

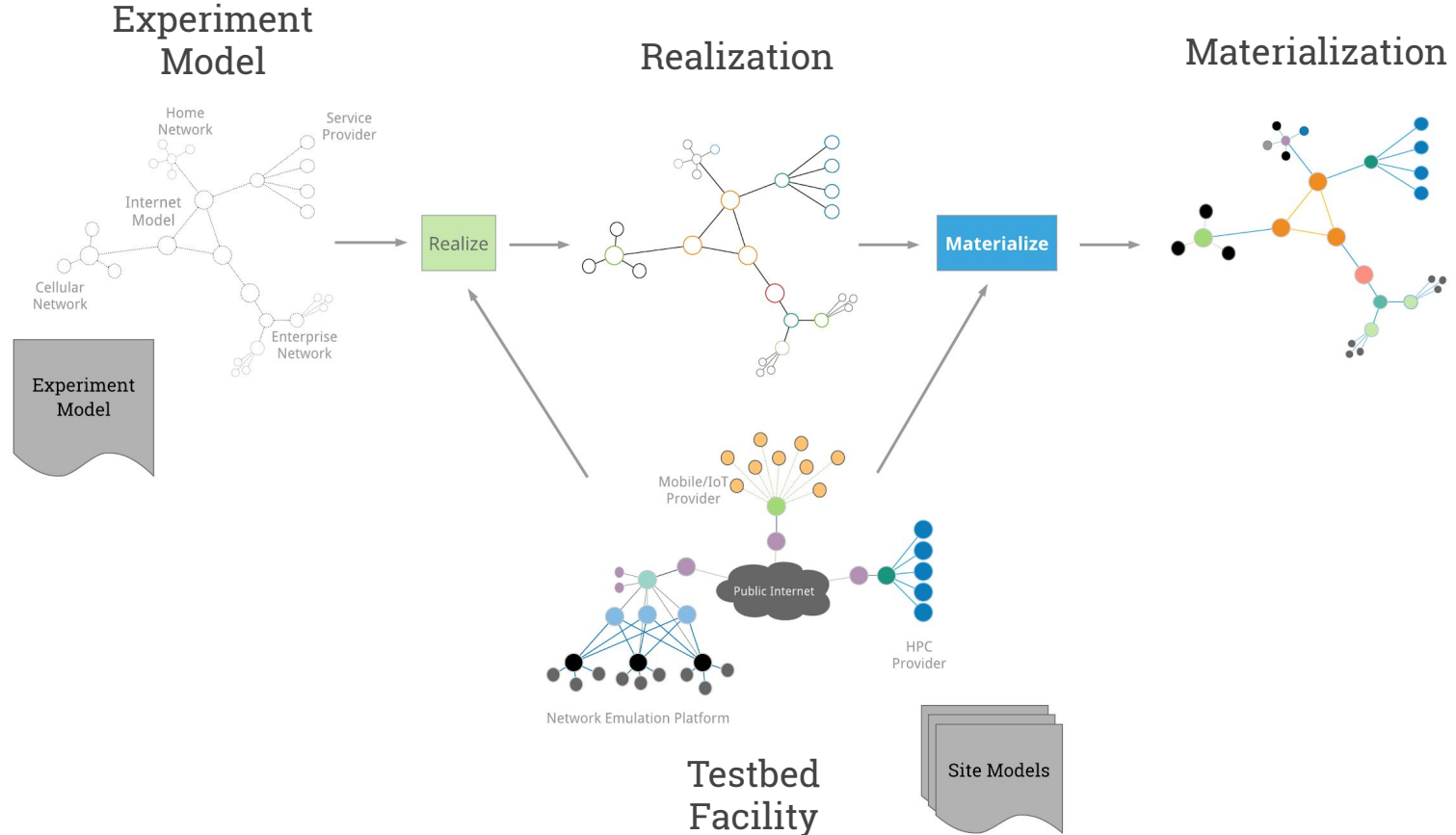
The DComp Testbed

Ryan Goodfellow, Stephen Schwab, Erik Kline,
Lincoln Thurlow and Geoff Lawler

Outline

- Context: DCompTB is a Merge testbed facility.
- Testbed hardware design and physical networks.
- EVPN-Based experiment networks.
- Infrapod experiment services.
- Network emulation.
- Experiment materialization runtime.
- Modular technology stack.
- Virtual testbed development environment.

Some Quick Vocabulary



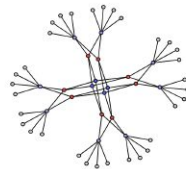
DCompTB as a Merge Testbed Facility

- The Merge Portal manages the experiment development process.
- Testbed facilities need only focus on materializing experiments.
- Facilities publish a testbed model to a merge portal and implement the merge driver interface.
- Facilities are implemented as an orchestrated set of self-contained software components

