Dynamic Compilation and Optimization of Software Network Functions

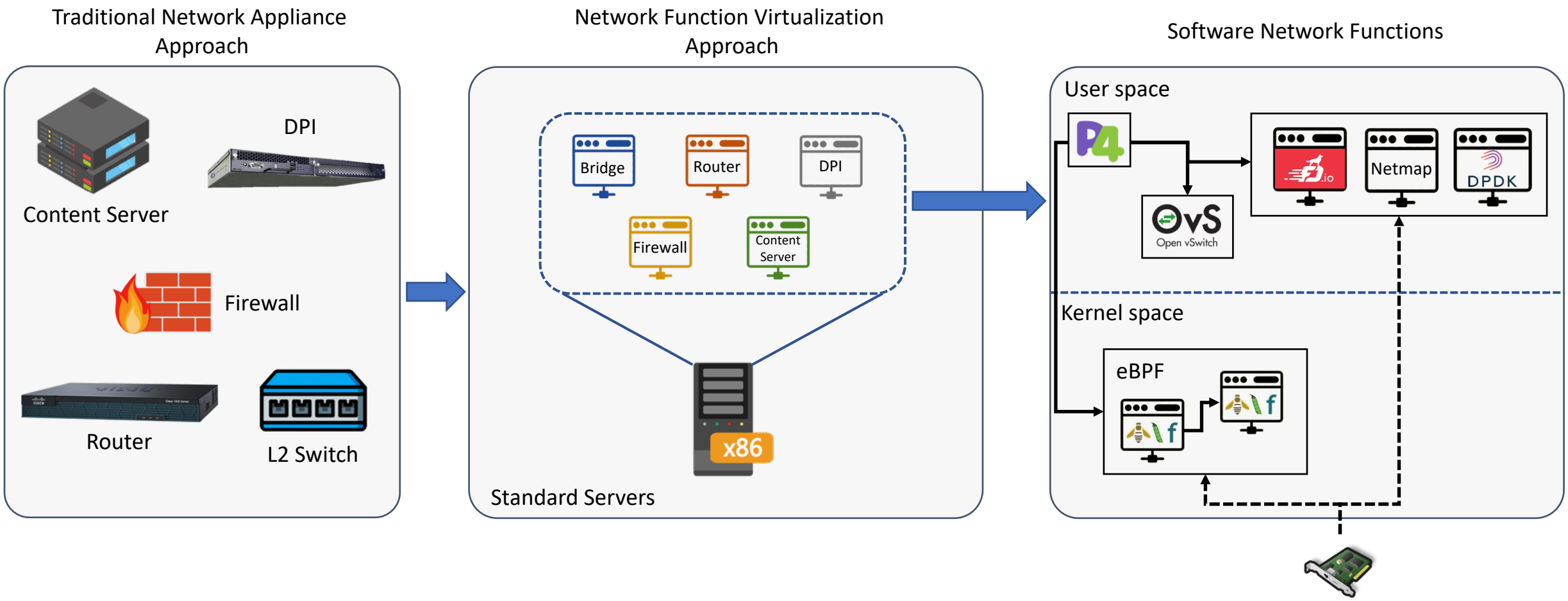
Sebastiano Miano, Gianni Antichi, Gábor Rétvári, Andrew W. Moore

COSENERS 2019

POLITECNICO
DI TORINO

UNIVERSITY OF
CAMBRIDGE

What is a Software Network Function?



