

Inline DDoS protection for cloud-native game servers at PlayStation with eBPF/XDP

Charalampos (Babis) Stylianopoulos
Jeffrey Barendse

Linux Plumbers Conference 2026, Prague

Connectivity to game servers at global scale



- SIE maintains a global backbone network



Connectivity to game servers at global scale

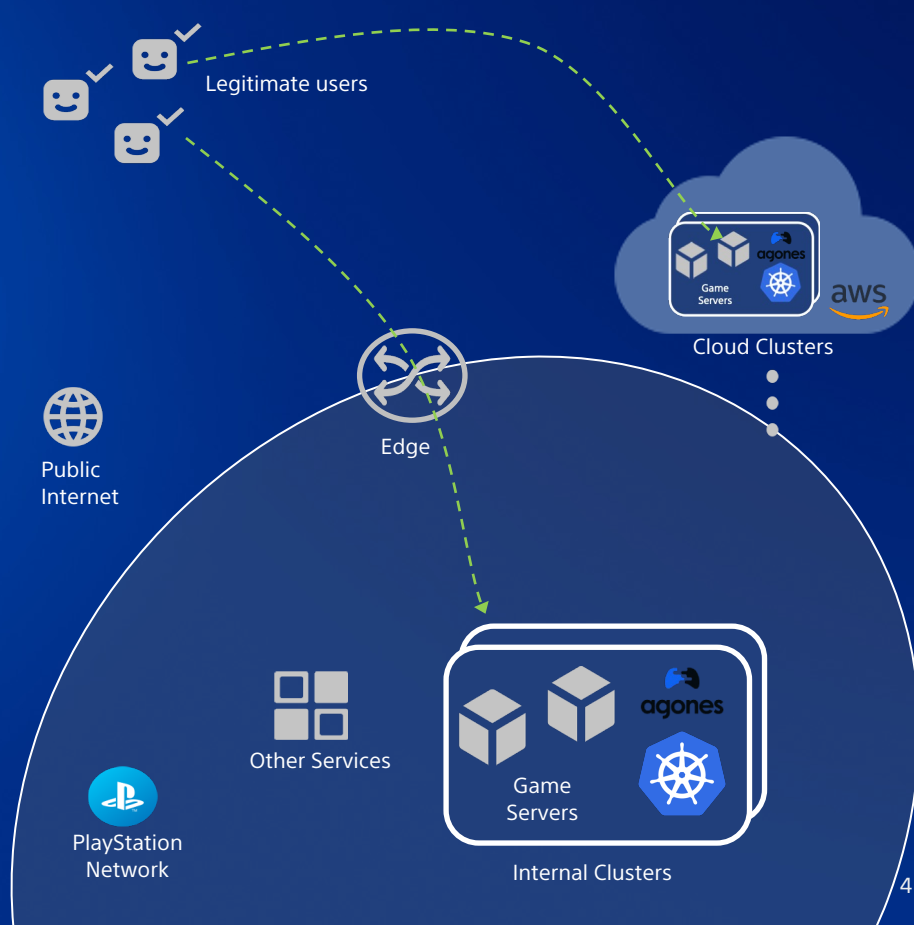


- SIE maintains a global backbone network
- Studios host game servers
 - On prem, within the SIE net
 - On public clouds, outside the SIE net
- Game servers are Kubernetes pods, allocated on-demand at game session start
- High churn rate



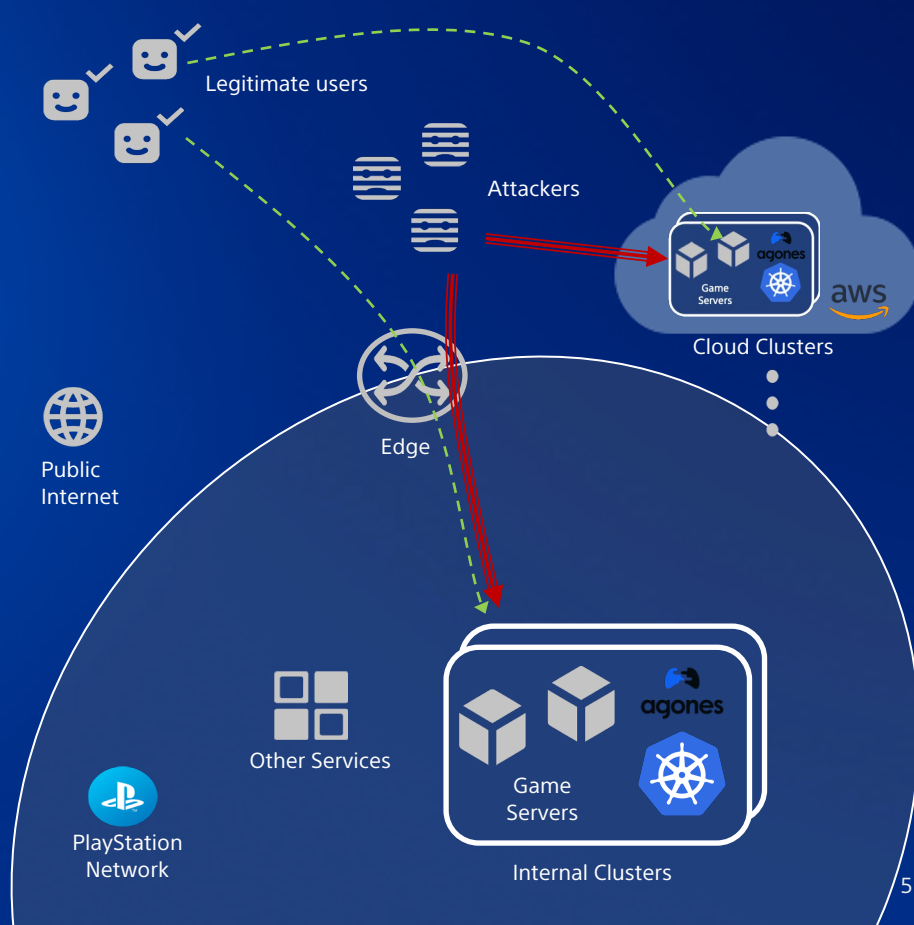
Why the traditional model is not a good fit

