K8s in the Datacenter: Integrating with Pre-existing Bare Metal Environments

Bloomberg

Engineering

KubeCon + CloudNativeCon Europe 2020 *Virtual*
August 20, 2020

Max Stritzinger
Site Reliability Engineer

TechAtBloomberg.com

Overview

— K8s networking overview for on-prem

— Our network environment and journey

— Tips, tricks, and tools for debugging your environments

TechAtBloomberg.com

Bloomberg

Engineering

K8s Networking Model

— Pods are schedulable units of work
  - Each has either a IPv4 or IPv6 address or both (dual stack)
  - Must be able to communicate without NAT

— Two broad categories of networking implementations
  - Overlay network model
  - Flat network model

— We need to support both models

Overlay Networks

— Pod to Pod requires encapsulation
- Pod IPs on the wire will be dropped by the network
- Encapsulate traffic in another protocol
- Destination IP on outer packet is host running destination pod
- Source address on outer packet is host running source pod
- Requires agent to program routes

— Pod to external network requires source NAT
- Pods assume address of the host
- Usually accomplished using MASQUERADE on Linux
- Hosts act as gateways between pod network and everything else

Flat Networks

— Pods are “first class citizens” in your network
  - Routable by IP from outside cluster

— No encapsulation/SNAT required

— Need some way of sharing routes with the rest of the network

TechAtBloomberg.com

Bloomberg

Engineering

Choosing Flat vs. Overlay: Overlays

— Overlay Pros

- May not have to engage network team
- Can reuse pod IP space between clusters
- Ingress points to cluster are well-defined

— Overlay Cons

- Encapsulation overhead
- Forces pod traffic to all have host IPs
- Can complicate debugging

Choosing Flat vs. Overlay: Flat

— Flat Pros

- Everything “looks as expected” when debugging
- Pods can be talked to directly
- Lots of options for source-address based filtering

— Flat Cons

- Definitely requires engaging your network team
- Pods can be talked to directly
- Have to worry, sometimes deeply, about IPAM

Other Considerations

— IPAM! IPAM! IPAM! (on flat pod networks or for your VIPs)
- Service VIPs should probably be kept local to cluster

— NodePorts may not be enough for L3/L4 ingress
- Clients need to get node IPs somehow
- Ports must be from a high range
- May need LoadBalancer implementation

— By default controller-manager assigns nodes CIDRs
- Single cluster CIDR doesn't work if you need to grow it
- Probably not going to work in a flat network…

Calico

— Open source CNI provider that supports flat and overlay network implementations

— Great IPAM support <3
- Adding new pools / growing the CIDR range
- Selecting pool to allocate based on node or pod labels
- Nodes are dynamically assigned blocks (multiple CIDR ranges per node)

