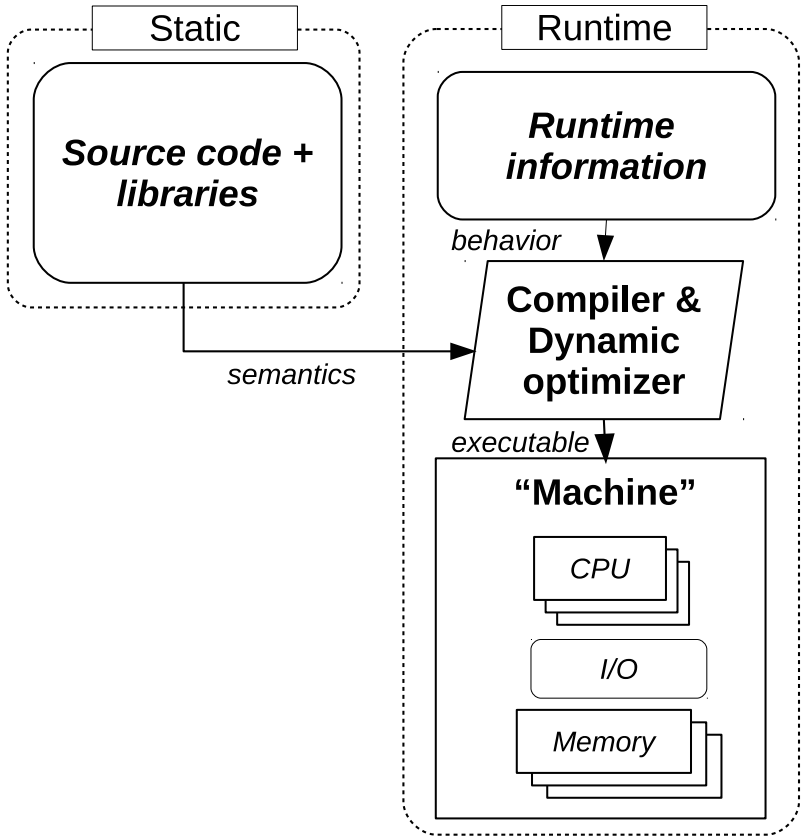
Dynamic Compilation and Optimization of Packet Processing Programs

Gábor Rétvári, László Molnár,
Gábor Enyedi, Gergely Pongrácz

MTA-BME Information Systems Research Group
TrafficLab, Ericsson Research, Hungary