- Traditionally game-server is exposed to the public Internet
- Operational and security burden



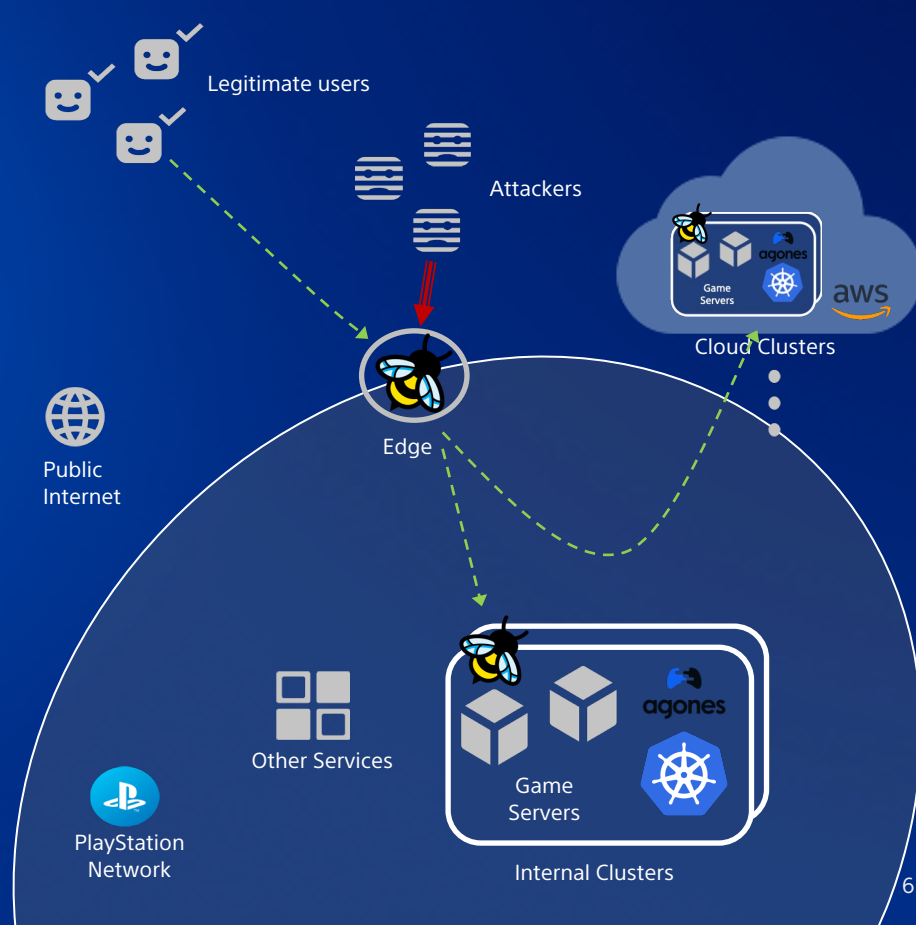
Why the traditional model is not a good fit

- Traditionally game-server is exposed to the public Internet
- Operational and security burden
- DDoS attacks can reach multi-terabit scale
- Can we utilize the edge for game-specific DDoS protection as early as possible, as fast as possible?



