Comparing the performance of SNMP to Network Telemetry streaming with gRPC/GPB

Ivan Ivanov[1]

Abstract – **This paper compares the performance of SNMP based network monitoring to Network Telemetry streaming with remote procedure call framework (gRPC) and Google Protocol Buffers (GPB). The study focuses on retrieving the** ifTable **from network elements using both SNMP and gRPC/GPB based approach. First, we evaluate the performance of SNMP using GetNext and GetBulk messages. Then, we get the same** ifTable **information encoded with Google Protocol Buffers using TCP and UDP for transport. The performance is then measured as a function of the number of retrieved objects. Several aspects are examined: bandwidth usage, round trip times and CPU.**

Keywords –**SNMP, gRPC, Network Telemetry, Google Protocol Buffers, Pipeline, XRV9k**

I. INTRODUCTION

Streaming network telemetry is a new paradigm in networking management and monitoring which hasn't been widely deployed yet. It utilizes the idea of "push the data" not "pull the data". To retrieve any information from a network device using SNMP, NMS (Network Management System) needs to first request this data in form of an SNMP request. Only then the data can be sent from the network device back to the NMS in form of SNMP response message/s. This is repeated every polling interval. To retrieve large amounts of data, SNMP polling relies on the GetBulk operation. It performs a continuous GetNext operation that retrieves all the columns of a given table (e.g. *ifTable*). The network device will return as many columns from the *ifTable* as can fit into a single packet. If the polling NMS detects that the end of the table has not yet been reached, it will do another GetBulk and will repeat the operation until the whole *ifTable* is fetched.

Streaming network telemetry gains efficiency over SNMP by eliminating the polling process altogether. Instead of sending SNMP requests with specific instructions that the network device must process every time, telemetry uses a configured policy on the device to know what data to collect, how often and to which NMS it should be sent.

This paper focuses on retrieving the *ifTable* from network devices using SNMP polling and at the same time streaming the same network information encoded with Google Protocol Buffers and compare the results.

II. RELATED WORKS

In literature several papers can be found that investigates the performance of SNMP [1] and compares the performance of SNMP to Web Services/XML-based monitoring systems [2]. Also, the performance of SNMP trap notification was directly compared to Web Services notifications using XML gateways [3]. Few papers have been published that discusses the performance of SNMP in large-scale deployments and provide analysis of the traffic patterns of large-scale monitoring systems [4].

Streaming network telemetry is a new approach for monitoring and managing communication networks. Early-release implementations of gRPC-based streaming telemetry are deployed by vendors like Cisco and Juniper. Several papers analyze different use cases of this approach [5][6].

This research was directly motivated by these publications and uses some of the proposed approaches, tools, prototypes and formulas. The purpose of this paper is to compare the performance of SNMP to streaming network telemetry using gRPC with Google Protocol Buffers. In this work, only SNMP versions 1 and 2c are used for measurements and analysis. Another study which focuses on SNMPv3 and compares the performance of the security features of SNMP to streaming network telemetry using gRPC is being worked upon and will be published as separate paper related to these works.

III. MEASUREMENT SET-UP

Within this study many of the measurements were performed on virtual network devices running on VMWare Workstation Pro 14.1.1 running on top of Windows 10. Used are the following virtual images and releases:

- Cisco IOS XRv 9000 Router release 6.4.x
- Cisco Nexus 9000/3000 Virtual Switch release 7.0.x

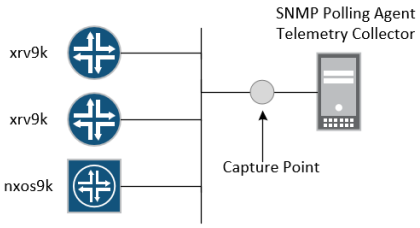

Figure 1. Measurement Set-up

In cases where additional *ifTable* rows were necessary, tunnels to other systems were created or dot1q sub-interfaces were added. ICMP traffic was generated between the systems to increase the counters of all interfaces. To measure bandwidth and delay, open source packet analyzer Wireshark was connected at the traffic capture point as shown Figure 1. The SNMP polling was done using the latest version of snmp daemon running on Ubuntu 16.04. Pipeline Telemetry Collection Service was used as network telemetry collector running on the same Ubuntu 16.04 host.

[1]Ivan Ivanov is with the Faculty of Telecommunications at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mail: ivanov.ivan.iliev@gmail.com.