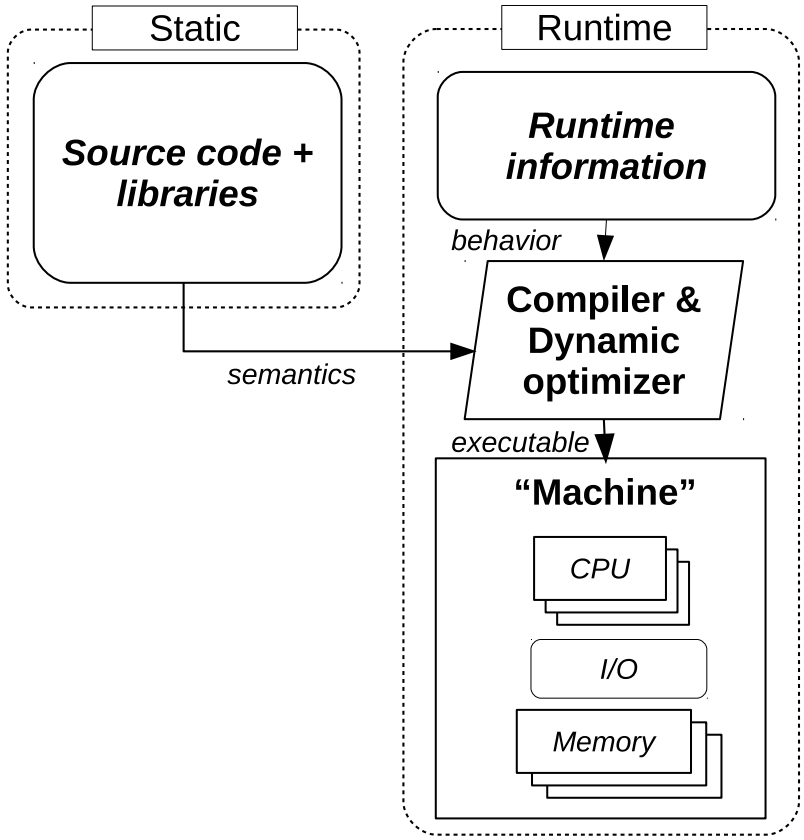
ERICSSON

Preface: Dynamic Optimization



- **Static compilation:** offline transformation of source code into an executable
- **Dynamic compilation:** online program optimization using information only available at run time

Preface: Dynamic Optimization

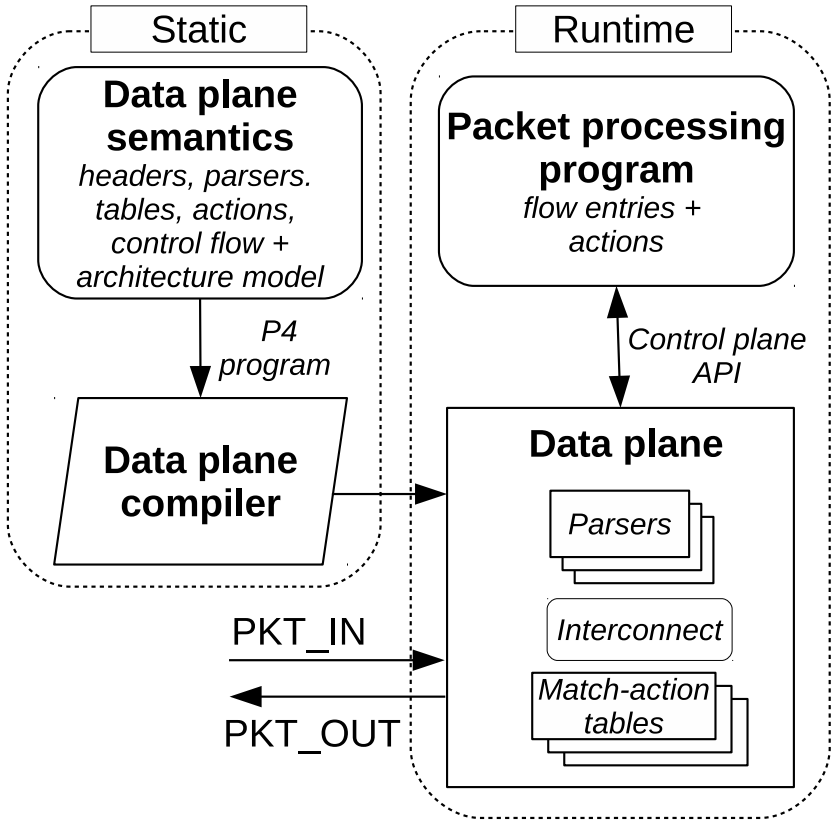


Can we use the same techniques for data-plane compilation?