Our design goals

- Keep individual game-server clusters private
- Drop DDoS traffic at the earliest practical hook
- Require no game-server application changes
- On-prem and cloud
- Support high pod churn rate



Architecture at a glance



Internet



Edge Router



XDP Router



XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

Architecture at a glance



UDP packet



Internet



Edge Router



XDP Router

Edge



XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

- Legitimate sessions are assigned short-lived admission/routing state
- Packets provide a lookup key that associates them with that state



Internet



Edge Router



XDP Router



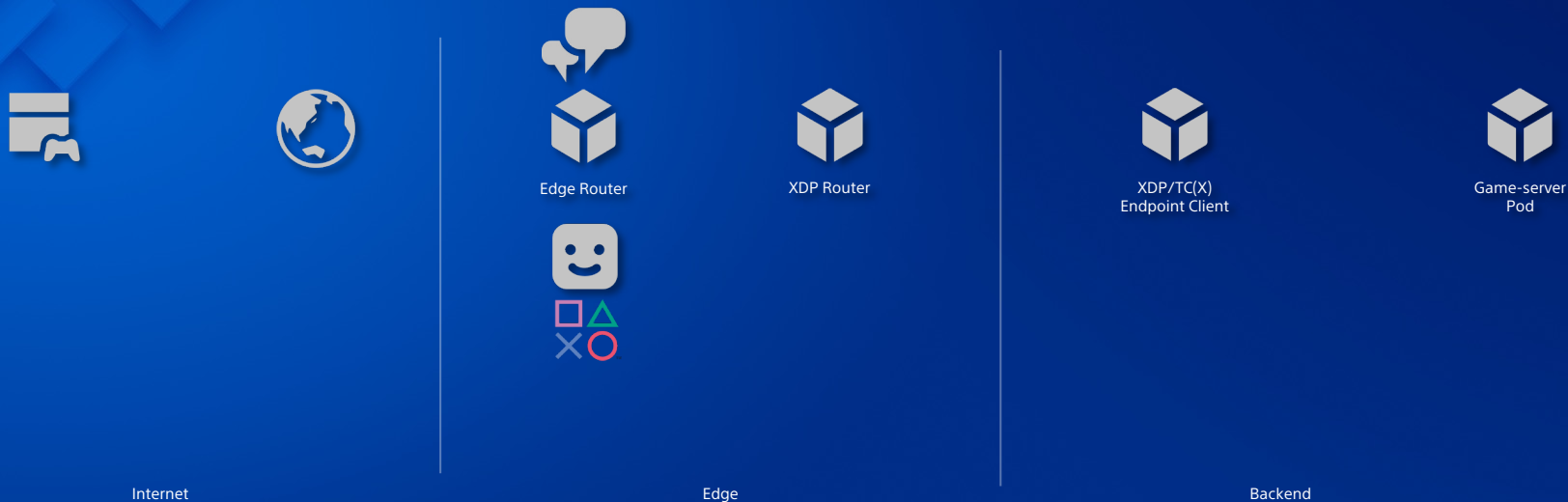
XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

- Internet sees the public VIP rather than individual game servers
- Public address remains stable while game-server placement changes



Architecture at a glance



- Control plane distributes admission/routing state 🗝️
- XDP/eBPF performs enforcement and routing lookup



Internet



Edge Router



XDP Router



Edge



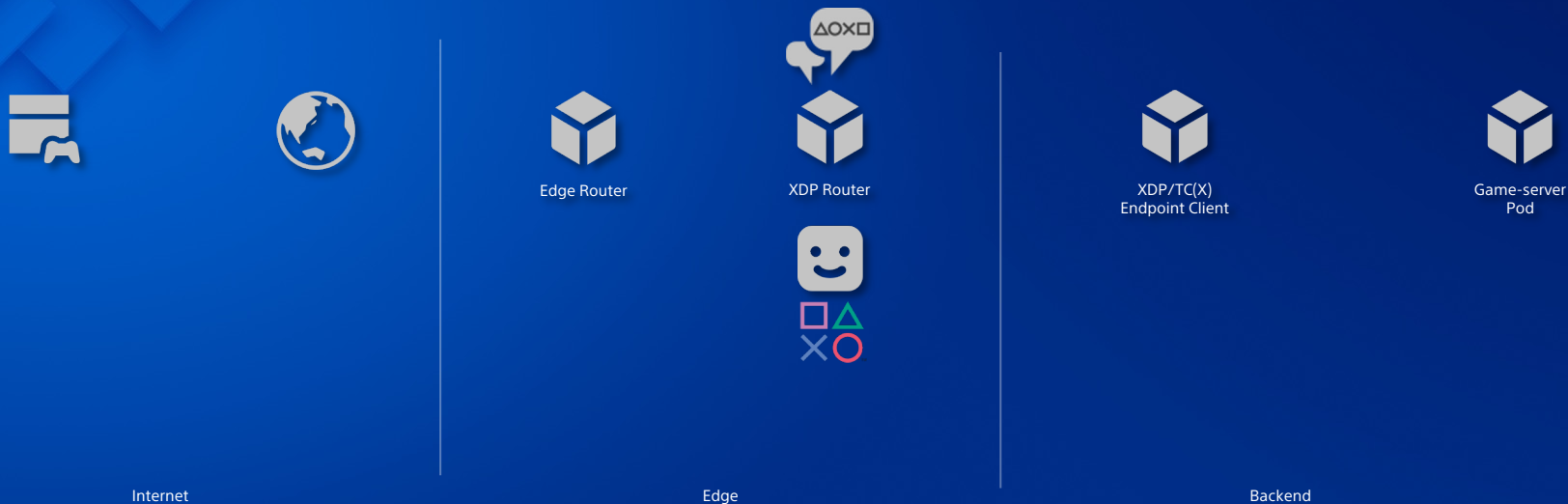
XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

