Dataplane Specialization for High-performance OpenFlow Software Switching

László Molnár, Gergely Pongrácz, Gábor Enyedi, Zoltán Lajos Kis
Levente Csikor, Ferenc Juhász, Attila Kőrösi, Gábor Rétvári

TrafficLab, Ericsson Research, Hungary
Department of Telecommunications and Media Informatics, BME
MTA-BME Information Systems Research Group

SIGCOMM'16, August 22-26, 2016, Florianopolis, Brazil

ERICSSON

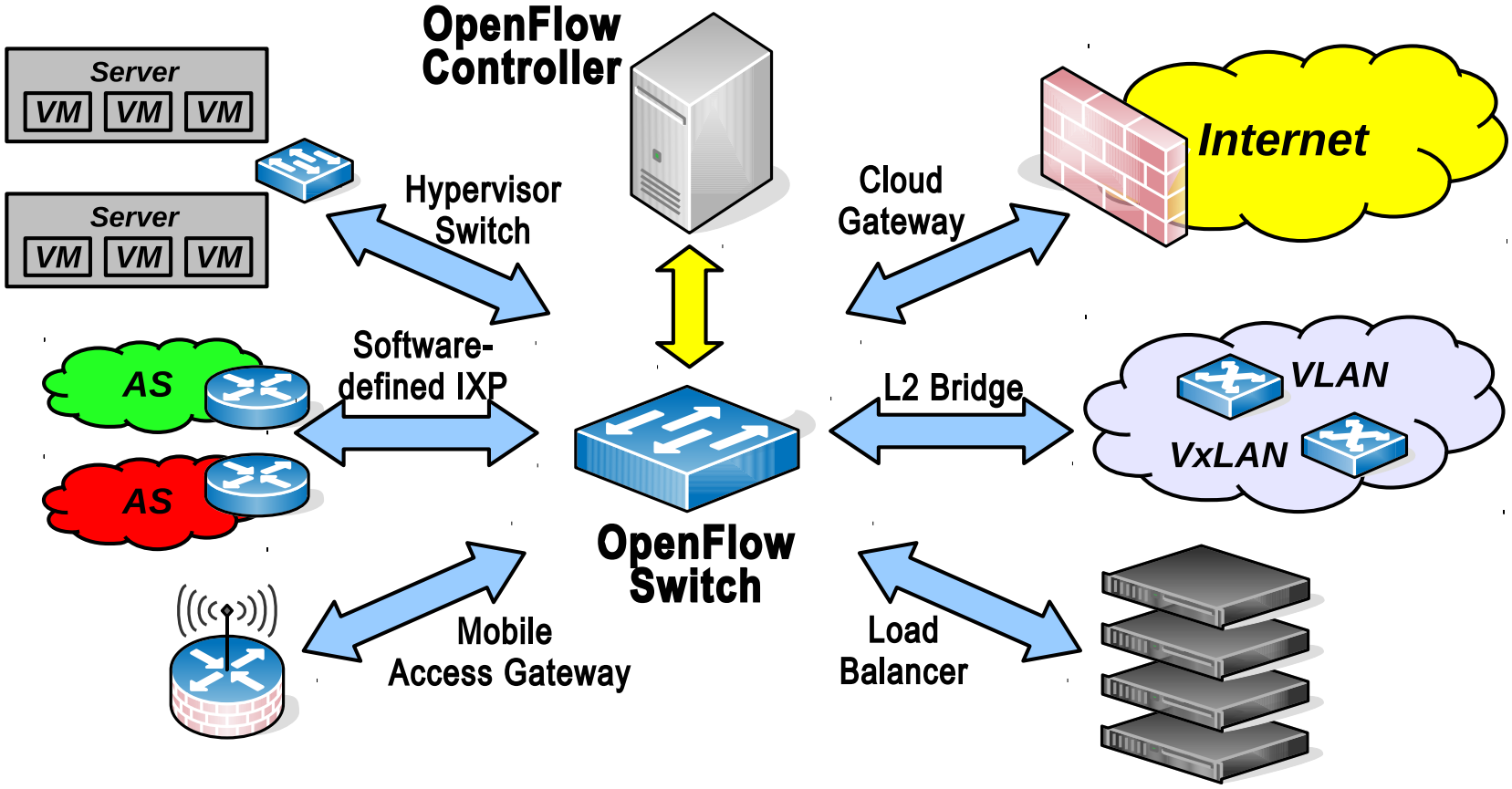
TL;DR

"OpenFlow is expressive but troublesome to make fast on x86."

