



# **Cisco Telemetry Concepts: Overview & Design**





# Keith Bogart

CCIE #4923

---

✉ [kbogart@ine.com](mailto:kbogart@ine.com)  
🐦 [@keithbogart1](https://twitter.com/keithbogart1)  
in [linkedin.com/in/keith-bogart-2a75042](https://www.linkedin.com/in/keith-bogart-2a75042)



CCIE Routing & Switching



## **Course Topic Overview**

- + In this course you will learn about the core concepts of telemetry, its fundamental elements, and things to consider before implementing telemetry into your network.

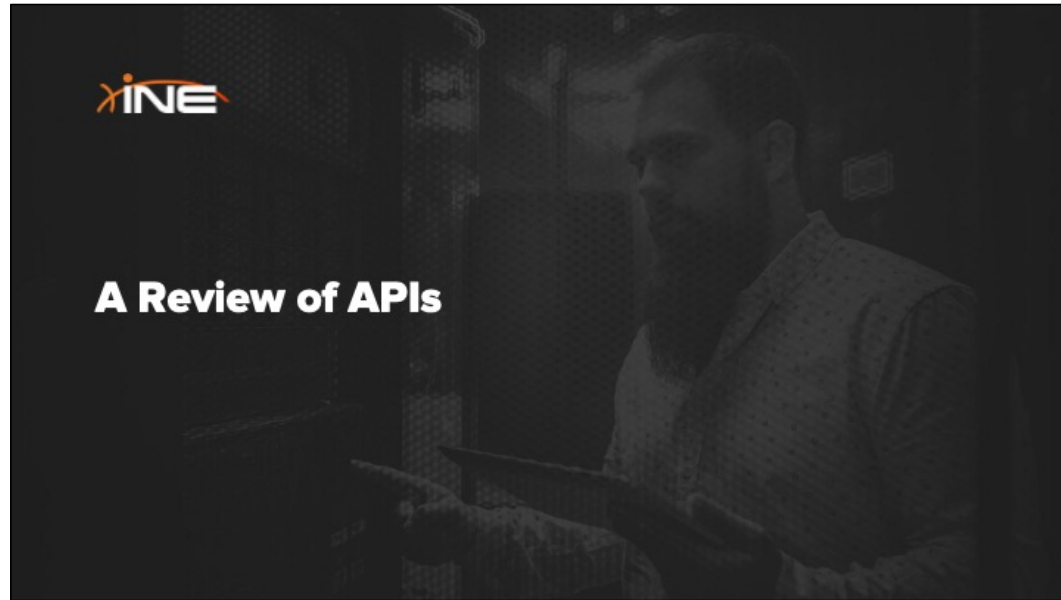


## **Learning Objectives:**

- Explain what an API is
- Summarize the differences between data encoding methods used with telemetry
- Compare and contrast NETCONF, RESTCONF and gRPC
- Explain how YANG is utilized for data modeling
- Explain the problems solved by model-driven telemetry
- Understand the core concepts involved with telemetry such as subscriptions, publications and collectors
- Summarize the problems solved by implementing telemetry brokers
- Identify seven (7) considerations when designing a telemetry solution



## **A Review of APIs**



### What Is An API?

---

- + Application Programming Interface
- + A piece of code to allow different applications to talk to each other.
- + Two generic types of APIs
  - + Those that allow internal applications in your local system to exchange data.
  - + Those that use IP networking to exchange data between remote applications.



## API Usage In Automation

---

- + APIs in automation and telemetry are the “bridge” that allow the transfer of configuration and state data between server applications and infrastructure devices.
- + APIs use a Client/Server model
- + API client typically initiates a session with an API server
  - + Examples:
    - + Server Application (API Client) communicates with a telemetry source (router or firewall) to subscribe to telemetry data
    - + SDN Controller (API Client) communicates with Switch/Router (API Server) to preconfigure a network path with the required QoS parameters
    - + Router (API Client) initiates a dial-out connection to a Telemetry destination



## API Differences

---

- + APIs specify the parameters to be used for applications to communicate such as;
  - + What type of network protocol can be used to initiate communications between endpoints (NETCONF? RESTCONF? gNMI?)
  - + How is data to be structured (JSON? XML?)
  - + What model should be used to identify data? (YANG?)
  - + What transactions are allowed (get-config? edit-config? subscribe?)
- + Common API Protocols
  - + SOAP (Simple Object Access Protocol)
  - + NETCONF
  - + Google RPC (gRPC)
  - + REST







## **A Review of Data Encoding Formats: XML**

### What Is Data Encoding

---

- + APIs allow the transfer, modification, and deletion of data between different systems
- + Data encoding is a method to describe the data in a structured way so that it can be serialized.
- + With network systems and applications, the most common types of data encoding are;
  - + XML
  - + JSON
  - + Google Protocol Buffers (Protobuf)



## XML Overview

- + XML = eXtensible Markup Language
- + Similar in look-and-feel to HTML
  - + Data is wrapped in beginning and ending tags
    - + `<font>` = opening tag
    - + `</font>` = closing tag
  - + Human-readable

```
CSR1-Ticket39#show running-config | format
<?xml version="1.0" encoding="UTF-8"?><Device-Configuration xmlns="urn:cisco:xml-pi">
<version><Param>17.3</Param></version>
<service><timestamps><debug><datetime><msec/></datetime></debug></timestamps></service>
<service><timestamps><log><datetime><msec/></datetime></log></timestamps></service>
[Output Omitted]
<interface><Param>Loopback1234</Param>
<ConfigIf-Configuration>
<ip><address><IPAddress>1.2.3.11</IPAddress><IPSubnetMask>255.255.255.255</IPSubnetMask></address></ip>
</ConfigIf-Configuration>
</interface>
```

Opening tag → `<interface>`

Closing tag → `</interface>`



- "show running-config | format" works with both IOSv and IOS-XE routers.