- Get admission/routing state and appropriate private backend route



- Accepted traffic enters an overlay toward a backend worker



Internet



Edge Router



XDP Router



Edge



XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

- Encapsulated packet flows to XDP on backend worker



Internet



Edge Router



XDP Router

Edge



XDP/TC(X)
Endpoint Client



Game-server
Pod

Backend

- Worker-side eBPF delivers traffic transparently to the game server



Internet



Edge Router



XDP Router

Edge



XDP/TC(X)
Endpoint Client



Backend



Game-server
Pod

- XDP native mode
 - Earliest practical programmable receive hook
 - Avoid unnecessary skb, routing, netfilter, socket, and application processing
 - Natural fit for parse → lookup → drop [MTU, fragmentation, offloads]

- XDP native mode

- Earliest practical programmable receive hook
- Avoid unnecessary skb, routing, netfilter, socket, and application processing
- Natural fit for parse → lookup → drop [MTU, fragmentation, offloads]

- Maps

- All maps are “Per-CPU” duplicated to avoid synchronization and cache-coherence traffic
- Programs change rarely; maps absorb high-churn state
- Expiration naturally limits stale state (LRU)

Globally distribute state to the network edge with xDS



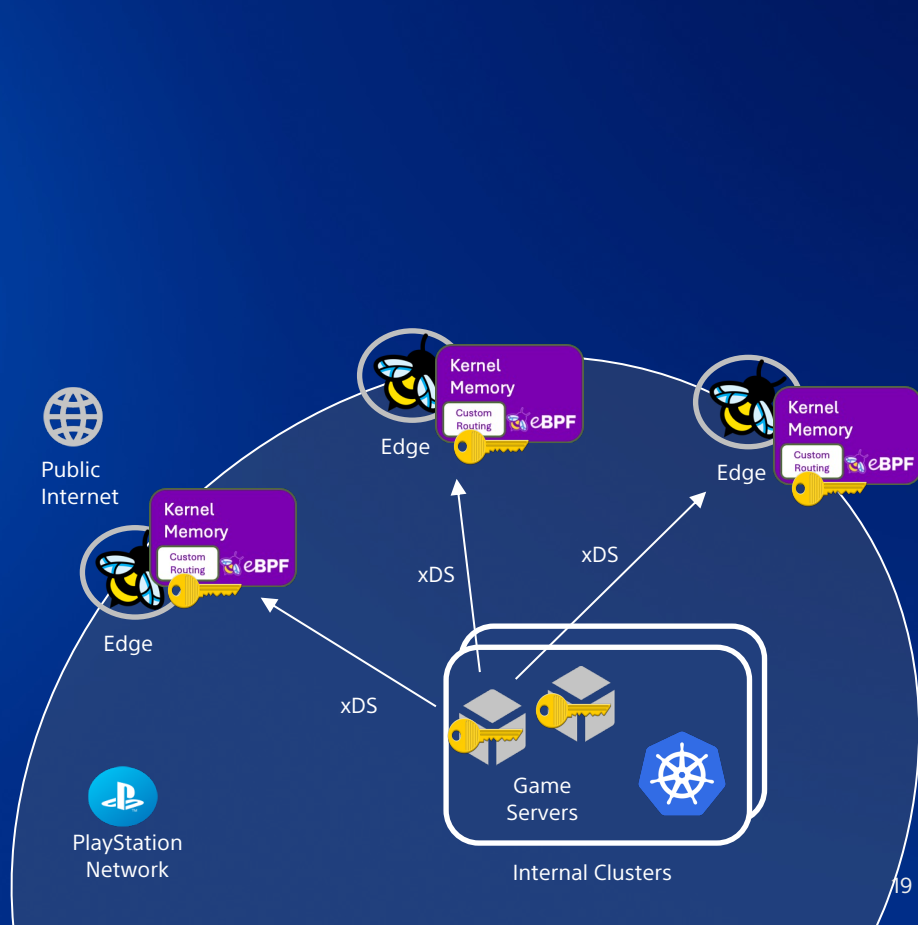
- Game cluster agent collects admission/routing state
- Inspiration from  QUILKIN



