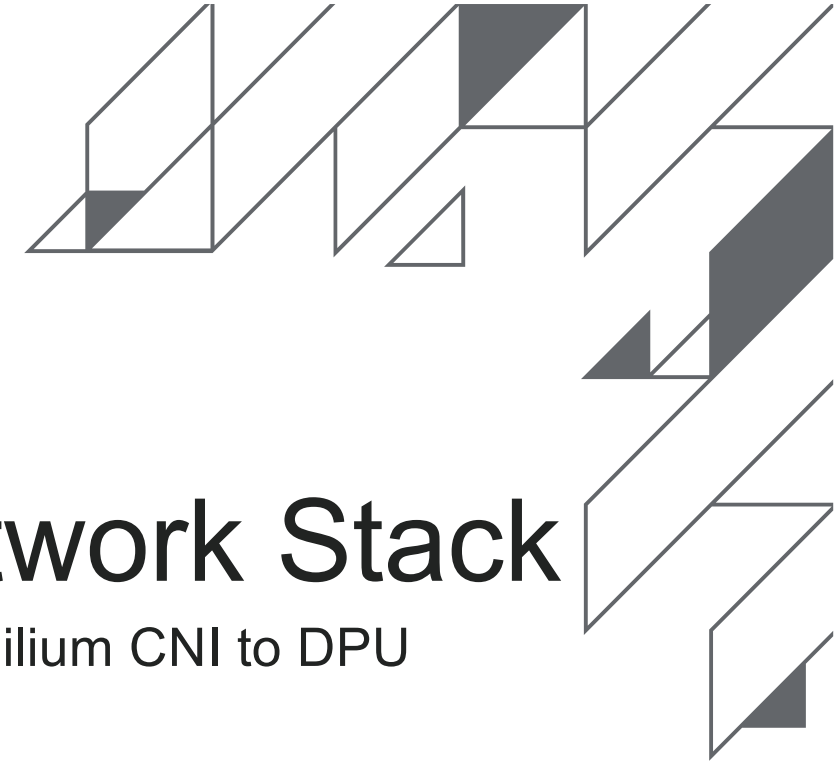


Accelerating an eBPF Network Stack

Our journey in completely offloading eBPF based Cilium CNI to DPU

Vijay Ram Inavolu
Alkama Hasan
Balakrishna Bhamidipati
Nagendra Puthane

NetDevConf 2025

AGENDA

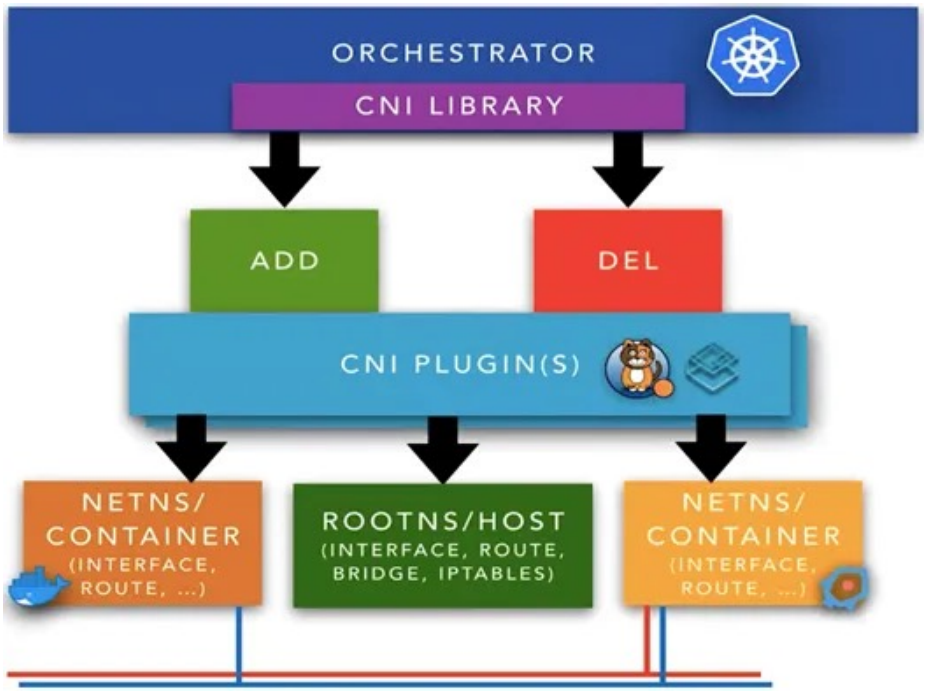
- Introduction and Background
- In Context: K8s & Cilium
- Offload Solution Goals
- Architecture
- eBPF DP Packet paths
- IPsec & Plain Traffic Offload Handling
- SW components & HW acceleration
- Demo
- Quantifying results
- Opensource details & References

WHY K8S/CILIUM OFFLOAD ?

- **Kubernetes is the backbone of modern cloud infrastructure**, ranking as the most active and rapidly growing open-source project—second only to Linux.
- **It dominates the market with a 90% share**, making it the de facto standard for container orchestration.
- **All major cloud providers offer Kubernetes-native services**, including Google's GKE, Microsoft's AKS, Amazon's EKS, and OpenShift for enterprises.
- **Cilium is a leading Kubernetes networking provider** and the default CNI (Container Network Interface) for many of these deployments.
- **It powers 65% of all Kubernetes deployments,** proving its widespread adoption.
- **Cilium's data plane is built on eBPF**, enabling high-performance, secure, and scalable networking.

