An Architecture of Highly Available Services using Anycast and Segment Routing in IPv6

Andrew Wang
Principal Software Engineer 2
Comcast

in/AndrewKeWang

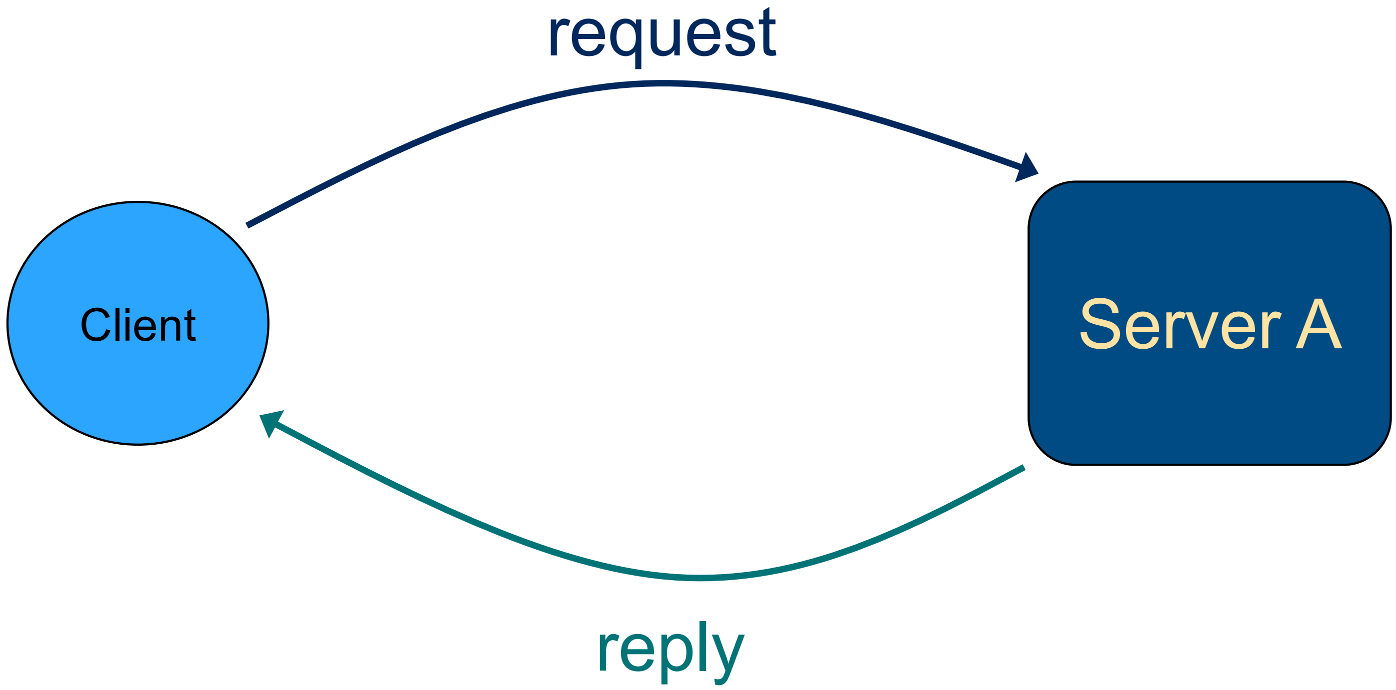
Who Am I

- Principal Software Engineer 2 at Comcast
- Past projects
  - GDB
  - Content based routing in ad hoc networks
  - Key value stores
  - Load balancing services
- Our team's name is Occam, right now working on the "Occam Gateway (OG)" project
  - Peter Cline
  - Daniel Jin
  - Zeeshan Lakhani
  - Chris Rollins
  - Andrew Wang

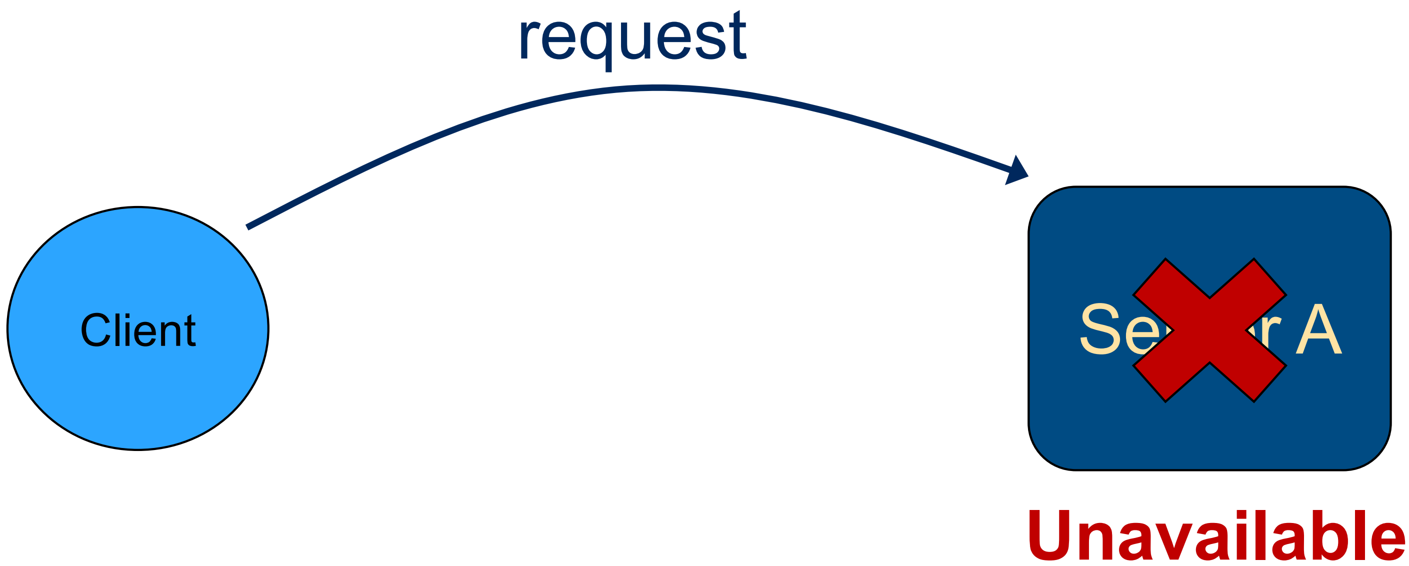
Agenda

- Motivation
  - Highly available services and how to make them happen
- Emergence of Anycast as part of the solution
  - What is Anycast and how it is achieved
- Segment Routing in IPv6 and what it can do for us
  - Brief introduction and what it can do for us
- Putting all the pieces together
- Demo in Containernet

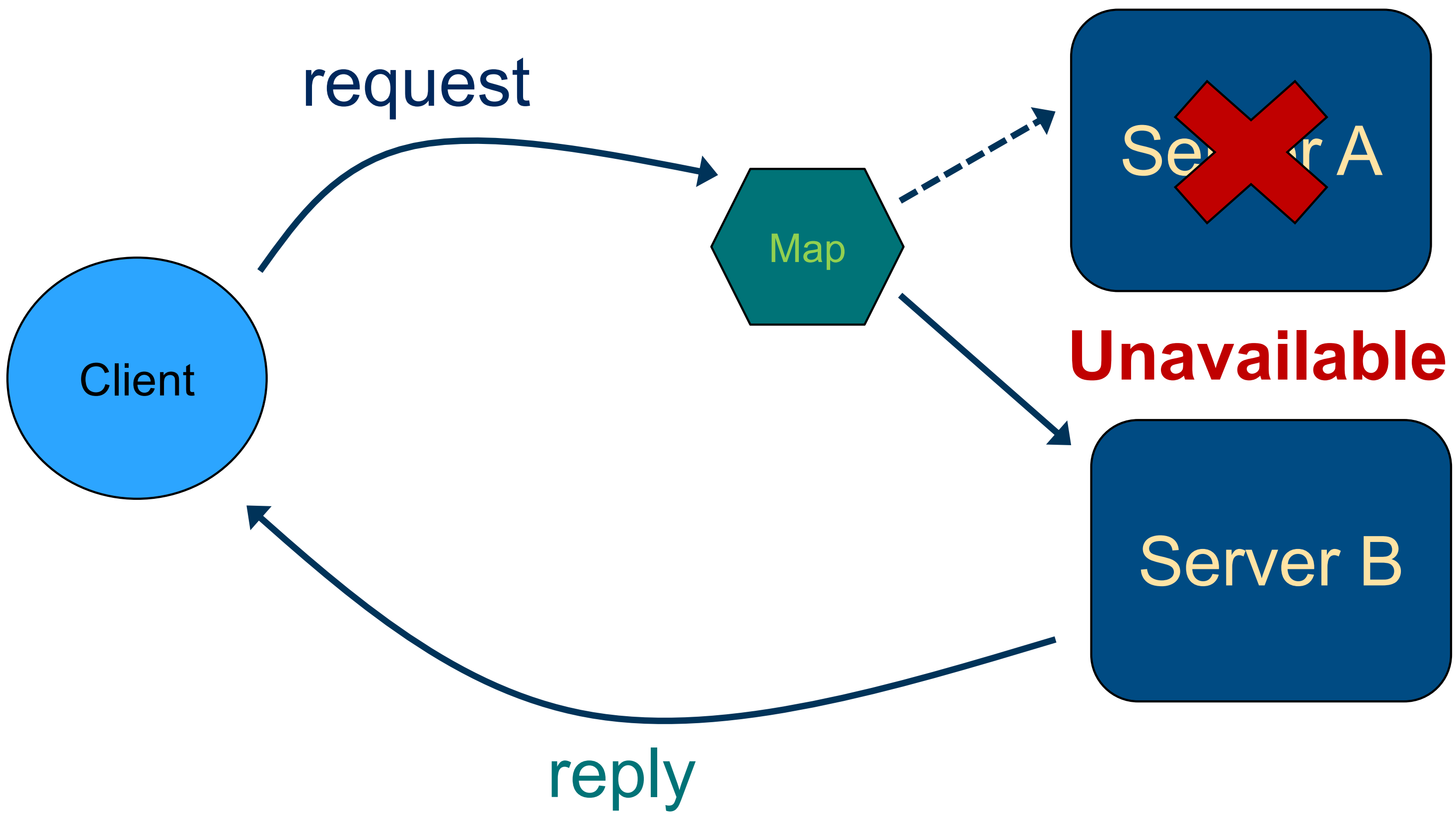
So we want to provide a highly available service



