



The bridge to possible

Media Streaming Mesh

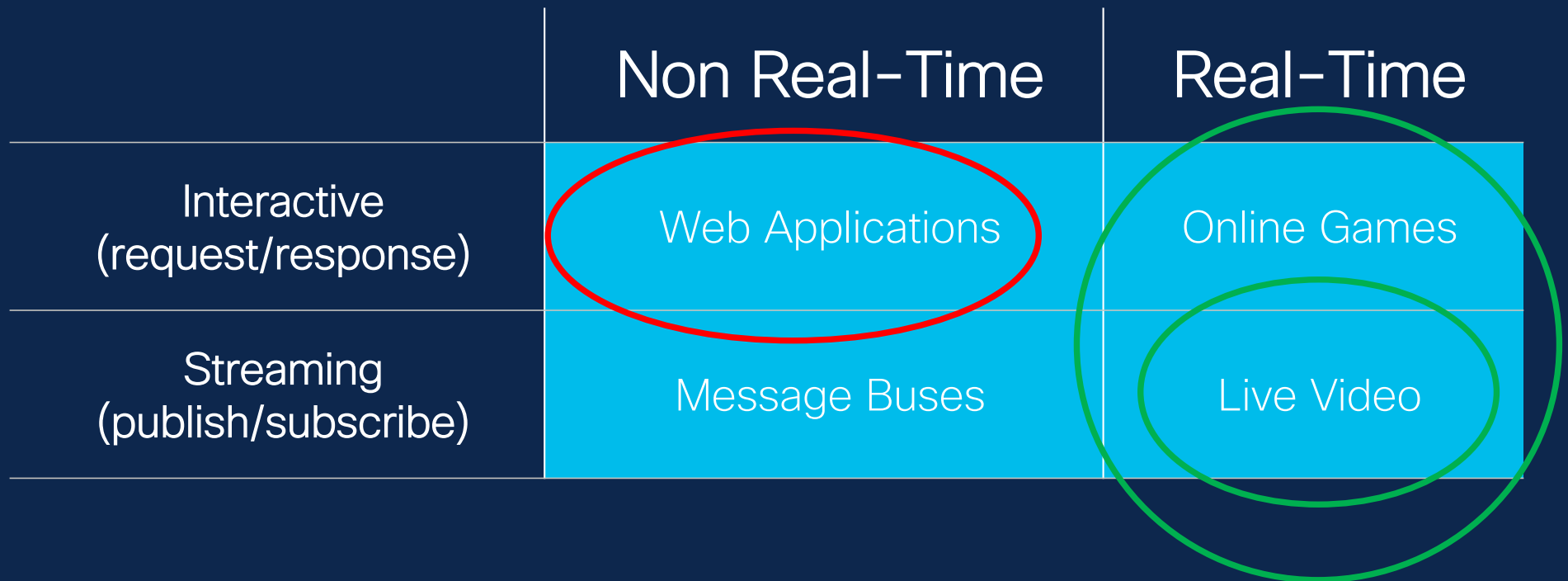
Giles Heron – Principal Engineer

Nikos Bregiannis – Technical Leader

26th January 2022

A (Fuzzy) Application Taxonomy

	Non Real-Time	Real-Time
Interactive (request/response)	Web Applications	Online Games
Streaming (publish/subscribe)	Message Buses	Live Video



Use-Cases for Media Streaming Mesh

- Contribution video (high bandwidth, RTP-based, no tolerance for loss/jitter)
- Live video distribution (RTP-based, low tolerance for loss/jitter)
- Retail/Industrial Edge (need low footprint, lots of RTSP video streaming/analytics)
- Real-Time Collaboration (WebRTC etc.)

Previously – no longer in focus (though potential scope to leverage RTP here):

- Gaming (latency is key so “action” games use UDP rather than TCP)
- Finance (high frequency trading etc. – latency critical).
- Mobile (GTP is tunnelled over UDP – and is latency sensitive).

Benefits of Media Streaming Mesh



Observability

Media Streaming Mesh monitors jitter and packet loss across the mesh, enabling DevOps teams to quickly locate and resolve connectivity issues.



