Egress traffic controller using Telemetry and Service Layer APIs

Mikhail Korshunov
Technical Marketing @ Cisco SP
May 10th 2018

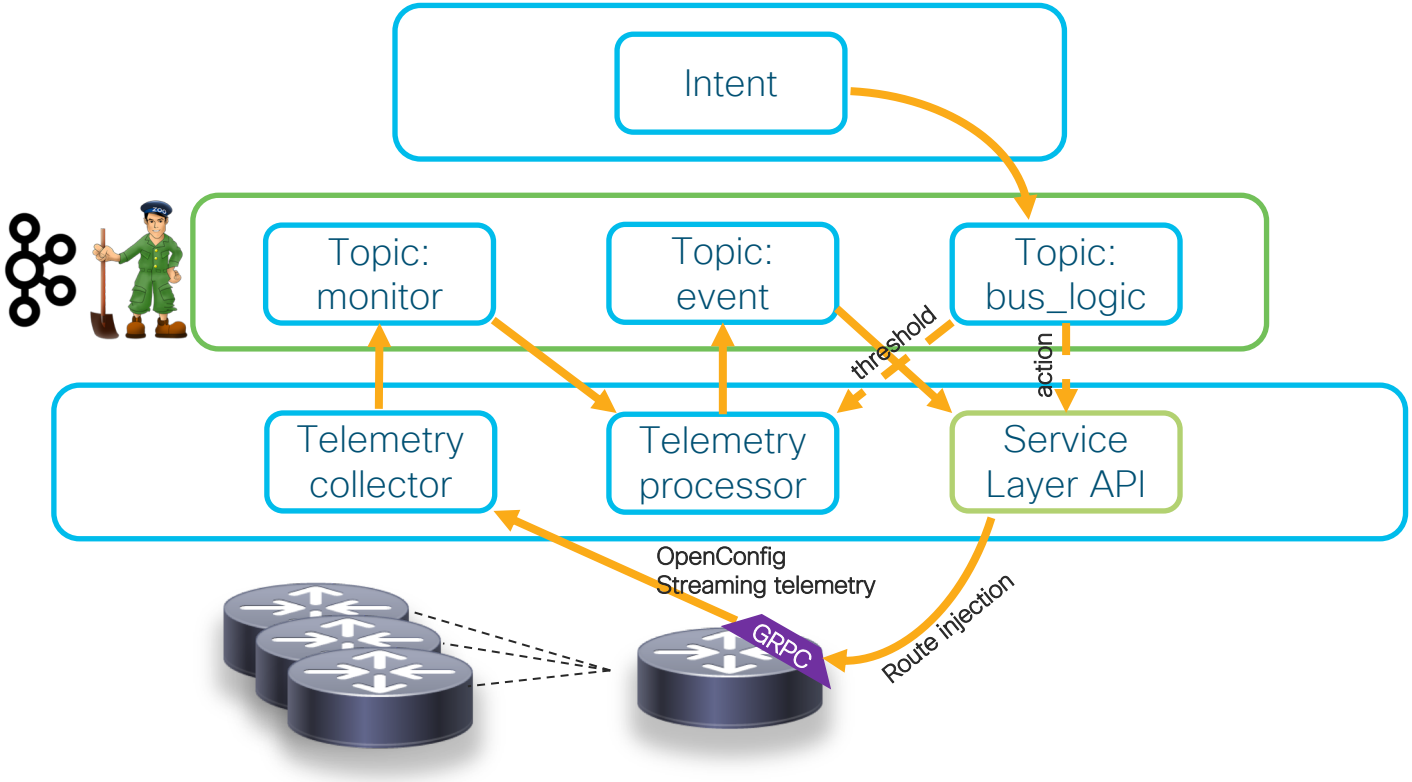
What Should a Controller Look Like?

