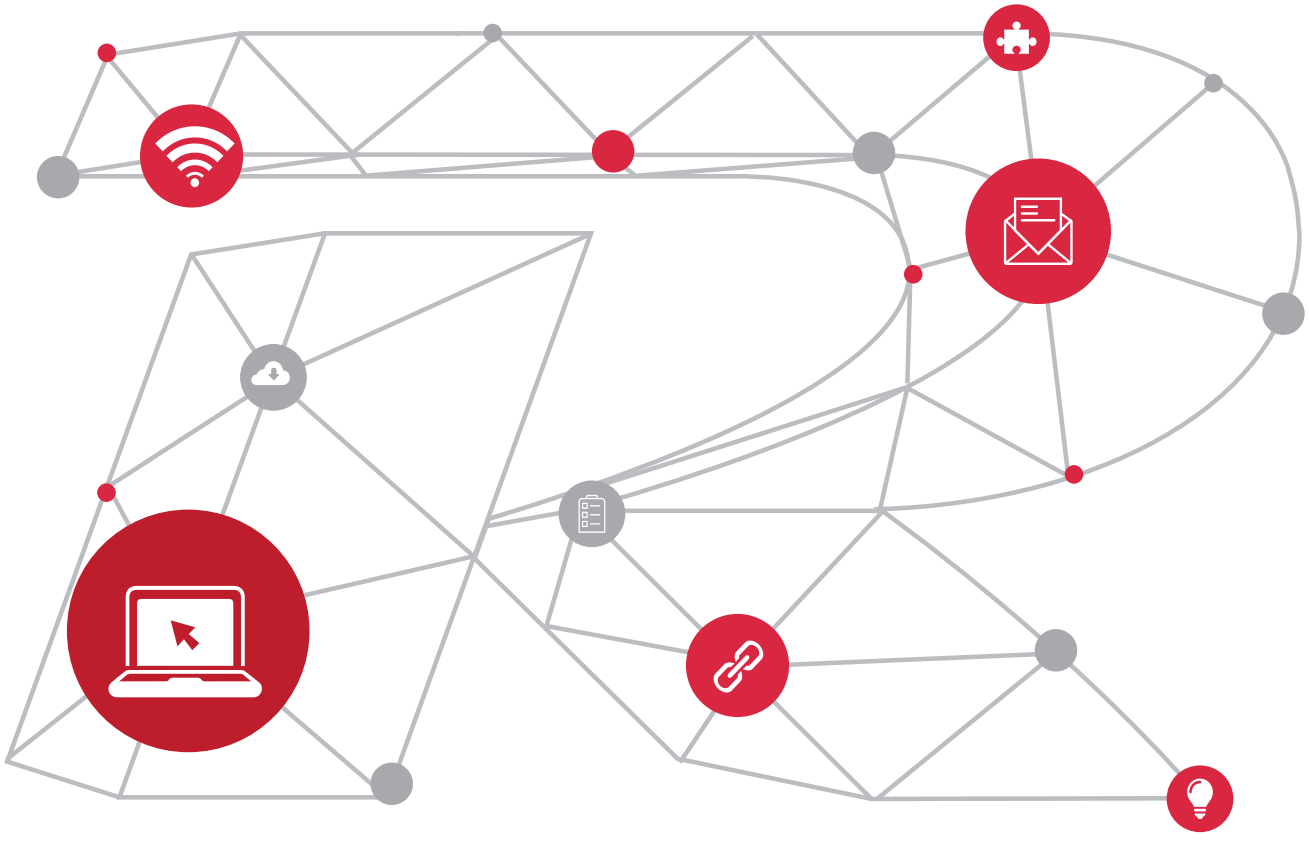
White Paper

A Feast of Technologies About gRPC



Contents

Getting Over the Last Step to gRPC 3

Background and Business Scope of gRPC 3

Overview 4

What is Protobuf? 5

Faster Transmission Owing to Serialization 6

Support for Cross-Platform Multilingual Programming 7

HTTP 2.0-Based Design 7

Bidirectional Communication and Multiplexing 8

Binary Frames 8

Header Compression 8

Summary 8

Getting Over the Last Step to gRPC

In the future, most Internet data centers (IDCs) are software-defined, and innovative technologies such as cloud computing, big data, and artificial intelligence (AI) are used to integrate conventional network resources, server resources, and storage resources. In addition, an increasing amount of graphics processing unit (GPU) and high-performance computing (HPC) service data is transmitted in IDCs. Higher bandwidth and lower latency are required for the network. During maintenance, an automated platform can be used to collect information to quickly adapt to the network. This improves maintenance efficiency, helping you build a network with higher availability, reliability, and controllability.