Low-Latency

The Media Streaming Mesh RTP data plane proxy adds minimal latency, in contrast to web proxies that terminate TCP connections at each hop.

Security

Media Streaming Mesh authenticates traffic senders using SPIFFE/SPIRE and can optionally encrypt traffic as it flows from pod to pod using sRTP.



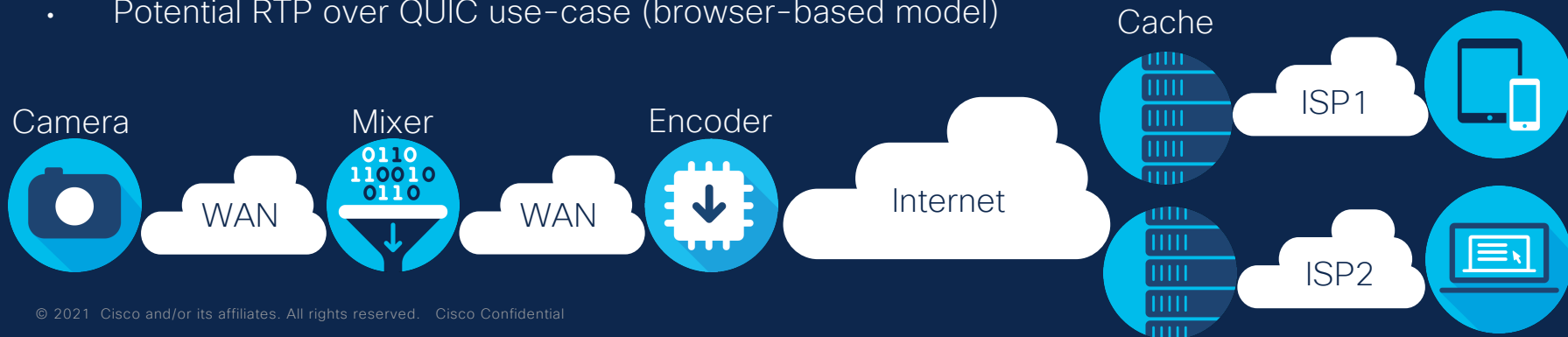
Deployability

Lightweight per-node data plane proxy, and per-cluster control plane proxy ensures a much lower footprint than per-pod web proxies, making it suitable for deployment at the edge.



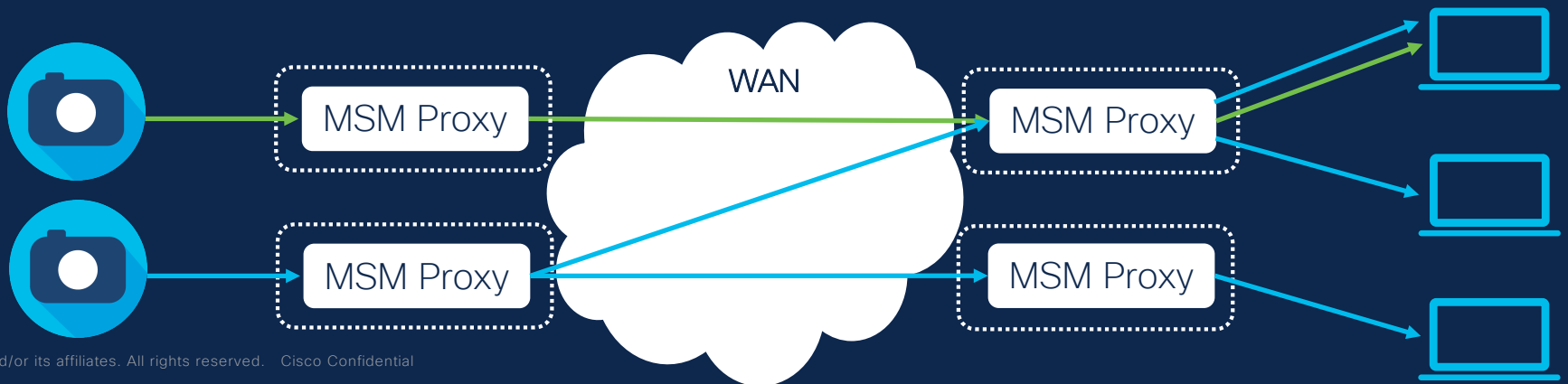
Broadcast Media Use-Cases for Media Streaming Mesh