So we want to provide a highly available service

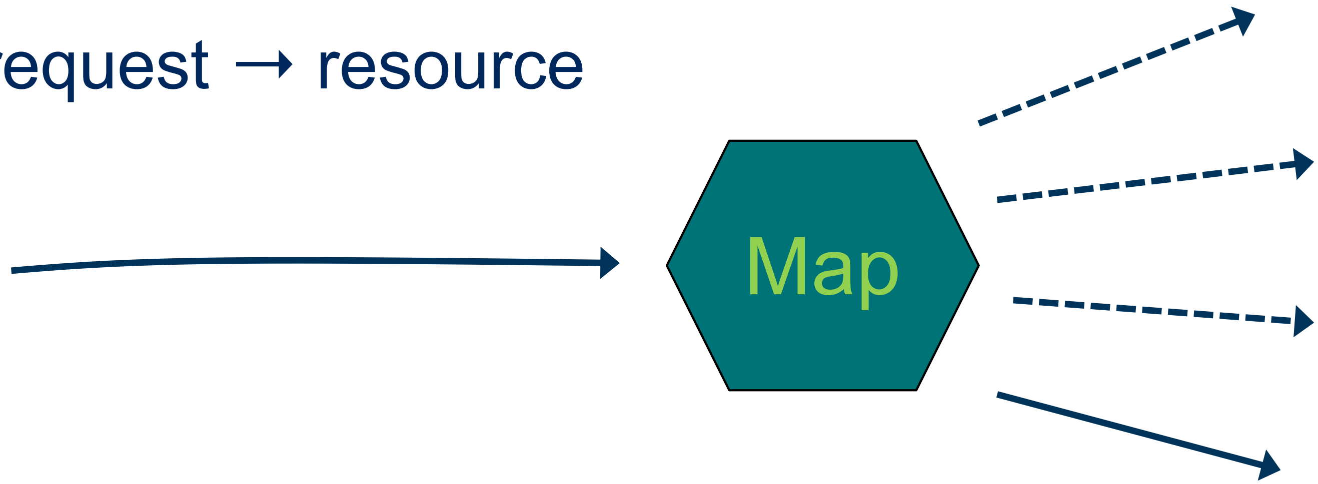


So we want to provide a highly available service



So we want to provide a highly available service

request → resource

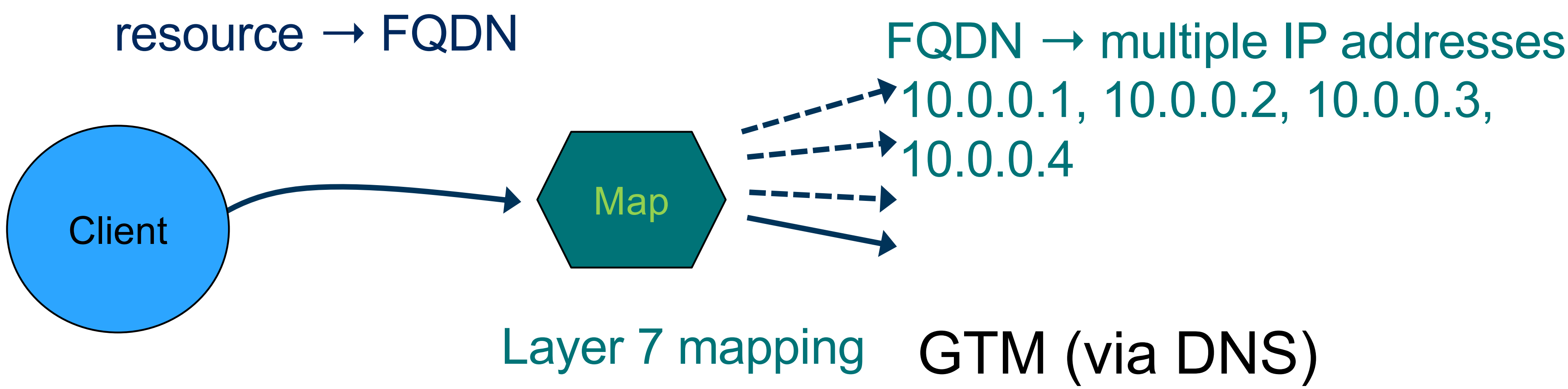


reply →

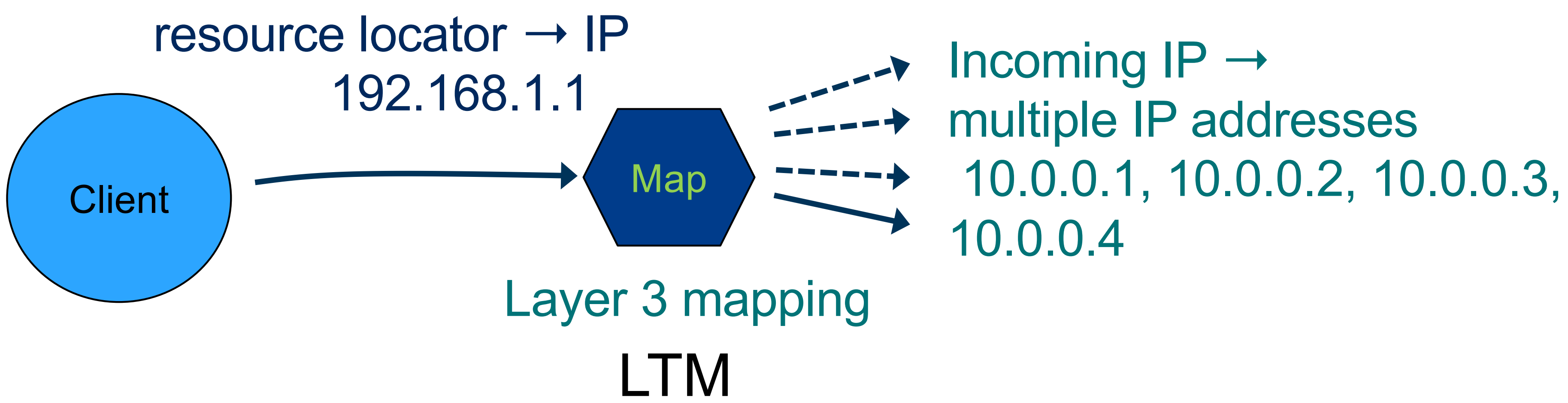
multiple resource providers

i.e. high availability

So we want to provide a highly available service

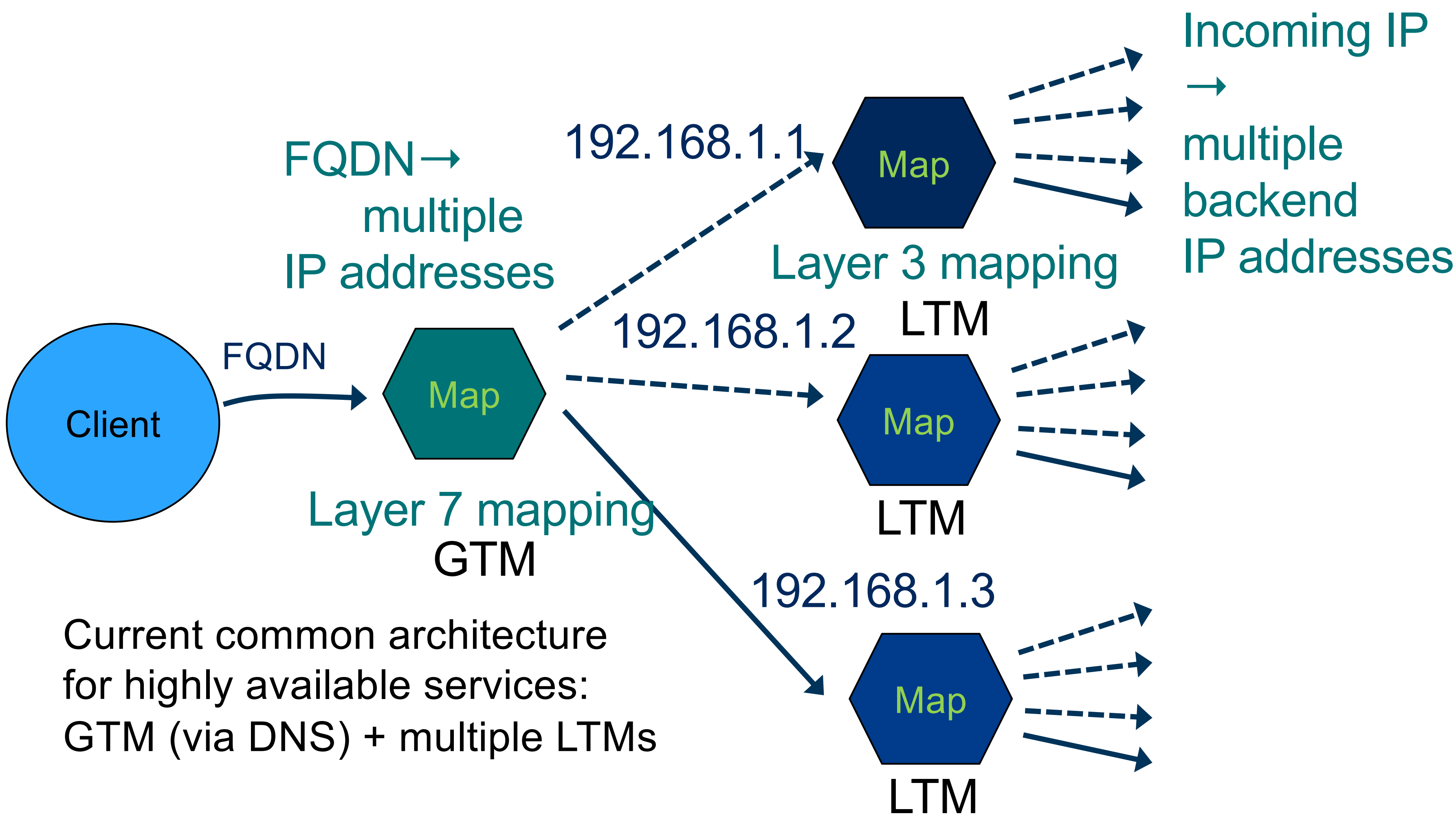


So we want to provide a highly available service



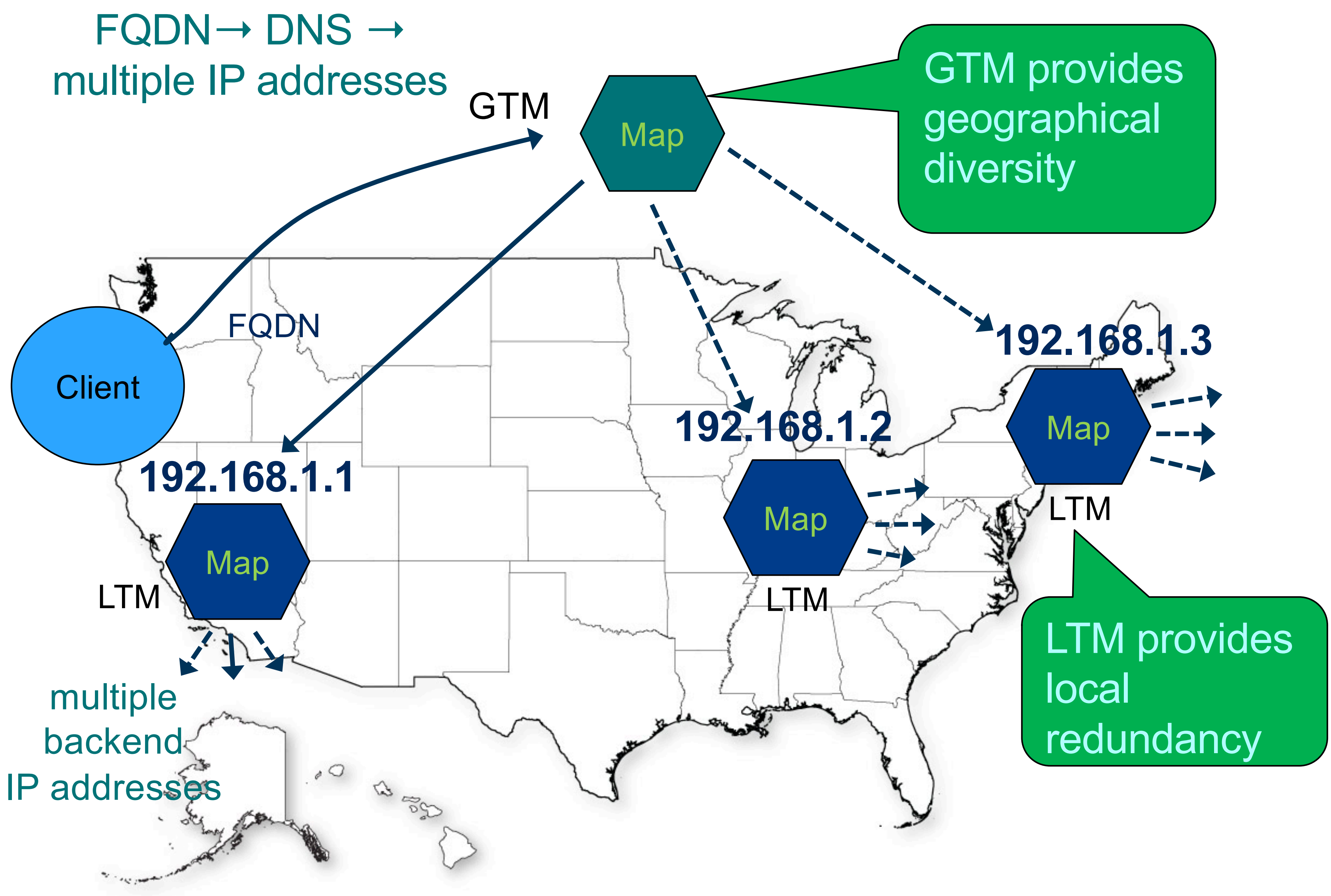
Local traffic manager: local backend servers

So we want to provide a highly available service



Current common architecture
for highly available services:
GTM (via DNS) + multiple LTMs

