PSNC

# Programmable Abstraction of Datapath

Damian Parniewicz, Łukasz Ogrodowczyk, Bartosz Belter

EWSDN 2014, Budapest

# Hardware datapath abstraction

*… is a common and simplified view of traffic processing in the network device.*

This abstraction should:

- Factor out hardware low-level details from those elements which matter in the practice
- Not loose network hardware functionalities
- Be applicable to any network hardware

On the physical level hardware differs quite heavily („switching" is done in ASICs, pipeline processors, CPU-based processors, FPGAs, MEMS, liquid crystals, tunable optical filters, etc.).

## We are still searching for a good network hardware abstraction…

OpenFlow

…showed us a good direction – the abstraction should reflect a common set of hardware mechanisms:

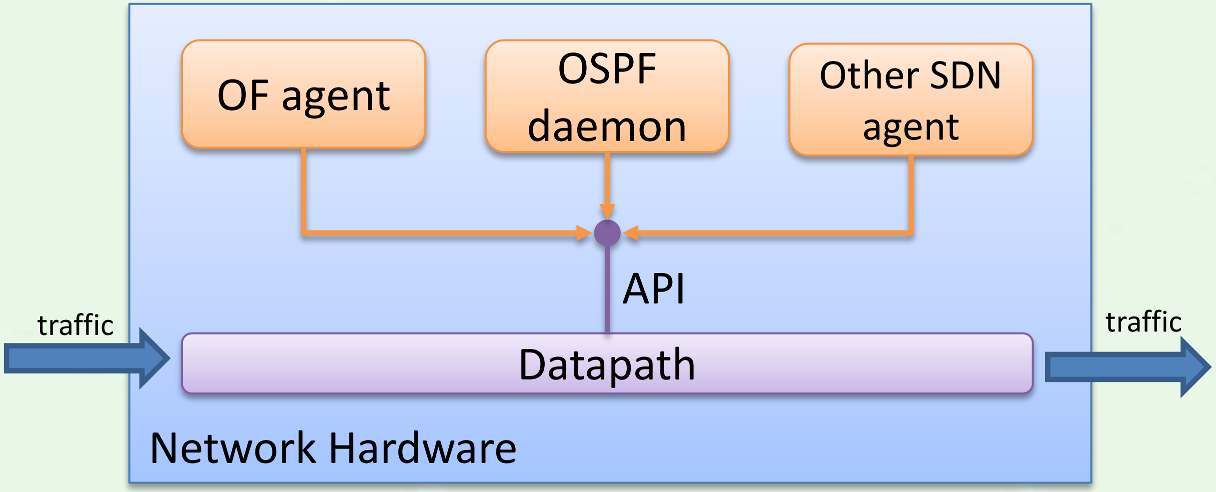
- Controllers controls exclusively some parts of device memory
- Lookups in search memory structures
- Usage of wildcard matching (in case of TCAM)
- A single matching performed on a set of fields belonging to different network protocols layers
- Traffic processing determined by a memory lookup result