## XML End Tag Options

- + XML provides for two ways of encoding **end tags**:
  - + Typically end tags start with an "angle bracket" followed immediately by a forward-slash

```
CSR1-Ticket39#show running-config | format
<?xml version="1.0" encoding="UTF-8"?><Device-Configuration xmlns="urn:cisco:xml-pi">
<version><Param>17.3</Param></version>
<service><timestamps><debug><datetime><msec/></datetime></debug></timestamps></service>
<service><timestamps><log><datetime><msec/></datetime></log></timestamps></service>
```

- + Another acceptable type of tag combines both a start and end tag into a single tag:

```
CSR1-Ticket39#show running-config | format
<?xml version="1.0" encoding="UTF-8"?><Device-Configuration xmlns="urn:cisco:xml-pi">
<version><Param>17.3</Param></version>
<service><timestamps><debug><datetime><msec/></datetime></debug></timestamps></service>
<service><timestamps><log><datetime><msec/></datetime></log></timestamps></service>
```



## XML Attributes

- + XML tags may provide additional descriptive data within the tag by adding XML "attributes".
  - + Attributes convey meta-level information about the message
  - + Attributes are added after the start tag but are still contained within the broken brackets
  - + Attributes contain a name, equals-sign, and a value
  - + A single tag may contain more than one attribute

```
CSR1-Ticket39#show running-config | format
<?xml version="1.0" encoding="UTF-8"?><Device-Configuration xmlns="urn:cisco:xml-pi">
<version><Param>1.3</Param></version>
<service><timestamps><debug><datetime><msec/></datetime></debug></timestamps></service>
<service><timestamps><log><datetime><msec/></datetime></log></timestamps></service>
```

XML Attributes



## XML Namespaces

- + Every tag and attribute in an XML document belongs to a namespace
- + The namespace provides the formal definition/meaning behind the tags and attributes.
- + An XML document must reference at least one namespace at the beginning of the document.
- + The namespace is identified within a special attribute, "xmlns"

```
CSR1-Ticket39#show running-config | format
<?xml version="1.0" encoding="UTF-8"?><Device-Configuration xmlns="urn:cisco:xml-pi">
<version><Param>17.3</Param></version>
<service><timestamps><debug><datetime><msec/></datetime></debug></timestamps></service>
<service><timestamps><log><datetime><msec/></datetime></log></timestamps></service>
```

XML Namespace Attribute

- + Multiple namespaces may be referenced within a single file





## **A Review of Data Encoding Formats: JSON**

## JSON Review

---

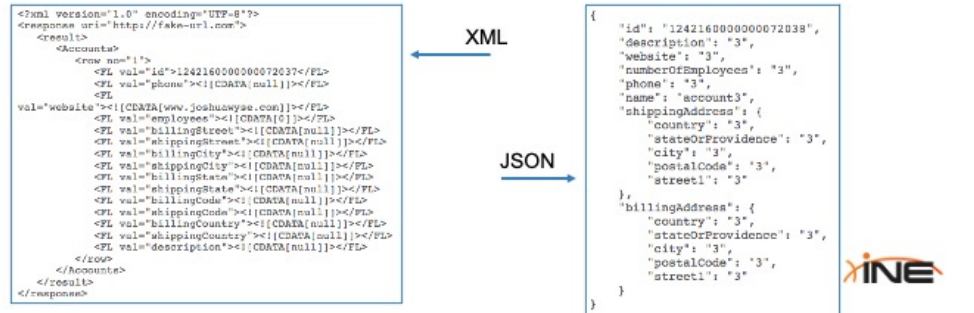
- + Pronounced “Jay Sahn”
- + JavaScript Object Notation
- + A subset of JavaScript syntax
- + JSON “Objects” used for representing data that is transferred between server and client.
  - + Simply put, it is a method of generically describing data.
- + Used extensively by web-service APIs (such as REST APIs)





## Benefits of JSON

- + It is light-weight
- + It is language independent
- + Easy to read and write
- + Text based, human readable data exchange format



## JSON Syntax Rules

- + Data/properties is in name/value (key-value) pairs separated by colons
- + Multiple name-value pairs within a single object are separated by commas
- + Curly braces hold objects { }
- + Square brackets hold arrays [ ]
- + Spaces and line breaks don't matter.

```
{"instructor": [{"name": "Keith", "employer": "INE", "salary": 70}]}
```

This...

Or this..

```
{  
  "instructor": {  
    {  
      "name": "Keith",  
      "employer": "INE",  
      "salary": 70  
    }  
  }  
}
```



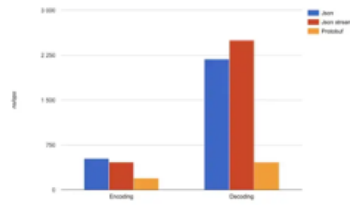
- "Instructor" is the object here
- "name", "employer" and "salary" are all "properties" of the object.



## **A Review of Data Encoding Formats: Google Protocol Buffers**

## Google Protocol Buffers (Protobuf)

- + A data encoding method designed to quickly and efficiently serialize data (same goals as XML and JSON)
- + Released by Google to the open source community in 2008.
- + Serialized data is compiled bytes, which is not human-readable
- + Encoding and decoding performance significantly better than JSON



Comparing encoding/decoding performance. Less than better.

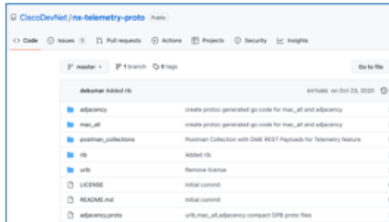
Graphic courtesy of <https://mnwa.medium.com/what-the-hell-is-protobuf-4aff084c5db4>



- More good information about Protobufs can be found here:  
<https://www.adaptiv.nz/protobuf-what-is-it-why-you-should-care-and-when-should-you-use-it/>


## Google Protocol Buffers (Protobuf)

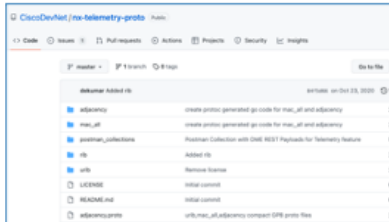

- + In order to receive and decode the data stream correctly, a telemetry receiver requires a .proto file
  - + Describes the encoding and the transport services.
  - + Allows for encoding and decoding of a binary stream into a key value string pair
- + Examples of Cisco .proto files can be found at:

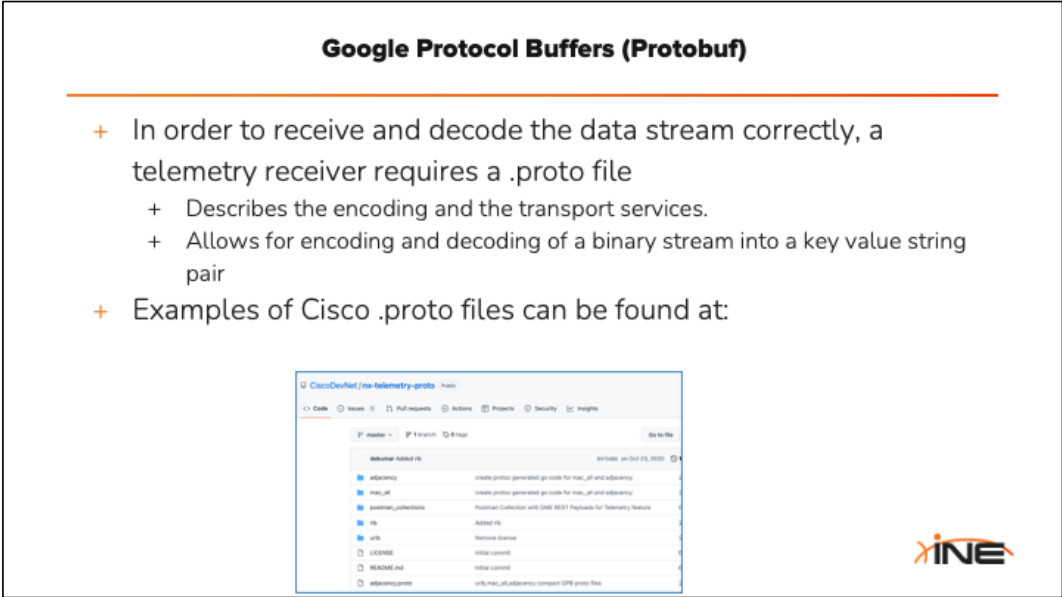
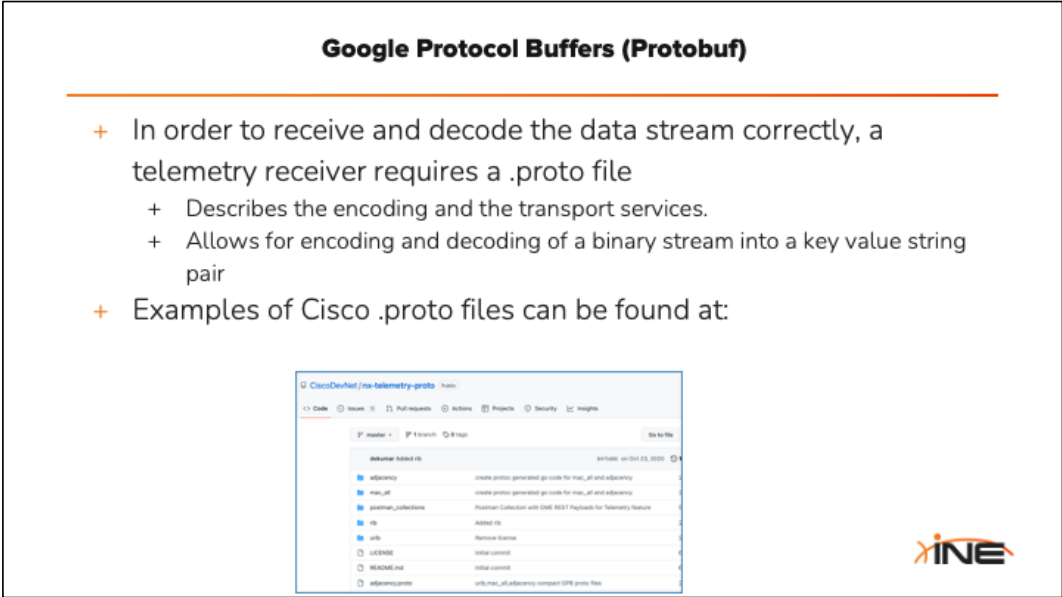


The screenshot shows the Cisco DevNet repository for 'no-telemetry-protobuf'. The file list includes:

File Name	Description
telemetry	create protocol generated go code for max_all and adjacency
max_all	create protocol generated go code for max_all and adjacency
postman_collections	Postman Collection with ONE REST Payloads for Telemetry feature
rb	Added rb
rb	Remove license
LICENSE	Initial commit
README.md	Initial commit
telemetry.proto	rb, max_all, adjacency compact ONE proto file



- ## Google Protocol Buffers (Protobuf)
- + In order to receive and decode the data stream correctly, a telemetry receiver requires a .proto file
    - + Describes the encoding and the transport services.
    - + Allows for encoding and decoding of a binary stream into a key value string pair
  - + Examples of Cisco .proto files can be found at:
- 
- The screenshot shows the Cisco DevNet repository for 'no-telemetry-protobuf'. The file list includes:
- | File Name           | Description   |
|---------------------|---|
| telemetry           | create protocol generated go code for max_all and adjacency     |
| max_all             | create protocol generated go code for max_all and adjacency     |
| postman_collections | Postman Collection with ONE REST Payloads for Telemetry feature |
| rb                  | Added rb  |
| rb                  | Remove license  |
| LICENSE             | Initial commit  |
| README.md           | Initial commit  |
| telemetry.proto     | rb, max_all, adjacency compact ONE proto file                   |
- 



## Google Protocol Buffer (GPB) Sample .proto File

---

Output has been  
truncated...

```
syntax = "proto3";

option go_package = "adjacency";

option cc_enable_arenas = true;

/* Adjacency event type
*/
enum AdjacencyEventType {
    ADJACENCY_EVENT_TYPE_NO_EVENT = 0;
    ADJACENCY_EVENT_TYPE_ADD = 1;
    ADJACENCY_EVENT_TYPE_DELETE = 2;
    ADJACENCY_EVENT_TYPE_UPDATE = 3;
    ADJACENCY_EVENT_TYPE_DOWNLOAD = 4;
    ADJACENCY_EVENT_TYPE_DOWNLOAD_DONE = 5;
}

/* Adjacency address family
*/
enum AdjacencyAddressFamily {
    ADJACENCY_AF_IPV4 = 0;
    ADJACENCY_AF_IPV6 = 1;
}
```





## **Comparing Network Automation APIs**

## APIs For Automation

---

### + RESTCONF

- + Representational State Transfer Configuration Protocol
- + Follows the conventions for REST APIs
- + Uses HTTP methods such as "get", "post", "put", and "delete"
- + Can utilize either XML or JSON for data representation/syntax
- + Can be used to preconfigure a device for telemetry

### + NETCONF

- + Network Configuration Protocol
- + Uses SSH for transport
- + Uses XML for data encoding/syntax
- + Stateful and session-oriented

### + gNMI (Google RPC Network Management Interface)

- + Open-source API framework
- + Uses gRPC (HTTP/2) for transport with TLS
- + Can use JSON for data representation/syntax or Google's "Protocol Buffer" method



- HTTP/2 leaves all of HTTP/1.1's high-level semantics, such as [methods](#), [status codes](#), [header fields](#), and [URLs](#), the same. What is new is how the data is framed and transported between the client and the server.



