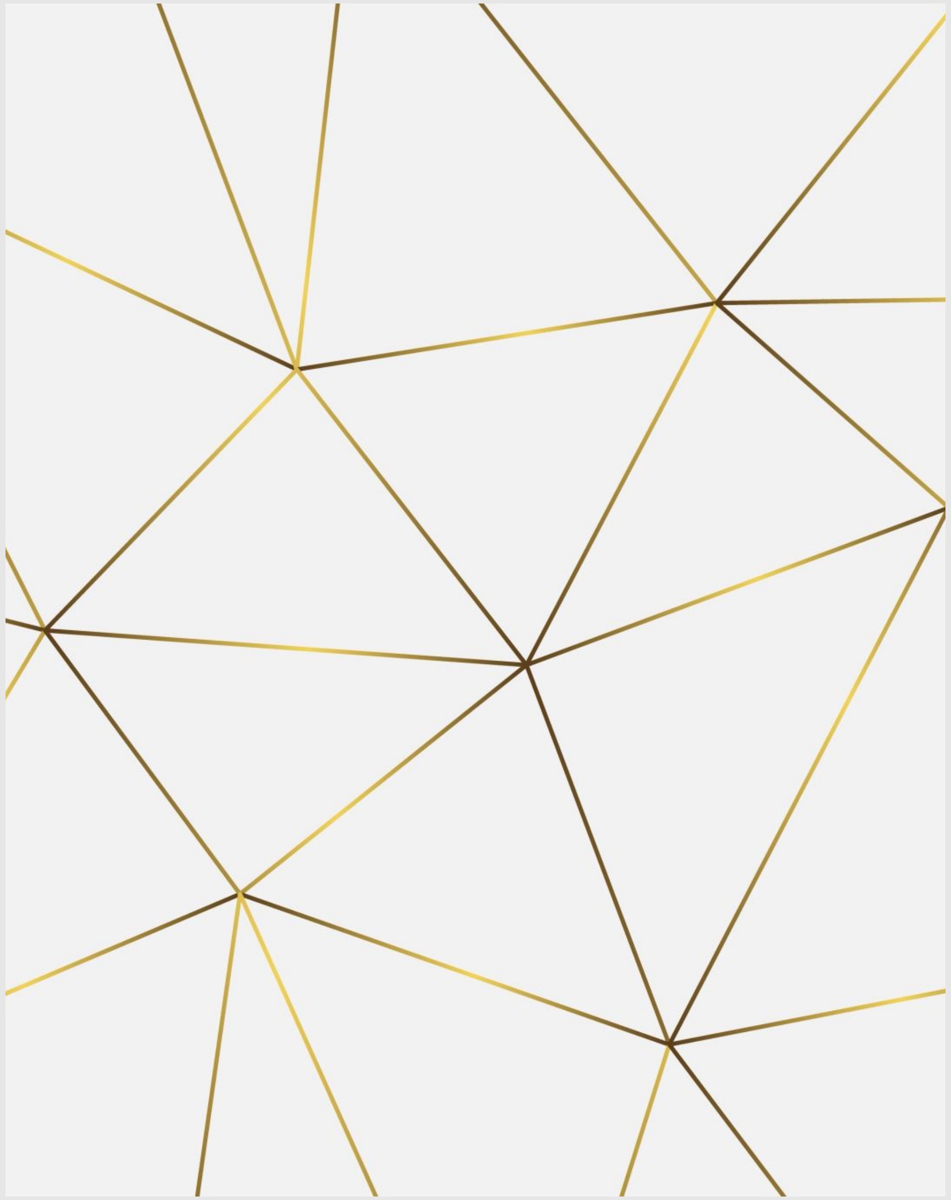
Arion: An intelligent programmable data plane framework

v0.3

Wei Yue

Futurewei Cloud Lab

2/2022



Where does the name Arion come from?

In Greek mythology, Arion is a divinely-bred, fabulously fast, black-maned horse.

According to the first-century BC Latin poet Sextus Propertius, "Arion spoke".

Arion is often pictured as a horse with wings.

In other words, Arion indicates *super fast* and *intelligent*.

https://en.wikipedia.org/wiki/Arion_(mythology)

https://greekgodsandgoddesses.net/myths/arion/

Problem Statement

In legacy OpenStack IaaS network solution, flow tables in each compute node become too big as the cloud scales up to tens of thousands of nodes; consequently, the resource cost for flow table in each compute node spikes which brings down the resource usability, limits its capability to run more services and degrades the overall performance.

On-demand flow table request from compute node to control plane as an optimization option is often too slow to meet the SLA requirement.

Also, the existing IaaS solution lacks sufficient observability capability which makes it hard to monitor and debug the services running on it.

Arion Design Objectives

We design Arion as a testbed to explore P4/XDP/eBPF based large scale intelligent programmable data plane framework. The initial target use case is to apply it as a cloud gateway in OpenStack like IaaS environment which can support hundreds of thousands of nodes in VPC.

