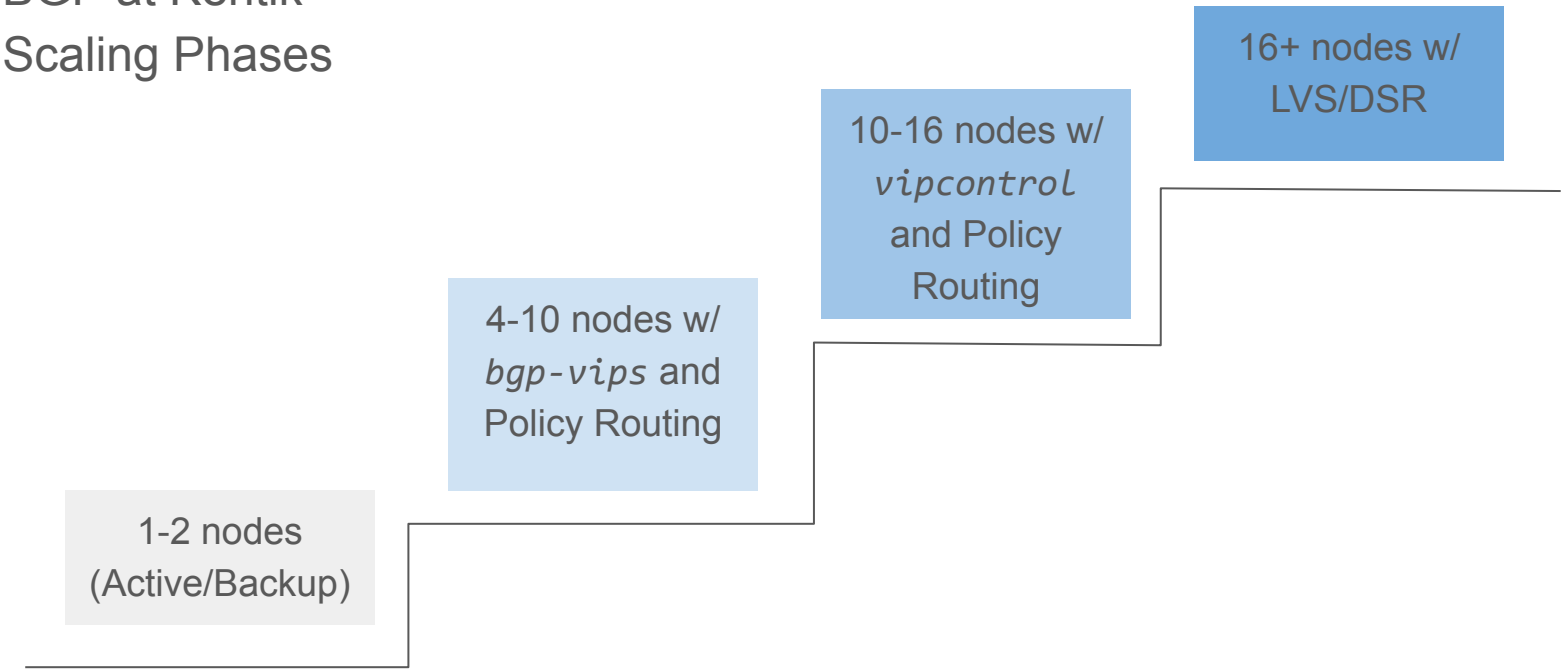
Scaling to support thousands of BGP peerings in a SaaS environment

