Goal

We want to be able to collect metrics via OpenConfig using telemetryd.

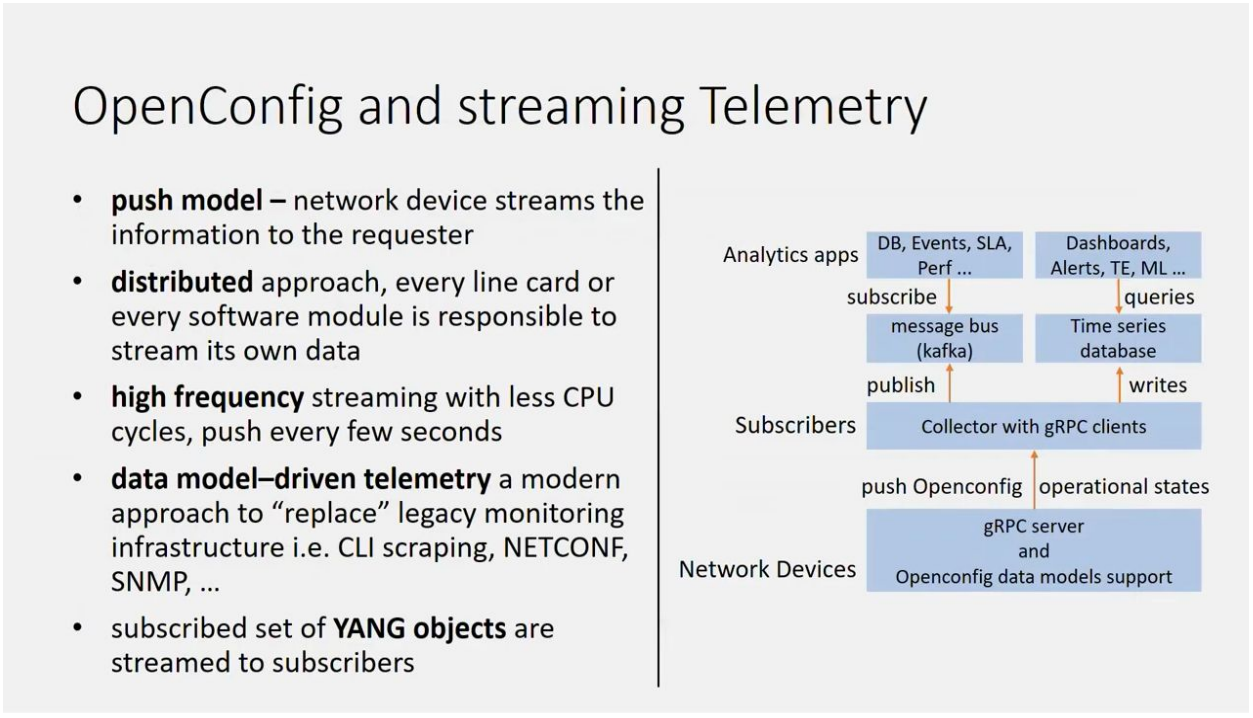
Background

What is OpenConfig?

The OpenConfig specification allows you to initiate a connection via gRPC and subscribe to one or more OpenConfig data paths (strings). The OpenConfig data is then forwarded back through the channel with meta-data and is structured as a list of key-value pairs.

This differs from our current model in telemetryd since we need to both initiate and maintain a long live session (gRPC) with the agent.

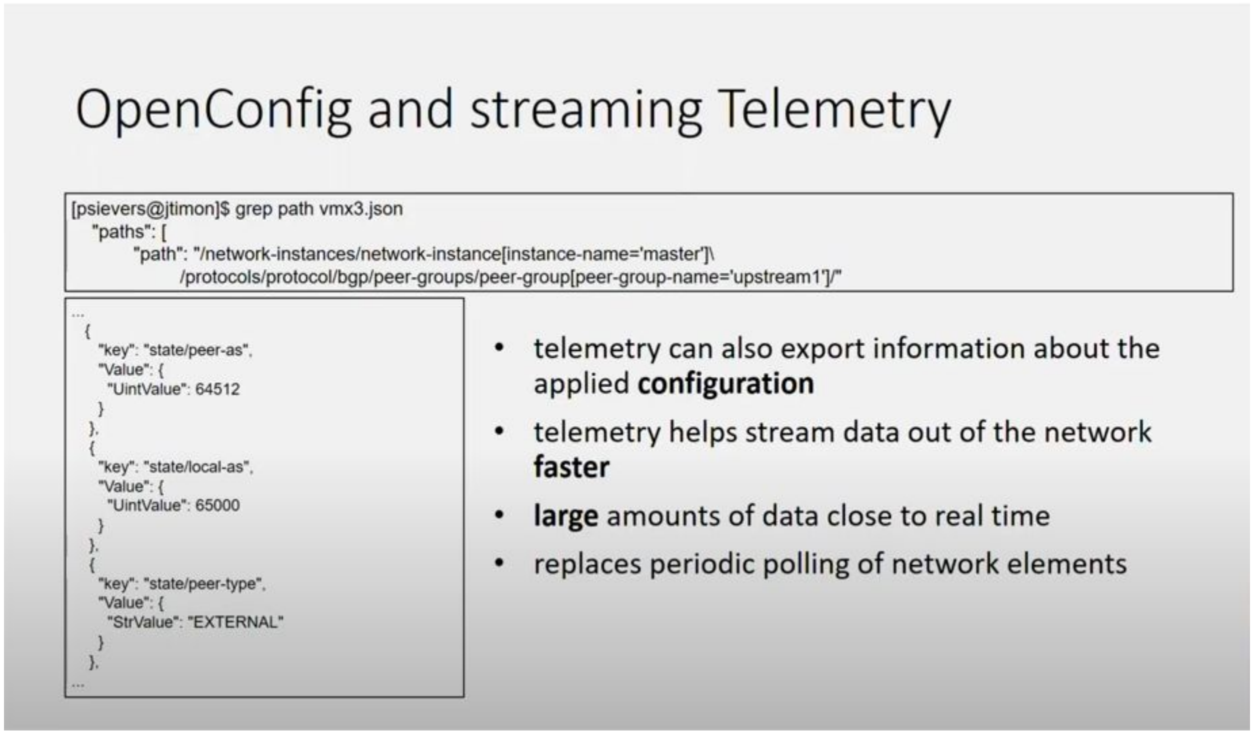
Sourced from: https://www.youtube.com/watch?v=vTicZ0m0szQ