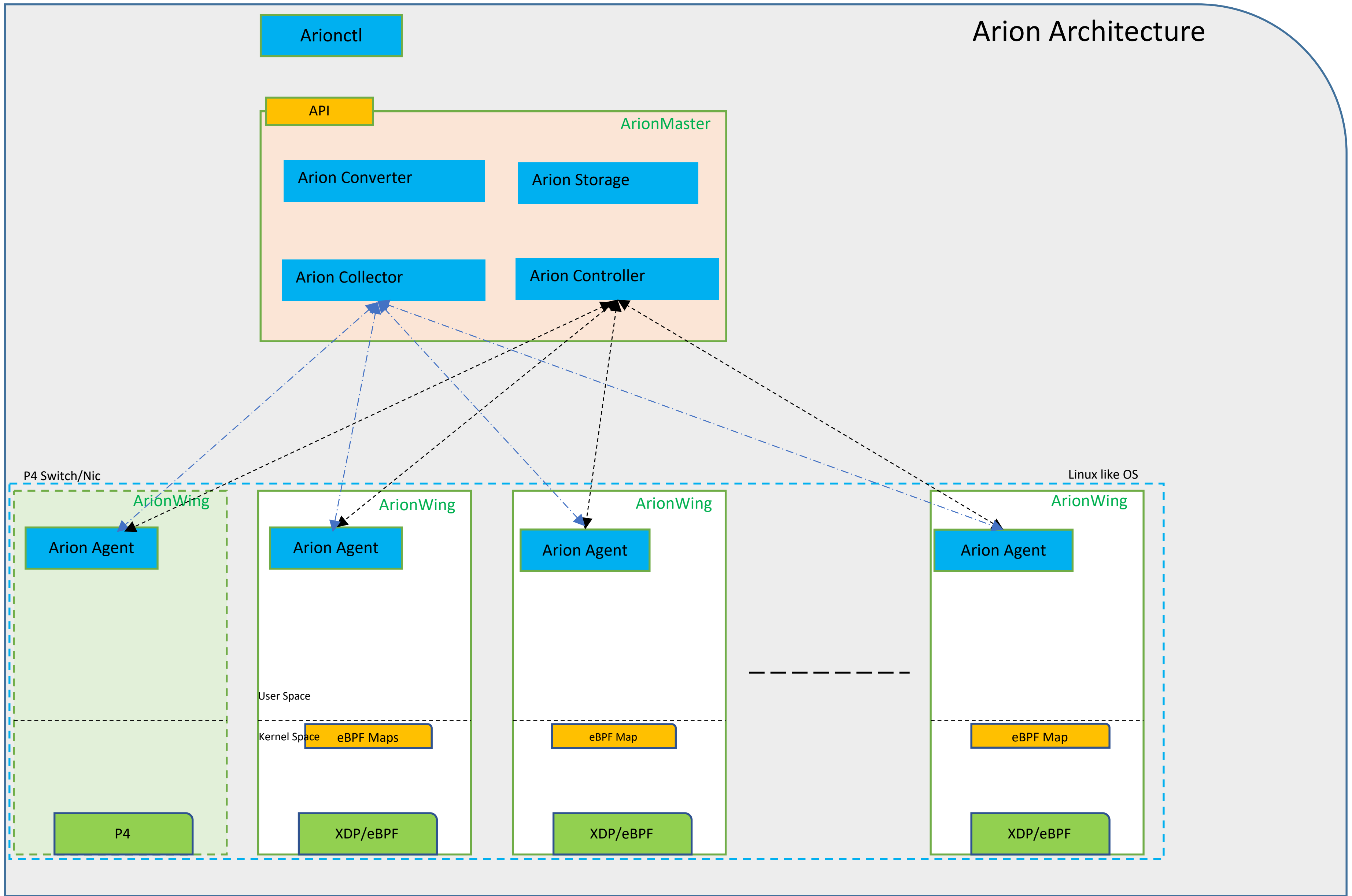
Arion DP tries to explore innovative in these areas initially:

1. Keep up with high throughput/low latency demand in large scale cloud, compare with other technologies like DPDK;
2. Innovative algorithm/data structure for efficient resource usage;
3. Programmability in support of intent description and run-time configuration.

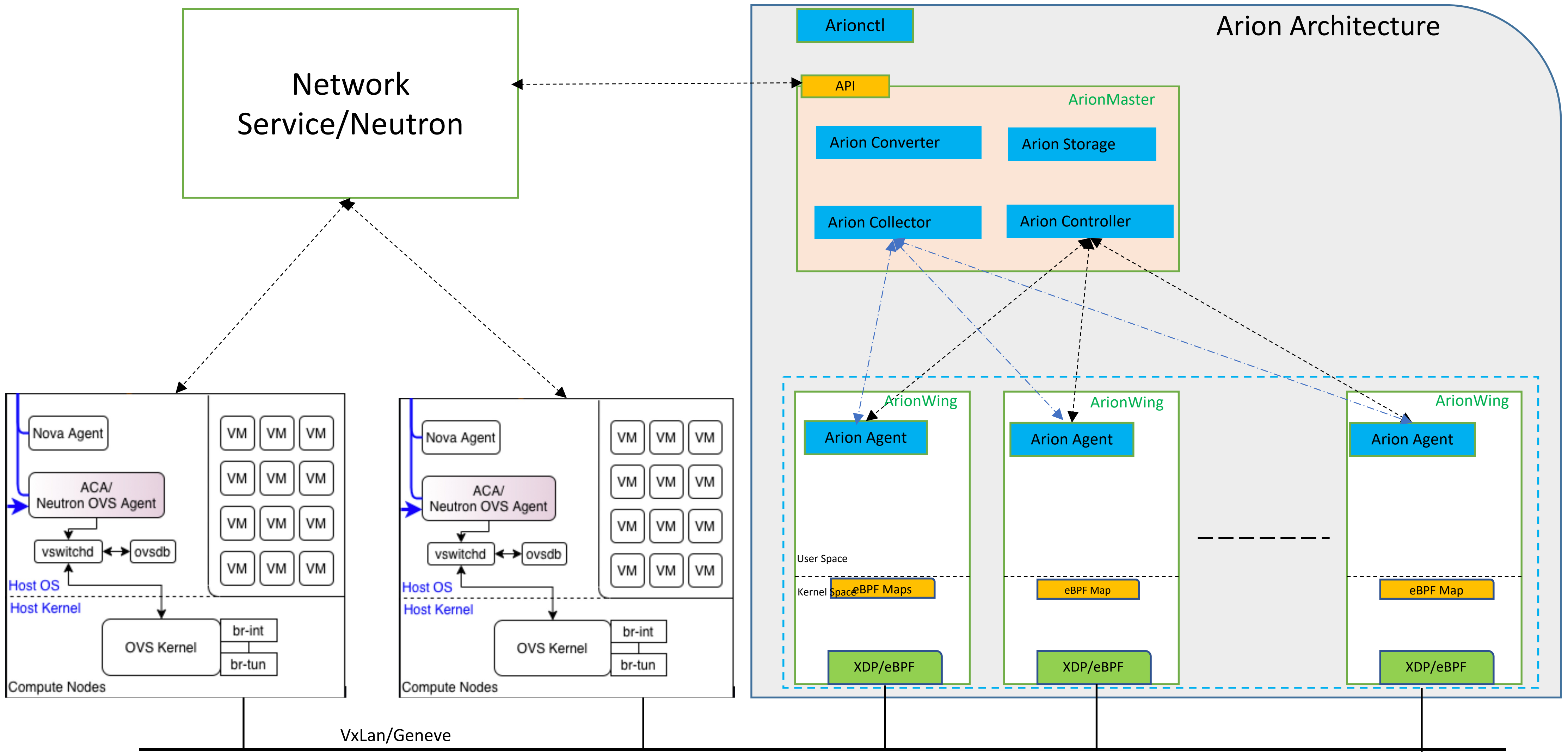
On top of that, we could explore the intelligence/new capabilities introduced by XDP/eBPF, e.g., anomaly detection, observability, etc.

