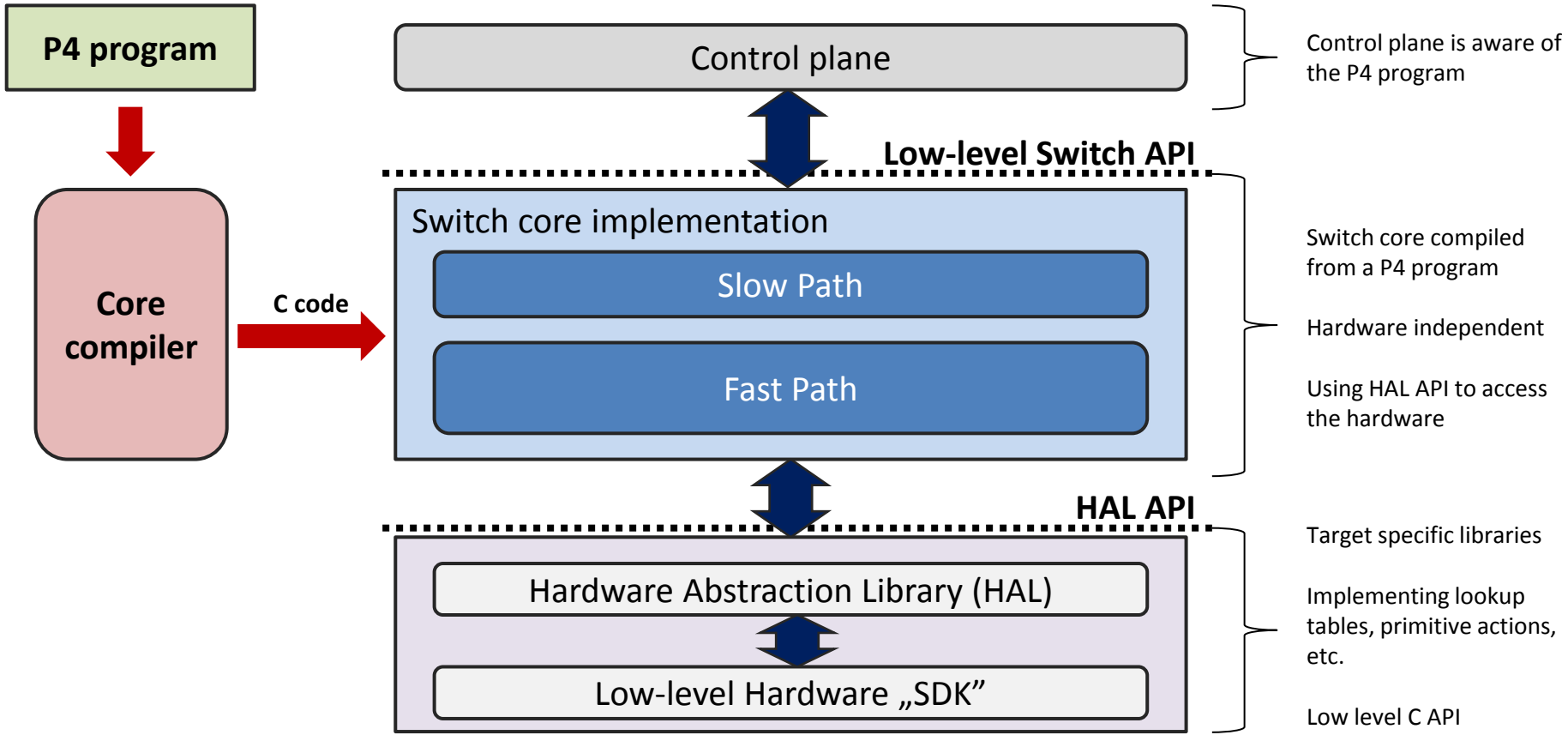
High-Speed Forwarding: A P4 Compiler with a Hardware Abstraction Library for Intel DPDK