# What are pitfalls of OF1.x ?

OpenFlow

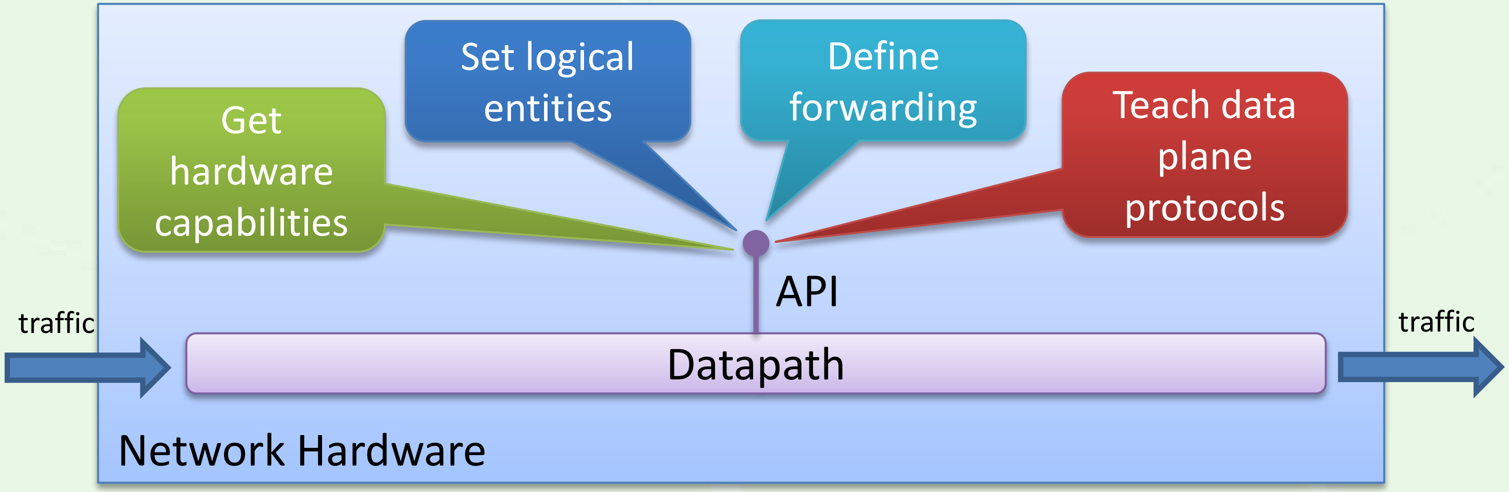
- Limited sizes of TCAM memory cannot contain all existing network protocol fields
- NAT cannot be efficiently realized
- How to handle ARP/ICMP responses in OF-based IP router in Denial-of-Service save-mode?
- What about Future Internet initiatives replacing existing IP protocol stack (e.g.: Named Data Networking/Content Centric Networking)?