Reference links at the end of the presentation

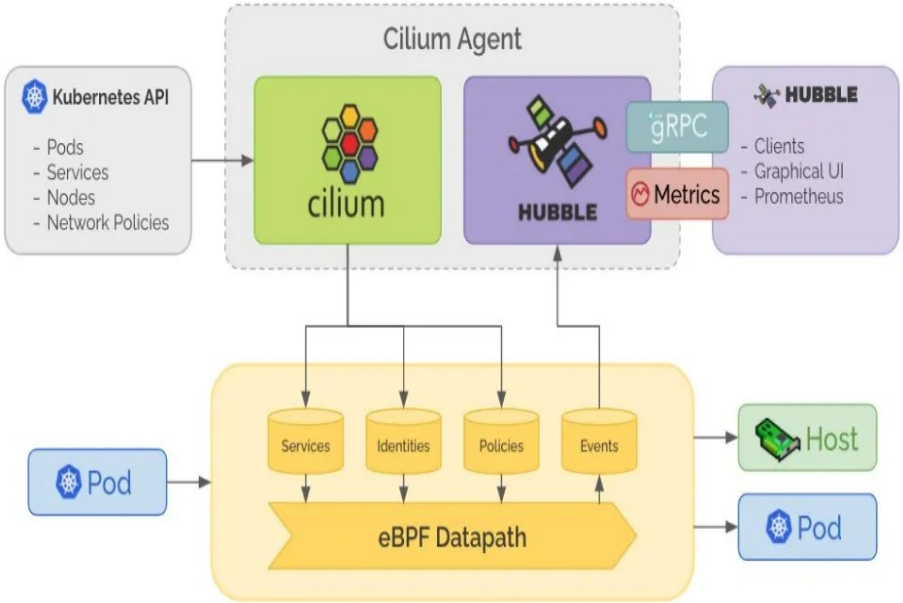
BACKGROUND: CNI



- **CNI is a Specification under CNCF**
- **Philosophy**
  - Networking separation from K8s core
  - Vendor Neutral
  - Flexibility
- **Typical Services of a CNI Plugin**
  - Pod Networking
  - IP address management
  - Services connectivity
  - Policy enforcement
  - Security: IPsec/Wire guard

and .. Dynamic + Massively scalable

CILIUM ARCHITECTURE



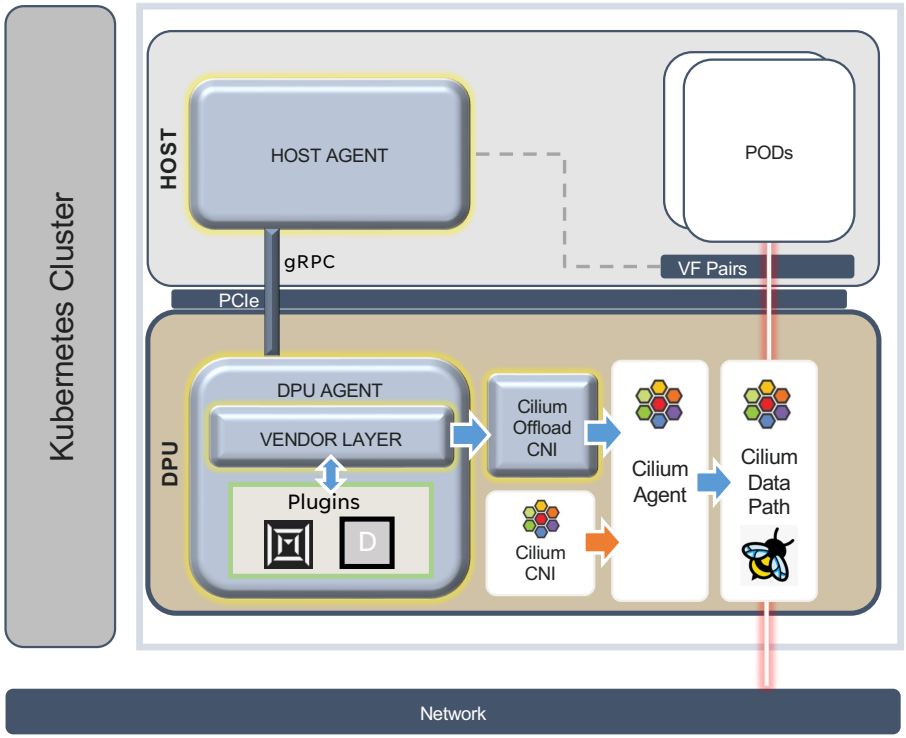
Reference: https://cilium.io/

- Cilium Agent is the control plane entity that gets deployed on every node
- Watches all Kubernetes APIs
- Programs eBPF data path
- Real time policy enforcement and load balancing
- Connects pods to node/external networking

OFFLOAD SOLUTION: GOALS & OVERVIEW

- **Minimal Impact on Existing Cilium:** Aim to keep changes to the Cilium agent and eBPF code as minimal as possible
- **Seamless User Experience:** No modifications to user-level configurations
- **Vendor-Neutral Architecture:** Open-source design that avoids locking in to any single DPU vendor
- **DPUs as Cluster Nodes:** Optionally treat DPUs themselves as Kubernetes nodes
- **Flexible Deployment:** Support mixed environments where some nodes have DPUs and others do not