## Comparing gNMI & NETCONF

	gNMI	NETCONF
Serialization	<b>Protobuf</b> Compact binary format	<b>XML</b>
Transport	<b>gRPC (HTTP/2.0)</b>	<b>SSH</b>
Diff oriented	<b>Yes</b> Returns only elements of the tree that have changed from last read	<b>No</b> Always returns entire sub-tree snapshot
Native support for streaming telemetry	<b>Yes</b> gRPC natively supports bidirectional streaming	<b>No</b> Needs YANG Push extension

Table courtesy of: <https://opennetworking.org/wp-content/uploads/2019/10/NG-SDN-Tutorial-Session-2.pdf>



- HTTP/2 leaves all of HTTP/1.1's high-level semantics, such as [methods](#), [status codes](#), [header fields](#), and [URLs](#), the same. What is new is how the data is framed and transported between the client and the server.

## NETCONF & RESTCONF Similarities

---

- + Both NETCONF and RESTCONF;
  - + Are Client/Server based
    - + Client = Server/controller
    - + Server = Network device
  - + Use TCP for packet transport
  - + Can use XML for data encoding
  - + Rely on YANG;
    - + *ietf-yang-library* used to discover server capabilities and features
    - + YANG models utilized to define data
  - + Both can be programmed using Python



## NETCONF & RESTCONF Differences

### NETCONF

SSH for transport

XML **only** for data encoding

Full feature set and support for network-wide **transactions** (ie. "stateful").

Uses RPCs such as <get-config> and <edit-config>

**Defines** and utilizes the concepts of "running", "candidate" and "startup" datastores.

Consumes **more overhead than RESTCONF** due to additional handshaking and payload bytes.

### RESTCONF

HTTP for transport (no built-in security)

XML **or** JSON for data encoding

A simplified version of NETCONF with less features. **Stateless** so no concept of "transactions". Uses HTTP verbs.

Uses **HTTP verbs** ("GET", "Post", "Put" and "Delete")

Utilizes datastores **defined in** NETCONF, specifically the "running" datastore.

Consumes less bandwidth than NETCONF.



- In this context, a "transaction" is the ability to start a TCP session with a device and maintain that session with back-and-forth communications occurring the entire time. So a "transaction" allows an Orchestrator to push some configuration information into a Candidate datastore, validate that the push was successful and that the commands and syntax are understood by the target device, and then commit that action once verification has been achieved and terminate the TCP connection.
- A "network-wide" transaction is the idea of initiating multiple, simultaneous transactions to different devices in order to achieve a common goal, like implement some concept (like a new leg for a L3VPN). With a network-wide transaction, configurations are not pushed from the Candidate to the Running datastores until all devices have reported back that the changes are supported without error.
- RESTCONF doesn't support the idea of transactions because even though it is TCP-based (uses HTTPS) it is stateless so it's only a one-two exchange...it wasn't designed to maintain a lengthy connection or exchange multiple back-and-forth messages like NETCONF.

## Response Codes

- + Any network API that utilizes HTTP (RESTCONF, gNMI, etc) will understand HTTP response codes
- + Before using these methods, ensure you also understand these codes so you can interpret logging and error messages on your apps

Response Code	Description
<i>Success Messages (2xxx)</i>	
200	Request succeeded
201	The request has been fulfilled; new resource created
204	The server fulfilled the request but does not return a body
<i>Client Errors (4xx)</i>	
400	Bad request; malformed syntax
401	Unauthorized
403	Server understood request but refuses to fulfill it
<i>Server Errors (5xx)</i>	
500	Internal server error
501	Not implemented

Table courtesy of: CCNP Enterprise Design (ENSLD 300-420) Official Certification Guide





## **A Review of Data Modeling with YANG**

## **What Are Data Models?**

- + An intuitive, standard way to describe something (data)
- + Describes all the characteristics/attributes of data
  - + What type of data is it?
    - + Interface
    - + Route Table
  - + What characteristics does the data have?
    - + A descriptive name?
    - + Numerical values that are descriptive of the data (interface numbers, subnet masks, etc)
  - + How is the data to be represented
    - + Booleans?
    - + Strings?
    - + Integers?

## Example Data Model

```
module example-sports {  
  namespace "http://example.com/example-sports";  
  prefix sports;  
  
  import ietf-yang-types { prefix yang; }  
  
  typedef season {  
    type string;  
    description  
      "The name of a sports season, including the type and the year, e.g.  
      'Champions League 2014/2015'.";  
  }  
  
  container sports {  
    config true;  
  
    list person {  
      key name;  
      leaf name { type string; }  
      leaf birthday { type yang:date-and-time; mandatory true; }  
    }  
  
    list team {  
      key name;  
      leaf name { type string; }  
      list player {  
        key "name season";  
        unique number;  
        leaf name { type leafref { path "/sports/person/name"; } }  
        leaf season { type season; }  
        leaf number { type uint16; mandatory true; }  
        leaf scores { type uint16; default 0; }  
      }  
    }  
  }  
}
```

- This image is courtesy of <https://en.wikipedia.org/wiki/YANG>

## A Review of YANG

- + YANG = Yet Another Next Generation
- + IETF standard defined in RFC 6020 and RFC 7950
- + A language for building and defining data models
- + Can be used to build data models for ANY kind of data
- + YANG models that describe things are called “Modules”
- + There isn't just one YANG module, but several YANG modules for different kinds of data, and new ones developed all the time by the IETF, private companies, etc.
  - + YANG modules to describe interfaces
  - + YANG modules to describe Access-Lists
  - + YANG modules to describe Routing tables

One of the earliest data modeling languages was known as Abstract Syntax Notation One (ASN.1). It was from this that the “Structure of Management Information” (SMI) was derived as another data modeling language which was used with SNMP. The structure of the SNMP MIB (Management Information Base) is based on SMI data modeling.

SMI was then upgraded to SMIv2 in 1999

Just prior to SMIv2 becoming standardized another effort was underway to upgrade it...YET AGAIN...and that effort was called SMIng (SMI Next Generation).

SMIng did not succeed in the IETF. However, later on the NETCONF (NETwork CONfiguration) Protocol was developed as a means of securely connecting with network devices for the purposes of managing and configuring them. “Yet Another Generation” of a Data Modeling Language was needed to go with NETCONF, so the principles behind SMIng were borrowed to create YANG.



## YANG Module Example

```
module ietf-interfaces {  
  import ietf-yang-types {  
    prefix yang;  
  }  
  container interfaces {  
    list interface {  
      key "name";  
      leaf name {  
        type string;  
      }  
      leaf enabled {  
        type boolean;  
        default "true";  
      }  
    }  
  }  
}
```

## Where Do YANG Models Come From?

- + YANG models come from a variety of sources
  - + Private companies such as Cisco and Juniper
    - + These are called “Native Models”
  - + Standards bodies such as the IETF
  - + Consortia of private companies, network operators and private individuals that are working together such as the OpenConfig Organization
- + Networking devices typically have built-in YANG module support depending on their software version
- + YANG module support included in Cisco IOS-XR, IOS-XE, and NX-OS platforms.

