
KubeCon
CloudNativeCon
North America 2024

KubeCon | CloudNativeCon

North America 2024

Thousands of Gamers, One Kubernetes Network

- Achieving End-To-End Quality of Service

KubeCon CloudNativeCon
North America 2024

Surya Seetharaman
Principal Software Engineer @ Red Hat
OVN-Kubernetes Maintainer
@tssurya

Girish Moodalbail
Distinguished Engineer @ NVIDIA
OVN-Kubernetes Maintainer
@girishmg


2
COUNTER STRIKE


DOTA 2


NARAKA
BLADEPOINT
SEASON 4: AURORA


THRONE AND LIBERTY


APEX
LEGENDS


BLACK MYTH
WUKONG


STARDEW VALLEY


RUST


WAR THUNDER
DANCE OF DRAGONS


SATISFACTORY


Baldur's Gate


MONSTER HUNTER
WORLD


WARFRAME
KOUMEI & THE FIVE FATES


CALL OF DUTY


TEAM FORTRESS 2


SID MEIER'S
CIVILIZATION VI

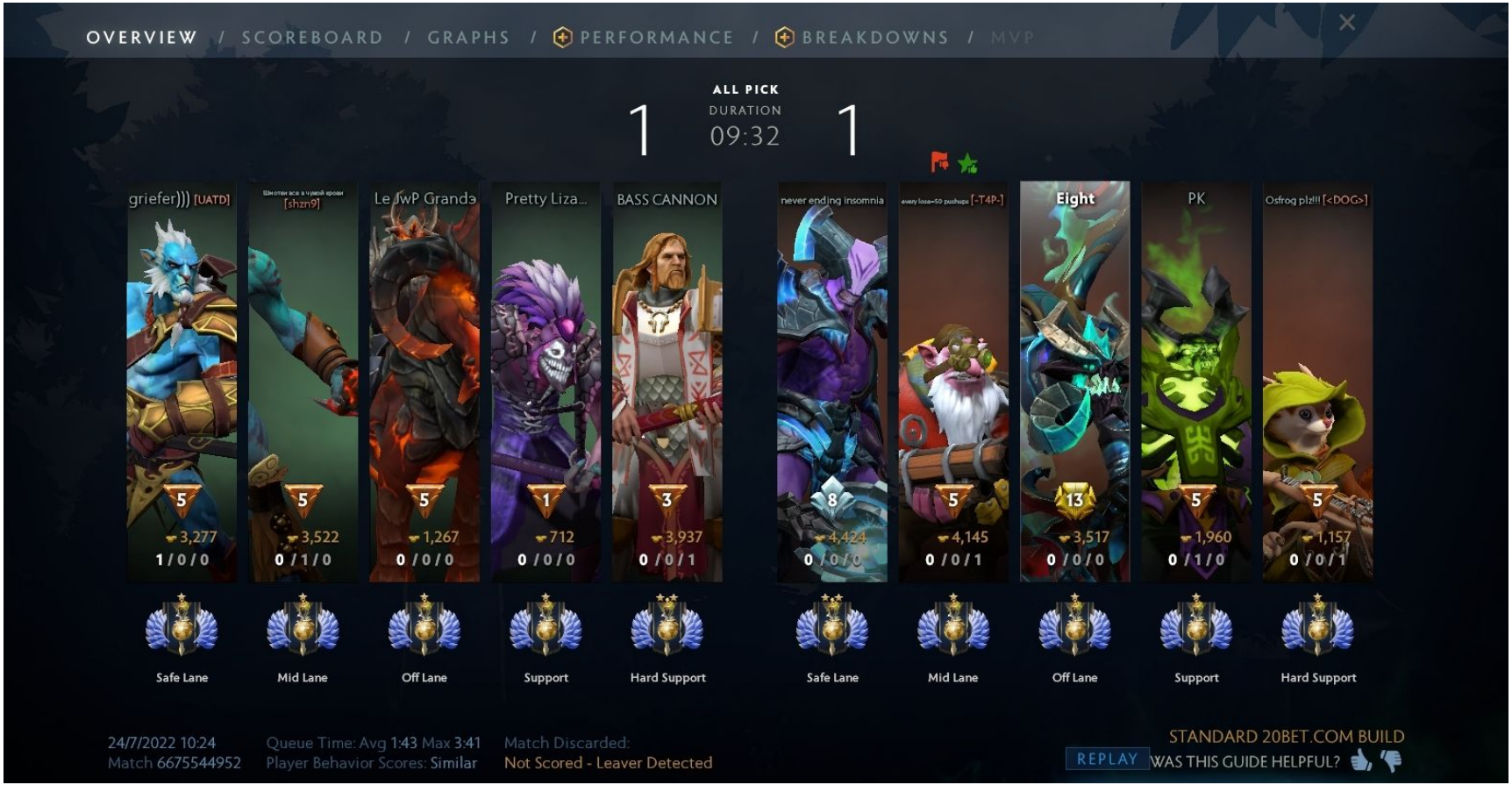

ONCE
HUMAN


DESTINY 2


TOM CLANCY'S
RAINBOW SIX SIEGE

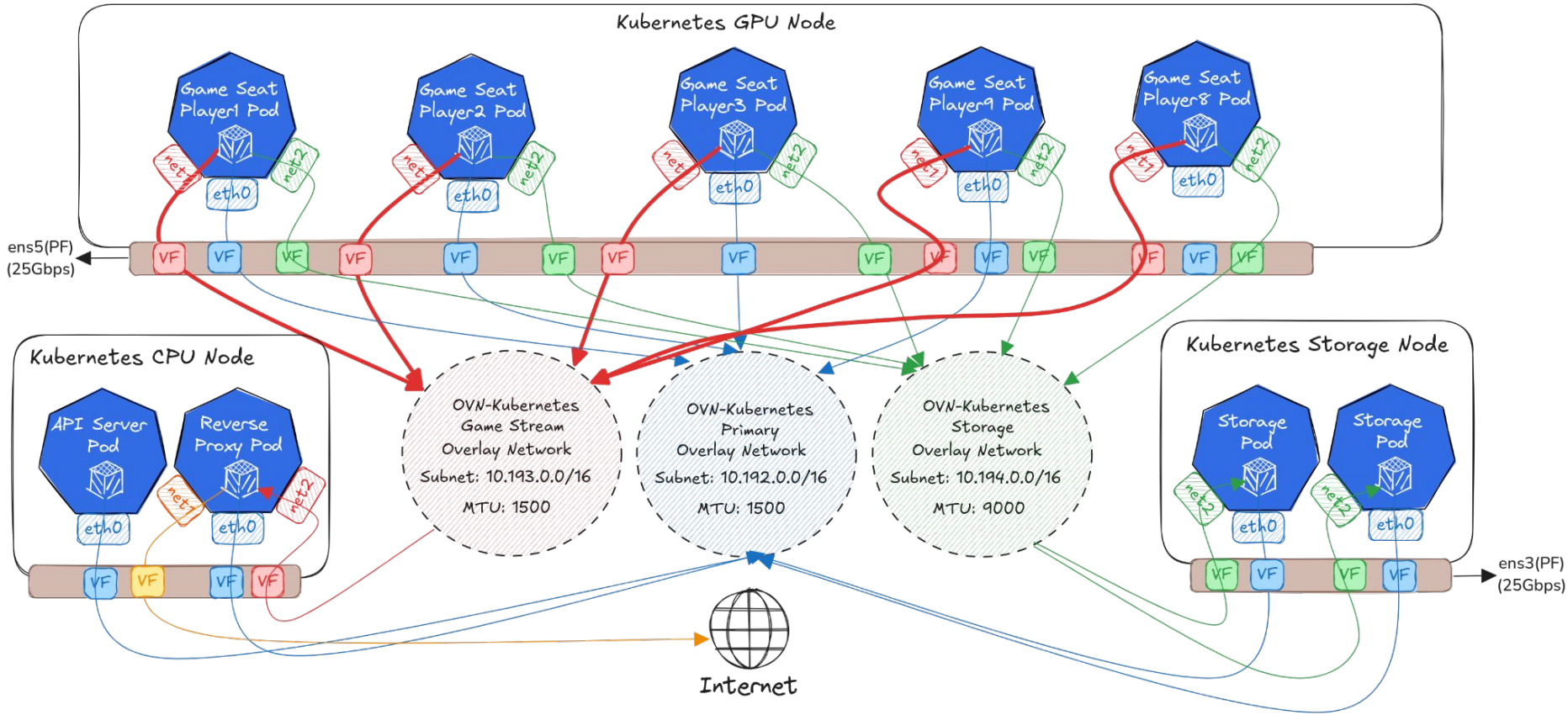

HEARTS OF IRON IV

Ongoing Game 1



Ongoing Game 2

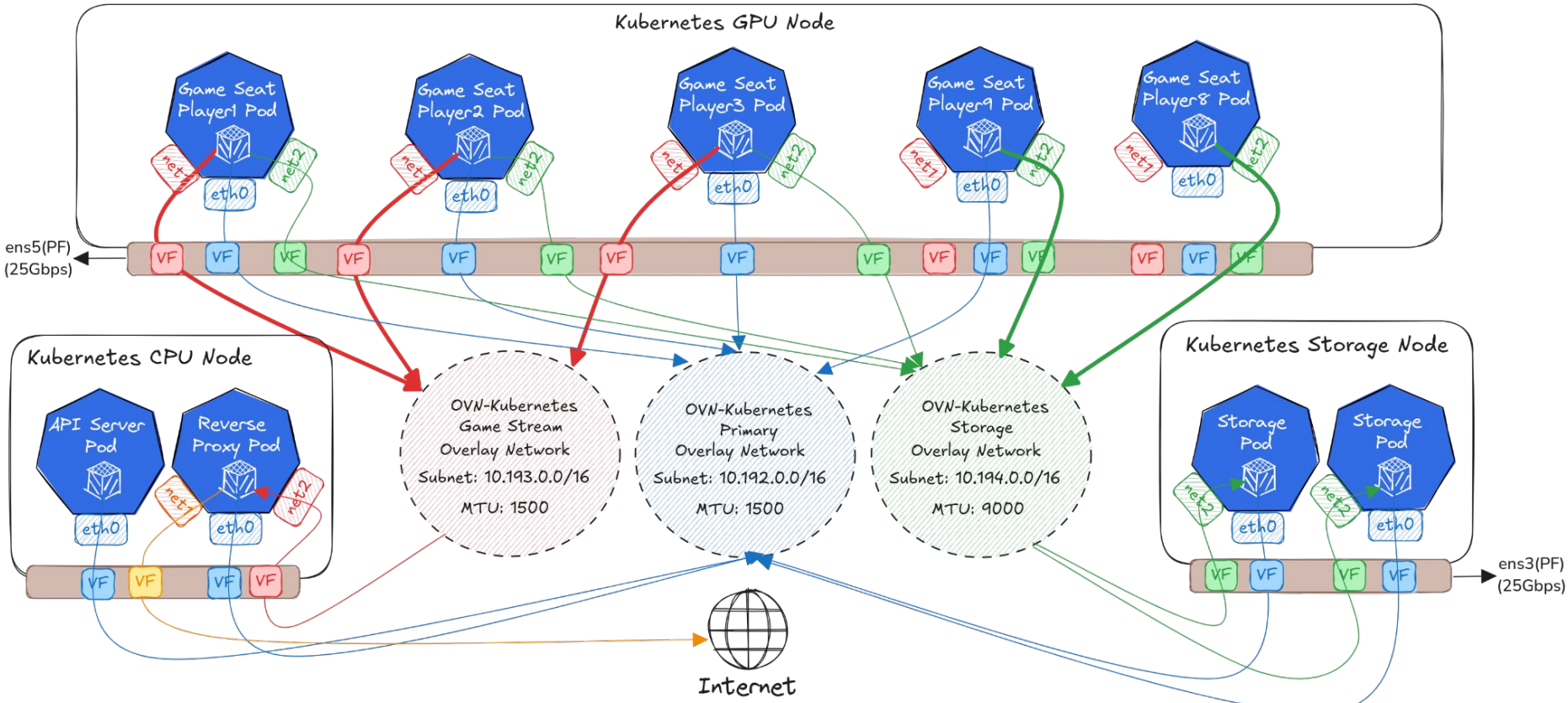
Ongoing Game 1 (P1, P2, P3), Ongoing Game 2 (P8, P9)



Uplink and Network for Node and TOR all have 25G capacity