Globally distribute state to the network edge with xDS

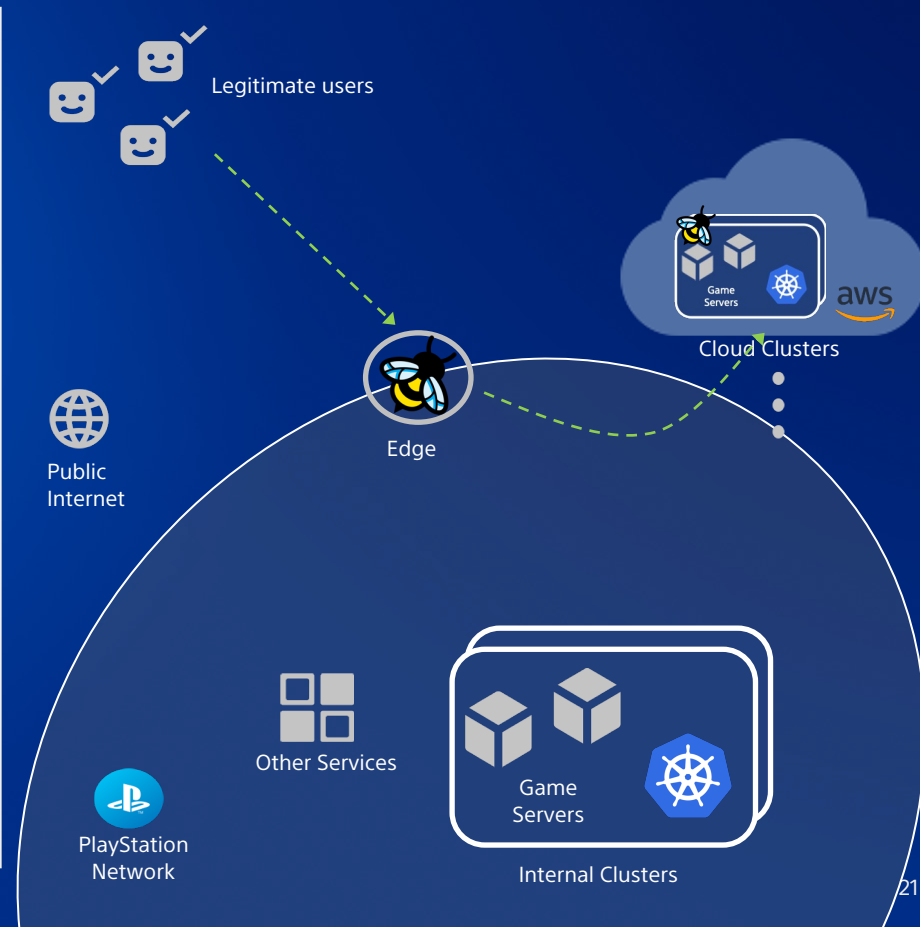


- Game cluster agent collects admission/routing state
- Inspiration from  QUILKIN
- xDS = x Discovery Service
- On diff, fan-out state to the edge
- From Kubernetes annotations to fleet-wide eBPF map updates