Models submitted from private companies are also sometimes called, “Vendor Models”.

### Comparing IETF, OpenConfig & Native YANG Modules

IETF	OpenConfig	Native Models
Standards Body	Standards Body	Vendor-Produced
Vendor-Neutral	Vendor-Neutral	Vendor-Specific
International group of network designers, operators, vendors, and researchers	Composed primarily of large network operators (Service Providers and Web Companies)	Developed by software developers and designers working for a specific company such as Cisco or Juniper



## **Introducing Telemetry**



## The Problems Defined

---

### + Traffic Optimization

- + How to quickly identify network problems (link utilization, packet drops, link failures) and apply actions?
- + Previously, SNMP polling (or SNMP Informs/Traps) or Python-scripted CLI polling was used for this
- + SNMP/Python scripts didn't provide fast enough response times
- + SNMP MIBs might not cover all required data

### + Preventative Troubleshooting

- + Identify network problems before they occur in order to take corrective action
- + Once again, SNMP was the common protocol used for this with the same limitations as above

### + Telemetry solves these problems



## What Is Telemetry?

- + Telemetry: An “automated communication process by which measurements are collected and transmitted to receivers for monitoring of the data” – *CCNP Enterprise Design Official Certification Guide*
- + Telemetry comes in many forms:
  - + NetFlow, IPFIX, NETCONF, syslog, and SNMP data sourced from routers, switches, and firewalls
  - + Performance records, uptime records, and usage data from servers and applications
  - + Logs from public cloud providers like AWS and Azure



The security benefits of telemetry.



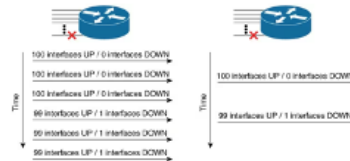
Graphic courtesy of: Cisco Telemetry Architecture Guide

- Legacy methods of collecting data from devices (such as SNMP and CLI scripts) utilized a “Pull” model. This wasn’t scalable
- Telemetry by definition uses a “push” model in which the telemetry source (i.e. router, switch, virtual appliance, etc) will “push” data to a telemetry receiver (eg server).

## Telemetry Frequency & Methods

### + Telemetry Frequency:

- + Cadence-based telemetry
- + Event-based telemetry



Graphic courtesy of Telemetry Configuration Guide for Cisco NCS 5500 Series Routers, IOS XR Release 7.1.x

### + Telemetry Methods

- + Policy-based telemetry
- + Model-driven telemetry (MDT)



- Cadence-based telemetry strictly refers to telemetry data being sent at regular intervals...sometimes as quickly as to be perceived as real-time. It does NOT refer to WHAT type of data is selected for streaming.
- Cadence-based telemetry paired with Policy-based telemetry means you are not only configuring the cadence (frequency) of the telemetry publication but also the specific data that is to be published, which has been defined in some kind of policy.

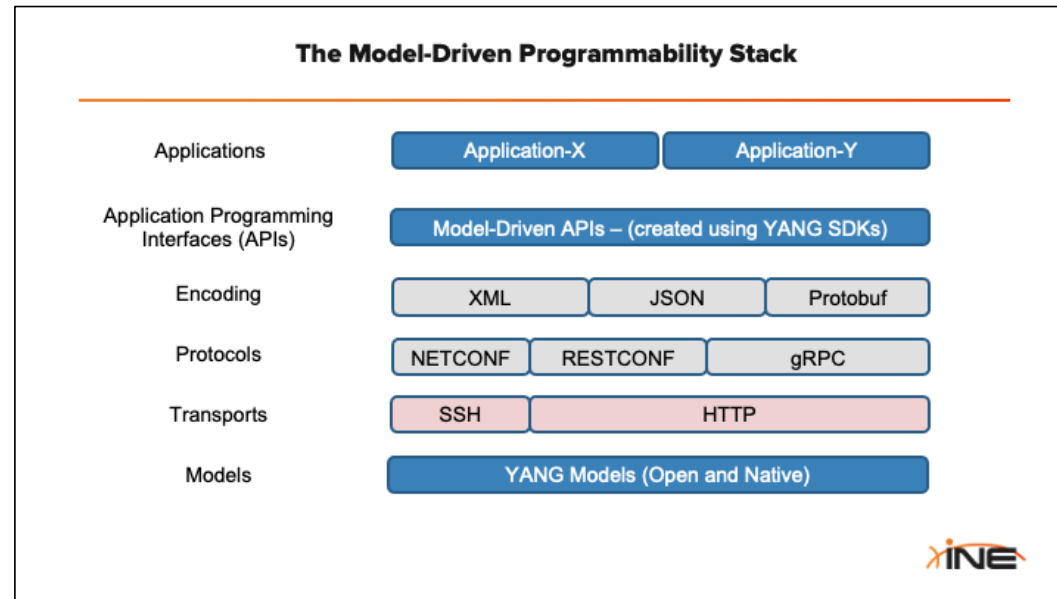
### What Is **Model-Driven** Telemetry?

---

- + “Model-Driven” telemetry pairs the concept of telemetry with standards-based (or native) YANG models for the representation of data.
- + Applications within a collector (ie. server running telemetry applications) allow one to subscribe to desired data hosted on the telemetry source (that will be streamed via telemetry) by selecting from relevant YANG modules.







- gRPC is “Google Remote Procedure Call”. An alternative to NETCONF or RESTCONF.
- gRPC was designed to encode data using their own method called “Protocol Buffers” rather than XML or JSON. But gRPC CAN work with JSON.
- gRPC is inherently more secure than RESTCONF because it incorporates TLS along with HTTP.
- REST APIs generally use JSON or XML message formats, while gRPC uses protocol buffers.
- To signal errors, REST APIs use HTTP status codes, while gRPC uses error codes.
- gRPC’s message sizes tend to be dramatically smaller than those of REST APIs.

### Comparing SNMP To Model-Driven Telemetry

SNMP	Model-Driven Telemetry
Utilizes UDP for transport	Utilizes TCP (NETCONF or gRPC with gNMI) for transport
Unreliable packet delivery (UDP)	Reliable packet delivery (TCP)
Supports both a Pull-Model (polling) and Push-Model (Traps/Informs)	Push-Model: Data continuously streamed from networking device to Telemetry collector or pushed upon event triggers.
Data modeling: MIBs and OIDs	Data modeling: YANG models
Delays induced due to SNMP polling intervals	Enables near real-time access to data monitoring



- NETCONF or gRPC is used to exchange the RPCs that allow one to subscribe to requested YANG models on the network device.
- Some platforms (such as Nexus-OS) support sending telemetry using UDP and secure UDP (DTLS)



## **Model-Driven Telemetry Core Concepts**

## Telemetry Concepts & Terminology

---

### + Subscriptions

- + Used to define the **set of data that is requested** as part of the telemetry data; **when the data is required**, how the data is to be **formatted**, and (when not implicit), who (**which receivers**) should receive the data.