Merge Portal



Multiuser
Collaborative
Portal



Experiment
Modeling and
Realization



Experiment
Development
Environments &
Experiment
Access

Testbed Facility



Physical
Hardware &
Networks



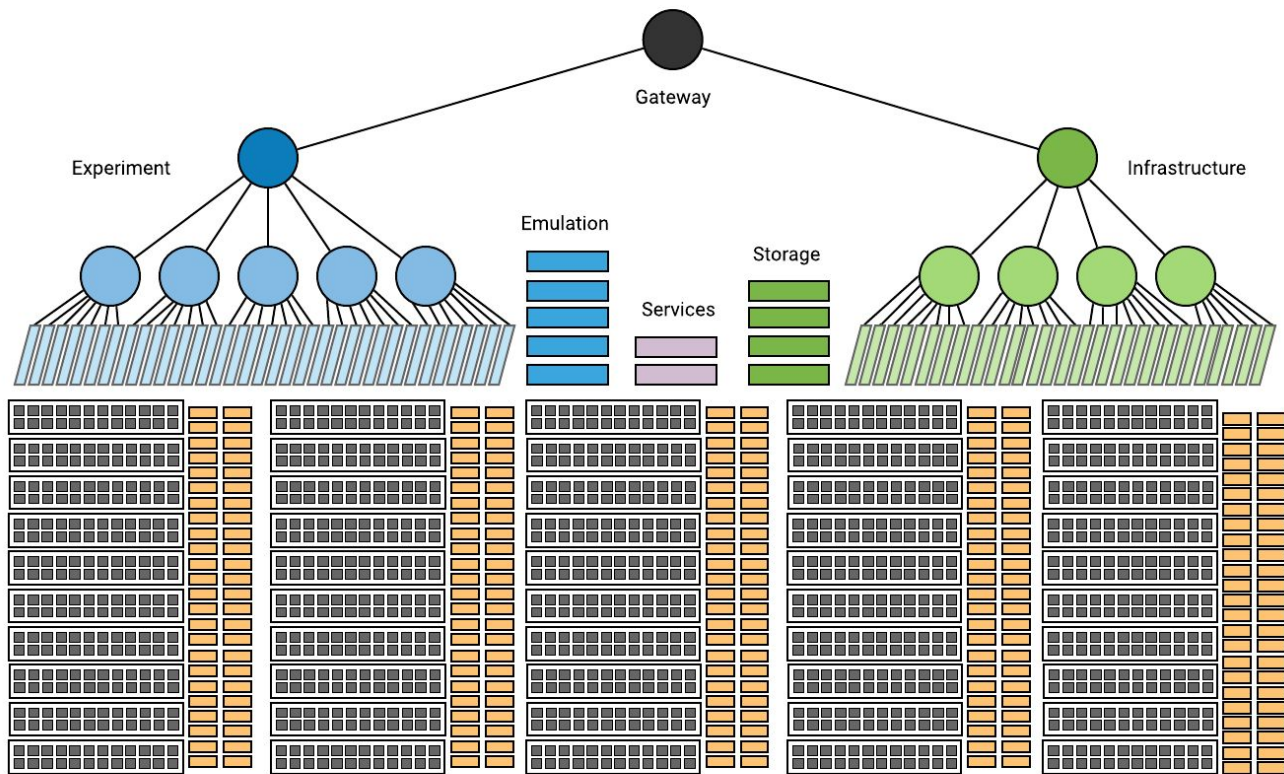
Modular Testbed
Building Blocks



Materialization
Orchestration

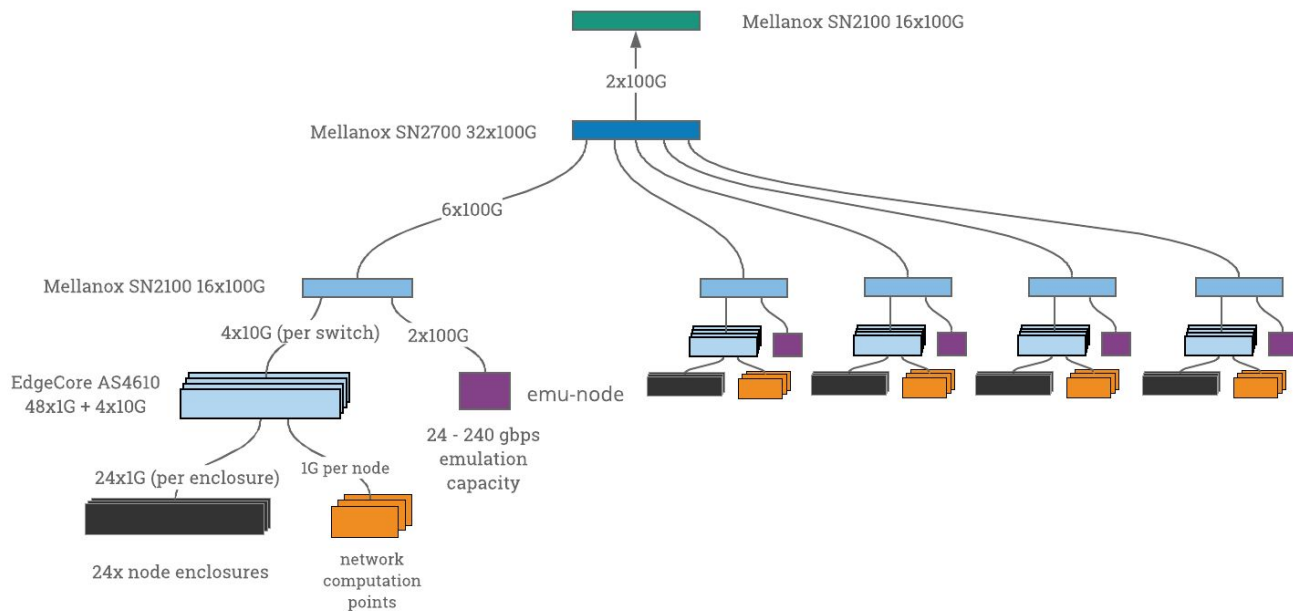
Testbed Hardware Design and Physical Networks

- 1440 nodes
 - 1200 minnow
 - 240 rohu
- 77 switches
- 5 emulation servers
- 4 storage nodes
- 2 infrastructure servers



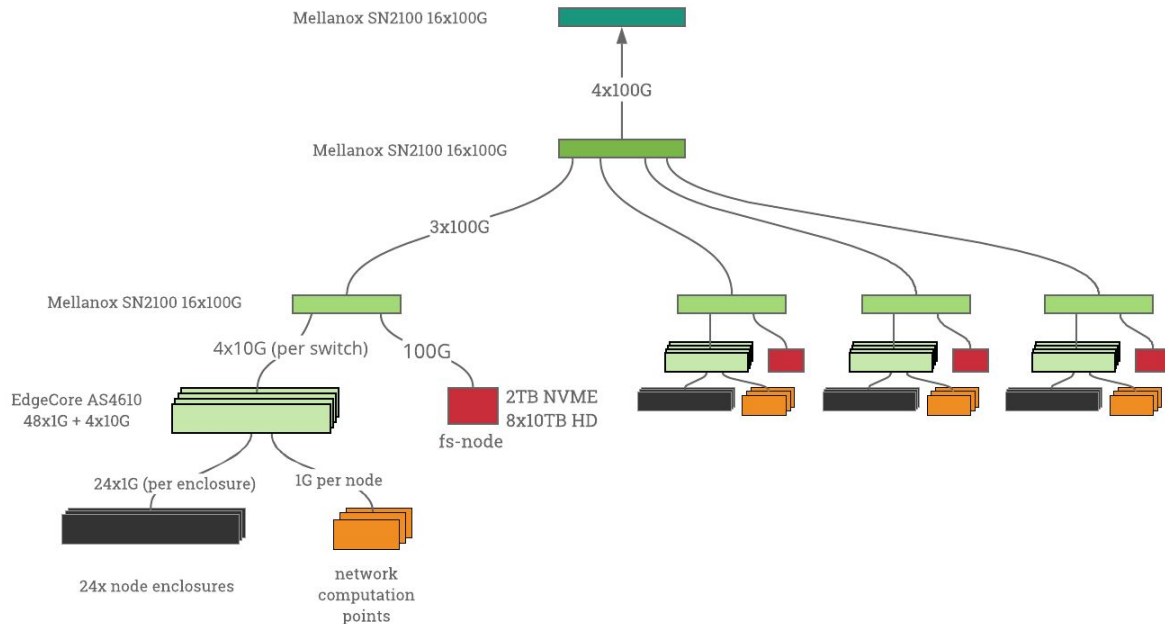
Testbed Hardware Design and Physical Networks

Experiment Network



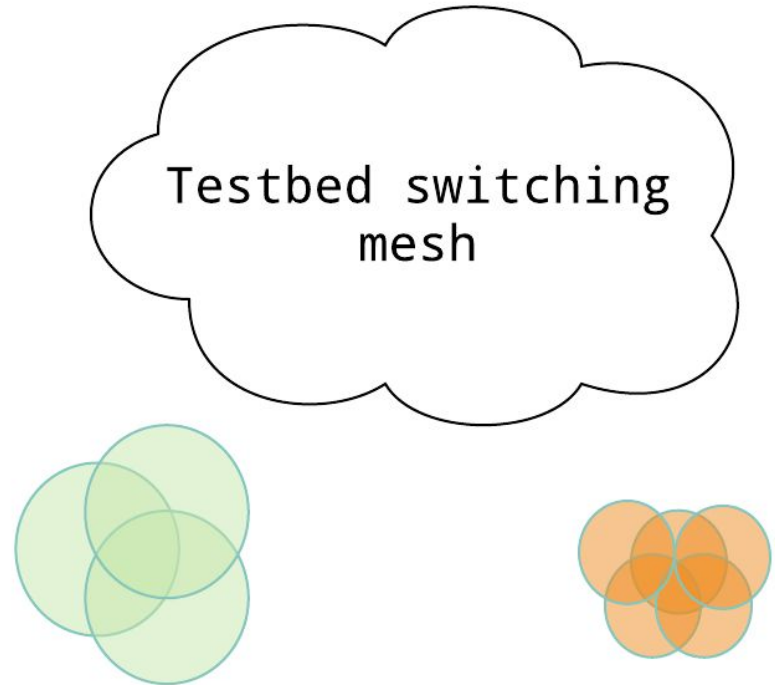
Testbed Hardware Design and Physical Networks

Infrastructure Network



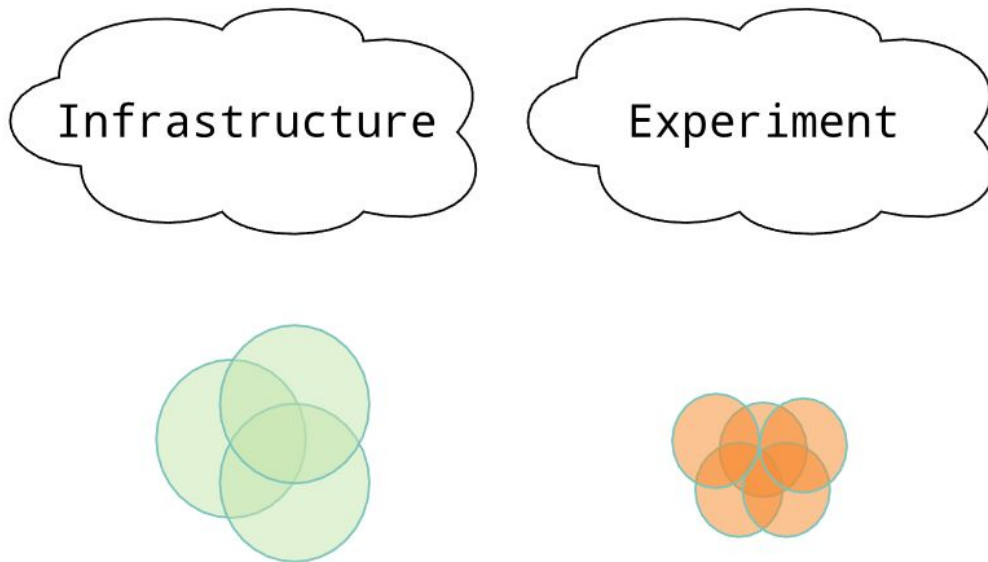
EVPN Testbed Networks

- Beginning life of a testbed
- Interesting devices exist
- Network equipment exists
- Let's make a testbed!



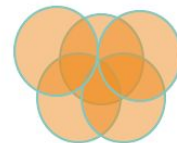
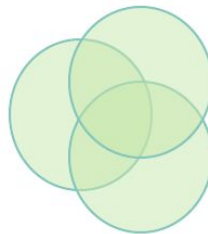
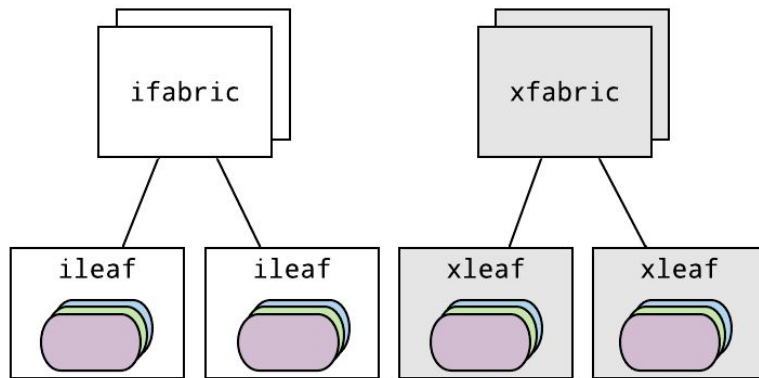
EVPN Based Experiment Networks

- Separation of experiment infrastructure networks from the networks over which experiments execute is critical
- Don't want to introduce artifacts from orchestration, remote file systems, etc...
- Experiments are just that **experiments**, if you're not destroying your experiment network on the first go - whatever you're doing is boring.
- Need a network to function independent of what's going on in the experiment network.
- Experiment networks are not generally flat - this is problematic for automation.