Sándor Laki
Eötvös Loránd University
Budapest, Hungary
lakis@elte.hu

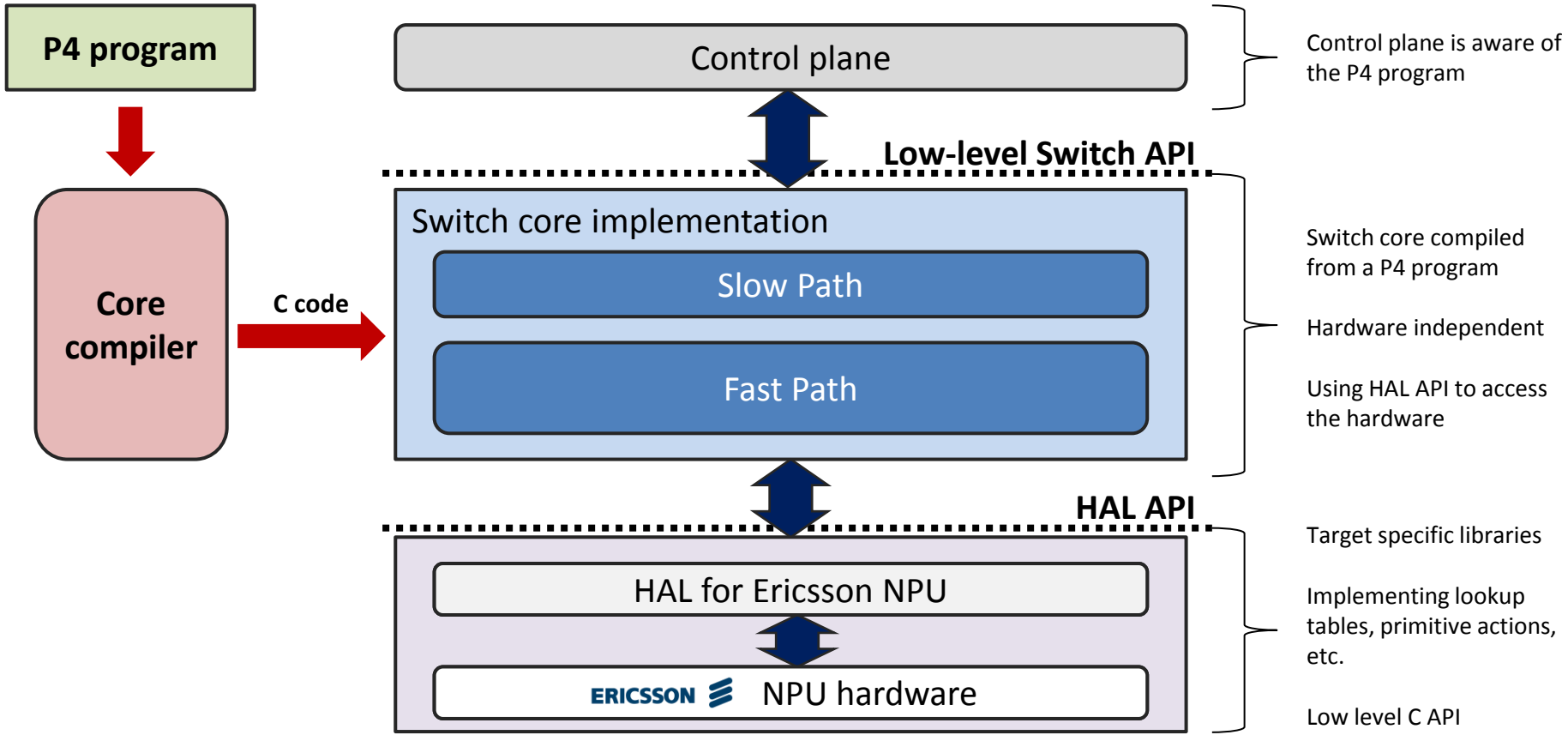
Motivation

- Programmability of network data plane
  - P4 code as a high level abstraction
- Different hardware targets
  - CPUs, NPUs, FPGA, etc.
- Create a compiler that separates hardware dependent and independent parts
  - Easily retargetable P4 compiler