### + Subscriber

- + The MDT Collector/Receiver which receives telemetry data from the networking device
- + Typically, this is an application on a server

### + Subscription Service

- + The telemetry service running within a networking device

### + Publications

- + Sources of telemetry data (i.e., networking devices) "publish" that data back to the subscribing collectors/receivers.



- A Telemetry subscription is defined as a combination of the specific YANG group or leaf requested as well as its cadence (or on-demand if that is appropriate).
- For each new pair of cadence and YANG model item...this is considered as a new subscription.
- Maximum number of subscriptions is platform-dependent: IOS-XE currently supports up to 100-subscriptions per networking device.

### **YANG Subscription Types**

---

- + Network devices can publish YANG data to collectors using different methods
- + Periodic publications
  - + Also called, "Frequency-based telemetry"
- + On-Change publications
  - + Also called, "Event-based telemetry"



### MDT Publication Frequency

---

- + Periodic publications:
  - + Telemetry data (formatted and described per relevant YANG models) is transmitted to one-or-more Collectors at specified intervals
  - + Data transmission can be defined in centiseconds (1/100<sup>th</sup> of a second) to enable near real-time statistics gathering
  - + Also called, "Cadence-based telemetry"
- + On-Change publications:
  - + Data is only streamed when a change to that data occurs
  - + Useful for data that changes infrequently (IGP neighbor peering, CPU thresholds, etc)
  - + The data to be modeled must be specified in a YANG module



- With periodic publications, although you can set the cadence using increments of centiseconds you CANNOT set this to LESS than one second. The minimum configurable value is "100" (100 centiseconds = 1 second).
- The Smallest cadence is platform dependent (IOS-XE = 1-second, NX-OS = 5-seconds)

## Dial-In & Dial-Out Approaches

- + Dial-In Approach:
  - + The collector initiates the TCP session (i.e. "dials into") to the networking device and subscribes to it, after which data is published back to that collector.
- + Dial-Out Approach:
  - + The networking device initiates the TCP session to the Collector and "dials out" to it. Once the subscription is established, data is published.

Dial-In (Dynamic)	Dial-out (Static or Configured)
Telemetry updates are sent to the initiator/subscriber.	Telemetry updates are sent to the specified receiver/collector.
Life of the subscription is tied to the connection (session) that created it, and over which telemetry updates are sent. No change in the running configuration is observed.	Subscription is created as part of the running configuration; it remains the device configuration till the configuration is removed.
Dial-in subscriptions need to be re-initiated after a reload, because established connections or sessions are killed during stateful switchover.	Dial-out subscriptions are created as part of the device configuration, and they automatically reconnect to the receiver after a stateful switchover.
Subscription ID is dynamically generated upon successful establishment of a subscription.	Subscription ID is fixed and configured on the device as part of the configuration.

Table courtesy of: Programmability Configuration Guide, Cisco IOS XE Gibraltar 16.10.x



## Dial-Out Telemetry Configuration Example

```

CSR-XE#show run | begin telemetry
telemetry ietf subscription 501
encoding encode-kvgpb
filter xpath /process-cpu-ios-xe-oper:cpu-usage/cpu-utilization/five-seconds
source-address 192.168.122.2
stream yang-push
update-policy periodic 500
receiver ip address 192.168.122.1 57000 protocol grpc-tcp
netconf-yang
end
  
```

Table 3. Supported Combination of Protocols

Transport Protocol	NETCONF		gRPC		gNMI	
	Dial-In	Dial-Out	Dial-In	Dial-Out	Dial-In	Dial-Out
Stream						
yang-push	Yes	No	No	Yes	Yes	No
yang-notif-native	Yes	No	No	No	No	No
Encodings	XML	No	No	Key-value Google Protocol Buffers (xvgpb)	JSON_IETF	No

Table courtesy of: Programmability Configuration Guide, Cisco IOS XE Gibraltar 16.10.x



- In a dial-out configuration, the subscription number is manually configured. In a dial-in session this would be dynamically derived. The number of “501” has no special significance.
- The encoding method of “KVGPB” (Key-Value Google Protocol Buffers) specifies the use of Google Protocol Buffers. This is required because the “protocol grpc-tcp” has been selected in the “receiver IP address” line.
- The update-policy is set in “centiseconds” or 1/100<sup>th</sup> of a second. In this example, telemetry info matching the YANG module selected in “filter xpath” statement will be sent from the router every 5-seconds.



## Software Version Differences

Table 3. Supported Combination of Protocols

Transport Protocol	NETCONF		gRPC		gNMI	
	Dial-In	Dial-Out	Dial-In	Dial-Out	Dial-In	Dial-Out
<b>Stream</b>						
yang-push	Yes	No	No	Yes	Yes	No
yang-notif-native	Yes	No	No	No	No	No
<b>Encodings</b>						
XML		No	No	Key-value Google Protocol Buffers (kvGPB)	JSON_IETF	No

Table courtesy of: Programmability Configuration Guide, Cisco IOS XE Gibraltar 16.10.x

Table 3. Supported Combination of Protocols

Transport Protocol	NETCONF		gRPC		gNMI	
	Dial-In	Dial-Out	Dial-In	Dial-Out	Dial-In	Dial-Out
<b>Stream</b>						
yang-push	Yes	No	No	Yes	Yes	No
yang-notif-native	Yes	No	No	Yes	No	No
<b>Encodings</b>						
XML		No	No	Key-value Google Protocol Buffers (kvGPB)	JSON_IETF	No

Table courtesy of: Programmability Configuration Guide, Cisco IOS XE Amsterdam 17.3.x

- Telemetry is a rather new and evolving feature. Here we can see that between two different software releases there was a change between the stream-types that were supported with gRPC and dial-out Telemetry.

## Sensor Paths

---

- + Both dial-in and dial-out MDT subscriptions utilize a component called, "Sensor Paths".
  - + This is also referenced as "XPath Filters"
- + A sensor path defines the particular elements of a YANG model (such as particular Leaf nodes or Leaf-List nodes) to stream via MDT.
- + Each MDT subscription contains a single sensor path as well as its related cadence (on-demand or periodic).