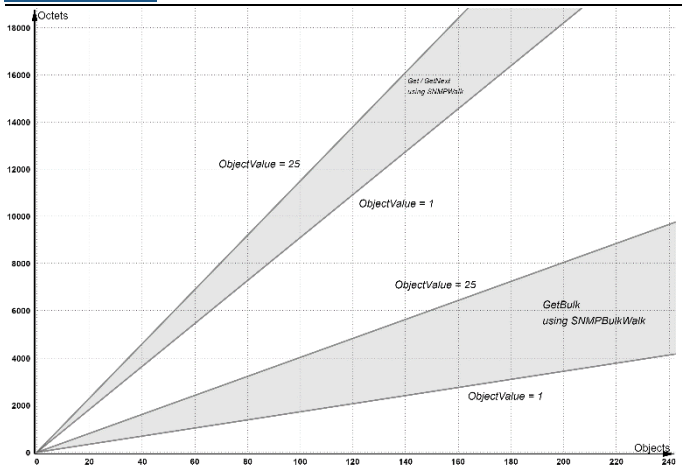

Figure 2. Theoretical SNMP bandwidth consumption

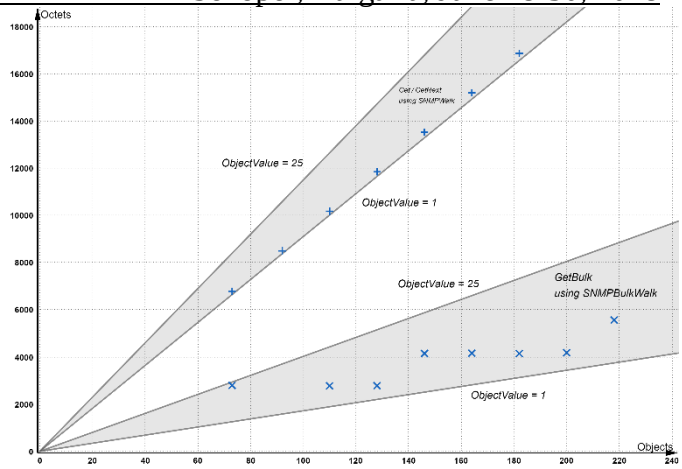

Figure 3. Measured SNMP bandwidth consumption

IV. BANDWIDTH USAGE

This section discusses and compares the bandwidth usage when retrieving data from network elements using SNMP polling and when receiving the same data via network telemetry service.

A. *SNMP messages and encoding*

Since SNMP is fully standardized protocol, the structure of a v1/v2c message is well defined. It consists of two parts: a header and a PDU. The header contains two fields: version (integer) and community (octet string). The PDU consists of five fields: PDU type (integer), Request ID (integer), Error Status / Non-repeaters (integer), Error Index/Max-repetitions (integer) and varlist (sequence). To fully understand the details of an SNMP frame, it is best to be considered as a set of nested fields. The main piece of information is the Object Identifier (OID), which identifies exactly the value to Get (read) or Set (write).

The message and it elements are defined as ASN.1 constructs. In SNMP there are two different sets of data: primitive data and complex data. The length of these data types is variable, so Basic Encoding Rules (BER) is used to solve this problem and transmit the message on the wire.

The most common ASN.1 types are INTEGER, OCTET STRING, OBJECT IDENTIFIER and SEQUENCE. In most cases, both their ASN.1 type part and ASN.1 length part takes a single octet. Therefore, the length of any of these common types depends directly on the length of the ASN.1 value part. The number of octets needed for the value part varies:

- INTEGER requires between one and five octets
- OCTET STRING requires the same number of octets as the length of the string.
- OBJECT IDENTIFIER requires the same number of octets as its length minus one.
- SEQUENCE is a construct for other types and does not require any octets for its value part.

Using the formulas derived in [2] "Comparing the Performance of SNMP and Web Services-Based Management" IEEE 2004, we can calculate the SNMP bandwidth usage in the next section.