Costas Drogos <costasd@kentik.com>
Kostis Fardelas <kosfar@kentik.com>

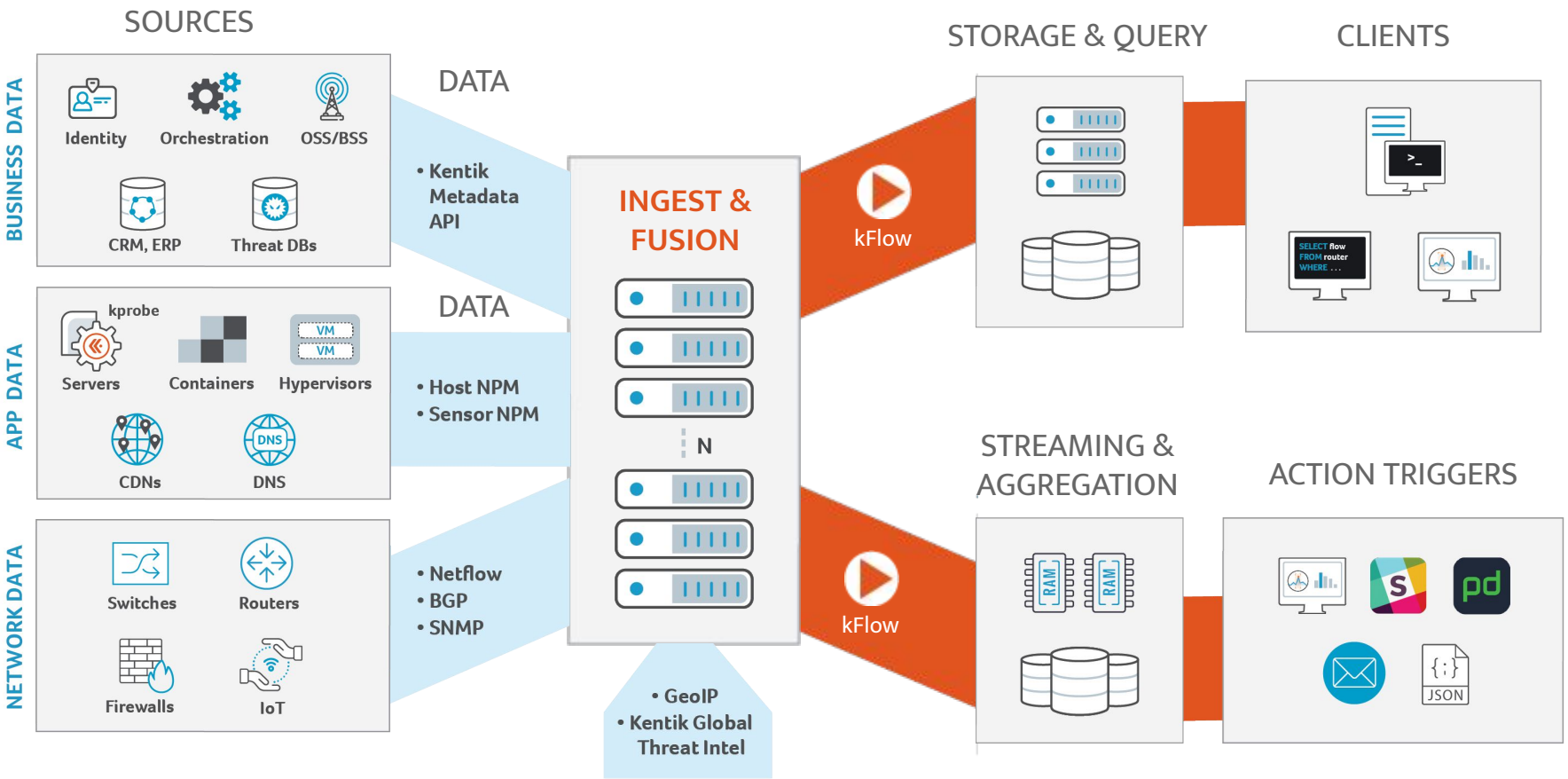
kentik

Contents

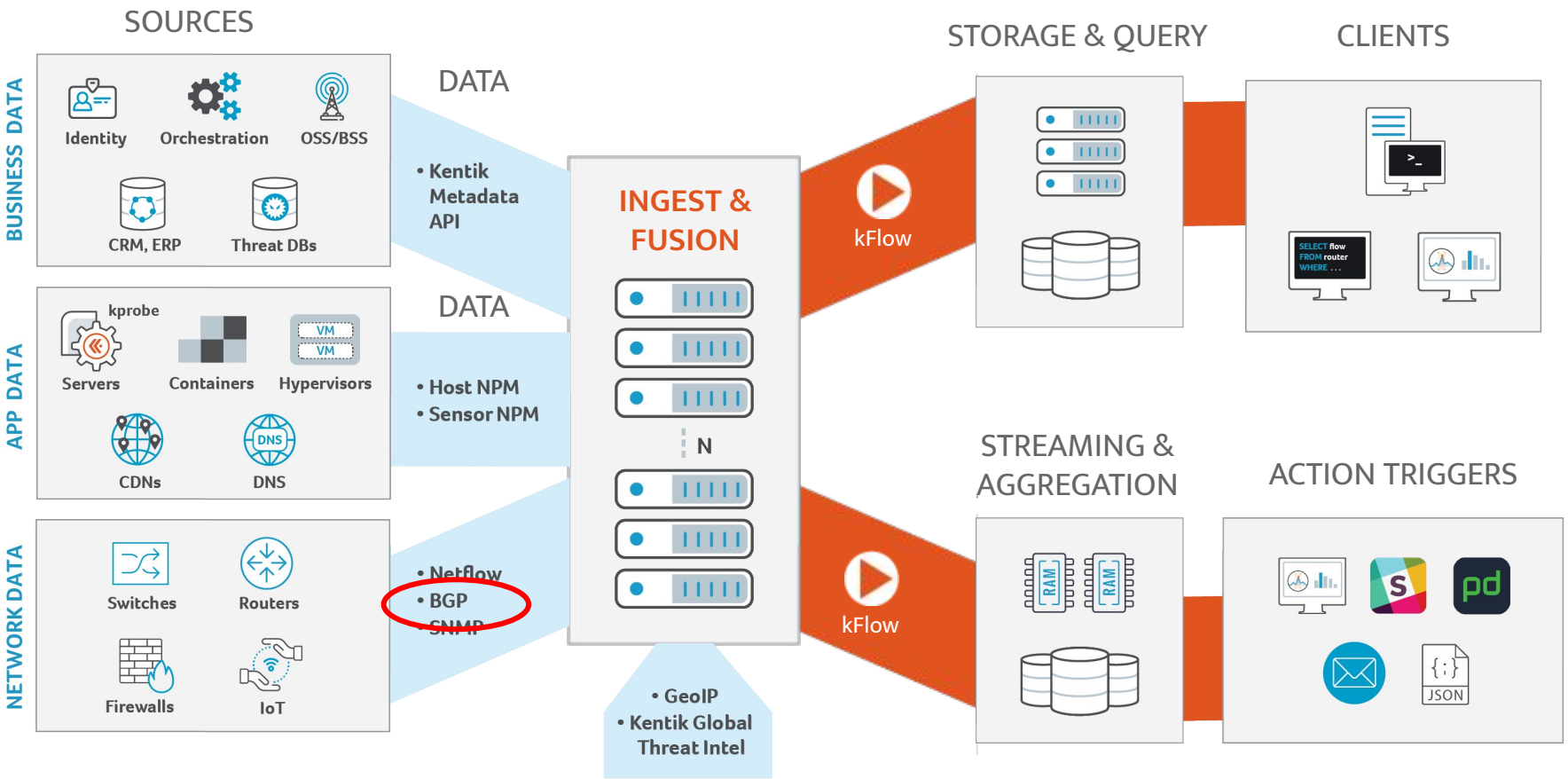
- BGP at Kentik
- Scaling Phases



Kentik Platform



Kentik Platform



BGP at Kentik: functionality

- Kentik performs peerings with Customers
  - Preferably with every device sending flow and running BGP for routing
  - Passive iBGP route-reflector speaker on commodity Debian GNU/Linux servers
  - BGP functionality is part of our contracted SLA - 99.99%

Just for filtering by BGP attributes in queries?

BGP at Kentik: functionality

- Kentik performs peerings with Customers
  - Preferably with every device sending flow and running BGP for routing
  - Passive iBGP route-reflector speaker on commodity Debian GNU/Linux servers
  - BGP functionality is part of our contracted SLA - 99.99%

Just for filtering by BGP attributes in queries?