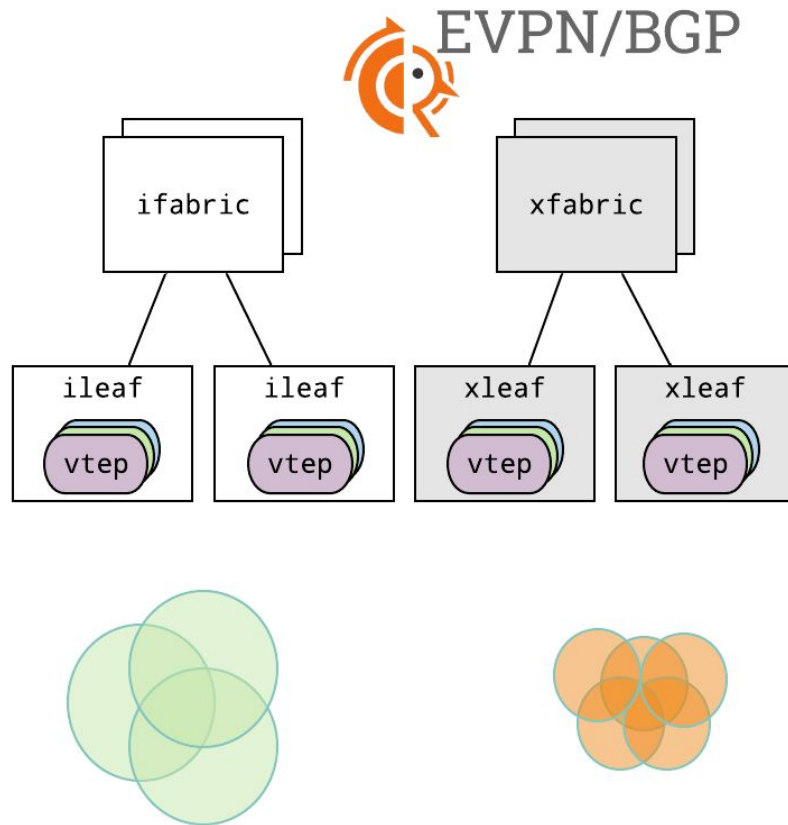
EVPN Based Experiment Networks

- Making things a bit more concrete than clouds, we have a set of leaf switches connecting to nodes and fabrics interconnecting leaves
- We need to provide isolation between experiments.
- Could use vlans, but experience has shown this to be problematic for a number of reasons.
 - Crossing routers presents issues.
 - Multicast support across switches and tunnels is problematic.
 - How about experiments that need to themselves tunnel, QinQ? Not viable for all sub-protocols, and stacking has issues for complex switching meshes.



EVPN Based Experiment Networks

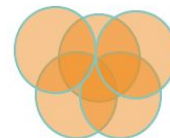
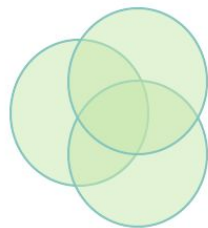
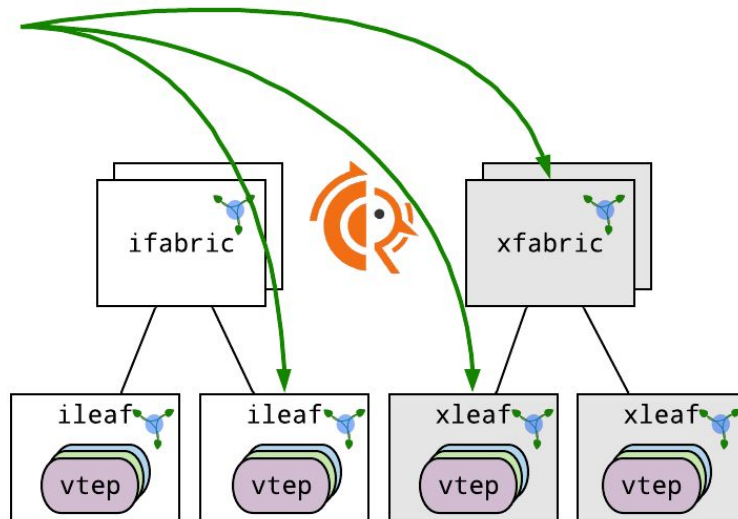
- Enter Ethernet virtual private network (EVPN)
- A multiprotocol border gateway protocol (MP-BGP) implementation that provides a control plane for VXLAN.
- VXLAN: similar in spirit to VLAN with the following significant differences
 - Full packet encapsulation L2 in (L3 in L2).
 - Clean underlay overlay model - underlay is L3 and routable just like normal L3.
 - 24 bit address space vs VLAN 12 bit (only 4096 identifiers avail)
- EVPN provides several route types we only use at 2
 - macadv: *mac X is reachable at VTEP P*
 - multicast: *VTEP P is reachable at addr A*



EVPN Based Experiment Networks

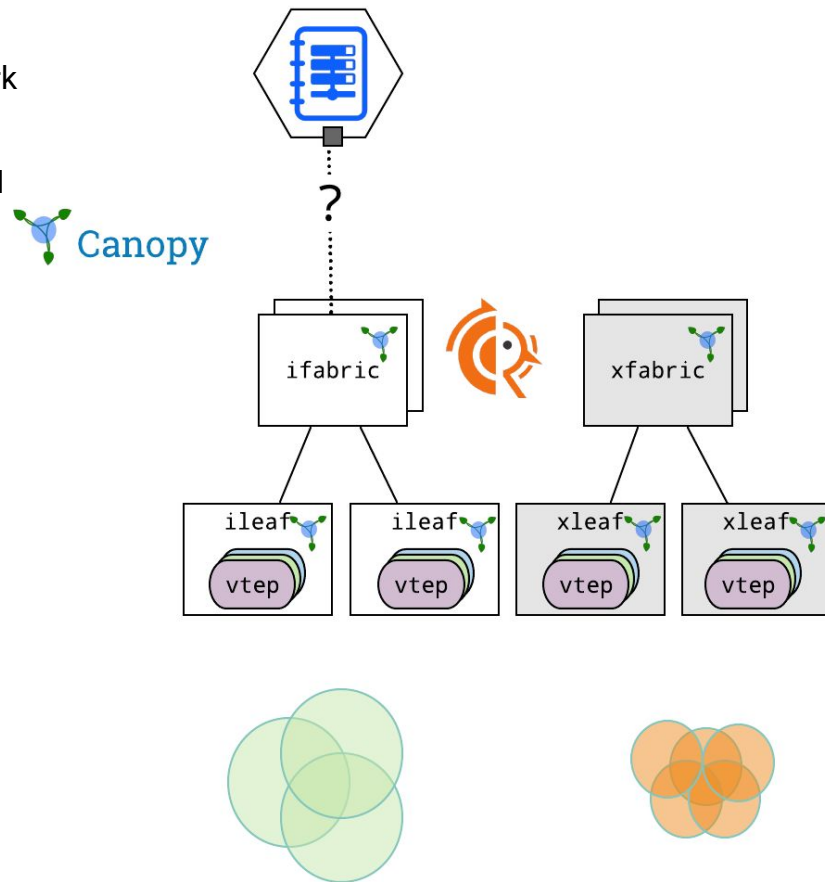


- Now that we have a virtual network isolation mechanism, let's make some virtual networks.
- **Canopy** is a virtual network synthesis system
- Client server model
 - Client runs at a sensible place in infrastructure with management access to network appliances
 - Servers run on network appliances
- Servers expose gRPC API for managing virtual network synthesis functions
 - VTEP create/destroy/parametrization
 - VLAN create/destroy (for internal VTEP plumbing and managing VLAN/VXLAN boundary networks)
 - MTU management
 - Port links state management