So we want to provide a highly available service



Issues with the DNS based architecture

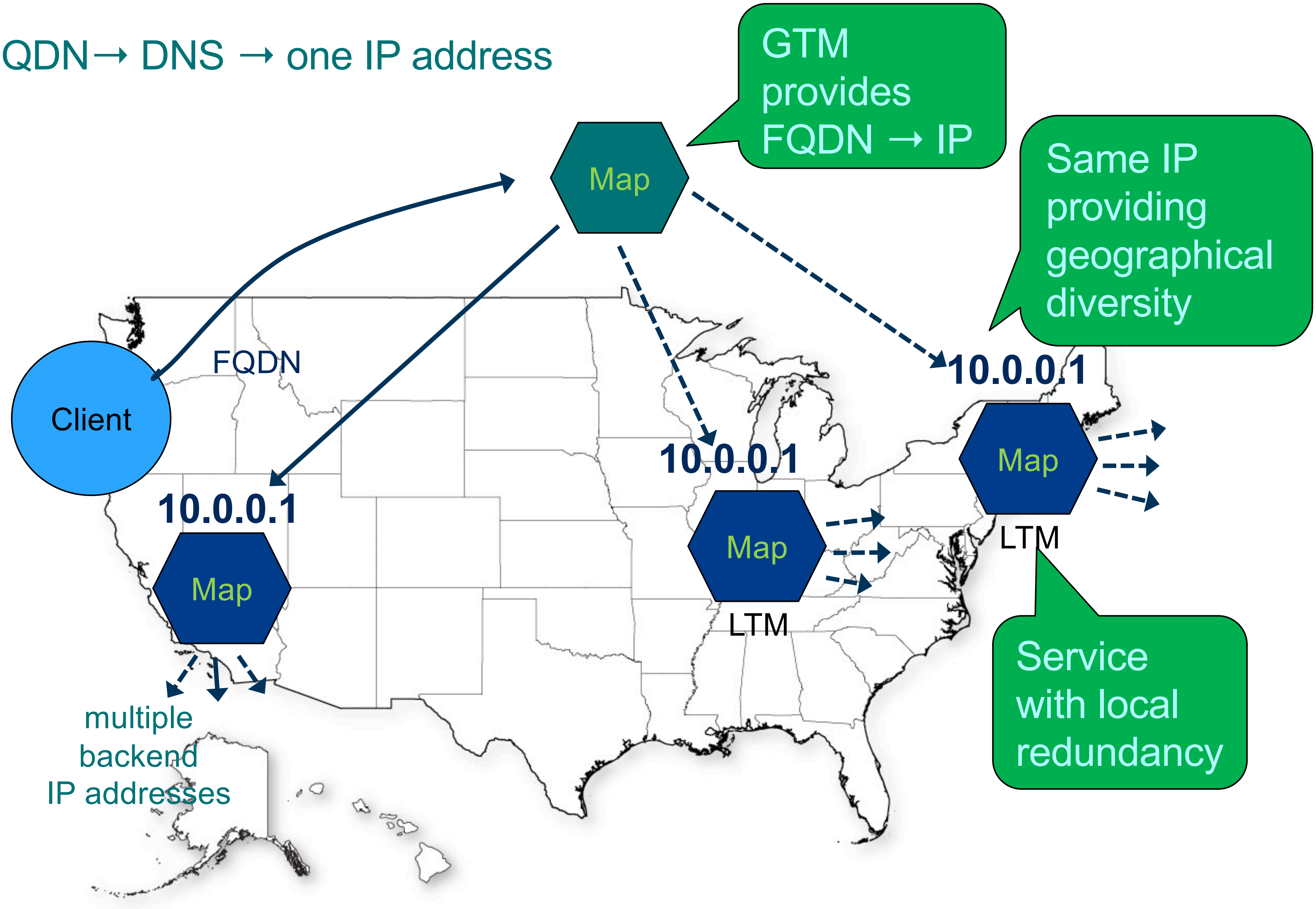
- Dependent on client behavior
  - Client can cache results indefinitely
  - Client may not receive service even though there are servers available (before cache timeout)
- No inherent leverage of proximity information present in the network (routing) layer, resulting in loss of performance
  - Client on the west coast can be mapped to LTM on the east coast
- Inflexible traffic control:
  - Local DNS resolver become the unit of traffic management
  - eDNS client subnet option can forward client subnets, but subnet mapping granularity is decided a priori, may face scalability issues