## Examples of Sensor Paths

- + Dial-in telemetry: Sensor paths are requested by the Collector/Receiver with RPCs
  - + Both NETCONF and OpenConfig utilize a "Subscribe" RPC however their formats differ:

### NETCONF "Subscribe" RPC (formatted in XML)

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <create-subscription xmlns="urn:ietf:params:xml:ns:netconf:notification:1.0">
    <stream>NETCONF</stream>
  </create-subscription>
</rpc>
```

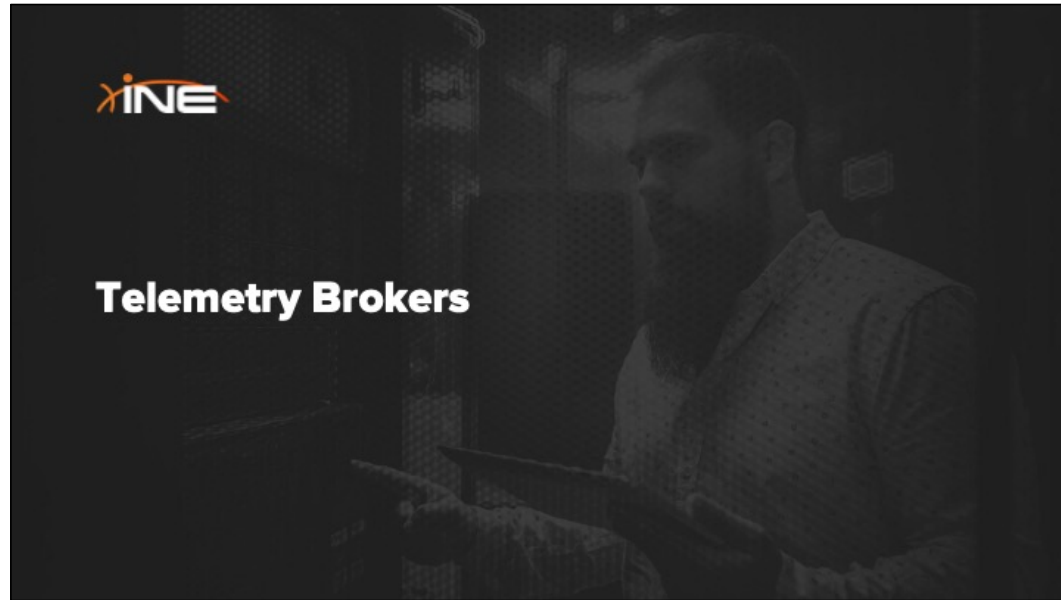
### OpenConfig's gNMI "Subscribe" RPC (defined in .proto files)