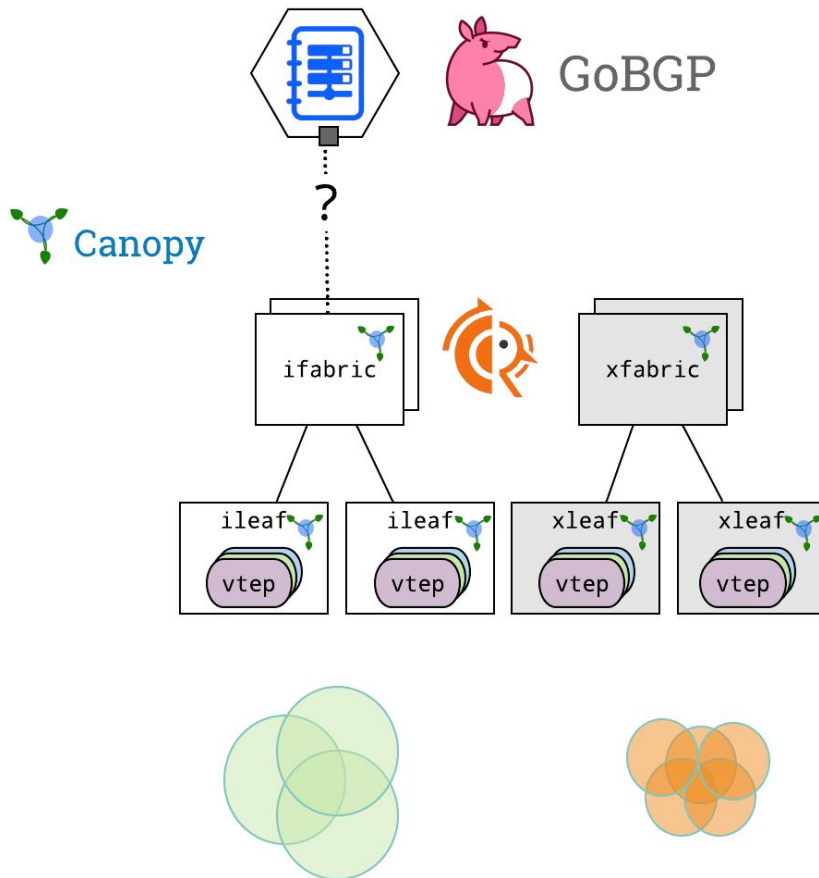
EVPN Based Experiment Networks

- Now we have isolated connectivity, but still lack basic network services for infrastructure networks.
- So we need DHCP and DNS, how do we hook it into our EVPN system
- The testbed switches run FRR's bgpd, so let's just run that.
- A few issues
 - Again no real API
 - FRR's self-compatibility is a zoo
 - Quagga variants
 - Cumulus variants
 - Pure FRR variants
 - Different BGPD/Zebra combos do things differently



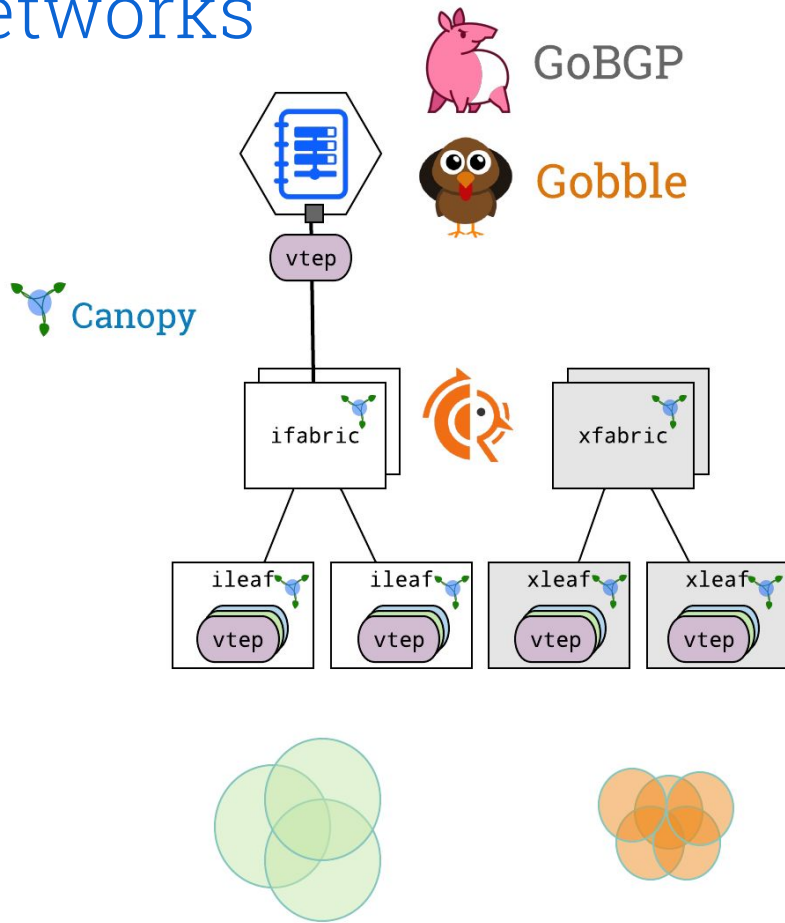
EVPN Based Experiment Networks

- GoBGP to the rescue.
- A BGP upper half daemon written in pure Go.
- Has a gRPC API
- Great EVPN support
- Active development community that's great to work with
- But it's only an upper half, meaning it will speak MP-BGP to its neighbors, but to be useful we also need to update our own forwarding tables in response to this information.



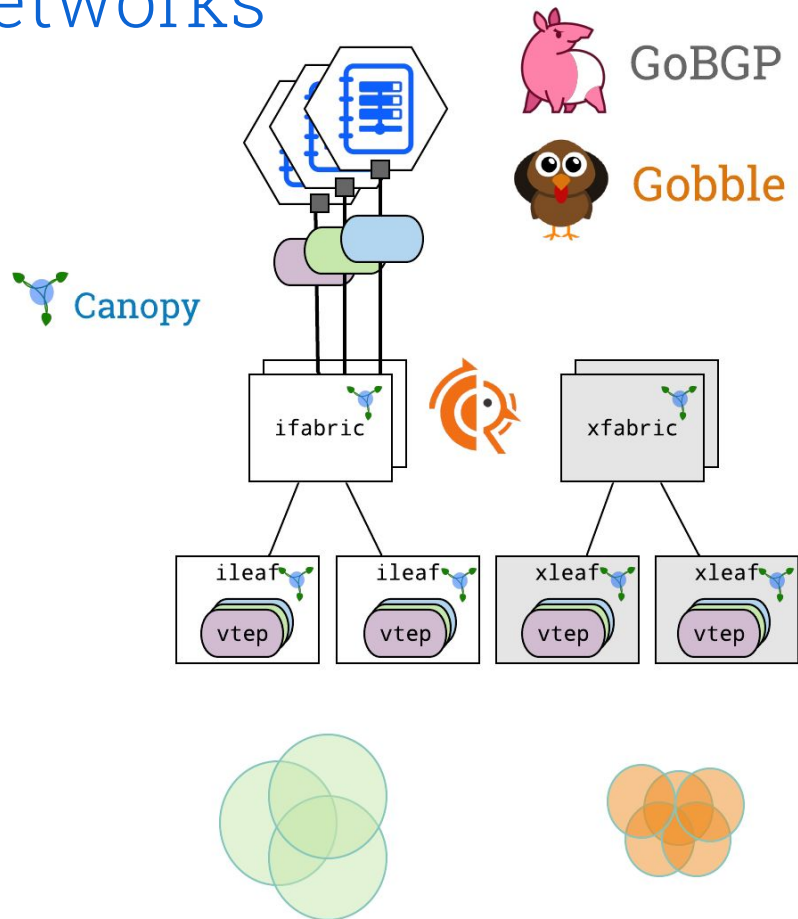
EVPN Based Experiment Networks

- So we developed Gobble, a lower half for GoBGP
- Uses the GoBGP gRPC to poll for macadv and multicast evpn routes
- Updates the kernel's neighbor, routing and bridge forwarding tables through netlink (we've also written an rtnetlink library in Go as we find many of our tools need to talk to the kernel through netlink)
- So now when we want to provide a service on a specific experiment's infrastructure network, it's as simple as advertising a mac address, standing up a VTEP and GoBGP/Gobble automatically takes care of the rest.
 - What mac address you say? Good question.
 - A few slides ago I snuck a hexagon around the Nex logo. That represents a container and the little box at the bottom represents a veth pair, with a MAC. More on this later.