B. *Theoritical SNMP bandwidth consumtion*

For each retrieval operation, two SNMP messages are required: a request and a response. The number of octets for the complete operation can be expressed as:

$$L_{DataRetrieval} = L_{request} + L_{response} \quad (1)$$

In an SNMP request, the BER encoding of the object value requires only two octets, because the L_{Value} is NULL. Therefore, the length of a request and response messages can be expressed as:

$$L_{Request} \approx 29 + n \cdot (5 + OID_{length}) \quad (2)$$

$$L_{Response} \approx 29 + n \cdot (5 + OID_{length} + L_{ObjectValue}) \quad (3)$$

For all measurements, all retrieved objects were from the *ifTable*, therefore the OID_{length} will be equal to 11 and we can rewrite (2) and (3) as:

$$L_{Get} \approx 58 + n \cdot (32 + L_{ObjectValue}) \quad (4)$$

$$L_{Bulk} \approx 74 + 16 \cdot n + n \cdot L_{ObjectValue} \quad (5)$$

Using (4) and (5), it is now possible to graphically represent the SNMP's bandwidth requirements as a function of the number of retrieved objects.

To verify the theoretical bandwidth projection on Figure 2, hundreds of MIB objects were retrieved from network elements. Retrieval included all rows of the *ifTable*. The results are shown on Figure 3. After all measurements were completed, all results fall into the expected areas.

C. *Streaming Network Telemetry encoded with Google Protocol Buffers (GPB).*

Telemetry describes how information from network elements can be collected using automated communication processes and transmitted to one or more telemetry collectors.

Network Telemetry is a new approach for network management and monitoring in which data is streamed from network elements continuously using a push model and provides near real-time access to operational statistics (e.g. *ifTable* for this paper).

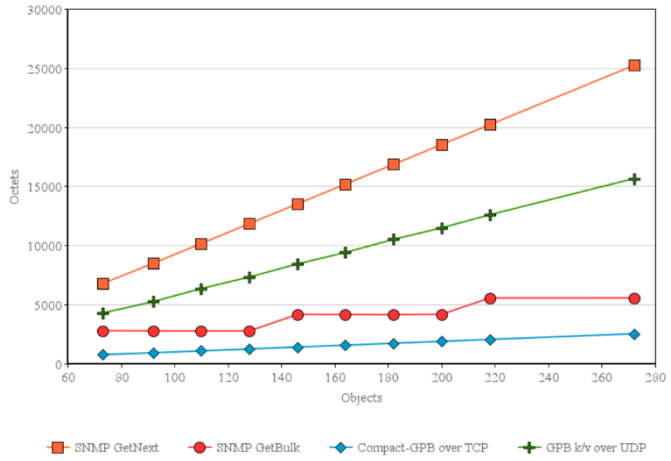

Figure 4. Measured bandwidth usage of SNMP vs Streaming Telemetry

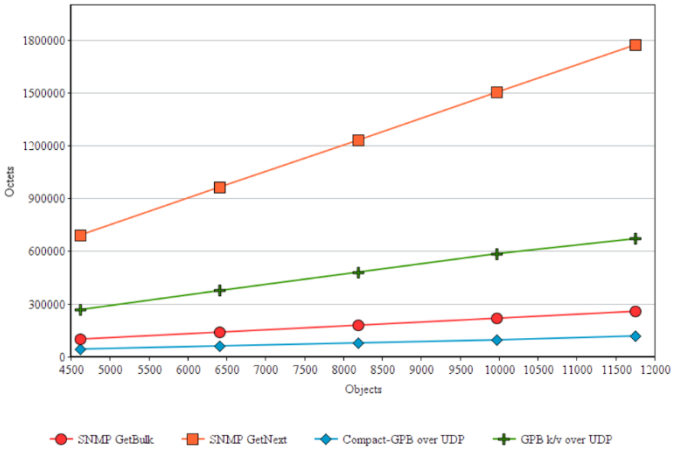

Figure 5. Large scale bandwidth usage of SNMP vs Streaming Telemetry

Since there is no standard yet for streaming network telemetry encoded with Google Protocol Buffers, it is not possible to accurately calculate the upper and lower bounds for the bandwidth needed to fetch the *ifTable* data. It had to be measured. In fact, the bandwidth required depends on specific Google Protocol Buffer definitions, which varies from case to case. This paper therefore only discusses the bandwidth requirements of our prototypes. The discussion focuses on the prototypes that receive the entire *ifTable* within a single interaction.

Interface statistics sent with telemetry using Google Protocol Buffers represent a superset of SNMP interface statistics since the network devices store 36 internal statistics for every interface and the *ifTable* has only 18 statistics per interface.

