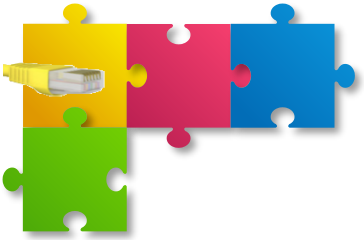
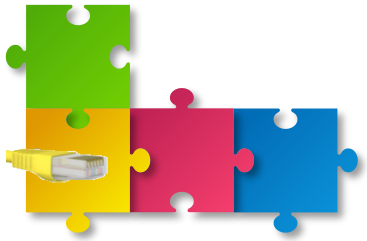


Toward an eBPF-based clone of iptables

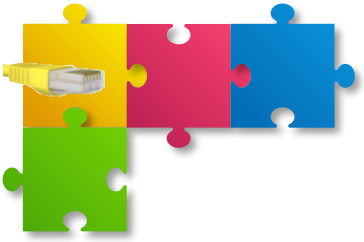
Matteo Bertrone, Sebastiano Miano, Jianwen Pi, Fulvio Risso, Massimo Tumolo

Netdev 0x12, Montréal (Canada), July 12th, 2018



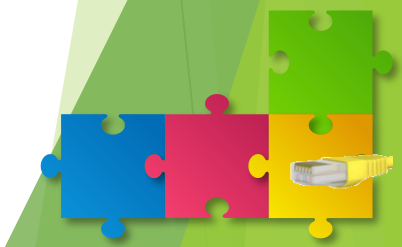
Objective

- Started in Nov 2017, with a (applied) research mindset
  - Objective: create a complex network software with eBPF, to assess strengths and weaknesses of the above technology
  - Earlier and, in some sense, orthogonal to bpfilter
- Create a (partial) clone of iptables, exploiting the eBPF technology
  - Understand the challenges
  - Provide hints about the feasibility of this project
  - Characterize performance of the prototype
  - Possibly use vanilla Linux kernels (no extensions)
  - Identify possible future improvements/directions in eBPF/Linux kernel