B. Pfaff et al. "The design and implementation of Open vSwitch," NSDI, 2015.

Dataplane specialization may help to alleviate the "expressibility vs. performance" tension.

Expressibility vs. Performance



How to support diverse workloads in a single device efficiently?

Datapath Programmability Is Hard

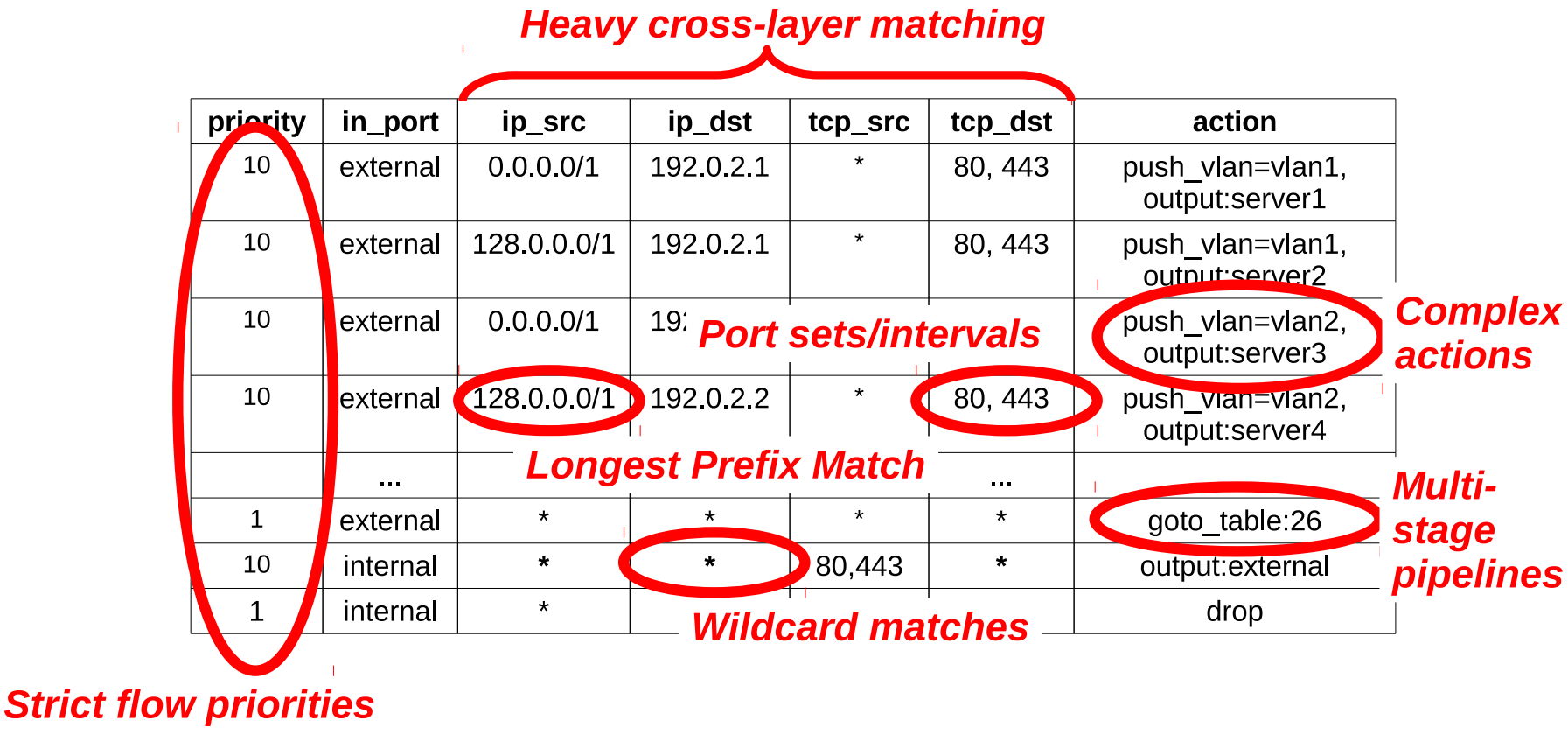
- **Packet forwarding:** map received packets to the action(s) to be executed on them (and execute these)

$$\texttt{packet} \mapsto \overbrace{\texttt{header tuple} \mapsto \texttt{flow entry}}^{\text{fast-path packet classifier}} \mapsto \texttt{action(s)}$$