1. Contribution Video (camera to studio and in-studio mixing)
 - Longer-term goal perhaps as cameras/switches/consoles are dedicated hardware platforms
2. Interconnection of cloud-based encoders
 - Most likely an intra-cluster Kubernetes use-case
3. Distributing live streams from encoders to caches
 - Fan out from encoders via MSM proxies
 - Proxies can add FEC, send dual streams over diverse paths etc.
4. Streaming RTP to clients
 - Potential RTP over QUIC use-case (browser-based model)

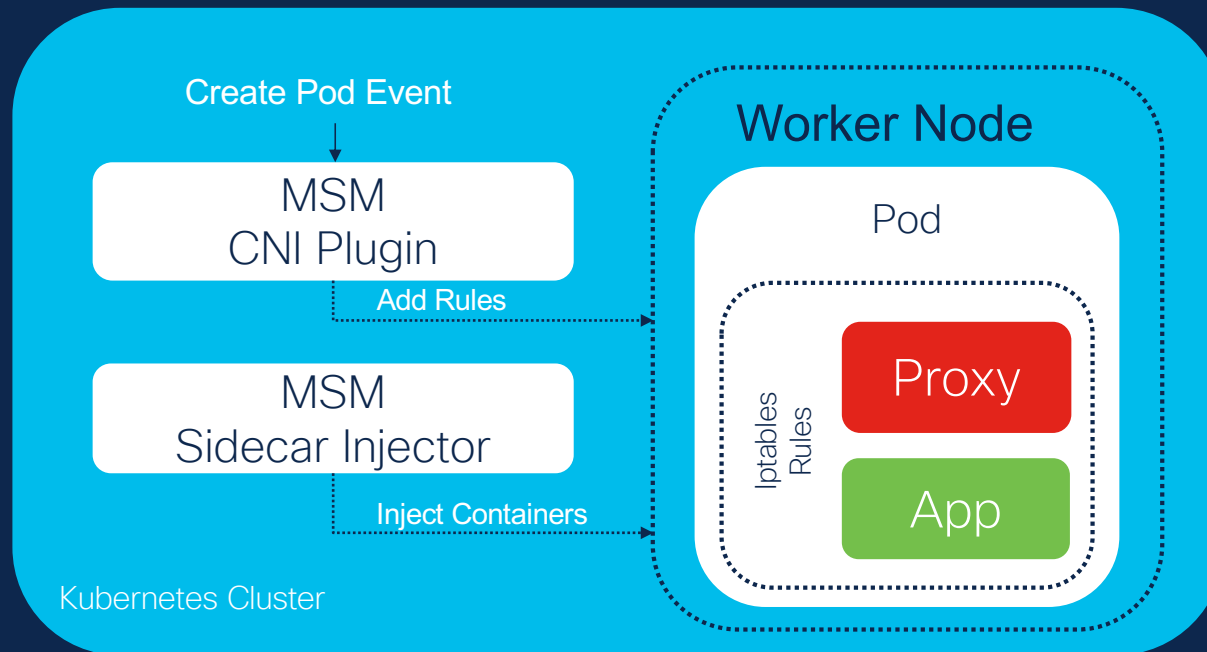


Video Surveillance Use-Cases for Media Streaming Mesh

- Large number of cameras
 - few per site in many small sites (e.g. retail)
 - large numbers in a few big sites (airports, factories etc.)
- Multiple viewers – probably remote from the camera locations
- One or more proxies per camera site and a proxy at each viewer site
 - Today's approach is typically RTP over UDP multicast