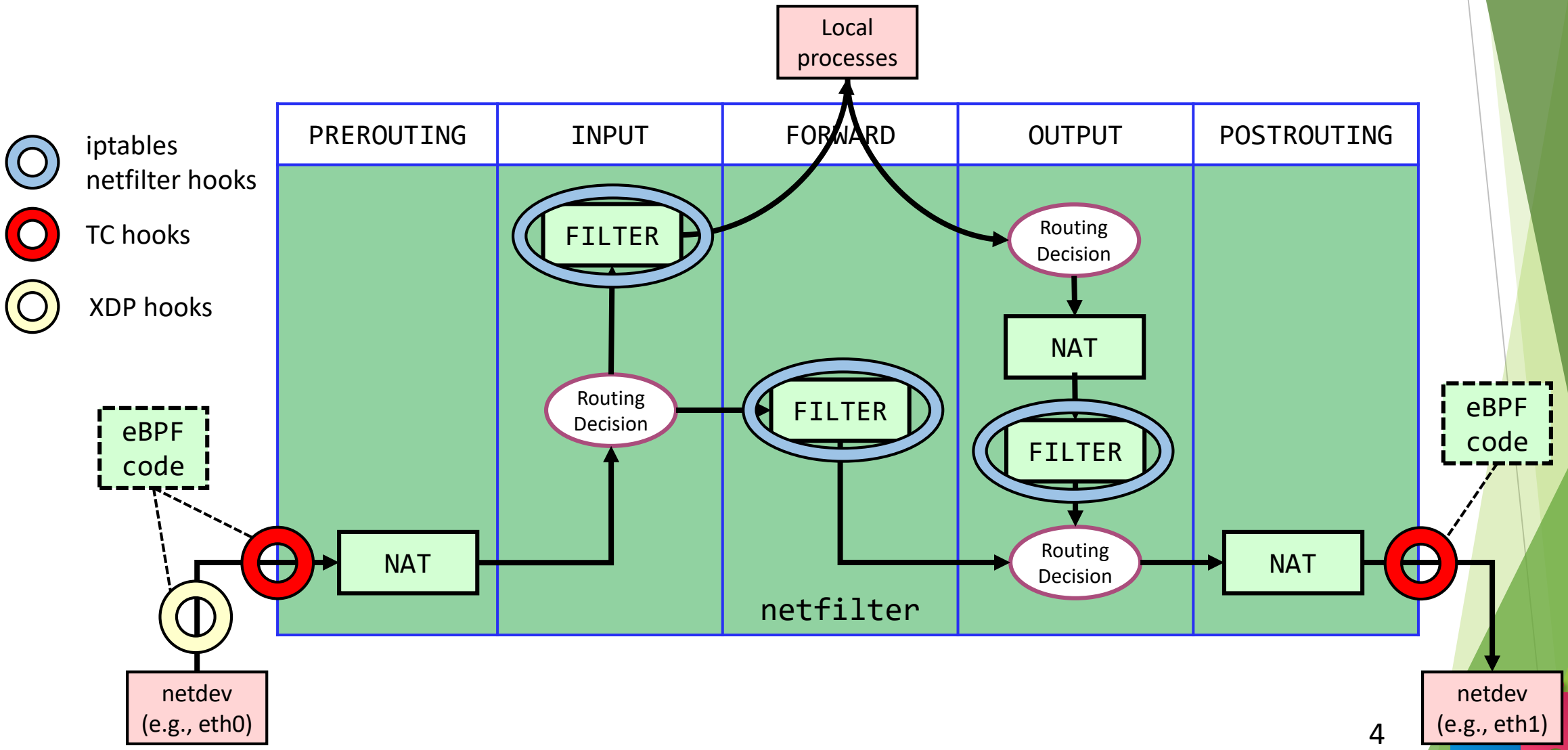


BPF-IPTABLES @ TURIN POLYTECHNIC

Part I

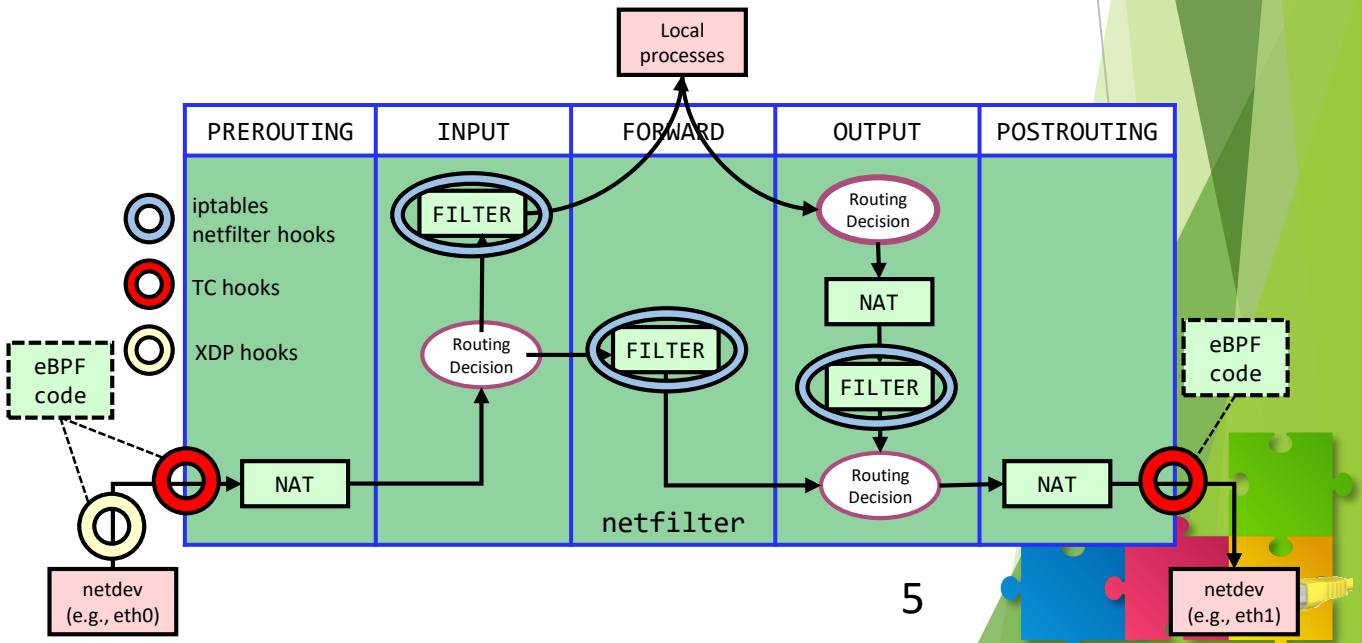


Key objective #1: Preserving iptables semantic

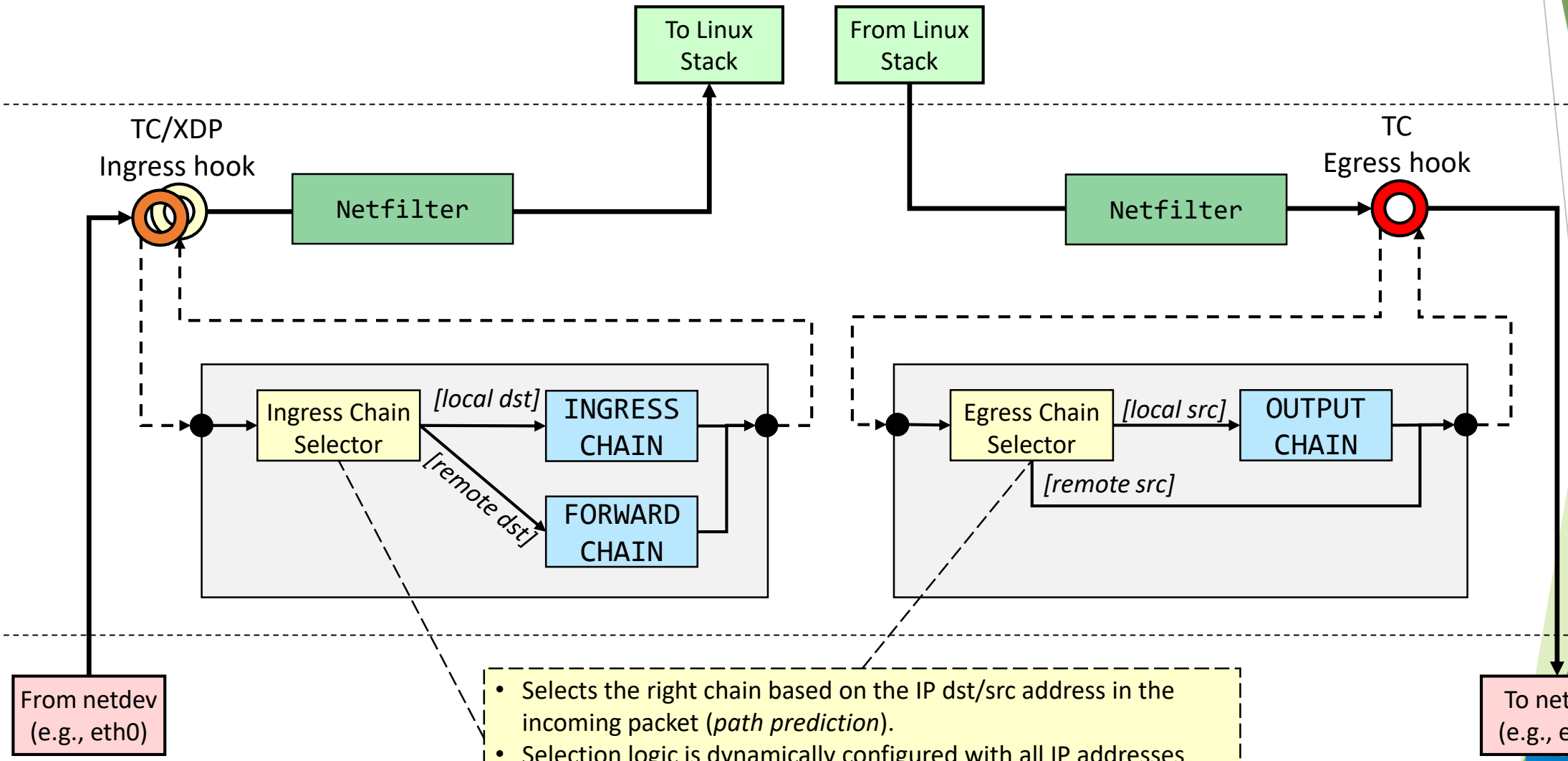


Key objective #1: Preserving iptables semantic

- We need to support existing services (e.g., security orchestrators) that are based on iptables, which is a different approach compared to bpfilter
- **Two** (TC_INGRESS/XDP_INGRESS and TC_EGRESS) hooks, which must **emulate three** chains
  - No hooks available in eBPF to easily intercept locally terminated/generated traffic
  - The eBPF code has to process only the packets that would hit each specific chain (INPUT, FORWARD, OUTPUT)
    - We have to "guess" very early which chain will be hit
  - XDP **alone** is not enough (no support for egress traffic)



bpf-iptables filtering architecture



- Selects the right chain based on the IP dst/src address in the incoming packet (*path prediction*).
- Selection logic is dynamically configured with all IP addresses visible from the root namespace (intercepts NETLINK messages).

Key objective #2: Fast matching algorithm

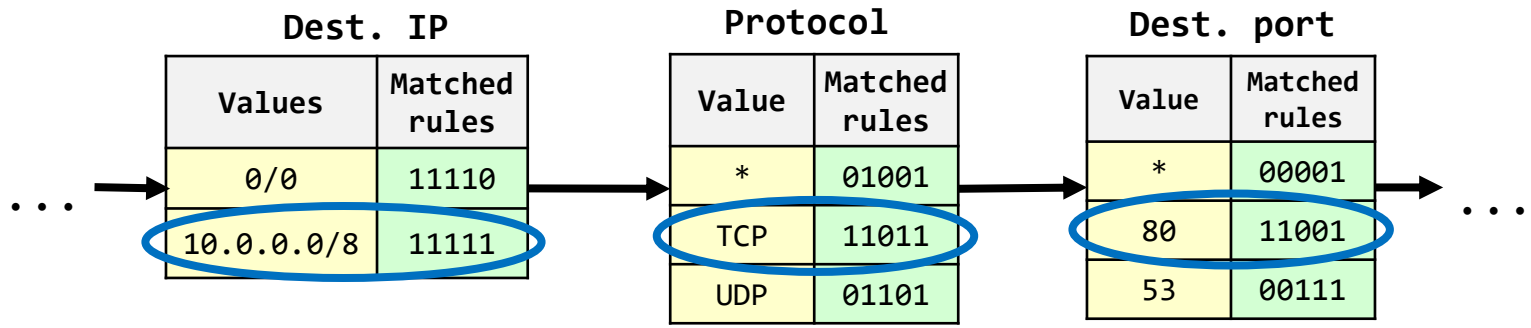
- iptables uses a linear search
- Constraints for the selection of the algorithm:
  - Feasible with available eBPF data structures (maps)
  - Possibility to rearrange the code in multiple eBPF programs to overcome the limitation of the maximum number of instructions per program
- Chosen Linear Bit Vector Search [1]