- Supporting OpenFlow's expressibility makes the fast-path packet classifier rather complex
- But software-based packet classification is slow

OpenFlow softswitch architectures are all about working around the complexity of fast-path packet classification

Simple Load Balancer + ACL



priority	in_port	ip_src	ip_dst	tcp_src	tcp_dst	action
10	external	0.0.0.0/1	192.0.2.1	*	80, 443	push_vlan=vlan1, output:server1
10	external	128.0.0.0/1	192.0.2.1	*	80, 443	push_vlan=vlan1, output:server2
10	external	0.0.0.0/1	19[illegible]			push_vlan=vlan2, output:server3
10	external	128.0.0.0/1	192.0.2.2	*	80, 443	push_vlan=vlan2, output:server4
	...				...	
1	external	*	*	*	*	goto_table:26
10	internal	*	*	80,443	*	output:external
1	internal	*				drop

Generic Switch Architectures

- Universal dataplane that supports all use cases “well” (CPqD, xDPd, LINC, OVS, 6WINDGate)
- Tackle difficultly of packet classification by avoiding it
  - do the classification for flows’ first packets
  - use result for subsequent packets: **flow caching**
- But flow caching introduces its own share of problems
  - breaks on widely changing traffic/header fields: **hidden assumptions** and **performance artifacts**
    [PAM 2009], [HotSDN 2013], [CCR 2014], [EANTC 2015]
  - cache management hard: **complex architecture**
    [NSDI 2015]
  - breaks tenant isolation: **DOS attacks on caches**
    [NSDI 2014], [CCR 2014]

Our Idea: Dataplane Specialization

- Generic switch architectures over-generalize: optimize for the lowest common denominator
- Instead, let the switch **automagically optimize its dataplane for the given workload**
  - into an Ethernet softswitch for L2 use cases
  - an LPM engine for IP
  - an optimal combination for mixed workloads
- This allows to **choose the best fast-path classifier for each flow table** in the pipeline separately
- Very efficient for simple pipelines, achieve what's possible for complex ones

ESWITCH

- A new dataplane compiler to **transform OpenFlow programs into custom fast-paths**

$$\texttt{OpenFlow pipeline} \xrightarrow{\text{ESWITCH}} \texttt{custom fast-path}$$

- Rebuild the datapath for each `add-flow`/`del-flow`: compilation speed is crucial
- ESWITCH invokes **template-based code generation**
  - deconstruct the pipeline into simple packet processing primitives
  - represent primitives with precompiled codelets
  - link templates into executable machine code

ESWITCH: Templates

- Unit of pkt processing behavior that admits a simple and composable machine code implementation
- **Parser template:** raw packets $\rightarrow$ matchable tuples
- Separate parser for each protocol in OpenFlow spec

```
PROTOCOL_PARSER: <set protocol bitmask in r15>
```

```
L2_PARSER:  mov r12, <pointer to L2 header>
```

```
L3_PARSER:  mov r13, <pointer to L3 header>
```

```
L4_PARSER:  mov r14, <pointer to L4 header>
```

- **Matcher template:** match on some header field
- E.g., a matcher for entry `ip_dst = ADDR/MASK`:

```
macro IP_DST_ADDR_MATCHER(ADDR,MASK):
  mov   eax,[r13+0x10]   ; IP dst address in eax
  xor   eax,ADDR         ; match ADDR
  and   eax,MASK         ; apply MASK
  jne   ADDR_NEXT_FLOW   ; no match: next entry
```

ESWITCH: Templates

- **Flow table template:** basic classification types

Name: direct code
Prerequisite: #flows ≤ 4
Match type: arbitrary
Implementation: machine code
Application: universal
Fallback: compound hash

Name: compound hash
Prerequisite: global mask
Match type: exact match
Implementation: perfect hash
Application: MAC switching & port filtering
Fallback: LPM

Name: LPM
Prerequisite: prefix masks
Match type: longest prefix match
Implementation: DPDK LPM lib
Application: IP forwarding
Fallback: linked list

Name: linked list
Prerequisite: none
Match type: tuple space search
Implementation: machine code
Application: complex pipelines
Fallback: none

- Start with best template, fallback if prerequisite fails
- **Action template:** packet processing functionality
- Separate for each action type, shared across flows

Directly Compiled Datapath

- An OpenFlow pipeline with the below flow entry

```
...
priority=i,ip_dst=ADDR/MASK,action=ACTION
...
```