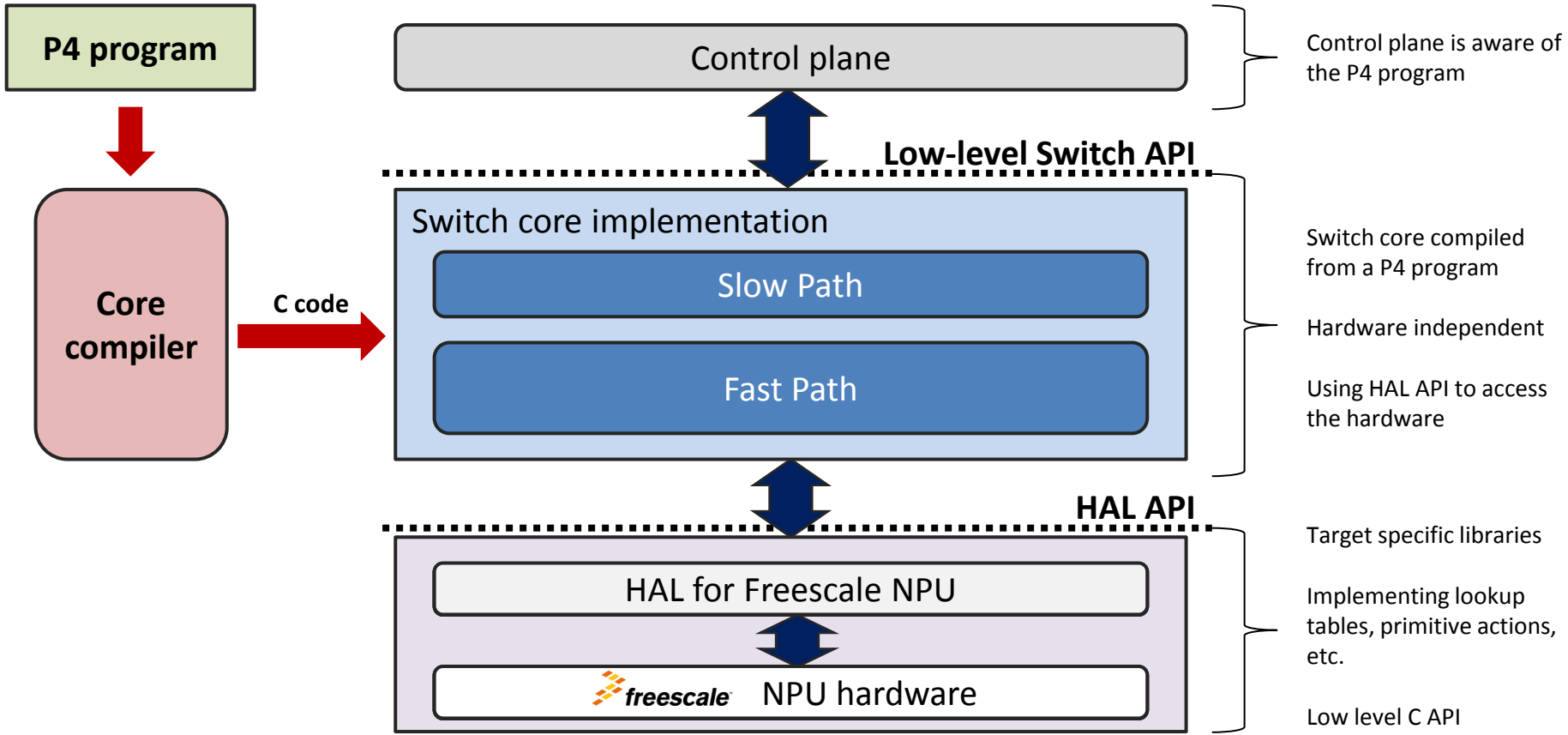
Multi-target Compiler Architecture



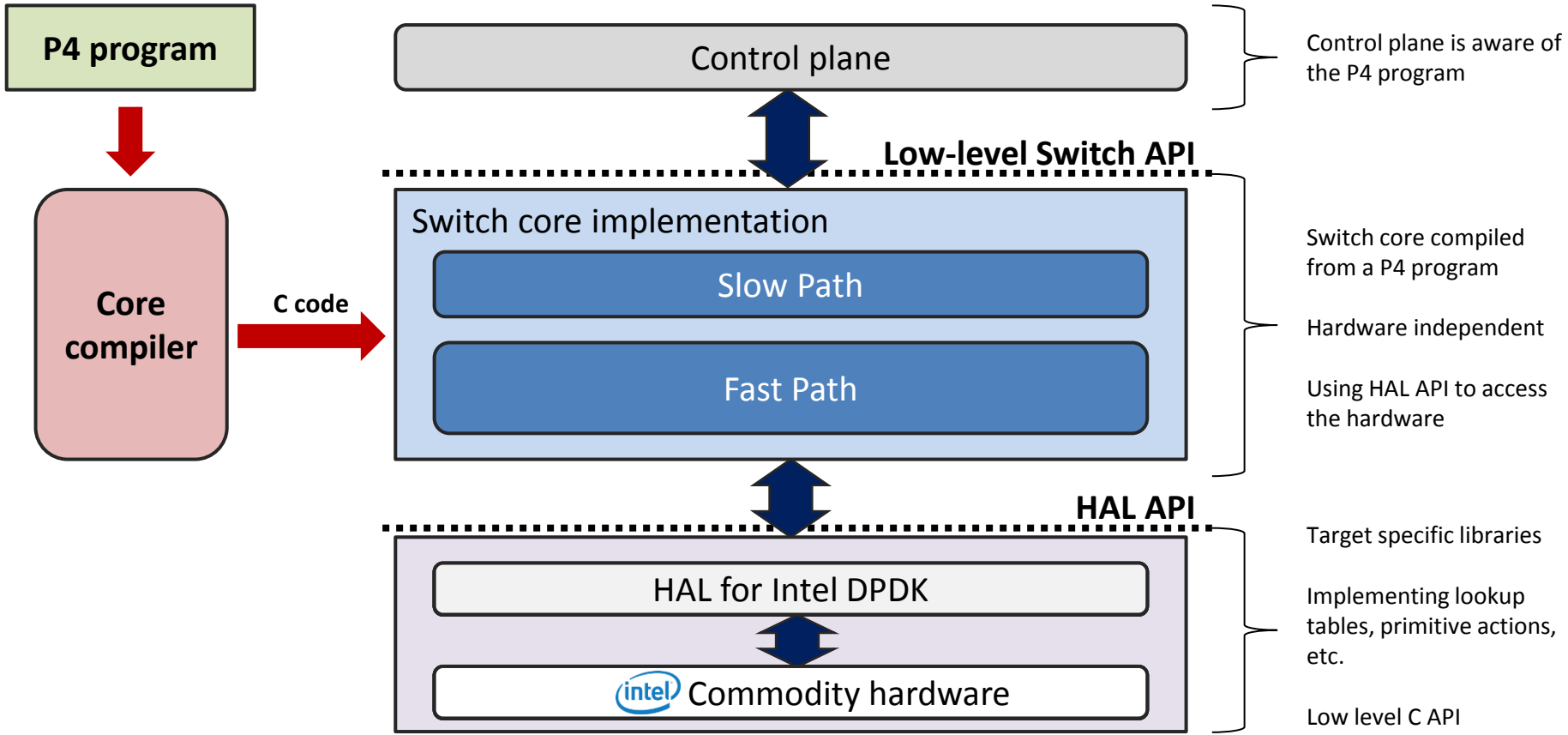
Multi-target Compiler Architecture



Multi-target Compiler Architecture

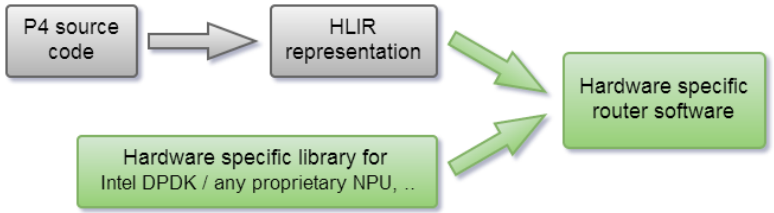


Multi-target Compiler Architecture



Multi-target Compiler Architecture

- Linking two carefully separated components of C source code
- Hardware-independent „Core”
  - generated by a compiler
  - based on the HLIR
- Hardware-dependent „HAL”
  - static library
  - written by a hw-expert
- Well-defined interface between them
  - mostly with C function prototypes



