Model Driven Telemetry Introduction

Timely collection of network statistics is critical to ensuring that a network performs as expected and foreseeing and preventing any problems that could arise. Technologies such as SNMP, syslog, and the CLI have historically been used to gather this state information from the network. Using a pull model to gather the data, in which the request for network data originates from the client, does not scale and restricts automation efforts. With such a model, the network device sends data only when the client manually requests it. A push model continuously streams data from the network device to the client. Telemetry enables the push model, providing near instantaneous access to operational data.

There are two types of telemetry subscriptions:

- **Dynamic**: The subscriber sends a request, usually via subscription to YANG Notifications (rfc8639) or via gNMI. The device approves the request and begins streaming telemetry data.
- **Configured**: The subscription is configured via the CLI, NETCONF, or RESTCONF and is persistent between reboots. Configured subscriptions can be modified and terminated at any point and can be used to stream data to more than one receiver.

A subscription can specify how notifications are sent to the receivers:

- **Periodic notifications**: These notifications are sent with a fixed rate defined in the telemetry subscription. This data is ideal for device counters or measures such as CPU utilization or interface statistics because of their dynamic, always-changing nature
- **On-change notifications**: These notifications are sent only when the data changes. These types of notifications might be sent, for example, for faults, new neighbors being detected, and thresholds being crossed.

Streamed information is modeled using a **data modeling language like YANG** which describes data hierarchy using a nodes tree structure and specifies nodes constraints and data types. YANG models are defined by IETF, Openconfig and vendors.

Modeled data is **serialized using data format like JSON, XML or Protobuf**. When both JSON and XML are human readable and can be parsed without knowing the schema in advance, Protobuf is very much optimized for telemetry as it is very dense and allow fast data processing. However, it is not human readable and requires a message definition described in .proto files shared between the encoder and decoder.

When serialized, data are **transported using a protocol like NETCONF or gRPC**. gRPC is a Cloud Native Computing Foundation project and uses HTTP/2. It is particularly well suited for streaming telemetry.

When it comes to Telemetry consumption, one might think about this high-level architecture:

- **Collection layer**: the first stop for the data streamed out of the router. The main goal is to collect all the streams and transform them from GPB/JSON into a format that will be supported by the layer above. Filtering might also be configured here.
- **Storage layer**: usually a TSDB (time-series database). The goal is to take the data from the “Collection layer” and store it together with timestamps. You can also have other types of databases here (think of telemetry as a big data solution).

- **Application layer**: this is where you apply your business logic tools for the data stored in the storage layer.