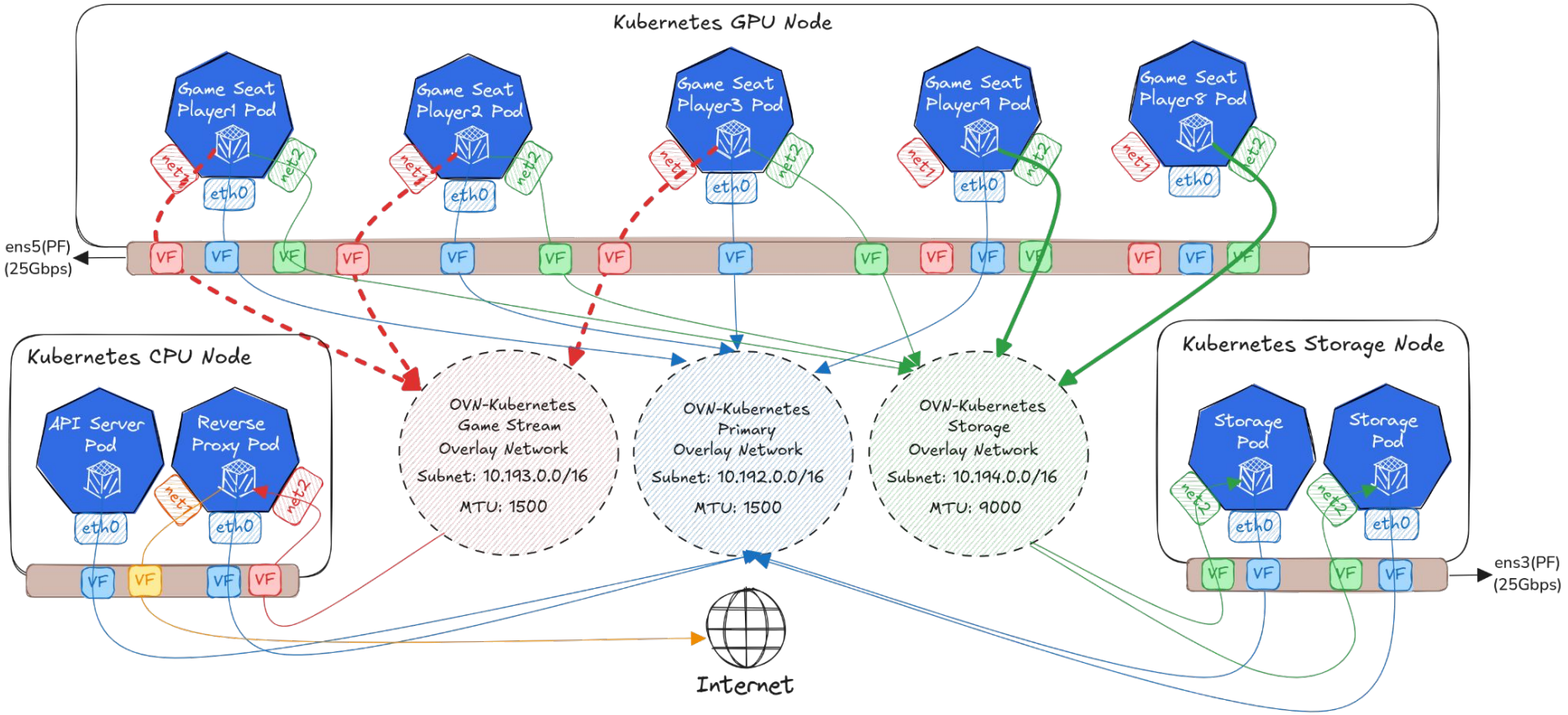
Game2 just ended

Game 2 just ended: Players 8, 9 about to logout



Uplink and Network for Node and TOR all have 25G capacity

Problem Statement: Bad Gaming Experience



Uplink and Network for Node and TOR all have 25G capacity

Gaming Network Requirements

- It is absolutely essential that we provide a Network with
  - Minimal Network Latency
  - Minimum Jitter
  - Maximum throughput

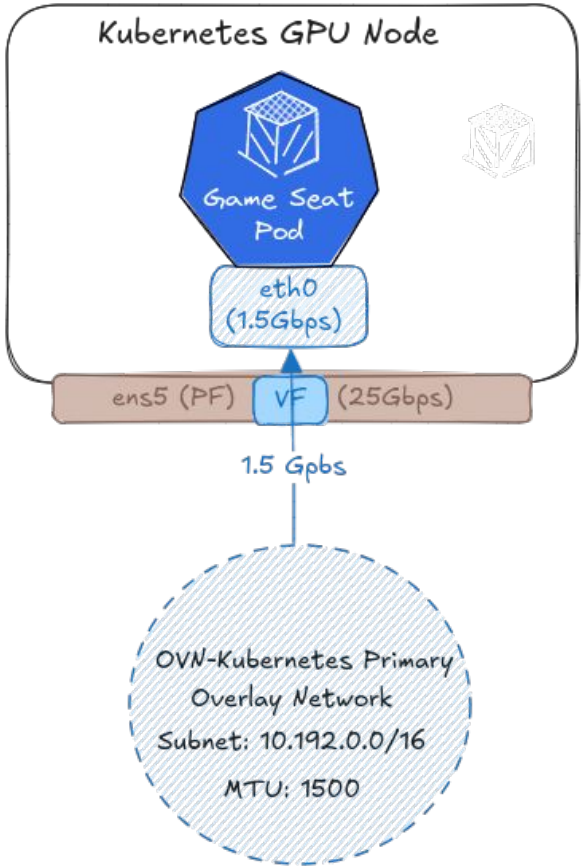
  at any given instance of time to ensure uninterrupted gameplay

- It is important to have bandwidth controls in place to shield players on the same node from “noisy neighbors”
  - Being able to fine tune bandwidth controls such that when same uplink is shared the storage traffic should not effect streaming traffic

Terminologies

Terminologies: Ingress Bandwidth

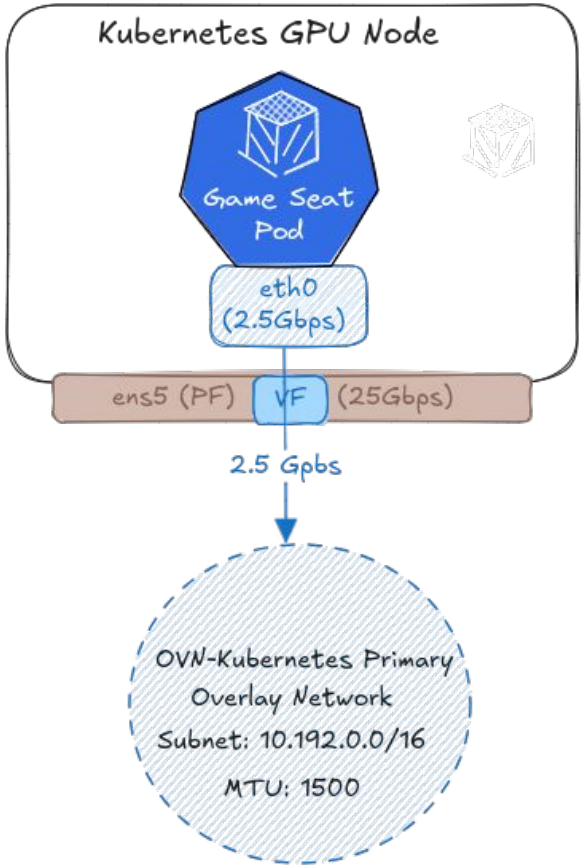
- Limiting traffic towards an application/port
- Usually applied last at destination just before packets reach the application - wasteful since most of the pipeline has already been traversed



