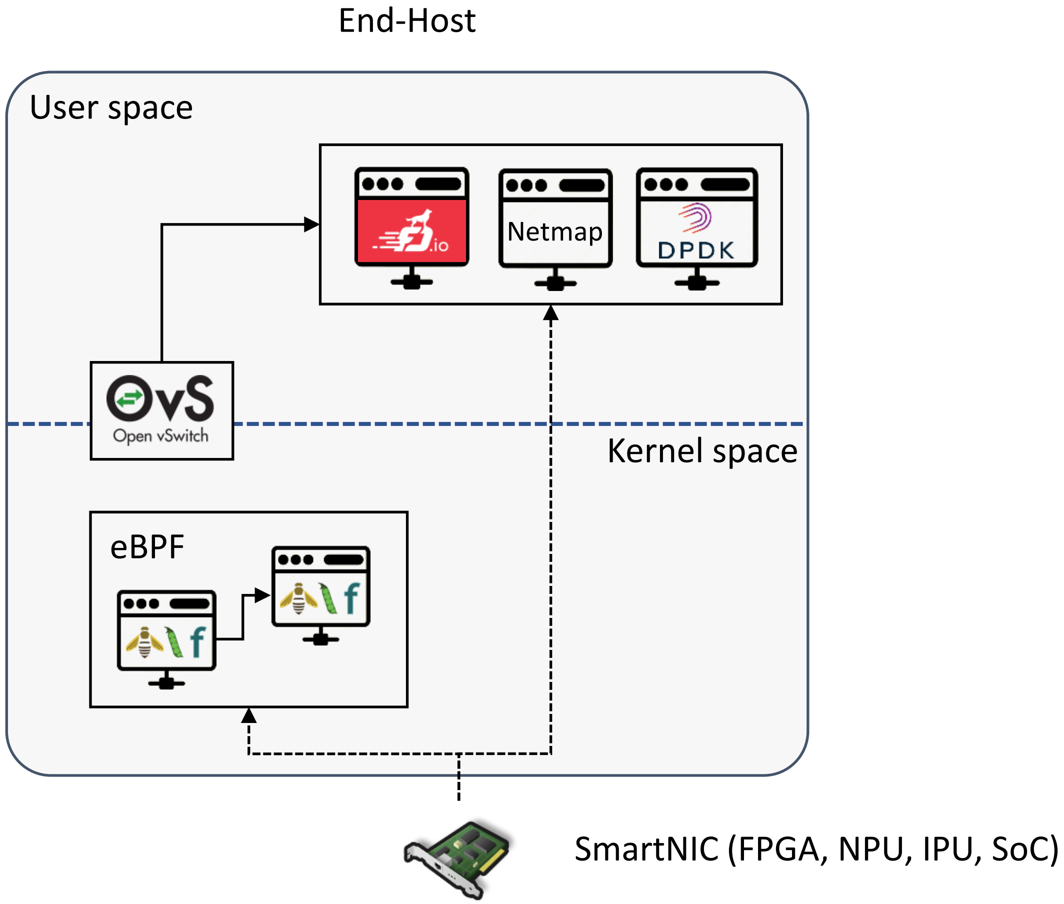
Compiler-Driven End-Host Network Stacks

Sebastiano Miano, Farbod Shahinfar, Alireza Sanaee, Gianni Antichi

COSENERS 2022

Queen Mary
University of London

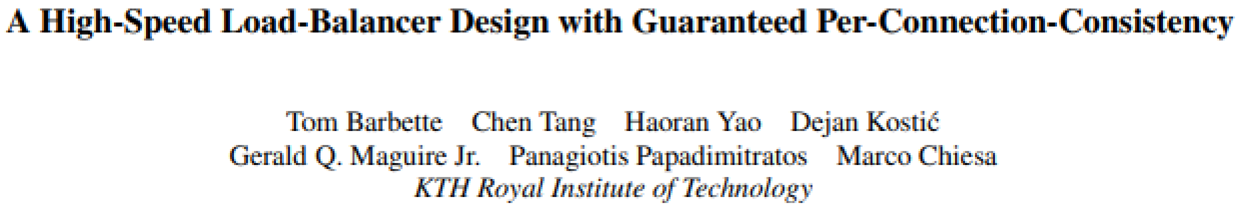
State of the Art in End-Host Network Programming