And we would explore SailFish like software hardware combination approach.

Arion Architecture



Arion as a cloud gateway in an OpenStack like environment



4 Key Arion DP Characteristics

1. Fast packet processing
    - XDP/eBPF
    - P4
    - Software-hardware combination

2. High efficiency, high resource usage
    - DP direct path notification
    - Consistent hashing

3. Resiliency and flexibility
    - Arion wings formed in group, sustain multiple Arion wing failure
    - Fall-back process in user space to handle all ”other” flows
    - XDP-redirect can also handle messed up CN routing

4. Programmable, extendable
    - P4/eBPF

Phase I target goal and constraints

For each Arion cluster

Target support:

Compute Node: 100K VMs: 2M

VNI: 20 Flow rules: Neighbor rules

Mapping rules: dstip+vni -> CN IP, total rule capacity: 2M(?)

Arion Master capacity:

256G physical memory, 1T disk

ArionWing capacity:

100G Nic card, 32G physical memory, 512G disk,

12(24) ArionWings

Multiple eBPF tables in each ArionWing, each ArionWing covers rules for 25K cn/500k vm(or 12.5K cn/250K vm); sustains multi ArionWing failure and leaves space for complex rule handling(e.g. ACL, SG, cross VPC, etc).

Target supported throughput capability:

1Tbps(2TGbps) throughput and sustains temporal multiple ArionWing failures.

Notes: If we use 400G Nic card, the capacity increases to 4Tbps/8Tbps in above calculations.

Notes:

T = total Arion Wings(min 6)
N = Arion Wings in each group(default 3)
G = T/N, total groups(min 2)

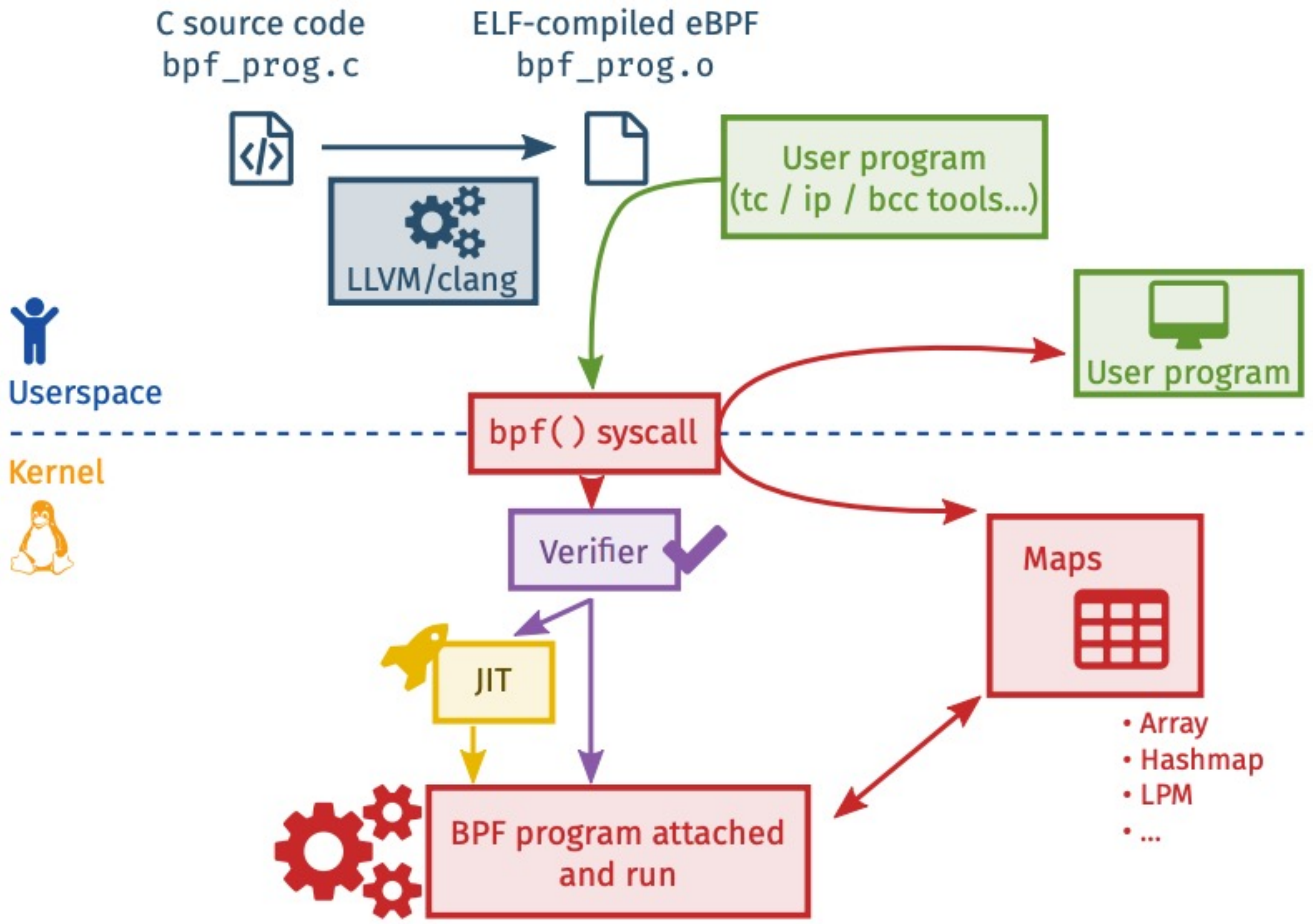
1. **Every N Arion Wings form a group, which has the same flow control rules;**
2. **Total flow control rules are sharding into G subsets.**
3. This is 1/10 of the designed deployment, which could have 120(240) Arion Wings with 10T(20T) bps throughput, or 40T/80Tbps with 400G Nic for 1M compute nodes in VPC.