- PROs
  - much simpler compiler
  - modularity = better maintainability
  - exchangeable HAL = retargetable switch (without rewriting a single line of code)
  - HAL is not affected by changes in the P4 program
- CONs
  - performance questions
  - hardware-dependent parts are not amenable to protocol-dependent optimization
  - communication overhead between the components (C function calls)

The „Core”

Packet parsing:

- Lightweight Parsed Representation
- Determining the positions and types of headers in the packet
- No "real" parsing or field extraction (we do it lazy)

Actions and controls:

- Fields are extracted when needed
- In-place field modifications
- Controls and actions translated to C functions
- Key calculation for lookup tables

Data structures used:

- data structures declared in the HAL but initialized in the core
  - stateful memory configuration
- the same idea is used for sharing information on header types
  - we could use macros instead, but arrays are more readable and they are optimized away by the C compiler (via constant propagation)

Hardware Abstraction Library

- Low-level generic C API for networking hardwares
- A target specific HAL implements:
  - states (tables, counters, meters etc.)
  - related operations (table insert/delete/lookup, counter increment, etc.)
  - packet RX and TX
  - primitive actions (header-related + digests)
  - helpers for primitive actions (field-related)
    - implemented as macros for performance reasons

add_header
```
add_header(packet_descriptor_t* p, header_reference_t h)
```

remove header
```
remove_header(packet_descriptor_t* p, header_reference_t h)
```

modify_field
```
modify_field_to_const32(packet_descriptor_t* p,
           field_reference_t f, uint32_t val)
modify_field_to_const(packet_descriptor_t* p,
           field_reference_t f, uint8_t *src, int srclen)
modify_field_to_field(packet_descriptor_t* p,
           field_reference_t dstf, field_reference_t srcf)
```

generate digest (message to the control plane)
```
generate_digest(backend bg, char* name, int receiver,
               struct type_field_list* digest_field_list)
```

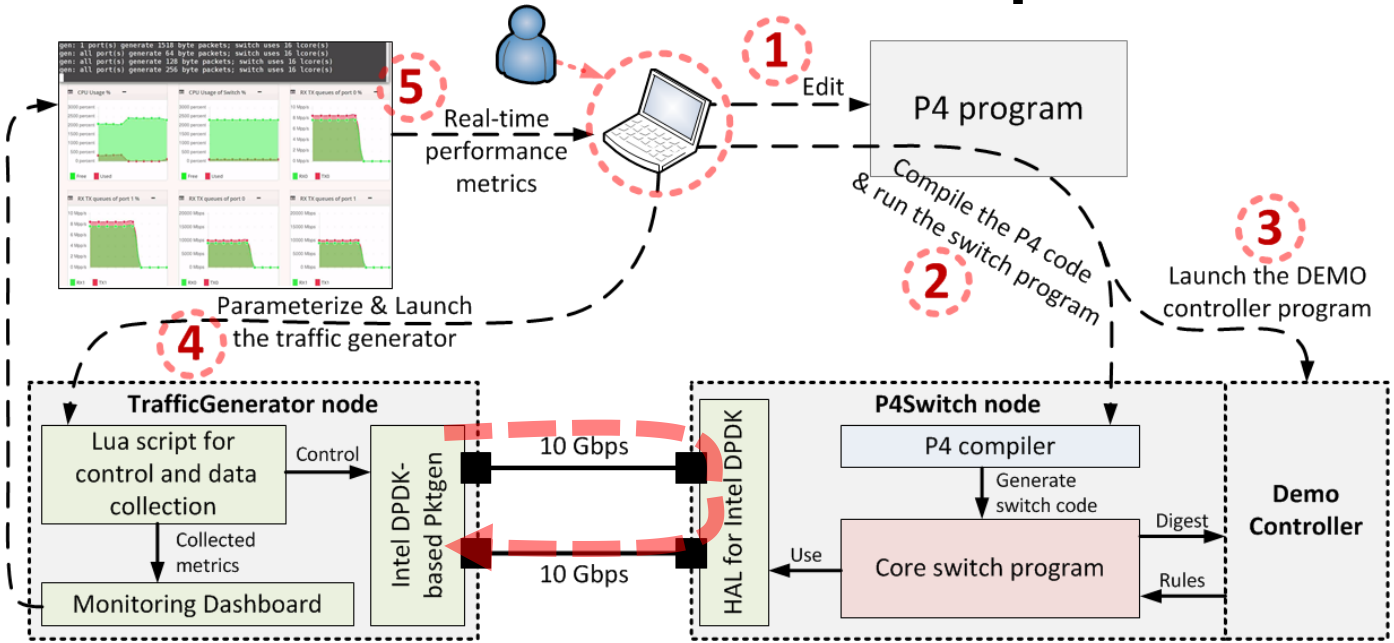
Table type and the table operations (e.g. for ternary tables):
```
typedef struct lookup_table_s;

naive_ternary_create(uint8_t keylen, uint8_t max_size);
naive_ternary_destroy(ternary_table* t);
naive_ternary_add(ternary_table* t, uint8_t* key, uint8_t* mask, uint8_t* value);
naive_ternary_lookup(ternary_table* t, uint8_t* key);
```