# Network hardware abstraction should be local !



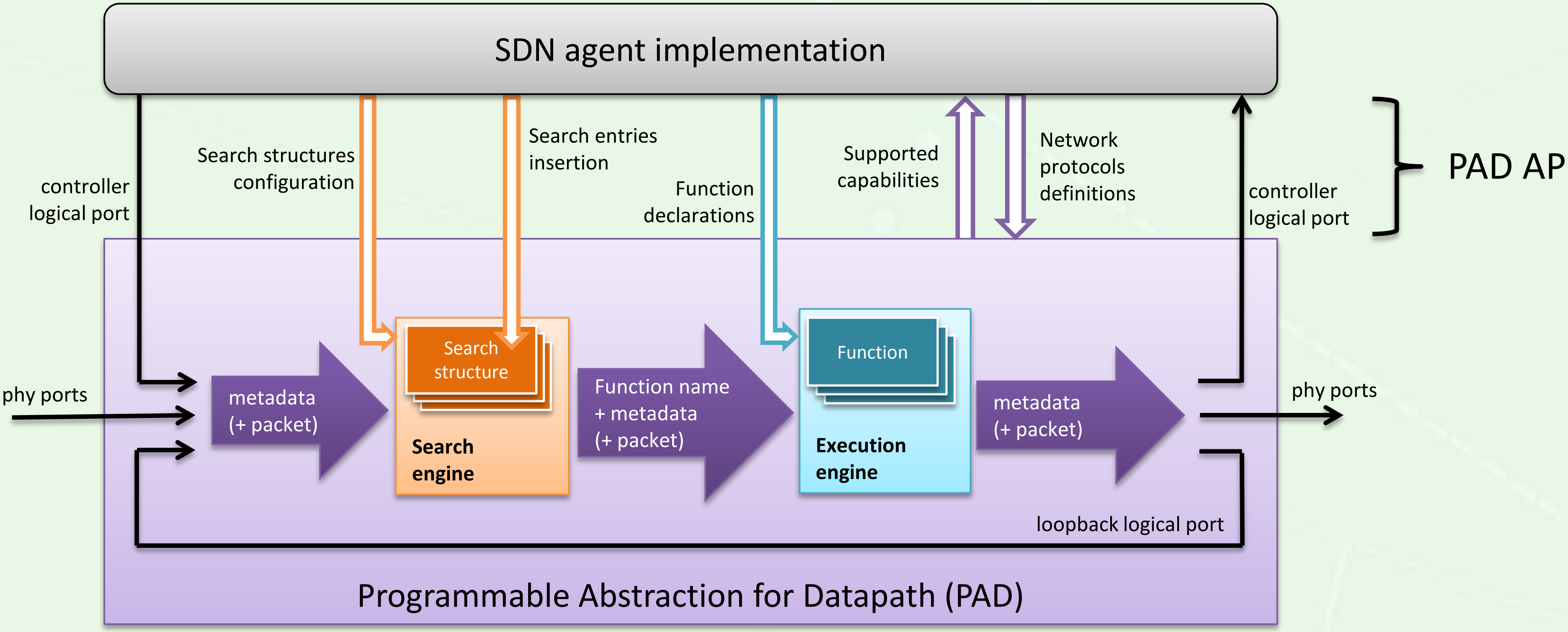

- Internal device API:
  - Provides abstracted view of the datapath
  - Used locally by any kind of management or control protocols
  - Supports both logically centralized and fully distributed architectures
  - Single implementation (e.g.: OF agent) for different devices
  - Inline with white-box switches initiative

# Network hardware abstraction should be programmable !




- Data plane protocols have quite simple structure (i.e.: strict order of byte/bit fields)
- Logical entities like lookup structures should use available memory efficiently
- Forwarding functions can contain the logic performed with a full datapath processing speed

# Programmable Abstraction of Datapath (PAD)



# Hardware capabilities exposed by PAD API

- Forwarding technology:
  - Packet-based (i.e.: Ethernet)
  - Circuit-based (i.e.: Fiber switching, Lambda switching)
- Search memory size
- Maximal length of a key in the search structure
- Byte endianness during packet processing
- Support for exact matches
- Controller and loopback logical ports
- Primitive instructions supported:
  - Remove, insert and modify a packet byte
  - Checksum computation
  - Arithmetical and fixed point operations
  - Logical and conditional operations

# PAD library C API

| API group | PAD API function |
|---|---|
| Capabilities | char* **get_all_capabilities**();<br>char* **get_capability**(); |
| Management | bool **add_protocol**(char* protocol_name, char* protocol_spec);<br>bool **remove_protocol**(char* protocol_name);<br>bool **remove_all_protocols**();<br><br>bool **add_structure**(uint8_t id, char* key, uint32_t type, uint32_t size);<br>bool **remove_structure**(uint8_t id);<br>bool **remove_all_structure**();<br><br>bool **add_function**(char* name, char* definition);<br>bool **remove_function**(char* name);<br>bool **remove_all_function**();<br><br>bool **commit_configuration**(); |
| Control | uint8_t **add_entry**(uint8_t structure_id, uint64_t key, uint64_t mask, char* result);<br>bool **remove_entry**(add_entry(uint8_t structure_id, uint64_t key, uint64_t mask);<br>bool **remove_all_entries**(uint8_t structure_id); |

Schemas used for both protocols and functions specification are transparent to PAD API.

# Data plane protocols definition schema (headers)

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

header ipv4 {
  fields {
    __skip__ : 8; // not interpreted bits
    dscp : 6;
    ecn : 2;
    __skip__ : 56;
    src_ip : 32;
    dst_ip : 32;
    __skip__ : 16;
    ip_proto : 8;
  }
}
```

```
header udp {
  field {
    src_port : 16;
    dst_port : 16;
    __skip__ : 32;
  }
}

header vxlan {
  field {
    __skip__ : 32;
    segment_id : 24;
    __skip__ : 8;
  }
}
```

Schema is based on P4 language*

* Pat Bosshart at al. “Programming Protocol-Independent Packet Processors”, arXiv:1312.1719v3 preprint, May 2014.

# Data plane protocols definition schema (protocol parsing tree)