No One-Size-Fits-All Solution

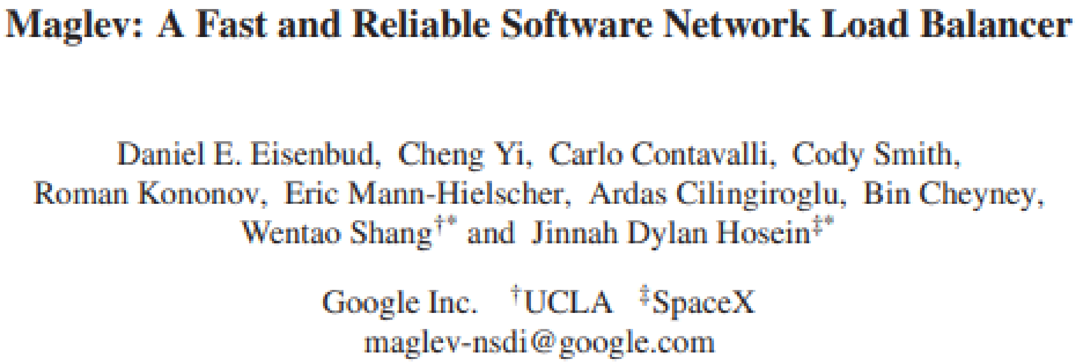
1. The choice of **where** to place a given functionality is not ***only*** restricted to the capabilities of a given layer
   - It may depend also on the traffic that the application is processing

- **E.g., NIC** or **Userspace** for traffic that is redirected from one host to the other



A High-Speed Load-Balancer Design with Guaranteed Per-Connection-Consistency

Tom Barbette Chen Tang Haoran Yao Dejan Kostić
Gerald Q. Maguire Jr. Panagiotis Papadimitratos Marco Chiesa
KTH Royal Institute of Technology



Maglev: A Fast and Reliable Software Network Load Balancer

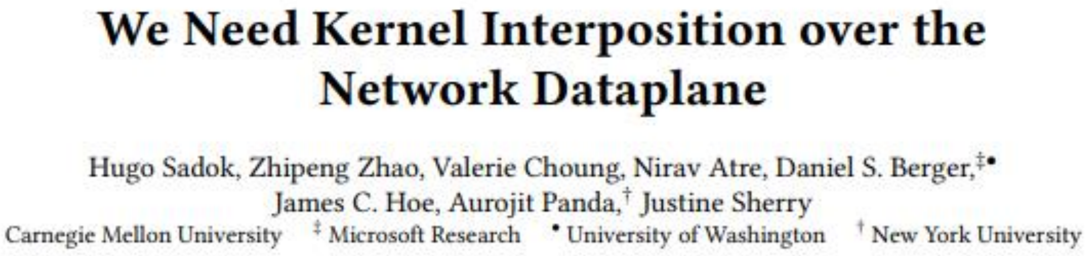
Daniel E. Eisenbud, Cheng Yi, Carlo Contavalli, Cody Smith,
Roman Kononov, Eric Mann-Hielscher, Ardas Cilingiroglu, Bin Cheyney,
Wentao Shang†* and Jinnah Dylan Hosein‡*

Google Inc. †UCLA ‡SpaceX
maglev-nsdi@google.com

No One-Size-Fits-All Solution

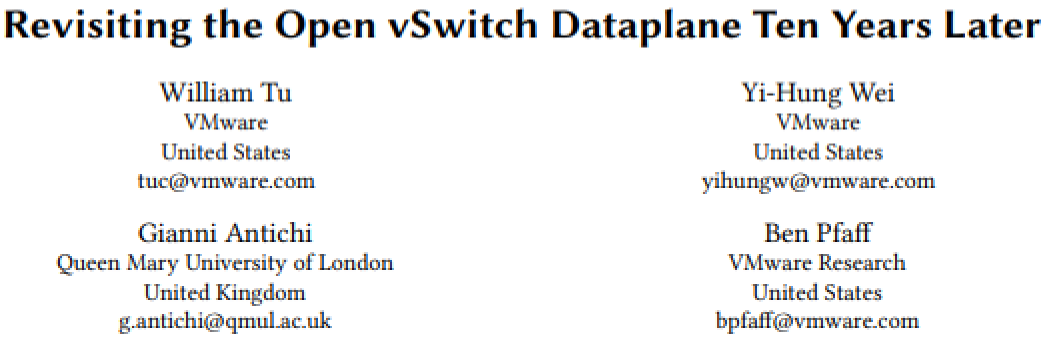
1. The choice of **where** to place a given functionality is not ***only*** restricted to the capabilities of a given layer
   - It may depend also on the traffic that the application is processing