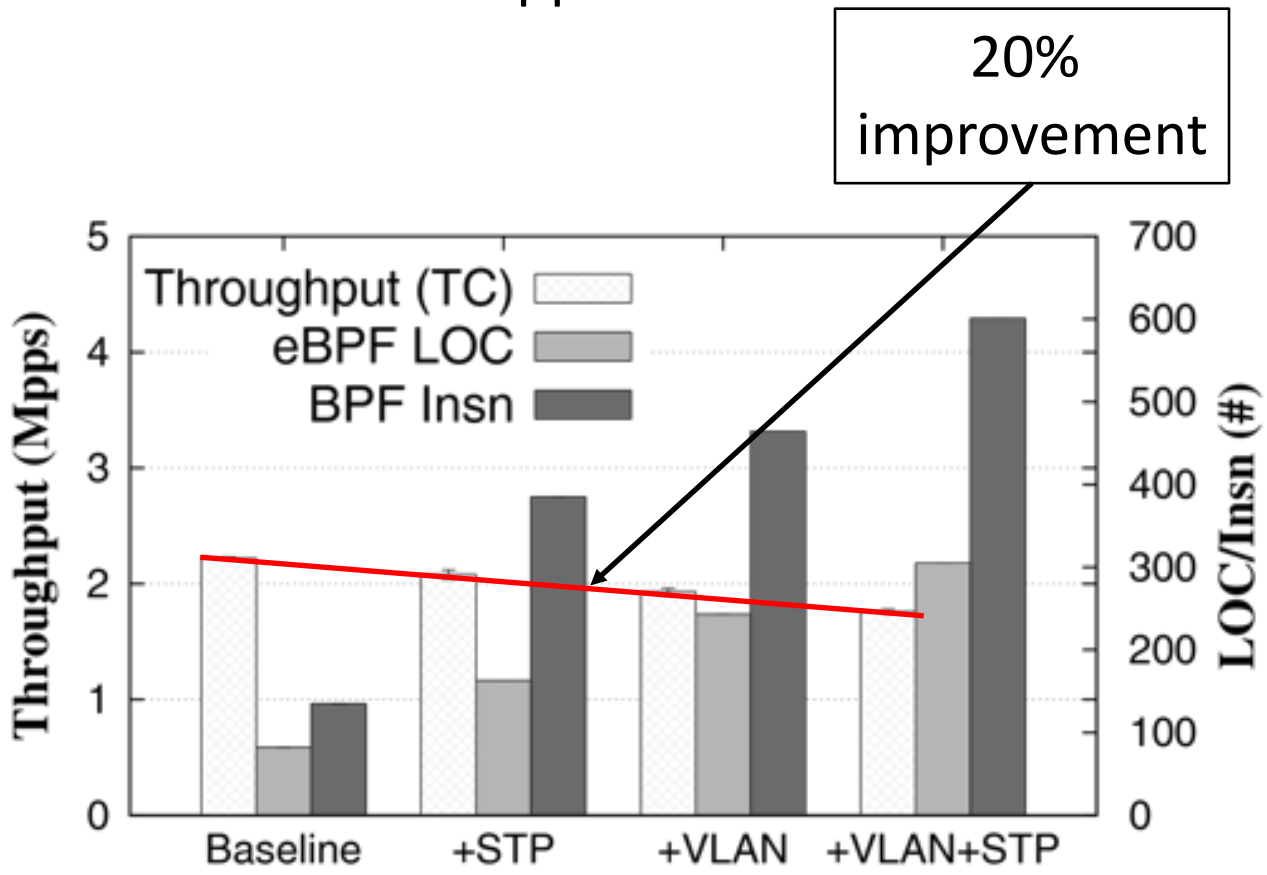
Software Network Functions – Pros and Cons

- Increase flexibility 
- Reduced capital and operating expenses 
- Programming errors 
- Slow (or unexpected) performance 

How can we make software NFs great?!

