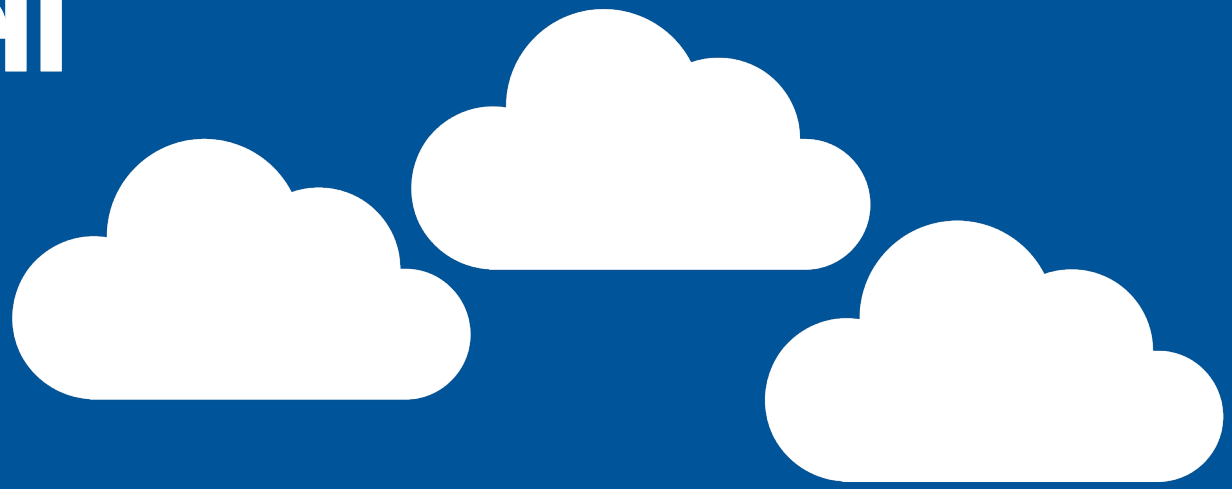


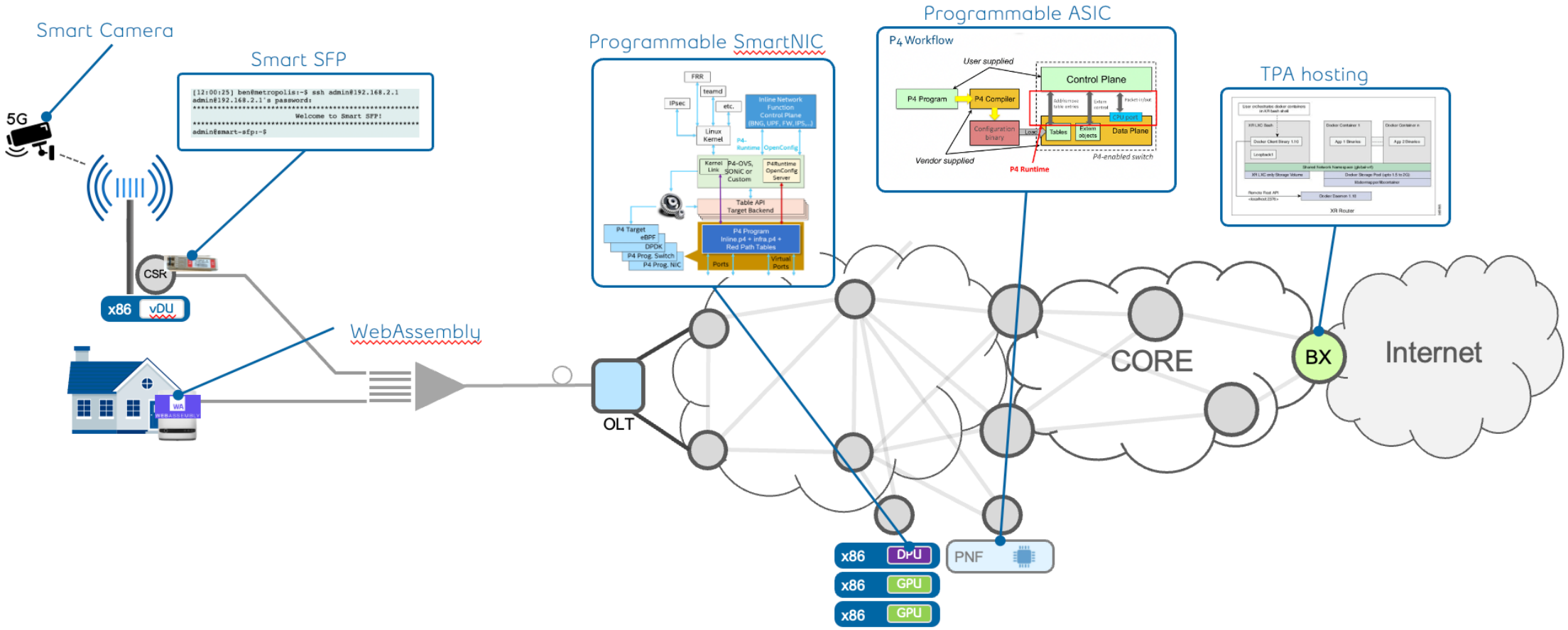
THE ALWAYS EVOLVING TELCO CLOUD

Daniel Bernier (Bell Canada)

IETF 120 - Telecom Cloud Services and Operations

Evolution Towards “Ubiquitous Computing”