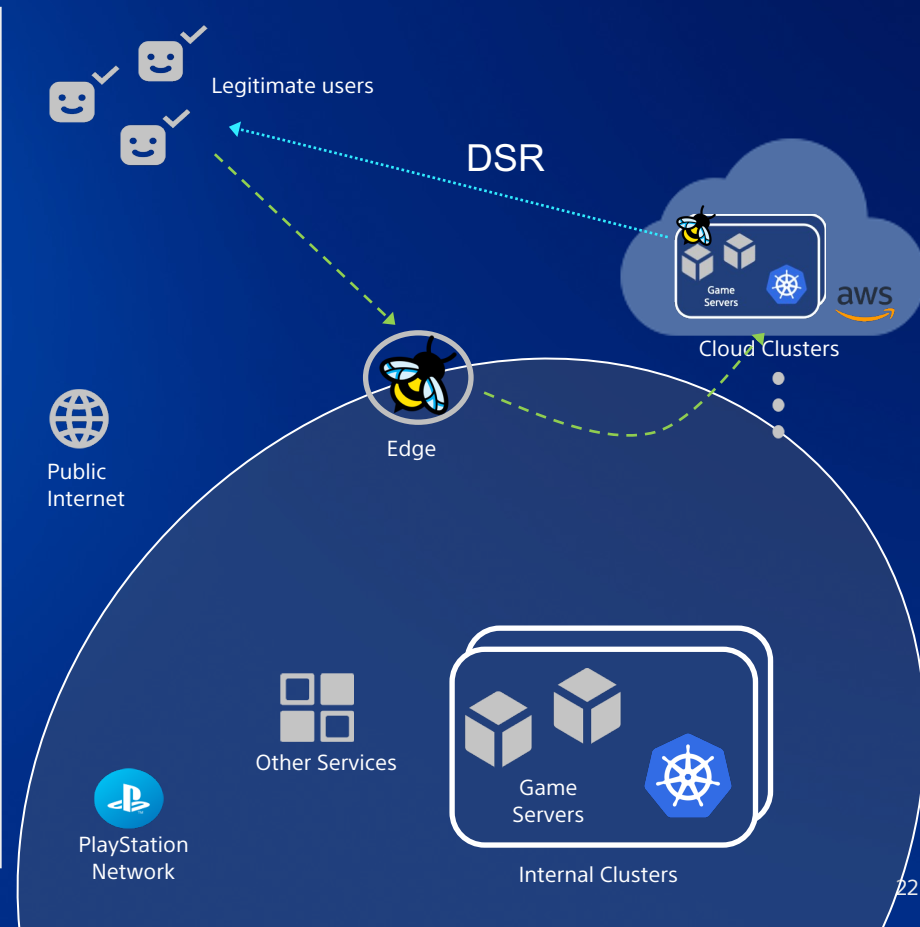
Design Decisions and Discussion

Return path: DSR or symmetric?



Return path: DSR or symmetric?

- Direct Server Return (DSR)
- No backbone resources spent on the return packet
- No encapsulation needed

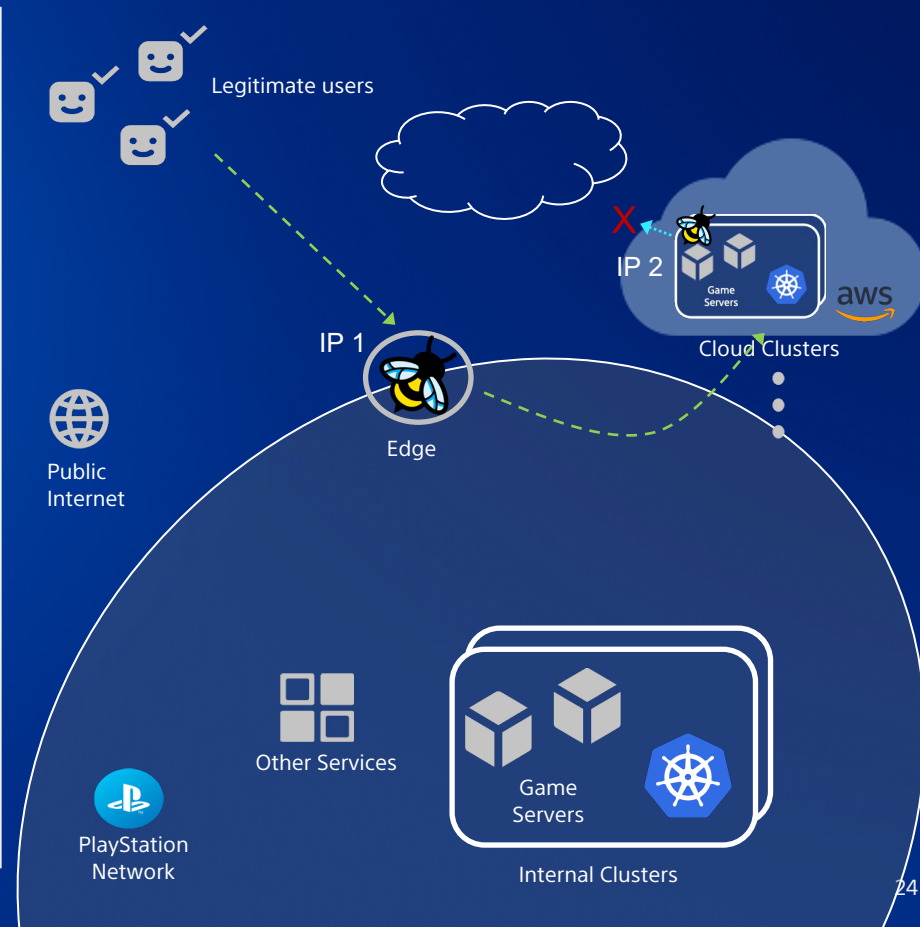




- [illegible]