OpenConfig and streaming Telemetry

- **push model –** network device streams the information to the requester
- **distributed** approach, every line card or every software module is responsible to stream its own data
- **high frequency** streaming with less CPU cycles, push every few seconds
- **data model–driven telemetry** a modern approach to “replace” legacy monitoring infrastructure i.e. CLI scraping, NETCONF, SNMP, …
- subscribed set of **YANG objects** are streamed to subscribers

Analytics apps
DB, Events, SLA, Perf …
Dashboards, Alerts, TE, ML …
subscribe
queries
message bus (kafka)
Time series database
publish
writes
Subscribers
Collector with gRPC clients
push Openconfig
operational states
Network Devices
gRPC server and Openconfig data models support



OpenConfig and streaming Telemetry

```
[psievers@jtimon]$ grep path vmx3.json
   "paths": [
         "path": "/network-instances/network-instance[instance-name='master']\
               /protocols/protocol/bgp/peer-groups/peer-group[peer-group-name='upstream1']/"
```

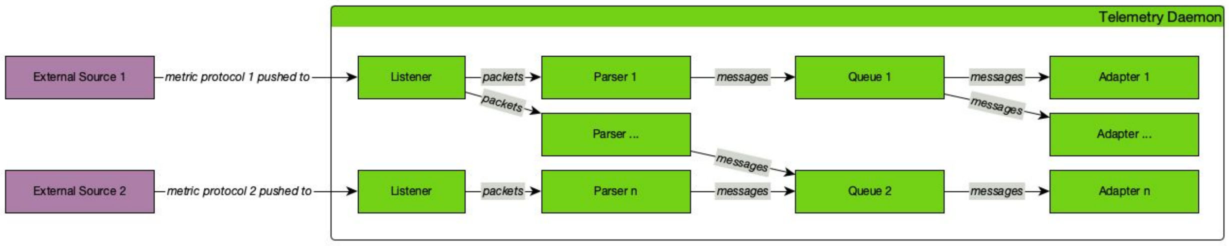
```
...
  {
    "key": "state/peer-as",
    "Value": {
      "UintValue": 64512
    }
  },
  {
    "key": "state/local-as",
    "Value": {
      "UintValue": 65000
    }
  },
  {
    "key": "state/peer-type",
    "Value": {
      "StrValue": "EXTERNAL"
    }
  },
...
```

- telemetry can also export information about the applied **configuration**
- telemetry helps stream data out of the network **faster**
- **large** amounts of data close to real time
- replaces periodic polling of network elements

What is telemetryd?

In Horizon 26.x:

telemetryd currently only listens for metrics pushed by agents.
These are typically forwarded to a load balancer that balances the streams against a pool of Minions.



telemetryd is composed of:

- Daemon
- Listeners
- Parsers
- Queues
- Adapters

The daemon is responsible for the lifecycle of the various components, while the components each focus on a specific task.

For reference see Telemetry Daemon

Solution

Requirements for Telemetryd

Use cases:

1. Streaming telemetry via OpenConfig
2. Streaming telemetry & events using TL1
    - *Unknown to most users and developers OpenNMS has support for streaming events via TL1. While the protocol details differ from OpenConfig, the communication flow for our use case remains the same: initial a connection, subscribe to certain attributes, handle the payloads when received, restart the connection on failure. While designing support for OpenConfig, we should also be able to accommodate other protocols like TL1.*

We need some mechanisms to:

1) Define the agents we want to connect to
2) Initiate and maintain the session
3) Forward and handle received payload

Define the agents we want to connect to

In OpenNMS, agents are normally defined by what we call “IP-services”. An IP-service is a named service attached to an IP interface on a specific node, which is associated with a specific location. Given that this model is already pervasive in OpenNMS, we should also consider using it here.