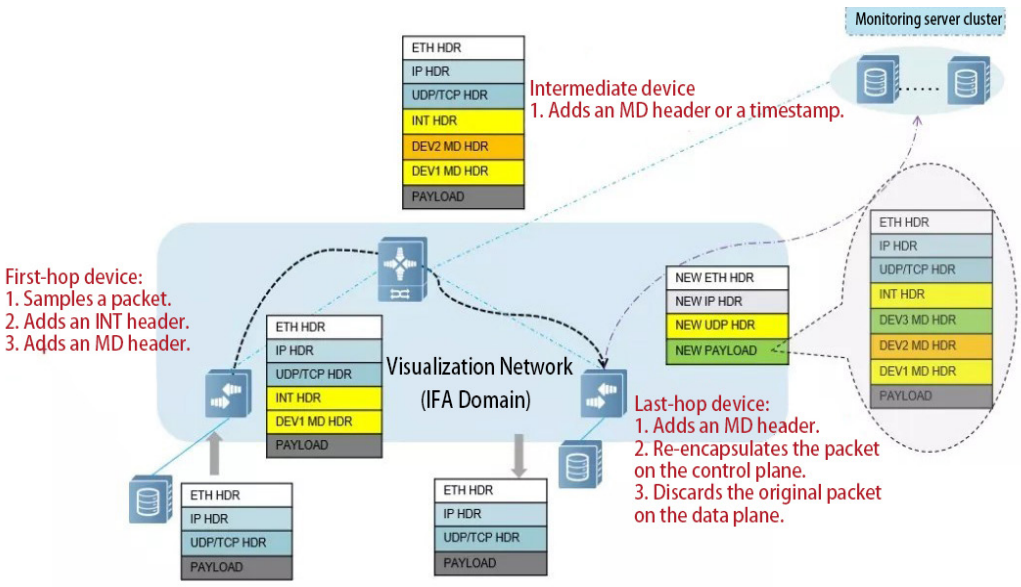
The previous article **A Feast of Technologies | The "Sword of Giant" for Maintenance in IDCs** delivers a general understanding of the gRPC technology by describing and comparing conventional maintenance technologies with Google Remote Procedure Call (gRPC). Today, we'll explore the gRPC framework in more detail.

Background and Business Process of gRPC

GPU and HPC services are prone to micro bursts of traffic. Maintenance personnel need to quickly detect such bursts, locate the cause, and make adjustments. Conventional network management tools such as the command line interface (CLI) and Simple Network Management Protocol (SNMP) cannot meet the requirements of automated maintenance. Therefore, we need a technology to achieve higher-precision data monitoring without affecting performance and functionality of network devices.