[1] T. Lakshman and D.Stidialis. High speed policy-based packet forwarding using efficient multi-dimensional range matching. In Proc. ACM Sigcomm'98, Sept. 1998

Linear Bit Vector Search: Overview

```
Rule #1: iptables -A INPUT                    -p tcp --dport 80 -j ACCEPT
Rule #2: iptables -A INPUT                           --dport 80 -j ACCEPT
Rule #3: iptables -A INPUT                    -p udp --dport 53 -j ACCEPT
Rule #4: iptables -A INPUT                    -p tcp --dport 53 -j ACCEPT
Rule #5: iptables -A INPUT -d 10.0.0.0/8                        -j ACCEPT
```

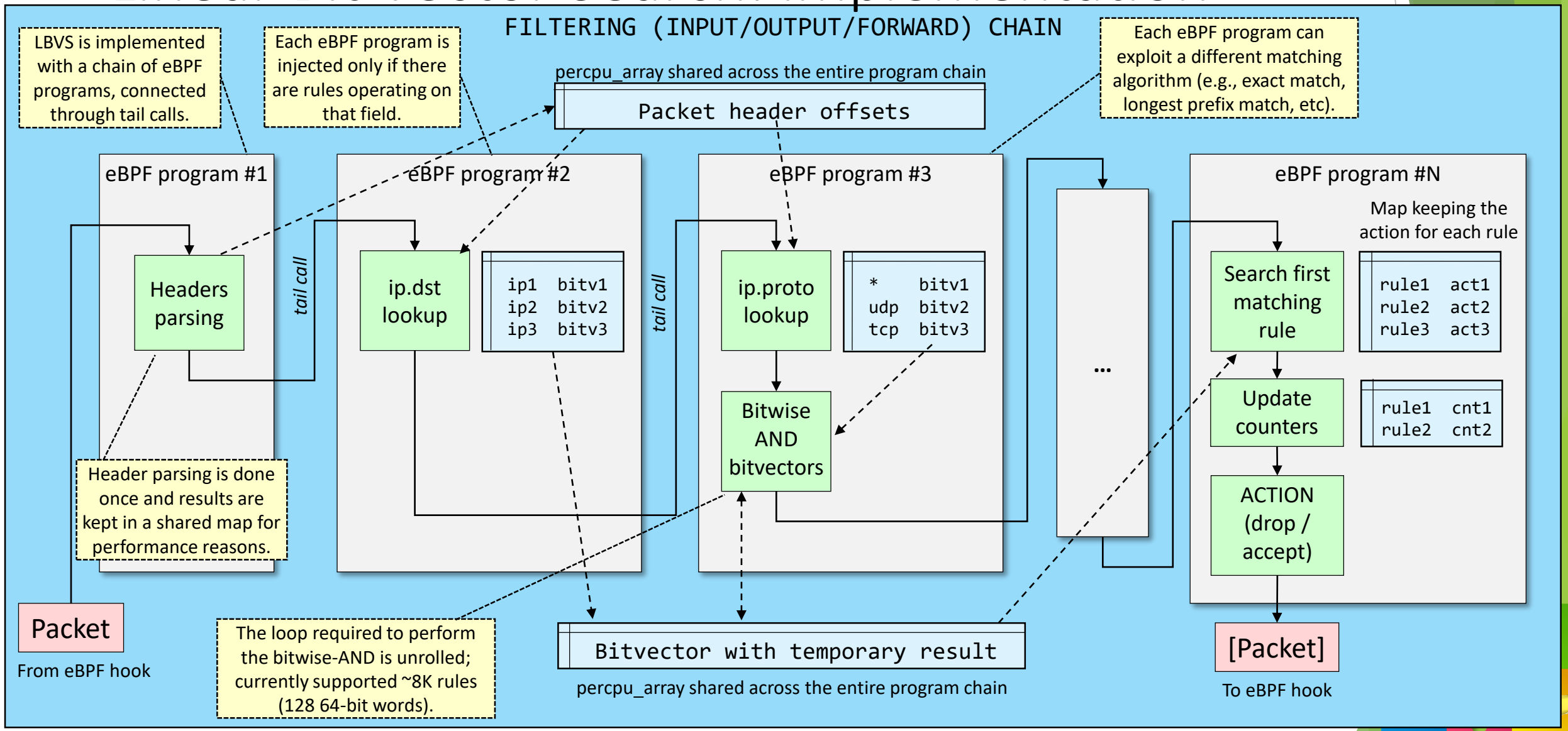
```
Input packet:
  ip.dst=10.1.0.1
  ip.proto= TCP
  tcp.dport= 80
```



11111 & 11011 & 11001 = 11001

→ **Rule #1**

Linear Bit Vector Search: implementation



Implementation details

- Currently supported fields:
  - IP src/dst: Longest Prefix Match (LPM TRIE MAPS)
    - E.g. –s 10.0.0.0/8 –d 8.8.8.8/32
  - L4 protocol: Exact Match (HASH MAPS)
    - E.g. –proto tcp; –proto 0x02; –proto icmp
  - TCP/UDP ports src/dst: Exact match (HASH MAPS)
    - E.g. –sport 80 –dport 8080
  - TCP flags: Set, NotSet and Wildcard (ARRAY MAP, with all combinations)
    - E.g. –tcp-flags SYN,ACK SYN
  - Conntrack state (more later)
- More fields can be added (e.g., netdev where traffic comes from/goes to)
- Possible future work: dynamically select the best algorithm per each field