— Also handles NetworkPolicies

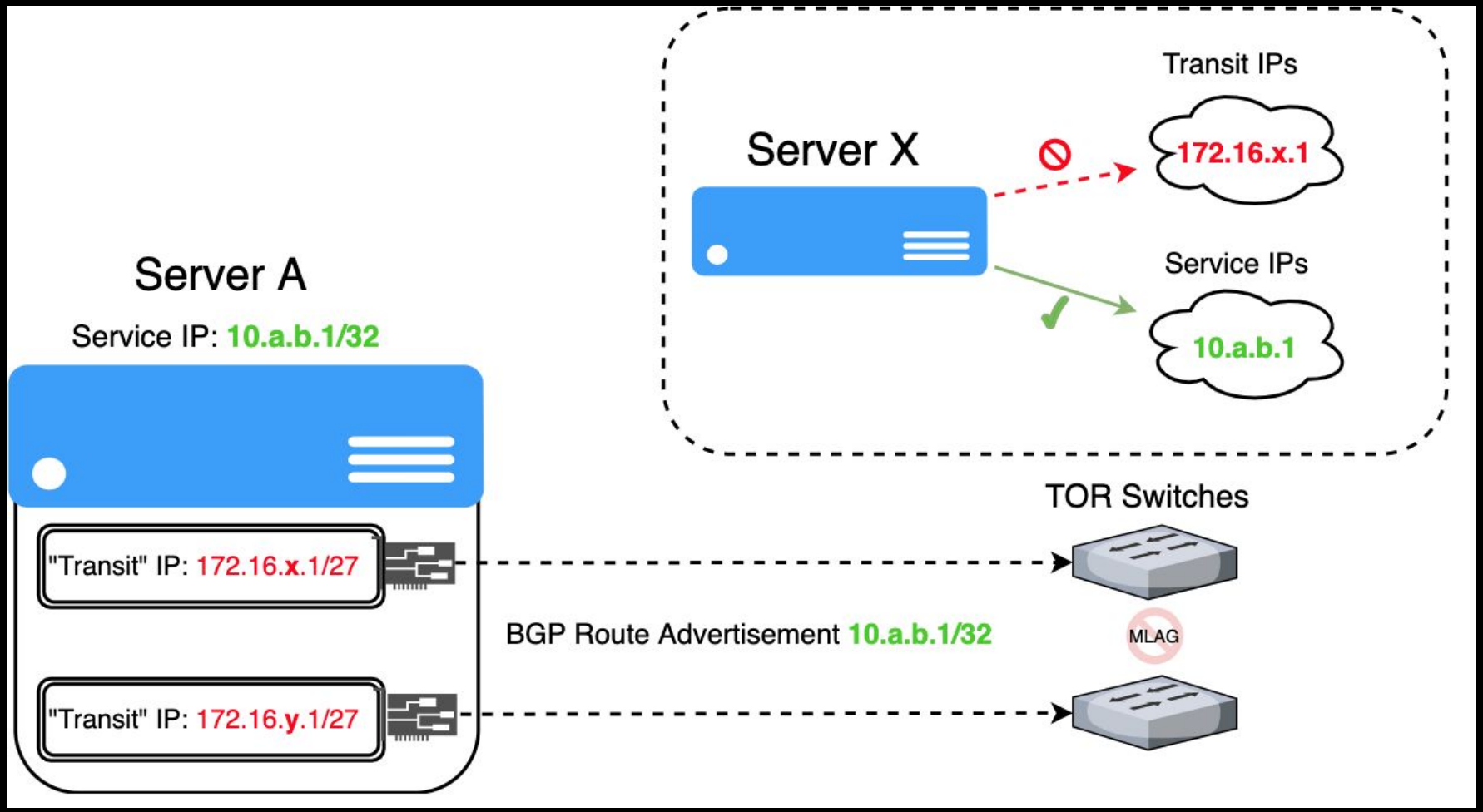
Implementing K8s in Our Environments

— We run on bare metal and private cloud VMs

— New network architecture
- Old tooling breaks for network reasons :(
- Let’s debug!

— What does our environment look like?
- L3 ECMP to each host
- Host advertising single IP to 2 independent ToRs
- Using BGP with BIRD to do so
- RFC 7938

Implementing K8s in Our Environments



TechAtBloomberg.com

Bloomberg

Engineering

Implementing K8s in Our Environments: Challenges

— Are we pioneers?

