Introduction to Streaming Telemetry
IX Forum 9
Carlos Campana
ccampana@cisco.com
December 2015
What has changed?

New Capabilities

- Speed and scale
- SDN and centralized control
- Faster traffic engineering
- Gray failures
- Fault prediction
- Automated remediation

New Requirements

Network Monitoring is a Big Data Problem
Network Telemetry

Where Data Is Created

sensing & measurement

Where Data Is Useful

SNMP

syslog

CLI

storage & analysis

Scale Issues

Subject to Change

Unstructured
We need something different

Pooling vs Streaming
Streaming Telemetry
Design Vision

**Performance**
- Get as much data off the box as quickly as possible

**Coverage**
- Grant full access to all operational data on the box

**Automation**
- Serialize the data in a flexible, efficient way that fits customers automated tools
Streaming Telemetry: The First Iteration
Initial Goal: Validate the Big Data Proposition

• “As much data as fast as possible”
• Enable a push model
• Make data simple to use
• Options for serialization/transport
• Focus on statistics
• Periodic delivery (~10 seconds)
• Give full access to operational data
Ultra-high level picture

Instruction on:
- What data to collect
- With what cadence
- And send to where

```
Table 3

Table 2

Table 1

Receiving
unit

```

```
<table>
<thead>
<tr>
<th>Interface</th>
<th>ifInErrors</th>
<th>ifOutErrors</th>
<th>ifHCOutOctets</th>
</tr>
</thead>
<tbody>
<tr>
<td>HundredGigabitEthernet 0/1/0/2</td>
<td>10</td>
<td>0</td>
<td>123456789</td>
</tr>
<tr>
<td>Bundle-Ether 42</td>
<td>3</td>
<td>0</td>
<td>234567890</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

“I am the interface counters table”
Streaming Telemetry Model

- **Telemetry Policy**
  - Described in JSON.
  - Define one or multiple collection group(s).
  - Each group contains a rate and a pointer to one or multiple objects in the SysDB path.

- **Telemetry Configuration**
  - Define the encoder, transport and the receiver(s) for each policy.

- **Telemetry Agent**
  - XR process that runs automatically and looks for registered policies to act on.
The Policy Plane

```json
{
    "Name": "ptt1",
    "Metadata": {
        "Version": 25,
        "Description": "This is a sample policy to demonstrate the syntax",
        "Comment": "This is the first draft",
        "Identifier": "ptt1-id"
    },
    "CollectionGroups": {
        "FirstGroup": {
            "Period": 30,
            "Paths": [
                "RootOper.InfraStatistics.Interface([*]).Latest.GenericCounters"
            ]
        },
        "SecondGroup": {
            "Period": 60,
            "Paths": [
                "RootOper.MemorySummary.Node({'NodeName': '0/RP0/CPU0'}).Summary",
                "RootOper.BGP.Instance({'InstanceName': 'default'}).InstanceActive.DefaultVRF.Neighbor([*])"
            ]
        }
    }
}
```

$ scp ptt1.policy cisco@172.16.1.150:/telemetry/policies/ptt1.policy
Control Plane

RP/0/RP0/CPU0:r1#show run telemetry
Telemetry
c    encoder json
c    policy group ptt-policies-group1
c      policy ptt1
c        destination ipv4 172.16.1.1 port 5555
c        destination ipv6 fd01:23:6::1 port 5556
Data Plane: Encoder Output