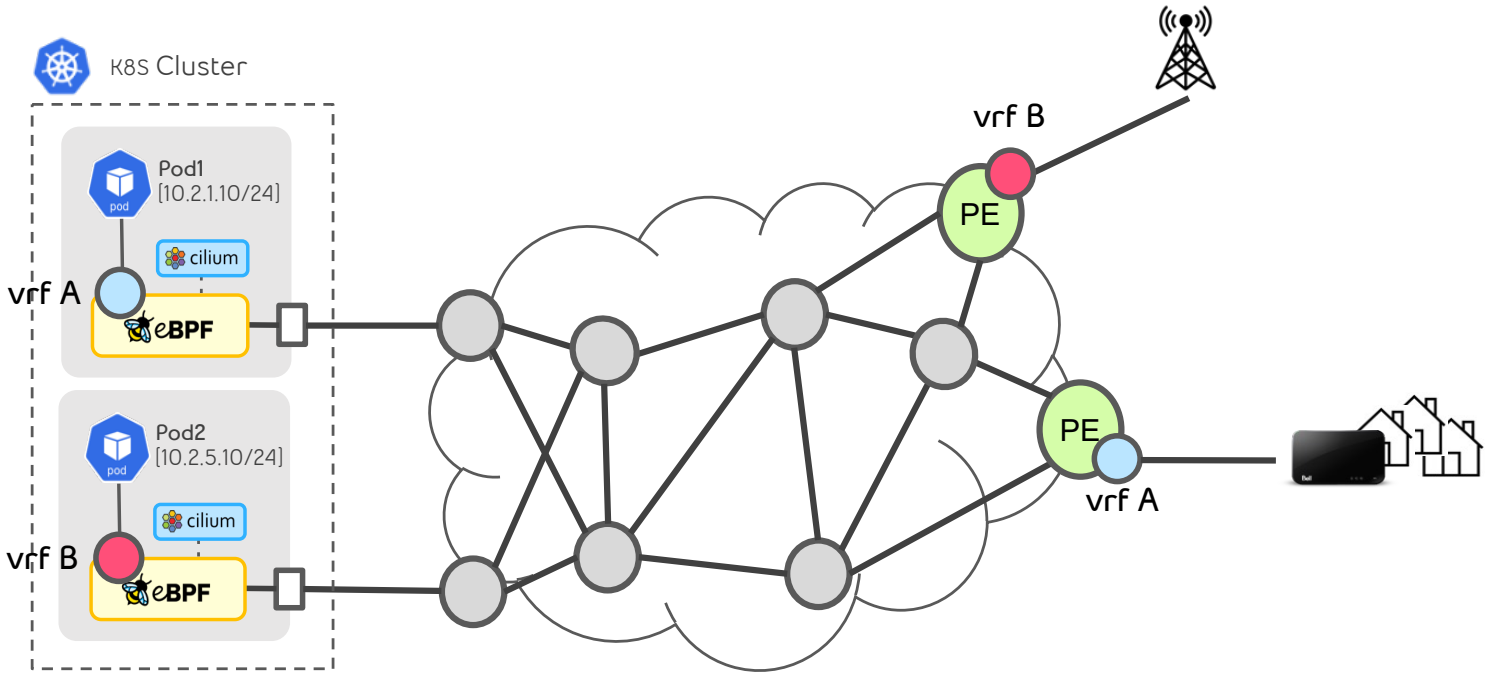
- Clouds are in constant evolution (now accelerated due to AI)
- Form factors and ”where” processing is performed will also be fluid.
- We are evolving towards In-Network computation and distributed scheduling of processing



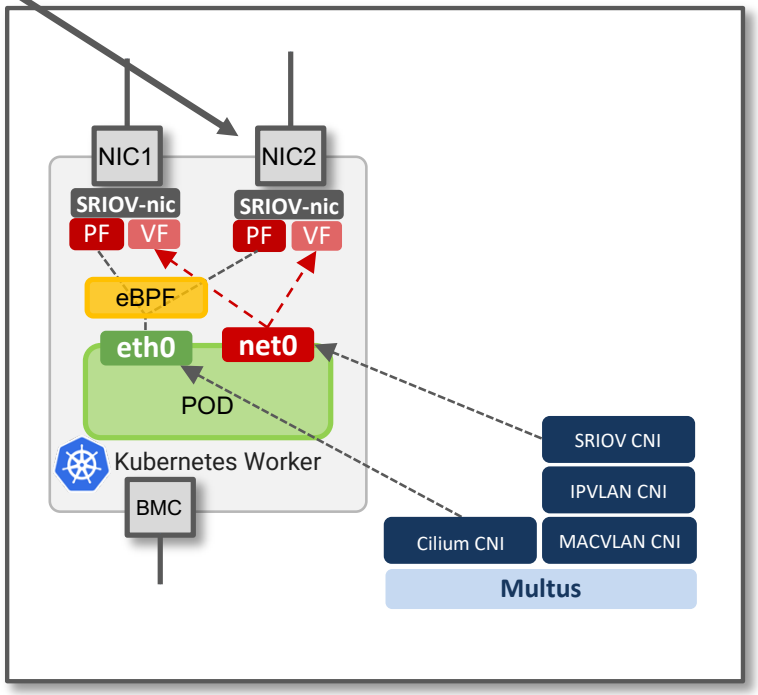
Simple, efficient E2E reachability is a must

The Famous Telco Workload Challenge

- How do work with network segmentation, high-throughput, multi-tenancy
- Currently being addressed by 3 approaches in CNCF (META-CNIs, KEP-3700, DRA)
- It was there before, got thrown into the spotlight with Telco workloads
- We are all being ingenious (although quite complicated) at solving the problem