- ESWITCH compiles it into a sequence of templates

```
PROTOCOL_PARSER: <set protocol bitmask in r15>
L2_PARSER:  mov r12, <pointer to L2 header>
L3_PARSER:  mov r13, <pointer to L3 header>
...
FLOW_i:                              ; flow entry starts
  bt   r15d, IP                      ; packet contains IP header?
  jae  ADDR_NEXT_FLOW                ; jump to next flow entry if not
  IP_DST_MATCHER(ADDR,MASK)          ; ip_dst=ADDR/MASK?
  jmp ACTION                         ; jump to ACTION
FLOW_(i+1):
...
ACTION: ...                          ; execute ACTION
```

Compilation Process

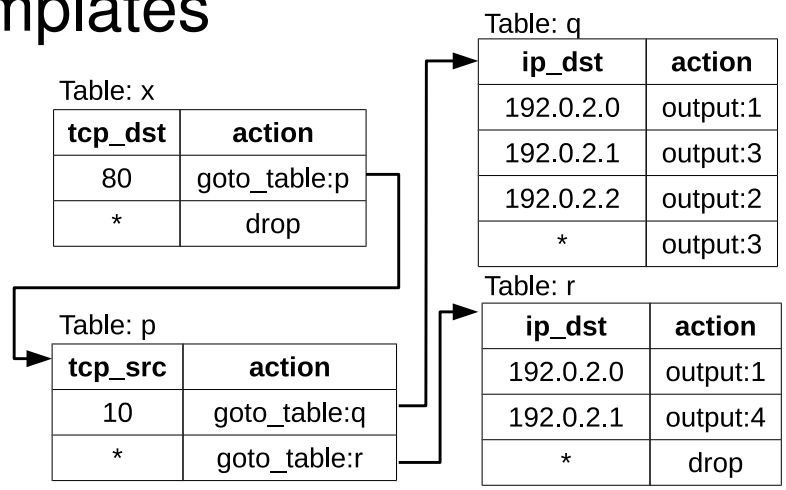
- ESWITCH divides code generation into 3 stages

1. **Flow table analysis:** divide pipeline into templates

- ESWITCH uses flow table decomposition to promote tables to efficient table templates

Table: x

ip_dst	tcp_src	tcp_dst	action
192.0.2.0	*	80	output:1
192.0.2.2	10	80	output:2
*	10	80	output:3
192.0.2.1	*	80	output:4
*	*	*	drop



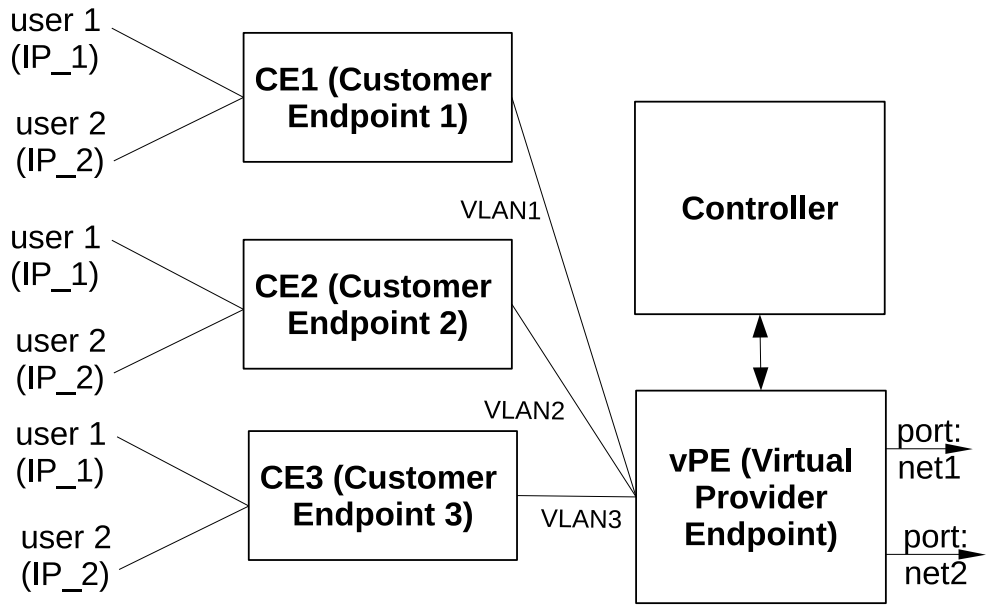
- **Theorem:** optimal table decomposition is NP-hard
- We use fast greedy heuristics