```
"RootOper": {
  "InfraStatistics": {
    "GigabitEthernet0/0/0/0": {
      "Latest": {
        "GenericCounters": {
          "OutputQueueDrops": 0,
          "LastDiscontinuityTime": 1449091544,
          "InputIgnoredPackets": 0,
          "PacketsReceived": 40881,
          "OutputDrops": 0,
          "UnknownProtocolPacketsReceived": 0,
          "RuntPacketsReceived": 0,
          "CRCErrors": 0,
          "SecondsSinceLastClearCounters": 0,
          "CarrierTransitions": 0,
          "MulticastPacketsSent": 26784,
          "BytesSent": 43668872,
          "ThrottledPacketsReceived": 0,
          "Applique": 0,
          "FramingErrorsReceived": 0,
          "GiantPacketsReceived": 0,
          "OutputUnderruns": 0,
          "OutputErrors": 0,
          "BroadcastPacketsReceived": 0,
          "OutputBuffersSwappedOut": 0,
          "Resets": 0,
          "SecondsSincePacketSent": 0,
          "InputAborts": 0,
          "InputOverruns": 0,
          "InputQueueDrops": 0,
          "InputDrops": 0,
          "AvailabilityFlag": 0,
          "MulticastPacketsReceived": 26775,
          "SecondsSincePacketReceived": 0,
          "OutputBufferFailures": 0,
          "BytesReceived": 43635706,
          "ParityPacketsReceived": 0,
          "BroadcastPacketsSent": 2,
          "LastDataTime": 1449320856,
          "InputErrors": 0,
          "PacketsSent": 40904
        }...
```
Data Plane: Encoder Output

Identifier: Telemetry  Source: 172.16.128.2
Start Time: Sun Jan 25 00:24:17 1970
End Time: Mon Dec 7 09:03:55 2015

# Tables: 1
  Schema
  # Rows: 5

Row 4:
  applique: 0
  availability_flag: 0
  broadcast_packets_received: 2
  broadcast_packets_sent: 0
  bytes_received: 864025
  ... 
  input_errors: 0
  input_ignored_packets: 0
  input_overruns: 0
  input_queue_drops: 0
  interface_name: GigabitEthernet0/0/0/1
  last_data_time: 1449507828
  last_discontinuity_time: 1449503558
  multicast_packets_received: 521
  multicast_packets_sent: 1438
  output_buffer_failures: 0
  output_buffers_swapped_out: 0
  output_drops: 0
  output_errors: 0
  output_queue_drops: 0
  output_underruns: 0
  packets_received: 1918
  packets_sent: 1606
  parity_packets_received: 0
  resets: 0
  runt_packets_received: 0
  seconds_since_last_clear_counters: 0
  seconds_since_packet_received: 0
  seconds_since_packet_sent: 0
  throttled_packets_received: 0
  unknown_protocol_packets_received: 0

GPB over UDP
Consuming the Data
Some Consumption Models

BYO
Black Box

Custom

Open Source, Customizable

Output Codec
Logstash
ST Input Codec

Kafka
Hadoop
Impala
Kibana

Proprietary or OS-based

Commercial Stack
A batteries-included docker-based collection of demo stacks adapting network streaming telemetry to common consumer formats.

<table>
<thead>
<tr>
<th>Branch</th>
<th>New pull request</th>
<th>New file</th>
<th>Find file</th>
<th>HTTPS</th>
<th>Download ZIP</th>
</tr>
</thead>
</table>

- **ccassar**: Fix reordered ports in documentation of default ports in stacks

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
<th>Latest commit</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>common</td>
<td>Initial commit</td>
<td>541e14b</td>
<td>8 days</td>
</tr>
<tr>
<td>gems</td>
<td>Initial commit</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td>stack_ek</td>
<td>Initial commit</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td>stack_prometheus</td>
<td>Initial commit</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td>stack_signalfx</td>
<td>Initial commit</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td>LICENSE</td>
<td>Initial commit</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td>README.md</td>
<td>Fix reordered ports in documentation of default ports in stacks</td>
<td>8 days</td>
<td></td>
</tr>
</tbody>
</table>

**Streaming Telemetry Collector Stacks**

This repository is made available to support users wishing to experiment with consumption of streaming telemetry using off-the-shelf stacks.
Future Iterations
Data Plane: YANG-Driven Telemetry
Management Plane: Dynamic Policy Config

“Send me QoS stats every 15 seconds in OC YANG model”

Policy

Remote Management Station

QoS Stats

Stream

Stream

Stream
What Kinds of Data Are Interesting

Statistics
- Interface
- QoS
- LSP
- ACL stats
- Environmental
- ...

Operational State
- Interface Up/Down
- BGP Neighbor
- LSP Changes
- Topology
- ...

![Graph showing bit rate over time](image)