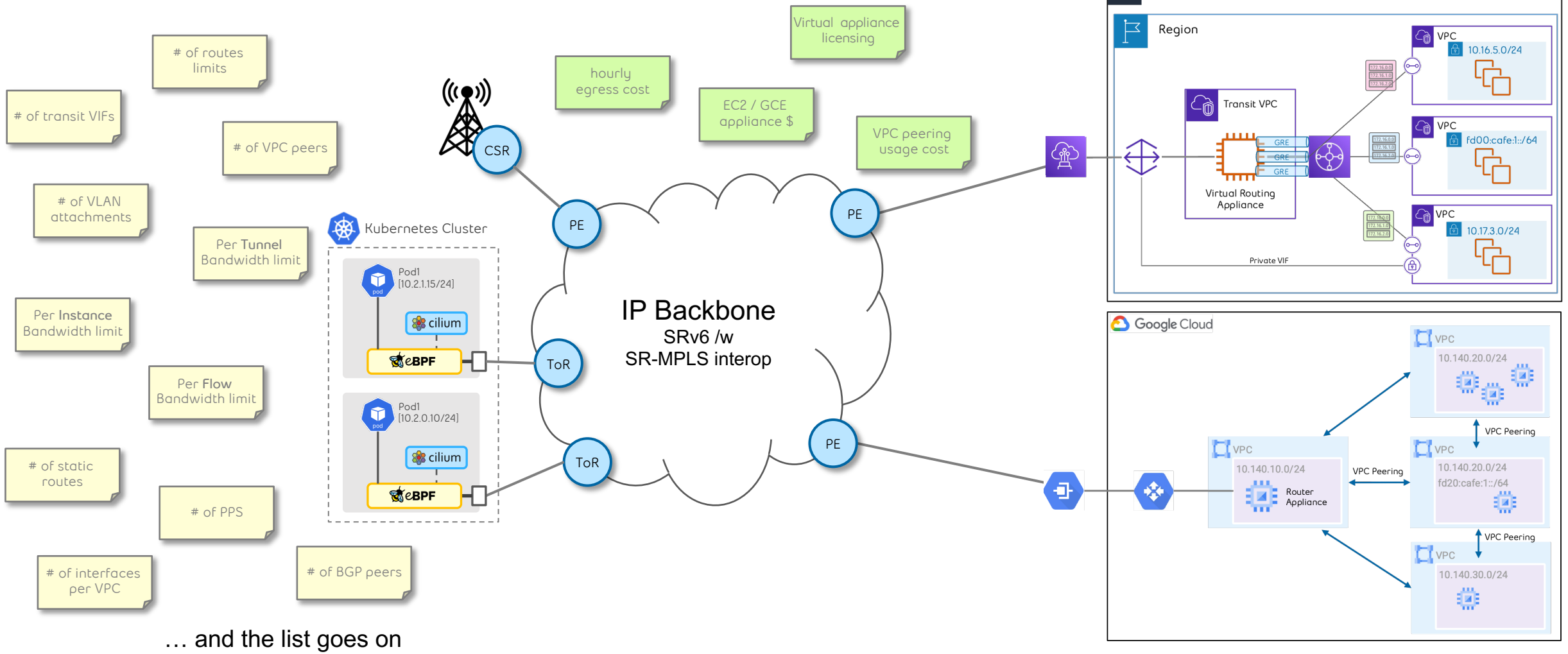


Do we really need all of this ?!



Multi-Cloud … Into the realm of “Limits, Quotas and FinOps”

- There is no such thing as a “free meal” when thinking about cloud networking … literally.
- Effectively extending an operator network to the cloud is nothing short of epic.
- Most 3rd party cloud networking solutions usually add even more complexity.



… and the list goes on

What if we tried to drastically simplify cloud networking instead ?

3.2 Gaps & Challenges in Today's Communication Services

The Network 2030 initiative is a structured approach to defining the capabilities of networks and corresponding communication services for the decade following 2030. The goal is to have networks ready for the market verticals that will utilize emerging technologies in 2030. Network 2030 extrapolates from what we know about technologies and develops a vision of the new media, new services, and new infrastructure. For this, we outline a few areas of importance to address towards building this vision.

Lacking service-network interaction: Failure to find an ordered and healthy relationship between the applications and the network has been a sticking point. Connectivity is just one of the workflows involved in application logic. It is a service consumed, with reachability being the only explicit means of setting up the application behaviour in the networks. However, a number of services offered by networks are not obvious to the end user. These include reliability of the network fabric, broadcast or multicast, integrity and security of data delivered, and network level awareness of congestion, capacity, and latency. Accommodating for such capabilities directly in the networks have been much of the focus of industry for the past thirty years. *Due to lack of direct support for such services through proper interfaces to the network, application developers have been left in a limbo, designing for every conceivable possibility of network failures and outages*. Many of these services are controlled over end-to-end interfaces between the endpoints using the transport (TCP) layer which is another example of free-form evolution aiming to solve the problems of the network without sufficient assistance from the networks.

Network 2030 – A Blueprint of Technology, Applications and Market Drivers Towards the Year 2030 and Beyond

Get the Network Out of the Way

Justin Pietsch Follow
Jan 21 · 15 min read

Getting the network out of the way has been very important for me in my thinking about networks, and is an easy way to help talk about a bunch of very important concepts, decisions, and arguments. This can sound trite, obvious, or insulting. In this post I'll try to describe the concepts I'm talking about and illustrate with some of most important examples in my career.

The more that the network is noticed the worse things are for everyone. Often times, especially when the network is noticed, networking and network engineers are thought of negatively. Instead, if you think of it as a challenge it can help you focus on making a great network. You can think about your goals: how important it is to keep the network working well, to not disrupt the business, and to be able to keep up with any changes that the business needs.

Invisinets: Removing Networking from Cloud Networks

Sarah McClure, Zeke Medley*, Deepak Bansal*, Karthick Jayaraman*, Ashok Narayanan†, Jitendra Padhye*, Sylvia Ratnasamy*†, Anees Shaikh†, and Rishabh Tewari**
**UC Berkeley †Google *Microsoft*

"In other words, *we believe that the best way for tenants to think about networking is to not think about networking at all.*"

https://www.usenix.org/system/files/nsdi23-mcclure.pdf

From Software Defined to Application Defined

- SDN is still fundamentally about packet delivery and connection management
  - Naturally, networking community optimizes for these concepts
- Application developers do not care about packets or connections; even the concepts are meaningless

Amin Vahdat - https://conferences.sigcomm.org/sigcomm/2021/workshop-nai.html#keynote-talk-2

How to Build For The Clouds

Application centric

- Ability to **encode network behavior** at the application level
- … encapsulate once, no header remapping
- Use **developer friendly** logic (i.e. spec/selector/label ’low latency’, ’video’. etc.)
- High-level **intent** versus imperative declaration (i.e. spec: ‘vrf blue’ not RT:577:1001)

Massive Simplification

- Brutally reduce underlay fabric protocol complexity (i.e. EVPN, EVPN-MH, etc.)
- Reduced overall orchestration overhead
- Reduced state requirements on underlay fabric