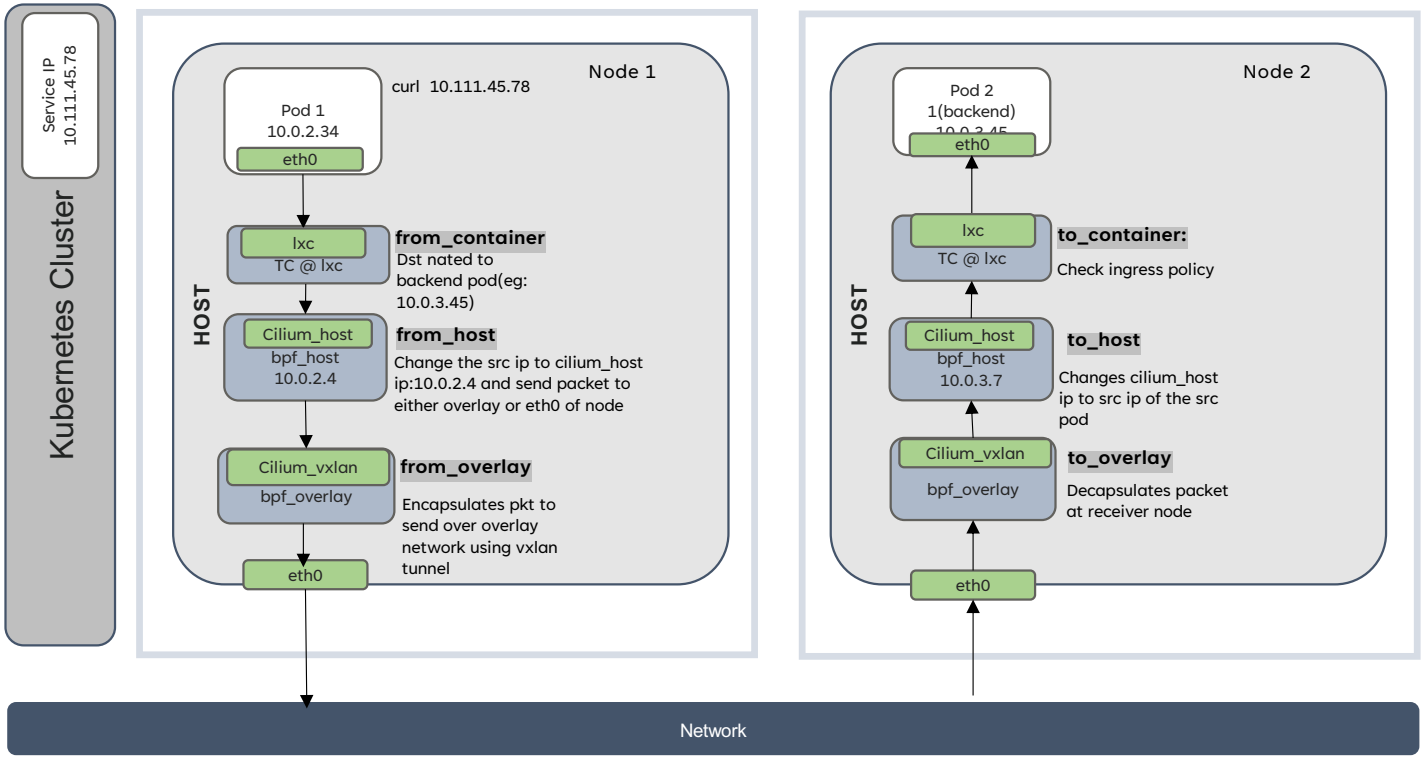
SOLUTION ARCHITECTURE



- **HostAgent :** On New Pod creation, moves VF as primary network to pod NS and configures it. Sends gRPC request to DPU Agent, to configure DPU side VF. Similarly Delete
- **DPU Agent**(cni-offload-agent): Listens and serves CNI requests from host. Call Cilium-Offload-CNI binary to create cilium endpoint with DPU side VF for host pod. Allocates IPAM for host pod
- **Plugins**: An Abstraction layer accommodate multiple DPU vendors/HW.
- **Vendor Layer**: An Abstraction layer to select multiple CNI
- **Cilium Offload CNI binary :** Creates cilium endpoints on DPU side for the pod on the host.

PACKET PATH IN DEFAULT CILIUM

Pod 1 trying to Access services through wget



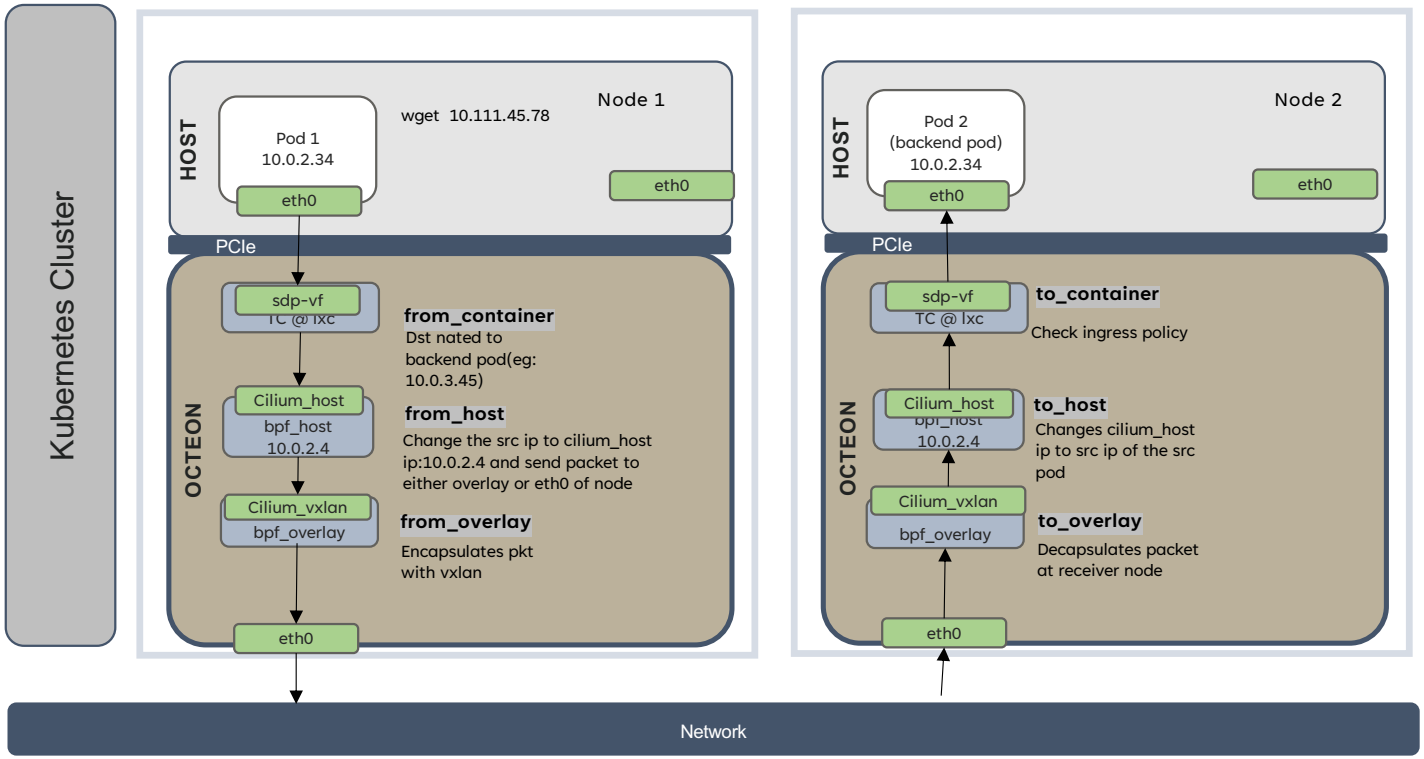
Green boxes represent linux network interfaces