Agenda

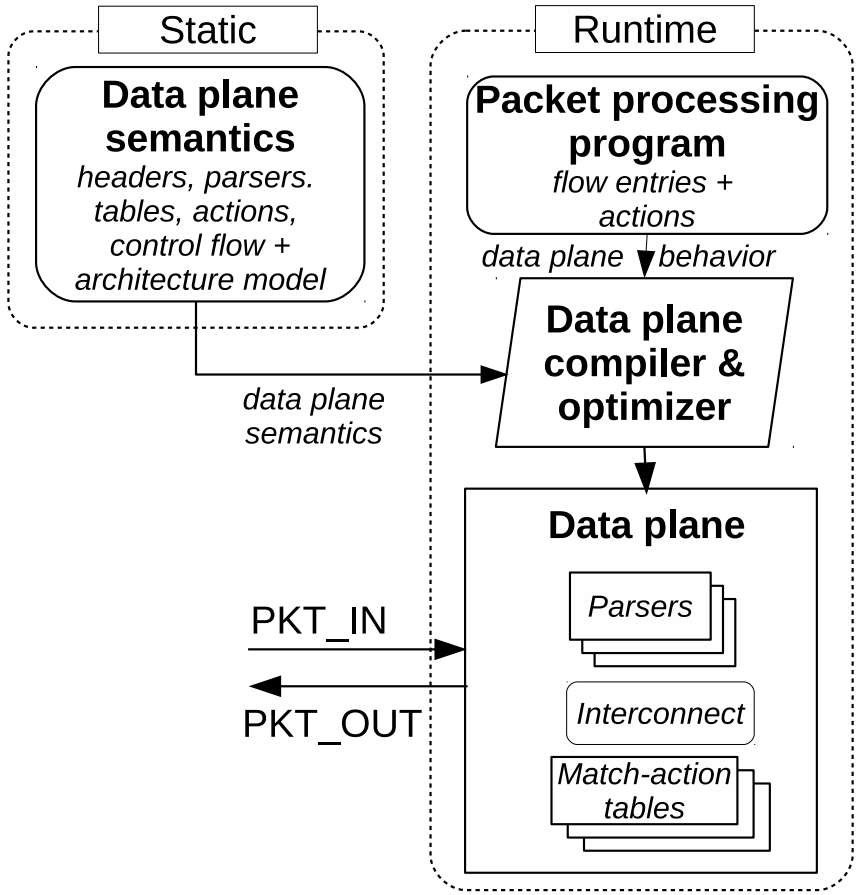
- What we mean by “dynamic data-plane compilation”
- ESWITCH4P4: a dynamically optimizing P4 compiler
- Case studies