Compilation Process

2. **Template specialization:** patch templates with flow keys, masks, etc.
   - Code contains constants to avoid memory references
3. **Linking:** resolve dangling pointers to direct address
   - `goto_table` pointers go through per-table trampolines
   - Thus **updates are transactional and per-flow-table**
     - new code built side-by-side with running datapath
     - trampoline updated when ready
     - all `goto_table` pointers thus updated atomically

Implementation/Evaluation

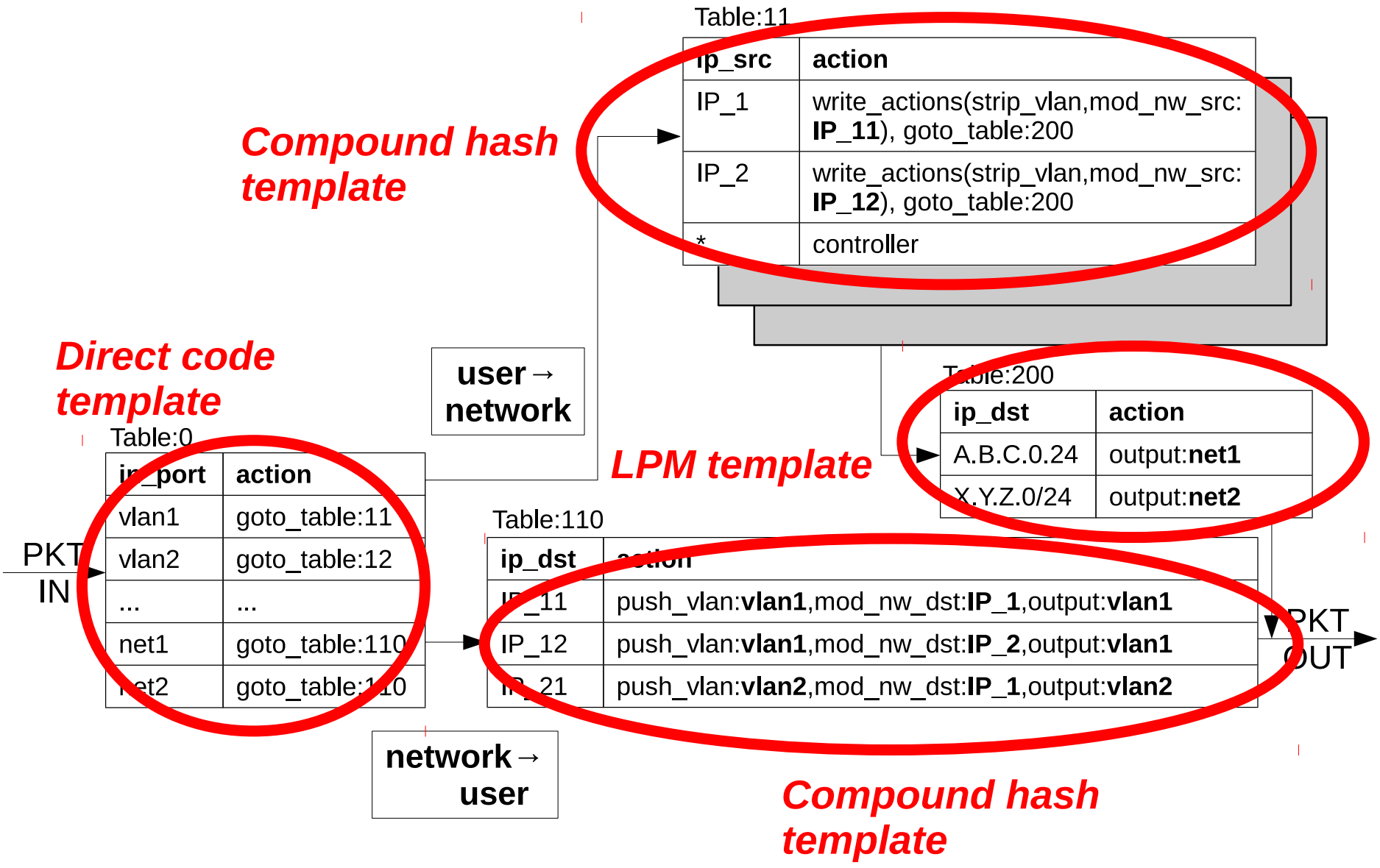
- PoC ESWITCH prototype on top of the Intel DPDK
- Measured against Open vSwitch (OVS): generic dataplane with multi-level flow cache hierarchy
- **Mobile access gateway** use case (among others)