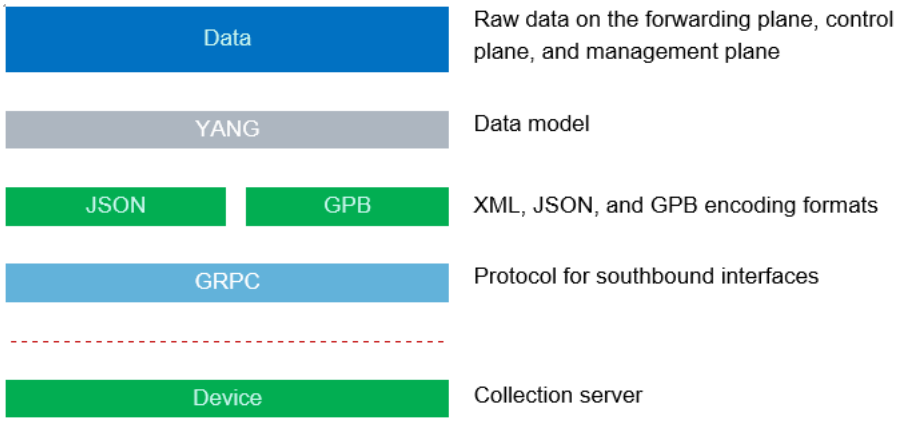
As previously mentioned in the "A Feast of Technologies" series, the In-band Network Telemetry (INT) technology can visualize end-to-end forwarding paths of traffic, as shown in Figure 1. However, the INT technology cannot comprehensively manage buffers of the switches, for example, buffers for ingress and egress ports and queues cannot be monitored in real time. In spite of this disadvantage, you can still properly visualize and monitor maintenance for network elements in a single network in real time by designing maintenance interfaces based on gRPC and Protocol Buffers (Protobuf).

Figure 1: INT Interaction Process



As known to all, the telemetry model on the device side includes the raw data, data model, encoding formats, and transmission protocol, as shown in Figure 2. The transmission protocol used here is gRPC. The following provides a simple analysis on gRPC.

Figure 2: Layered Telemetry Model



Overview

gRPC is an open-source, high-performance, HTTP 2.0-based software framework released by Google. It allows you to configure and manage network devices by using multiple programming languages. In an open-source framework, both communication parties can carry out secondary development. Therefore, the communication between the client and the server focuses on service data, reducing workload required for communication at lower layers, which can simply be implemented by the gRPC framework. As shown in Figure 3, the data layer processes service data, and all data at lower layers is encapsulated by gRPC.

Figure 3, the data layer processes service data, and all data at lower layers is encapsulated by gRPC.

Figure 3: Layered gRPC Framework

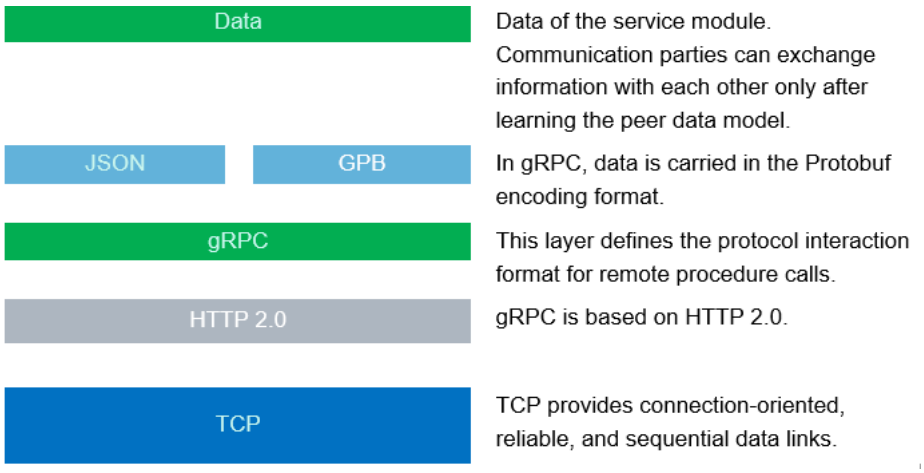


Figure 4 shows the structure of gRPC packets.