Map Name	Scope	Default Limit	Scale Implications
Connection Tracking	node or endpoint	1M TCP/256k UDP	Max 1M concurrent TCP connections, max 256k expected UDP answers
NAT	node	512k	Max 512k NAT entries
Neighbor Table	node	512k	Max 512k neighbor entries

Cilium default eBPF map size

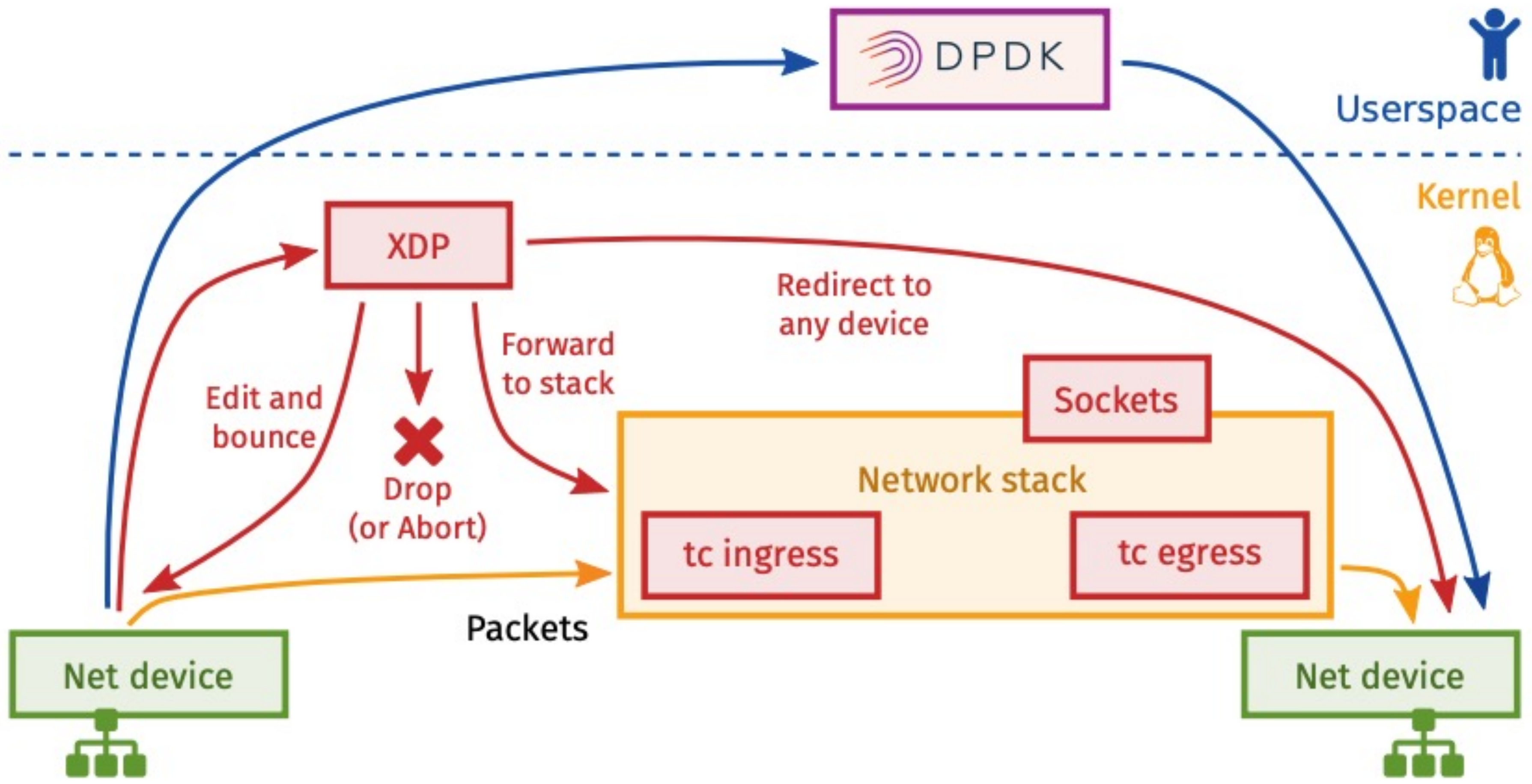
1. Fast Packet Processing

XDP/eBPF basics



1. Fast Packet Processing

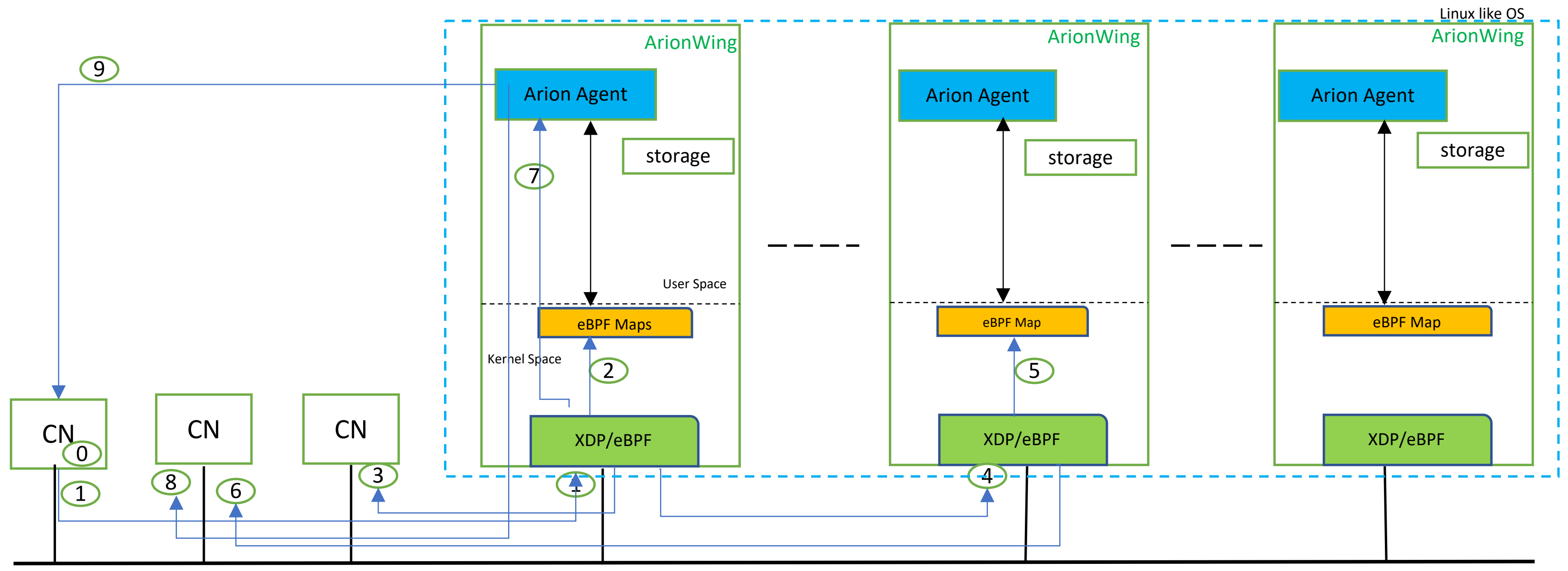
XDP/eBPF basics



1. Fast Packet Processing

Packet flow

0. Packets are routed to specific ArionWing based on vni+destSubnet;
1. Packet arrives at ArionWing;
2. If there are rules in eBPF map to handle it locally;
3. Processing it and send to destination CN;
4. Or, forward pkt to next ArionWing(designated one);
5. Check local eBPF map for rules;
6. Processing it and send to destination CN.
7. Or, forward to user space process which has remaining flow rules.
8. Processing it and send to destination CN.
9. Elephant flow notifying back to source CN



Notes

- 1,2,3 are for within VPC flow;
- 4,5,6 are for within VPC flow or cross VPC flow;
- 7,8 are with fall back process when all Arion Wings(N) within one group fails or the clients(CN) messed up routing process.
- Per flow stats are collected and elephant flow are identified and notified back to source CN in 9.

Design constraints and hardware programmable switch(P4 Tofino) consideration

Please note that throughput capacity is limited by the total Nic bandwidth in Arion cluster.

If our ultimate target use case is to support cross region with 500T throughput capacity, we need to add P4 switches(12T capacity) into the equation as a software-hardware combination solution, with P4 switch primary for cross region traffic which must go through gateway.

In this scenario, we'll start by adapting P4 representation while exploring XDP/eBPF capability in phase I at the same time.

2. High efficiency, high resource usage

2.1 DP notification flow and mechanism

Packet flow

0. CN query is routed to specific ArionWing based on vni+destSub;
1. Packet arrives at ArionWing;
2. If there are rules in eBPF map to handle it locally;
3. *Process pkt, fill in source info and send to destination CN;*
4. Or forward to user space process which has remaining flow rules.
5. *Processing it, fill in source info and send to destination CN.*
6. *CN recognize it and updates direct path information accordingly.*

*Direct-path notification capability in DP is proposed here. It’s designed to be with **minimal cost** and **minimal impact** to other components.*