How could the issues be addressed

- Dependent on client behavior
  - *One possible solution: results obtained by client are always valid, e.g. DNS lookup will give one IP address that is always valid, the service IP*
- No inherent leverage of proximity information present in the network (routing) layer, resulting in loss of performance
  - *One possible solution: use the routing layer for packet forwarding, since it has proximity ("cost") information*
- Inflexible traffic control:
  - *One possible solution: gateways that intercept packets to the service IP can direct traffic to the appropriate host (closest, or policy based)*

Emergence of Anycast as part of the solution

FQDN→ DNS → one IP address

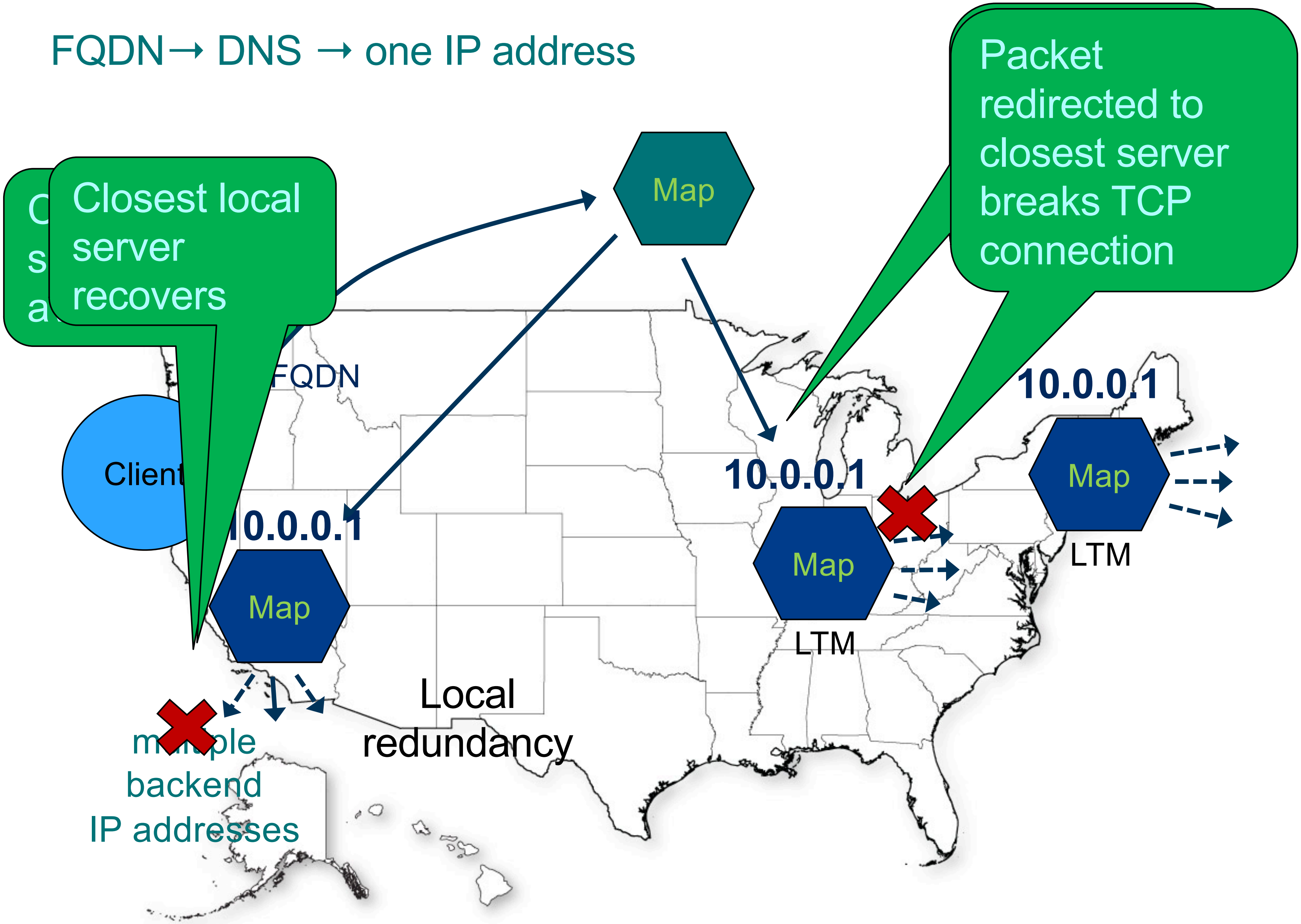


Emergence of Anycast as part of the solution

- So, with Anycast the following issues are resolved:
  - ~~Dependent on client behavior~~
  - ~~No inherent leverage of proximity information present in the network (routing) layer, resulting in loss of performance~~
- But
  - Since much of Internet traffic is over TCP, how would traffic redirection work with TCP flows in case of failover and recovery?

Emergence of Anycast as part of the solution

FQDN→ DNS → one IP address



Emergence of Anycast as part of the solution

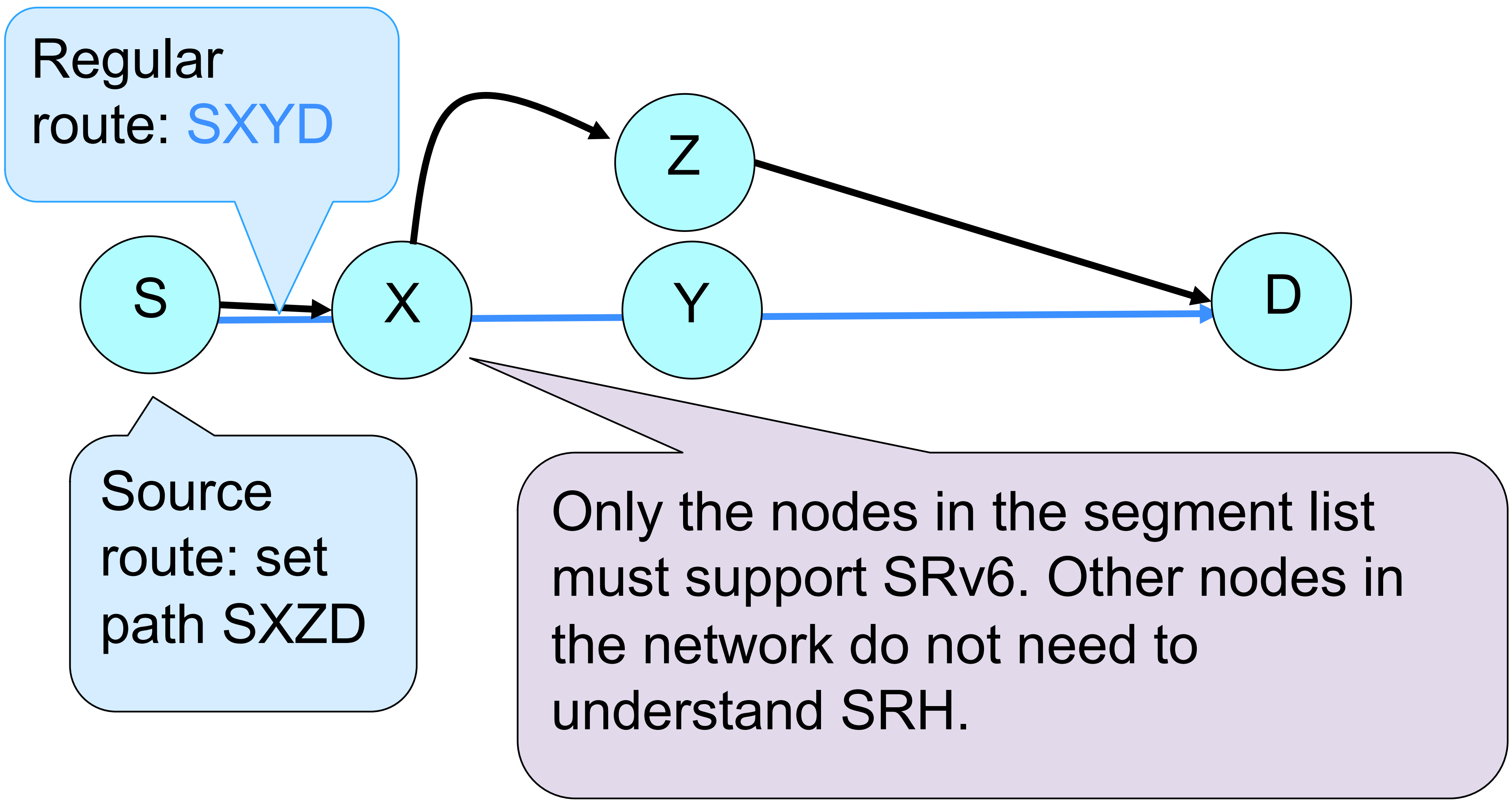
- Anycast and issues that remain
  - ~~Issue: dependent on client behavior~~
  - ~~Issue: no inherent leverage of proximity information present in the network (routing) layer, resulting in loss of performance~~
- Issue: inflexible traffic control
  - *One possible solution: gateways that intercept packets to the service IP can direct traffic to the appropriate host (closest, or policy based)*
- Issue: TCP connection stability
  - *One possible solution: the same gateways that intercept packets to the service IP can handle TCP state so that recovery of a local server will not break connection*
- How can we redirect packets in an IPv6 network?