```
parser start {
  ethernet;               \\ which header is parsed first
}
parser ethernet {
  switch(ethertype) {  \\ header field based lookup
    case 0x800: ipv4;  \\ what is next header
  }
}
parser ipv4 {
  switch(ip_proto) {
    case 0x11: udp;
  }
}
parser udp {
  switch(dst_port) {
    case 0x12B5: vxlan;
  }
}
```

Schema is based on P4 language*

* Pat Bosshart at al. "Programming Protocol-Independent Packet Processors", arXiv:1312.1719v3 preprint, May 2014.

# Primitive instructions used in the definitions of forwarding functions

| Instruction type | Target | Instruction |
|---|---|---|
| Frame modification | bits, bytes | set(value, offset)<br>move(from, to, length)<br>insert(value, offset, length)<br>remove(offset, length)<br>checksum(from, to) |
| | protocol fields | operator=, operator+, operator- (field_name, value)<br>copy_field(from, to) |
| | protocol headers | insert_header(header_name, offset)<br>remove_header(header_name)<br>operator. (header, field_name) |
| Forwarding | physical/logical ports | send_to(port_id)<br>drop() |

# Defining protocols and search structures (Python PAD API)

```
from pad import add_structure, add_function,
                add_entry, commit_configuration

add_protocol(protocol_name=„ethernet”,
             protocol_schema=“”””
                              header ethernet {
                                  fields {
                                       dst_addr : 48;
                                       src_addr : 48;
                                       ethertype : 16;
                                  }
                               }
                               parser start {ethernet;}
                               parser ethernet {
                                   switch(ethertype) { case 0x800: ipv4;}
                                   }
                               }”””””)

# … others protocols also added …

add_structure(id   = 0,
              key  = “metadata.ingress_port, udp.dst_port, vxlan.segment_id”,
              type = “hash”,
              size = 1000)
```

# Defining forwarding functions (Python PAD API)

```
@add_function
encapsulate_vxlan(port, segment_id):
    offset = insert_header(ethernet, 0)
    offset = insert_header(ipv4, offset)
    insert_header(udp, offset)
    ethernet.src_addr = 0x52540066f59a
    ethernet.dst_addr = 0x5cf3fce85308
    ethernet.ethertype = 0x800
    ipv4.src_ip = 0x0a000001
    ipv4.dest_ip = 0x0a000002
    ipv4.ip_proto = 0x11
    udp.dst_port = 0x12b5
    vxlan.segment_id = segment_id
    send_to(port)

@add_function
decapsulate_vxlan(port):
    remove_header(ethernet)
    remove_header(ipv4)
    remove_header(udp)
    send_to(port)