- **E.g., Kernel** for Container-to-Container traffic



We Need Kernel Interposition over the Network Dataplane

Hugo Sadok, Zhipeng Zhao, Valerie Choung, Nirav Atre, Daniel S. Berger,[‡•]
James C. Hoe, Aurojit Panda,[†] Justine Sherry
Carnegie Mellon University [‡] Microsoft Research [•] University of Washington [†] New York University



Revisiting the Open vSwitch Dataplane Ten Years Later

William Tu
VMware
United States
tuc@vmware.com

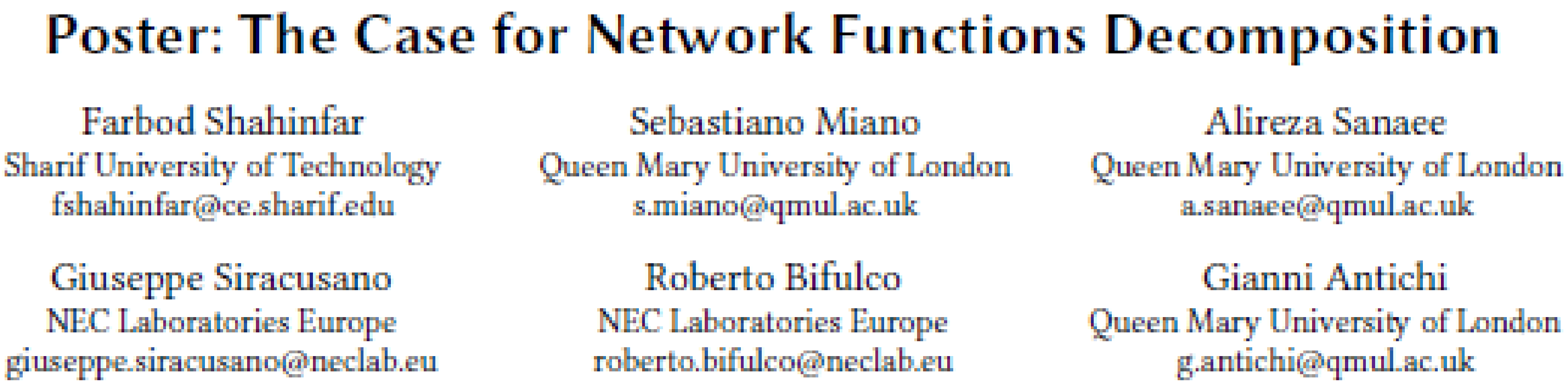
Yi-Hung Wei
VMware
United States
yihungw@vmware.com

Gianni Antichi
Queen Mary University of London
United Kingdom
g.antichi@qmul.ac.uk

Ben Pfaff
VMware Research
United States
bpfaff@vmware.com

It gets even worse

2. It’s not only matter of deciding **where** to place a given program, but also **how** to place it (or part of it)
   - E.g., by splitting a program logic between kernel/userspace we can get better performance [1]



Poster: The Case for Network Functions Decomposition

Farbod Shahinfar
Sharif University of Technology
fshahinfar@ce.sharif.edu

Sebastiano Miano
Queen Mary University of London
s.miano@qmul.ac.uk

Alireza Sanaee
Queen Mary University of London
a.sanaee@qmul.ac.uk

Giuseppe Siracusano
NEC Laboratories Europe
giuseppe.siracusano@neclab.eu

Roberto Bifulco
NEC Laboratories Europe
roberto.bifulco@neclab.eu

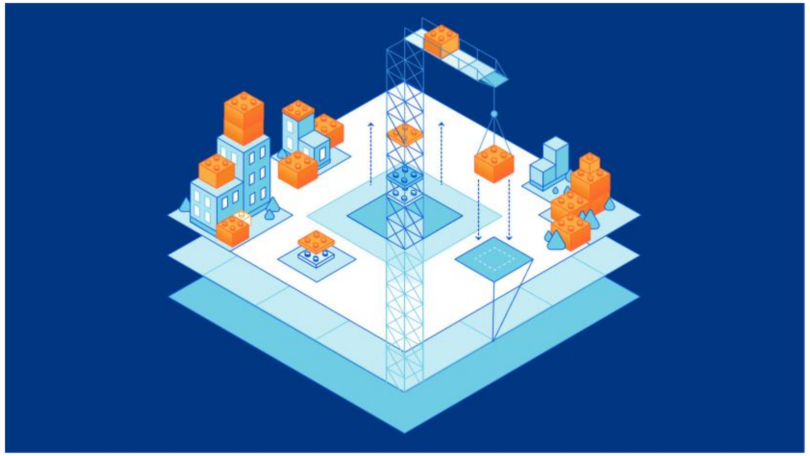
Gianni Antichi
Queen Mary University of London
g.antichi@qmul.ac.uk

[1] Shahinfar, F., Miano, S., Sanaee, A., Siracusano, G., Bifulco, R., & Antichi, G. (2021, December). The case for network functions decomposition. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies* (pp. 475-476).

