [PORT_IN:1, ETH_SRC:0xa0b0c0d0e0f, ETH_DST:0x10203040506,
   ETH_TYPE:0x800, ICTP_VERSION:2, ICTP_SLICE:21828, ICTP_NID:285212672,
   ICTP_CSN:857866240, ]
```

Finally, we have captured frames on both interfaces of the switch. Frames entering xDPd with ICTP header on the first interface:

```
PadSwitch2:~$ sudotcpdump -i eth1.700 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1.700, link-type EN10MB (Ethernet), capture size 65535 bytes
08:51:04.301151 0a:0b:0c:0d:0e:0f (oui Unknown) > 01:02:03:04:05:06 (oui
   Unknown), ethertype 802.1Q-9100 (0x9100), length 66:
```

```
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 0255  ...............U
       0x0010:  4400 1100 0000 3322 0000 6161 6161 6161  D.....3"..aaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0040:  6161                                     aa
08:51:06.303656  0a:0b:0c:0d:0e:0f  (oui  Unknown)  >  01:02:03:04:05:06  (oui
   Unknown), ethertype 802.1Q-9100 (0x9100), length 66:
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 9100 0255  ...............U
       0x0010:  4400 1100 0000 3322 0000 6161 6161 6161  D.....3"..aaaaaa
       0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0040:  6161
Aa
```

Frames leaving xDPd switch without ICTP header on the second interface:

```
PadSwitch2:~$ sudotcpdump -i tap0 "ether host 0a:0b:0c:0d:0e:0f" -XX
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on tap0, link-type EN10MB (Ethernet), capture size 65535 bytes
08:51:52.360493 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161                           aaaaaa
08:51:53.403529 IP6 , wrong link-layer encapsulationbad-hlen 4
0x0000:  0102 0304 0506 0a0b 0c0d 0e0f 0800 6161  ..............aa
       0x0010:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
0x0020:  6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
       0x0030:  6161 6161 6161
```