Static Data-plane Compilation



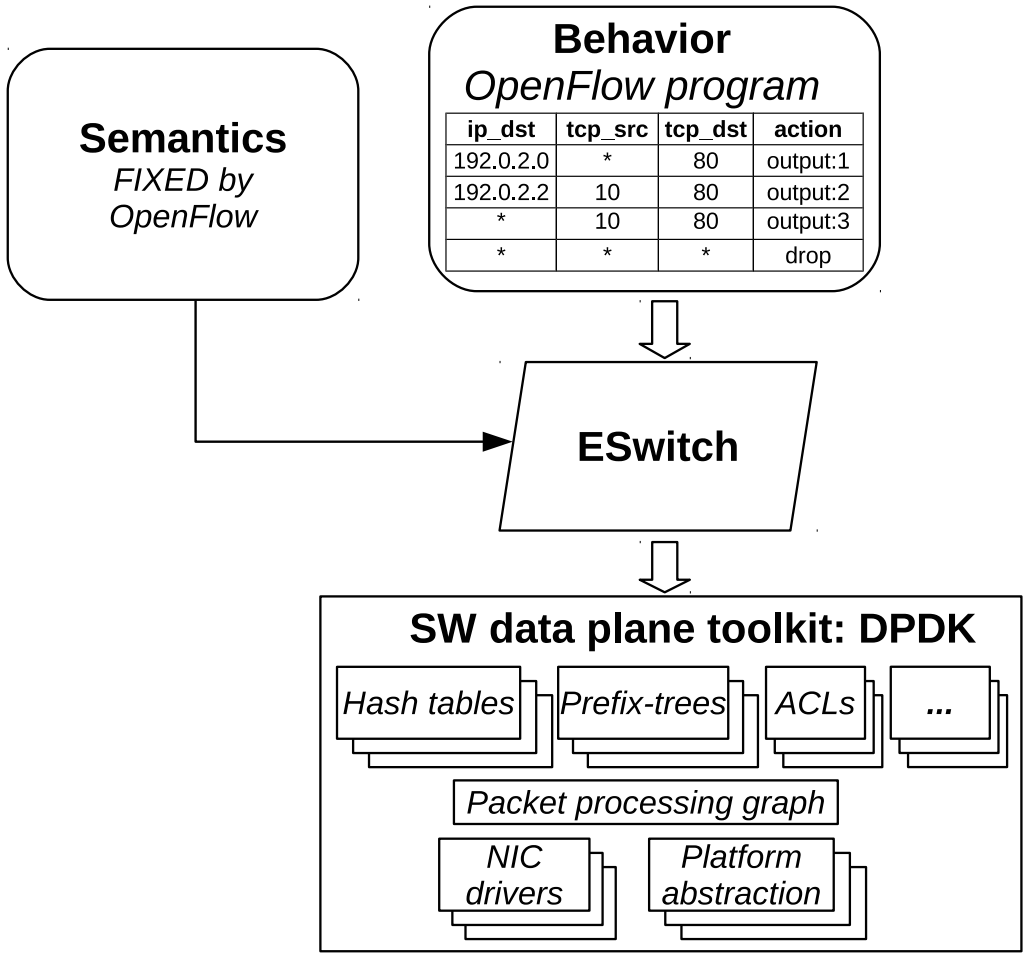
- P4 program describes data-plane **semantics**
- Data-plane **behavior** can be configured online