Our idea: Compiler-Driven End-Host Network Stack

- We should start thinking to the end-host network stack as a programmable platform.
  - Behavior described at top, partitioned, compiled and run across elements

- This can allow us to introduce software engineering techniques to be used in all the layers of abstractions that we use to program the network
  - Semantic verification
  - Dynamic optimization
  - Performance prediction



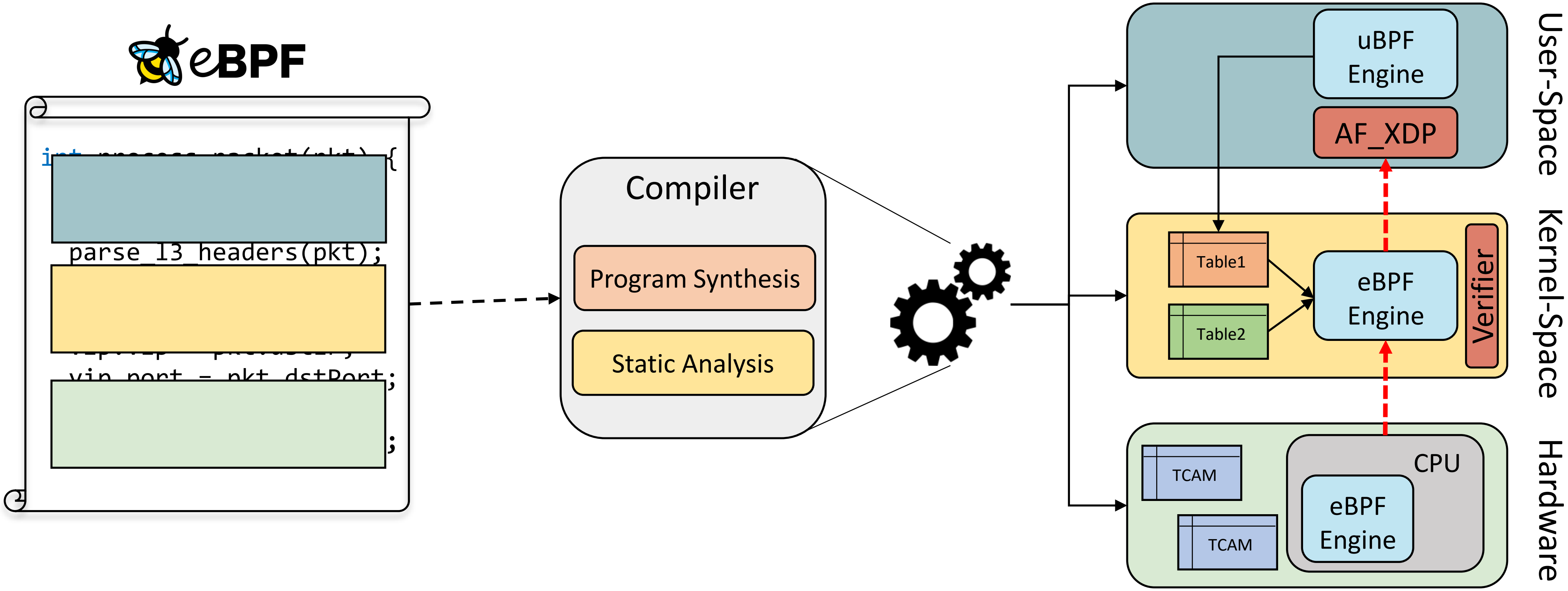
How to do it?