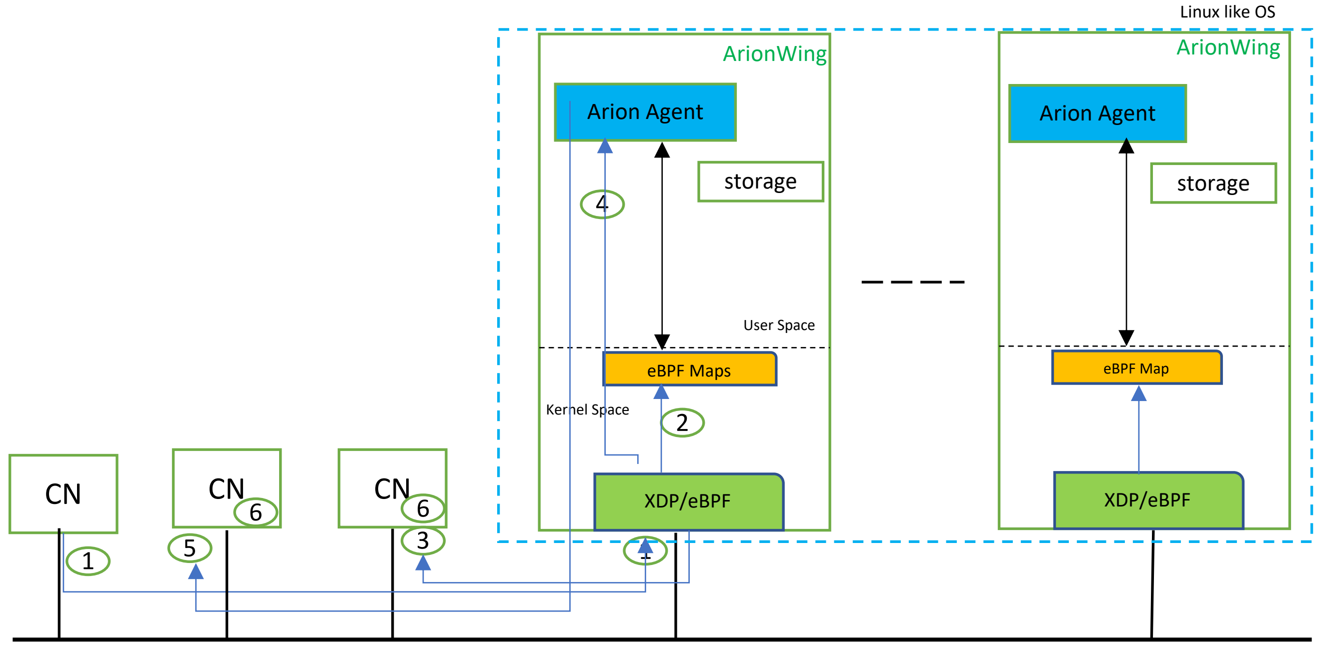
In step 3 or 5, we set ***O*** bit in

a. VXLAN-GPE header (if VXLAN-GPE is supported);
b. Or VXLAN’s flag field in header(not compliant with RFC though);
c. Or top bit in VNI(in-house work, with CN’s cooperation);

source CN IP/Mac/vni tupple is added to the packet for sending over to the destination CN.

in step 6, a match-action rule is added to recognize DP notification:
match: if ***O*** bit is set(e.g. check top bit in vni);
action: send to controller(ACA) to collect source tuple for direct path update.

This mechanism is configurable for performance concern:
1. *only apply for initial packet(s) per flow;*
2. *or apply some rate limiting; etc.*



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|R|R|Ver|I|P|B|O|       Reserved                |Next Protocol  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                VXLAN Network Identifier (VNI) |   Reserved    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: VXLAN-GPE Header

2. High efficiency, high resource usage

2.2 Arion's sharding strategy and algorithm

1. ArionMaster provides API(Restful, gRPC or others) for CN to obtain ArionWing list(<120) and responsible for sync up;

2. Arion framework provides consistent hashing APIs for CNs which maps
   "***DestSubNet + vni" to "designated ArionWing***";

3. CN/ACA uses API to locate designated ArionWing to send packet to;

4. Inside Arion framework, same consistent hashing algorithm is used so that
   "***DestIp + vni --> destination CN Ip/mac***" rules
   are injected to the same ArionWing if
   "DestIp+vni" belongs to the same "***destSubNet+vni"*** above.

2. High efficiency, high resource usage

How does Arion DP achieve high efficiency and high resource usage?

1. No full flow table in CN, only ArionWing list stored locally;
2. No on-demand request to NCM in CN, use API(algorithm) to find designated ArionWing;
3. Each ArionWing only stores partial flow table(1/G);
4. Flows are “mostly” routed to ArionWing which can handle them directly, minimizes extra hop(s);
5. DP notification is piggy-backed in existing packet, no extra channel/packets needed.

3. Resiliency and flexibility

Flow rules deployment/sync up principal/algorithm

1. ***N*** Arion wings in the same group share same flow rules for redundancy; each Arion wing is dedicated for a subset(1/***G***) flow rules; packets are routed to specific Arion wing based on the subset rules;
2. Arion maintains notification service, each CN obtains list of Arion Wings info(health, signature) via notification service and do regular health check for Arion Wing changes;
3. When there are Arion wing failures, the existing group category maintains the same when each group still has live Arion Wing(s);
4. Regroup process happens when requested or some Arion Wing remains lost/failure for threshT.
5. Notification are sent to CN only when regrouping is done(flow rule redeployment)
6. When there're no flow rules for incoming packet, we can opt to forward to next group Arion Wings or send to user space process which has the complete(or remaining) flow rules(fallback mechanism).

Notes:
Arion sustains temporal multple ArionWing failures in different scenarios.
T = total Arion Wings(min 6); N = Arion Wings in each group(default 3);
G = T/N, total groups(min 2); threshT = threshold Time(default 2mins)

4. Programmable, extendable

P4/eBPF provides programmability for how we want to deal with flow dynamically.