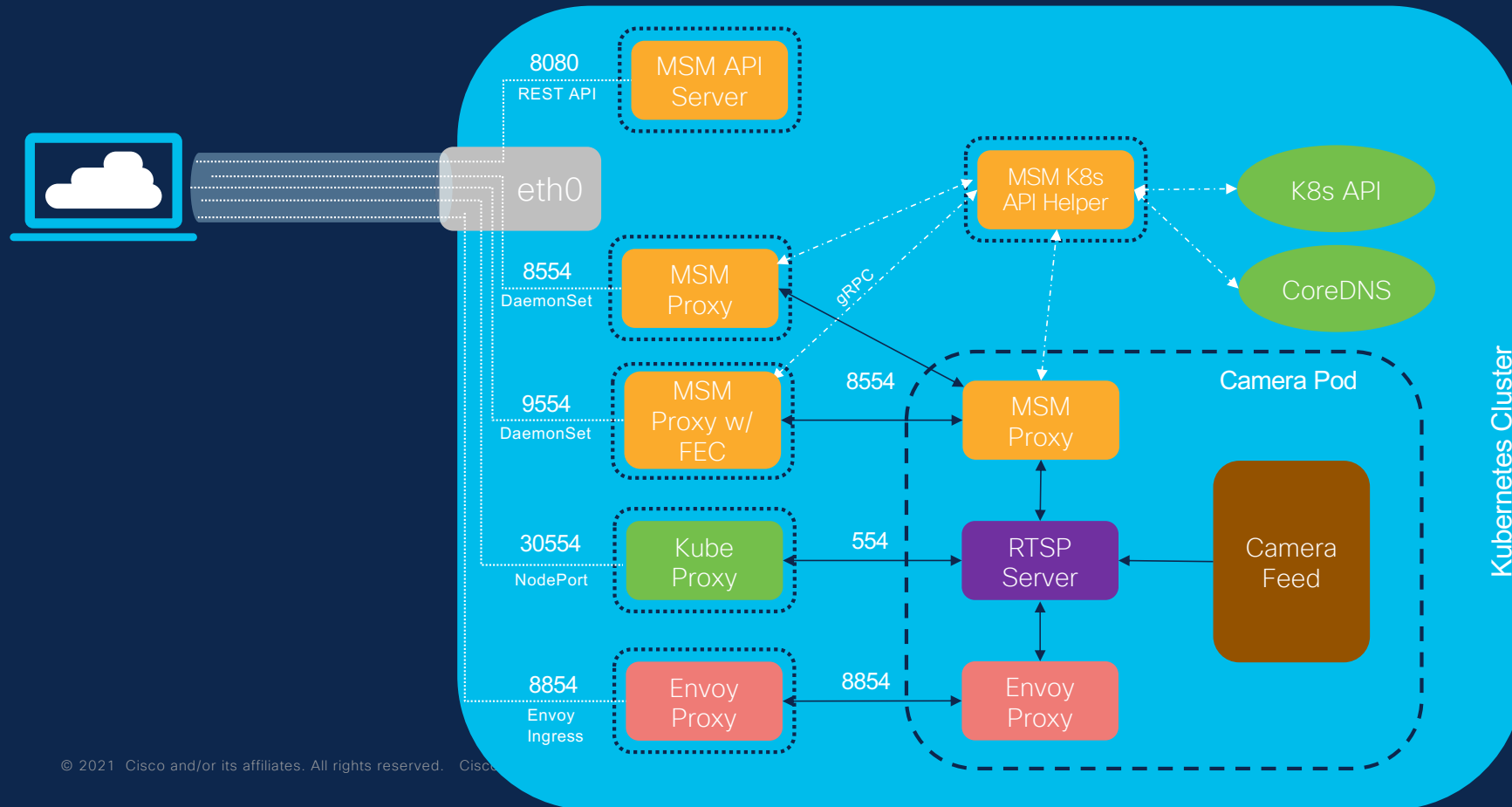
MSM RTSP Demo Software Architecture



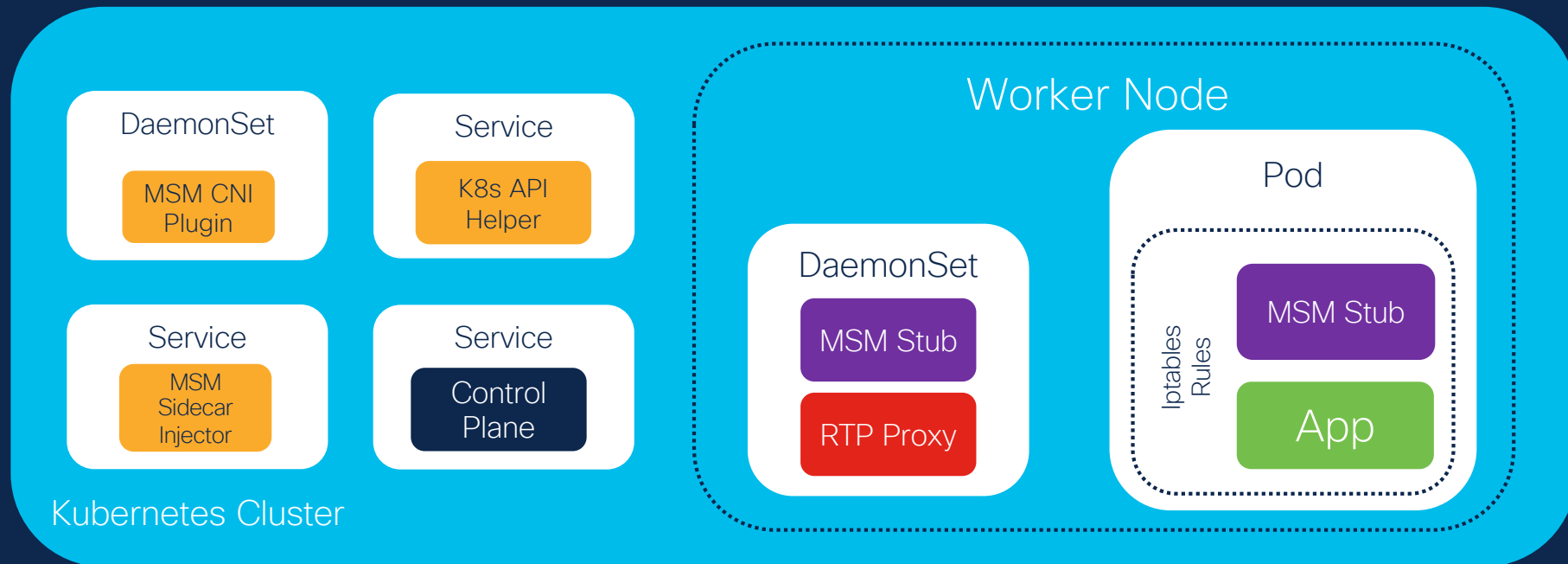
Flow of Events

1. Create Application Pod
2. MSM Sidecar Injector injects the necessary containers in the Pod
3. MSM CNI plugin operates as a chained plugin to other CNI plugins (container runtime invokes each plugin in order), allowing to program iptables rules without explicitly allowing the pod user for privileged access
4. Proxy/App containers start, traffic goes through proxy