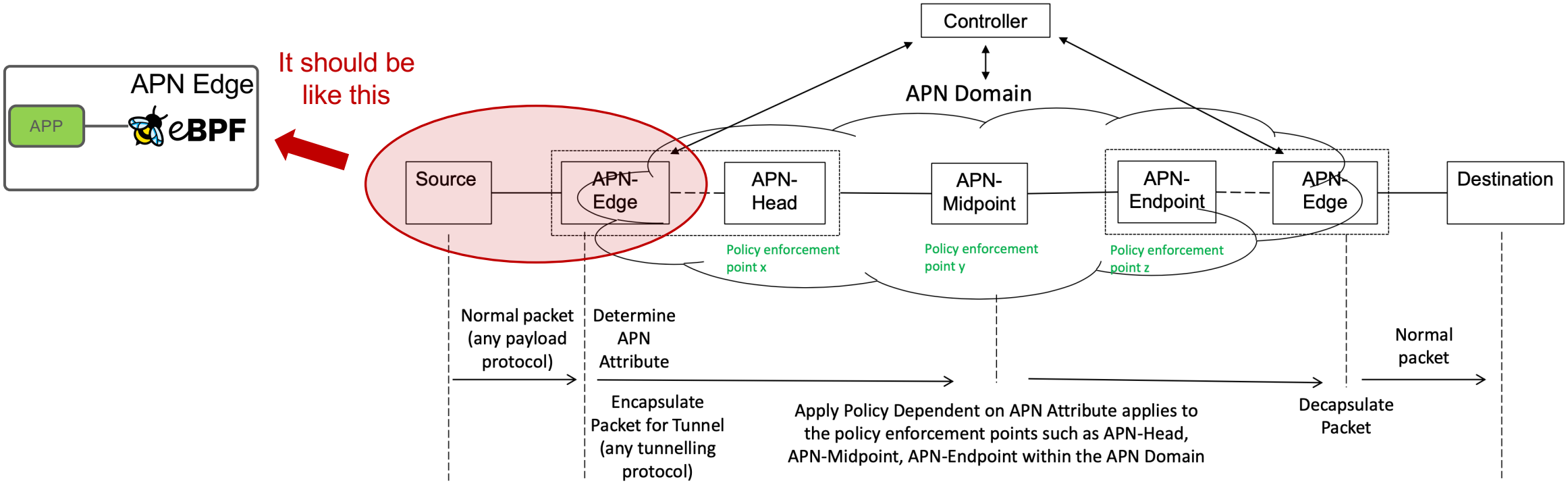
Massive Scalability

- Avoid exponential growth of underlay network (i.e. ISIS scale, BGP peers, etc.).
- Journey towards self-forming compute attachment
- Follow simplification rule of avoiding abuse of protocols … the “*Not everything needs to be solved by an IETF standard*”

Cloudified Control

- If you build for cloud … why not use cloud approaches
- Leveraging cloud-native principles versus monolithic controls

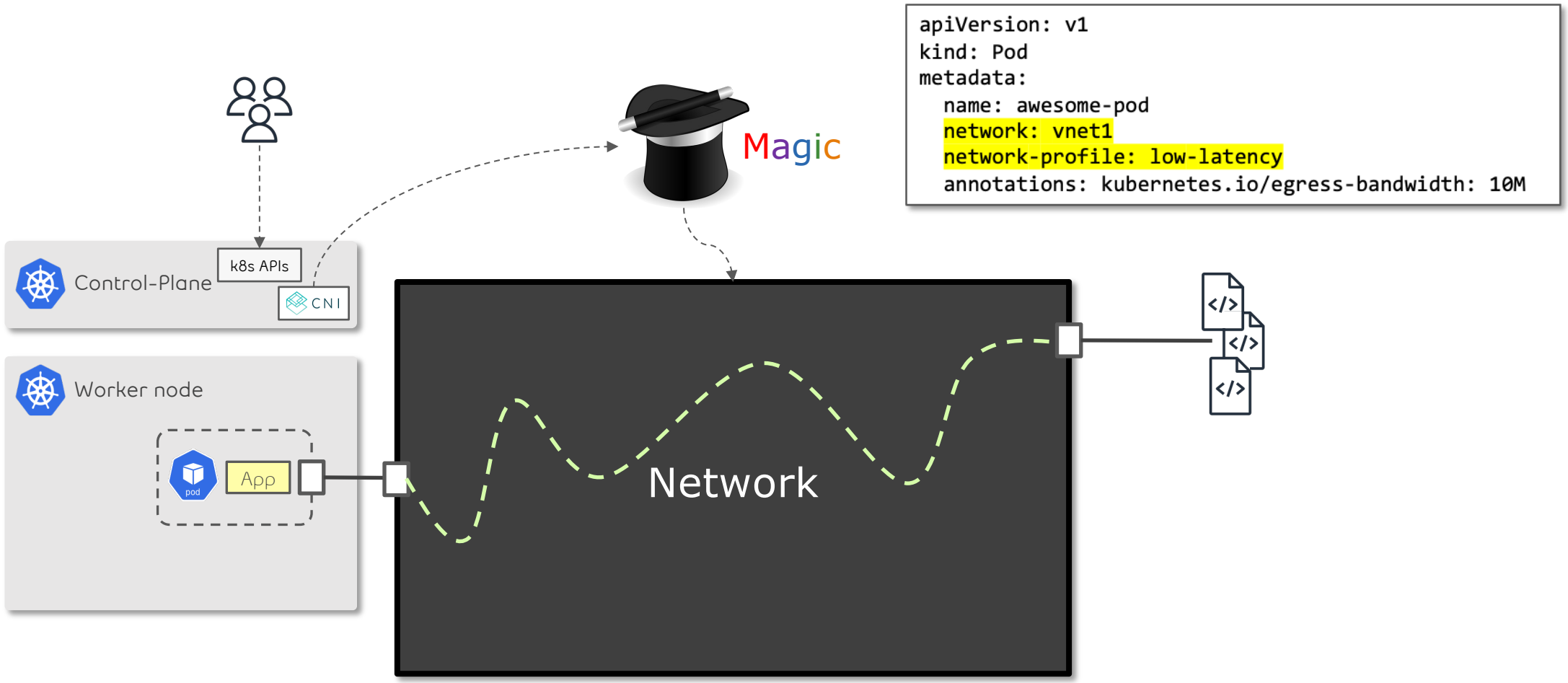
Application Oriented Networking Starts At the Socket