— Trying to understand what the purposes of different implementation details are

— Need to modify open source projects

— Sometimes people don’t understand your use case :(

TechAtBloomberg.com

Bloomberg

Engineering

Issue #1: Running Multiple BIRD Instances

— Our hosts use BIRD to advertise their service addresses

— Calico uses BIRD to advertise addresses

— Both run in root network namespace, binding same ports and addresses

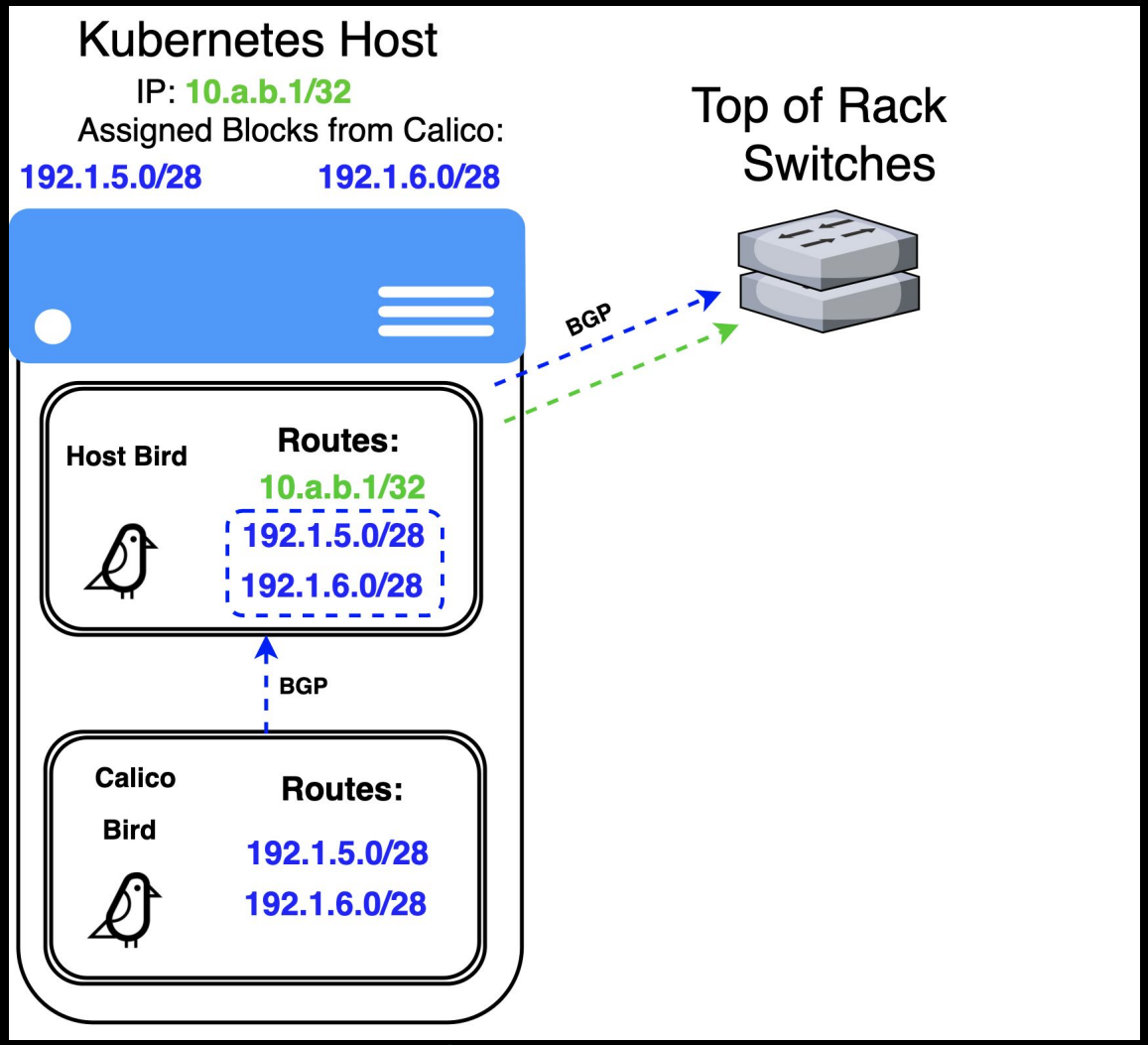
— ToR will only accept one peering connection

Issue #1: Running Multiple BIRD Instances

— Same ports and addresses
- “Host” BIRD does not need incoming connections
- Overwrite Calico BIRD template files with volumes

— ToR will only accept one peering connection
- Multiple BGP speakers is a common problem
- Host BIRD is the most critical, because if it dies we cannot reach hosts
- Don’t want to need CNI to be up in order for host networking to work
- Peer Calico BIRD to host’s BIRD

Issue #1: Running Multiple BIRD Instances



TechAtBloomberg.com

Bloomberg

Engineering

Issue #1: Running Multiple BIRD instances

— Hosts now drop on and off the network

— Routing table alternating between K8s and network routes

— BIRD acts similarly to K8s operator

— Sync to different routing tables!

