Programming Network Devices with gRPC and OpenConfig
Nicolas Leiva

• Network Engineer
  • nleiva @ cisco
  • % @
  • % @
  • nlev4 @

Slides
The rise of API's

Beyond the command line
An API is simply a specification of remote calls exposed to the API consumers.
Remote Calls

1. Define the **data** to be transmitted
2. Determine how the data is **serialized** over the wire
3. Choose a **transport** protocol
Define the data to be transmitted
LLDP Neighbors API

Vendor A

```
{
    ...
    "ttl": 120,
    "neighborDevice": "router2.lab.com",
    "neighborPort": "Ethernet4",
    "port": "Ethernet4"
    ...
}
```

Vendor B(J)

```
{
    "lldp-local-port-id": {
        "data": "et-0/0/13"
    }
    ...
    "lldp-remote-port-description": [
        {
            "data": "to router3.lab.com et-0/0/13"
        }
    ],
    "lldp-remote-system-name": {
        "data": "router3.lab.com"
    }
}
```

Vendor C

```
{
    ...
    "port_id": "Ethernet2/1",
    "l_port_id": "Eth2/1",
    "sys_name": "router1.lab.com",
    "ttl": 108,
    ...
}
```
DATA MODELING LANGUAGE

DESCRIBES DATA HIERARCHY
○ CONFIG AND OPERATIONAL DATA AS A TREE STRUCTURE

SPECIFIES RESTRICTIONS, DATA TYPES, ETC.
Interfaces YANG Models (*)

IETF

```yaml
module ietf-interfaces {
  revision 2018-02-20 {
    "RFC 8343: ...";
  }
}
```

OpenConfig

```yaml
module openconfig-interfaces {
  revision "2018-04-24" { reference "2.3.1"; }
}
```

Cisco

```yaml
module Cisco-IOS-XR-ifmgr-cfg {
  revision 2017-09-07 {
  }
}
```
OpenConfig Interfaces (*)

```yaml
container interfaces {
    list interface {
        key "name";
        leaf name {...}
    }
    container config {
        uses interface-phys-config;
    }
    container state {
        uses interface-phys-config;
        uses interface-common-state;
        uses interface-counters-state;
    }
    uses interface-phys-holdtime-top;
    uses subinterfaces-top;
}
```
Vendor neutral, driven by network operators

Combines config and operational data (intended vs derived state)
- Config
- Statistics (e.g., counters)
- Operational State (e.g., BGP session status)
- Applied config (...is part of the state)

Model consistency and semantic versioning
Determine how the data is serialized over the wire
Most Common Options

<table>
<thead>
<tr>
<th>JSON</th>
<th>XML</th>
<th>PROTOBUF (*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
<td>&lt;PERSON&gt;</td>
<td>1 {</td>
</tr>
<tr>
<td>&quot;PERSON&quot;: {</td>
<td>&lt;NAME&gt;John Doe&lt;/NAME&gt;</td>
<td>1: &quot;John Doe&quot;</td>
</tr>
<tr>
<td>&quot;NAME&quot;: &quot;John Doe&quot;,</td>
<td>&lt;EMAIL&gt;<a href="mailto:jdoe@example.com">jdoe@example.com</a>&lt;/EMAIL&gt;</td>
<td>2: &quot;<a href="mailto:jdoe@example.com">jdoe@example.com</a>&quot;</td>
</tr>
<tr>
<td>&quot;EMAIL&quot;: &quot;<a href="mailto:jdoe@example.com">jdoe@example.com</a>&quot;</td>
<td>&lt;/PERSON&gt;</td>
<td>}</td>
</tr>
<tr>
<td>}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
✘ **Human readable/editable**

✘ **Can be parsed without knowing schema in advance**

```json
{
    "PERSON": {
        "NAME": "John Doe",
        "EMAIL": "jdoe@example.com"
    }
}
```

```xml
<Person>
    <Name>John Doe</Name>
    <Email>jdoe@example.com</Email>
</Person>
```
**Protocol Buffers**