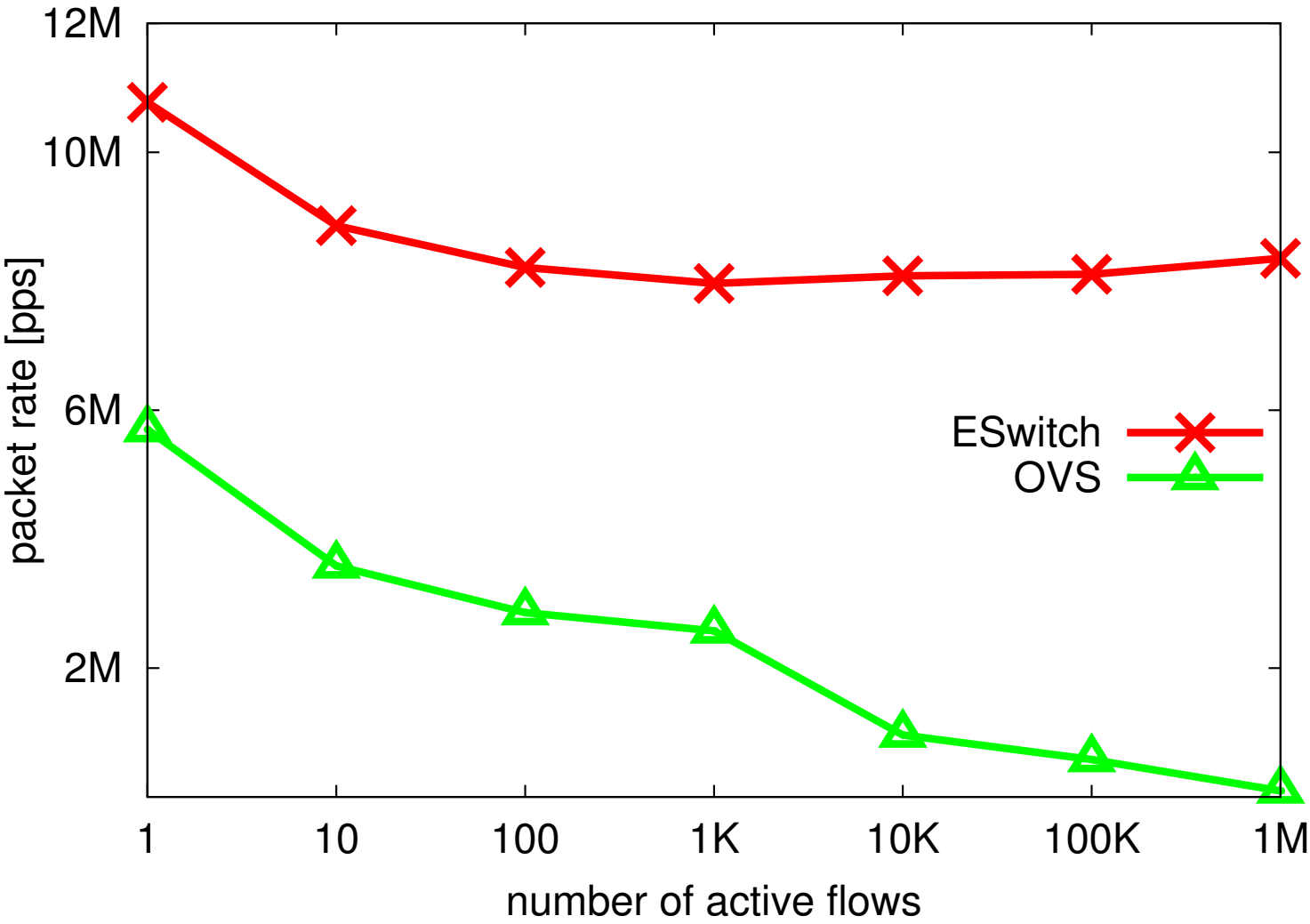
10 CEs, 20 users per CE, IP routing table: 10K IP prefixes, couple of dozen flow tables

Intel, "Network function virtualization: Quality of Service in Broadband Remote Access Servers with Linux and Intel architecture.", 2014.