An APN Domain may span multiple network domains controlled by the same operator

Because in the end this is what we want

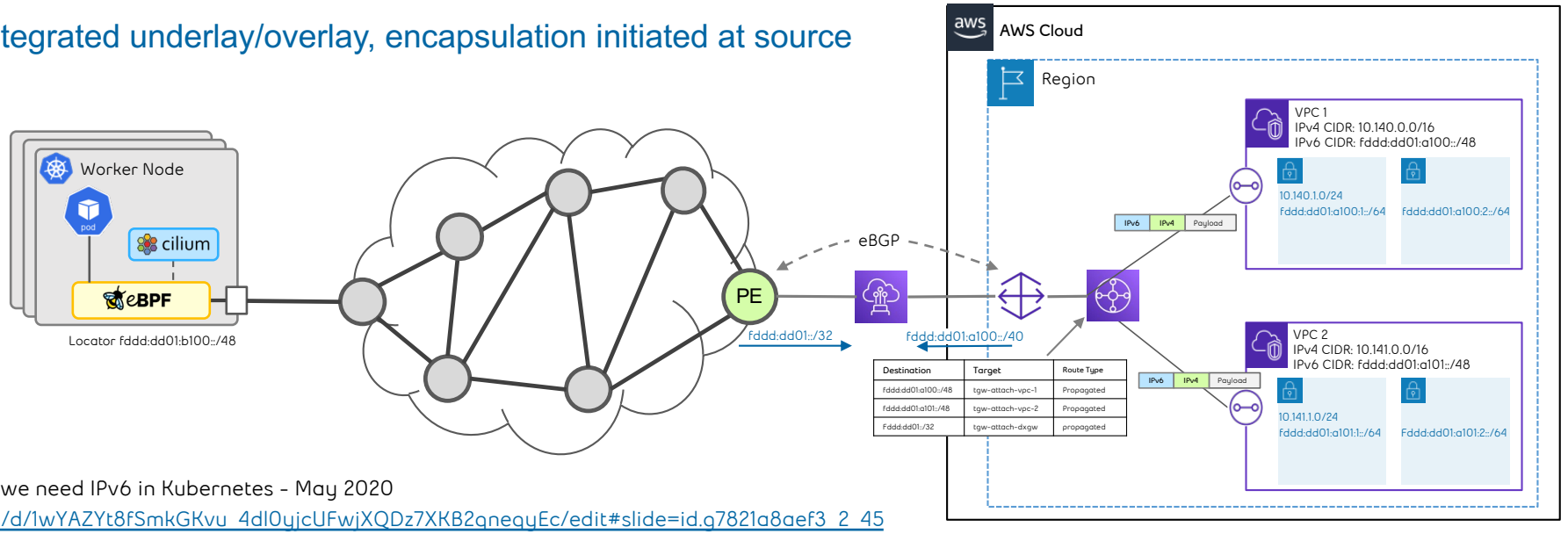
- Simple, declarative (Intent Driven) , developer friendly
- Evolving towards a *network spec* which represents more than L2/L3 definitions



Leveraging SRv6 to Evolve Cloud Connectivity

- Let's assume everything was IPv6 … would we need to use overlay VPNs ? *
- Existing technical or business constraints still require us to perform some sort of segmentation.
- SRv6 lets us achieve both end-to-end IPv6 WITH VPN service capabilities.
- When deployed at the upmost edge of an end-to-end flow (src/dst), SRv6 allows for MASSIVE cloud networking simplification.

SIMPLICITY	Cloud networks only needs to carry locator address space, no added complexity
COST EFFICIENCY	Reduced need for add-on cloud features or extending quota limits.
PERFORMANCE	Even without flow-label, SRv6 allows optimized flow performance (1 flow per uDT/DX or END function)
OBSERVABILITY	Integrated underlay/overlay, encapsulation initiated at source



* Ref: Nicola Leiva, Three reasons why we need IPv6 in Kubernetes - May 2020
https://docs.google.com/presentation/d/1wYAZYt8fSmkGKvu_4dl0yjcUFwjXQDz7XKB2qneqyEc/edit#slide=id.g7821a8aef3_2_45