HAL for Intel DPDK

- Reuses the current LPM and HASH table implementations of DPDK
- Atomic integers for counters and meters
- NUMA support
  - Two instances of each table on each socket - lock-free solution
    - active/passive instances
    - lcore always turns to its socket's instance
  - Counter instances for each lcore on the corresponding socket
    - With the aim of keeping them in CPU cache

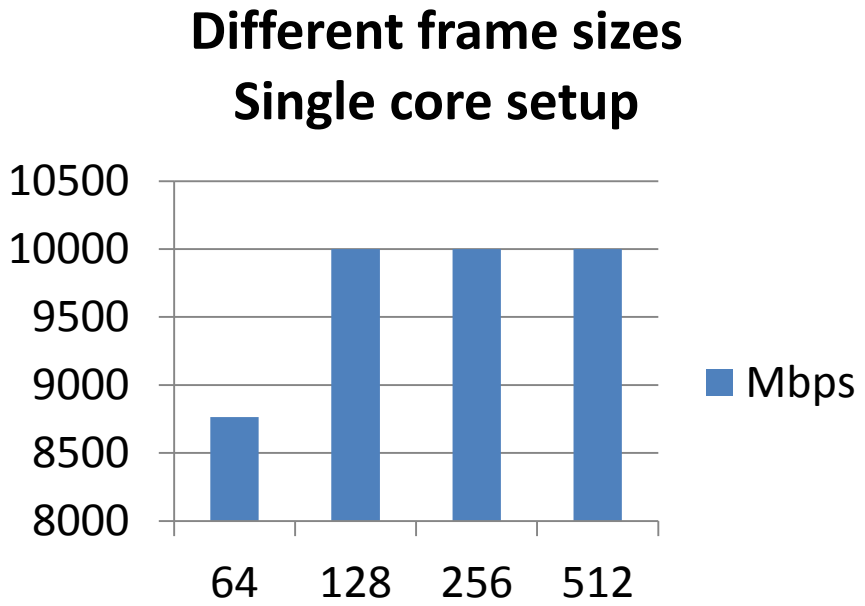
Evaluation setup



- Intel XEON E5-2630 4x8 cores 2.3GHz,8x4GB DDR3 SDRAM
- dual 10 Gbps NIC (Intel 82599ES)