- Very dense data (small output)
- Very fast processing
- Not human readable (native format)
- Only meaningful if you have the message definition

```python
1 {  
  1: "John Doe"  
  2: "jdoe@example.com"  
}
```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;
  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }
  repeated PhoneNumber phone = 4;
}

required int32 id = 2;

Value type

Field number

Name
Comparing data-format speeds

Protocol Buffers

Size of data by format
Choose a transport protocol
<table>
<thead>
<tr>
<th>Transport Mechanisms</th>
<th>NETCONF</th>
<th>RESTCONF</th>
<th>gRPC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SSH</strong></td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td><strong>RPC</strong></td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>✔ <code>&lt;get-config&gt;</code></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>✔ <code>&lt;edit-config&gt;</code></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>✔ <code>&lt;commit&gt;</code></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>✔ <code>&lt;lock&gt;</code></td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>✔ ...</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td><strong>HTTP</strong></td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td><strong>Methods</strong></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>✔ GET</td>
<td></td>
<td>✔ GET</td>
<td>✔</td>
</tr>
<tr>
<td>✔ POST</td>
<td></td>
<td>✔ POST</td>
<td>✔</td>
</tr>
<tr>
<td>✔ DELETE</td>
<td></td>
<td>✔ DELETE</td>
<td>✔</td>
</tr>
<tr>
<td>✔ PUT</td>
<td></td>
<td>✔ PUT</td>
<td>✔</td>
</tr>
<tr>
<td>✔ ...</td>
<td></td>
<td>✔ ...</td>
<td>✔</td>
</tr>
<tr>
<td><strong>HTTP/2</strong></td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td><strong>RPC</strong></td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>✔ Unary</td>
<td></td>
<td>✔ Unary</td>
<td>✔</td>
</tr>
<tr>
<td>✔ Server streaming</td>
<td></td>
<td>✔ Server streaming</td>
<td>✔</td>
</tr>
<tr>
<td>✔ Client streaming</td>
<td></td>
<td>✔ Client streaming</td>
<td>✔</td>
</tr>
<tr>
<td>✔ Bidirectional</td>
<td></td>
<td>✔ Bidirectional</td>
<td>✔</td>
</tr>
<tr>
<td><strong>streaming</strong></td>
<td></td>
<td>✔ streaming</td>
<td>✔</td>
</tr>
</tbody>
</table>
HTTP/1.1

Jun 1999
The Secure Shell (SSH) Transport Layer Protocol

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (RFC 1700) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

The Secure Shell (SSH) is a protocol for secure remote login and other secure network services over an insecure network.

This document describes the SSH transport layer protocol, which typically runs on top of TCP/IP. The protocol can be used as a basis for a number of secure network services. It provides strong encryption, server authentication, and integrity protection. It may also provide compression.

Key exchange method, public key algorithm, symmetric encryption algorithm, message authentication algorithm, and hash algorithms are all negotiated.

This document also describes the Diffie-Hellman key exchange method and the mixed set of algorithms that are needed to implement the SSH transport layer protocol.
Abstract

This specification describes an optimized expression of the semantics of the Hypertext Transfer Protocol (HTTP), referred to as HTTP/2. HTTP/2 enables a more efficient use of network resources and a reduced perception of latency by introducing header field compression and allowing multiple concurrent exchanges on the same connection. It also introduces unidirectional push of representations from servers to clients.

This specification is an alternative to, but does not obsolesce, the HTTP/1.1 message syntax. HTTP/2's existing semantics remain unchanged.