blue boxes are Cilium eBPF hooks

- from_container:
  1. service load balancing(backend pod)
  2. Perform DNAT
- to_container:ingress policy checking for the pod and allow/deny the packet
- from_host: changes src ip to cilium_host src ip
- to_host : changes cilium_host ip to src ip of the src pod
- from_overlay: encapsulates pkt to send over overlay network using vxlan tunnel
- to_overlay: decapsulates packet at receiver side node

PACKET PATH WITH OFFLOADED CILIUM

Pod 1 trying to Access services through wget



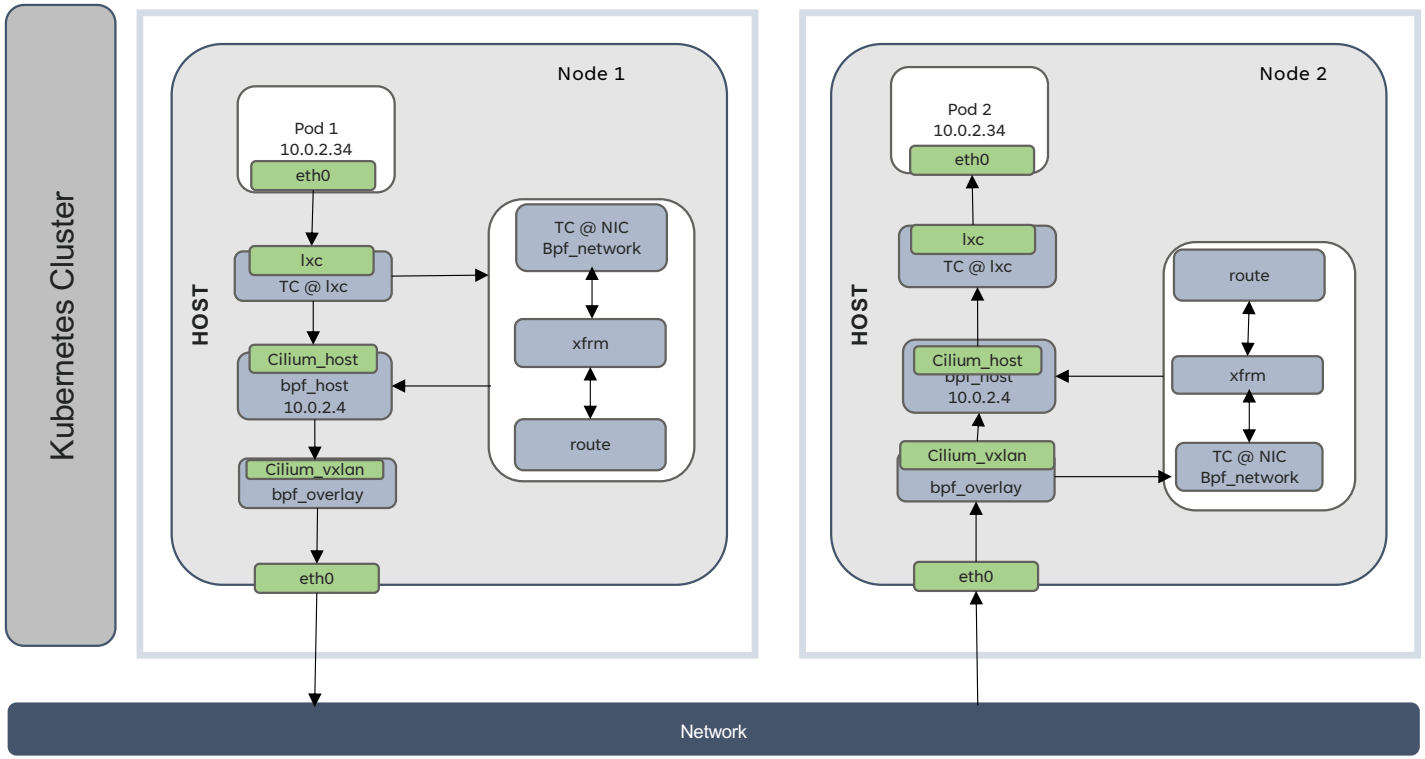
Green boxes represent linux network interfaces

Grey boxes are Cilium eBPF hooks

- Host and DPU are connected via PCIe
- Pod is assigned SDP VF
- DPU end of SDP VF sits in DPU Cilium DP
- No bpf hooks are present in host as CA is not running on host
- SocketLB.HostNameSpaceOnly=true is set to enable service lookup in the tc bpf program instead in socketLB.