Figure 4: Structure of gRPC Packets

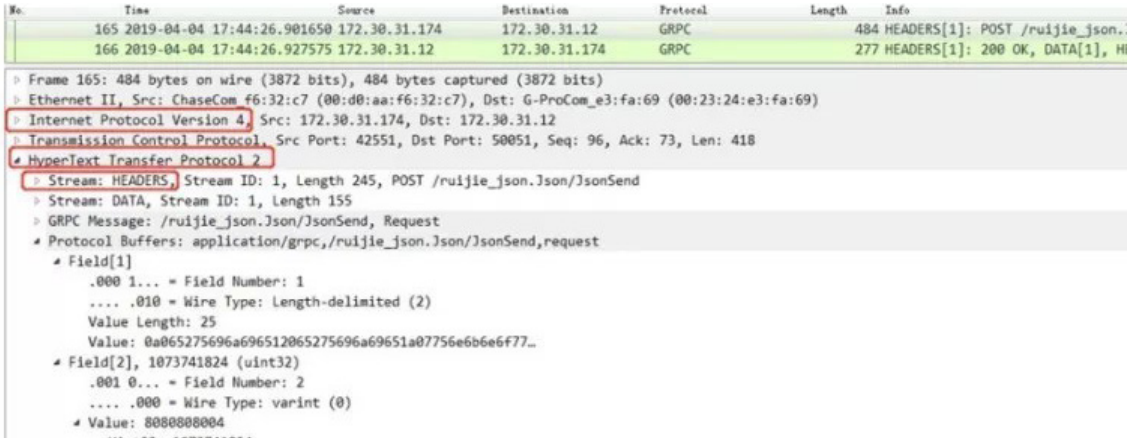
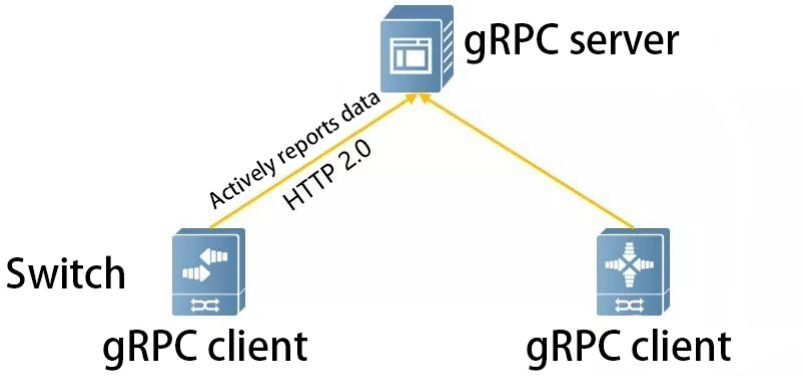


Figure 5 shows the gRPC interaction process.

Figure 5: gRPC Interaction Process



* A switch with gRPC enabled serves as a gRPC client and the collection server serves as a gRPC server.

* The switch constructs packets in GPB or JSON format based on the subscribed event, defines a .proto file by using Protobuf, establishes a gRPC channel to the server, and sends a request message to the server over gRPC.

* After receiving the request message, the server parses the .proto file by using Protobuf to restore the previously defined data structure, and then processes the service.

* After the data is sorted, the server compiles response data by using Protobuf and sends a response message to the switch over gRPC.

* After the switch receives the response message, the gRPC interaction ends.

The preceding workflow shows a detailed gRPC interaction process. The process is one of the telemetry trigger modes, called the dial-out mode. Simply put, after gRPC is enabled, it can establish a connection between the client and the server, and push subscribed data configured on the device to the server. As illustrated, Protobuf is used to define the to-be-processed structured data in the .proto file in the entire process.

What is Protobuf?

Protobuf can be understood as a more flexible and efficient data format. Similar to XML and JSON, it is very suitable for some data transmission scenarios that require high performance and fast responses.