```
service gNMI {
  rpc Capabilities(CapabilityRequest) returns (CapabilityResponse);
  rpc Get(GetRequest) returns (GetResponse);
  rpc Set(SetRequest) returns (SetResponse);
  rpc Subscribe(stream SubscribeRequest) returns (stream SubscribeResponse);
}
```



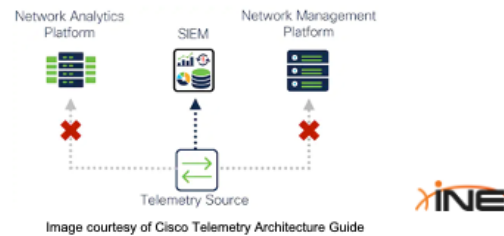


## **Telemetry Brokers**



### Telemetry Scalability Concerns (Single Destination)

- + In large Enterprise, Campus, and Service Provider networks several departments may compete for the same telemetry data
  - + The Networking Team may require telemetry data from all devices in Subnets-A through D
  - + The Security Team may require telemetry data from all devices in Subnets-C through G.
- + Many telemetry sources can only send data to a single destination collector which can lead to;
  - + Reduced visibility
  - + Silos between departments



- FYI...there is no standard pronunciation for "S.I.E.M". Some pronounce it as "SIM" (like "simple") and others pronounce it like "Seem"

### Telemetry Scalability Concerns (Multiple Destination)

---

- + Telemetry sources which can send data to a multiple destination collectors may also induce problems such as;
  - + Reduced bandwidth on all links between sources and receivers
  - + Increased compute resources for each source to replicate each subscription N-times.



## Introducing Telemetry Brokers

- + Telemetry Sources can send their data to a telemetry broker instead of directly to the Receiver
- + Telemetry Broker replicates and routes telemetry to all telemetry destinations that can use that data.
- + Access to telemetry data is no longer limited to how many destinations the telemetry source can export to, removing limitations that would hinder scalability and visibility within a network.



Image courtesy of Cisco Telemetry Architecture Guide



## Capabilities Of Telemetry Brokers

- + In addition to allowing for predictable network paths, telemetry brokers can also:
  - + Filter telemetry data: Filters data that is being replicated to consumers. Provides fine-grain control over what they see and analyze.
  - + Transform data: Transforms data protocols from the exporter to the consumer's protocol of choice.
- + The Cisco Telemetry Broker is a virtual appliance installed within your own hardware.

Distributed	
Management Server	Streaming Node
CPU: 4 CPUs	1 CPU/1-2 CPUs 10 GB/1-5 CPUs Transformer Capacity: 8 GB
Memory: 8 GB	1 GB/1-4 GB 10 GB/1-8 GB Transformer Capacity: 12 GB
Storage: 48 GB	70 GB

Image courtesy of Cisco Telemetry Broker Requirements Data Sheet





## Capabilities Of Telemetry Brokers (2)

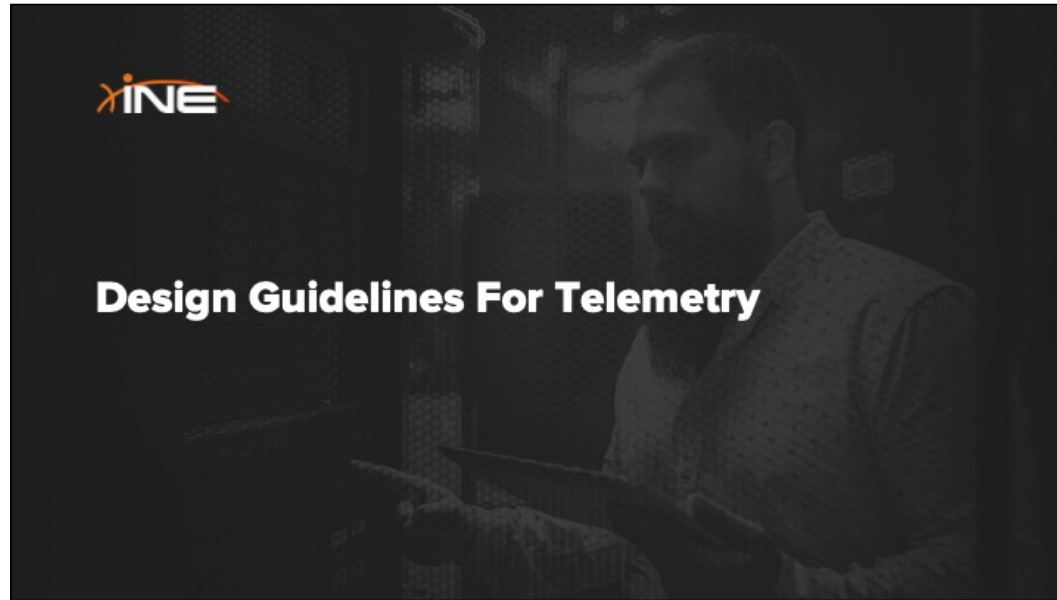
---

- + Bear in mind that the Cisco Telemetry Broker virtual appliance is NOT for model-driven telemetry (MDT sent with NETCONF, gRPC or gNMI) but telemetry from other protocols such as:
  - + IPFIX
  - + NetFlow (all versions)
  - + sFlow
  - + Syslog
  - + SNMP
- + Examples of transforming data with a Telemetry Broker:
  - + VPC Flow Logs: An input that consumes Amazon Web Services (AWS) VPC Flow Logs from an s3 bucket, [transforms them into IPFIX](#), and sends the IPFIX to your destinations.
  - + NSG Flow Logs: An input that consumes Azure NSG Flow Logs from an Azure Storage Account, [transforms them into IPFIX](#), and sends the IPFIX to your destinations





## **Design Guidelines For Telemetry**



## MDT Design Consideration #1

+ Do you wish to utilize dial-in or dial-out telemetry?

Dial-In MDT Considerations	Dial-Out MDT Considerations
Only one command necessary to configure on network devices ("Netconf-yang")	Requires extensive manual configuration on each network device.
Greater selection of protocol support (NETCONF, gNMI, gRPC w/JSON, etc)	Limited protocol support (eg No NETCONF or gNMI support in IOS-XE)
Might require multiple Collector apps within a single server to support different protocols (NETCONF, gNMI, etc)	Configuration complexity grows with each protocol added for subscriptions.
XPATH usually selectable via GUI	Requires detailed XPATH configuration to define sensor paths



## MDT Design **Consideration #2**

---

- + Determine what cadence you wish for your telemetry subscription:
  - + Periodic publications
  - + On-Change publications



- The moral of the story here is that Cisco is admitting in their documentation that you basically must experiment with different YANG modules to see which one's support on-demand publications.

### Cadence Pros and Cons (1)

+ Considerations for “Periodic Publications” (**frequent** cadence);

Cadence Frequency	Pros	Cons
Cadence set to low value for publication	Changes to data received in nearly real-time	Consumes additional bandwidth on the wire
What is the minimum supported configurable cadence? (IOS-XE = 1-second (100 centiseconds))	Highly accurate baselining	May overwhelm receiving database
	Fastest option for quick troubleshooting and error correction	Multiple subscriptions with low cadence values may drive up network device CPU



- The NX-OS Telemetry configuration guide states that telemetry can consume up to 20% of CPU resources!

## Cadence Pros and Cons (2)

- + Considerations for “Periodic Publications” (infrequent cadence);

Cadence Frequency	Pros	Cons
Cadence set to higher value for publication	Consumes less bandwidth on the wire	Real-time visibility to critical changes is lost
	Consumes less resources within receiving databases	Less accurate baselining
	Preserves networking device CPU for multiple subscriptions with low cadence values	Induces delay in troubleshooting and error correction



### MDT Design Consideration #3

---

- + How much time do you want to spend finding relevant YANG models?
- + Most every YANG datatype supports periodic subscriptions.
- + A logical question for on-demand subscriptions;
  - + *"How can I determine **which** YANG models support 'on-demand' publications?"*

#### Determining On-Change Capability

Currently, there is NO indication within YANG models about the type of data that can be subscribed to, by using an on-change subscription. Attempts to subscribe to data that cannot be subscribed to by using on-change subscription results in a failure (dynamic) or an invalid subscription (configured).

Quote courtesy of: Programmability Configuration Guide, Cisco IOS XE Gibraltar 16.10.x



- The moral of the story here is that Cisco is admitting in their documentation that you basically must experiment with different YANG modules to see which one's support on-demand publications.

## MDT Design Consideration #4

+ What MDT encoding method should you select?

gNMI / gRPC	NETCONF
Stateless; interactions are "one and done"	Stateful; Can test for command validity and recover from errors
No support for network-wide transactions	Supports network-wide transactions
No concept of datastores	Can work with multiple datastores
Typically uses protobufs (not human-friendly) but can use JSON	Uses human-friendly XML
Quick and efficient encoding/decoding	Slower encoding/decoding



- A "network-wide" transaction is the idea of initiating multiple, simultaneous transactions (TCP stateful connections) to different devices in order to achieve a common goal, like implement some concept (like a new leg for a L3VPN). With a network-wide transaction, configurations are not pushed from the Candidate to the Running datastores until all devices have reported back that the changes are supported without error.



### MDT Design Consideration #5

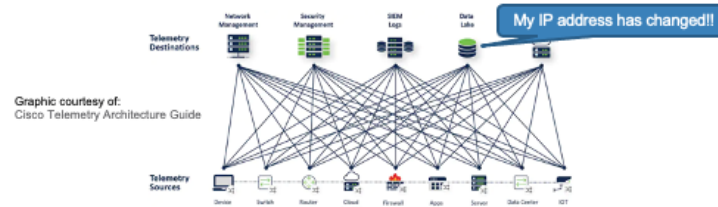
---

- + MDT can consume significant network bandwidth given circumstances such as;
  - + MDT subscriptions enabled between many network devices and receivers
  - + MDT subscriptions set to periodic with very low cadence values
- + Consider how this increased traffic load will affect data on your network.
  - + Perhaps offload MDT data to reserved links
  - + Consider QoS markings and Policing policies for MDT subscriptions



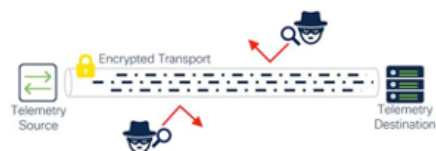
## Non-MDT Design Consideration #6

- + How will you handle the addition of **new** telemetry destinations to your existing DT dial-out configurations?
- + What about changes to the FQDN or IP address of existing telemetry destinations?
- + Consider implementing telemetry brokers so that changes to telemetry destination information can be centrally managed without the need to change telemetry source configurations.



### MDT Design Consideration #7

- + Telemetry data is often sent unencrypted due to lack of encryption support in telemetry sources.
- + This can lead to eavesdropping, stealing of data, or manipulation of data.
- + Considering encrypting telemetry data either at the source (if supported) or implementing IPsec on routers connected to telemetry sources and receivers.



Graphic courtesy of: Cisco Telemetry Architecture Guide