Enabling SRv6 VPN End-to-End

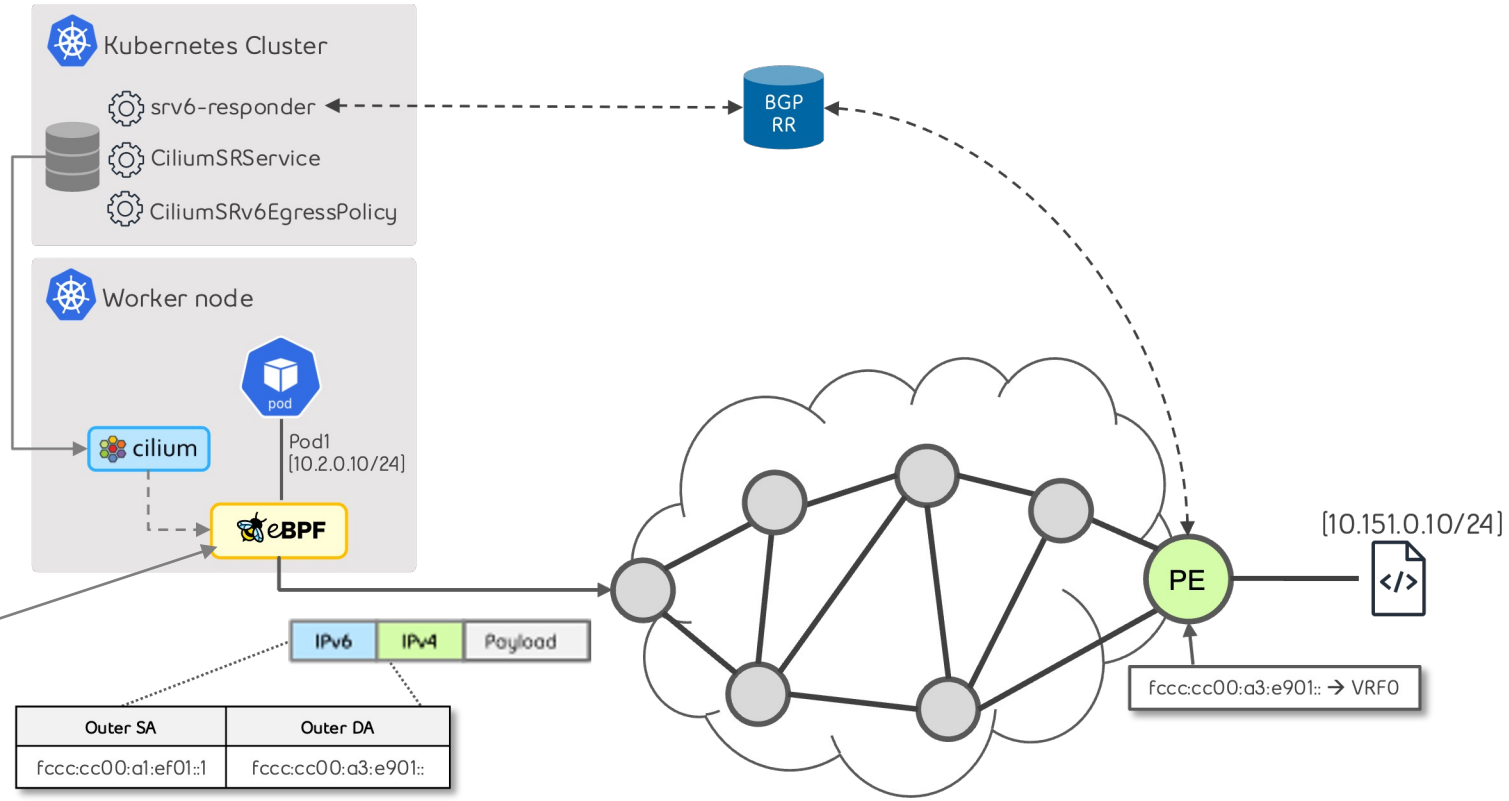
```
enterprise:
   srv6:
       locatorPoolEnabled: true
   srv6:
     enabled: true
   bgpControlPlane:
     enabled: true
```

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeeringPolicy
metadata:
  name: pe0
spec:
  nodeSelector:
     matchLabels:
     kubernetes.io/hostname: srv6-responder
  virtualRouters:
  - localASN: 64577
     exportPodCIDR: true
     mapSRv6VRFs: true
     neighbors:
     - peerAddress: “fccc:cc00:1:1”
     peerASN: 64577
```

```
apiVersion: cilium.io/v2alpha1
metadata:kind: CiliumSRv6VRF
name: vrf0
spec:
  vrfID: 1
  importRouteTarget: "64577:661”
  exportRouteTarget: "64577:661"
  rules:
  - selectors:
    - podSelector:
        matchLabels:
        vrf: vrf0
    destinationCIDRs:
    - 0.0.0.0/0
```

```
apiVersion: cilium.io/v2alpha1
kind: CiliumSRv6EgressPolicy
metadata:
  creationTimestamp: "2022-05-16T02:03:15Z"
  generation: 1
  name: bgp-ontrol-plane-…
  resourceVersion: "791"
  uid: f8519c91-d6d3-4cc8-941d-e5efd69e4317
spec:
  destinationCIDRs:
  - 10.151.0.10/24
  destinationSID: ‘fccc:cc00:a3:e901::’
  vrfID: 1
```

- POD has a single interface with only a default route.
- POD is associated with one or more VRFs through annotations
- CRD defines VRF attributes (RT, name, etc.) and BGP configuration parameters, etc.)
- SRv6 encapsulation done in eBFP via CiliumSRv6EgressPolicy which can be statically configured or auto-generated through insights such as BGP NLRI data.



Building SRv6 the Cloud Way … Git and ConfigMaps

sidpool.yaml 361 B

```yaml
apiVersion: isovalent.com/v1alpha1
kind: IsovalentSRv6LocatorPool
metadata:
  name: pool0
  labels:
    # This label will be used in the next step
    export: "true"
spec:
  # behaviorType: Base
  behaviorType: uSID
  prefix: fccc:cc00:0800::/40
  structure:
    locatorBlockLenBits: 32
    locatorNodeLenBits: 16
    functionLenBits: 16
    argumentLenBits: 0

```

bgppeer.yaml 1.09 KiB