Segment Routing in IPv6 and what it can do

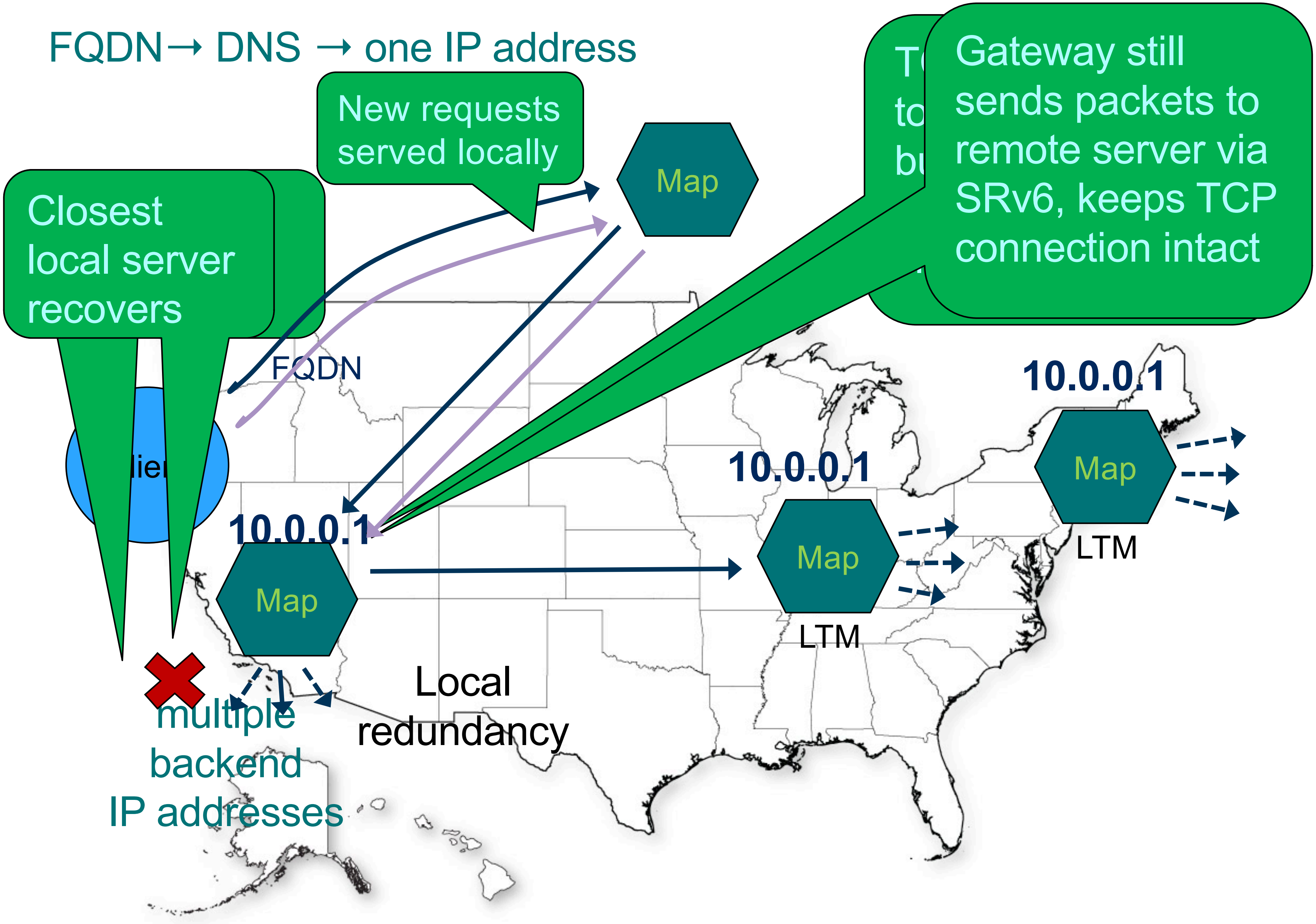
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Last Entry   |     Flags     |              Tag              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

- Segment routing header (SRH) is a type of routing extension header.
  - Routing extension header in IPv6 is defined in RFC 8200
- Segment routing header current status: IETF draft - IPv6 Segment Routing Header (SRH):
  - Segment routing header contains a list of IPv6 addresses defined at the source that a packet traverses on its way to the destination

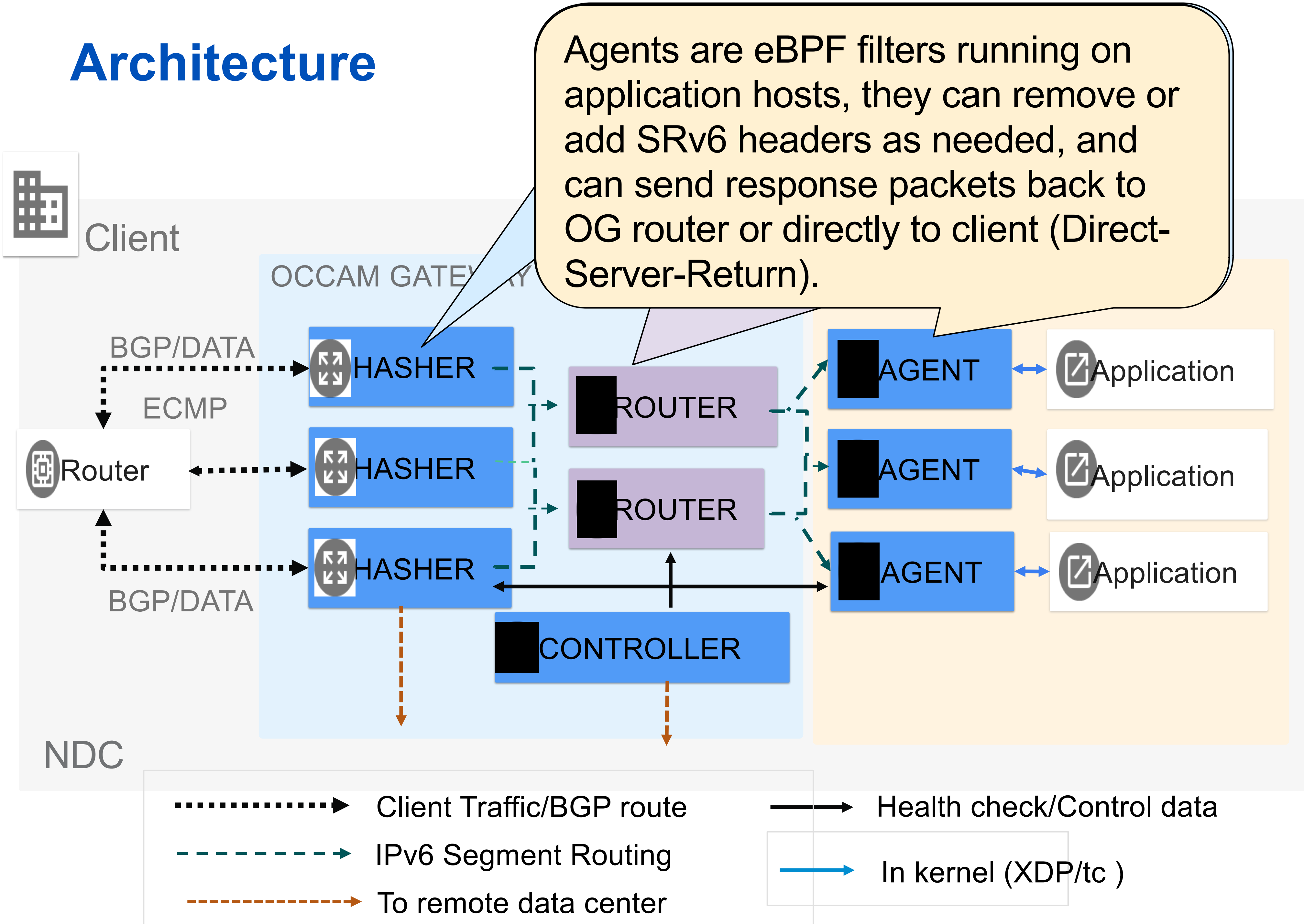
Segment Routing in IPv6 and what it can do