- Flexibility
- Scale the development process
- Run-time Extensibility & Modularity
- Adapt to data schemas (models) discovered in the network
- Performance & Scale

Egress controller Use Cases

- Open platform for network programmability;
- Users/contributors can add value at any level;
- Enables controller capability for any network size and scale;
- Tool independent, flexible in stack;

Egress controller High Level Structure

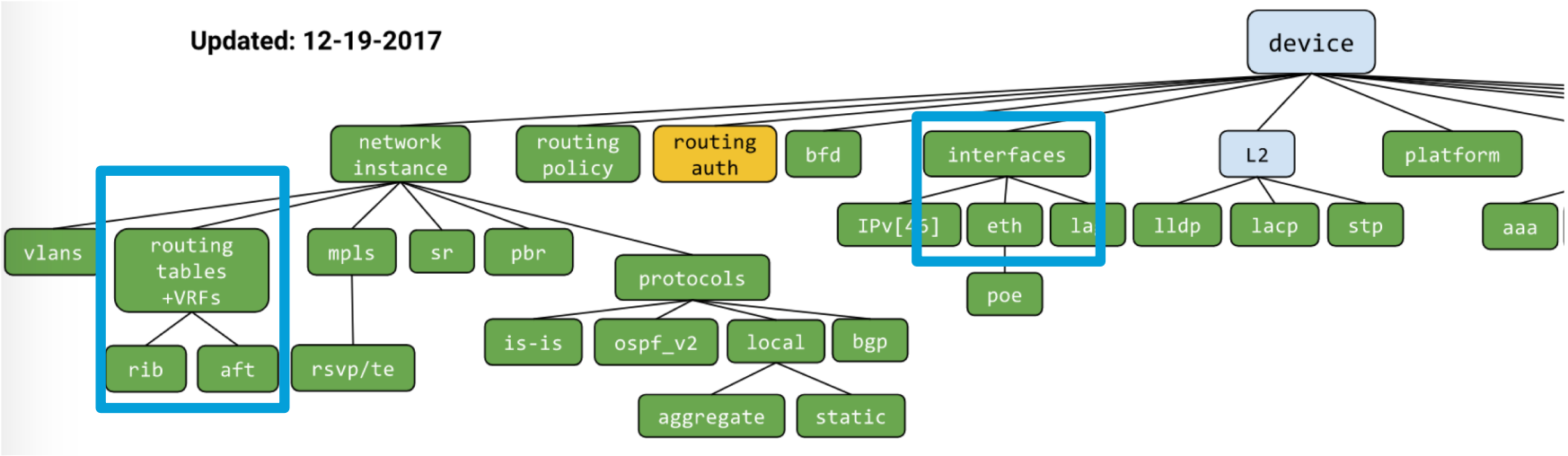


Slicing – Telemetry

- New approach to get insights about your network.
- Push, not pull
- Data-Model driven
- Analytics ready, a lot of open sourced tools to store and work with received data.
- OpenConfig models supported across vendors: http://www.openconfig.net/projects/models/

Slicing – Telemetry #2

For controller we will stream models from rib (**`openconfig-rib-bgp-tables`**) and interfaces (**`openconfig-if-ip`**). Streamed data will go into collector for normalization.



OpenConfig [models]