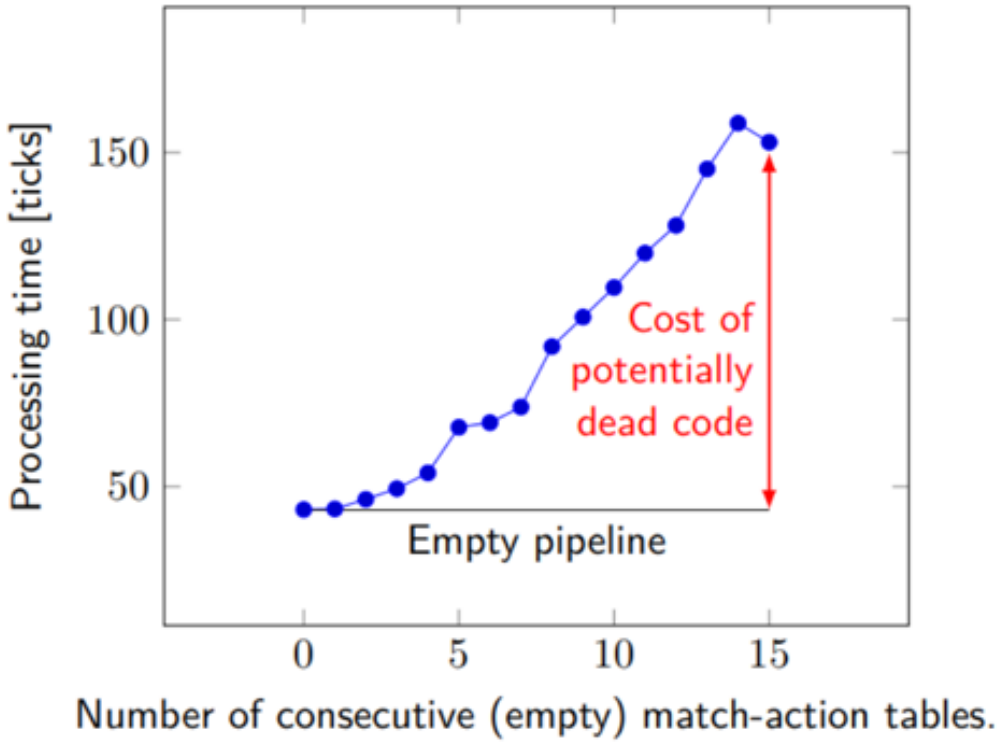
Motivating Informed-optimizations

- At any point, many NF features may go unused. Many switches
  - may not need VLAN or STP support



Cost of potentially dead code

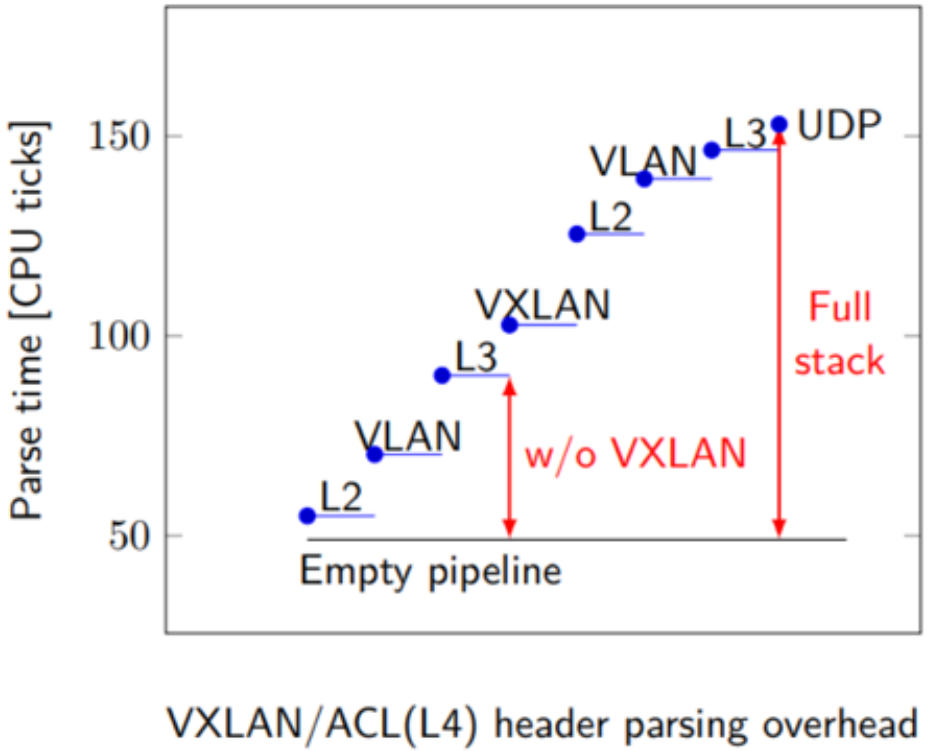
- At any point, many NF features may go unused. Many switches
  - may run with empty ACLs



Rétvári, G., Molnár, L., Enyedi, G., & Pongrácz, G. (2017). Dynamic Compilation and Optimization of Packet Processing Programs. *ACM SIGCOMM NetPL*.