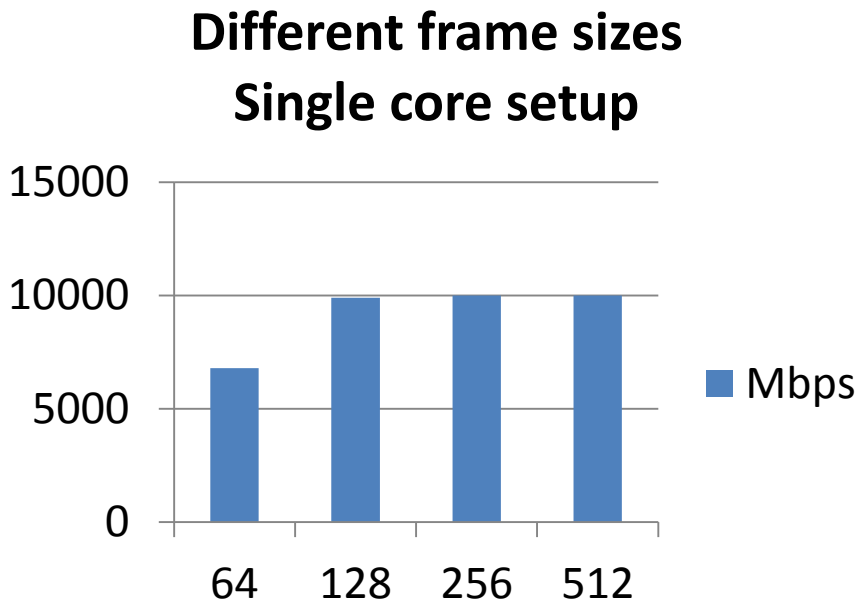
Performance test
L2 example

- Simple L2 forwarding with mac learning
- Two lookup tables
  - smac & dmac
- Generating digests
- Demo controller fills tables smac and dmac



Performance test
L3 example

- Simple L3 forwarding example
- Three lookup tables
  - Ipv4_lpm, nexthops, send_frame
- Demo controller fills tables in advance

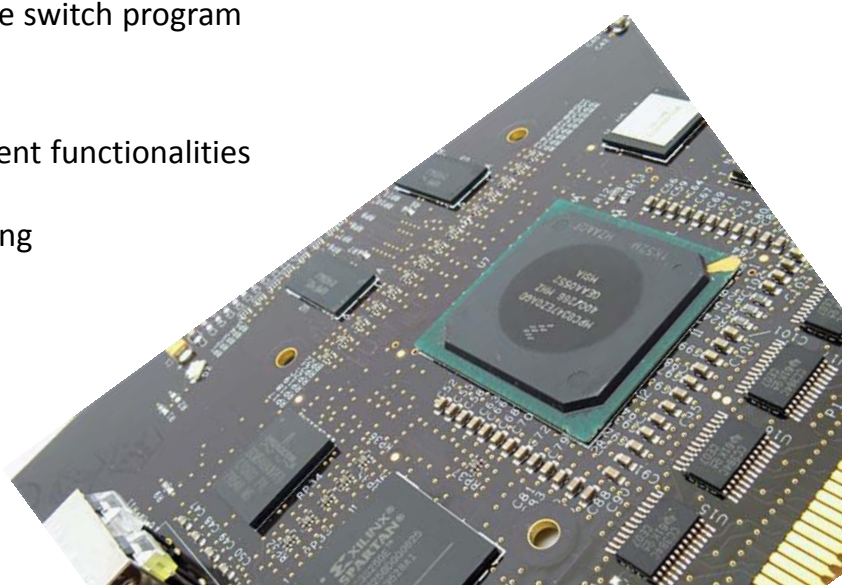


Conclusion & Future Work

- Lessons learnt
  - not easy to find the boundary of the HAL
    - compared to the first version, our HAL has become thinner
  - P4 primitive actions are not fully implemented in the HAL, in most cases only small hw-dependent helper functions are defined
    - As a PRO: the hand-written code is smaller
  - inspection of the assembly code is needed to optimize the switch program

- Current state
  - Our compiler separates the hw dependent and independent functionalities
  - Supports P4 1.0 specification (almost complete)
  - HAL for Intel DPDK is under testing and performance tuning

- Future work:
  - HAL for Freescale and other NPUs
  - Code optimization to get better performance
  - Performance and scalability tests
  - First public release of the compiler



Thank you for you attention!

The first release of our P4 compiler will soon be available at
http://p4.elte.hu

The team: Dániel Horpácsi, Róbert Kitlei, Sándor Laki, Dániel Leskó, Máté Tejfel, Péter Vörös