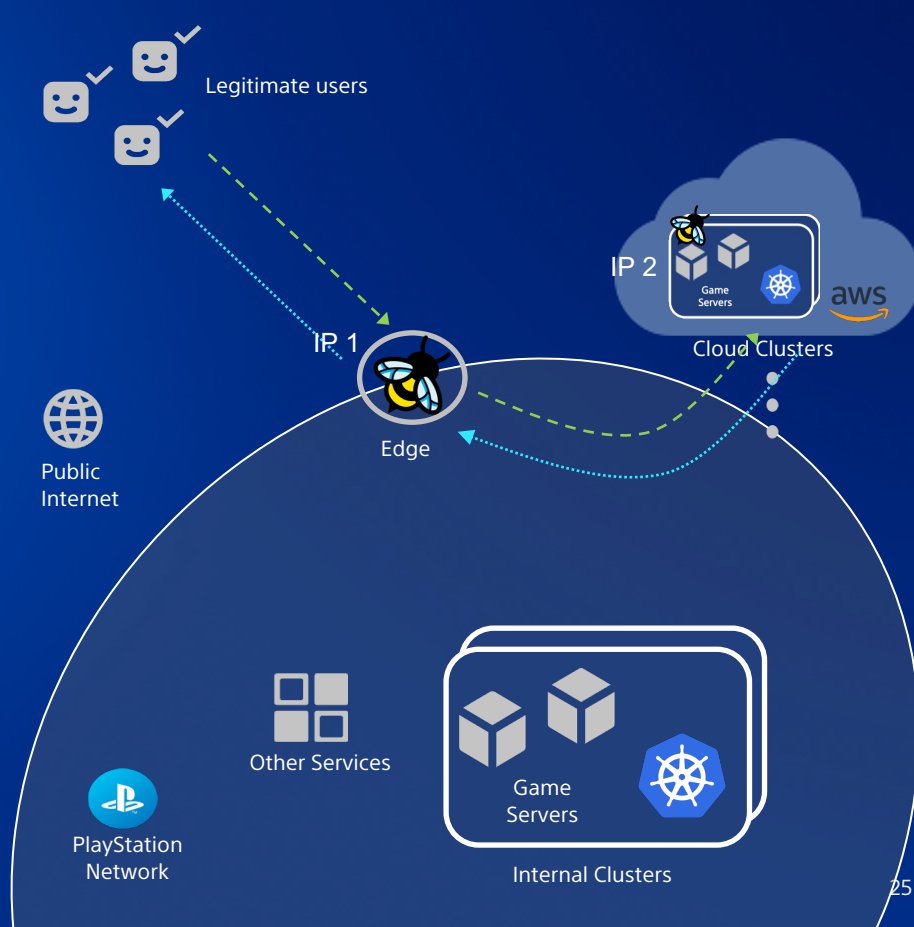
Return path: DSR or symmetric?

- Direct Server Return (DSR)
- No backbone resources spent on the return packet
- No encapsulation needed
- But...



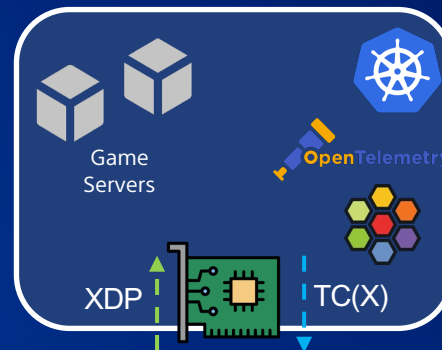
Return path: DSR or symmetric?

- Symmetric return path
- Encapsulate return packet and send back via the edge



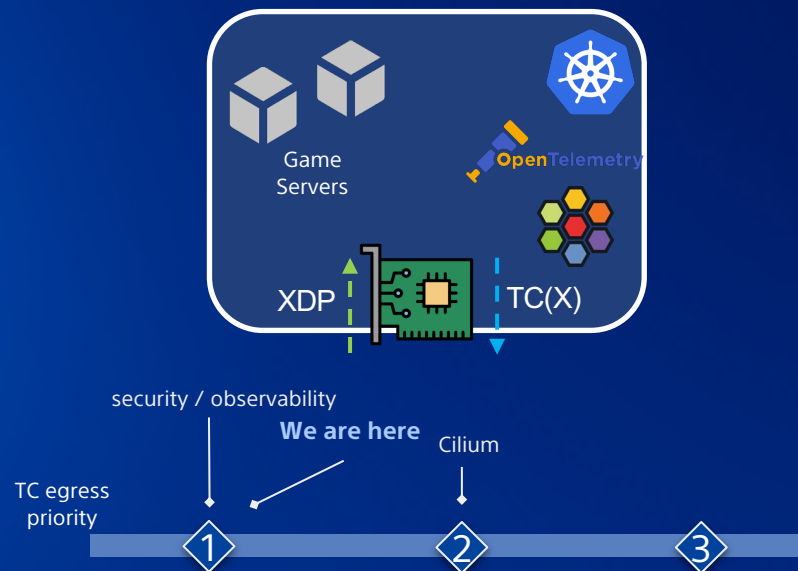
Why Kubernetes CNI integration is required