Issue #1: Running Multiple BIRD instances

```
vagrant@node-0:~$ ip rule
0:      from all lookup local
1000:   from all lookup main
1001:   from 172.20.236.225 lookup enp0s8_table
1002:   from all lookup default
1003:   from all lookup enp0s8_table
32766:  from all lookup main
32767:  from all lookup default
vagrant@node-0:~$ ip route
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15
10.0.2.2 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 100
10.244.39.193 dev cali60a8fdefce2 scope link
10.244.39.194 dev calif0c528fdc39 scope link
10.244.84.128/26 via 10.244.84.128 dev vxlan.calico onlink
172.20.236.224/31 dev enp0s8 proto kernel scope link src 172.20.236.225
198.18.0.0/15 dev docker0 proto kernel scope link src 198.18.0.1 linkdown
vagrant@node-0:~$ ip route show table main
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15
10.0.2.2 dev enp0s3 proto dhcp scope link src 10.0.2.15 metric 100
10.244.39.193 dev cali60a8fdefce2 scope link
10.244.39.194 dev calif0c528fdc39 scope link
10.244.84.128/26 via 10.244.84.128 dev vxlan.calico onlink
172.20.236.224/31 dev enp0s8 proto kernel scope link src 172.20.236.225
198.18.0.0/15 dev docker0 proto kernel scope link src 198.18.0.1 linkdown
```