```yaml
apiVersion: "cilium.io/v2alpha1"
kind: CiliumBGPPeeringPolicy
metadata:
  name: rr
spec:
  nodeSelector:
    matchLabels:
      # kubernetes.io/hostname: acm1
      bgppeer: rr
  virtualRouters:
  - localASN: 64577
    srv6LocatorPoolSelector:
      matchLabels:
        export: "true"
    exportPodCIDR: true
    mapSRv6VRFs: true
    neighbors:
    - peerAddress: "fccc:cc00:a::1/128"
    # - peerAddress: "fc00:1::254/128"
      # eBGPMultihopTTL: 255
      peerASN: 64577
      families:
      - afi: ipv4
        safi: mpls_vpn
      # - afi: ipv4
      #   safi: unicast
      # - afi: "ipv6"
      #   safi: "unicast"
    - peerAddress: "fccc:cc00:800::253/128"
      # eBGPMultihopTTL: 255
      peerASN: 64577
      families:
```

srv6-covenml.yaml 318 B

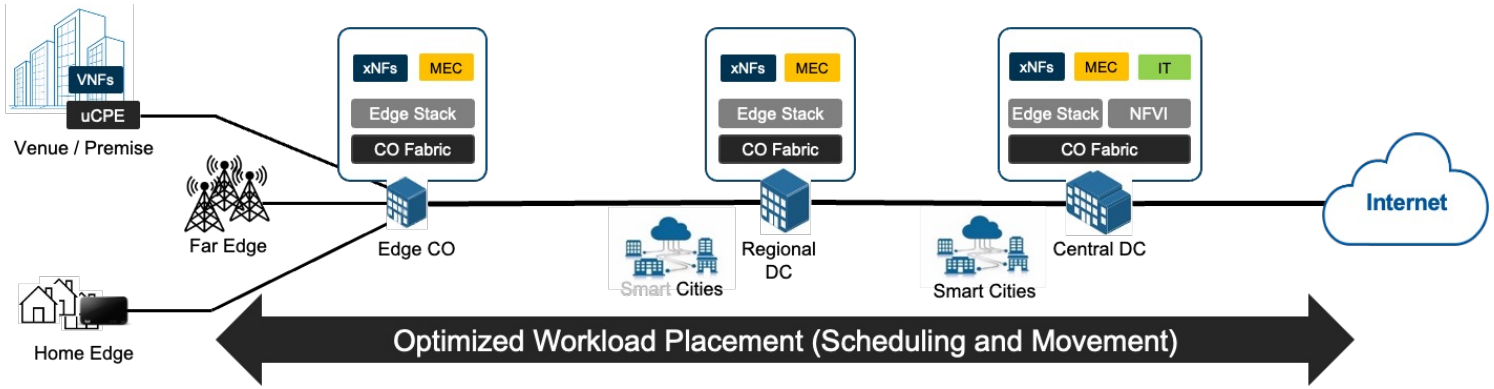
```yaml
apiVersion: cilium.io/v2alpha1
kind: CiliumSRv6VRF
metadata:
  name: covenml
spec:
  vrfID: 20
  importRouteTarget: "960:6960"
  exportRouteTarget: "960:6960"
  locatorPoolRef: pool0
  rules:
  - selectors:
    - podSelector:
        matchLabels:
          vrf: covenml
    destinationCIDRs:
    - 192.168.137.128/25

```

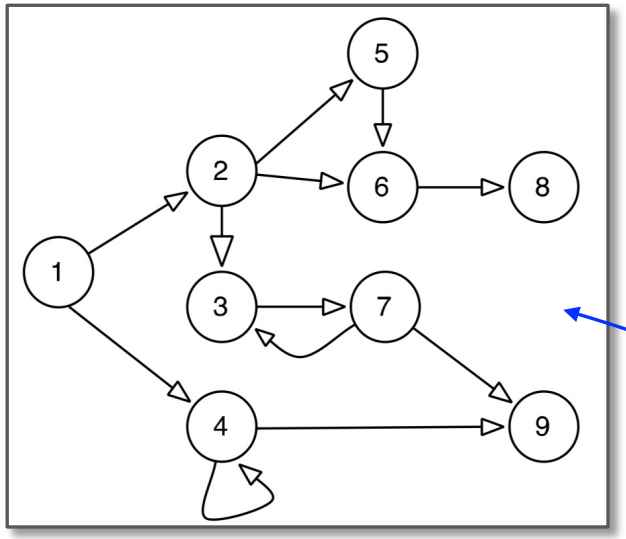
pod-covenml.yaml 263 B

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: covenml
    vrf: covenml
  name: covenml
  namespace: default
spec:
  containers:
  - command:
    - sleep
    - "10000000"
    image: alpine
    imagePullPolicy: Always
    name: test
  restartPolicy: Always
```

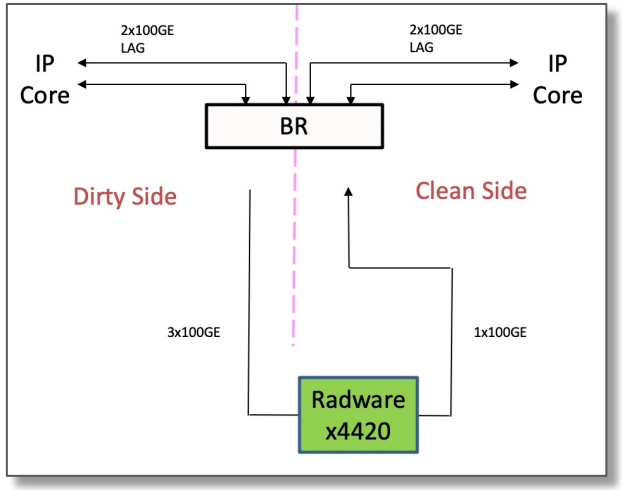
Service Composition Problem Statement

- Real Deployable Network Service Composition is Hard !
- We largely still rely on
  - ACL / Interface based classification
  - MacGyver “like” networking tricks (dot1q, ToS, route leaking, etc.)
- Every SD-WAN overlay flavor comes with its own chaining magic
- Still based on linear patterns while application flows are not
- Still no reconciliation between *network* and *application* awareness
- Nor the local versus remote and/or multi-cloud dilemma.
- Service scaling is vertical, growth or feature add requires physical box replacements
- How do we reconcile the existing, high $ PNF devices already in the field ?
- Services are physically inline and mostly centralized (hinders ECMP, orchestrated reconfiguration, etc.)
- Service composition always hindered by the weakest link in the chain