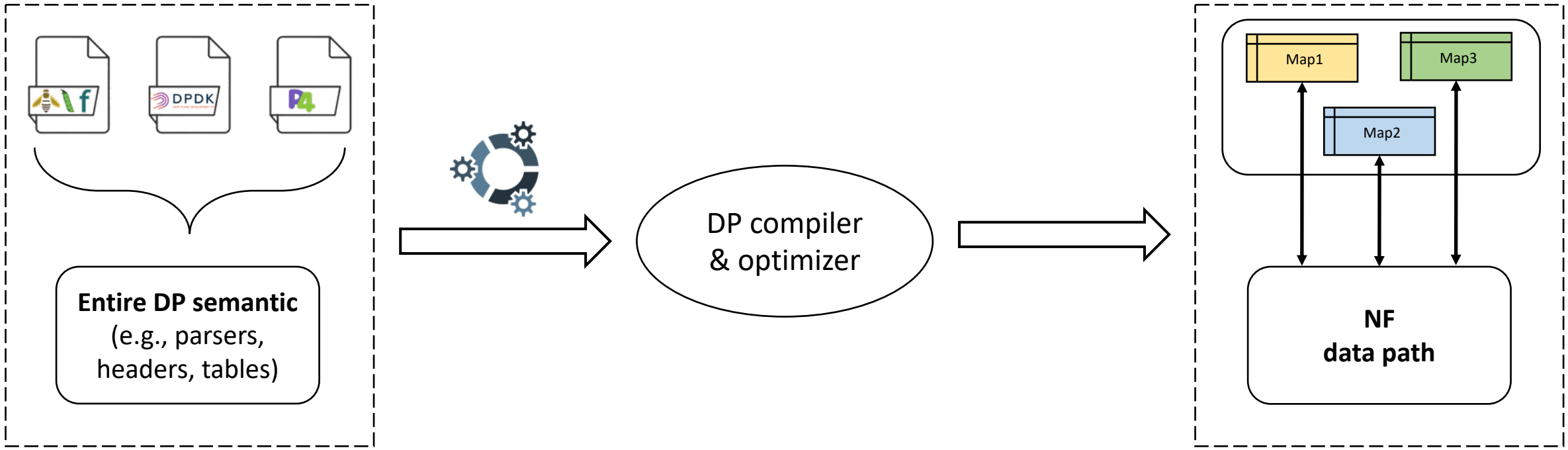
Cost of potentially unused fields

- At any point, many NF features may go unused. Many switches
  - may not use all parsed fields

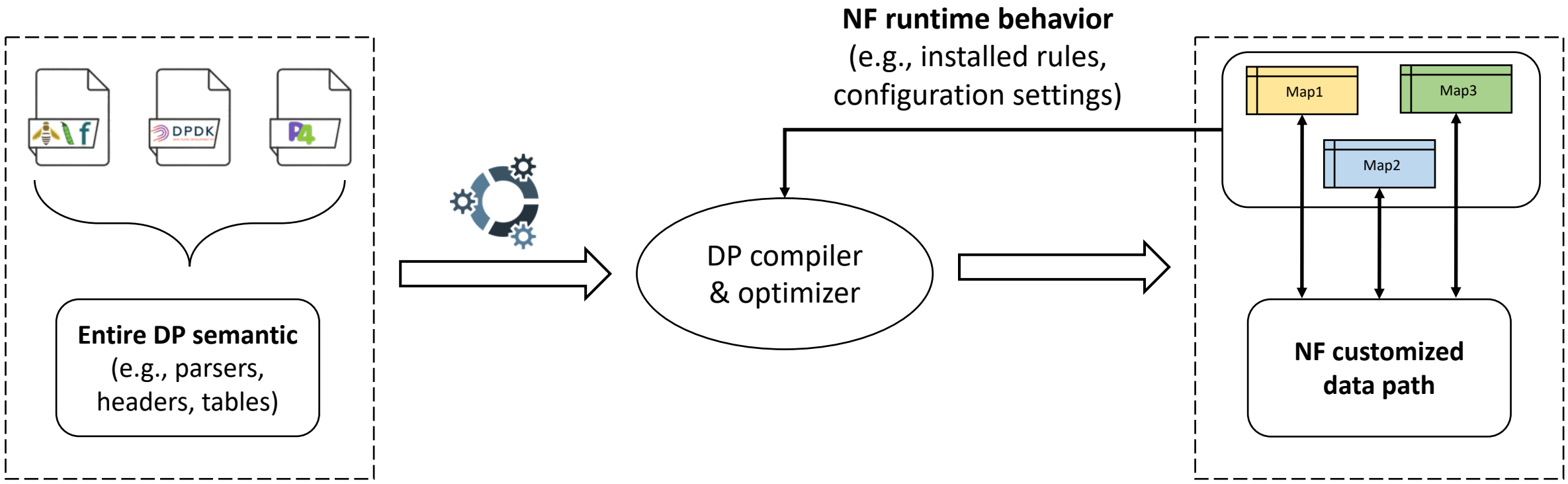


Rétvári, G., Molnár, L., Enyedi, G., & Pongrácz, G. (2017). Dynamic Compilation and Optimization of Packet Processing Programs. *ACM SIGCOMM NetPL*.