- TCX hook points are crowded



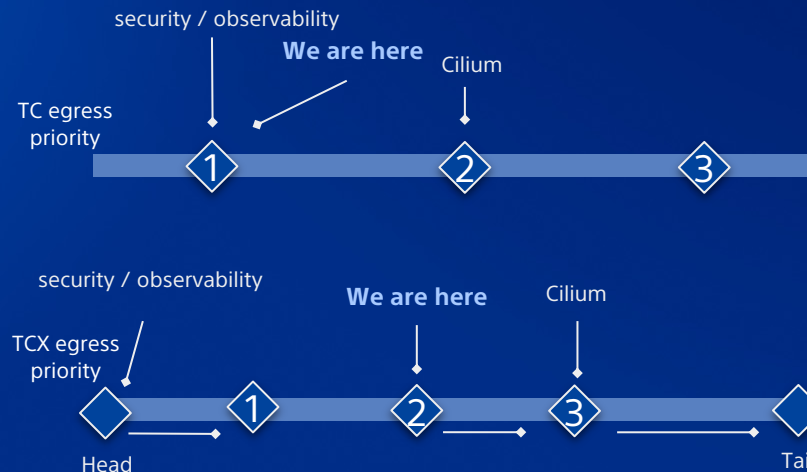
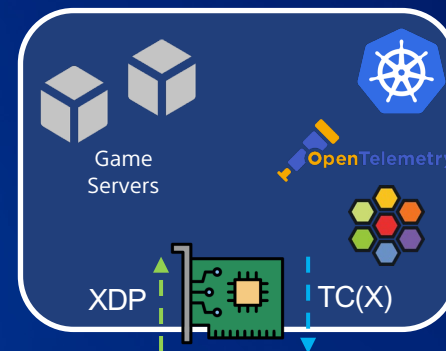
Why Kubernetes CNI integration is required

- TCX hook points are crowded
- Older kernels
 - TC, priority declared numerically
 - Hard to manage on customer owned clusters

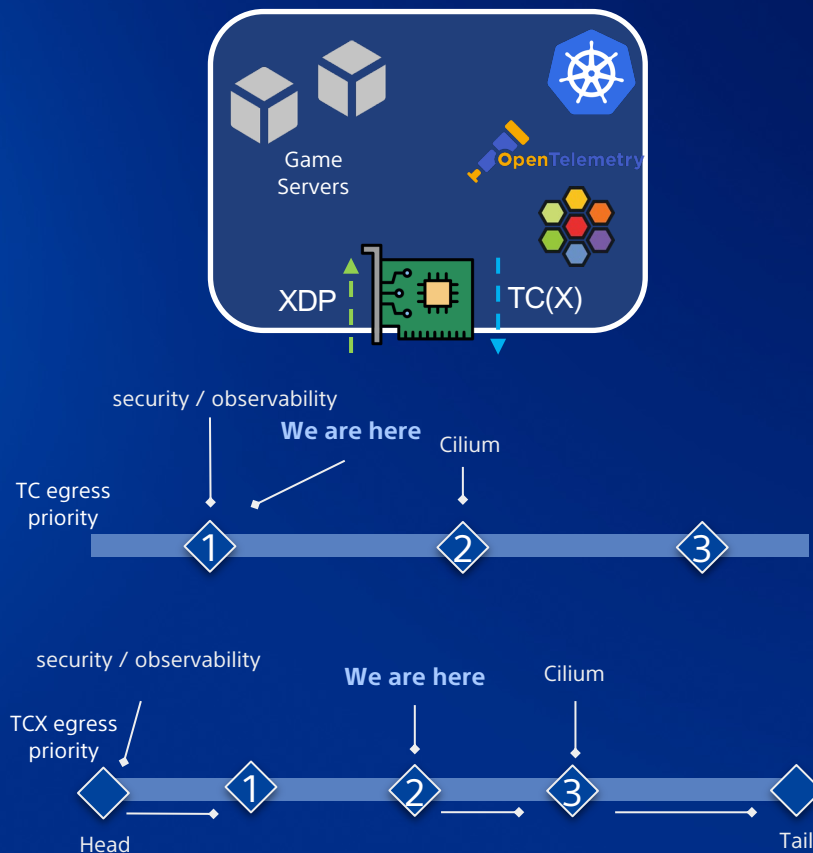


Why Kubernetes CNI integration is required

- TCX hook points are crowded
- Older kernels
 - TC, priority declared numerically
 - Hard to manage on customer owned clusters
- Newer kernels
 - TCX offers more flexibility
 - Head, Tail, or relative to another program



- CNI specific logic
 - If Cilium, find egress prog at runtime and hook
 - If AWS VPC CNI, no overlaps, but still needs tweaks to run native XDP
- Possible paths forward
 - Cilium datapath plugins
 - Bpfman



What we learned operating this at global scale



- Make unwanted traffic cheap rather than trying to classify every attack
- CNI integration is fundamentally a dataplane-composition problem
- Scalable state distribution matters as much as fast packet processing

- Make unwanted traffic cheap rather than trying to classify every attack
- CNI integration is fundamentally a dataplane-composition problem
- Scalable state distribution matters as much as fast packet processing

Open problems / proposed direction

- Fleet-wide eBPF state distribution would benefit from a generic kernel/service mechanism for synchronizing map state
- In need of standardized ordering and ownership model for independently managed XDP/TCX programs
 - Bpfman, Custom dispatchers, ... ?