Every network device has a big number of internal databases which store raw data used for operational tasks that the device is performing. Before this raw information gets available for exporting out of the device, it has to be indexed and mapped to a data model. In the case of SNMP, the information is organized hierarchically using Management Information Bases (MIB) and Object Identifiers (OID). SNMP imposes a very tight model when it comes to indexing and exporting. In the case of the *ifTable*, each column of the table represents a different parameter for a given interface, indexed by the *ifIndex* as show in Table I.

TABLE I
IFTABLE EXAMPLE

ifIndex	ifDescription	ifType	ifSpeed	ifMTU	...
11	Loopback0	6	10000000	1500	...
12	GigabitEthernet0/0/0/1	24	1000000000	1500	...
13	FastEthernet0/0	6	100000000	1514	...

In the case of network telemetry, the internal raw data is mapped to an open-source data modeling language YANG [7]. The language, being protocol independent, can then be converted into any encoding format, e.g. XML, JSON or GPB, that the network configuration protocol supports. In our case, two types of message encoding with Google Protocol Buffers are used: Compact-GPB and GPB key-value (GPB k/v). In compact GPB, the "key" that the network device includes in the packet is just an integer.

For the interface statistics, the telemetry collector will get data that looks like this:

1: GigabitEthernet2/0/1
2: 10000
3: 1500
4: 4243242
5: 43243

Obviously that number 1 stands for the interface name, but what about 2, 3, 4, 5 etc.? To decode these keys, the telemetry collector needs a Google Protocol Buffers definition file called ". proto". With compact GPB, a ".proto" file must be generated on the network element for every path that is to be streamed and uploaded to the network collector.

With this ".proto" file, the network telemetry collector can determine that key (or "field number") "4" means packets_received, "5" means bytes_received, and so on.

This encoding is compact. It is far more efficient to send integers like "34" across the wire than strings like "MulticastPacketsReceived." And GPB is really good at sending integers on the wire: it uses the concept of "varints"[8] to serialize integers even more efficiently (i.e. a 64 bit integer doesn't actually need to take up 64 bits to be sent on the wire). From our measurements shown on Figure 6, it is obvious that the compact-GPB encoding uses the least amount of bandwidth to transmit the same amount of data (or more) than the other encoding methods and SNMP GetNext / GetBulk methods.

In the GPB key-value format, the key is sent as a string. Strings are much less efficient on the wire than varints but they are self-describing. This means that the network collector doesn't need a Google Protocol Buffers definition - ".proto" file for every path. It uses a single ".proto" file for all paths, then read the keys to figure out what the values refer to. This encoding method is easier to set-up on both sides – network telemetry collector and the network element itself but note how larger the data usage gets. For example, sending one instance of the statistics of 653 interfaces takes 610Kbytes of data. Sending the same interfaces' statistics encoded with Compact-GPB takes only 106Kbytes of data. At the same time, retrieving the same information with SNMP polling takes 258Kbytes. Therefore, we can conclude that getting the