In the gRPC framework, Protobuf can:

* Define the data structure.

```
syntax = "proto3";

import "google/protobuf/any.proto";
import "google/protobuf/descriptor.proto";
import "github.com/openconfig/gnmi/proto/gnmi_ext/gnmi_ext.proto";

message Notification {
  int64 timestamp = 1;
  Path prefix = 2;
  string alias = 3;
  repeated Update update = 4;
  repeated Path delete = 5;
  bool atomic = 6;
}
```

* Define the service API.

```
syntax = "proto3";

import "google/protobuf/any.proto";
import "google/protobuf/descriptor.proto";
import "github.com/openconfig/gnmi/proto/gnmi_ext/gnmi_ext.proto";

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

* Serialize and deserialize data to increase transmission efficiency.

• Faster Transmission Owing to Serialization

As known to all, text data compiled by using XML and JSON is easier to read. However, during data exchange, the device needs to spend a lot of CPU resources on data I/O, which affects the entire transmission rate. Unlike the preceding formats, Protobuf serializes the strings into binary data before transmission.

Table 1: GPB Code and Corresponding JSON Code

Protobuf Code	JSON Code
{	{
1: "Ruijie"	"producerName": "Ruijie",
2: "Ruijie"	"deviceName": "Ruijie",
}	}

According to the table above, the two formats are both intuitive and not much different from each other. The Protobuf code is provided only for operators. The data to be transmitted is actually not in this text format, but is serialized binary data. The number of bytes in Protobuf is much less than that in JSON and XML. Therefore, the transmission rate is higher.

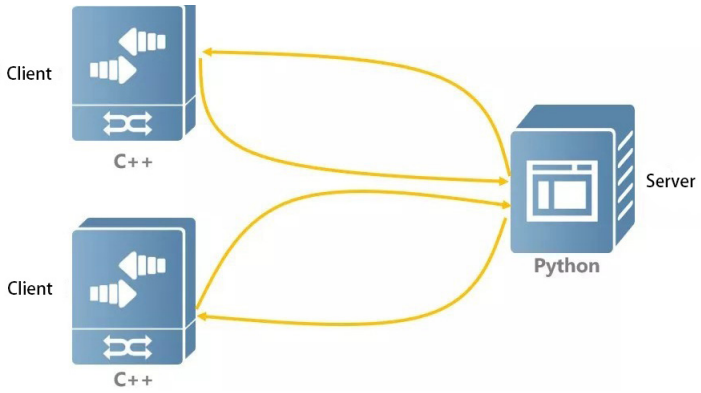
In the current or future era of data explosion, Protobuf is naturally advantageous in transmission efficiency due to its binary transmission when compared with XML and JSON. Data collection efficiency is definitely a key point in architecture design and maintenance construction.

• Support for Cross-Platform Multilingual Programming

Another advantage of Protobuf is its built-in compiler. The preceding .proto file is compiled by using the compiler. The .proto file must be compiled into a library file, based on which you can develop data applications. What programming language can be used to generate this library file? Network device maintenance personnel and server maintenance personnel are usually different and they may use different programming languages to maintain and develop the devices. In this case, Protobuf stands out owing to its support for multiple languages.

For example, in an IDC network, the server side uses Python while the client side (switch side) is more likely to use C++. This difference does not affect interaction between the two sides when Protobuf is used. Figure 6 shows the cross-platform transmission in multiple languages.

Figure 6: Cross-Platform Transmission in Multiple Languages



According to the preceding description, we can summarize the following encoding advantages of Protobuf compared to JSON and XML:

* Simple and small-size data: The data description file has a size of only 1/10 to 1/3 of that in JSON and XML.

* Fast transmission and parsing: The parsing speed is 20 times or even higher than that of XML.

* High compilability

As shown in FIG. 3 and FIG. 5, gRPC has another advantage in addition to Protobuf: gRPC is based on HTTP 2.0.

HTTP 2.0-Based Design

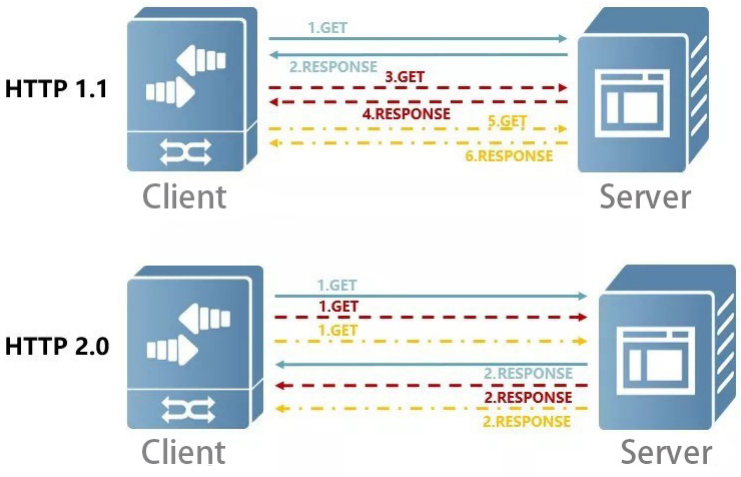