Status of this Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errors, and how to provide feedback on it may be obtained at http://www.ietf.org/rfc/rfc7540.txt.
Binary, easier framing

Request and response multiplexing over a single TCP connection

Header compression

Bidirectional streams

Performance difference between HTTP/2 and HTTP/1.1: [https://imagekit.io/demo/http2-vs-http](https://imagekit.io/demo/http2-vs-http)
Strongly typed service and message definition

Takes care of all the underlying plumbing

cross-platform (C, Java, Go, Python, etc)

Runs over HTTP/2

Cloud Native Computing Foundation Project
## Processing time

<table>
<thead>
<tr>
<th>RPC</th>
<th># of clients</th>
<th>total time</th>
<th>per-request time</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsonrpc</td>
<td>1</td>
<td>8m 7.2s</td>
<td>1.624ms</td>
</tr>
<tr>
<td>gRPC</td>
<td>1</td>
<td>36.7s</td>
<td>122.3µs</td>
</tr>
<tr>
<td>gRPC</td>
<td>100</td>
<td>7.1s</td>
<td>23.8µs</td>
</tr>
</tbody>
</table>

## Memory usage

<table>
<thead>
<tr>
<th>RPC</th>
<th># of clients</th>
<th>AllocsPerOp</th>
<th>AllocatedBytesPerOp</th>
</tr>
</thead>
<tbody>
<tr>
<td>jsonrpc</td>
<td>1</td>
<td>32.7M</td>
<td>3.1GB</td>
</tr>
<tr>
<td>gRPC</td>
<td>1</td>
<td>25.2M</td>
<td>1.7GB</td>
</tr>
<tr>
<td>gRPC</td>
<td>100</td>
<td>25.2M</td>
<td>1.7GB</td>
</tr>
</tbody>
</table>
gRPC service interface definitions

**gNMI**

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

**gNOI**

```plaintext
service System {
  rpc Ping(PingRequest) returns (stream PingResponse) {};
  rpc Traceroute(TracerouteRequest) returns (stream TracerouteResponse) {};
  rpc Time(TimeRequest) returns (TimeResponse) {};
  rpc SetPackage(stream SetPackageRequest) returns (SetPackageResponse) {};
  ...
}
```

**Cisco**

```plaintext
service gRPCConfigOper {
  rpc GetConfig(ConfigGetArgs) returns(stream ConfigGetReply) {};
  rpc MergeConfig(ConfigArgs) returns(ConfigReply) {};
  ...
  rpc CreateSubs(CreateSubsArgs) returns(stream CreateSubsReply) {};
}
```

```plaintext
service gRPCExec {
  ...
  rpc ActionJSON(ActionjsonData) returns(stream ActionJSONReply) {};
}
```
Demo Time

Any questions?
Is the session still UP?

Am I receiving all the prefixes?

Am I receiving more prefixes than expected?

Prefix delta > 10-30%?
GROUPING BGP-NEIGHBOR-STATE {
  DESCRIPTION
  "Operational state parameters relating only to a BGP neighbor";

  LEAF SESSION-STATE {
    TYPE ENUMERATION {
      IDLE {
        DESCRIPTION
        "Neighbor is down, and in the Idle state of the FSM";
      }
      CONNECT {
        DESCRIPTION
        "Neighbor is down, and the session is waiting for
        the underlying transport session to be established";
      }
      ACTIVE {
        DESCRIPTION
        "Neighbor is down, and the local system is awaiting
        a connection from the remote peer";
      }
      OPENSENT {
        DESCRIPTION
        "Neighbor is in the process of being established.
        The local system has sent an OPEN message";
      }
      OPENCONFIRM {
        DESCRIPTION
        "Neighbor is in the process of being established.
        The local system is awaiting a NOTIFICATION or
        KEEPALIVE message";
      }
      ESTABLISHED {
        DESCRIPTION
        "Neighbor is up – the BGP session with the peer is
        established";
      }
    }
  }
}

https://github.com/nleiva/xroc
Further reading

- gRPC and GPB for Networking Engineers
  https://github.com/nleiva/gmessaging

- Programming IOS-XR with gRPC and Go

- Validate the intent of network config changes
  https://xrdocs.github.io/programmability/tutorials/2017-08-14-validate-the-intent-of-network-config-changes/

- ygot (YANG Go Tools)
  https://github.com/openconfig/ygot

- YANG Development Kit (YDK)
  https://developer.cisco.com/site/ydk/

- OpenConfig GitHub
  https://github.com/openconfig