Slicing – Data bus - Kafka

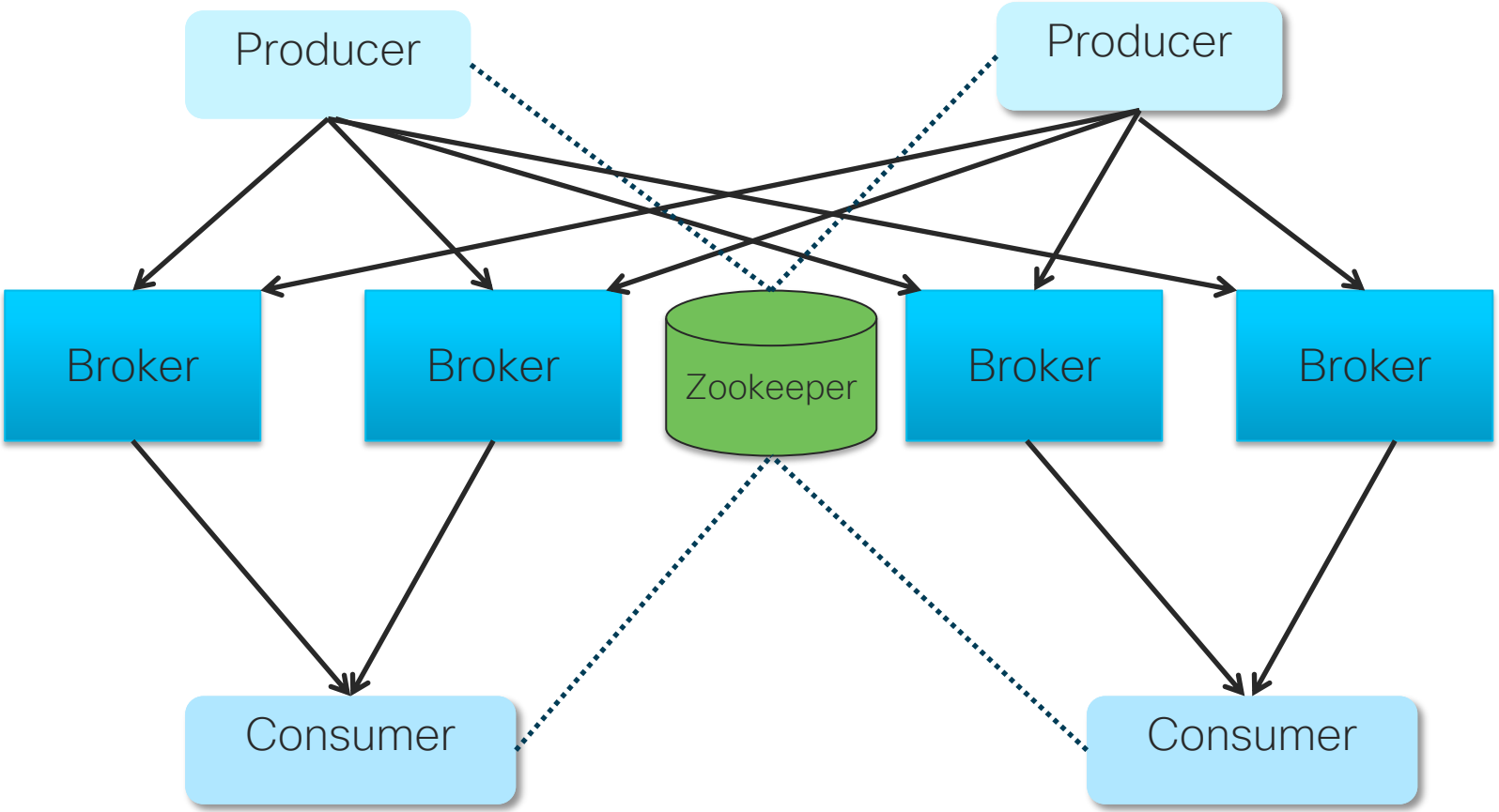
Kafka is Distributed Streaming Platform

- Publish and Subscribe to streams of records
- Fault tolerant storage
- Process records as they occur

Kafka offers:

- Very high performance
- Elastically scalable
- Low operational overhead
- Durable, highly available

Slicing – Data bus - Kafka



Slicing – Data bus - ZooKeeper

Apache ZooKeeper is an effort to develop and maintain an open-source server which enables highly reliable distributed coordination.

ZooKeeper maintaining configuration information, naming, providing distributed synchronization, and providing group services.

Slicing – Intent – Craft you own app

- We will use Python and Flask as a popular choice and easy to start solution.
- Service Layer APIs would be utilized to trigger action.
- Pub/Sub mechanism available via pip.

python

Flask

web development,
one drop at a time

```
from flask import Flask
app = Flask(__name__)

@app.route('/announce')
def announce_routes():
    controller.trigger(routes)
    return "Routes announced!"

if __name__ == '__main__':
    app.run()
```