gRPC is designed based on HTTP 2.0. This brings more powerful features, such as multiplexing, binary frames, header compression, and push mechanism. These features are significantly beneficial to devices, for example, saving the bandwidth, reducing TCP connections, and lowering CPU utilization. gRPC can be applied on both the client and the server, enabling transparent communication between the two sides and simplifying construction of the communication system.

There are HTTP 1.X and HTTP 2.0, where the former is most widely used and the latter is called the second generation of HTTP. HTTP 1.X defines four methods for interacting with the server: GET, POST, PUT, and DELETE. These methods are all inherited in HTTP 2.0. Let's take a look at what is new in HTTP 2.0.

• Bidirectional Communication and Multiplexing

In HTTP 1.X, the number of requests sent by the client to access the same domain name at the same time is limited. When the threshold is reached, other requests are blocked. By contrast, this case does not occur in HTTP 2.0. HTTP 1.X transmits plaintext data of a relatively large size. HTTP 2.0 transmits data by frame, where each frame contains a message. In addition, HTTP 2.0 allows simultaneously initiating multiple "request-response" messages through one connection. This can implement concurrent streams with no need to set up multiple TCP connections, thereby increasing the throughput. Moreover, this allows managing priorities of multiple messages within one connection and performing flow control. Figure 7 shows the bidirectional communication and multiplexing in HTTP 2.0.

Figure 7: Bidirectional Communication and Multiplexing



• Binary Frames

Different from HTTP 1.X that transmits plaintext, HTTP 2.0 transmits binary data, and is mutually complementary with Protobuf. Such practice enables a small data size and low load, ensuring compact and efficient transmission.

• Header Compression

HTTP is a stateless protocol. It does not memorize service processing information. As a result, each request must carry all the details of the device. In particular, the header contains a large amount of duplicate data. This means that the device keeps repeating the same work. In comparison, the "header table" is used in HTTP 2.0 to track the data that is previously sent. You no longer need to repeatedly request or send the same data. Therefore, the data size is reduced.

Summary

While high-performance services such as AI and HPC depend increasingly heavily on the network, you must consider status monitoring at the beginning of network design. In particular, you must consider how to quickly and accurately monitor the status of network-wide devices and links in real time during maintenance, so as to ensure smooth running of services. gRPC has been applied on switches in IDCs and deployed in partial scenarios of some Internet companies. In addition, it is now being fully explored to replace the SNMP protocol as the only southbound interface.

During gRPC-based communication, both the client and server sides must define the .proto file and then define the service APIs based on the .proto file. These APIs allow atomic operations, such as Get, Set, Notification, and Subscribe. The data model may be JSON or YANG. The YANG model is recommended, because it is easier to maintain and scale. However, as described in a previous article, the key challenge lies in how to unify the YANG model, which needs to be further explored.