Segment Routing in IPv6 and what it can do



Architecture



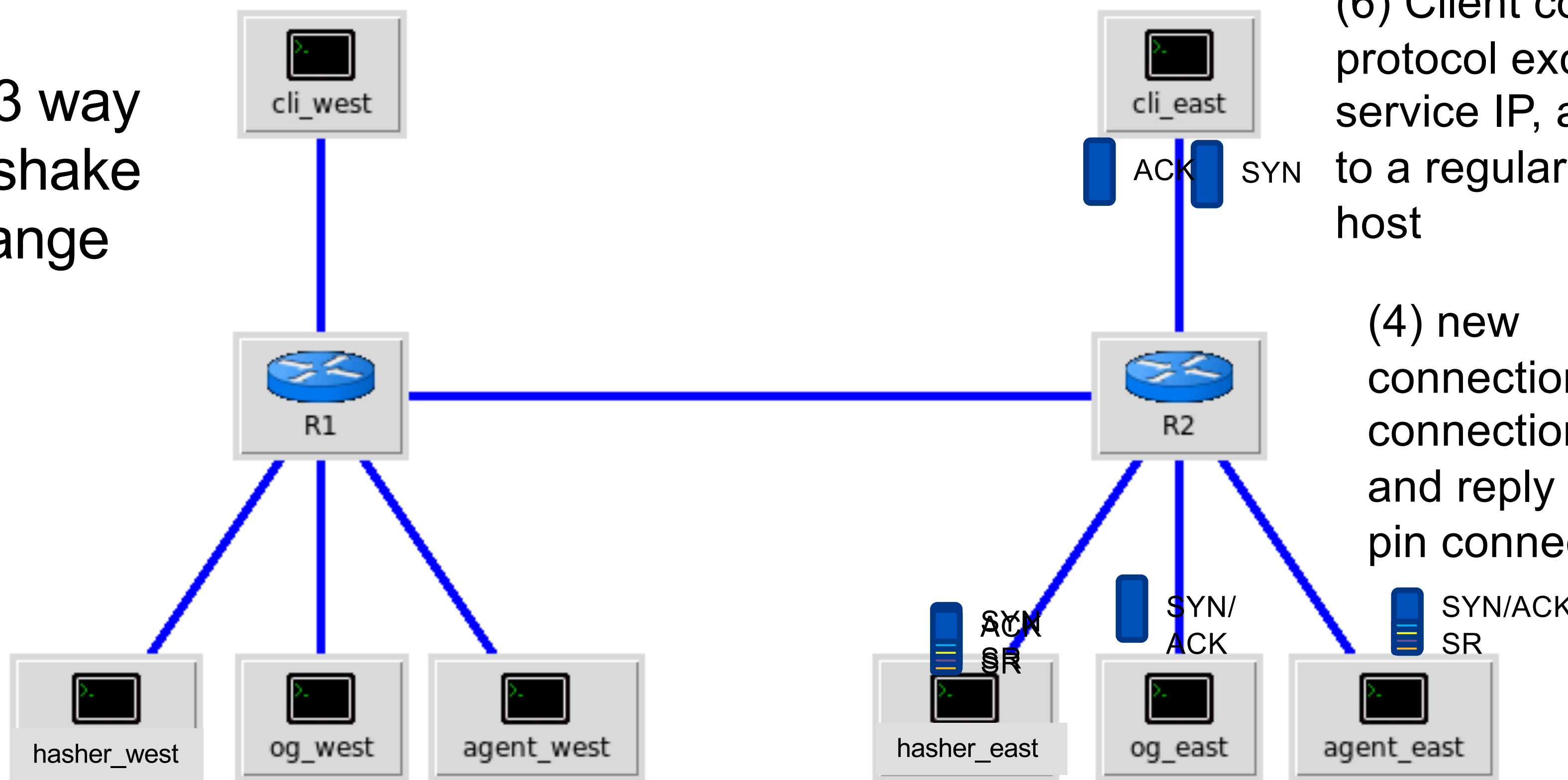
OG: Local Accept

TCP 3 way handshake exchange

(1) Client initiate connection, send packet to service IP (anycast addr)

(6) Client continue protocol exchange to service IP, as if talking to a regular remote host

(4) new connection: track connection, accept and reply to OG to pin connection

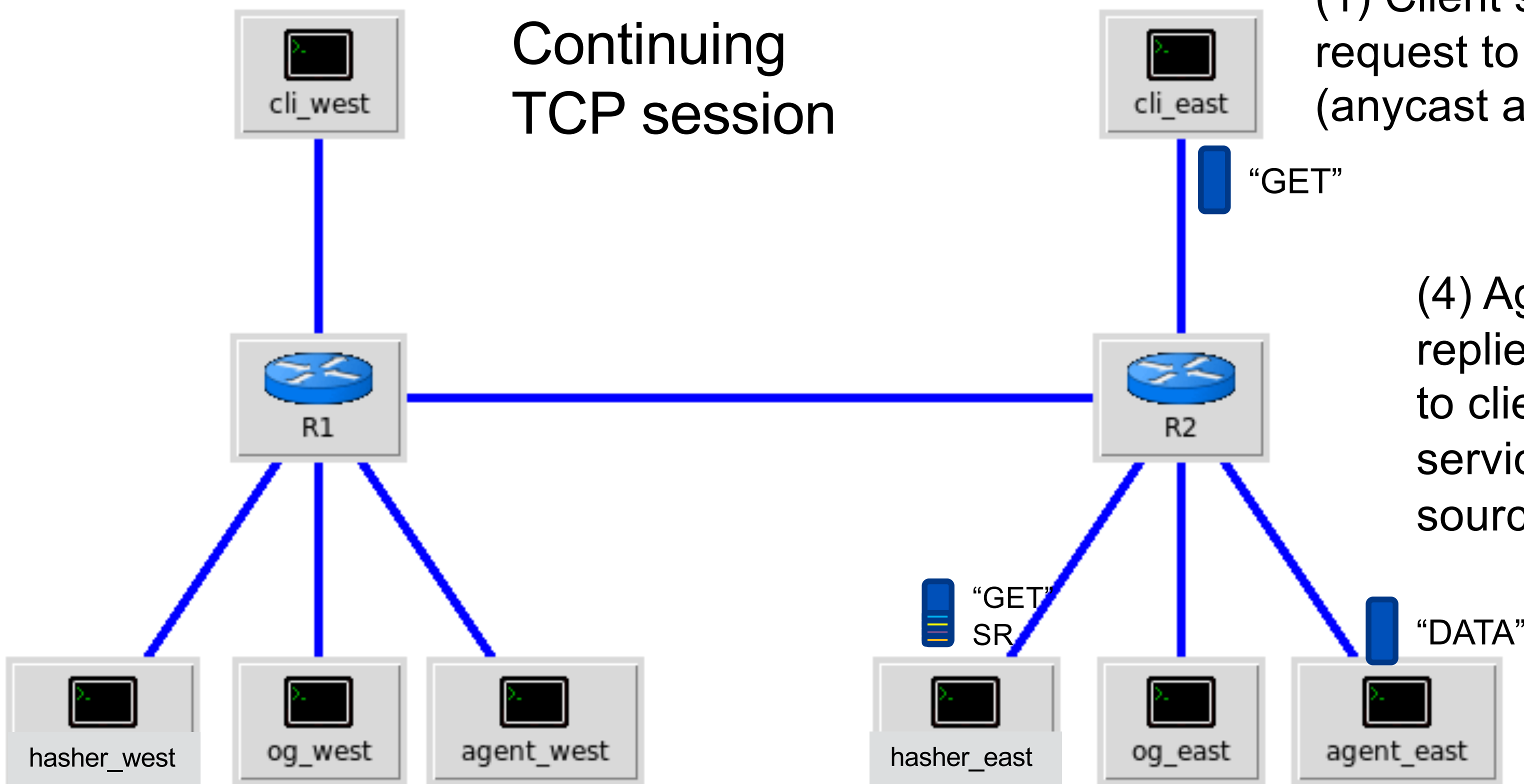


(2) Hasher picks a router: add segment list and forward to router
(3) Router picks a backend server: update segment and forward to agent
(5) Remove segment list, forward to client, use service IP as source
(7) Hasher adds segments, sends to Router, Router checks it's pinned flow, update segments, forward to agent handling flow

OG: Local Accept

Continuing TCP session

(1) Client sends request to service IP (anycast addr)