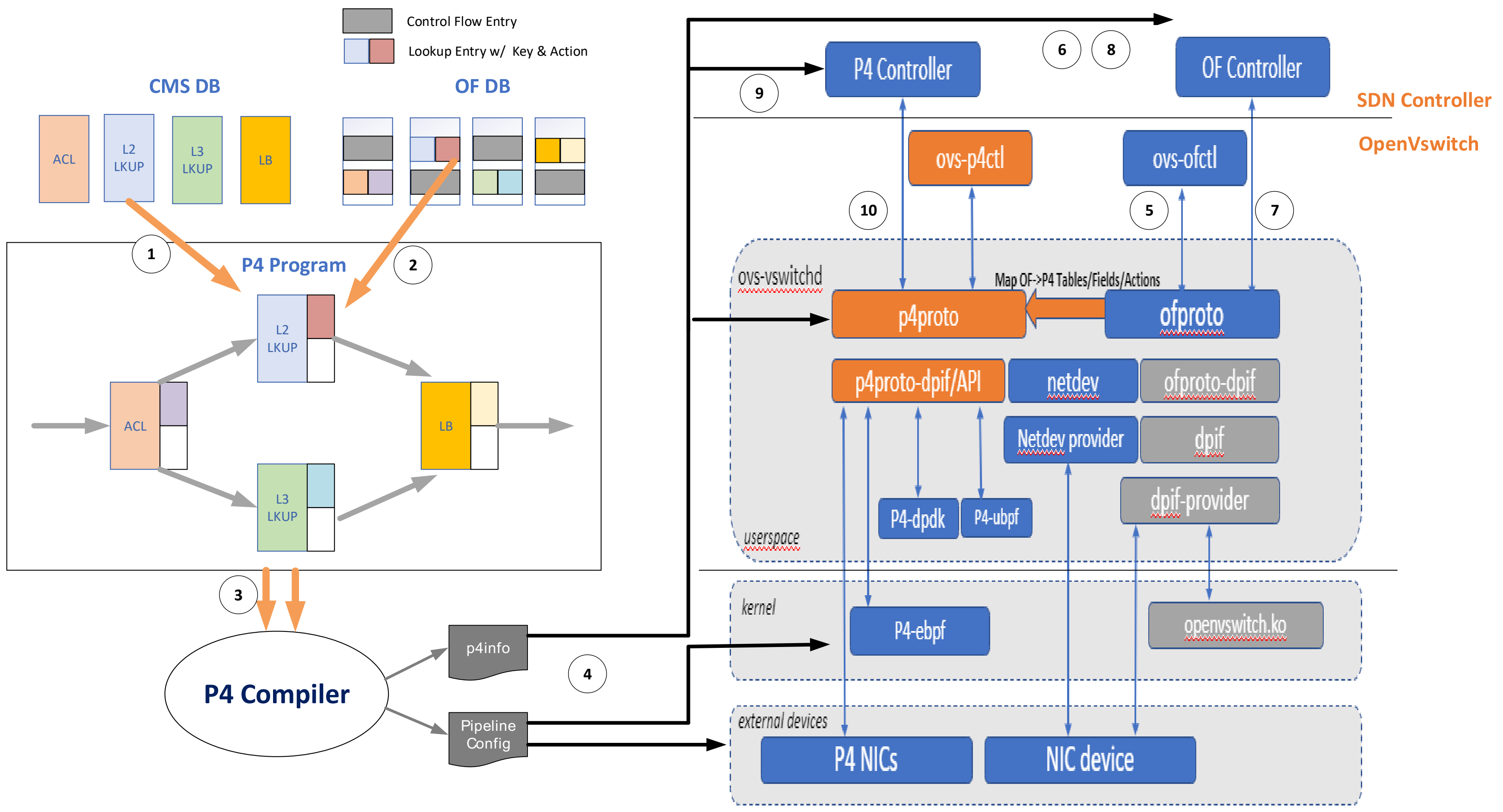
As mentioned previously, Arion throughput capacity is limited by the total Nic bandwidth in Arion cluster. Pure XDP/eBPF solution is unfeasible to handle massive cross region traffic(e.g. 500TB+), or handle an elephant flow whose throughput is higher than one single NIC's capacity.

If our ultimate target use cases need to support above scenario, we need to add P4 switches(12T capacity), which has already put lots of effort on those cases, into the equation as a software-hardware combination solution, with P4 switch acts primary for cross region traffic which must go through the gateway.

P4 hardware switch has its limitation as well(see sailfish paper), Arion DP framework gives out perfect chance to explore how P4, XDP/eBPF could work together in a software-hardware mix solution.

We'll put more extensible effort on this in Phase II. We'll start by adapting P4 representation while exploring XDP/eBPF capability in phase I at the same time.