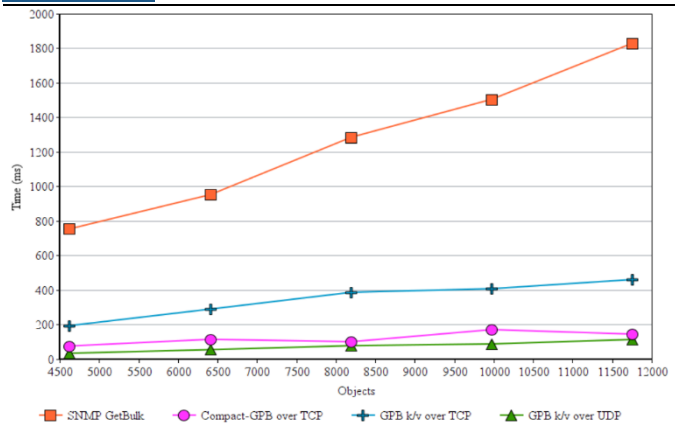

Figure 6. Round-trip delay of SNMP vs delay of Network telemetry

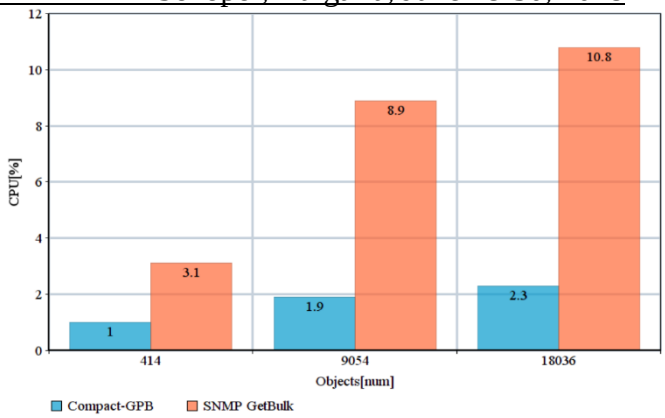

Figure 7. CPU usage of SNMP vs Network telemetry using Compact-GPB

IfTable statistics encoded with Compact-GPB is approximately 2.5 times more bandwidth efficient than SNMP polling with GetBulk. On the other hand, SNMP GetBulk is approximately 2.5 times more efficient than streaming network telemetry using GPB key-value pairs.

V. Delay and CPU usage

The strict semantics of the SNMP GetNext/GetBulk operations force the network device to traverse the *ifTable* column by column from lowest index value to highest. From a network device's perspective, that is not optimal. Network devices store their internal information in a way that is most efficient for their operational needs. In IOS XR, the internal data structure for interface statistics is indexed by interface name and is stored in a structure called a bag. The router's most efficient internal bulk data retrieval is to grab a whole bag of data at once. But the router cannot just send the bag in SNMP. Instead, it has to re-order the data into a table and walk the columns to fulfill the GetBulk request indexed by the *ifIndex*.

Telemetry collects data using the internal bulk data collection mechanisms, does some minimal processing to filter and translate the internal structure to a Google Protocol Buffer, and then pushes the whole thing to the network collector at the configured intervals.

By eliminating the process of re-ordering of the information like in the case of SNMP, streaming network telemetry is more process efficient and requires less CPU cycles. As seen on Figure 6, measured round-trip delays, including packetization, serialization and processing of network telemetry data is approximately 4 times lower than that of SNMP. Figure 7 shows the measured CPU usage at the time of retrieval of the interface statistics using both SNMP and Compact-GPB.

VI. Conclusion

This paper compared the performance of SNMP to network telemetry streaming encoded with Google Protocol Buffers. In particular, it investigated the bandwidth usage, delays and CPU usage.

Our measurements show that retrieving the *ifTable* statistics encoded with Compact-GPB is approximately 2.5 times more bandwidth efficient than SNMP polling with GetBulk. Measured round-trip delays, including packetization, serialization and processing of the telemetry stream is approximately 4 times lower than that of SNMP. Additionally, the measurements show that network telemetry is less CPU intensive than SNMP polling.

Network telemetry is still complex to set-up and is not standardized. It lacks compatibility between vendors and even between different platforms of the same vendor.

References

[1] C. Pattinson, "A Study of the Behaviour of the Simple Network Management Protocol," in Proceedings of DSOM2001, October 2001.

[2] A. Pras, T. Drevers, R. van de Meent and D. Quartel, "Comparing the performance of SNMP and Web services-based management," in IEEE Transactions on Network and Service Management, vol. 1, no. 2, pp. 72-82, Dec. 2004.

[3] W. Queiroz de Lima, R. S. Alves, R. L. Vianna, M. J. B. Almeida, L. M. R. Tarouco and L. Z. Granville, "Evaluating the Performance of SNMP and Web Services Notifications," 2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006, Vancouver, BC, 2006, pp. 546-556.

[4] J. Schonwalder, A. Pras, M. Harvan, J. Schippers and R. van de Meent, "SNMP Traffic Analysis: Approaches, Tools, and First Results," 2007 10th IFIP/IEEE International Symposium on Integrated Network Management, Munich, 2007, pp. 323-332.

[5] F. Paolucci, A. Sgambelluri, M. Dallaglio, F. Cugini and P. Castoldi, "Demonstration of gRPC Telemetry for Soft Failure Detection in Elastic Optical Networks," 2017 European Conference on Optical Communication (ECOC), Gothenburg, 2017, pp. 1-3.

[6] T. Choi, S. Yoon and S. Song, "Information fusion based agile streaming telemetry for intelligent traffic analytics of softwarized network," 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS), Seoul, 2017, pp. 399-402

[7] YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF); Internet Engineering Task Force (IETF) M. Bjorklund, Ed.; Tail-f Systems; October 2010

[8] Google Protocol Buffers Encoding Integers - binary wire format https://developers.google.com/protocol-buffers/docs/enco