(4) Agent replies directly to client, using service IP as source addr

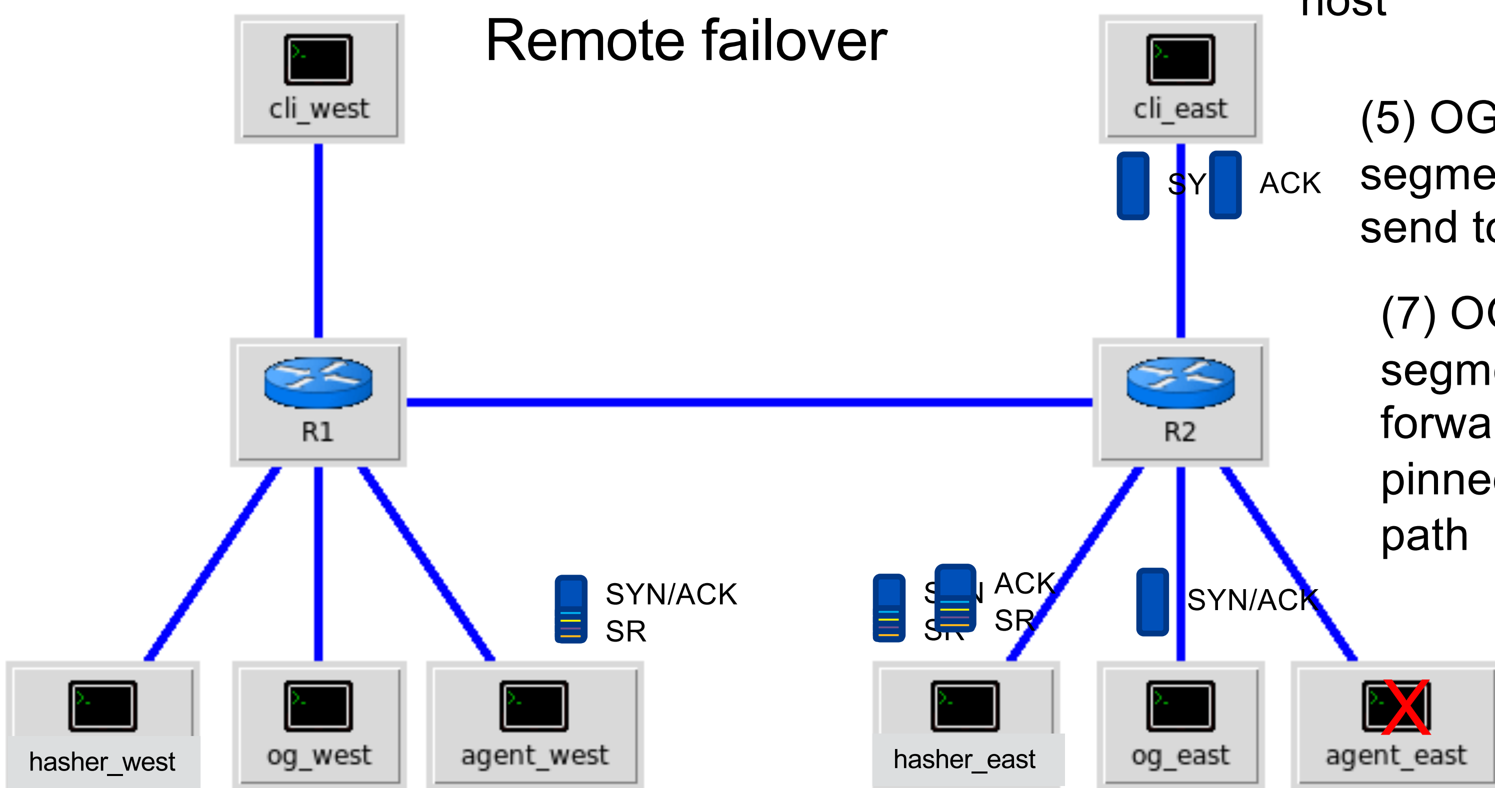
(2) Hasher adds segment list, sends to Router

(3) Router updates segment list and forward to agent

OG: Remote Failover

(1) Client initiate connection, send packet to service IP (anycast addr)

(6) Client continue protocol exchange to service IP, as if talking to a regular remote host



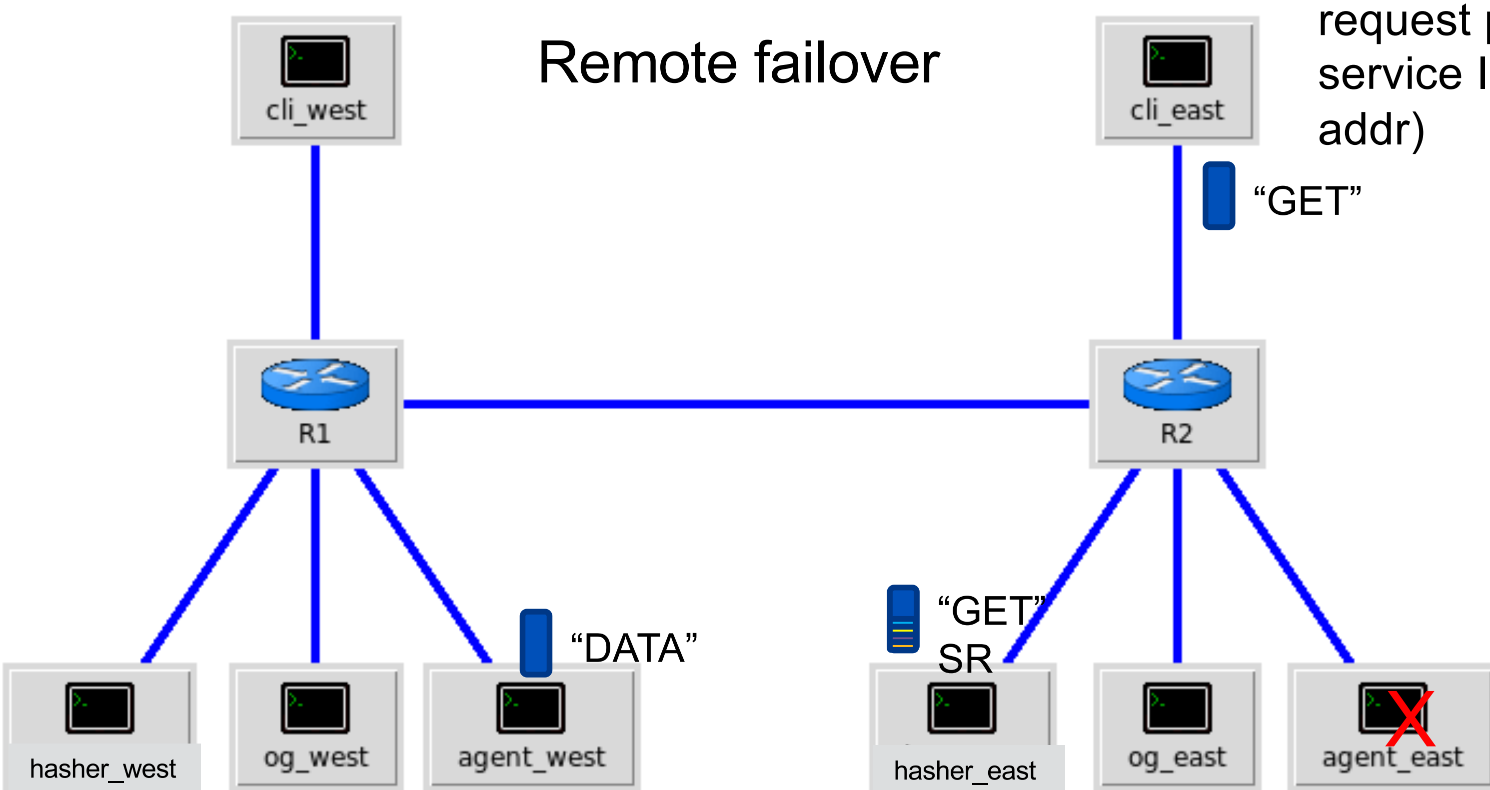
(5) OG strips segment list, send to client

(7) OG adds segment list, forwards along the pinned connection path

(2) new flow, send to OG router but local server unhealthy, forward to remote OG cluster
(3) remote OG has healthy server, accepts connection, forward to agent
(4) agent tracks new connection, reply to OG to pin connection; packet is sent back to forwarding OG to also pin connection

OG: Remote Failover

(1) Client sends request packet to service IP (anycast addr)



(2) OG add segment list and forward to remote OG cluster according to connection table

(3) agent replies directly to client (DSR), using service IP as source addr

Failover as an instance of policy based traffic management

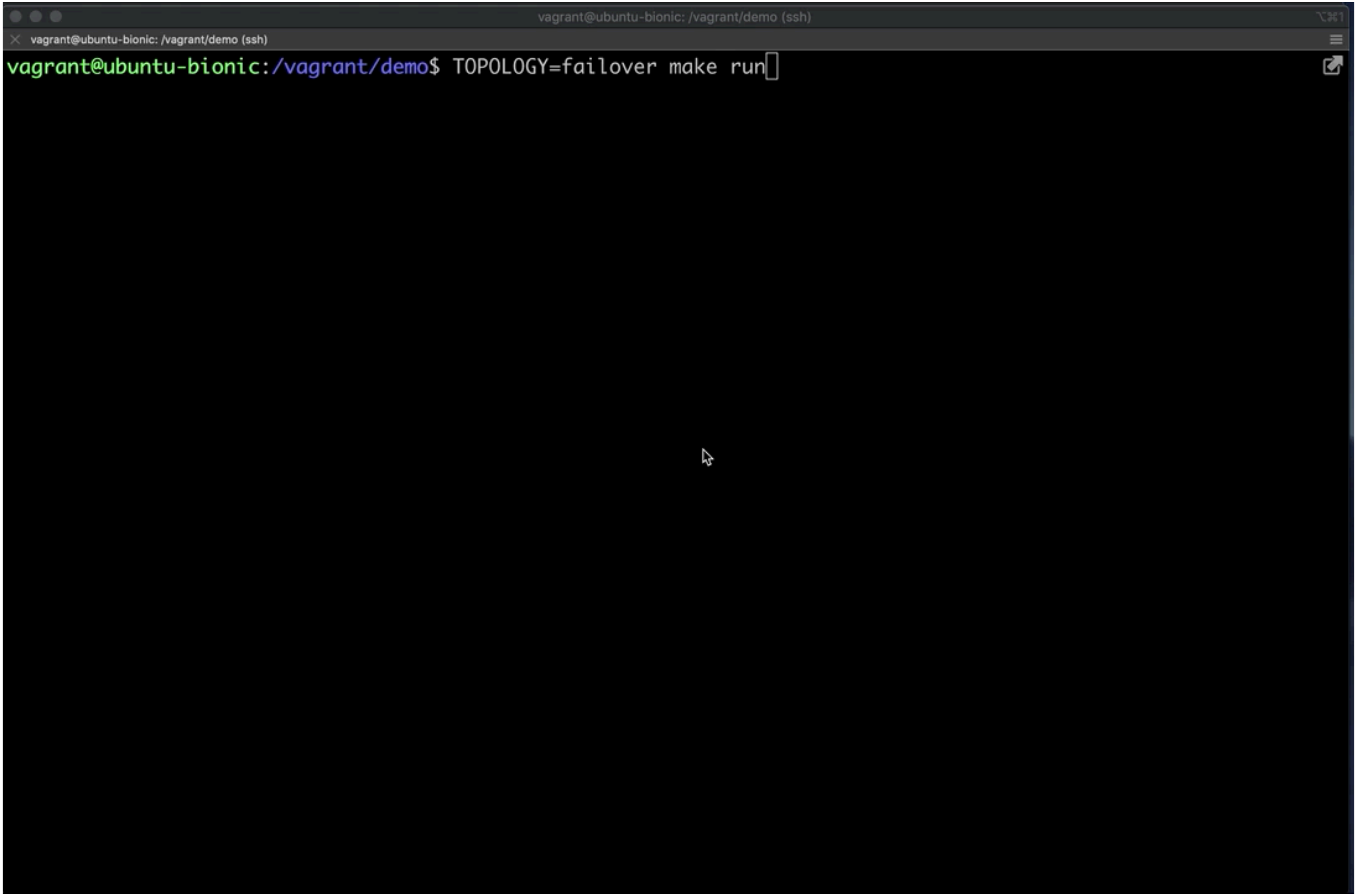
- OG nodes can act as gateways to incoming traffic
- Shape traffic based on the needs of backend server
- Create policy profiles for traffic management:
  - Example: when incoming traffic to local cluster hits 90% of cluster capacity, start forwarding traffic to remote cluster
  - *Failover: cluster capacity dropped to 0, any new incoming traffic is forwarded to remote cluster*