Some of the agents may require different settings (port, authentication, TLS, etc…) and for this we have normally used the notion of packages with configurable filter expressions & meta-data support for further enrichment.

Agents can be added/removed at runtime, and node attributes may also change in ways that make them start or stop making an existing package filter.

- Administrators should be able to control **which** agents we connect to by managing IP-services
- Administrators should be able to control **how** we communicate with those agents by using a combination of packages, filters & meta-data

Initiate and maintain the session

We need to initiate and maintain sessions with a dynamic list of agents.

Ideally, the solution should be structured so that the component or module only has to deal with the protocol handling and not worry about maintaining the list of agents.

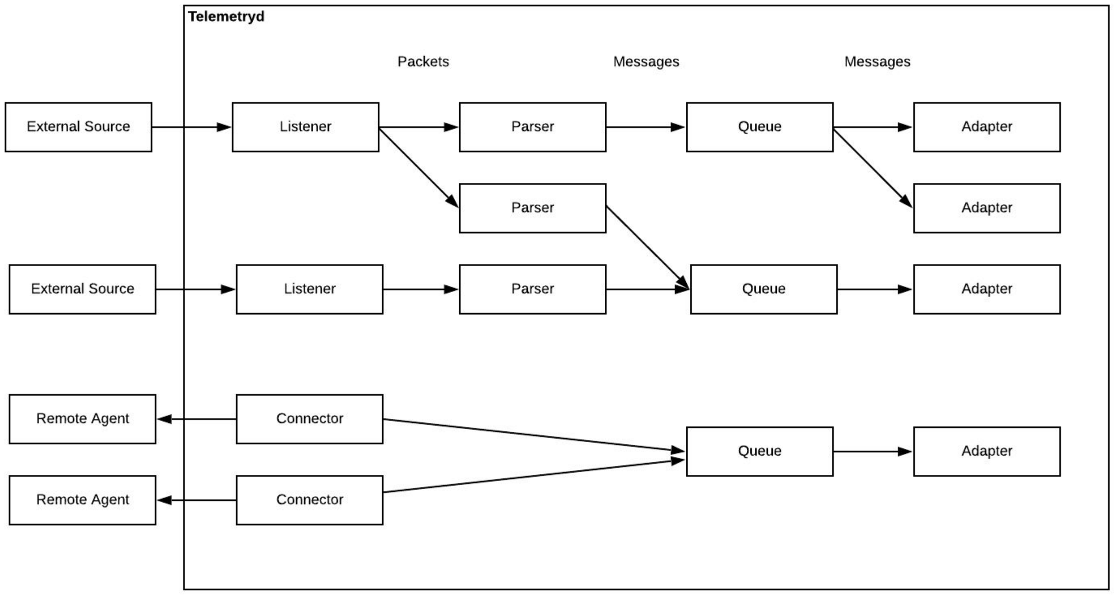
Forward and handle the received payload

We can reuse the existing parse/queue/adapter components for this.

Proposed updates to Telemetryd

Connectors

A "connector" is responsible for initiating & maintaining a session with some "agent".



A new connector is created for every agent it is responsible for connecting to.

Differently than listeners, connectors don't use parsers and dispatch messages directly to the queue. This is done since the lifecycle of the connector doesn't match the lifecycle of the parser (connections are constantly created/destroyed whereas parsers are currently expected to be "long lived"). We may address this in a future release.

Configuration for connectors on the core looks like:

```
<connector name="OpenConfig-Connector"
class-name="org.opennms.netmgt.telemetry.protocols.openconfig.connector.OpenConfigCon
nector"
            service-name="OpenConfig"
            queue="OpenConfig"
            enabled="true">
      <package name="OpenConfig-Default">
         <filter>IPADDR != '0.0.0.0'</filter>
         <parameter key="port" value="${requisition:oc.port|9000}"/>
         <parameter key="paths"
value="/network-instances/network-instance[instance-name='master'],/protocols/protocol/bgp"
/>
       </package>
</connector>
```

- “service-name” is the name of the service to match
- “queue” is the name of the queue for dispatch to
- At least one package must be present for the connector to function
    - Multiple packages can be used to customize the parameters subsets of nodes
- The “filter” element is optional, if missing, all services with the given name will be considered
- Parameters are passed to the connector and are interpolated for node/interface & service level meta-data

Thoughts Minion Support

Ideally each location would have a single Minion initiate the connection and push out the results. If this Minion fails, then another Minion should take over the task. Tasks should be spread evenly across the pool of Minions, etc…

To simplify we could let every Minion at a location perform the connection, stream the data and de-duplicate if necessary when processing.

Even with the simplified case, the Minion needs to know:

A) What different connectors and packages are defined on the core
B) Which agents are associated with those packages
C) When the agents associated with those packages change
D) When the configuration changes
    a) Ideally we'd only be required to define configuration on the core and not repeat it on each Minion

Conclusion: We need a way of sharing configuration & state (i.e. inventory) with the Minion before we can make it handle connectors.