Linear Bit Vector Search: Comments

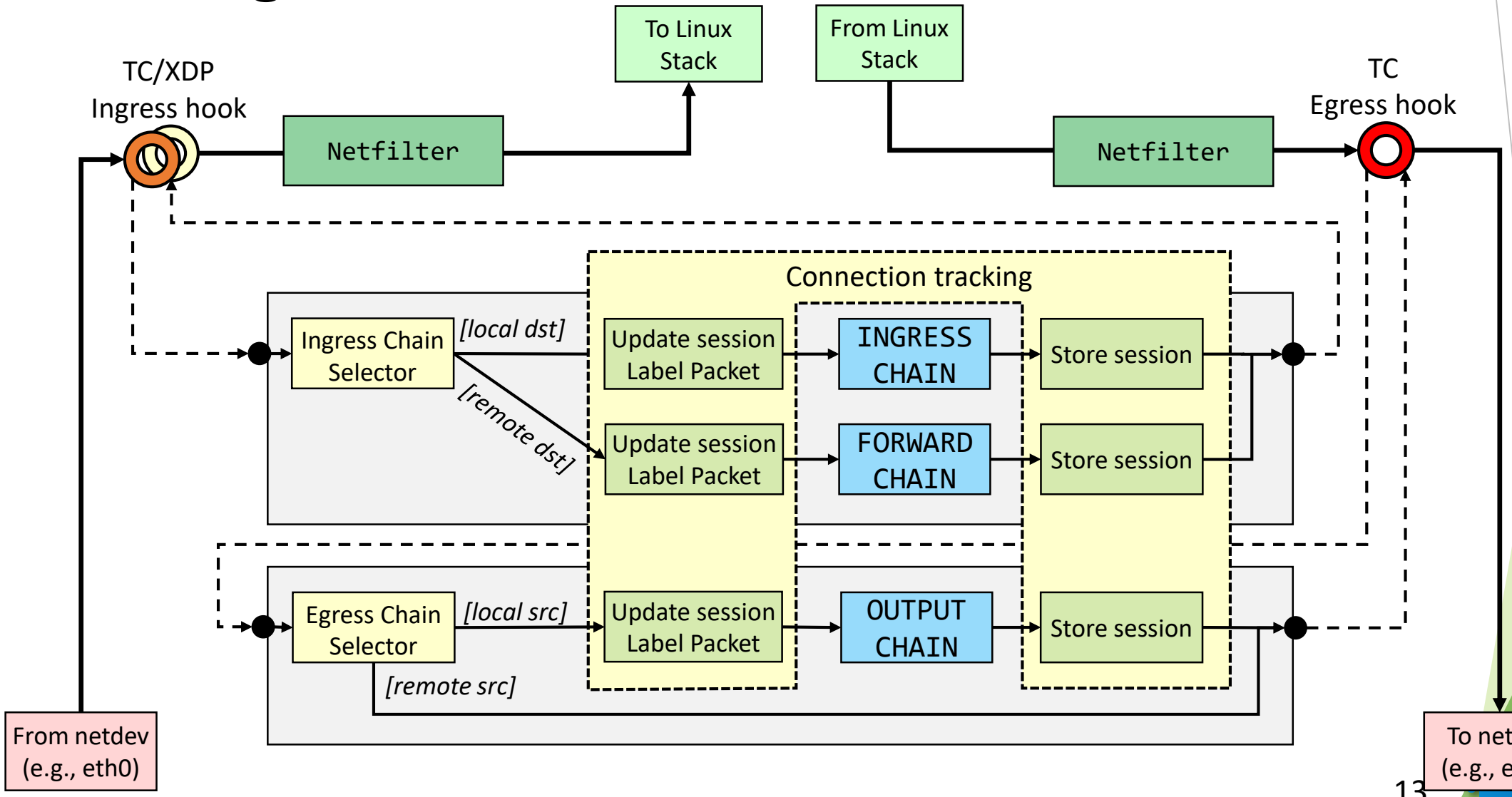
- Good
  - Good overall performance, particularly when an high number of rules is involved
  - Feasible with current vanilla eBPF
- Bad
  - Rule update time could be critical, as it may require ~second(s)
  - Still linear in the number of rules (timed the number of fields), although executed with x64 parallelism
- Future steps
  - Among the many options (e.g., HiCuts, Efficuts, etc.) Tuple Merge[1] could be an interesting algorithm to look at
    - Fast “online rule update” time
    - Should be doable with eBPF
  - In our opinion this is not the most critical point right now.

[1] James Daly and Eric Torng. “TupleMerge: Building Online Packet Classifiers by Omitting Bits.” *Proceedings of the 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017.

Key objective #3: Connection tracking

- Enable rules operating on connection state
  - E.g., -A INPUT –m conntrack –ctstate=ESTABLISHED –j ACCEPT
  - E.g., -A INPUT -s 10.1.0.0/16 –m conntrack –ctstate=NEW –j ACCEPT
- Prototypal eBPF-based implementation, with **basic** support for TCP/UDP/ICMP

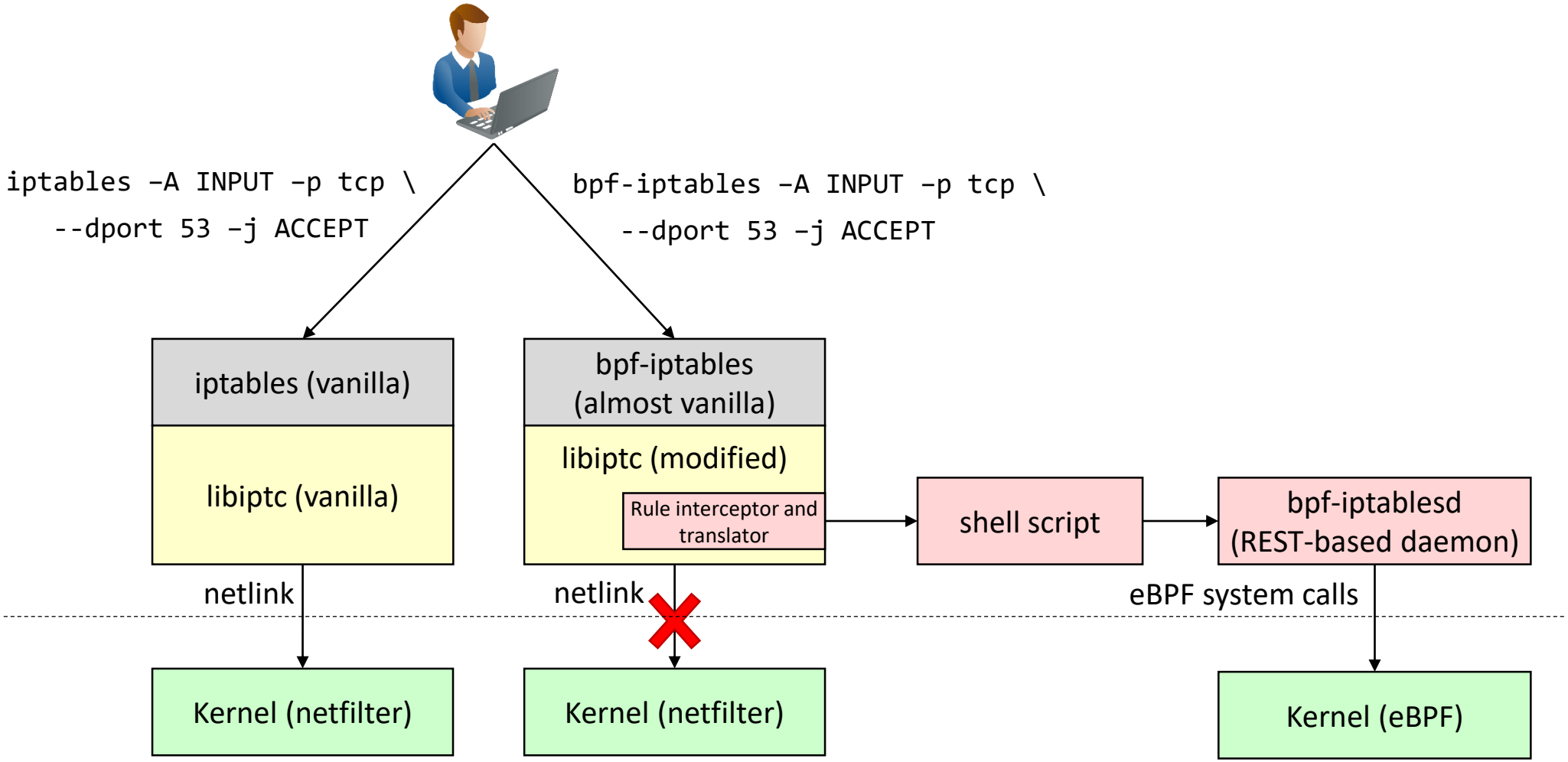
Filtering architecture with conntrack



Pro and con

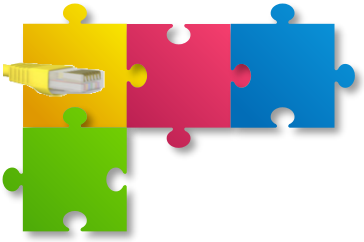
- Limitations
  - No support for the massive validation checks (e.g. on TCP window) performed by Linux conntrack, as well as IP de-fragmentation, which could not be straightforward in eBPF
  - No support for (application-layer) RELATED connections
    - “RELATED”: the packet belongs to a “new” connection, but is associated to an existing connection (e.g., FTP data transfer, a VoIP RTP stream)
    - Access to L7 fields and handle upper layer protocols with eBPF could not be doable at the time maing
    - Instead, ICMP errors are recognized and handled
- Advantages
  - Available also when the eBPF program is attached to XDP hooks