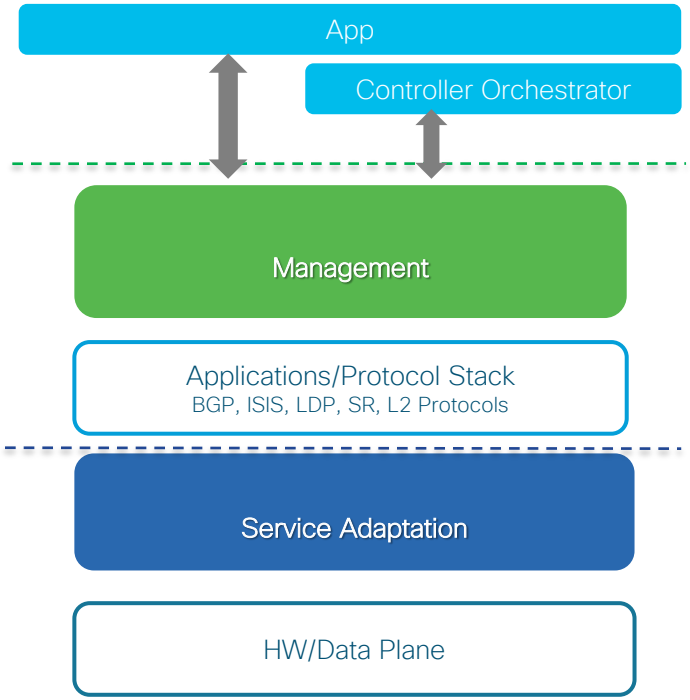
Slicing – Closing the Loop

There are multiple ways to close the loop and initiate action to program the device:

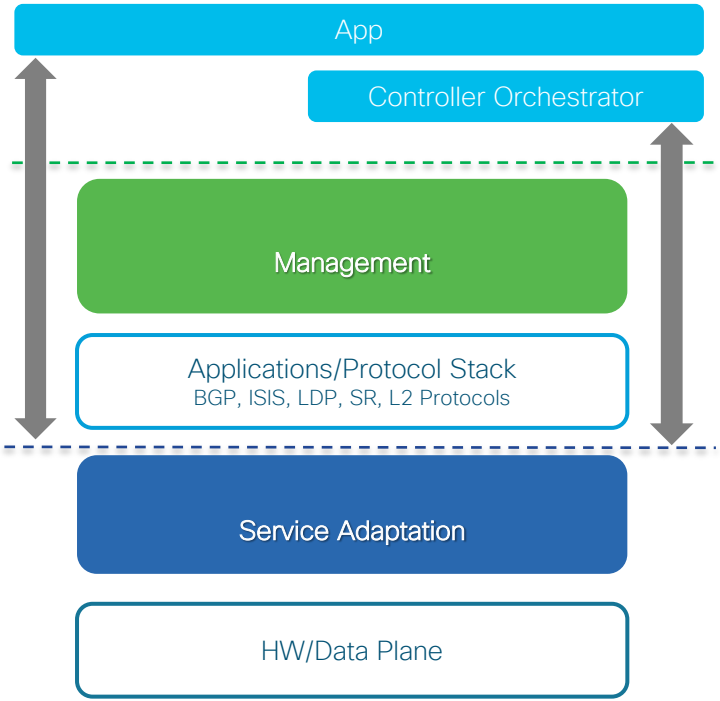
- Netmiko / NAPALM – unified across multiple vendors. Simplification compare to traditional paramiko connection.
- NETCONF / YANG – apply model directly on the box;
- Configuration management tools: Ansible, Puppet or Salt.
- RIB API – used in controller example. Lowest level communication.