commit_configuration()
```

Inside function body:

- Access to frame bytes, fields
- Frame modification
- Creation of integer variables
- Access to memory structures
- Conditional statements
- Calls to other defined functions

# Adding search entries to memory (Python PAD API)

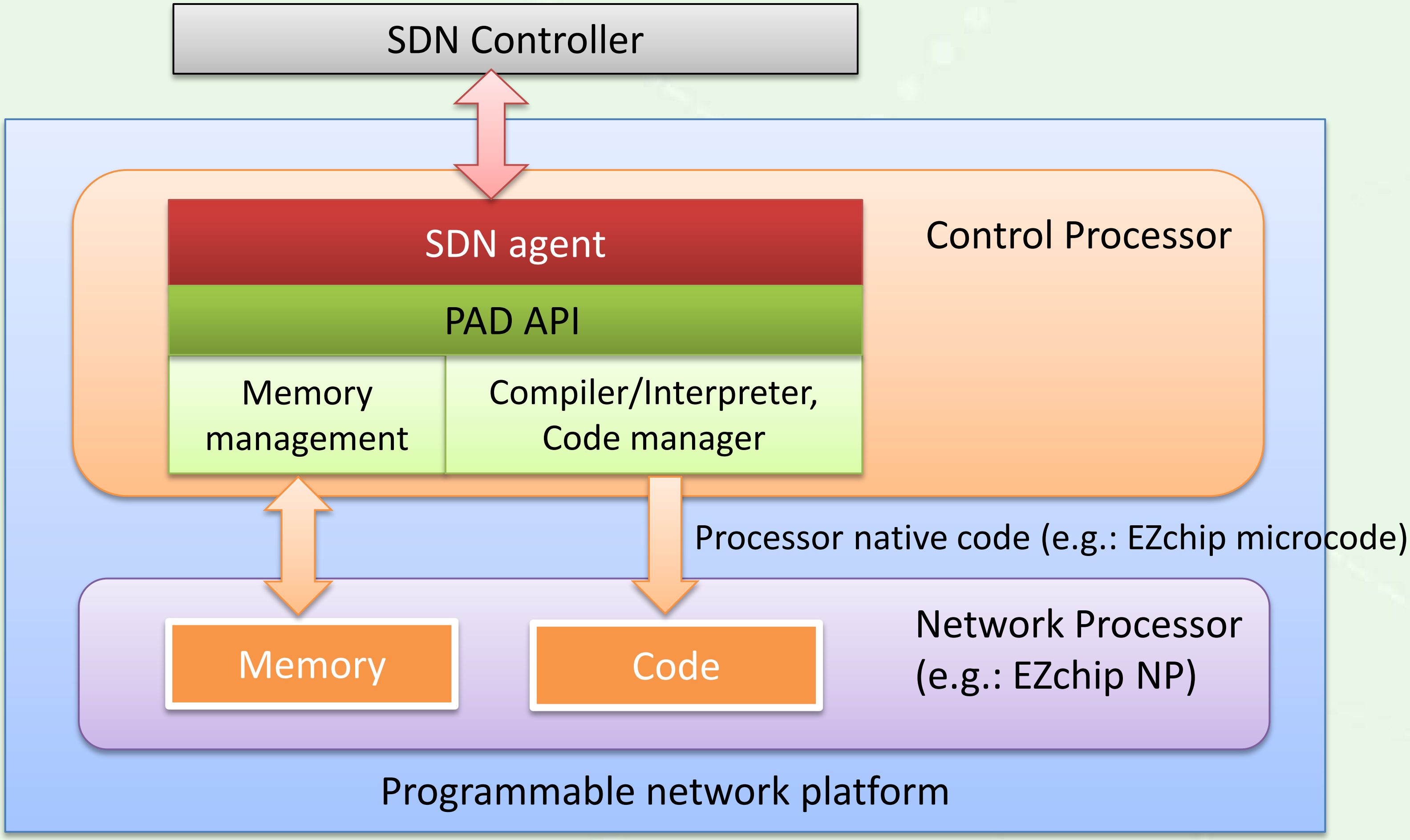
```
add_entry(structure_id=0,
          key    = 0x000a125b001,  # ingress_port=10, dst_port=4789, segment_id=1
          mask   = 0xffffffffff,
          result = ”decapsulate_vxlan(port=2)“
)

add_entry(structure_id=0,
          key    = 0x00020000000,    # ingress_port=2
          mask   = 0xffff0000000,
          result =”encapsulate_vxlan(port=10, segment_id=1)“
)
```

Search memory entries:

1. 0x000a125b001, 0xffffffffff — decapsulate_vxlan, 1, 2
2. 0x00020000000, 0xffff0000000 — encapsulate_vxlan, 2, 10, 1

# PAD implementation in a network processor



## Conclusions

http://www.fp7-alien.eu/

- PAD represents a low level network abstraction:
  - Focuses on physical capabilities of network devices (memory organizations, memory content, platform code execution)
  - Node logical abstraction should be done by SDN agent (e.g.: OF pipeline)
  - Allows for non-typical forwarding (e.g.: NAT)

- Decouples network hardware from network protocols:
  - Teach only those protocols which are really used
  - Smaller entry -> More rules in memory
  - Easy migrate network to new protocols and applications (i.e.: IPv6, VXLAN, CCN)

# Questions?

Poznań Supercomputing and Networking Center

affiliated to the Institute of Bioorganic Chemistry of the Polish Academy of Sciences,

ul. Noskowskiego 12/14, 61-704 Poznań, POLAND,
Office: phone center: (+48 61) 858-20-00, fax: (+48 61) 852-59-54,
e-mail: office@man.poznan.pl, http://www.psnc.pl