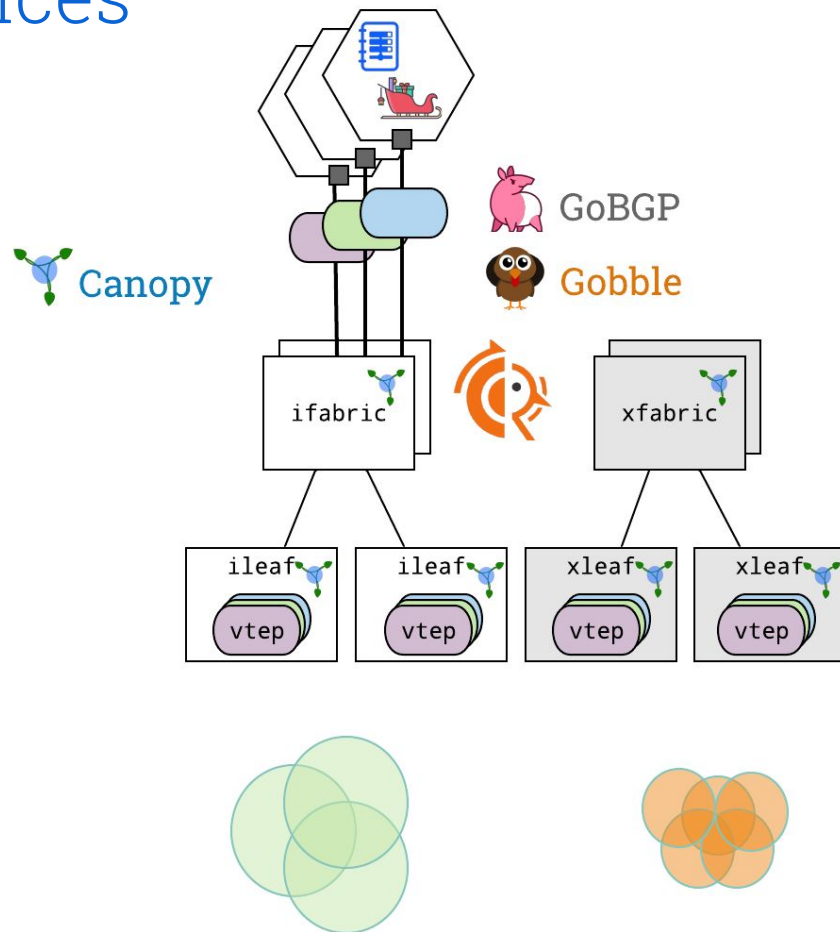
EVPN Based Experiment Networks

- Now adding services on multiple infrastructure networks in an isolated way becomes almost trivial.
- Launching a new service simply means launching a container and advertising it's MAC on the appropriate EVPN domain.
- This lets us run multiple services on an experiment's infranet (common EVPN domain) as well as support multiple independent infranets without worry about crosstalk.



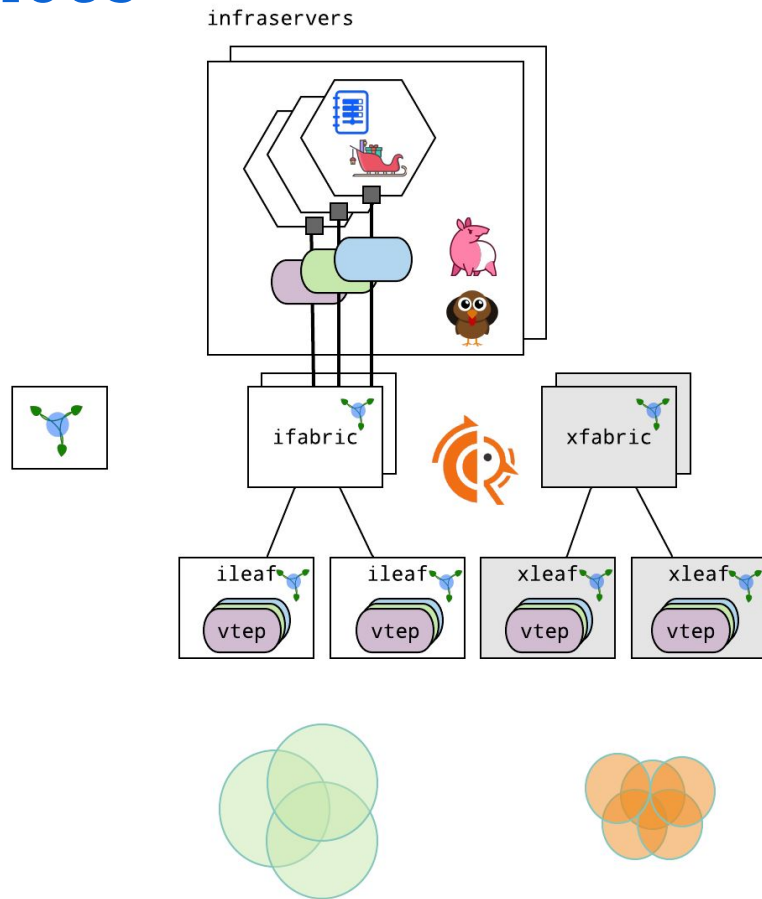
Infrapod Experiment Services

- Services do not end at DHCP/DNS, many others are needed.
- A container for each would work fine, but would create quite a bit of bloat.
- So we just burgle the Kubernetes Pod abstraction.
- Only need 1 experiment interface per N containers
 - (N-1) fewer
 - Interfaces
 - Network namespaces
 - Mac advertisements
 - IP addresses
- Now we have a common landing spot for all container based services needed for any given experiment
- Not only for basic infrastructure needs, but also for experimenters who would like a place to place containerized services on their network but not allocate a testbed resource for it (something like Prometheus immediately comes to mind)



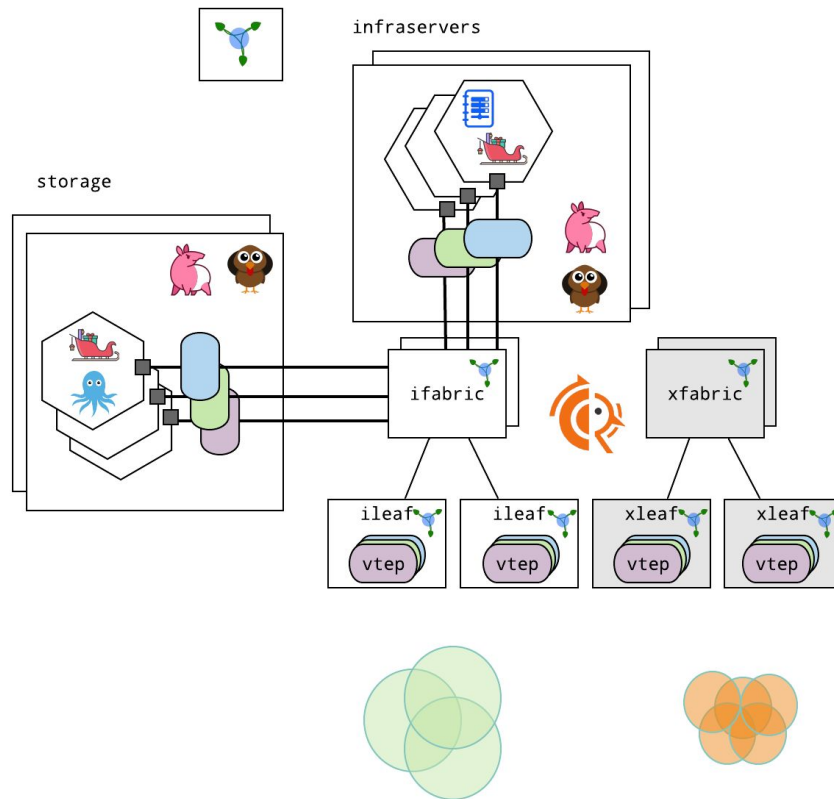
Infrapod Experiment Services

- Let's make things a bit more concrete by putting server boundaries around things and call the servers that host services in the way just described *infraservers*.
- One cool thing is that an infraserver, is that its very easily replicated thanks to the dynamically routed nature of EVPN-based services.



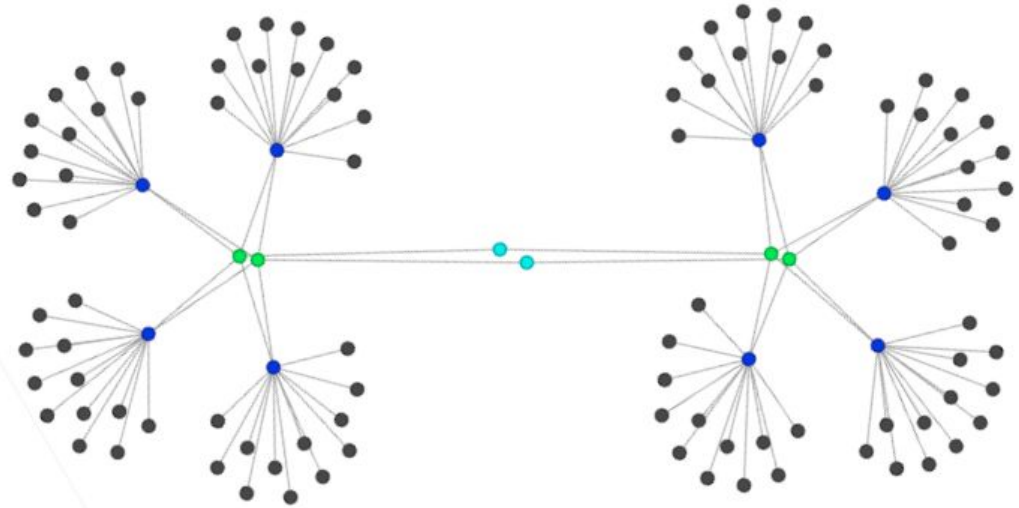
Infrapod Experiment Services

- It's also easy to have specific infraservers for specific things.
- As an example in our DARPA DComp testbed, we have purpose built storage servers that we run as another flavor of infraserver but the software stack is the same.
- ***The takeaway is that the basic architecture allows for a diverse family of specific building blocks to come together as a cohesive testbed under a robust underlay/overlay virtual multi-network.***