Uplink and Network for Node and TOR all have 25G capacity

Terminologies: Egress Bandwidth

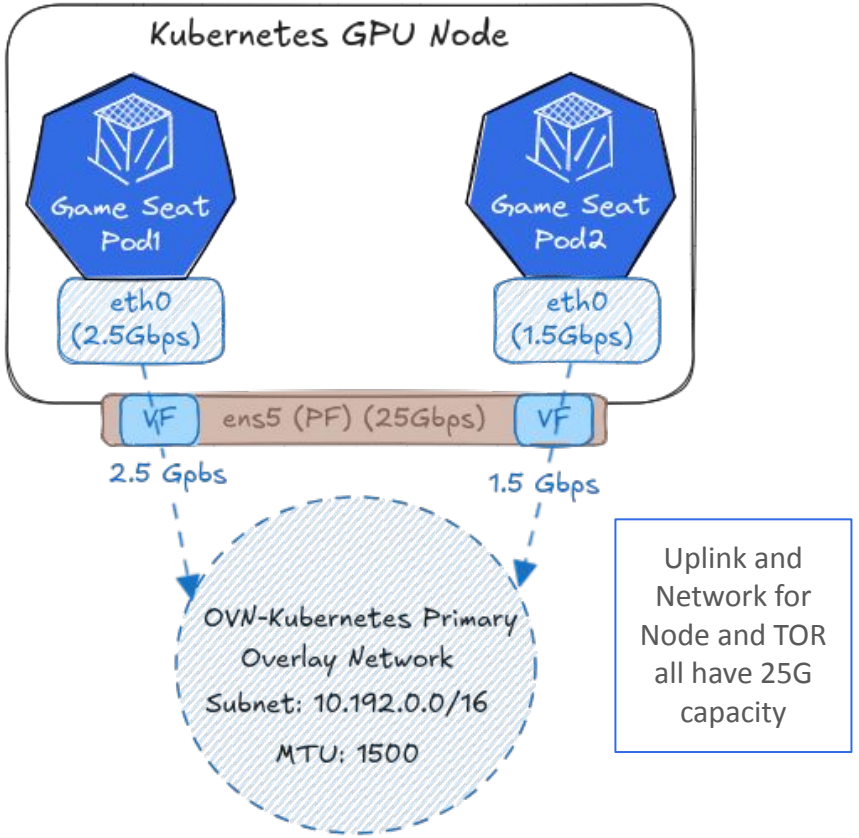
- Limiting traffic coming from an application/port to a specified rate
- Usually applied first at source just before packets leave the pod - will be the focus of our talk
- FYI: For ingress; we rely on egress bandwidth controls on switch port side



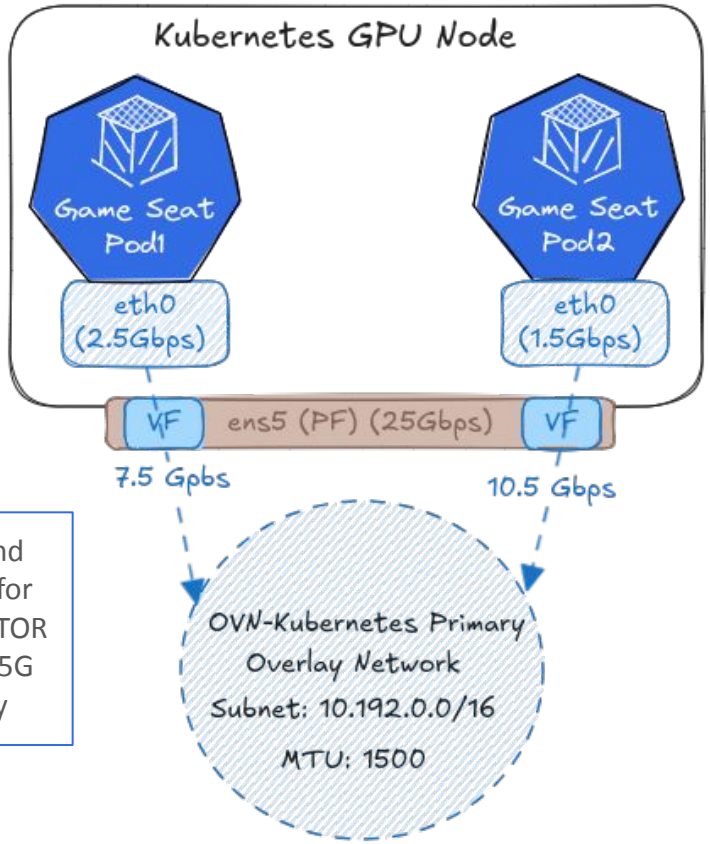
Uplink and Network for Node and TOR all have 25G capacity

Terminologies: Maximum Bandwidth

- Maximum Limit: The maximum bandwidth an application/port can use. Even if no one is using the rest of the uplink, this application cannot exceed its limit.



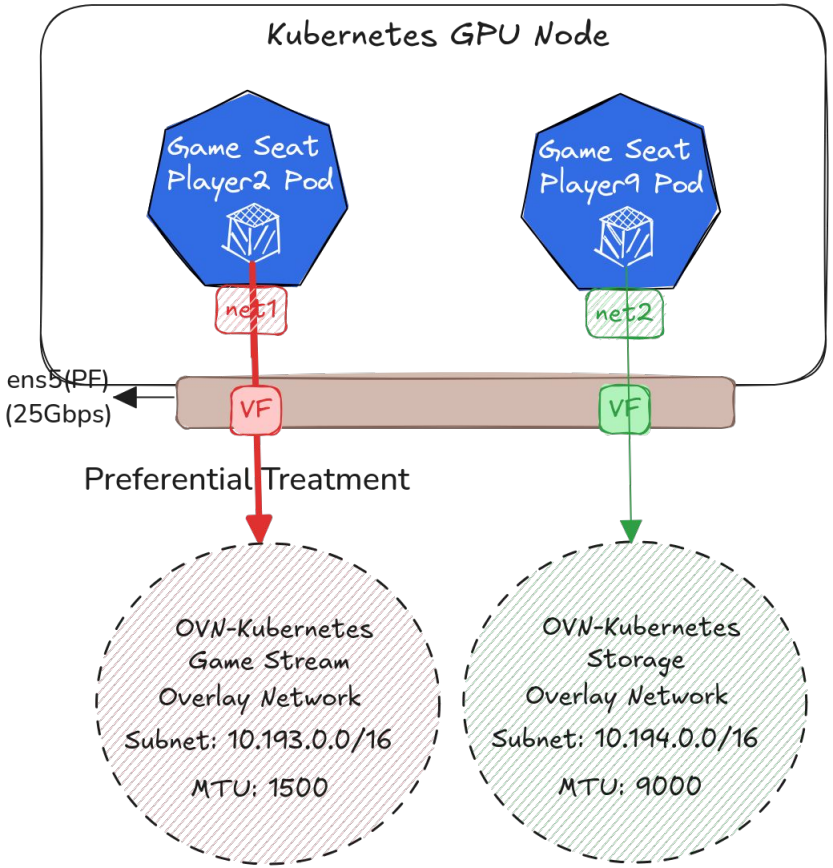
Terminologies: Guaranteed(Minimum) Bandwidth



Uplink and Network for Node and TOR all have 25G capacity

- Guaranteed(Minimum) Limit: The guaranteed minimum this application/port can use. If no one is using the rest of the uplink bandwidth, this application can use it until another application requires those resources in which case the first application starts to get throttled to its minimum limit.

Terminologies: Quality of Service (QoS)



- QoS: A network management concept that controls and prioritizes network traffic to ensure better performance for critical applications.
  - Traffic prioritization
  - Traffic classification
  - Traffic queuing

Current Status in Kubernetes Ecosystem

Kubernetes Ingress Bandwidth Annotation

- Limiting traffic towards a Pod by shaping queued packets
- In Kubernetes it is available on a pod level via **kubernetes.io/ingress-bandwidth** annotation
- Experimental Feature; requires adding bandwidth plugin to the CNI configuration file

```yaml
apiVersion: v1
kind: Pod
metadata:
 annotations:
   kubernetes.io/ingress-bandwidth: 1.5G
```

```
{
  "cniVersion": "0.3.1",
  "name": "mynet",
  "plugins": [
    {
      "name": "slowdown",
      "type": "bandwidth",
      "ingressRate": 1500000000,
      "ingressBurst": 1300000000,
    }
  ]
}
```

Kubernetes Egress Bandwidth Annotation

- Limiting traffic coming from a Pod by policing and dropping packets in excess of the configured rate
- In Kubernetes it is available on a pod level via **kubernetes.io/egress-bandwidth** annotation
- Experimental Feature; requires adding bandwidth plugin to the CNI configuration file

```yaml
apiVersion: v1
kind: Pod
metadata:
 annotations:
   kubernetes.io/egress-bandwidth: 2.5G
```