Our proposal

Use **eBPF** as main language to **program** the entire **end-host** networking stack

Frankenstack

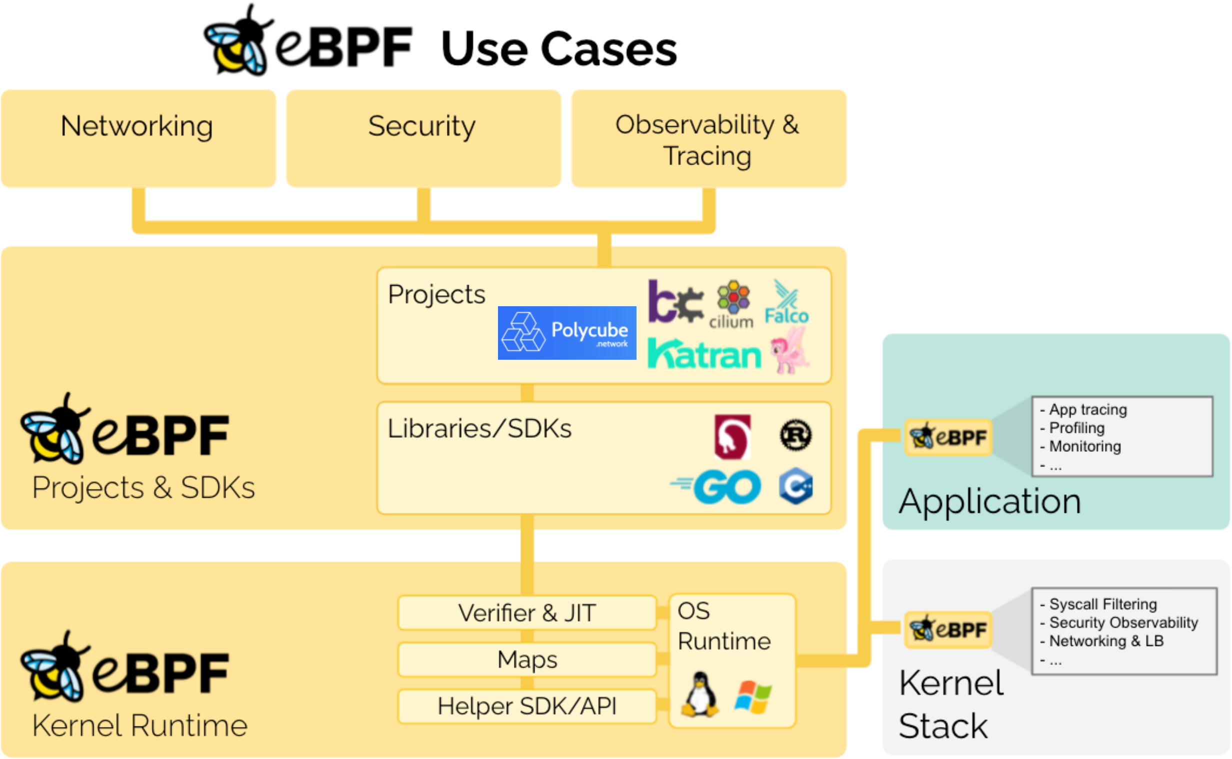
- **Ambitious goal:** Self adapting network stacks
  - The compiler decides how to **split/combine** data plane programs, where to **place**, and how to **optimize** them **at runtime**



```
int process_packet(pkt) {
  parse_l3_headers(pkt);
  vip_port = pkt.dstPort;
  ;
```

eBPF as the “perfect” DSL for host-based NFs

1. eBPF is the “de-facto” language to program the Linux kernel
   - This is not restricted only to networking functionality but also tracing, observability, security, and many others…



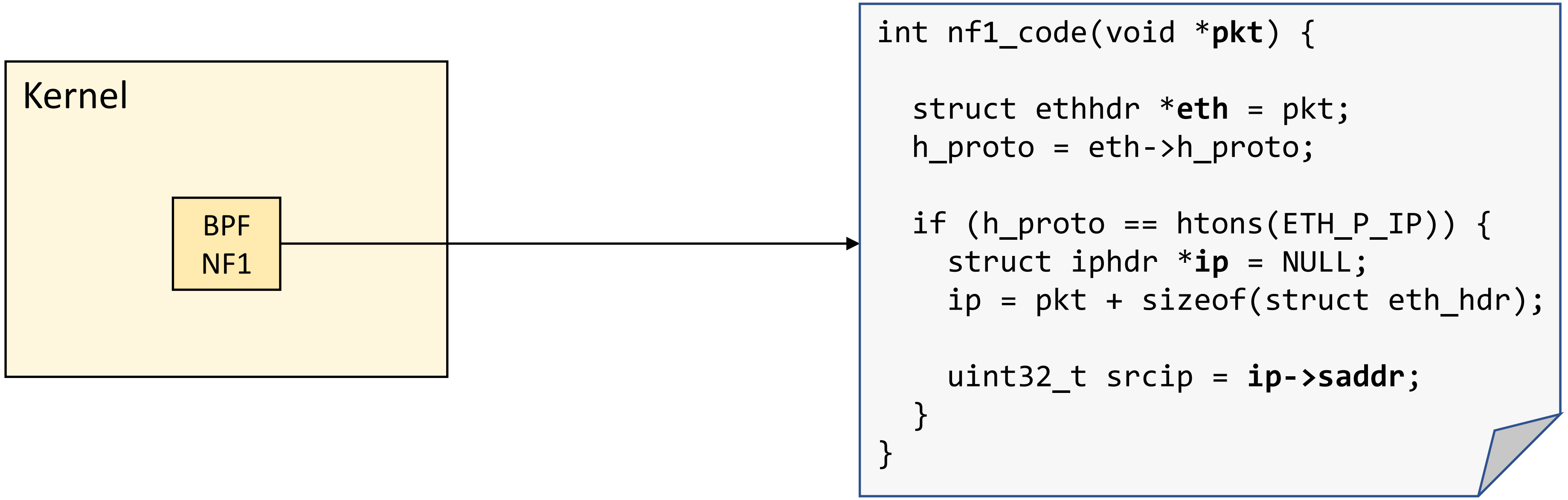
eBPF as the “perfect” DSL for host-based NFs