Key objective #4: Preserving iptables syntax



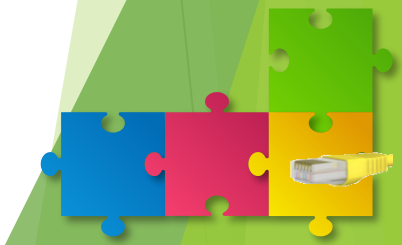
iptables vs. bpf-iptables

- Allow both executables to coexist on the same machine
  - User can choose the one he likes based on a different executable name, all supporting the same command line
- bpf-iptables supports a subset of the iptables commands
  - Examples of unsupported cases:
    - Negation: -proto tcp –sport !80
    - Range: -proto tcp –dport 1024-2048
  - An unsupported command triggers an error on the command line
- Probably does not make sense to support *all* the features of iptables

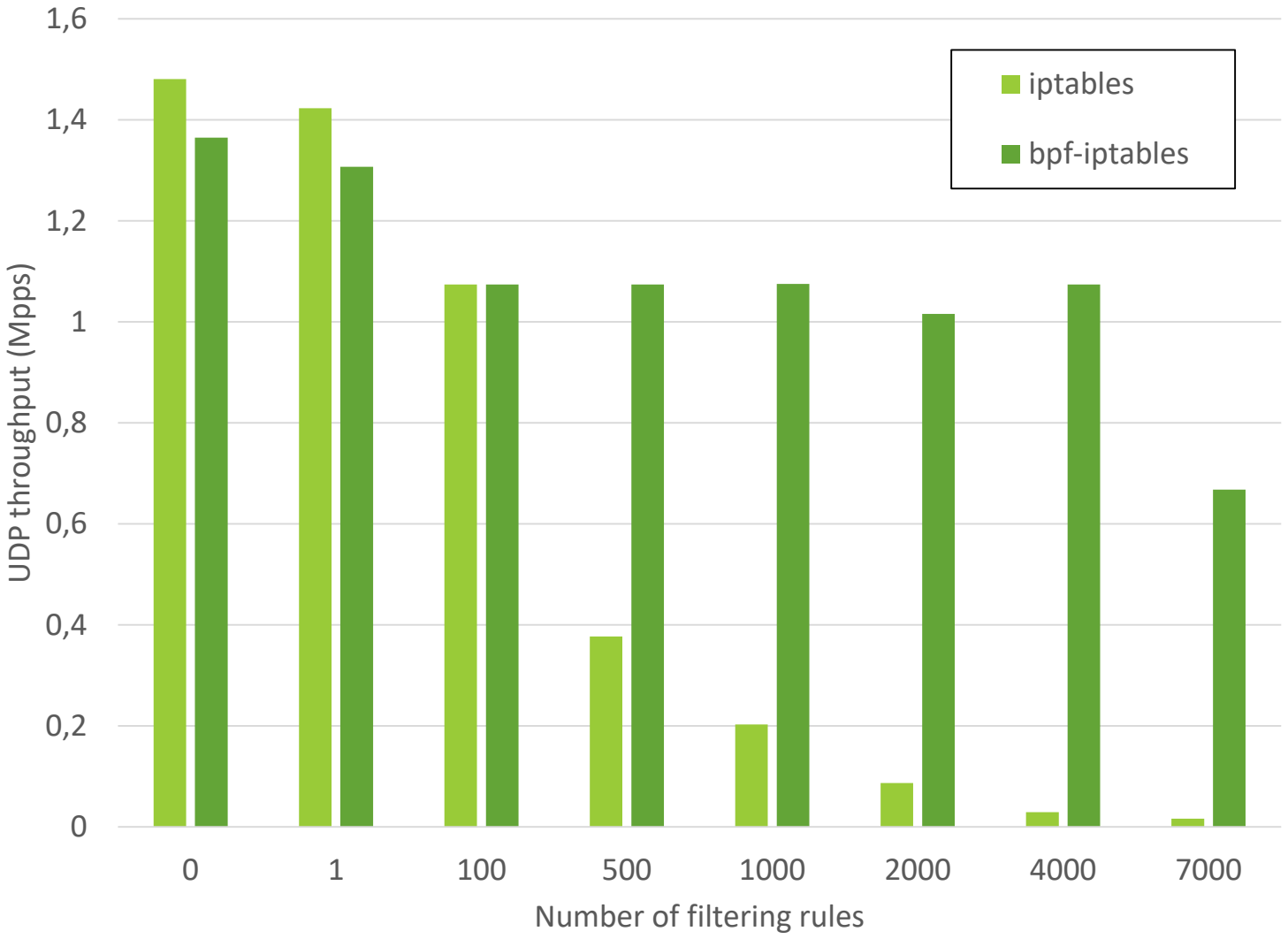


Preliminary performance evaluation

Part II



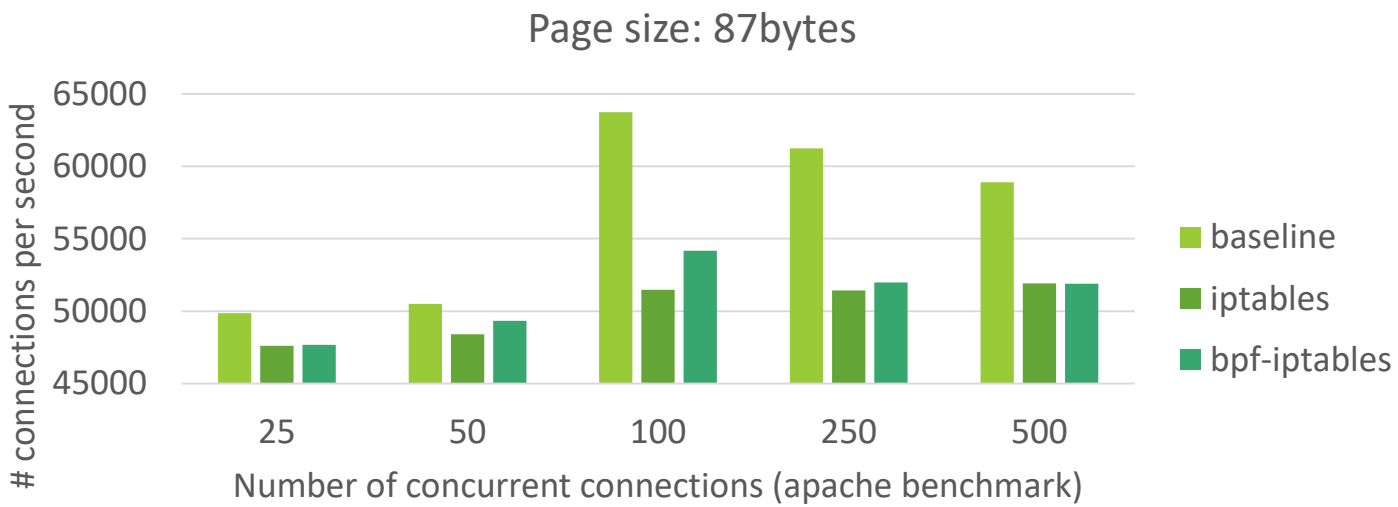
Throughput (packets per second)