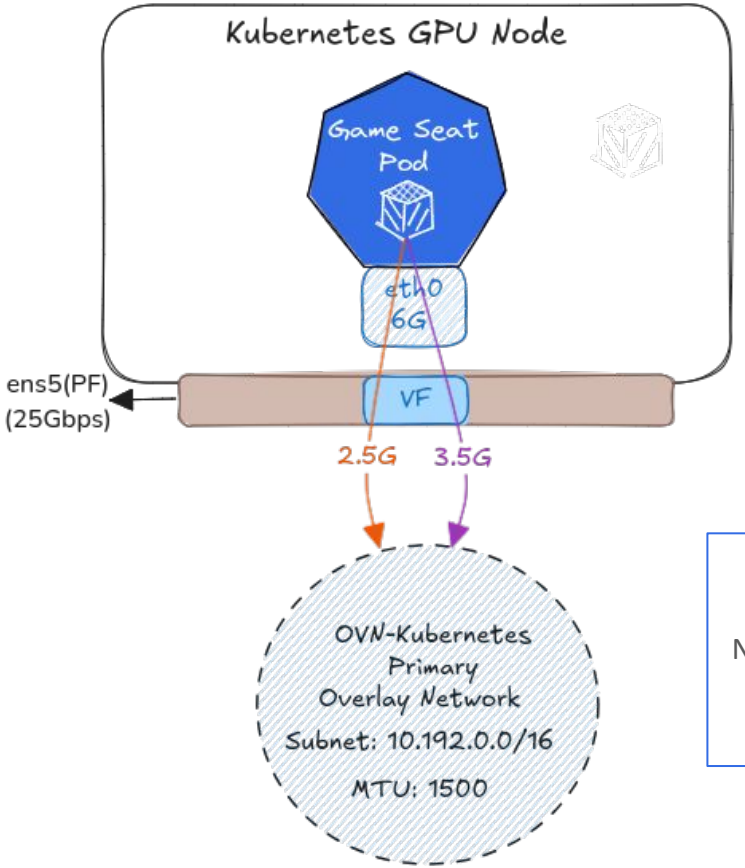
```
{
  "cniVersion": "0.3.1",
  "name": "mynet",
  "plugins": [
    {
      "name": "slowdown",
      "type": "bandwidth",
      "egressRate": 2500000000,
      "egressBurst": 2600000000,
    }
  ]
}
```

Limitations of Kubernetes Bandwidth Annotations

- API doesn’t have a way to specify guaranteed bandwidth

Limitations of Kubernetes Bandwidth Annotations

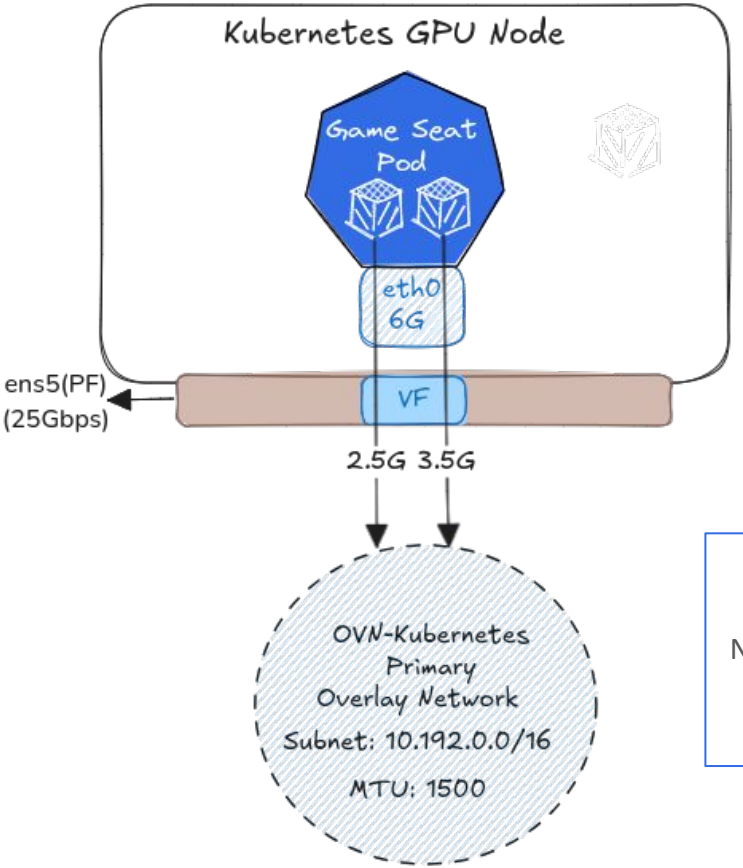
- API doesn't have a way to specify guaranteed bandwidth
- Limited to pod level
  - Not tunable for different types of traffic coming from same application



Uplink and Network for Node and TOR all have 25G capacity

Limitations of Kubernetes Bandwidth Annotations

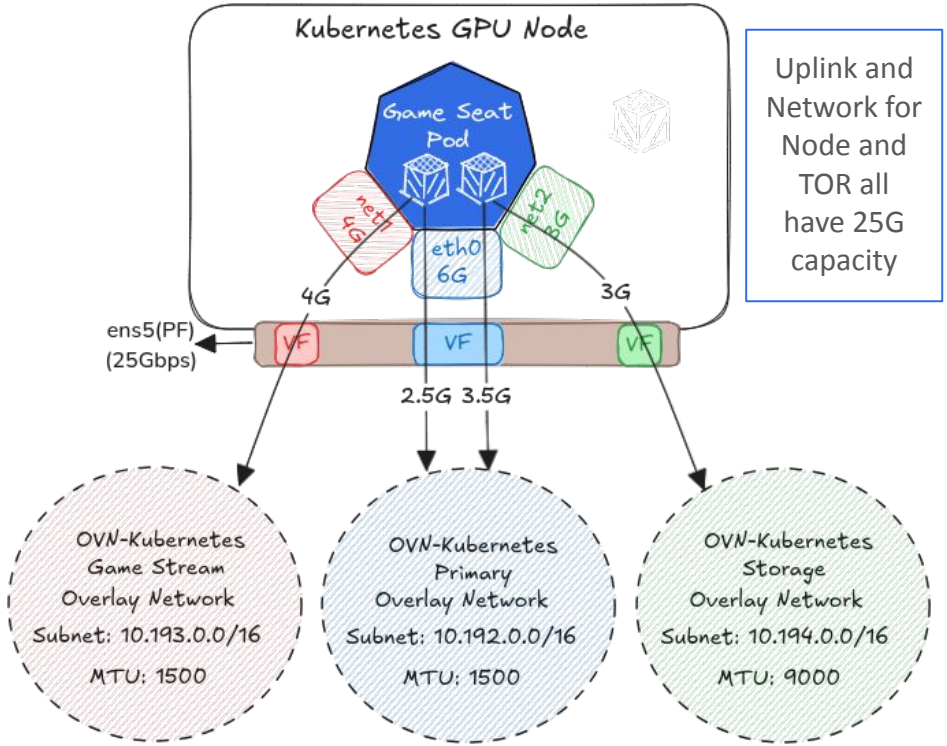
- API doesn't have a way to specify guaranteed bandwidth
- Limited to pod level
  - Not tunable for different types of traffic coming from same application
  - Not tunable for different applications within the pod



Uplink and Network for Node and TOR all have 25G capacity

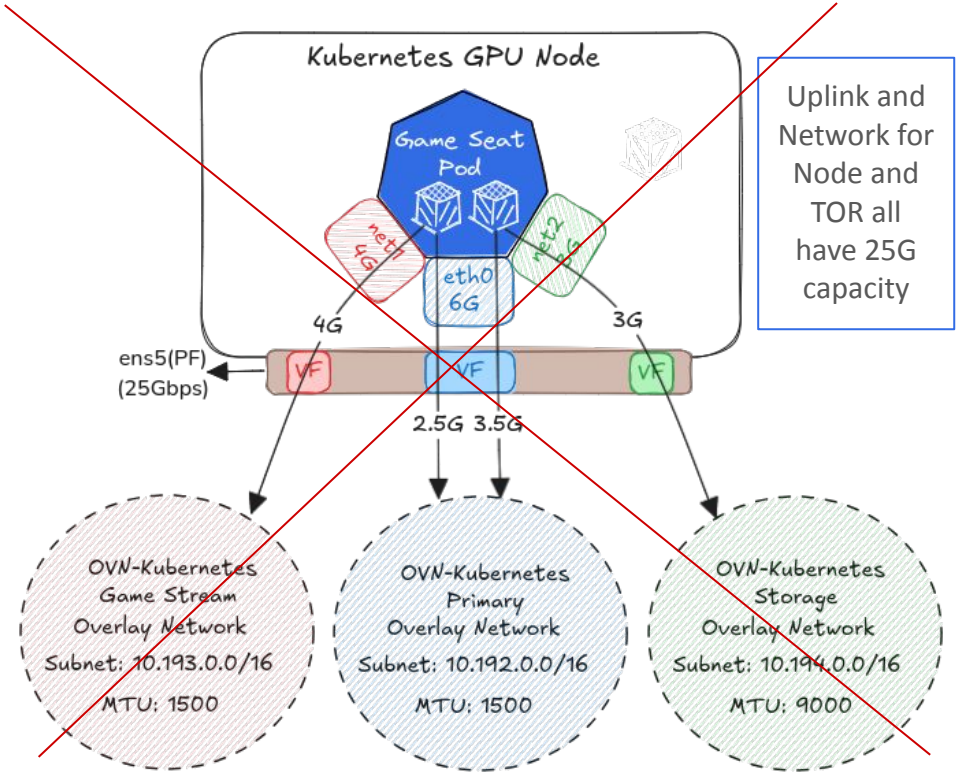
Limitations of Kubernetes Bandwidth Annotations

- API doesn't have a way to specify guaranteed bandwidth
- Limited to pod level
  - Not tunable for different types of traffic coming from same application
  - Not tunable for different applications within the pod
  - Not tunable for secondary pod interfaces



Limitations of Kubernetes Bandwidth Annotations

- API doesn't have a way to specify guaranteed bandwidth
- Limited to pod level
  - Not tunable for different types of traffic coming from same application
  - Not tunable for different applications within the pod
  - Not tunable for secondary pod interfaces



Packet parameter tuned traffic shaping is a must - but this cannot be achieved today using Kubernetes Annotations