2. eBPF language is Turing complete

- Not the kernel code, since it is constrained by the verifier
  - Only bounded loops
  - Limited complexity (for verifiability)
  - Restricted functions & libraries

eBPF as the “perfect” DSL for host-based NFs

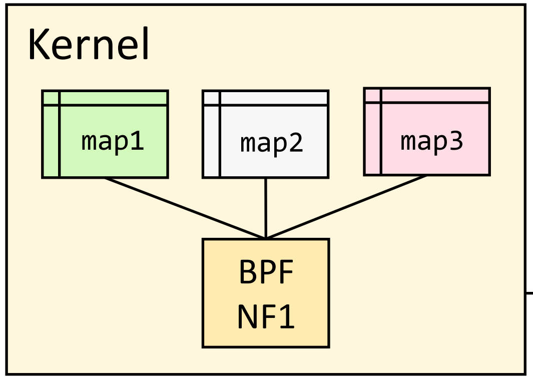
3. Packet as first-class citizen
    - This makes it easy to analyze the type of operations performed on the packet
        - E.g., Which packet fields are read/written



```
int nf1_code(void *pkt) {

  struct ethhdr *eth = pkt;
  h_proto = eth->h_proto;

  if (h_proto == htons(ETH_P_IP)) {
    struct iphdr *ip = NULL;
    ip = pkt + sizeof(struct eth_hdr);

    uint32_t srcip = ip->saddr;
  }
}
```

eBPF as the “perfect” DSL for host-based NFs

4. Clear definition of data structures (and their algorithm)
5. Explicit separation between stateless and stateful operations

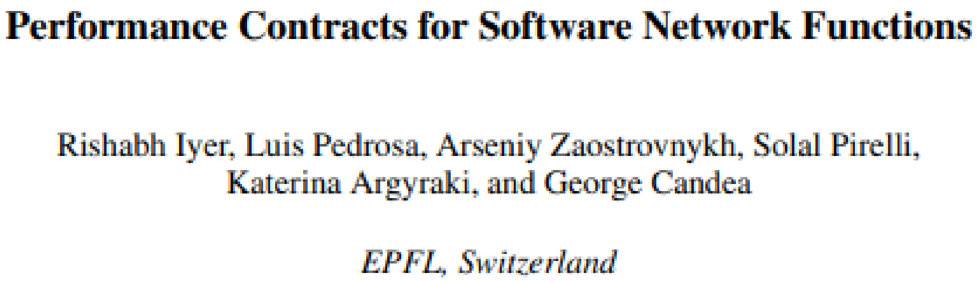


KEY VALUE

```
BPF_LPM(map1, uint32_t, uint64_t)
BPF_HASH(map2, uint16_t, uint64_t);
BPF_ARRAY(map3, uint16_t, uint64_t);

int nf1_code(void *pkt) {
  ...
  uint32_t srcip = ip->saddr;
  uint64_t *value;
  value = bpf_map_lookup(map1, srcip);
  ...
  bpf_map_update(map2, &h_proto, 1);
}
```

eBPF as the “perfect” DSL for host-based NFs

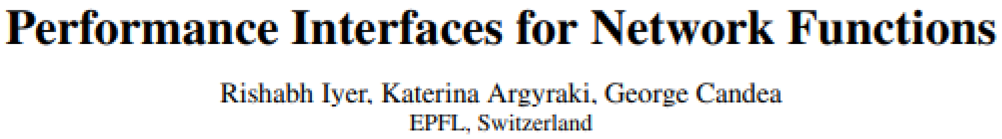
4. Clear definition of data structures (and their algorithm)
5. Explicit separation between stateless and stateful operations
   - Better **performance prediction** and **semantic verification**