PACKET PATH IN DEFAULT CILIUM WITH IPSEC ENCRYPTION

Pod 1 to pod 2 traffic is encrypted using IPSec policy



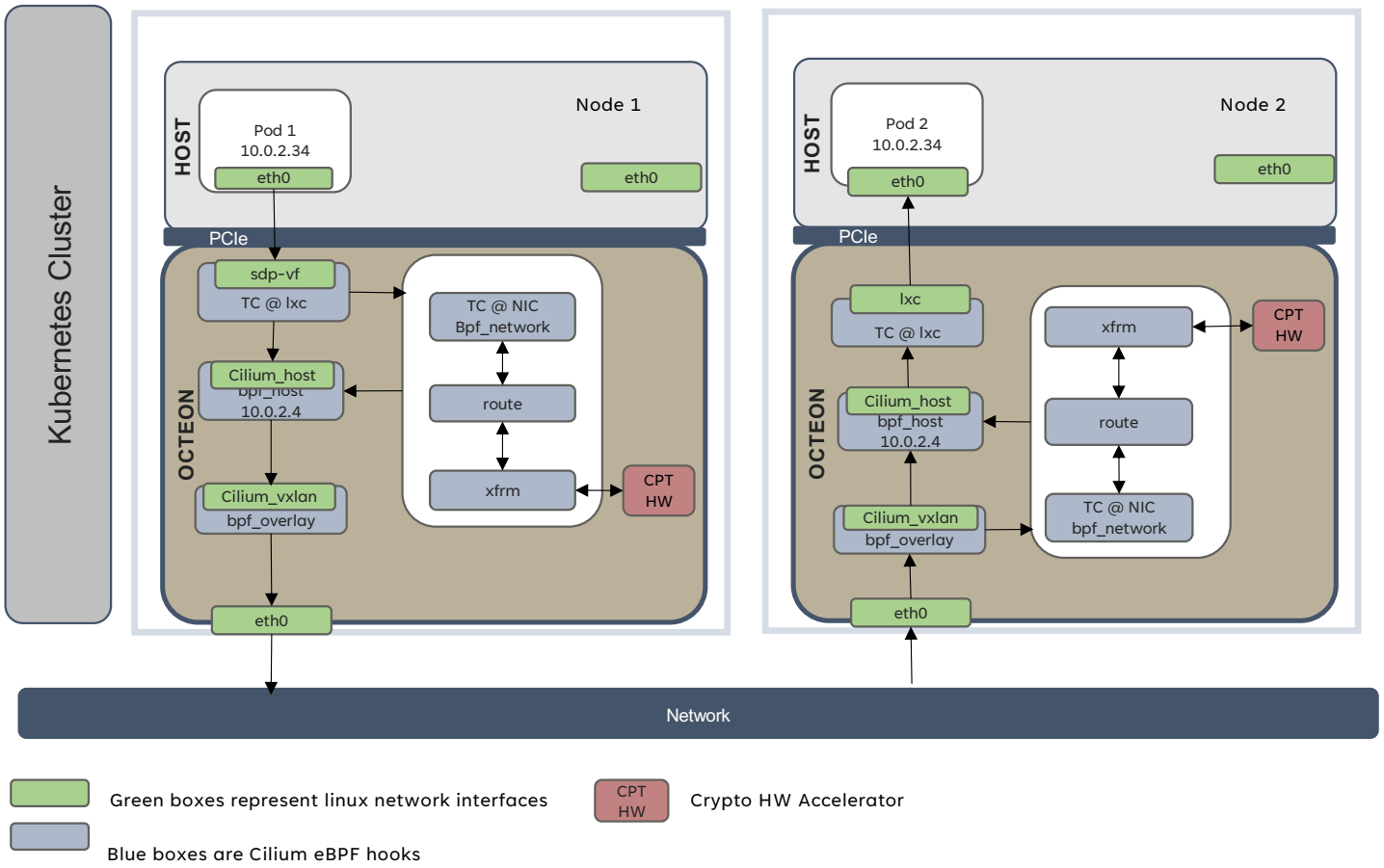
- In egress bpf_network encrypts pkt using xfrm framework.
- In ingress, bpf_network decrypts the ipsec pkt using xfrm framework.

Green boxes represent linux network interfaces

Blue boxes are Cilium eBPF hooks

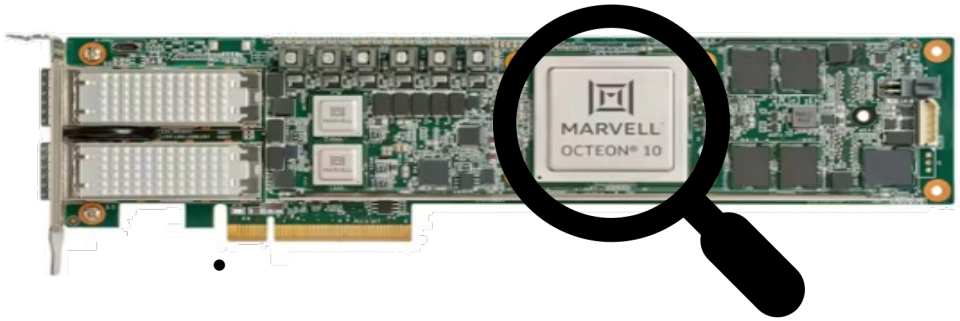
PACKET PATH IN OFFLOADED CILIUM WITH IPSEC ENCRYPTION

Pod 1 to Pod 2 traffic is encrypted using IPSec

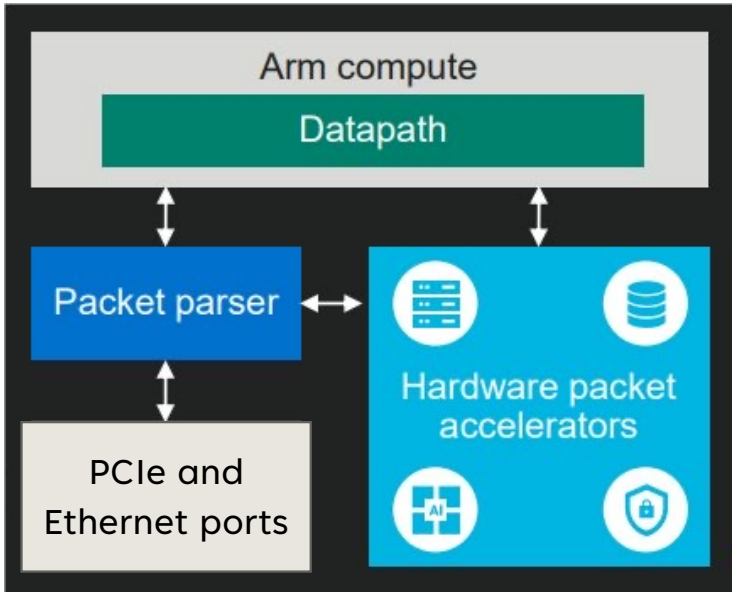


- Cilium can exploit DPU HW capabilities such as CPT HW using xfrm
- No extra configuration in IPSec SAs and policies
- Linux xfrm framework can detect underlying HW support and perform IPSec offload processing
- Cilium transparent encryption is offloaded using xfrm + CPT HW