Step back: How software NFs get compiled?



The case for dynamic NFs compilation

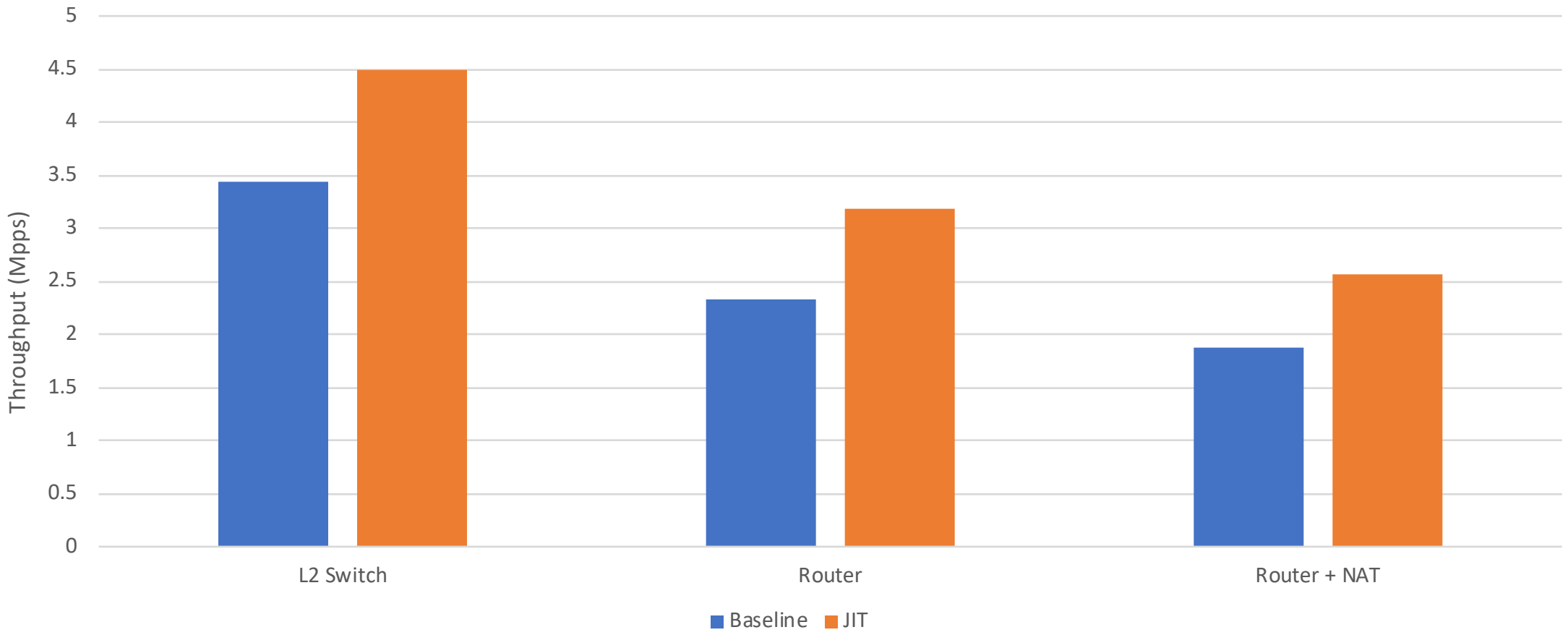


JIT Compiler for software Network Functions

- Runtime compiler responsible for finding the realization of the NF pipeline that achieves near-optimal performance based on:

1. **Dynamic configuration**
   - Optimization based on changes related to data structures
2. **Runtime traffic**
   - Optimization based on runtime traffic patterns and workloads
3. **NF static analysis**
   - Complete understanding of different paths in the code and their performance