Performance Contracts for Software Network Functions

Rishabh Iyer, Luis Pedrosa, Arseniy Zaostrovnykh, Solal Pirelli,
Katerina Argyraki, and George Candea

EPFL, Switzerland

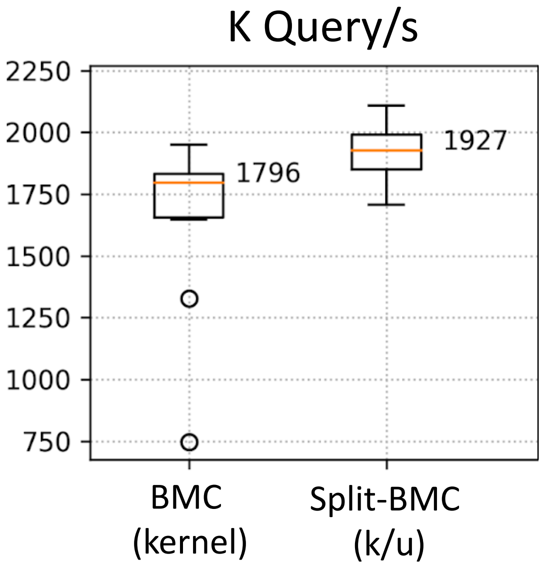


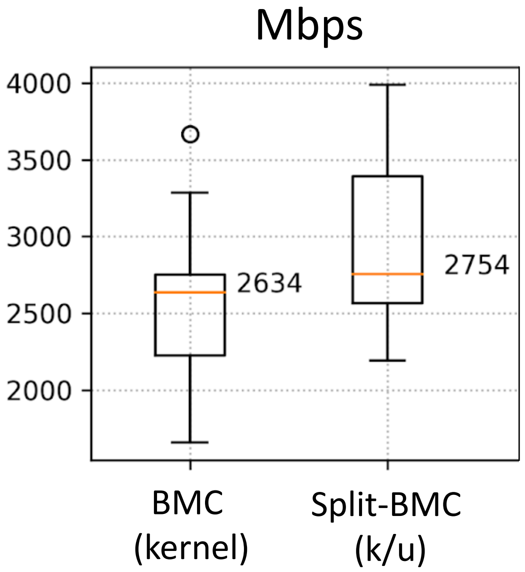
Performance Interfaces for Network Functions

Rishabh Iyer, Katerina Argyraki, George Candea
EPFL, Switzerland

First Step: Automatic splitting between k/u

- Automatic decomposition of eBPF programs between kernel and userspace [1] to achieve:
  - **Expressiveness**
  - **Performance**