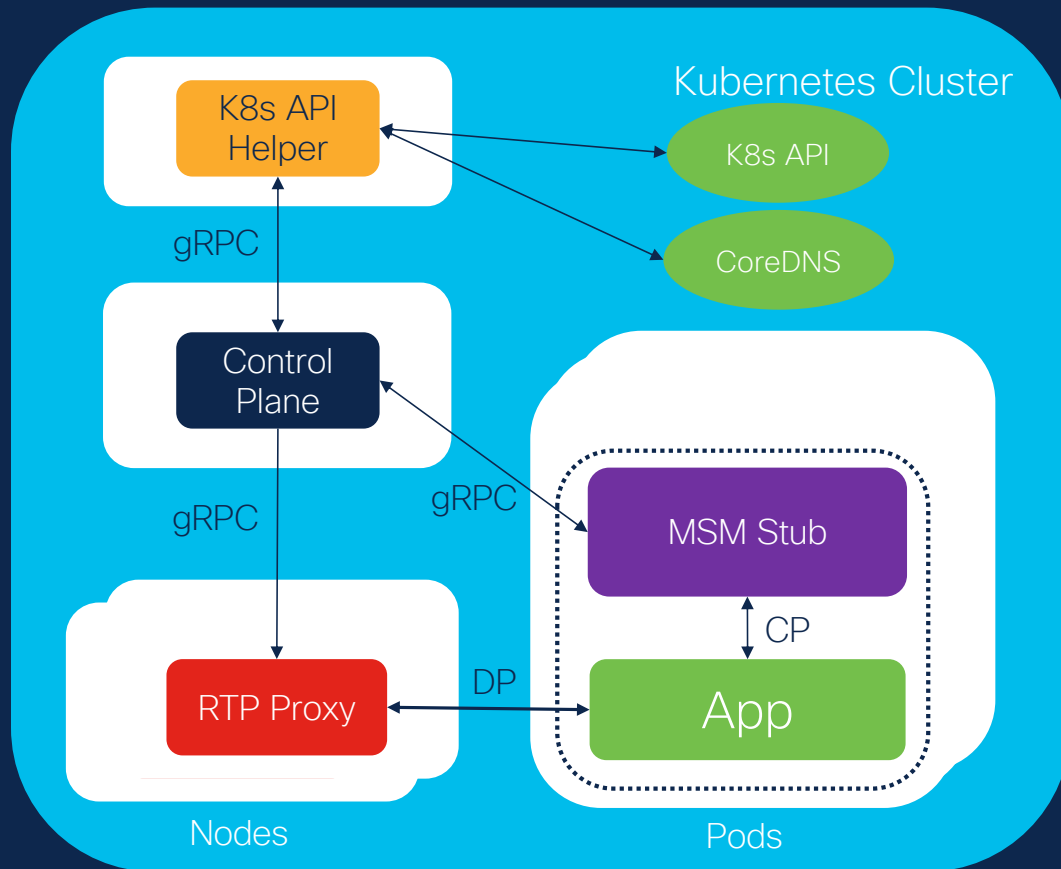
MSM RTSP Proxy Demo Setup



MSM Software Architecture



Per-Cluster Control Plane



Deploys as a Kubernetes Service

- Likely 3 replicas with RAFT etc.

Uses gRPC Southbound:

- To send/receive commands to/from the MSM Stubs
- To program the RTP Proxies (and potentially punt/inject, stats etc.)

Uses gRPC Northbound

- Integrates to K8s API Helper

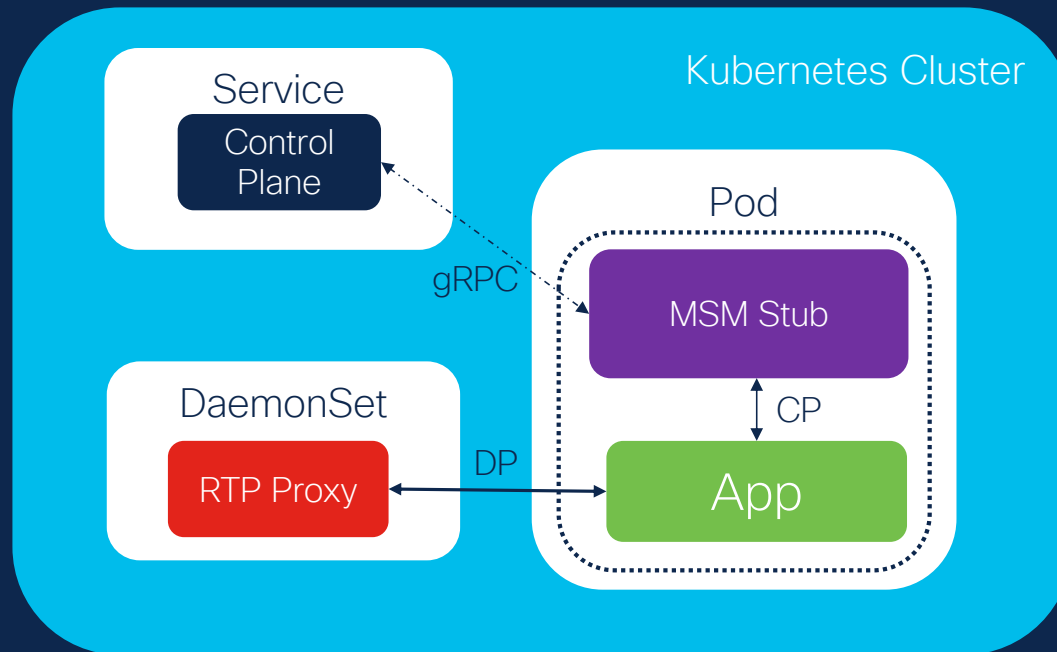
Written in Golang

- Leverage existing libraries (e.g. Pion)

L7 protocols implemented as plug-ins:

- RTSP
- WebRTC
- RIST
- SIP
- Others?