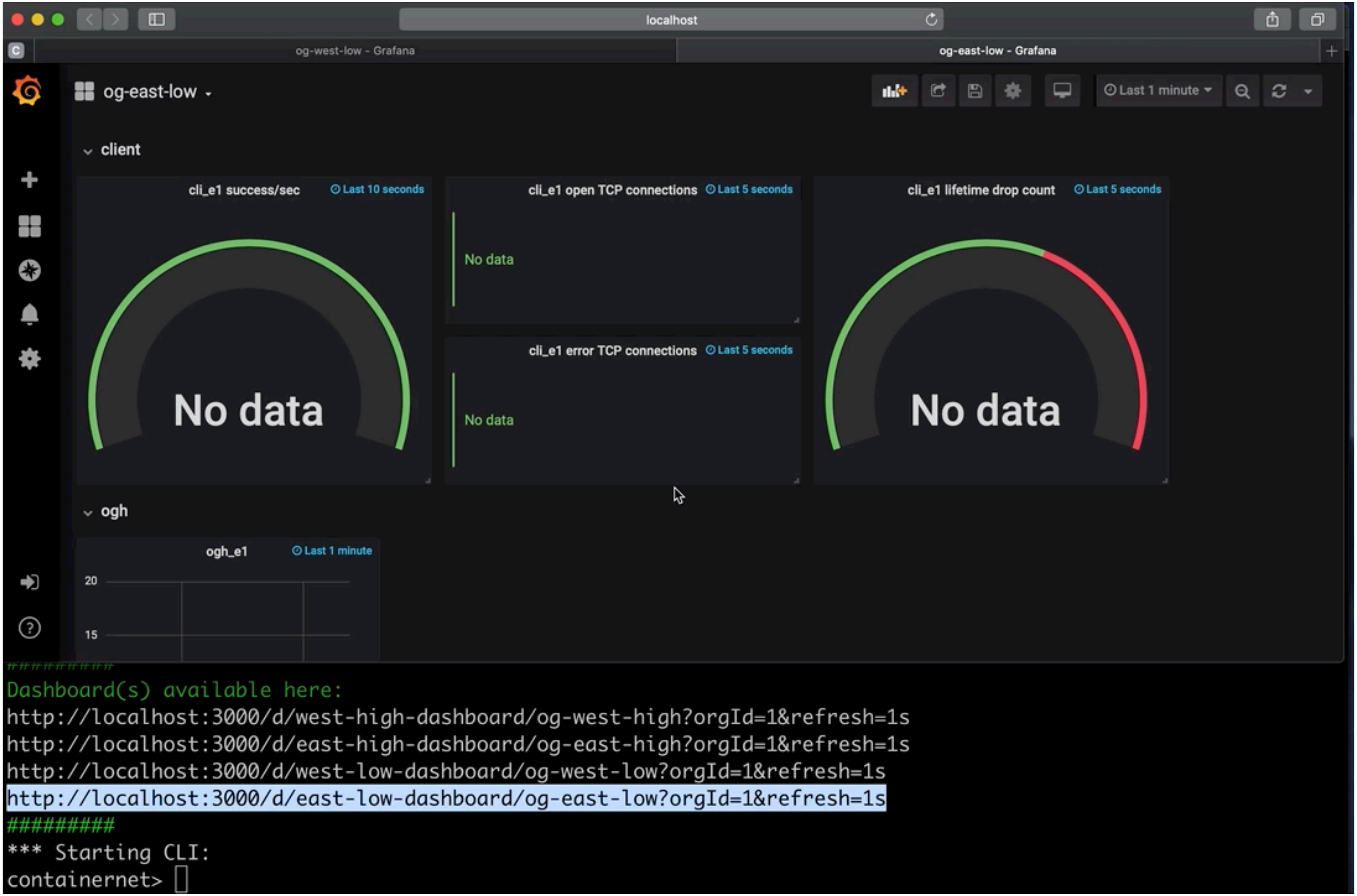
Demo in Containernet

- What is containernet?
  - Port of Mininet that supports running docker containers as nodes in a network
  - Mininet is a network emulation orchestration system, running a collection of end-hosts (nodes, links, switches), to provide an “instant virtual network on your laptop”
  - Setup the topology desired, and start traffic sources, routing software, etc
  - Software runs as-is, interacting with real network stack, at wall clock speed
    - When TCP BBR was released as a linux module, it could be run in Mininet
    - Shared resources limited by hardware speed (cannot emulate link speed faster than supported by underlying hardware)
- Linux only, Ethernet links

Demo in Containernet

- OG nodes (hasher, router) uses DPDK for fast packet processing.
  - Code written in Rust using the Netbricks framework
- Agent (running in backend servers) implemented via extended Berkeley Packet Filter (eBPF)
  - Code written in C using bcc for code load on XDP for packet ingress and at tc for packet egress
- Routers in the network run quagga/zebra
- OG nodes run GoBGP to peer with Routers
- Nodes running consul for health check and status
- Nodes post stats to an InfluxDB node, which we can see on a Grafana dashboard


vagrant@ubuntu-bionic: /vagrant/demo (ssh)
vagrant@ubuntu-bionic: /vagrant/demo (ssh)
vagrant@ubuntu-bionic:/vagrant/demo$ TOPOLOGY=failover make run


localhost
og-west-low - Grafana
og-east-low - Grafana
og-east-low
Last 1 minute
client
cli_e1 success/sec
Last 10 seconds
No data
cli_e1 open TCP connections
Last 5 seconds
No data
cli_e1 error TCP connections
Last 5 seconds
No data
cli_e1 lifetime drop count
Last 5 seconds
No data
ogh
ogh_e1
Last 1 minute
20
15
Dashboard(s) available here:
http://localhost:3000/d/west-high-dashboard/og-west-high?orgId=1&refresh=1s
http://localhost:3000/d/east-high-dashboard/og-east-high?orgId=1&refresh=1s
http://localhost:3000/d/west-low-dashboard/og-west-low?orgId=1&refresh=1s
http://localhost:3000/d/east-low-dashboard/og-east-low?orgId=1&refresh=1s
#########
*** Starting CLI:
containernet>

vagrant@ubuntu-bionic: /vagrant (ssh)

vagrant@ubuntu-bionic: /vagrant (ssh)

```
vagrant@ubuntu-bionic:/vagrant$ c
```

vagrant@ubuntu-bionic: /vagrant (ssh)

```
vagrant@ubuntu-bionic:/vagrant$
```

vagrant@ubuntu-bionic: /vagrant/demo (ssh)

```
Running node-specific post-start task for ogr_w4
Running node-specific post-start task for agent_w1
Running node-specific post-start task for agent_w2
Running node-specific post-start task for agent_w3
Running node-specific post-start task for cli_w1
Running node-specific post-start task for mon_w1
Topology "recovery" started!

#########
Dashboard(s) available here:
http://localhost:3000/d/west-high-dashboard/og-west-high?orgId=1&refresh=1s
http://localhost:3000/d/west-low-dashboard/og-west-low?orgId=1&refresh=1s
#########
*** Starting CLI:
containernet>
```

Conclusion

- Presented a common architecture for highly available services that addresses issues we have found in a DNS based system
- How Anycast and SRv6 can together provide a solution that:
  - Provides clients with the closest available server, leveraging the network layer to provide proximity
  - Speed of failover is as fast as detection of downed local server, i.e., no longer depends on client behavior
  - OG traffic forwarding can be policy based and not only for failover, i.e., we have far greater flexibility in traffic management

Thank You

COMCAST