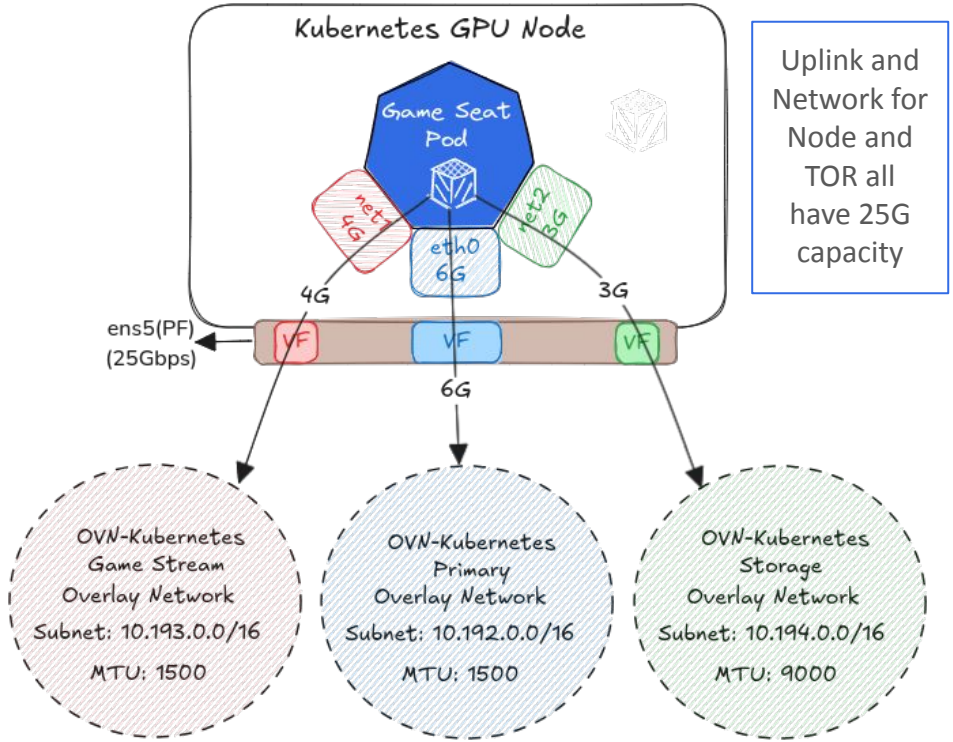
Multus Annotations on Pod Interface level

- We can achieve tunable bandwidth for secondary pod interfaces using K8s NPWG's [k8s.v1.cni.cncf.io/networks] annotation.

```
apiVersion: v1
kind: Pod
metadata:
 annotations:
     k8s.v1.cni.cncf.io/networks: |
      [
        {
          "name":"game-stream",
          "bandwidth": {
            // Limit ingress to 2Kbps
            "ingressRate": 2048,
            // Allow small ingress bursts of 300b
            "ingressBurst": 300,
            // Limit egress to 8000bps
            "egressRate": 8000,
            // Allow small egress bursts of 200b
            "egressBurst": 200
          }
        }
      ]
```

Multus Annotations on Pod Interface level

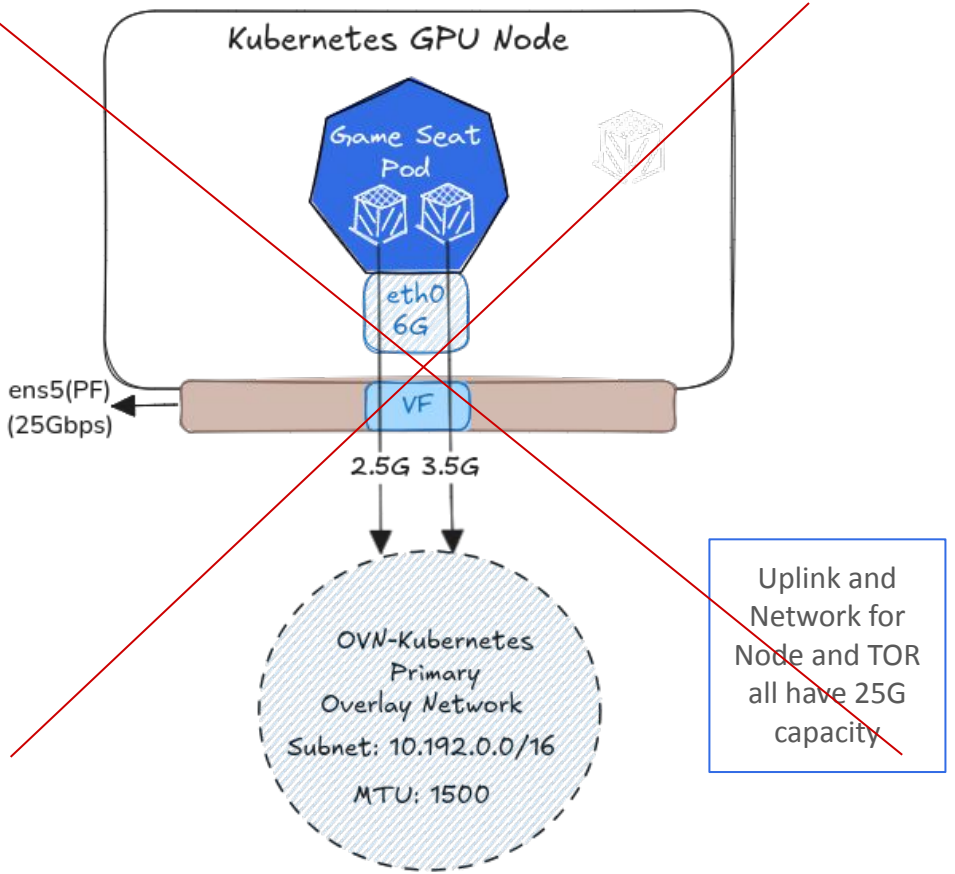
- We can achieve tunable bandwidth for secondary pod interfaces using K8s NPWG's [k8s.v1.cni.cncf.io/networks]() annotation.



Limitations of Multus Bandwidth Annotations

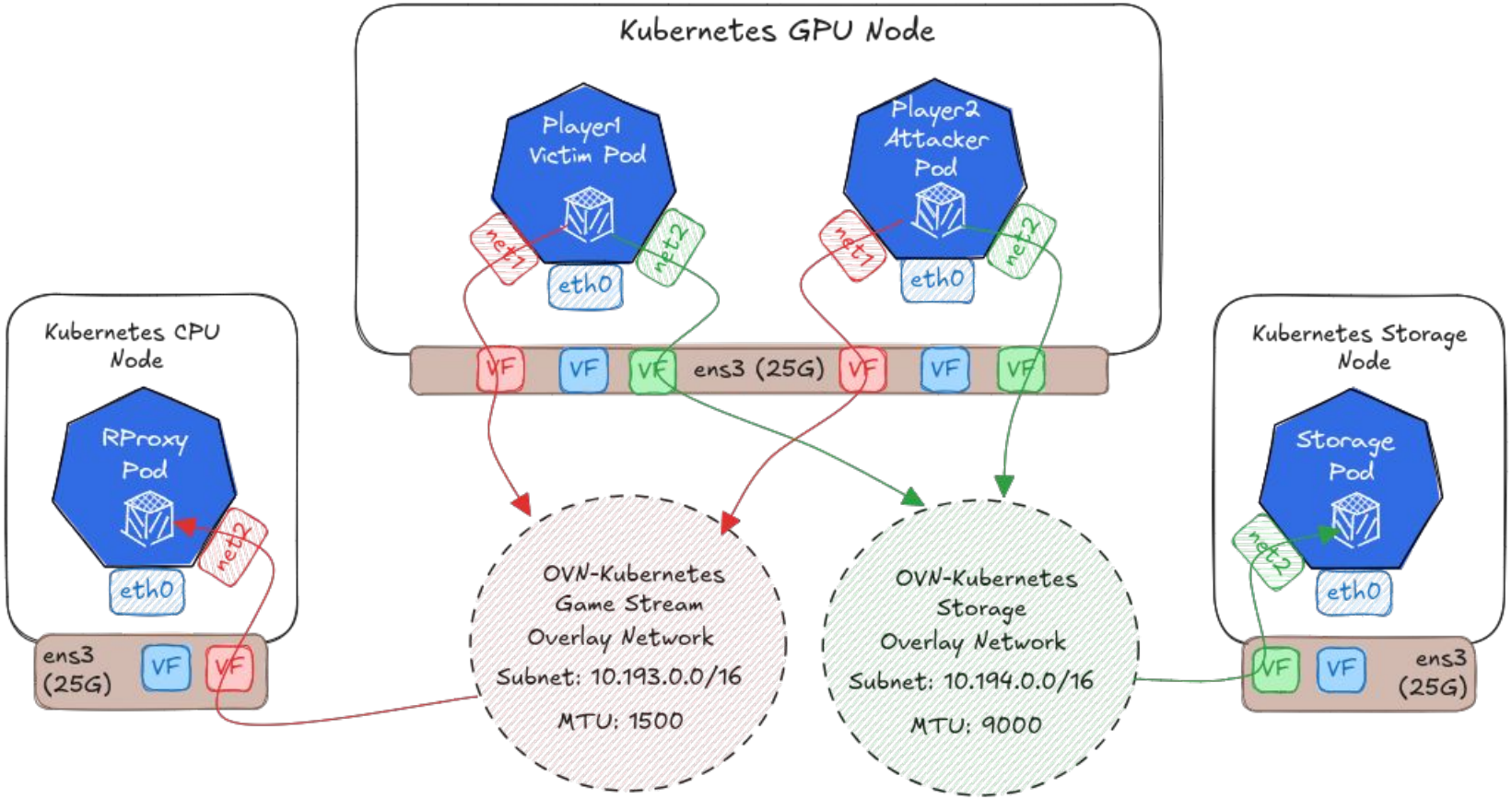
- API doesn't have a way to specify guaranteed bandwidth
- Limited to pod interface level
  - Not tunable for different types of traffic coming from same application
  - Not tunable for different applications within the pod