Dynamic Data-plane Compilation



- **A dynamic compiler has access to the semantics as well as the behavior and optimizes for both**

Example for OpenFlow: ESWITCH



L. Molnár, G. Pongrácz, G. Enyedi, Z. L. Kis, L. Csikor, F. Juhász, A. Kőrösi, and G. Rétvári. *Dataplane specialization for high performance OpenFlow software switching.* In ACM SIGCOMM, 2016.

ESWITCH4P4

- A proof-of-concept dynamic P4 compiler and software switch we have started to experiment with
- **Template-based code generation** for fast data-plane synthesis (runs on every `table_add`/`table_delete`!)
- Currently uses a small (64-bit) per-packet scratchpad and supports only 3 general templates
  - `read`: read field from header to scratchpad (parse)
  - `match`: match scratchpad content at given offset against some key (match)
  - `write`: write scratchpad to header field (deparse)
- Demonstrate some dynamic compilation techniques on hand-crafted P4 use cases

Dead Code Elimination

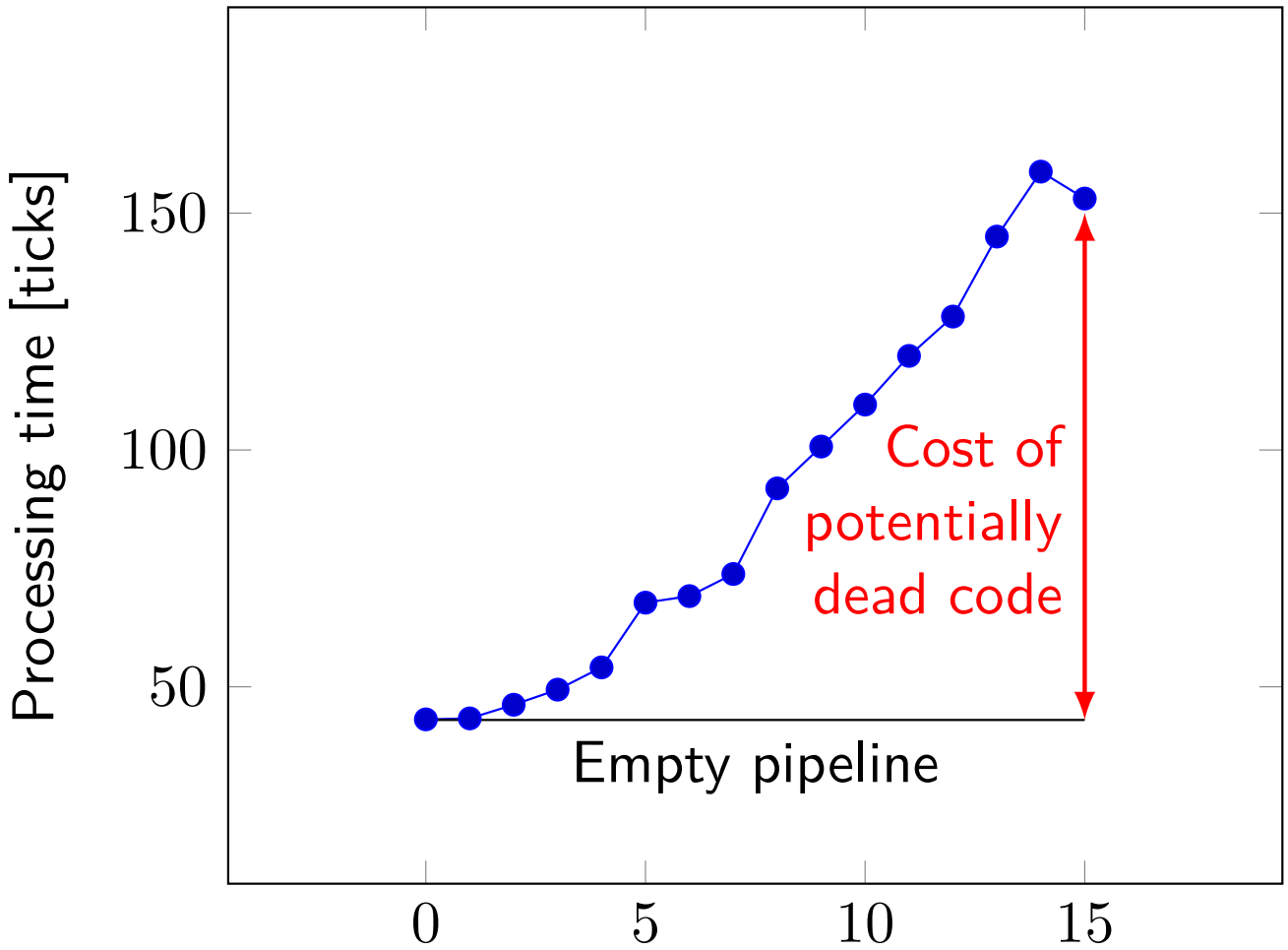
- At any point in time many packet processing features may go unused, like many switches
  - may run with empty ACLs
  - may not terminate VXLAN/GRE/MPLS tunnels
  - may not use all possible rewrite rules
- The corresponding, statically compiled code is “dead”
- Configuration-dependent, revealed only at run-time
- **ESwitch4P4 compiles only the templates that are actually used:** automatic dead code elimination

Dead Code Elimination: Tables

```
table acl {
  key = { ... }
  actions = { ... }
  size = ...;
  default_action = drop;
}
...

apply {
  ...
  acl.apply()
  ...
}
```