- Routing analytics
- Peering Analytics
- RPKI
- Ultimate Exit calculation
- VRF integration

- Prefix alerts
- RTBH
- Flowspec

The beginning

- Less than 200 peers, IPv4 only

- 2 nodes in Active/Backup mode
  - Kentik's BGP software runs in both nodes
  - A floating VIP handles the HA/failover part managed by uCARP

- A script in `/root` sets everything up on boot via `rc.local`

Kentik grows

- RTBH feature is released to customers
  - BGP peerings now play a more active role
- Kentik's internal fabric gets upgraded to 10G
  - But also supports OSPF & BGP now
- Peerings don't fit in 1 node anymore
  - More than 300 peers in a server
  - Memory and CPU intensive
- uCARP doesn't scale well for 2+ nodes
  - Lots of corner cases, health checks

Kentik grows

- uCARP gets replaced by *bgp-vips*
  - Shell script talking to a spawned exaBGP
  - Floating BGP VIP is now mounted on the loopback interface
  - Announced to Kentik's BGP fabric with a different MED per node

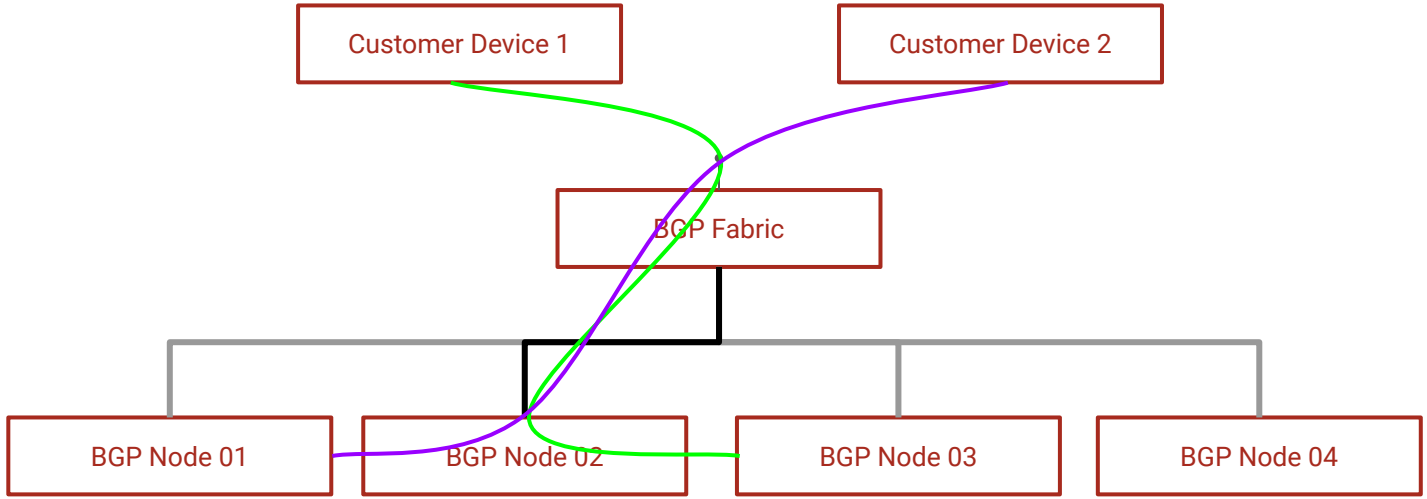
```
R1=$(( ( RANDOM % 10000 )  + 1 ))
STATE="down"
while true; do
  ....
  if [[ "$STATE" == "up" ]]; then
      echo "announce route 208.76.14.223/32 next-hop self med" $R1
  fi
  if [[ "$STATE" == "down" ]]; then
      echo "withdraw route 208.76.14.223/32 next-hop self med" $R1
  fi
  sleep 2
done
```

Kentik grows

- Connections still land to a single node, but don't fit there
- We have to offload connections to other nodes
  - So the *landing* node has to act as a router of sorts
  - Enter policy routing, aka MARK and ip rule / ip route in the Linux world
  - How to mark them to achieve a balanced distribution?
    - Oldest trick in the book: wildcard masks