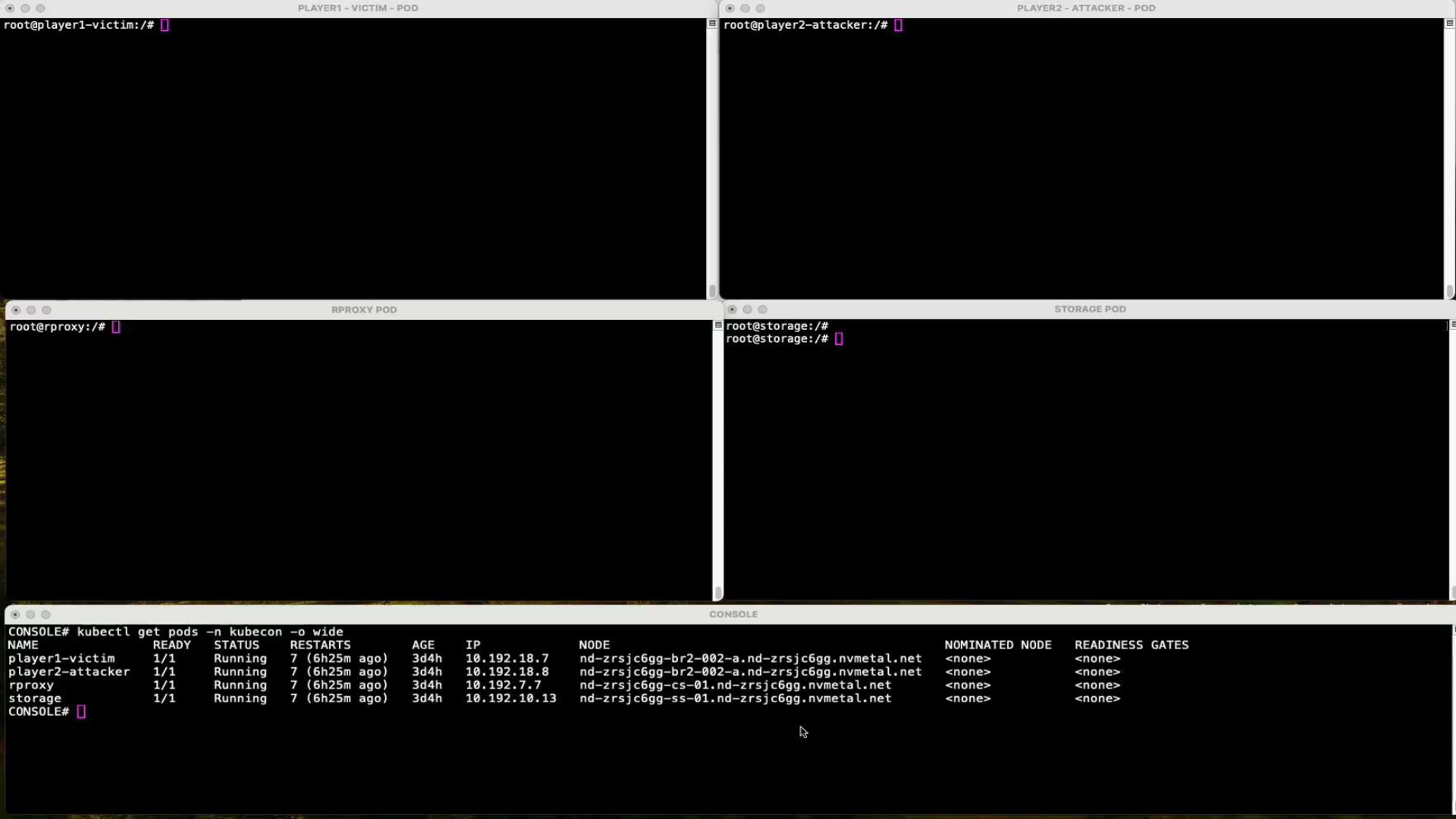
Packet parameter tuned traffic shaping is a must - but this cannot be achieved today using Multus Annotations



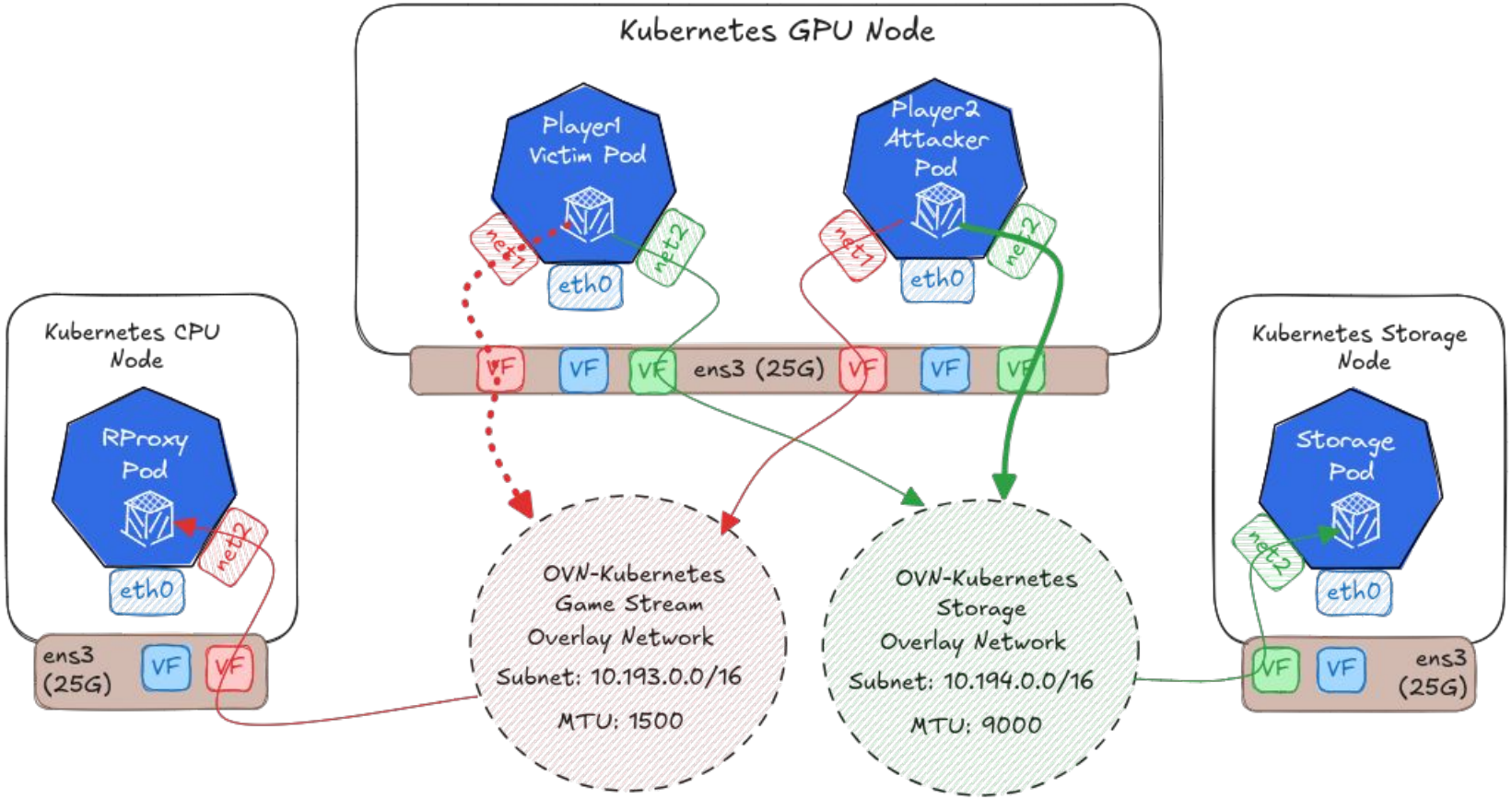
DEMO: Part1
Problem Statement

DEMO Setup


Kubernetes GPU Node
Player1 Victim Pod
Player2 Attacker Pod
net1
net2
eth0
VF
ens3 (25G)
Kubernetes CPU Node
RProxy Pod
Kubernetes Storage Node
Storage Pod
OVN-Kubernetes Game Stream Overlay Network
Subnet: 10.193.0.0/16
MTU: 1500
OVN-Kubernetes Storage Overlay Network
Subnet: 10.194.0.0/16
MTU: 9000


PLAYER1 - VICTIM - POD
root@player1-victim:/#
PLAYER2 - ATTACKER - POD
root@player2-attacker:/#
RPROXY POD
root@rproxy:/#
STORAGE POD
root@storage:/#
root@storage:/#
CONSOLE
CONSOLE# kubectl get pods -n kubecon -o wide
NAME READY STATUS RESTARTS AGE IP NODE NOMINATED NODE READINESS GATES
player1-victim 1/1 Running 7 (6h25m ago) 3d4h 10.192.18.7 nd-zrsjc6gg-br2-002-a.nd-zrsjc6gg.nvmetal.net <none> <none>
player2-attacker 1/1 Running 7 (6h25m ago) 3d4h 10.192.18.8 nd-zrsjc6gg-br2-002-a.nd-zrsjc6gg.nvmetal.net <none> <none>
rproxy 1/1 Running 7 (6h25m ago) 3d4h 10.192.7.7 nd-zrsjc6gg-cs-01.nd-zrsjc6gg.nvmetal.net <none> <none>
storage 1/1 Running 7 (6h25m ago) 3d4h 10.192.10.13 nd-zrsjc6gg-ss-01.nd-zrsjc6gg.nvmetal.net <none> <none>
CONSOLE#

DEMO Setup showing the problem



DEMO: Part2
Interim Solution with DSCP Markings

What is **D**ifferentiated **S**ervices **C**ode **P**oint?