Unnecessary when no ACL

Dead Code Elimination: Tables



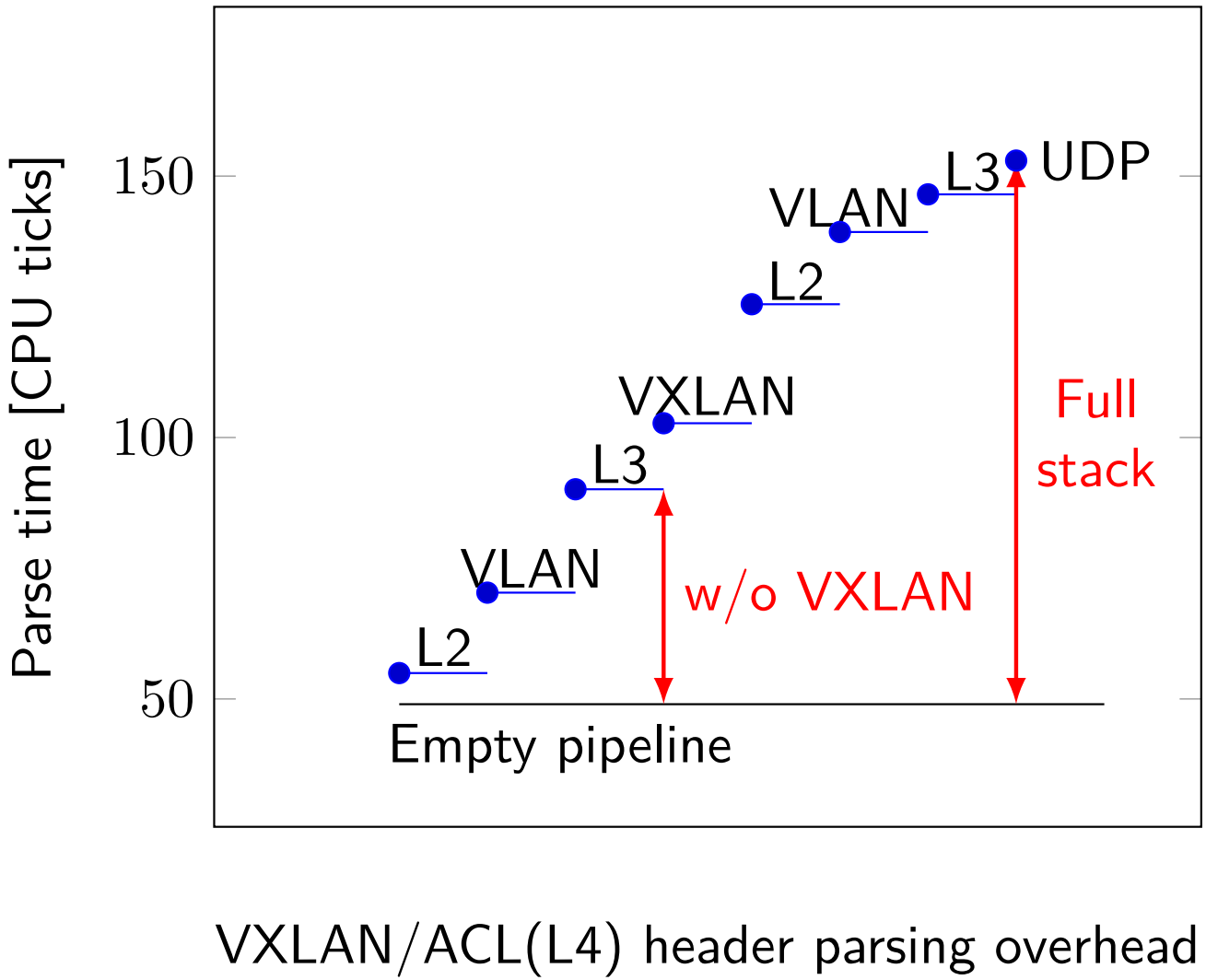
Hand-crafted pipeline as a sequence of JITted empty tables, 10 million packets measured on Intel Core5@2.40GHz CPU/4GB DRAM/Debian/GNU Linux with `pmu-tools/jevents`.

Dead Code Elimination: Parser

```
parser main_parser(packet_in b, out pkt_t p) {
    state start {
        b.extract(p.ethernet);
        transition select(p.ethernet.etherType) {
            ...
            0x800 : parse_ipv4;
            ...
        }
    }

    state parse_ipv4 {
        b.extract(p.ip);
        ...
        transition select(p.ip.protocol) {
            ...
            0x06 : parse_tcp;
            0x11 : parse_udp;
            ...
        }
    }

    state parse_tcp {
        b.extract(p.tcp);
        ...
    }

    state parse_udp {
        b.extract(p.udp);
        ...
    }
}
```

Unnecessary when ACL table empty

Dead Code Elimination: Parser



Hand-crafted header parser with JITted `read/match` templates, 10 million identical packets measured on Intel Core5@2.40GHz CPU/4GB DRAM/Debian/GNU Linux with `pmu-tools/jevents`.