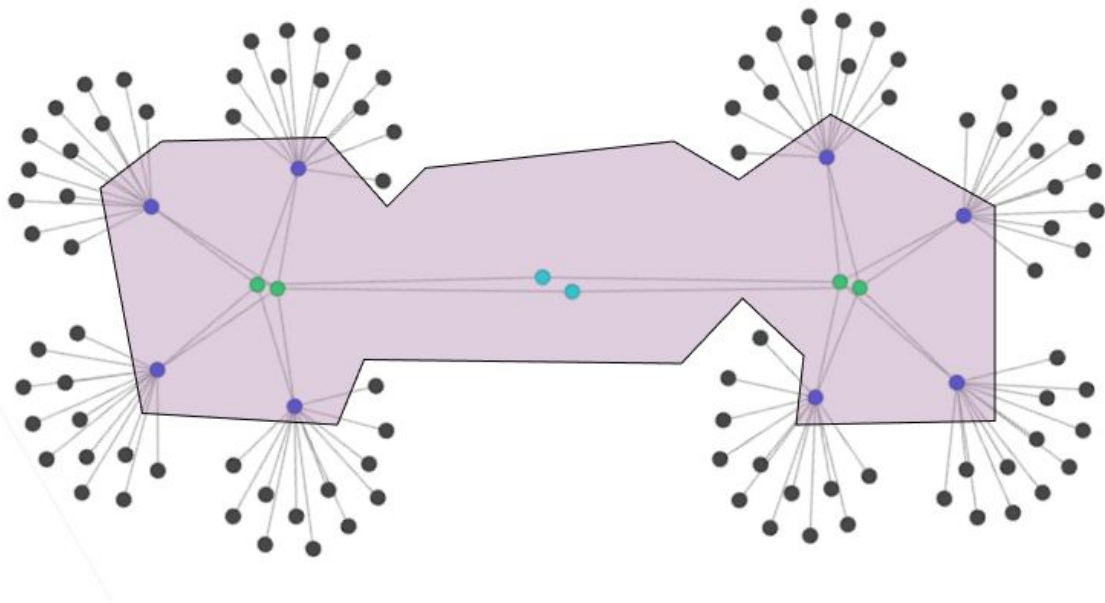
Network Emulation

- Suppose this diagram is our experiment topology
- Black nodes at the edges are servers.
- Blue nodes are leaf switches
- Green nodes are fabric switches
- Teal nodes are spine switches
- The core network is BGP/ECMP based.



Network Emulation

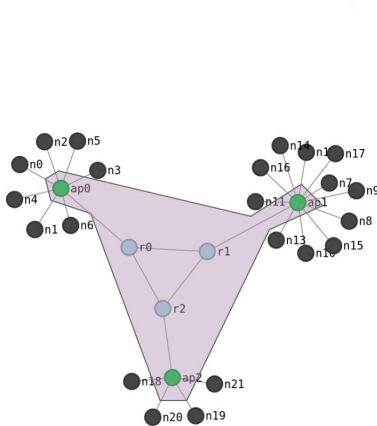
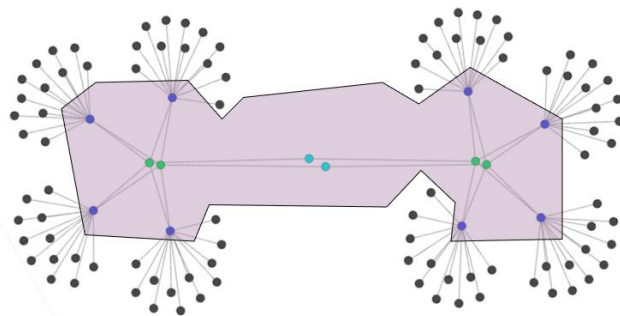
- We want high fidelity at the edges (e.g. testbed nodes)
- Don't want to have to implement an actual fattree load balanced network.
- DCompTB supports modeling and **emulating** the entire routed network and interconnecting nodes through it.



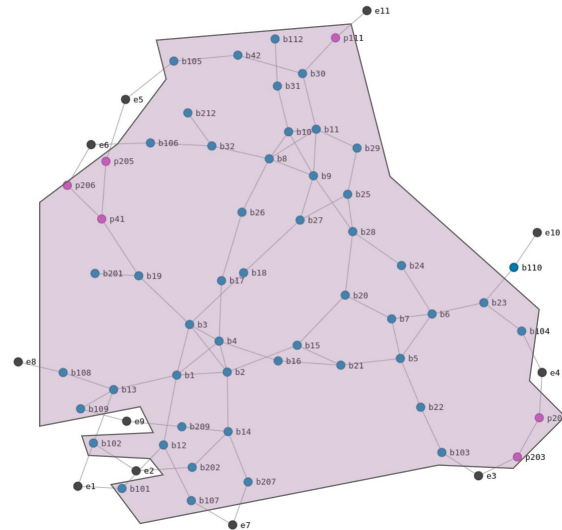
Network Emulation

- Support for multiple network types
 - Latency, capacity and loss parameterized p2p and multipoint links.
 - Wireless networks with mobility and migration models.
 - Several types of switched and routed networks.
- Basic traffic emulation included
 - Workload emulators (mimes)
 - Various source/sink models

Data Center Networks

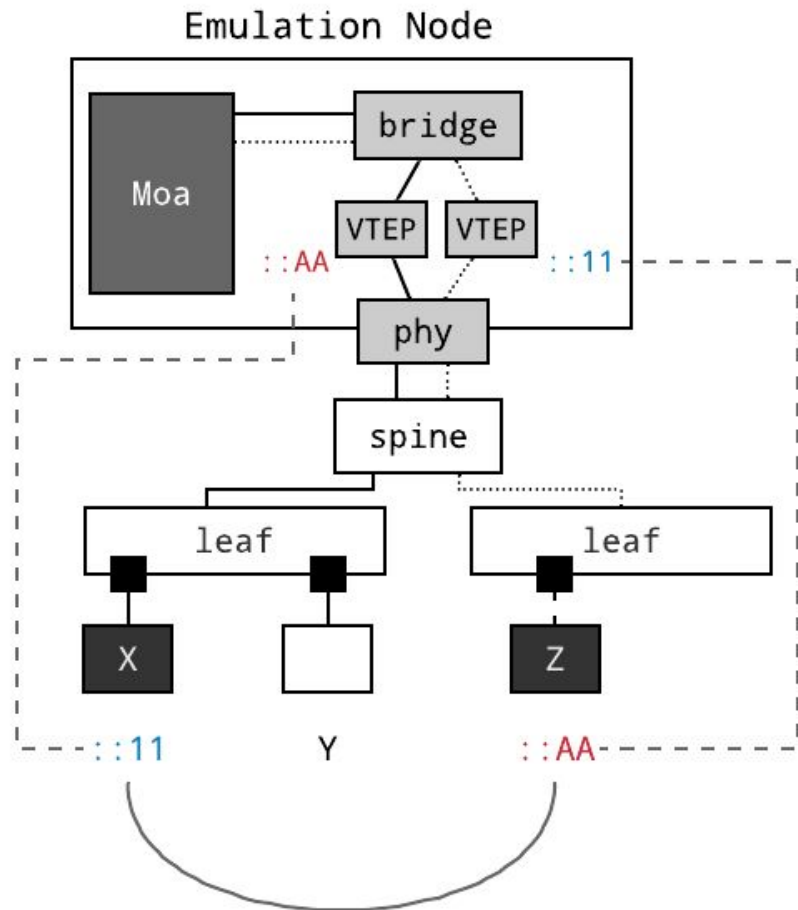


Wireless Networks



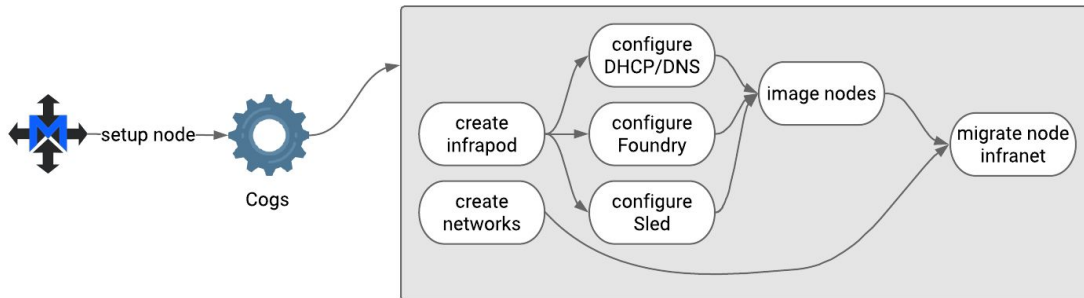
Network Emulation

- Emulation plumbing works through surrogate type-2 (MACADV) EVPN advertisements.
- When X needs to talk to Z through an emulator
 - X is advertised to Z at the emulation server.
 - Z is advertised to X at the emulation server.
 - The emulation server emulates the network in between and kicks out the packets on VXLAN segment in which the real MACADV lives.