Network Device Programmability

Model-Driven Manageability



Service-Layer APIs



Slicing – Agent code to program the box 1/3

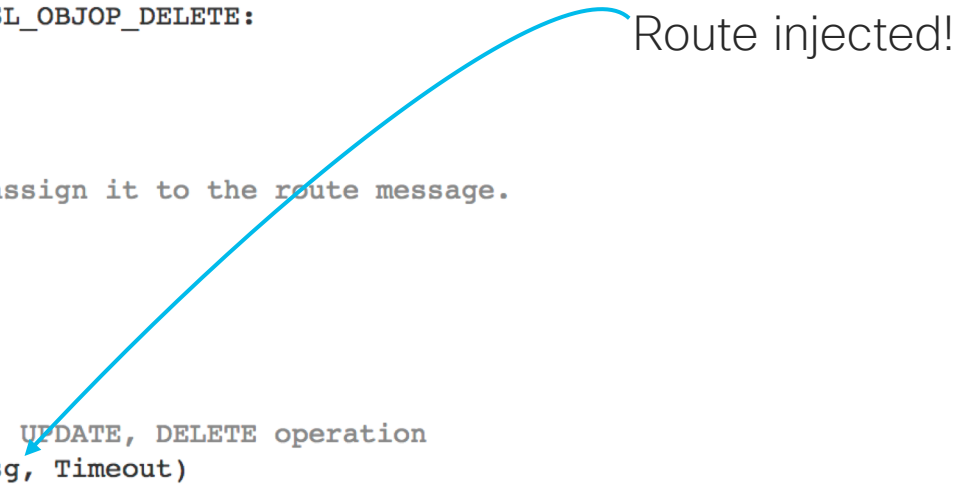
```
def route_operation(channel, oper):
    # Create the gRPC stub.
    stub = sl_route_ipv4_pb2.beta_create_SLRoutev4Oper_stub(channel)
    # Create an empty list of routes.
    routeList = []
    # Create the SLRoutev4Msg message holding the SLRoutev4 object list
    rtMsg = sl_route_ipv4_pb2.SLRoutev4Msg()

    # Fill in the message attributes attributes.
    # VRF Name
    rtMsg.VrfName = 'default'
    # Fill in the routes
    # Create an SLRoutev4 object and set its attributes
    route = sl_route_ipv4_pb2.SLRoutev4()

    # IP Prefix and length
    route.Prefix = (int(ipaddress.ip_address('20.0.10.0')))
    route.PrefixLen = 24
    # Administrative distance
    route.RouteCommon.AdminDistance = 2
```

Slicing – Agent code to program the box 2/3

```
paths = []

# Create an SLRoutePath path object.
path = sl_route_common_pb2.SLRoutePath()
# Fill in the path attributes.
# Path next hop address
path.NexthopAddress.V4Address = (int(ipaddress.ip_address('10.10.10.1')))
# Next hop interface name
path.NexthopInterface.Name = 'GigabitEthernet0/0/0/0'

# Add the path to the list
paths.append(path)

# Let's create another path as equal cost multi-path (ECMP)
path = sl_route_common_pb2.SLRoutePath()
path.NexthopAddress.V4Address = (
    int(ipaddress.ip_address('10.10.10.2'))
)
path.NexthopInterface.Name = 'GigabitEthernet0/0/0/0'

paths.append(path)
```

Slicing – Agent code to program the box 3/3

```
path = sl_route_common_pb2.SLRoutePath()
path.NexthopAddress.V4Address = (
    int(ipaddress.ip_address('10.10.10.2'))
)
path.NexthopInterface.Name = 'GigabitEthernet0/0/0/0'

paths.append(path)
# Assign the paths to the route object
if oper != sl_common_types_pb2.SL_OBJOP_DELETE:
    route.PathList.extend(paths)

routeList.append(route)

# Done building the routeList, assign it to the route message.
#
rtMsg.Routes.extend(routeList)

# Make an RPC call

Timeout = 10 # Seconds
rtMsg.Oper = oper # Desired ADD, UPDATE, DELETE operation
response = stub.SLRoutev4Op(rtMsg, Timeout)
```



Scalability

- Load on each component of the controller could be distributed;
- Such architecture decoupled by design;
- Python could be replaced to more performant language if needed;
- Kafka available in cluster configuration.

Demo & components walkthrough

Summary

- Controller is built from open-source tools;
- You can introduce new logic and complicate rules as you grow;
- Components are independent from each other;
- Better monitoring -> better sleep.

Resources

- Streaming telemetry – http://www.openconfig.net/projects/telemetry/
- IOS-XR telemetry – https://xrdocs.github.io/telemetry/
- IOS-XR Service Layer API – https://xrdocs.github.io/cisco-service-layer/
- Apache Kafka – https://kafka.apache.org/
- Apache Zookeeper – https://zookeeper.apache.org/