Just-in-time Compilation

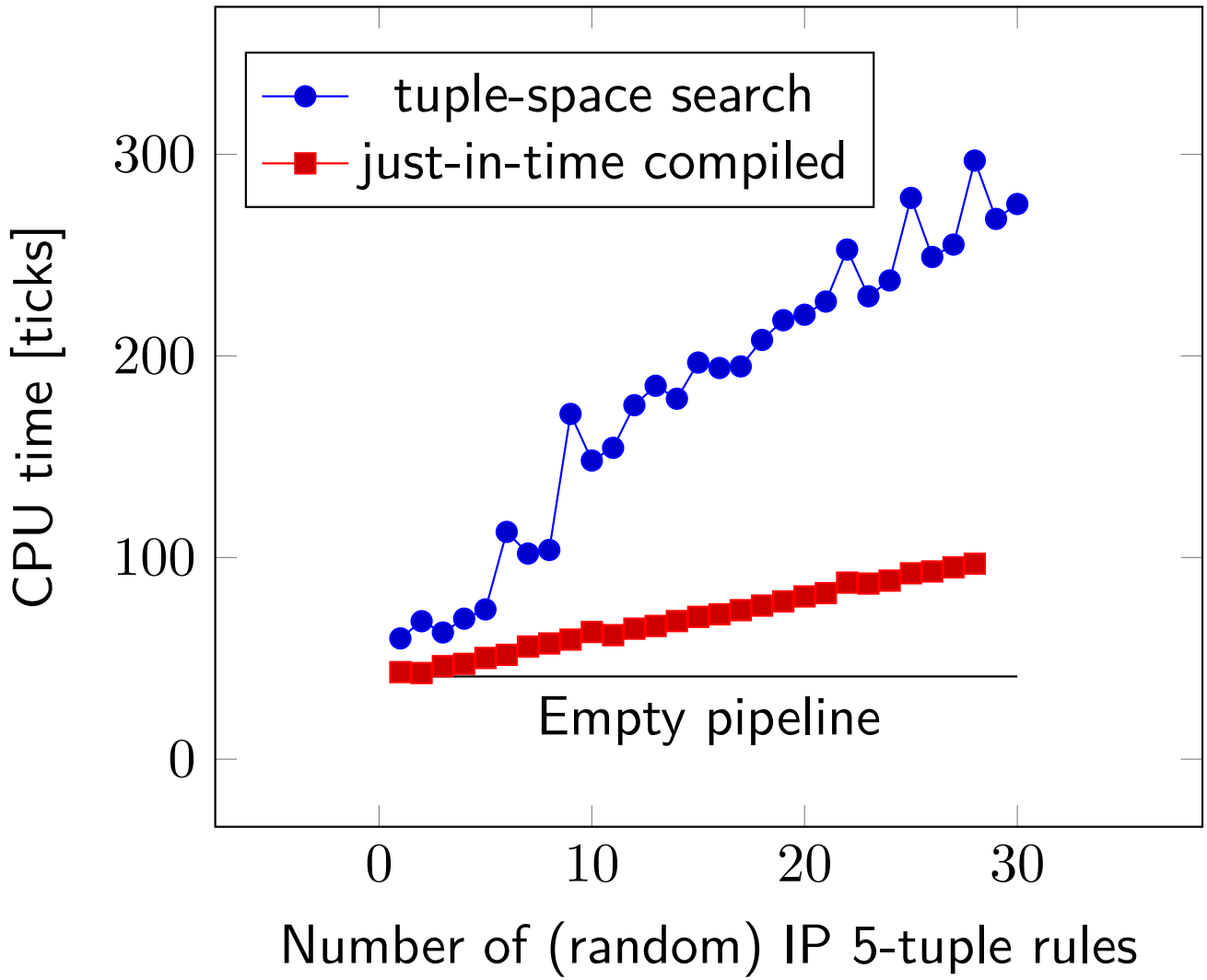
```
table acl {
  key = {
    h.ip.srcAddr          : ternary;
    h.ip.dstAddr          : ternary;
    h.ip.protocol         : ternary;
    h.transport.srcPort   : ternary;
    h.transport.srcPort   : ternary;
    }
  actions = { ... }
  size = 50000;
  default_action = drop;
}
```