- **DSCP** is a **6 bit** (2^6 = 64 possible values) field in the **IP** header of packets that tells network devices how to prioritize packets.
  - Packets marked with higher DSCP values get faster, preferential treatment through the network
  - Routers and Switches look at these values and decide which packets need to be processed first when there is traffic congestion
- Example: Voice traffic uses DSCP 46 (Expedited Forwarding) to get highest priority versus regular internet browsing uses DSCP 0 (Best Effort)

DSCP (Differentiated Services Code Point)

Common Values:

- Default (0): Best Effort
- AF41 (34): Streaming
- CS3 (24): Signaling
- EF (46): Voice/Video
- AF21 (18): Business Data
- CS1 (8): Background

https://datatracker.ietf.org/doc/html/rfc7657

Using DSCP markings at application level on pod traffic

- 64 values are mapped into 8 traffic classes in NVIDIA environments
- On the host we configure guaranteed bandwidth for each of the traffic class
- Added Benefit: Using this we can achieve true end-to-end QoS
  - Staging from source, to all the intermediary network switches/routers and finally to sink
- NVIDIA then uses hardware offload to offload these to NICs for optimal performance

DSCP (decimal)	IP TOS Byte (hex)	Traffic Class	Usage (For Example)	Traffic allocation % of uplink speed
0	0x00	0	BE	5
32	**0x80**	**1**	**Streaming**	**70**
33	0x84	2	App1	1
34	**0x88**	**3**	**Storage**	**20**
1-31	0x4-0x7C	4	App2	1
35	0x8C	5	Admin	1
48	0xC0	6	Net Ctrl.	1
56	0xE0	7	Reserved	1

PLAYER1 - VICTIM - POD

```
root@player1-victim:/#
```

PLAYER2 - ATTACKER - POD

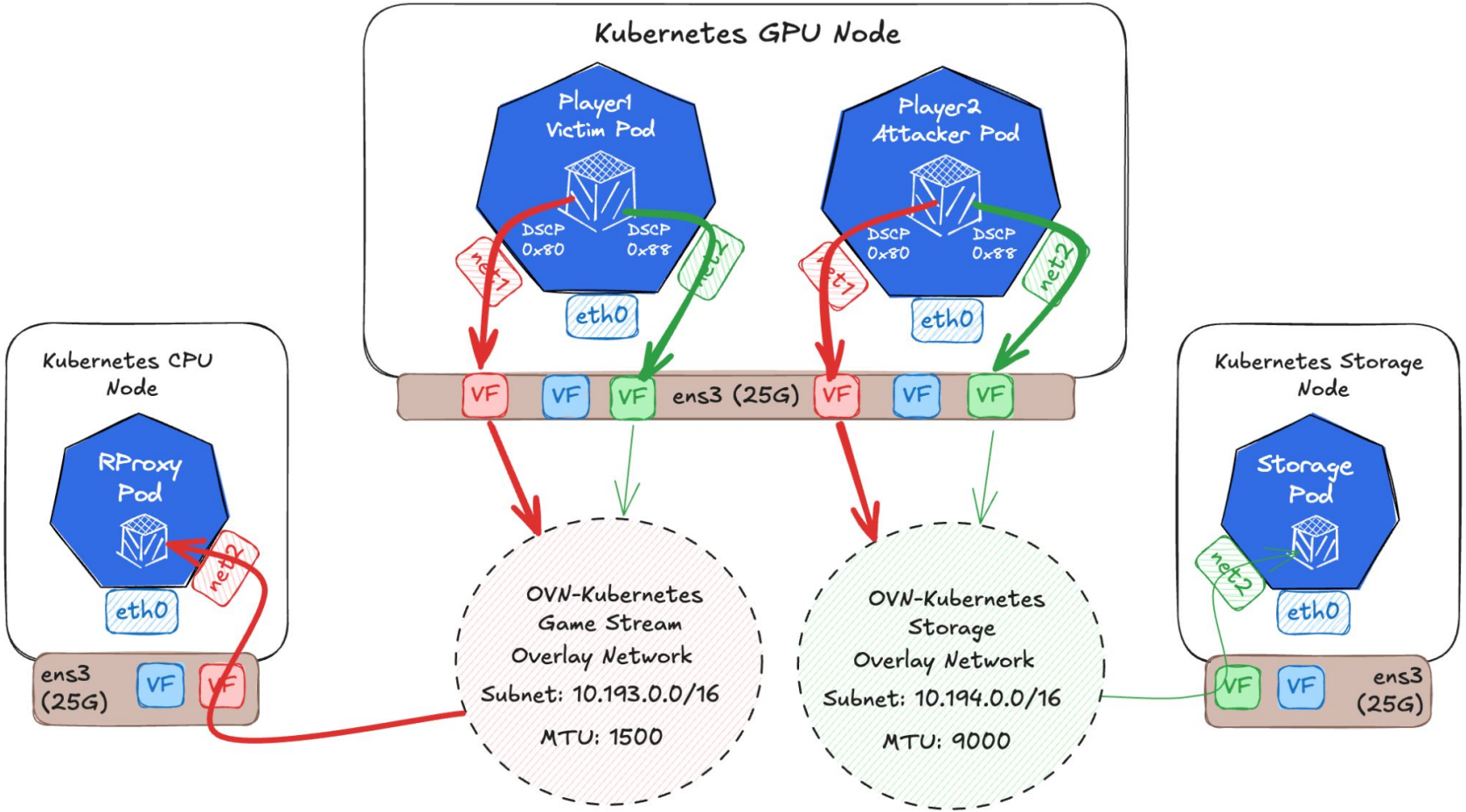
```
root@player2-attacker:/#
```

root@nd-zrsjc6gg-br2-002-a:/home/ngnadmin

```
CONSOLE# mlnx_qos -i enp197s0f0
****** WARNING: lldpad service is running and may overwrite your settings ******

DCBX mode: OS controlled
Priority trust state: dscp
dscp2prio mapping:
        prio:0 dscp:63,62,61,60,59,58,57,55,
        prio:0 dscp:54,53,52,51,50,49,47,46,
        prio:0 dscp:45,44,43,42,41,39,38,37,
        prio:0 dscp:36,00,
        prio:1 dscp:32,
        prio:2 dscp:40,33,
        prio:3 dscp:34,
        prio:4 dscp:31,30,29,28,27,26,25,24,
        prio:4 dscp:23,22,21,20,19,18,17,16,
        prio:4 dscp:15,14,13,12,11,10,09,08,
        prio:4 dscp:07,06,05,04,03,02,01,
        prio:5 dscp:35,
        prio:6 dscp:48,
        prio:7 dscp:56,
default priority:
Receive buffer size (bytes): 0,156096,0,0,0,0,0,0,max_buffer_size=1027728
Cable len: 7
PFC configuration:
        priority    0   1   2   3   4   5   6   7
        enabled     0   0   0   0   0   0   0   0
        buffer      1   1   1   1   1   1   1   1
tc: 0 ratelimit: unlimited, tsa: ets, bw: 5%
         priority:  0
tc: 1 ratelimit: unlimited, tsa: ets, bw: 70%
         priority:  1
tc: 2 ratelimit: unlimited, tsa: ets, bw: 1%
         priority:  2
tc: 3 ratelimit: unlimited, tsa: ets, bw: 20%
         priority:  3
tc: 4 ratelimit: unlimited, tsa: ets, bw: 1%
         priority:  4
tc: 5 ratelimit: unlimited, tsa: ets, bw: 1%
         priority:  5
tc: 6 ratelimit: unlimited, tsa: ets, bw: 1%
         priority:  6
tc: 7 ratelimit: unlimited, tsa: ets, bw: 1%
         priority:  7
CONSOLE#
```

DEMO Visualization



Drawbacks of using DSCP Markings in raw fashion

- Trust Boundary Issues 
  - Users/Apps can mark traffic however they want
  - No authentication
- Management Issues
  - Multiple teams trying to use the same DSCP space
  - No central coordination/validation
- System Fragility
  - Overlapping Markings
  - Error prone

Lacking proper API and RBAC

DEMO: Part3
Final Solution using Network QoS CRD - Adding an API to do DSCP Markings

NetworkQoS API

- A Custom Resource Definition that allows users to define a DSCP marking and metering for pods' ingress/egress traffic on an application level