OCTEON DPU: IPSEC ACCELERATION AND MORE



OCTEON DPU
24 ARM Neoverse N2 cores
Crypto, IPsec, TLS Accelerators
Packet Parsing & Classification
Flow Ordering and Sync
Traffic Management and QoS
Inline AI/ML Inferencing
PCIe 5.0 and 56G SerDes
Virtualized Accelerators



EBPF FROM X86 TO ARM: **ARCHITECTURE TRANSPARENCY**

Unified eBPF Model

- Cilium logic remains architecture-independent.
- No changes in logic, as end-points visibility is same between host/DPU

Docker-based eBPF Compilation

- Utilizes Docker containers to compile eBPF code.
- Container architecture matches underlying hardware (x86 or ARM).

No change for eBPF Bytecode Generation

- eBPF bytecode compilation tailored to host architecture.
- Bytecode injected via standard Linux kernel BPF hooks.

Simplified Management

- eBPF compilation and injection managed transparently by Cilium-agent.
- Architecture-specific details abstracted away, treated as a "black box."

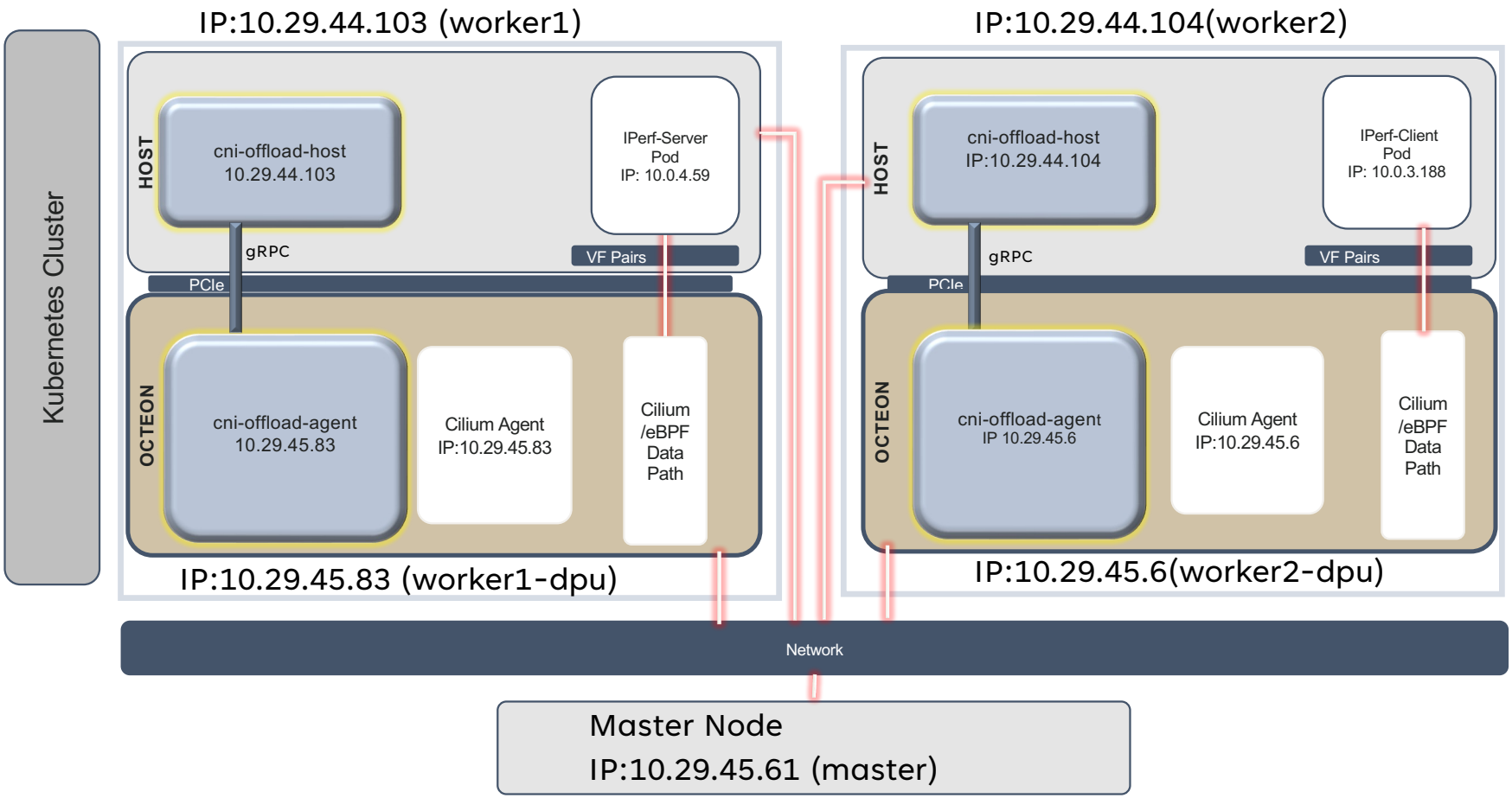
NEW SW COMPONENTS

Component Name	Location	Details
Host Agent	Host	CRI Calls Host Agent during Pod creation on host. Host Agent with its CNI handles CNI Add/Del/Check, sends down Pod Spec and Interface details to DPU Agent. Parses IPAM allocated from DPU CA , and configures Pod linux interface.
DPU Agent	DPU	Serves gRPC requests from Host. Invokes Offload Cilium CNI with details such as CNI Spec, Pod details and Interface Details as CNI Request. From the CNI result, parses the allocated IPAM, sends back to the Host Agent.
Offload Cilium CNI	DPU	Called from DPU Agent to handle CNI Add/del/check request for the pod launched on Host. On CNI Add request, it will post endpointCreateRequest to CA to create endpoint for host pod. It will also post IPAM request to allocate IPAM for host pod.