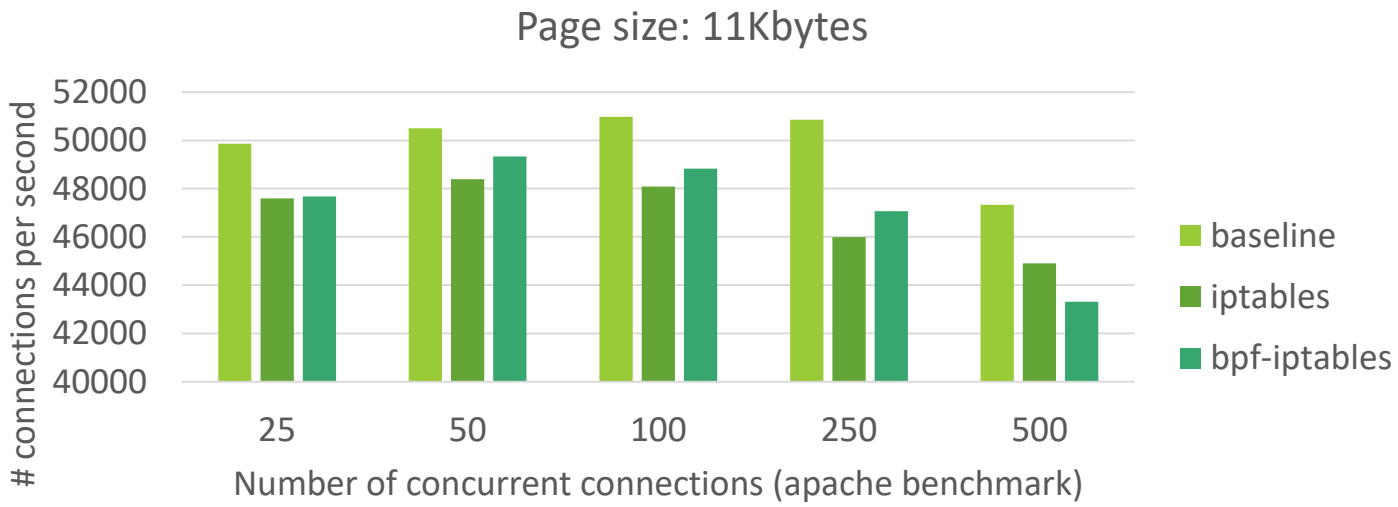


Testing conditions

Two Intel i7-4770 servers, 32GB RAM
Two direct 10GbE links (A → B → A)
Server 1: pktgen-dpdk traffic generator
(UDP traffic, 64B Ethernet frames)
Traffic measured when losses < 1%
Server 2: iptables + IP forwarding enabled

Throughput (TCP connections per second)

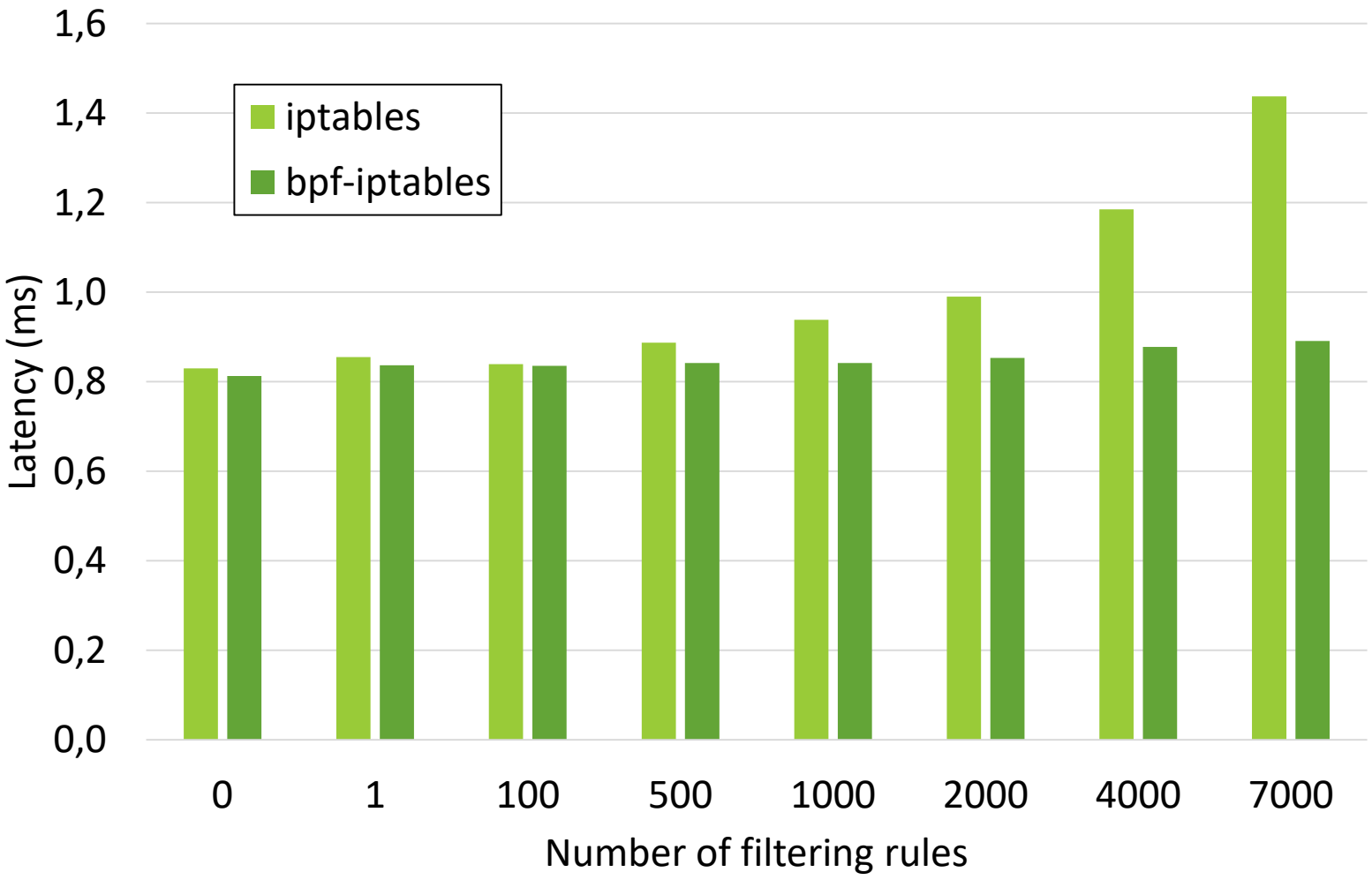




Testing conditions

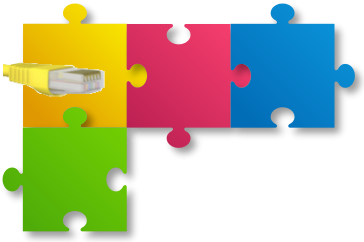
Two Intel i7-4770 servers, 32GB RAM
One bidirectional direct 10GbE link (A ←→ B)
1000 filtering rules active
Server 1: traffic generator
(ab, different degrees of parallelism)
Server 2: iptables + apache web server
(iptables forced to run on a single CPU core by configuring interrupt affinity)

Processing latency (ms)



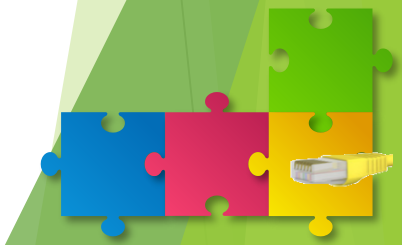
Testing conditions

Two Intel i7-4770 servers, 32GB RAM
Two direct 10GbE links (A → B → A)
Server 1: traffic generator
(ping, one ICMP echo req. per second)
Server 2: iptables + IP forwarding enabled