The MSM Stub



Deployed:

1. In each pod that uses MSM
2. With RTP Proxy as “gateway”

Terminates Control Plane

- Punts to per-cluster CP over gRPC

May intercept the Data Plane

- RTSP interleaved data case
- Memif towards VPP-based proxy
- Monitoring at pod
- “Live-Live” replication/de-duplication

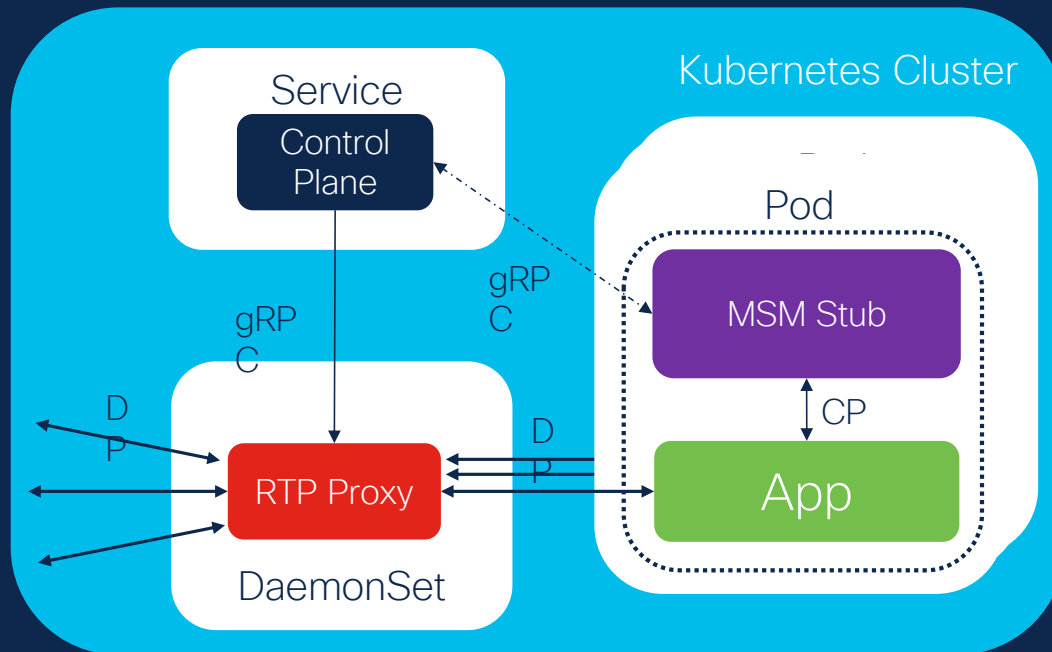
Potentially uses SPIFFE/SPIRE

- RTP Proxy does RTP/SRTP, will need end-to-end authentication

“Stub” because footprint is minimal

- Complexity is in the CP and the RTP Proxy
- Written in Rust to keep memory footprint low, and to avoid data-plane latency spikes due to garbage collection

The RTP Proxy



Deployed as a per-node DaemonSet

- supports North/South and East/West flows

Probably an RTP “Translator” (RFC3550)

- Unicast to multicast, IPv4 to IPv6, RFC1918 to public IP, tunnelling, MTU conversion etc.
- RTP/UDP, RTP/TCP and RTP/QUIC support
- Minimises attack surface

Golang and VPP implementations

- Golang using kernel sockets

Proxy with a Filter chain

- Replication, encryption, protection, congestion control etc.
- Key is to drive a filter ecosystem.

Can work with multiple CPs

- Spawn an instance per CP in golang case
- CP programs proxy using gRPC
- Proactive or reactive programming

RTP Proxy – Internal Architecture