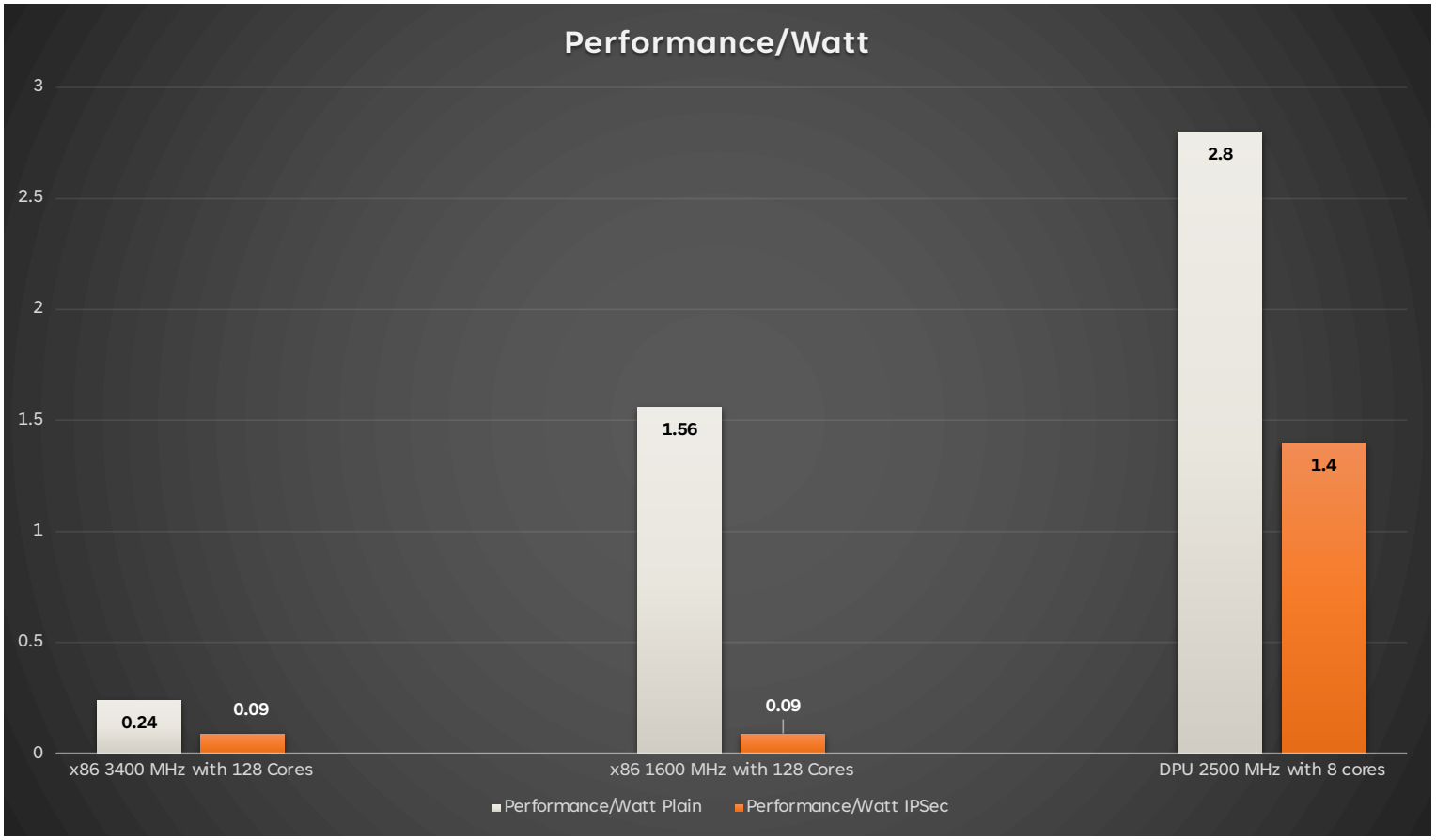
CHANGES TO CILIUM

Functionality	Details	Files modified
A DPU to know and handle pod launch events on it's host	CA should create endpoint for each pod running on host. It needs pod identities. CA runs on DPU, is not aware of K8s events on host, and lacks pod identities. To watch host node events, an extra listener, watchRemotePods() is added. It watches k8s.PodResource using spec.nodeName=hostName.	Pkg/cilium/daemon/k8s/resources.go Pkg/cilium/pkg/k8s/watchers/pod.go pkg/cilium/pkg/k8s/watchers/watcher.go
Nodes to use DPU IP to reach to host	Pod with host VF runs in Host. Its lxc interface (DPU VF) is managed by CA in DPU. Hence, Pod traffic should be forwarded to DPU instead of host. endpointUpdated() Upserts ipcache on endpoint creation. This should set nodeIP to DPU address.	pkg/k8s/watchers/cilium_endpoint.go

DEMO SETUP


IP:10.29.44.103 (worker1)
IP:10.29.44.104(worker2)
Kubernetes Cluster
HOST
cni-offload-host
10.29.44.103
IPerf-Server
Pod
IP: 10.0.4.59
gRPC
VF Pairs
PCIe
OCTEON
cni-offload-agent
10.29.45.83
Cilium Agent
IP:10.29.45.83
Cilium
/eBPF
Data
Path
IP:10.29.45.83 (worker1-dpu)
HOST
cni-offload-host
IP:10.29.44.104
IPerf-Client
Pod
IP: 10.0.3.188
gRPC
VF Pairs
PCIe
OCTEON
cni-offload-agent
IP 10.29.45.6
Cilium Agent
IP:10.29.45.6
Cilium
/eBPF
Data
Path
IP:10.29.45.6(worker2-dpu)
Network
Master Node
IP:10.29.45.61 (master)