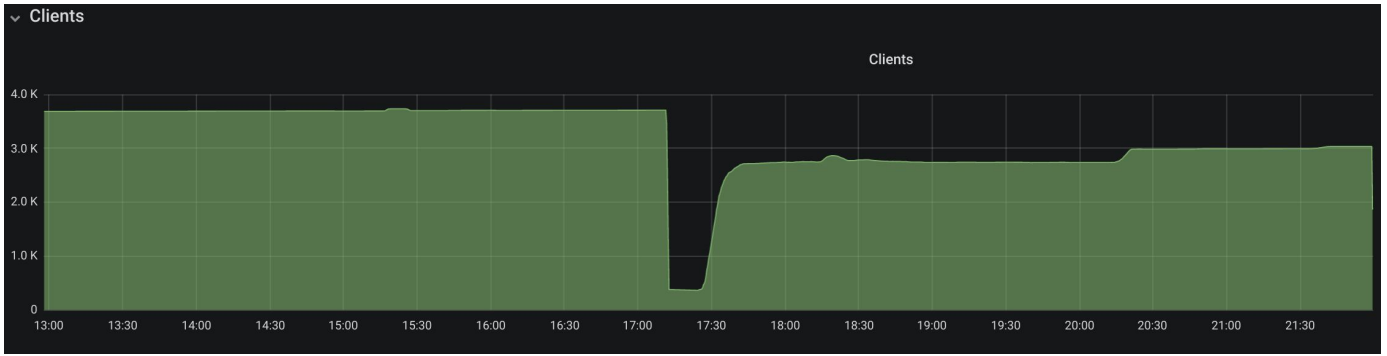
```
iptables -A PREROUTING -t mangle -p tcp -s 0.0.0.1/0.1.1.1 --sport 179 -d 208.76.14.223/32 -j
MARK --set-mark 100
iptables -A PREROUTING -t mangle -p tcp -s 0.0.1.0/0.1.1.1 --sport 179 -d 208.76.14.223/32 -j
MARK --set-mark 101
...
ip route add 208.76.14.223/32 via 1.1.1.2 table bgp-even
ip route add 208.76.14.223/32 via 1.1.1.3 table bgp-odd
ip rule add pri 29000 fwmark 100 table bgp-even
ip rule add pri 29000 fwmark 101 table bgp-odd
```

BGP + MARK + policy routing



Issues

- Can’t find an IPv6 mask good enough for a uniform distribution
  - /32?
  - /48?
  - /64?
- Topology modification means full exaBGP restart



Kentik grows more

- We've passed 1300 peers
  - IPv6 doesn't fit in one node anymore (yay!)
- Mask-based balancing not very optimal anymore
  - Lots of bigger device customers have devices in the same /24 or /26
- Various shortcomings with *bgp-vips*
  - Can't modify MEDs without restarting the exabgp process
  - Need to support more complex health checks

But, not much time to design something from scratch, we grow fast!

Kentik grows more

So, we decide to improve the current setup

- We keep policy routing, routing tables, routing rules
- But we replace mask-based routing with hash-based routing
  - from MARK to HMARK
  - https://lwn.net/Articles/488663/

```
iptables -A PREROUTING -t mangle -d ${BGPVIP1} -j HMARK --hmark-offset 100 --hmark-tuple src
--hmark-mod 10 --hmark-rnd 0xdeadbeef


ip6tables -A PREROUTING -t mangle -d ${BGP6VIP1}/128 -j HMARK --hmark-offset 200
--hmark-tuple src --hmark-mod 10 --hmark-rnd 0xdeadbeef
```