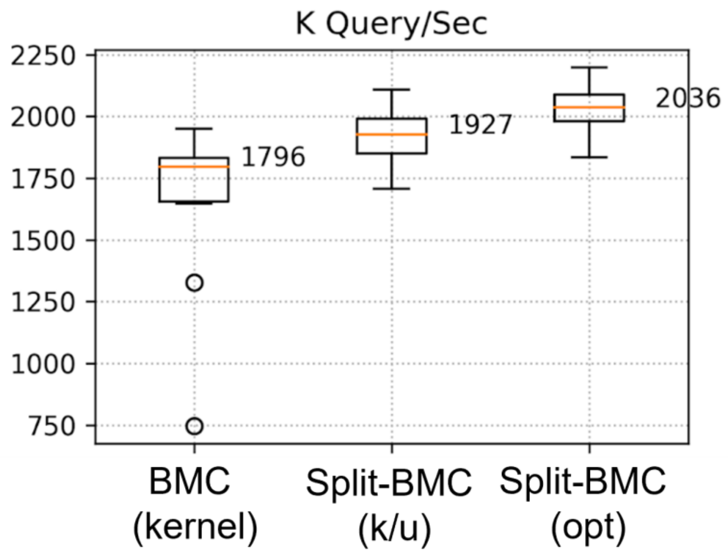


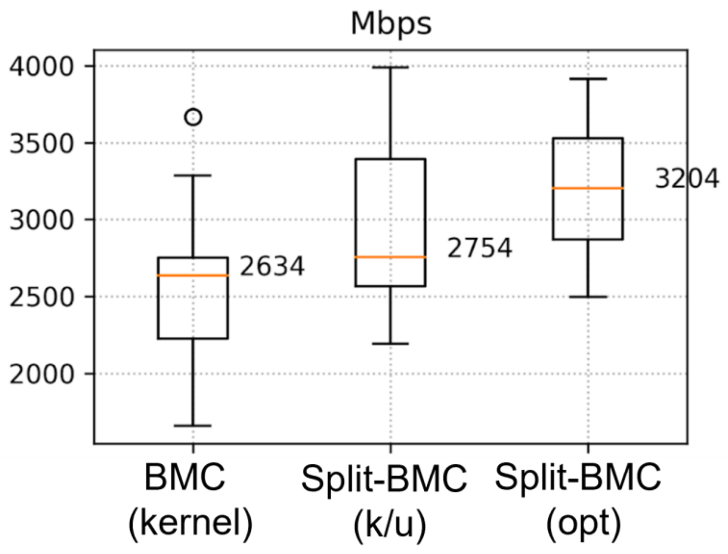


[1] Shahinfar, F., Miano, S., Sanaee, A., Siracusano, G., Bifulco, R., & Antichi, G. (2021, December). The case for network functions decomposition. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies* (pp. 475-476).

Second Step: Optimize the split program

- Automatic decomposition of eBPF programs between kernel and userspace [1] to achieve:
  - **Expressiveness**
  - **Performance**

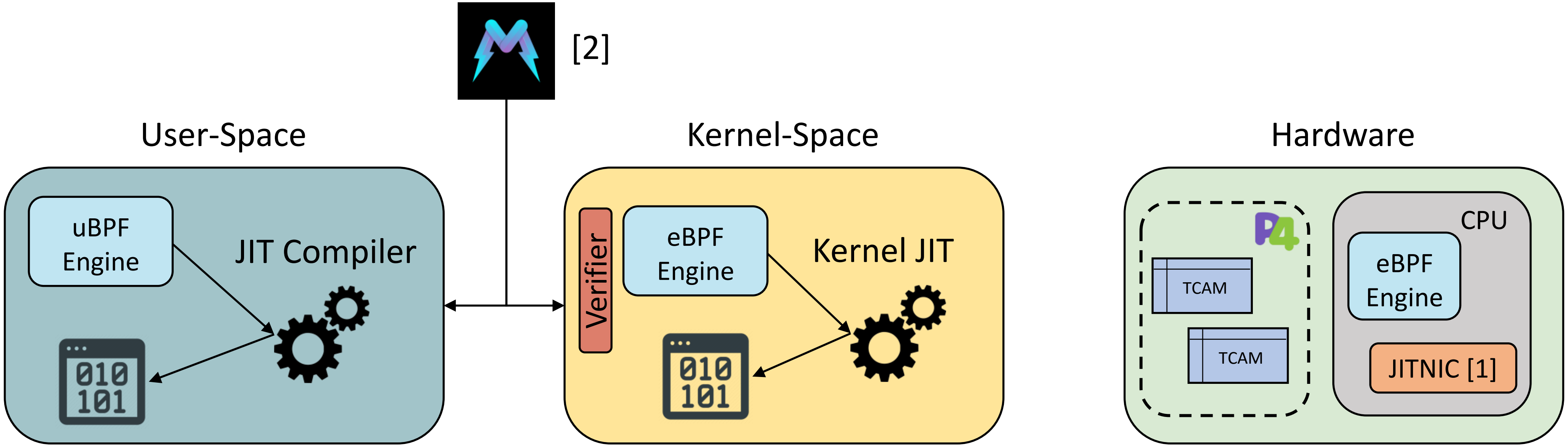




[1] Shahinfar, F., Miano, S., Sanaee, A., Siracusano, G., Bifulco, R., & Antichi, G. (2021, December). The case for network functions decomposition. In *Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies* (pp. 475-476).

Nth Step: JIT-Compile Entire Network Stack

- **Insight:** the performance of data places depend on **runtime** conditions
  - Why don't we dynamically optimize the generated programs?
  - Across all the layers in the stack?



[1] JITNIC – eBPF and P4: Better Together – Nate Foster, Cornell (https://youtu.be/CFjZfIJ1NaU)

[2] Miano, Sebastiano, et al. "Domain specific run time optimization for software data planes." *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2022.

Challenges

1. How can we guarantee correctness when the program is split between multiple parts? (e.g., hardware pipeline and software pipeline)
2. How can we handle the hardware/kernel heterogeneity?
   - Different NICs have different accelerators and different hardware architecture.
3. How to make use of available hardware accelerators?
   - Extract portion of code that can be “accelerated”