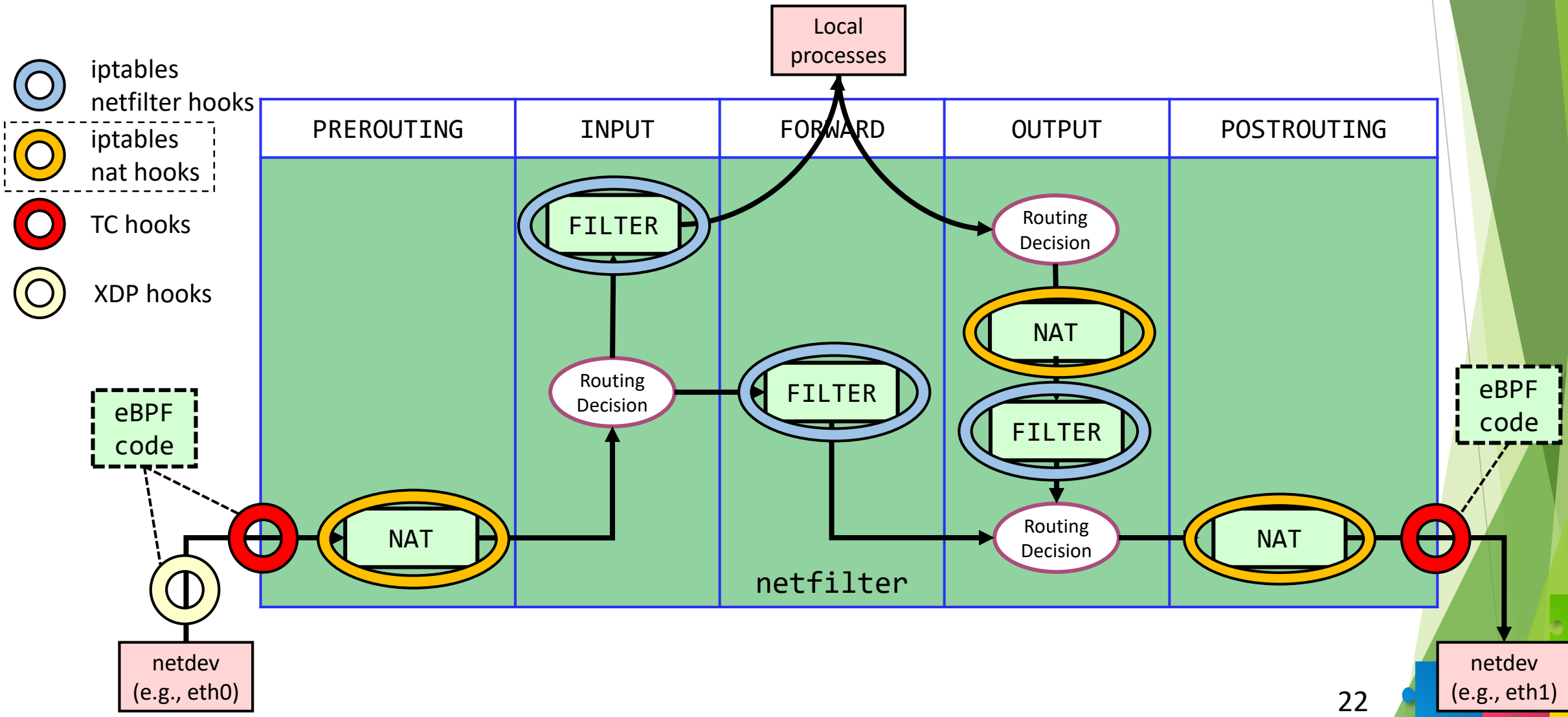


Future Work

Part III



#1. Integration with NAT



#1. Integration with NAT

- Netfilter NAT is no longer semantically equivalent when filtering is done with eBPF
  - E.g., in INPUT packets would cross filter and then NAT, while in vanilla Linux they do the opposite
  - We do not have any hook to get packets *after* the netfilter NAT
  - Hence, we have to implement both NAT and filtering in eBPF
- Integration for PREROUTING and POSTROUTING is simple, just put a NAT module in front of the eBPF filtering chain
  - OUTPUT NAT is rarely used (as far as we now), so we're not investigating that part
- Should we really need to add NAT as well?
  - It depends, but even simple apps (e.g., Docker) make use of NAT

#2. Integration with bpfilter

- bpfilter has two main objectives
  - Provides a way to intercept iptables command when being injected in the kernel
  - Defines a mechanism that can evaluate some policy rules as early as possible
- bpfilter does not have a way to preserve the semantic of iptables rules

- Our work is largely orthogonal to bpfilter
  - We can easily get rid of our code that intercepts policy rules and move to the mechanism provided by bpfilter
  - Possible nice direction to explore: moving the evaluation of some rules as early as possible (**while preserving the iptables semantic**)