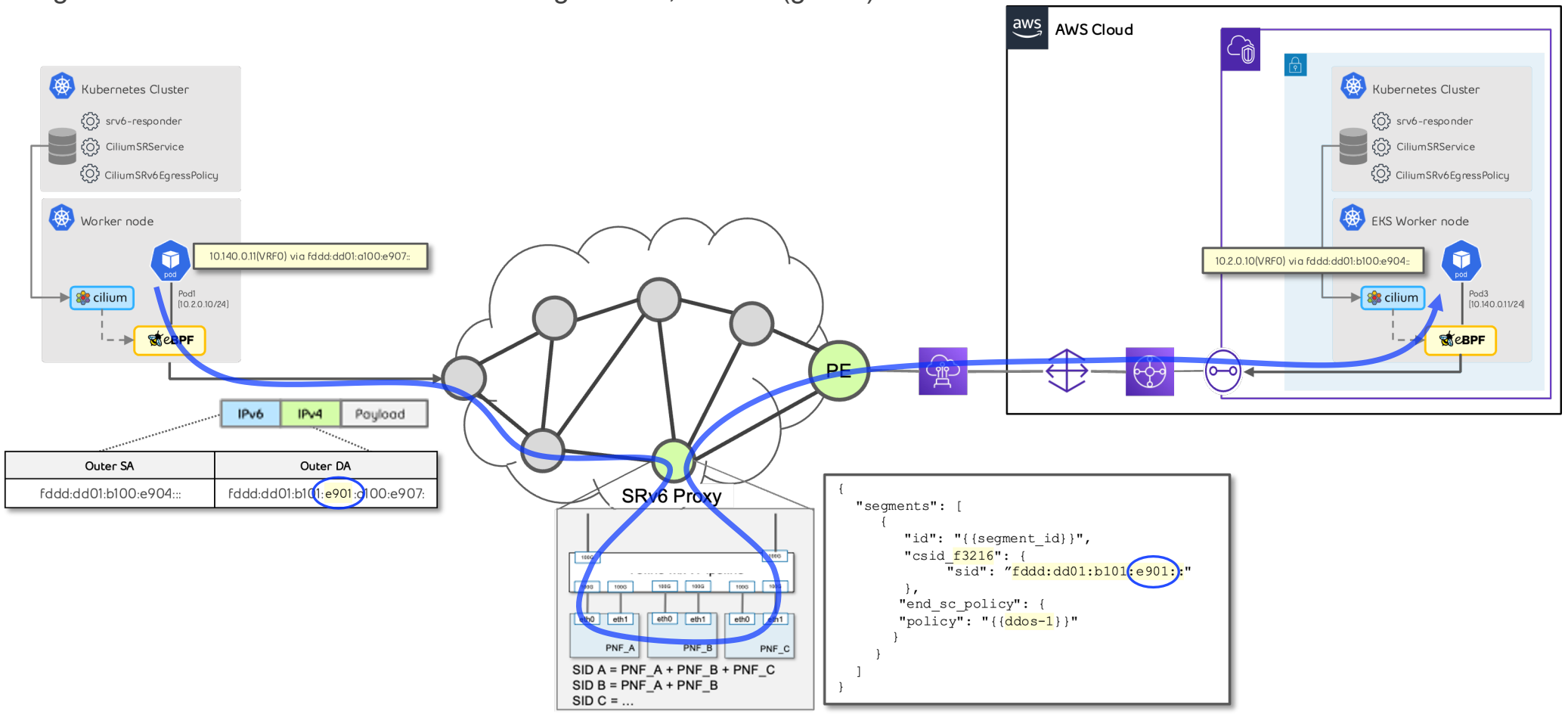


Strange Resemblance to Service Meshes ?!



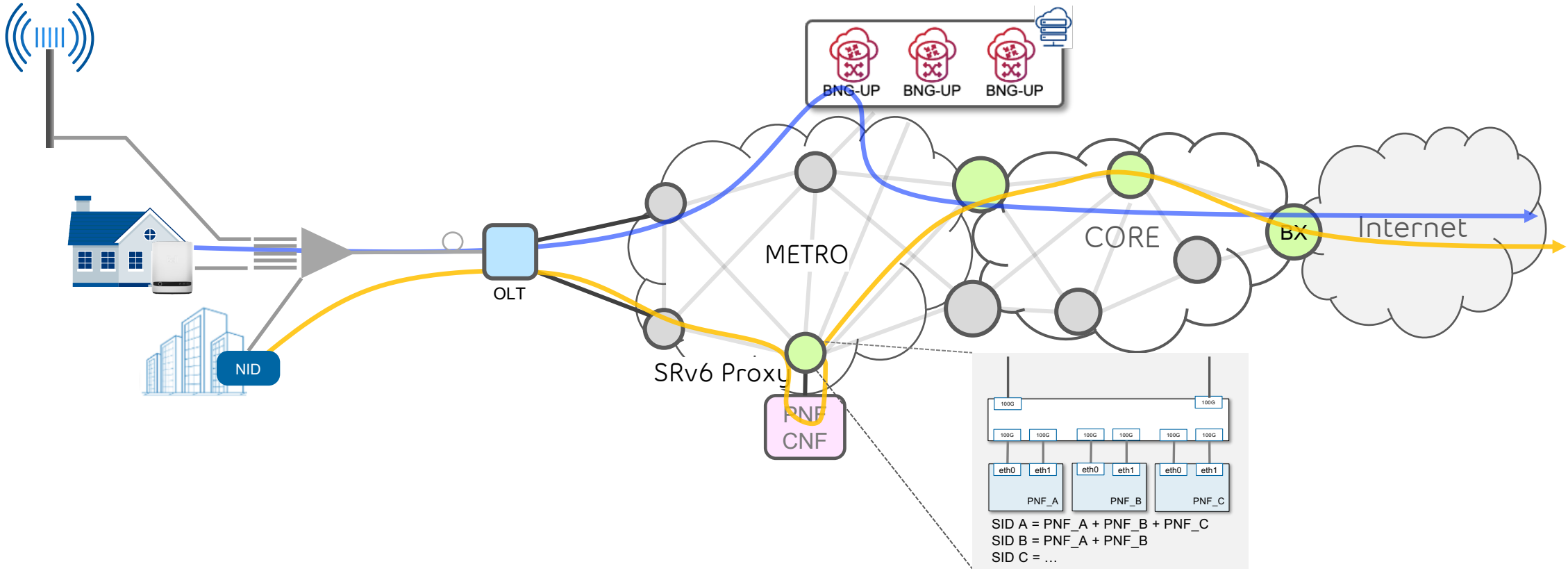
Coming full circle … cloud-to-cloud inline services

- Coming full circle with our previous work on P4/P4RT based SRv6 “in-network” service proxy.
- Can easily insert logically inline services through SRv6 service programming policies.
- Services can be inserted at any location in the network and steered using SRTE explicit paths.
- Paths can be configured via NETCONF or API driven through CRDs, SL-API (gRPC) or PCE.



Fixed or Wireless In-Network Service Insertion

- Same principle applies to fixed or wireless connectivity
- Alignment with BBF WT-474 to make it standardized



Thank You