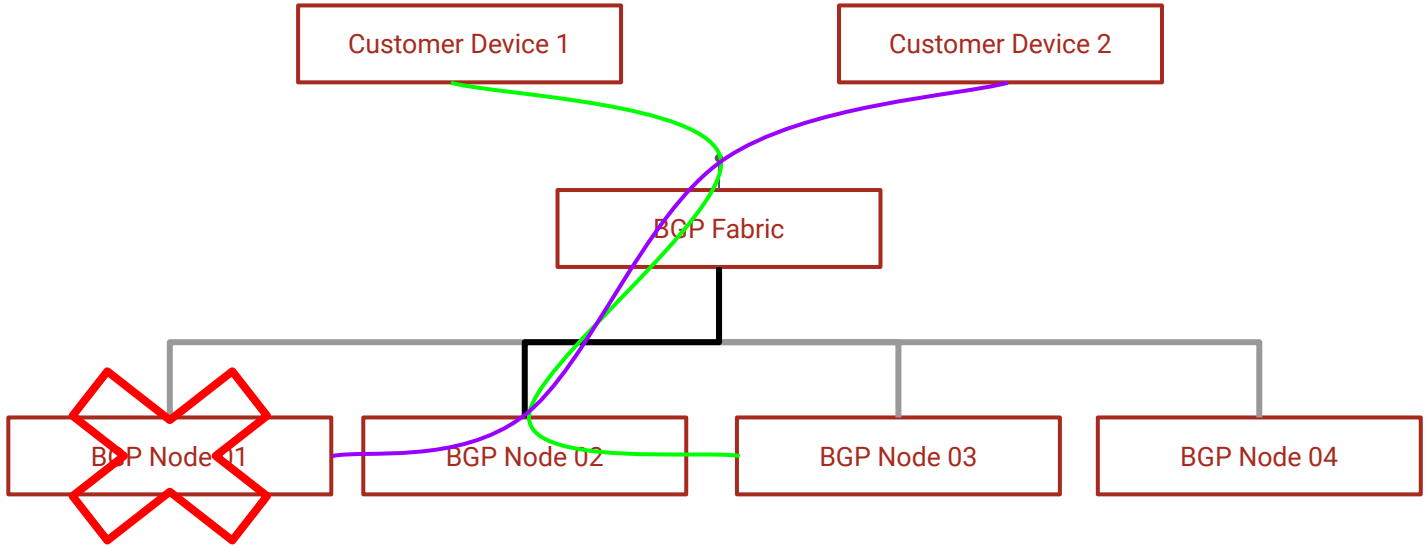
Kentik grows more

...and we decide to replace *bgp-vips* with *vipcontrol*

- A real daemon in Python communicating with a side-running exaBGP
- Assigning random MEDs to configured VIPs on startup
- Offering the ability to modify exaBGP's state
- Accompanied with a new *vipcontrol-health* sidecar daemon that performs HC and is able to dynamically modify MEDs

```
$ sudo vipcontrolctl list
...
# DEBUG KEY: 208.76.14.223/32 VALUES: {'action': 'announce', 'med': 7854, 'details':
'Added by viphealth: 2019-07-19 18:35:26.414150'}
announce route 208.76.14.223/32 next-hop self med 7854
...
```

What if a node needs to be removed?