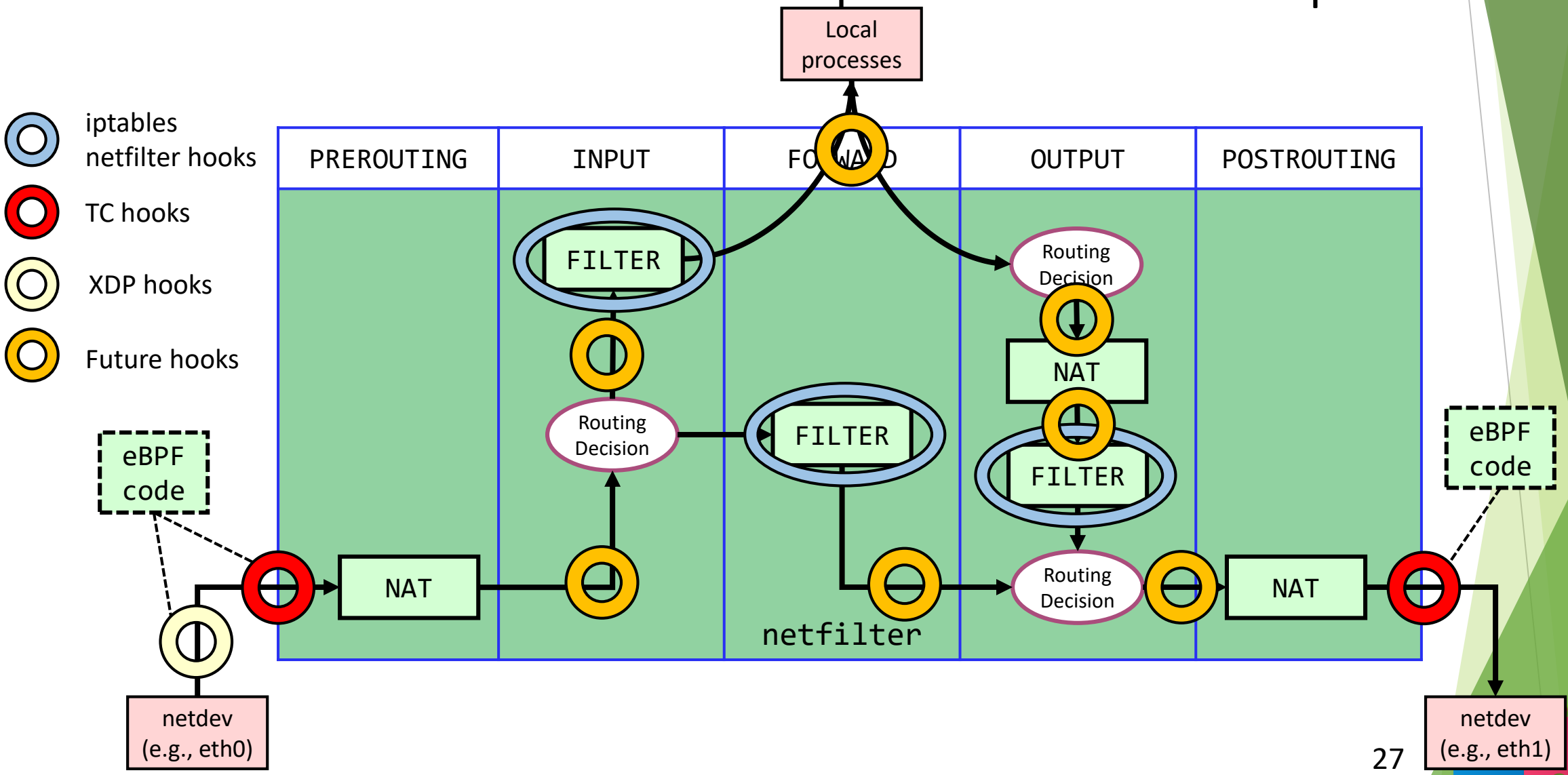
#3. Improving time for updating rules

- Rather high compared to other solutions (~1s)
- Two reasons
  - We should optimize our control plane algorithms
  - eBPF code injection time is not that small, particularly when an high number of programs and maps need to de changed

Lessons Learned (or Ideas for Discussion)

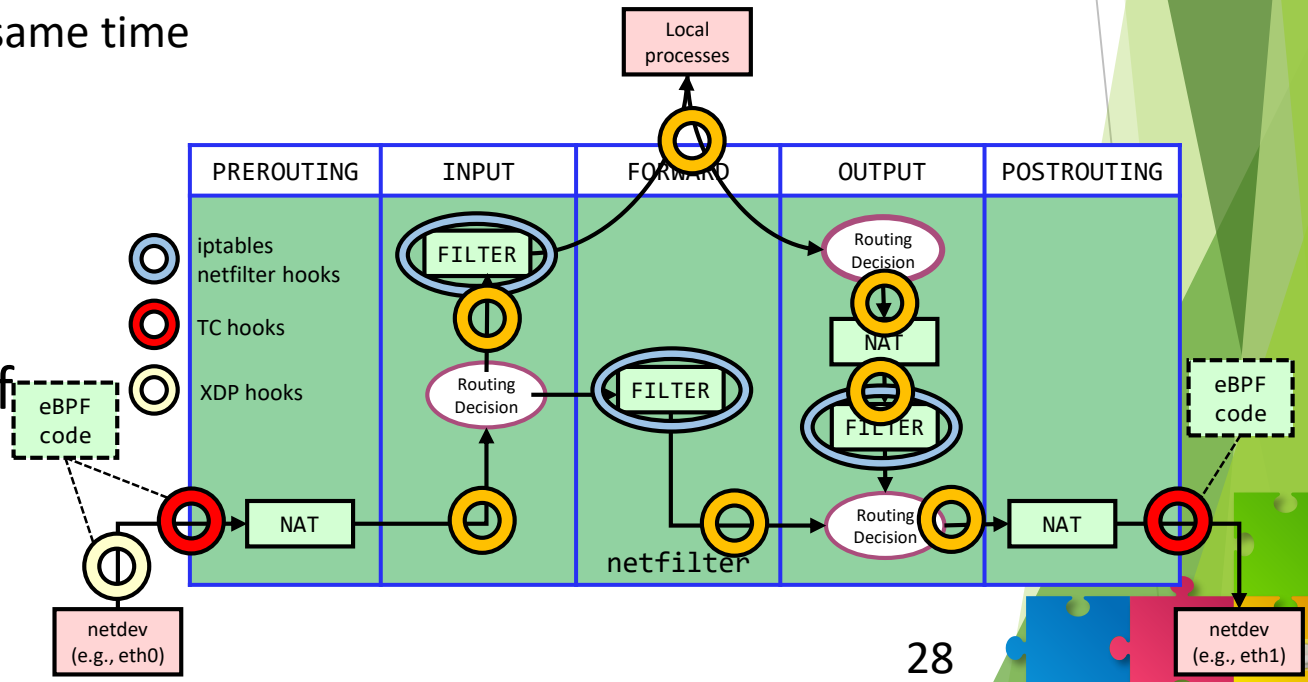
Part IV

#1. eBPF vs. Netfilter: Cooperation or Competition?



#1. eBPF vs. Netfilter: Cooperation or Competition?

- Difficult to enhance netfilter by adding individual components based on eBPF
  - E.g., cannot replace filtering while using the netfilter NAT
- Adding more hooks around would simplify the integration and coexistence of both technologies
  - From **competition** …
    - Cannot have both technologies at the same time
  - … to **cooperation**
    - Use the best of both worlds
- E.g., an eBPF hook to intercept local traffic would remove the necessity of the “Chain Selector” block



#2. A way for the “perfect” matching algorithm

- Hard to find a *one size fits all* algorithm
  - We chose LBVS because of its **run-time performance**
  - Other people need an algorithm supporting **high frequency rule updates**
  - And more…

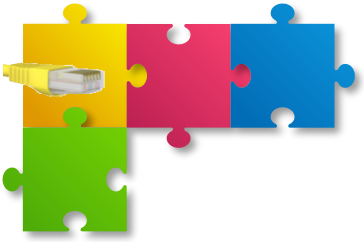
- eBPF can enable different people to implement the matching algorithm that is the best fits for their objectives
  - Difficult to achieve with other technologies (e.g., current iptables)

#3. Replacing or offloading iptables?

- Our work so far has been in the direction of **replacing** iptables
  - A (almost) fully compatible clone
  - People in our group are even working on NAT
- Is this the right way to go?
- What about **offloading** some iptables rules, e.g., in a dedicated BPF program executed in XDP?
  - E.g., rules that filter traffic coming from a huge set of malicious hosts
  - Preserving the iptables **semantic** is definitely a must

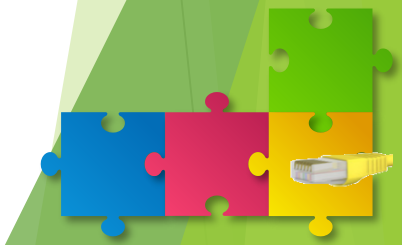
#4. Need more work on Connection Tracking

- We implemented our own minimal conntrack
  - Other people did the same, e.g., Cilium folks
  - But no advanced features, e.g., IP reassembly, “related” connections, etc, which are currently not feasible in eBPF
- Is it the right choice to have conntrack as eBPF?
  - Difficult to have all the features of the existing (native) connection tracking
- We probably need to investigate more



Concluding remarks

Part V



Conclusions

- 100% compatible version of iptables does not look a good idea
  - What about *ip6tables*, *arptables* and *ebtables*?
- A reduced (and faster) version looks more appealing
  - Enables best of both words (command line compatibility, features, performance)
- Performance characterization
  - Excellent results when an high number of rules are configured
  - Below iptables with a few rules (or none)
  - Rule update not so fast (~ 10 seconds for a database of 1000 rules)
- Under investigation if new eBPF hooks/helpers could improve the performance
  - For sure, it may enable a better cooperation between the eBPF world and netfilter