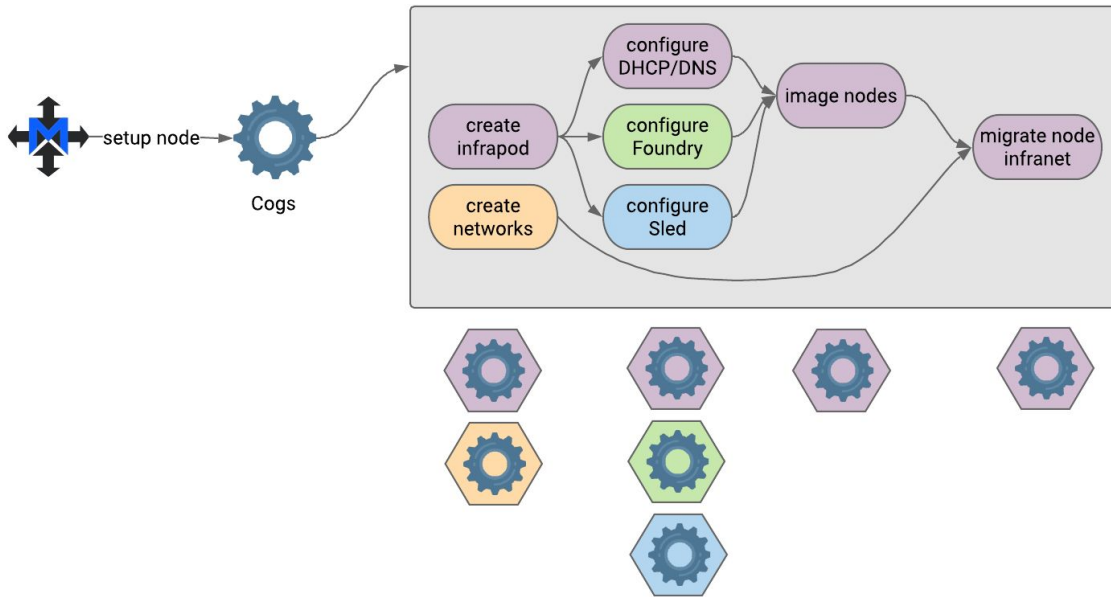
Experiment Materialization Runtime

- The cogs system transforms Merge materialization commands into a dependency graph of tasks.



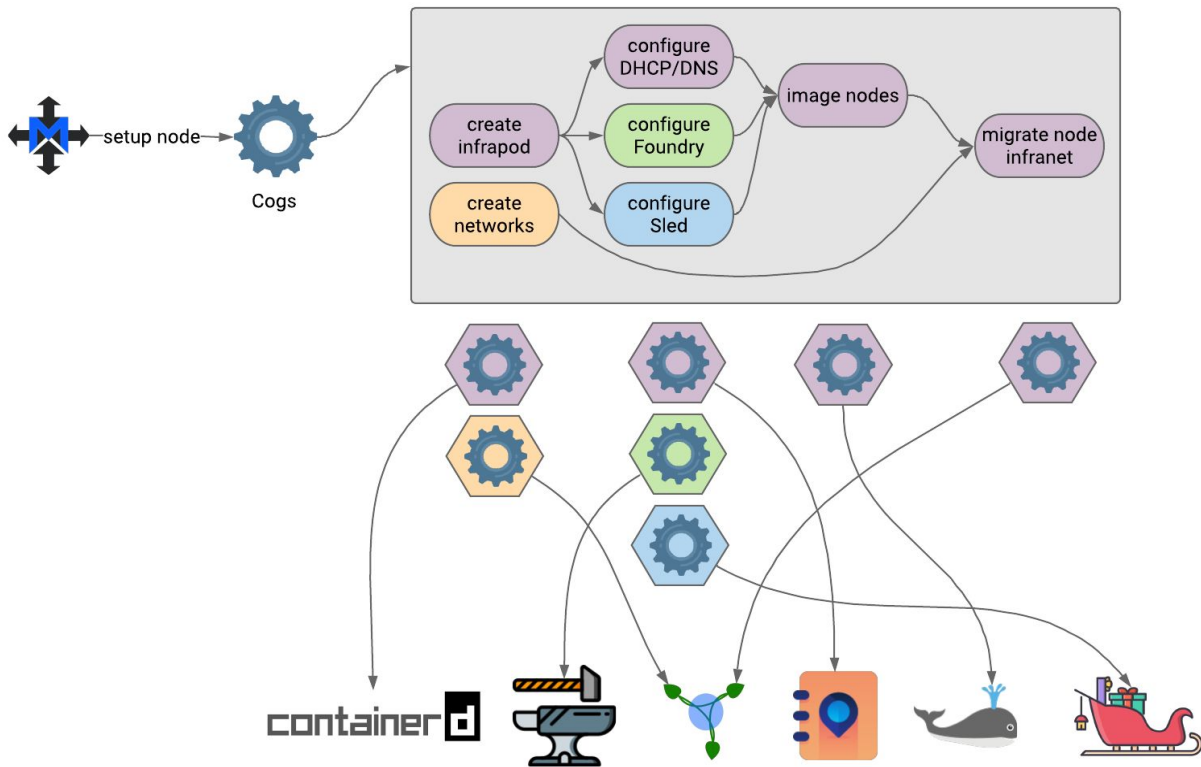
Experiment Materialization Runtime

- The cogs system transforms Merge materialization commands into a dependency graph of tasks.
- Tasks are executed by a pool of replicated cog workers.



Experiment Materialization Runtime

- The cogs system transforms Merge materialization commands into a dependency graph of tasks.
- Tasks are executed by a pool of replicated cog workers.
- The cogs complete tasks using Merge technology stack components.


















Modular Technology Stack

All of the following were built to support DCompTB but are

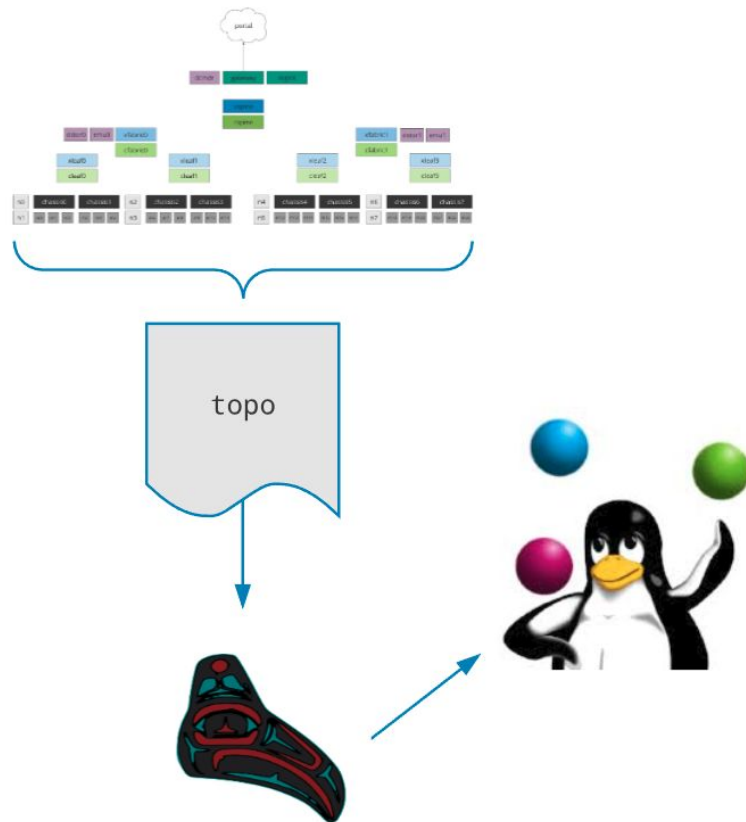
- Useful in any testbed setting.
- Provide strong APIs
- Are completely open source.

<https://gitlab.com/mergetb/tech>

	cogs	Testbed facility automation system.		wgd	Wireguard configuration daemon
	canopy	Virtual network synthesis		sled	Imaging system
	nex	Integrated DHCP/DNS server		tftp	A TFTP server for PXE
	rally	Ceph-based network storage		ipxe	A fork of iPXE node bootstrapping
	beluga	Power control daemon		rtnl	A Go-based rnetlink library
	moa	Network emulation engine		images	Testbed image creation automation
	gobble	GoBGP lower half for Linux		stor	Cogs storage library
	foundry	Testbed node configuration			

Virtual Testbed Development Environment

- Testbed in a bottle.
- Created Raven virtualization tool to make it easy to describe and deploy complex networks of virtual machines.
- <https://gitlab.com/rygoo/raven>
- Turn-key testing environment for Merge portal + testbed facility.
- <https://gitlab.com/mergetb/prototypes>