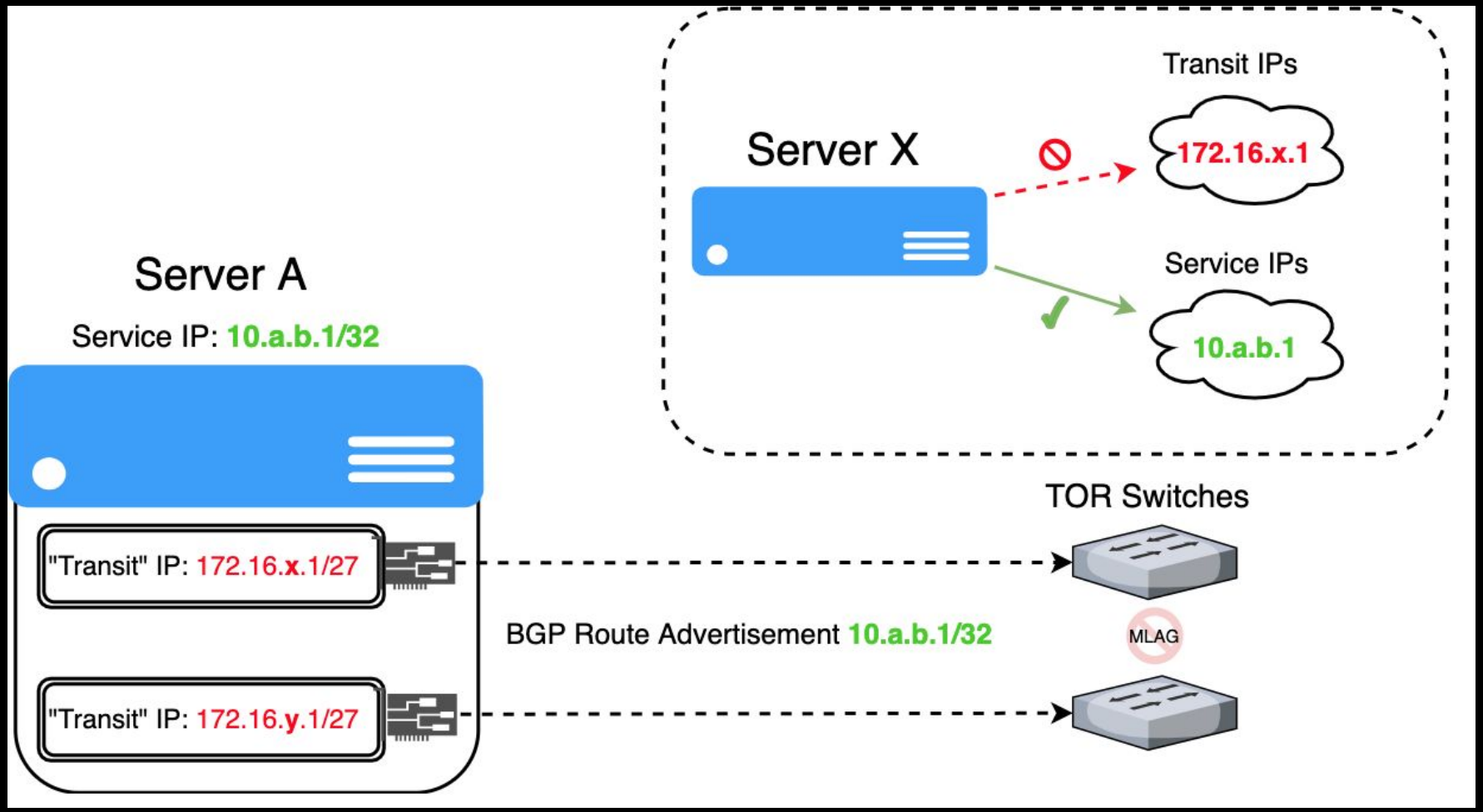
Issue #2: Iptables Masquerade

— Iptables target that marks connection for source NAT
  - Used by Calico, kube-proxy, portmap CNI, and many others

— Determines address for source NAT automagically
  - Looks at “primary address” on outgoing interface
  - Takes no other routing information into consideration
  - Really useful in any situation other than ours!

— Bad for us!

Issue #2: Iptables Masquerade



TechAtBloomberg.com

Bloomberg

Engineering

Issue #2: Iptables Masquerade

— Where is MASQUERADE used?
- Used for source NATing of pod → external
- Used for hairpin traffic in kube-proxy, host ports
- Used for host → local pod traffic
- Used for NodePorts
- ….

— In **most** of these instances we need SNAT rule instead
- Allows for specifying a specific, single IP

— Still sometimes need MASQUERADE though. Ugh!

Addressing MASQUERADE: Calico

— MASQUERADE rule used for pod → external traffic

— Traffic has transit address attached and gets dropped

— We modified Calico to be able to use SNAT rule instead of MASQ

— Changes are upstream

TechAtBloomberg.com

Bloomberg

Engineering

Addressing MASQUERADE: Kube-Proxy

— When is MASQUERADE used?

- Hairpin traffic
- Arriving off-cluster traffic
- NodePorts
- LoadBalancers
- ...

— When do we *need* it?

- Only with tunnel devices
- MASQUERADE takes the IP on the tunnel device
- Only when destination pod for a service is off-node

Addressing MASQUERADE: Kernel Bug?

— Overlay environment, IP in IP

— Some hosts talking to K8s service VIP are MASQ'ing on the encapsulating packet
  - But only to some VIPs

— Some hosts are fine

— Issue seems to show up and then disappear intermittently

— Let's debug!

Network Debugging: Starting Points

— ping

— dig

— netcat
  - Basic tests of TCP, UDP connectivity

— iproute2 suite
  - ss -- socket stats
  - ip route/rule/addr/link/neigh

Network Debugging: Going Deeper

— tcpdump
- Old faithful
- See packets as they are “on the wire”
- See **IF** they’re on the wire

— iptables
- Read the iptables (or nftables) rules. It’s doable!
- Use iptables TRACE target for help debugging

— conntrack
- Keeps track of connections and any NAT done

Network (Kernel) Debugging: Even Deeper

— perf trace
- Use it like strace, but also see tracepoints in the kernel
- net:*, skb:*, tcp:*, udp:* tracepoints for network debugging

— eBPF: It’s not just for CNI implementations
- Inspect arguments to kernel functions on live systems

— ftrace
- Get call graph for every function in the kernel used on behalf of a process

Our issue?

— kube-proxy uses iptables MARKs to select packets for MASQing

— Using eBPF, we were able to see that those MARKs persist through packet encapsulation (e.g., with a tunnel device)

— This makes the conntrack entry for the encapsulating packet marked for MASQ

— Encapsulating protocol is IP in IP, so all subsequent encapsulating packets to the destination host are MASQ'd

— BUT, if the first encapsulating packet is not MARK'd, then future ones will also not be

Thank you!

https://www.bloomberg.com/careers

Bloomberg

Engineering

TechAtBloomberg.com