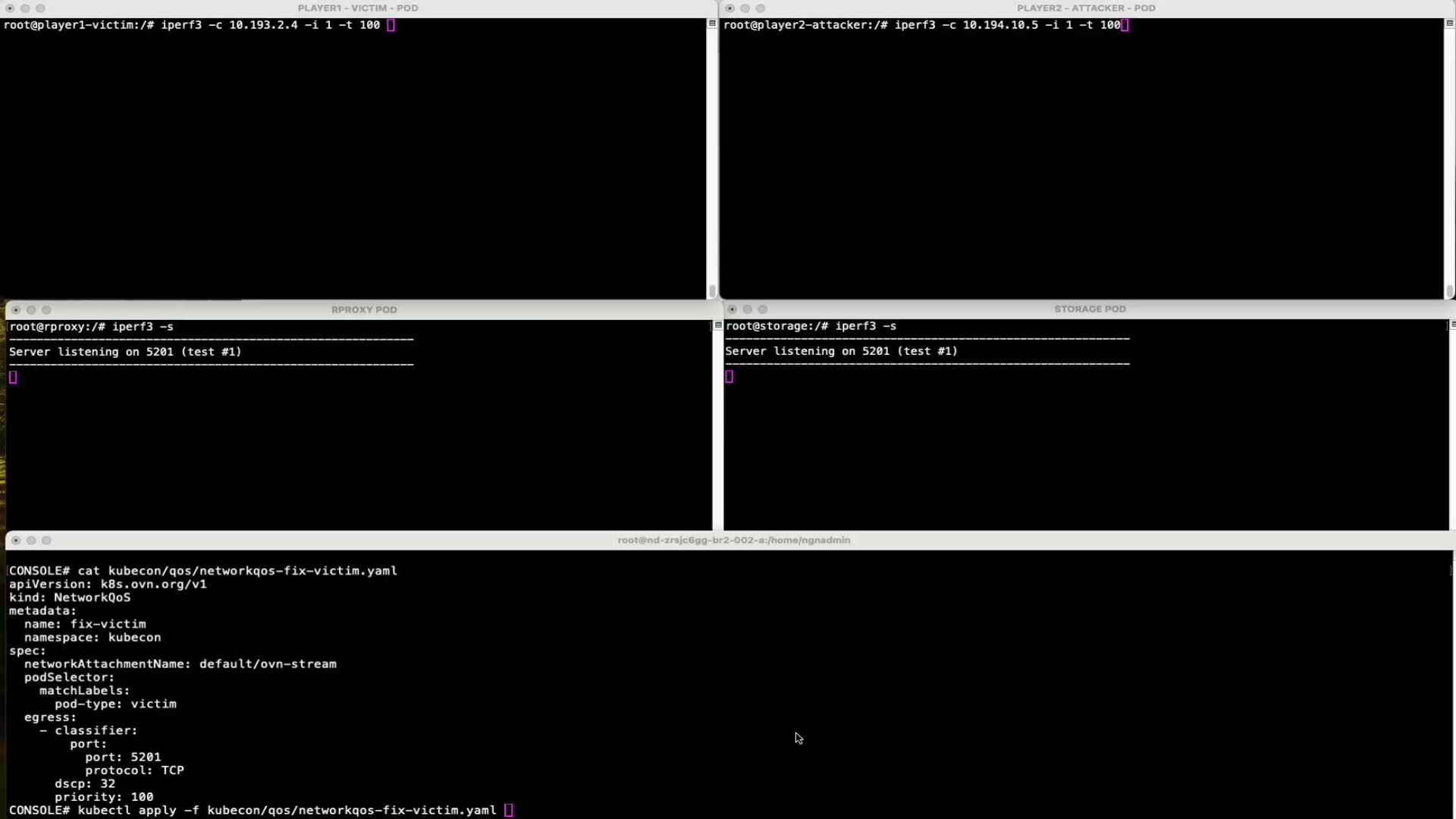
```
spec  <Object>
  podSelector  <Object>
    <output snipped>
  networkAttachmentName  <string>
  egress  <[]Object> -required-
    classifier  <Object>
      port  <Object>
        port  <integer>
        protocol  <string>
      to  <[]Object>
        ipBlock  <Object>
          cidr  <string> -required-
          except  <[]string>
        namespaceSelector  <Object>
          <output snipped>
        podSelector  <Object>
          <output snipped>
    bandwidth  <Object>
      burst  <integer>
      rate  <integer>
    dscp  <integer> -required-
    priority  <integer> -required-
```

Specifies the network interface inside the pod for which QoS Rules will be applied

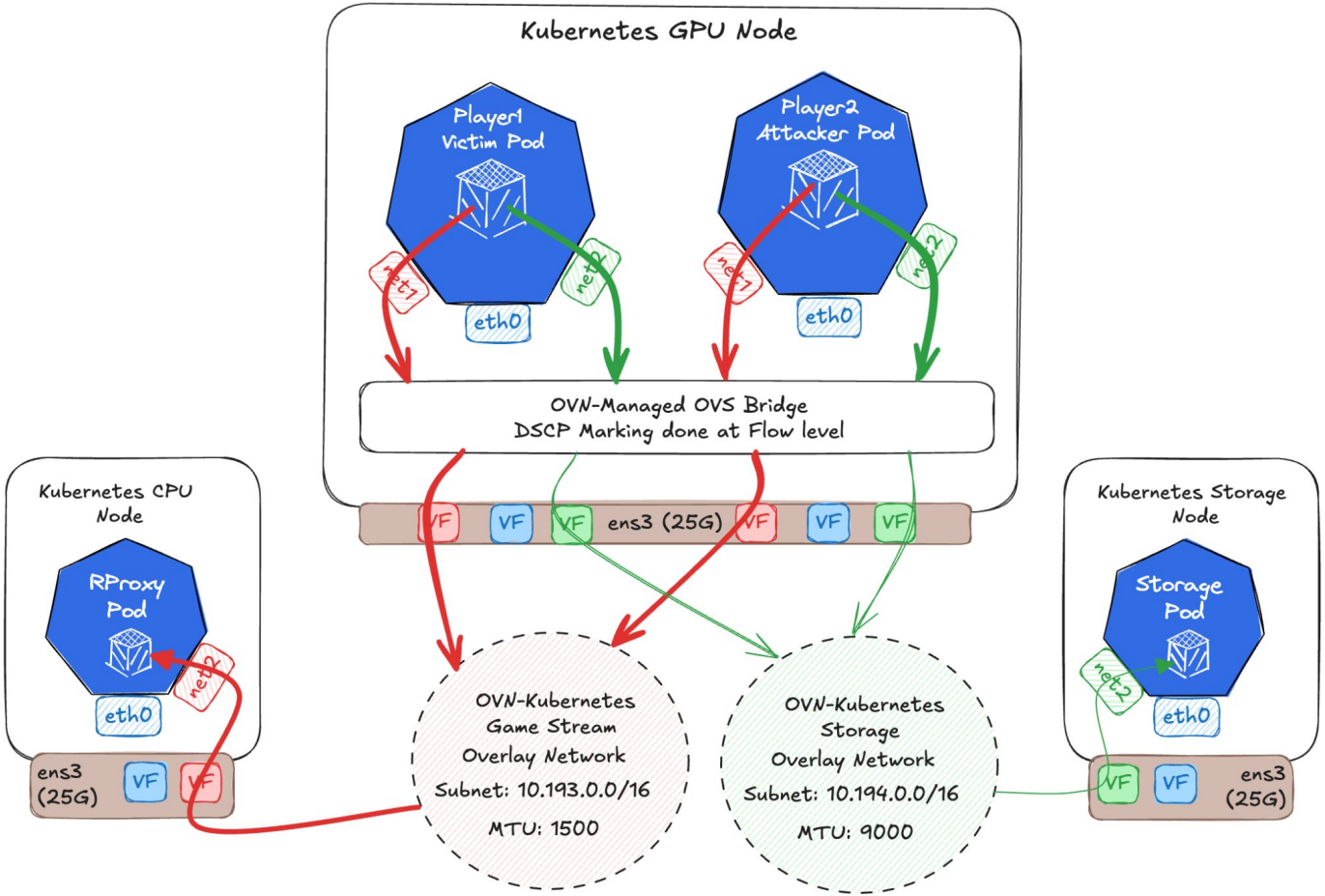
Fine grained way of identifying a particular traffic on the interface

Maximum bandwidth allowed for the matched traffic

DSCP marking for E2E QoS and for minimum bandwidth configuration

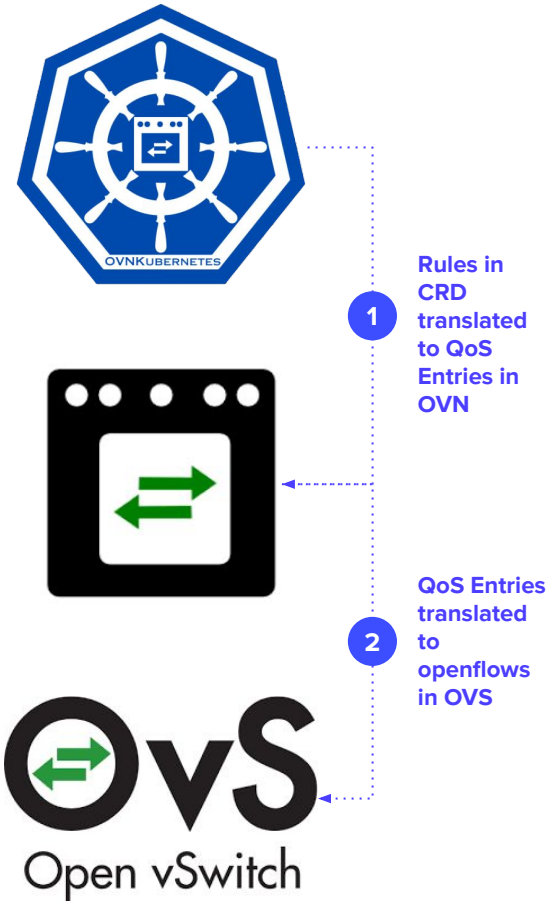

PLAYER1 - VICTIM - POD
root@player1-victim:/# iperf3 -c 10.193.2.4 -i 1 -t 100
PLAYER2 - ATTACKER - POD
root@player2-attacker:/# iperf3 -c 10.194.10.5 -i 1 -t 100
RPROXY POD
root@rproxy:/# iperf3 -s
Server listening on 5201 (test #1)
STORAGE POD
root@storage:/# iperf3 -s
Server listening on 5201 (test #1)
root@nd-zrsjc6gg-br2-002-a:/home/ngnadmin
CONSOLE# cat kubecon/qos/networkqos-fix-victim.yaml
apiVersion: k8s.ovn.org/v1
kind: NetworkQoS
metadata:
name: fix-victim
namespace: kubecon
spec:
networkAttachmentName: default/ovn-stream
podSelector:
matchLabels:
pod-type: victim
egress:
- classifier:
port:
port: 5201
protocol: TCP
dscp: 32
priority: 100
CONSOLE# kubectl apply -f kubecon/qos/networkqos-fix-victim.yaml

DEMO Visualization



NetworkQoS API Implementation details

- A Custom Resource Definition that allows users to define a DSCP marking and metering for pods' ingress/egress traffic on an application level
- Currently being developed in OVN-Kubernetes CNI (*new CNCF sandbox project*)
  - Enhancement
  - PoC
  - Uses OVN QoS and OVS QoS underneath



Thank YOU! Questions?

- Adding NetworkQoS API to Kubernetes?
  - Regardless of Implementation details we think having an API to do bandwidth controls more than what the pod-level annotation provides is useful for the community.
  - Having bandwidth controls for network traffic is a common requirement in a lot of types of networks - not just in gaming
  - If there is enough interest we would like to contribute or develop this feature in upstream Kubernetes rather than just doing it in OVN-Kubernetes
  - So certainly provide feedback around this!
- Check out our Design and PoC!

KubeCon | CloudNativeCon

North America 2024