QUANTIFYING EBPF DP OFFLOAD: OPEX SAVINGS

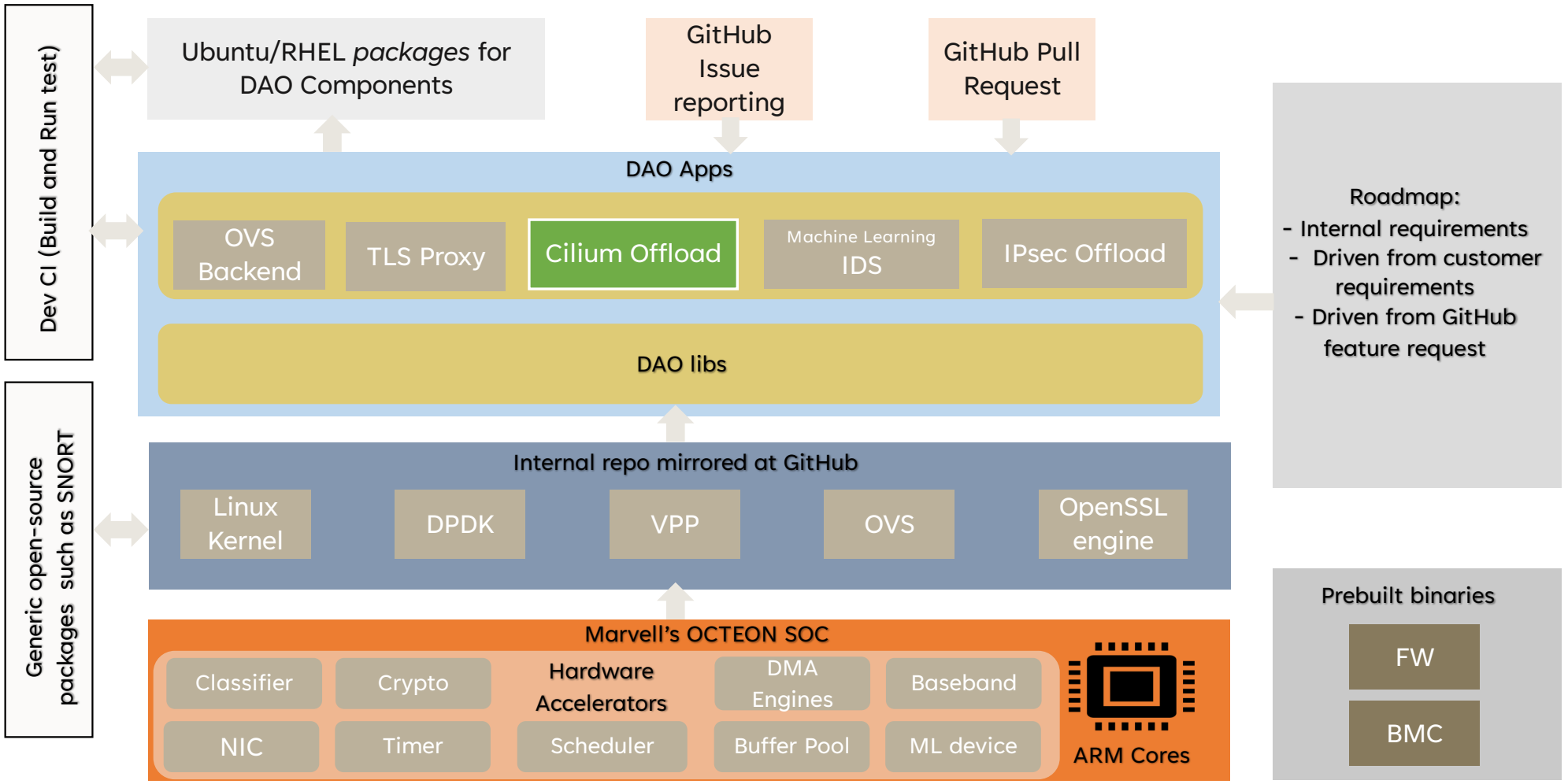


POWER DATA CALCULATIONS

- Host power is measured at idle and at load conditions using server's BMC
- For offload and non-offload cases iPerf traffic generator pod is running on host, only data path is moved.
- This is accounted in calculation: Data path power on host = Total power – iPerf power
- Since application-level power can't be differentiated using BMC, the ratio of iPerf vs DP is derived from CPU utilization of both of these.
- On DPU power is measured at load for cilium data path using Linux kernel hardware monitor
  - cat /sys/class/hwmon/hwmon0/power1_input
- Constant test parameters for all measurements: Streams 32, MTU 1500, IPsec Algo AES-GCM 128, host cpus 128, dpu cpus 24, power in watts
- Power measurements rely on 3rd party tooling & deriving estimates from CPU utilization, so although we've taken steps to minimize inaccuracies, these measurements inherently carry a margin of error that should be acknowledged.

Device	Traffic Type	Gbps	CPU %	iPerf CPU %	Power @idle	Power @test	Total Power	In % CPU Ratio: iPerf Power	Data Plane Power	Performance/Watt
Host @1600 MHz	Plain	25	3	34	210	234	24	8	16	1.56
Host @1600 MHz	IPsec	20	14.5	7	210	434	224	15	209	0.09
Host @3400 MHz	Plain	27	1.5	6	290	412	122	8	114	0.24
Host @3400 MHz	IPsec	20	13.5	6	290	514	224	13	211	0.09
DPU @2500 MHz	Plain	20	44.3	N/A	10	17	7	N/A	7	2.8
DPU @2500 MHz	IPsec	20	72.6	N/A	10	24	14	N/A	14	1.4

DAO SOFTWARE SUITE



REFERENCES

- Software Opensource Link (DAO)

https://github.com/MarvellEmbeddedProcessors/dao

- Hardware:

https://www.marvell.com/products/data-processing-units.html

https://www.hawkeyetech.com.tw/products/hardware-acceleration/

- Cloud SW and References

https://kubernetes.io

https://cilium.io

https://www.cncf.io/wp-content/uploads/2020/12/CNCF_Survey_Report_2020.pdf

https://enlyft.com/tech/products/kubernetes
https://docs.cilium.io/en/stable/overview/component-overview/

Q N A

MARVELL

THANK YOU