“Potential” OpenFlow to P4 mapping steps



OpenFlow to P4 mapping basic steps

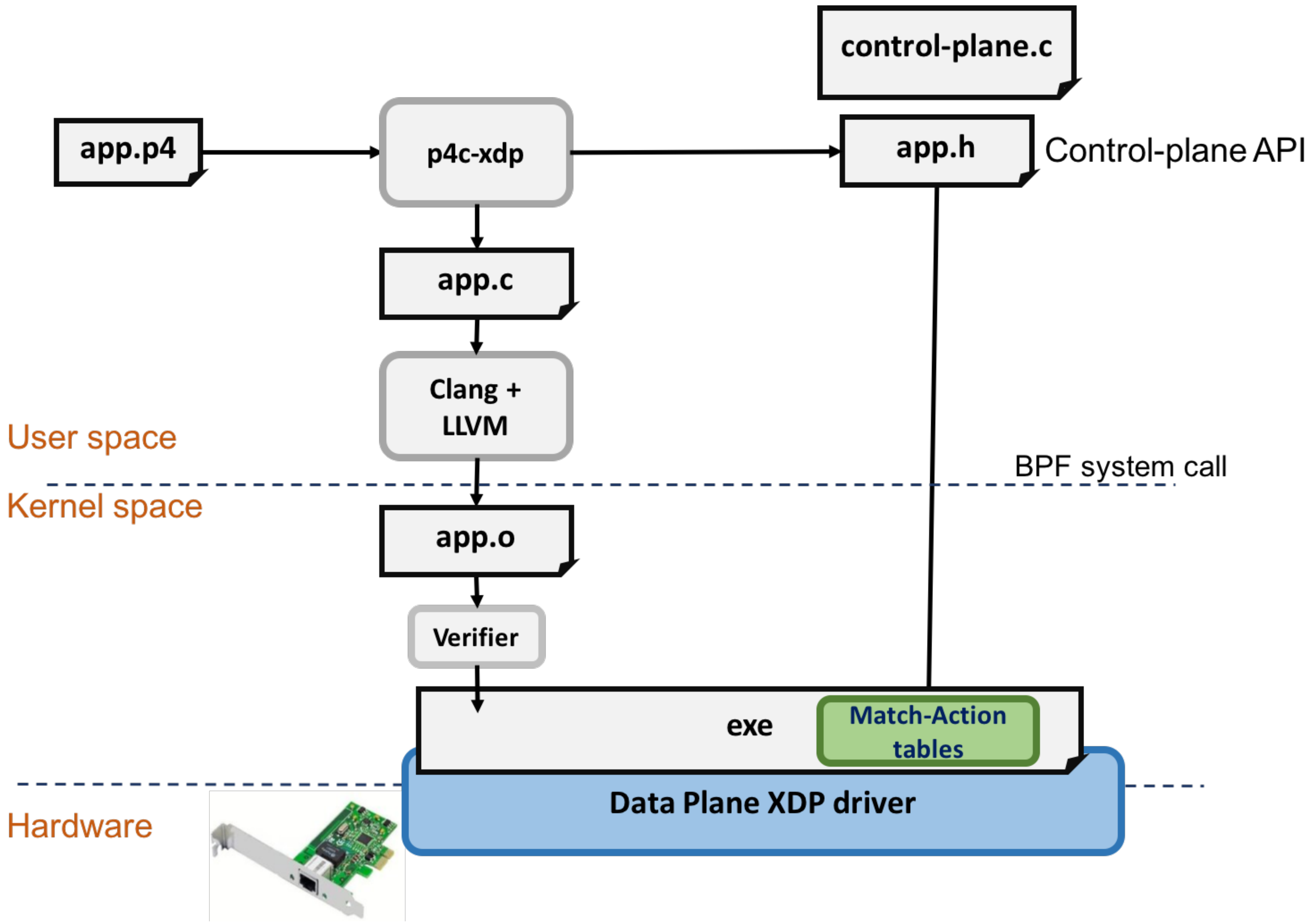
1. Identify P4 lookup tables from CMS DB
   - Identify precise encoding of the keys that reflects the intended use of each lookup table.
2. P4 control flow and P4 actions from OpenFlow DB tables
   - Identify precise encoding of P4 actions that reflects only the sequence of actions used for each table
   - Identify annotation for each OF rule to a P4 table and P4 action
3. Compile P4 program with a P4 compiler with targeted SW or HW pipeline backend support
4. Load compiler output to P4 OVS and device
   - Load P4 dataplane information to P4 supported datapath
   - Load P4 runtime information to P4 OVS
   - Load P4-to-pipeline mapping information to P4 support SDK/API
5. Configure P4 OVS OpenFlow rules with identified annotations for P4 tables and P4 actions

These steps are sufficient to start running with OpenFlow to P4 mapped pipeline.

OpenFlow to P4 mapping next steps

6. Integrate P4 into SDN controller that originates OpenFlow messages (e.g. ovn-controller)
   - Support integrating P4 runtime information into controller
   - Update table entries in OpenFlow DB with P4 table and P4 action mapping information
7. Encode the mapping info in OpenFlow messages
8. Integrate P4 into SDN controller that populates OpenFlow DB from CMS DB
9. Integrate P4 Runtime into SDN controller
10. Use P4 Runtime between P4 driven controller and P4 OVS

P4 to xdp/eBPF



Other DP related features not covered here

Some features are not covered here but are needed and considered in Arion platform:

- Flow stats collect in each Arion Wing
- Elephant flow detect and notification
- ArionWing notification and health check
- ArionWing recovering mechanism(version sync, etc)
- ….

Performance and design consideration

- Minimize Framework Overhead on Data-Path
  - Convert most complex multi-stage match-action rules into direct eBPF map lookup
  - Innovative algorithm and data structure for flow tracking
  - Xdp/eBPF logic is designed to be as simple and efficient as possible, the intelligence of Arion are in user space daemons and algorithms.
  - Flows can be handled mostly at xdp/eBPF without going through kernel network stack and user space.
- Flows route to the right ArionWing directly to avoid extra hops
- DP direct path notification capability for elephant flow or any flows
- Live DDOS mitigation easy to be injected for resiliency and robustness

References

1. Faster OVS Datapath with XDP(https://legacy.netdevconf.info/0x14/pub/papers/41/0x14-paper41-talk-paper.pdf)
2. Revisiting the Open vSwitch Dataplane Ten Years Later
3. Toward an eBPF-based clone of iptables(https://sebymiano.github.io/publication/2018-fulvio-toward/)
4. Sebastiano Miano, F. Risso, M. V. Bernal, M. Bertrone, Y. Lu (2021). A Framework for eBPF-Based Network Functions in an Era of Microservices. *IEEE Transactions on Network and Service Management*.
5. Prototyping an eBPF-based 5G Mobile Gateway(https://webthesis.biblio.polito.it/15302/1/tesi.pdf)
6. Federico Parola, Sebastiano Miano, Fulvio Risso (2020). A Proof-of-Concept 5G Mobile Gateway with eBPF. *Proceedings of the ACM SIGCOMM 2020 Conference on Posters and Demos*.
7. Alcor(https://github.com/futurewei-cloud/alcor)
8. Flowvisor
9. OpenFlow-Based Server Load Balancing Gone Wild. -- Jennifer Rexford
10. Concury: A Fast and Light-weighted Software Load Balancer(https://arxiv.org/pdf/1908.01889.pdf)
11. https://opennetworking.org/news-and-events/blog/clarifying-the-differences-between-p4-and-openflow/
12. https://github.com/p4lang/switch/blob/master/p4src/openflow.p4
13. Converting openflow to P4
14. https://www.openvswitch.org/support/ovscon2020/
15. https://docs.cilium.io/en/v1.11/concepts/ebpf/maps/
16. https://datatracker.ietf.org/doc/html/draft-ietf-nvo3-vxlan-gpe-12
17. https://datatracker.ietf.org/doc/html/rfc7348
18. More to be added …

Thank you!