Access Gateway: Custom Dataplane

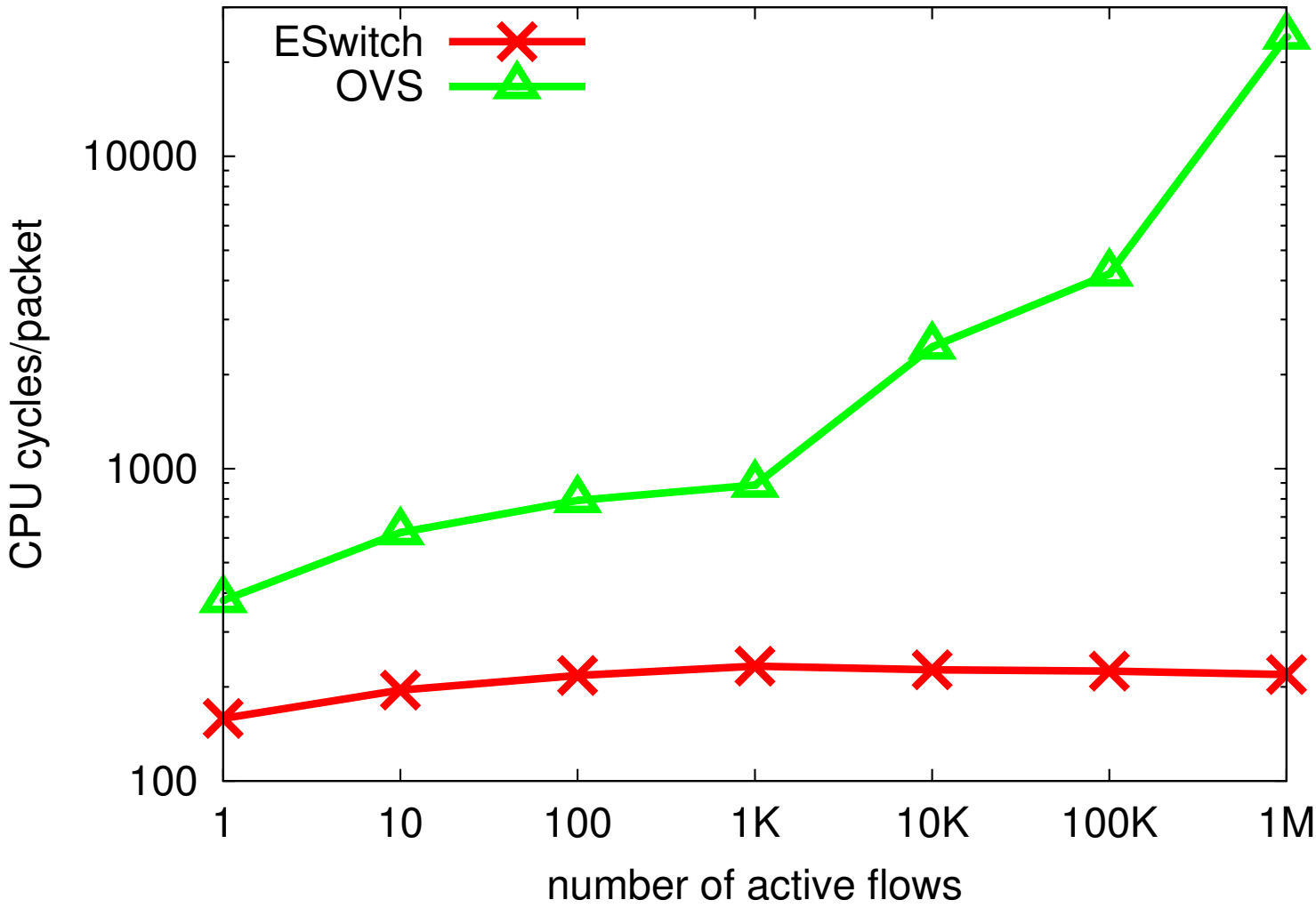


Throughput



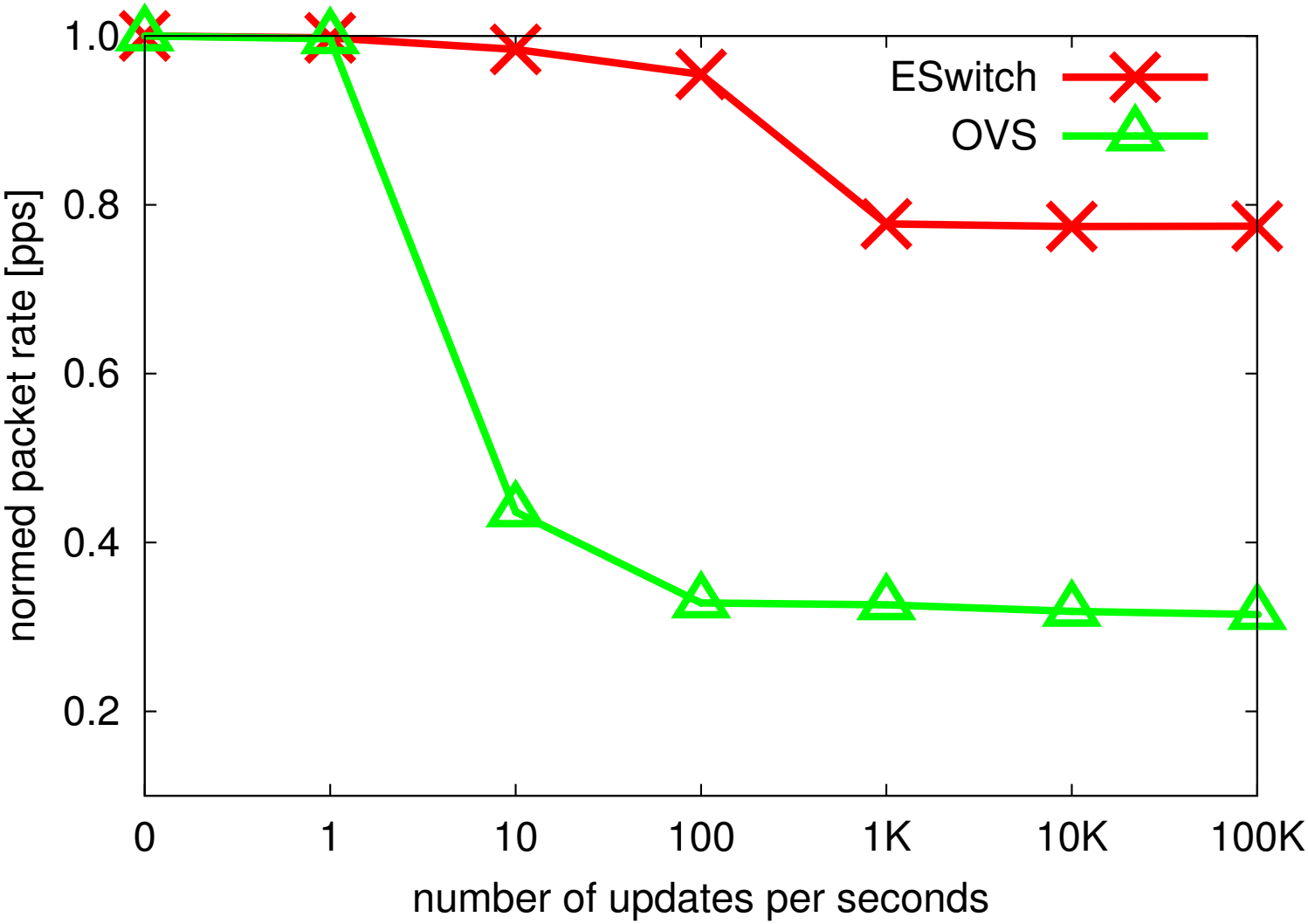
single core, 64-byte packets, Intel Xeon, XL710 @ 40 Gb

Latency



single core, 64-byte packets, Intel Xeon, XL710 @ 40 Gb

Throughput Under Updates



single core, 64-byte packets, random updates to IP routing table

Conclusions

- For a switch to be truly programmable, the dataplane itself must also be adaptable
- ESWITCH is a datapath compiler to turn OpenFlow programs into runnable fast-paths
  - (at least) twice the packet rate of OVS
  - orders of magnitude smaller latency
  - even under heavy update load
- Admits analytic performance models (see paper)
- ESWITCH is now in production at Ericsson!

Hope you've seen the demo! If not, please talk to us, we may find a way to show you ESWITCH in operation

ESWITCH is about to become open-source (as soon as we resolve IPR issues)!

Besides, we are looking for visiting researcher positions...

ESWITCH vs P4

- Both P4 and ESWITCH are datapath compilers, but ESWITCH is restricted to OpenFlow while P4 is generic
- OTOH, P4 is **static** (knows pipeline semantics only), while ESWITCH sees the actual pipeline contents
- The allows ESWITCH to use several **runtime optimization** techniques, similar to JIT compilers:
  - template specialization with full constant inlining
  - direct jump pointers
  - small tables JITted to the direct code template
- Potentially more efficient code with ESWITCH than with equivalent P4 program
- There is no reason why dataplane specialization could not be extended to P4