Dynamic configuration: JIT table entries



Dynamic configuration: Dead-code elimination

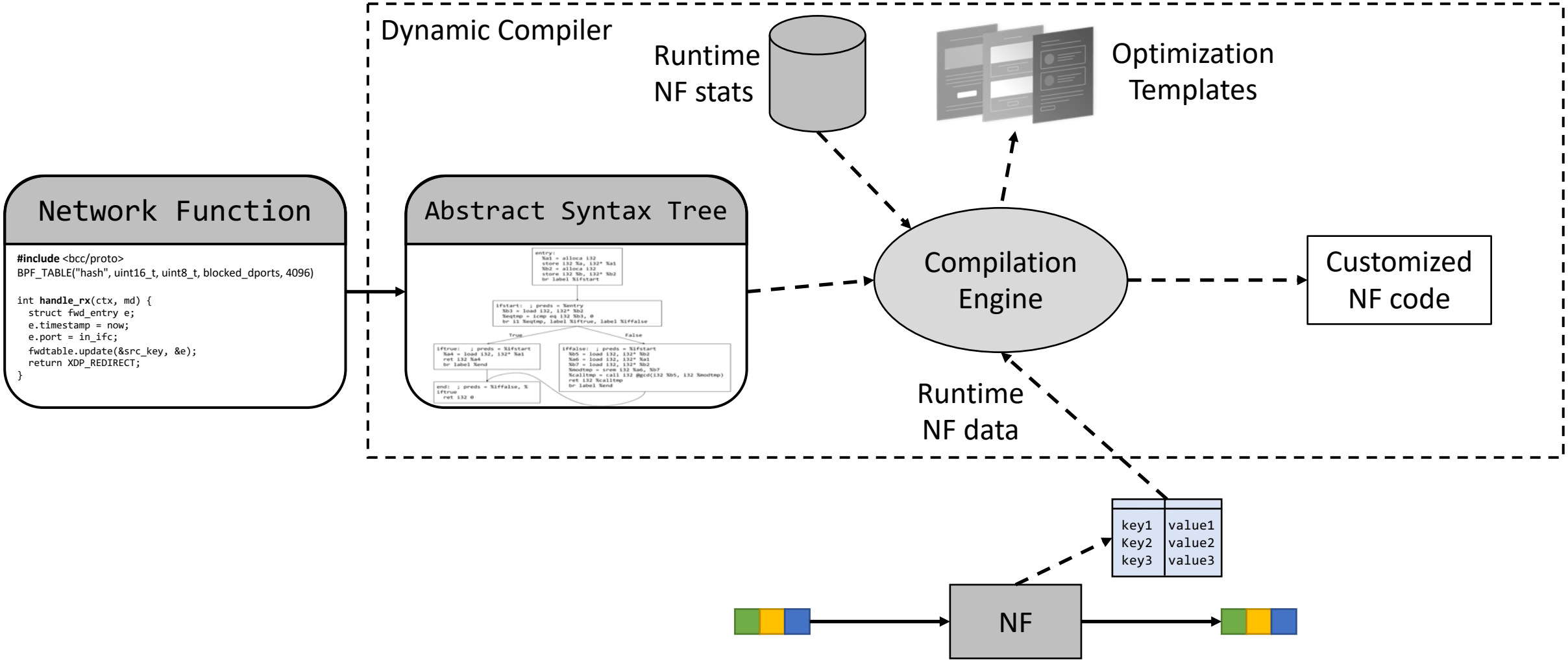
```
parser main_parser(pkt) {
  pkt.extract(eth);
  pkt.extract(ip);
  pkt.extract(l4proto);

  if(ip.dst == X.X.X.X) {
    ret acl.lookup(l4proto);
  }
}
```

```
parser main_parser(pkt) {
  pkt.extract(eth);
  pkt.extract(ip);
  ~~pkt.extract(l4proto);~~

  if(ip.dst == X.X.X.X) {
    ~~ret acl.lookup(l4proto);~~
    ret DROP;
  }
}
```

Overall Architecture



Conclusions

- Dynamic compiler for software NFs that optimize the user code based on the workload and the runtime configuration
- In a first stage we decided to limit the target to eBPF. Why?
  - In-kernel eBPF interpreter that execute eBPF code (similar to JVM)
  - Simpler code
- However, most of the optimization can be (easily) applied to every type of software NF (e.g., P4, DPDK)
- First results look promising
  - JIT tables + dead-code elimination ~ 40-60% improvement