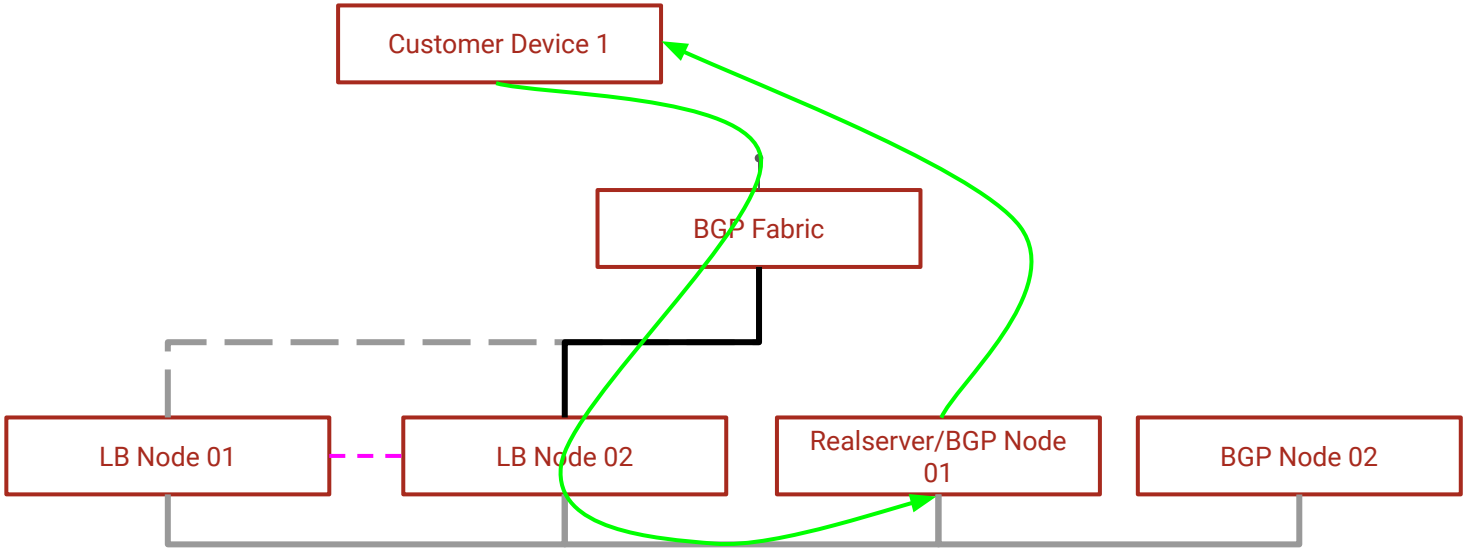


Kentik keeps growing

A new design to handle 4000+ sessions and replace policy routing and HMARK

- IPv4 & IPv6 support
- Uniform session distribution
- Horizontal scaling
- Health checking
- Provisionable, iterable & testable like the rest of our infra
- Low-disruption maintenance, pooling, depooling, code deploys
- Easier instrumentation and debugging

LVS/DSR setup



Under the hood

- VIP announcement via Bird with BFD enabled for faster route failover
- Keepalived in LVS/DSR mode with healthchecking and weighting
- Connection sync with LVS sync daemon for seamless L4LB failover

LVS/DSR setup

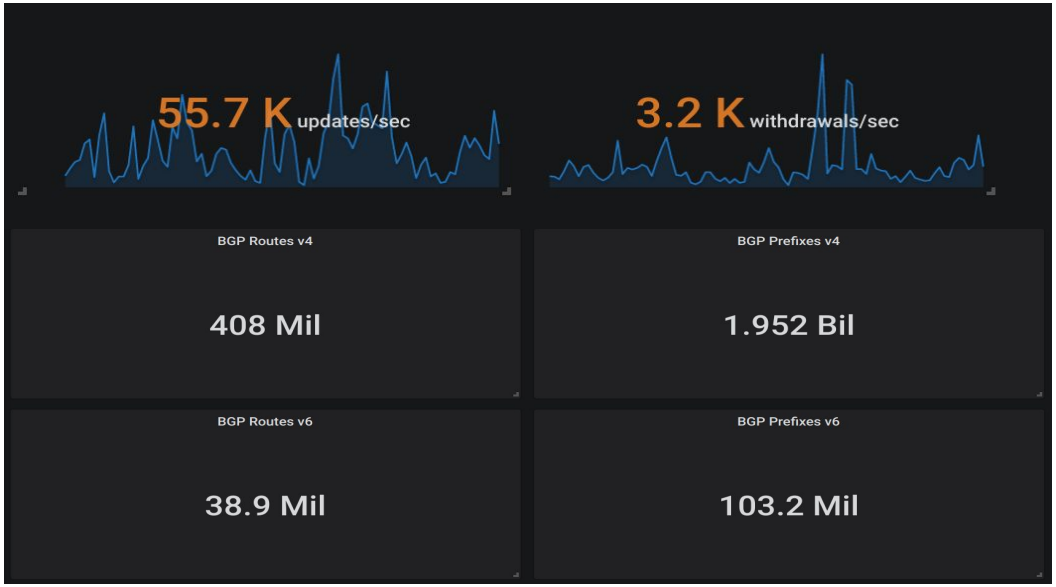
- Pros
  - Ability to depool/repool servers with low customer impact
  - Offload server-to-client traffic to realservers
  - Online configuration changes are possible
  - Everything is persisted in Configuration Management
- Cons
  - Can’t scale out of single LAN
  - All BGP connections pass through a pair of LB nodes
  - No more consistent hashing (yet!)

Testing & Tuning

- Need to emulate thousands of BGP connections
  - Spotify's https://github.com/spotify/super-smash-brogp

- Lots of tuning knobs
  - IPVS
    - net.ipv4.vs.* sysctl, connection expire timeouts etc
  - Keepalived
    - notify_up/notify_down, quorum_up/quorum_down, interface tracking etc
  - Bird + BFD
    - timers

What lies ahead



- Scale the L4LB layer
- Minimize backend servers maintenance disruption
- Better resilience to failure

Q&A

Thanks!