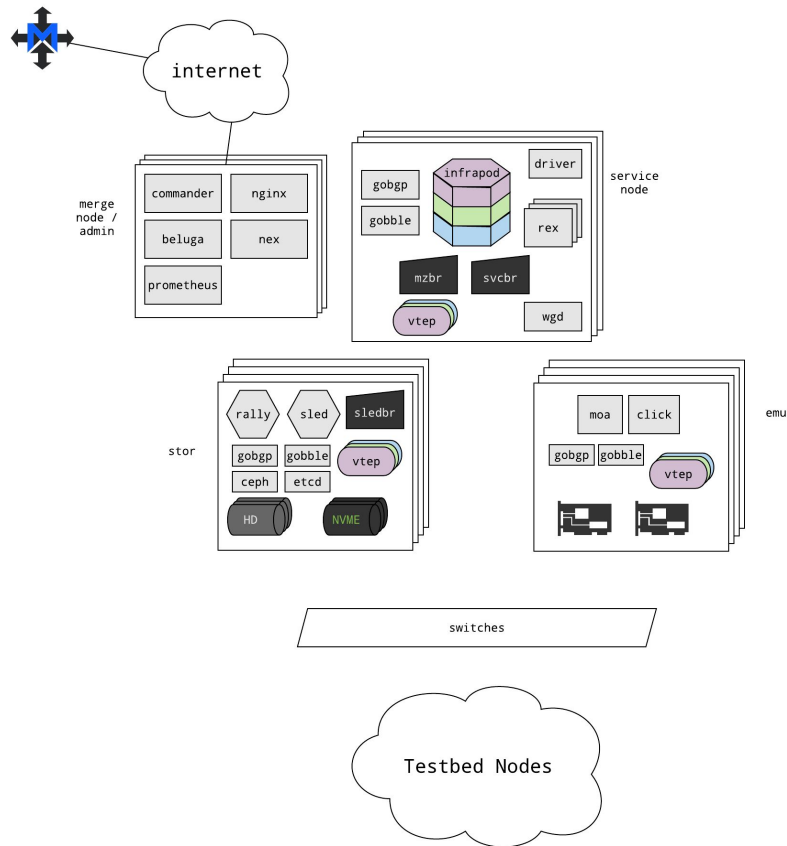


Thanks!

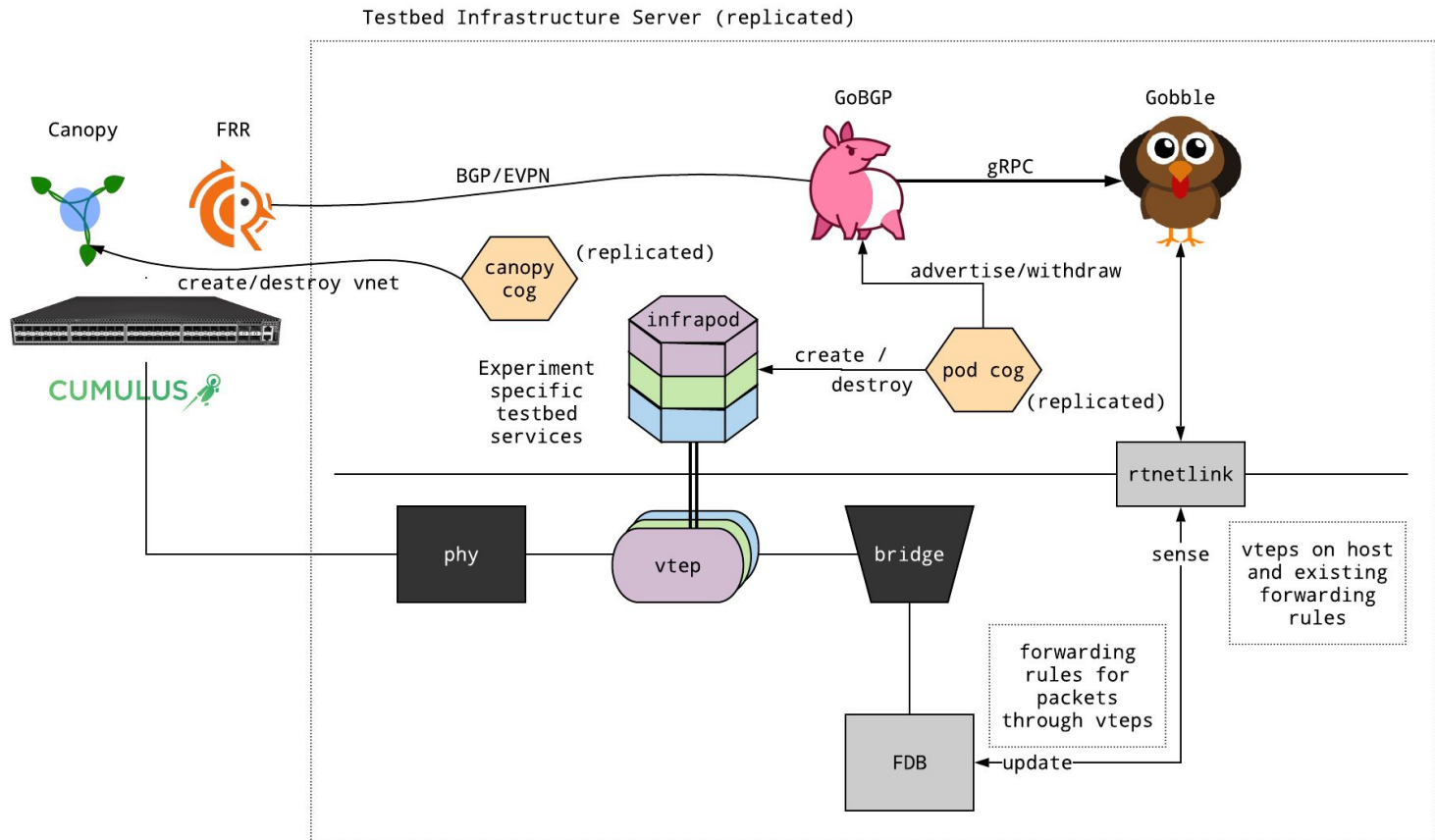
Backup

Popping up the stack

- Popping back up the stack, the general system looks like this
- In the top left we have a server that talks to Merge and acts as an entry point into the testbed. Then we have several flavors of infraserver.
- New in this diagram are network emulation nodes that live on *experiment* networks but are connected using the same EVPN stack.
- ***The takeaway is that the basic architecture allows for a diverse family of specific building blocks to come together as a cohesive testbed under a robust underlay/overlay virtual multi-network.***

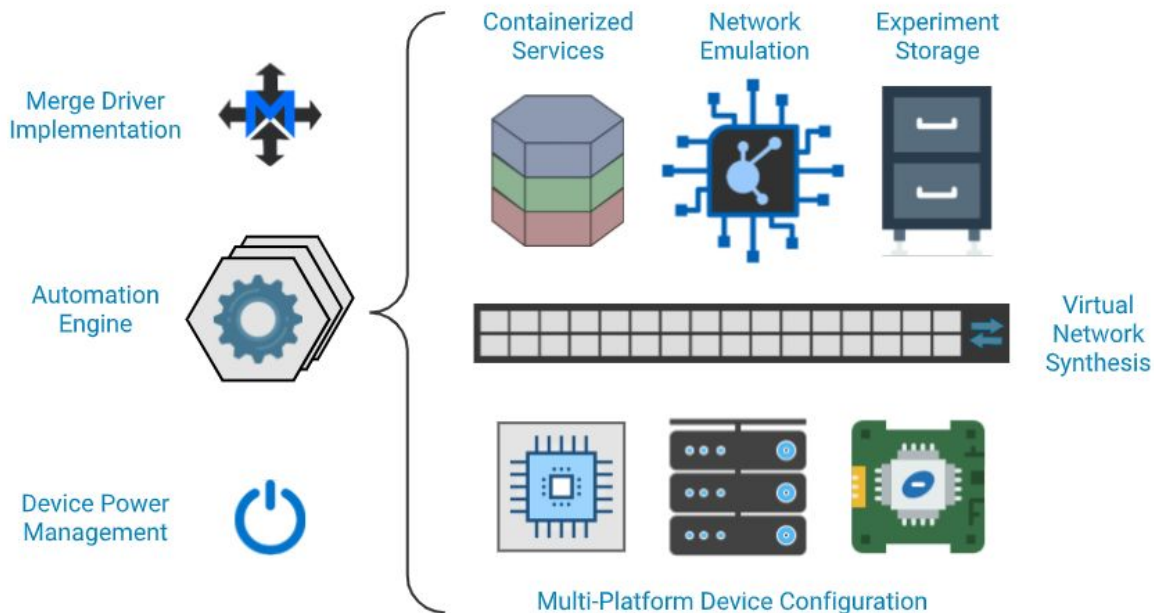


EVPN + Infrapods



Experiment Materialization Runtime

- The cogs system transforms Merge materialization commands into a dependency graph of tasks.
- Tasks are executed by a pool of replicated cog workers.
- The cogs complete tasks using Merge technology stack components.
- Every component in the Merge technology stack has a gRPC API making interfacing from the cogs simple.



EVPN Based Experiment Networks

- Services do not end at DHCP/DNS.
- Another interesting one is Sled - our **S**ystem **L**oader for **E**phemeral **D**evelopments (SLED). This is our OS imaging system.
- The Sled system does the following things
 - PXE-boots a u-root image with the sledc software on it
 - Sledc runs a protocol with the Sled server sledd that requests what image is supposed to be running on the node and where to get it
 - Sledc then grabs the image, stamps it on the device and kexecs into the images kernel (or bootloader)
- An interesting question is where to put sled? Just spin up another container and another address, maybe ...