ACLs may not match on all fields and match type may not be ternary

Size should not need to be statically provisioned

- ESWITCH4P4 performs **on-the-fly match-action table optimization**
  - optimize packet classifier depending on content
  - remove parsing for unused header fields
  - do not depend on user-defined max size
- **Just-in-time-compile "hot" tables to machine code**

Just-in-time Compilation

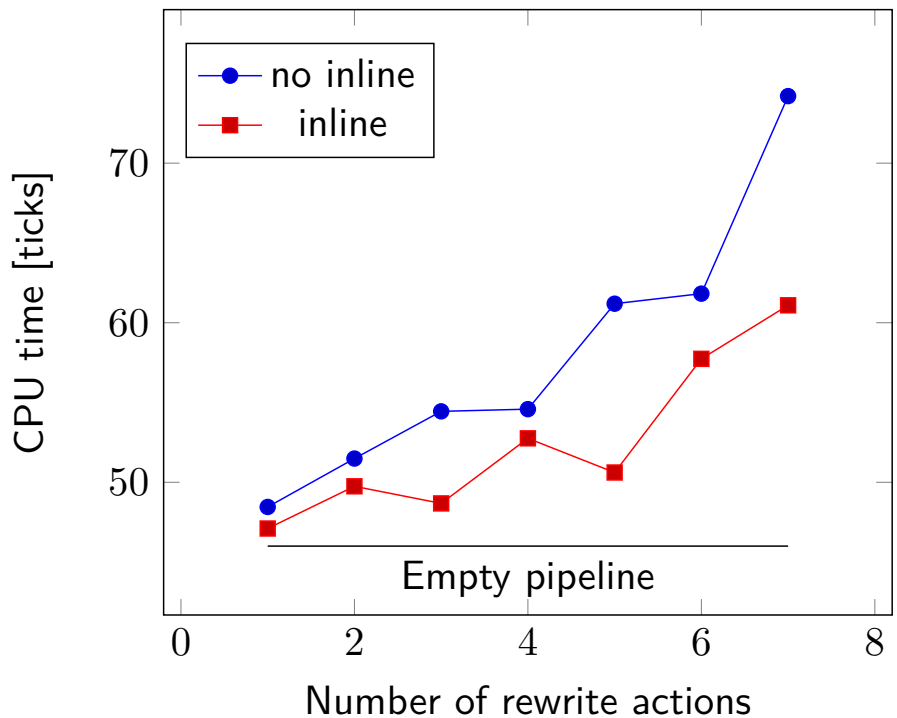


Hand-crafted pipeline with random `match` templates, 10 million identical packets measured on Intel Core5@2.40GHz CPU/4GB DRAM/Debian/GNU Linux with `pmu-tools/jevents`.

Constant Inlining

```
table ipv4_lpm {
    reads   { ipv4.dstAddr : lpm; }
    actions { set_nhop; drop; }
}
action set_nhop(nhop_ipv4, port) { ... }
table_add ipv4_lpm set_nhop 10.0.0.1/32 => 10.0.0.1 1
table_add ipv4_lpm set_nhop 10.0.0.2/32 => 10.0.0.2 2
table_add ipv4_lpm set_nhop 10.0.0.3/32 => 10.0.0.3 3
...
```

} Subject to inlining



Hand-crafted pipeline with 15 JITted rewrite actions and `write` templates, 10 million identical packets measured on Intel Core5@2.40GHz CPU/4GB DRAM/Debian/GNU Linux with `pmu-tools/jevents`.

Conclusions

- Complete switch configuration becomes available only at runtime: why compiling datapaths statically?
- Well-known **runtime optimization techniques can be used to improve switch performance substantially**
- Comes at a price: **additional complexity and latency on updates**
- Of course there remain questions...
- Is dynamic compilation worth it, after all? For SW targets definitely, but for HW???
- Which precisely are the right templates for P4?