Technical Disclosure Commons

Defensive Publications Series

December 07, 2017

Consistent Hashing for Generating Protobuf Field Numbers

Tomasz Madejski

Rob Shakir

Consistent Hashing for Generating Protobuf Field Numbers

Abstract

To convert a YANG-based model into a protobuf format, a unique field identification numbers needs to be assigned to each node in the YANG schema tree. Field identification numbers can be generated by calculating a numeric hash of characteristics of the YANG schema node; for example, its identifier or schema tree path. The resulting hash values provide backwards compatible field identification numbers suitable for protobuf field numbers.

Introduction

Protocol buffers, also referred to as “protobuf” is a language-neutral, platform-neutral extensible mechanism for serializing structured data. The schema for the data to be serialized using protobuf is defined in a .proto file. The .proto file defines a set of message types. Each message type includes a respective set of uniquely numbered fields. In addition to the unique number, each field has a name and a value type. In general, when binary serialized, protobuf encoding tends to require many fewer bits than other encoding schemes, for example, XML or JSON encoding.

YANG is a protocol-independent data modeling language originally intended for data sent via the NETCONF network configuration protocol. More recently, YANG has been adopted for modelling of data related to network devices, independently of the transport protocol used to carry the data. For example, the OpenConfig working group has focused on compiling vendor-neutral data models described in YANG to support operational needs of network users. YANG models are generally serialized in XML or JSON. Encoding YANG models using protobuf

would be desirable to, among other things, reduce the amount of data needed to encode the models.

Discussion

A challenge with generating a protobuf definition from a YANG model is that there are no field numbers specified in YANG, yet they are required in Protobuf. The field numbers must be consistent, such that the generated protobufs remain backwards compatible; for example, when converting OpenConfig YANG models into protobufs used for managing network devices, such as switches, routers, etc. Such OpenConfig models may change over time, yet it is desirable to have any resulting protobuf generated for any updated models to be consistent with earlier generated models; for example, the fields that were present in previous version and are present in the current should have the same numeric ID.

One possible solution to this problem would be to add specific field-number and field-number-offset annotations to a YANG schema. The annotations would allow an indication of the field numbering at authoring time. However, this would require YANG extensions to be defined, and the OpenConfig models to be edited to add these annotations.

As an alternative to the above proposal, we propose computing a hash of a characteristic of the YANG field to compute a number to be used as a field tag. This tag should be in the range 1 to 2^{29}-1, excluding the range 19000-19999, which is reserved for protobuf internal usage.

For a YANG to protobuf transformation, each container or list (directory entry) in the YANG model is mapped to a protobuf message. We can exploit the fact that the entity (leaf, leaf-list, container, or list) name in the YANG model must be unique within such YANG directories, and hence we can use YANG node names as the input to the hash function. This ensures that we have unique input to the hash function for each container that is being utilized.

In the case that the hash function is:

- consistent - i.e., computes the same hash value for the same input each time it is run;
- independent for each value computed - i.e., does not share any state about other values it has been asked to hash;
- produces sufficiently low collisions between hashes

then it is possible to use such a hash function to generate a unique field number for a particular input. Hash functions meeting the above criteria can be used to compute field tags for an entity within a YANG model, without requiring explicit annotation of the model.

This approach has a significant advantage in that it allows generation of protobufs from *any* YANG schema, with no need for particular annotation to be used within the schema. Figure 1 below shows a flowchart of an example implementation of a method of assigning field numbers to a protobuf message.

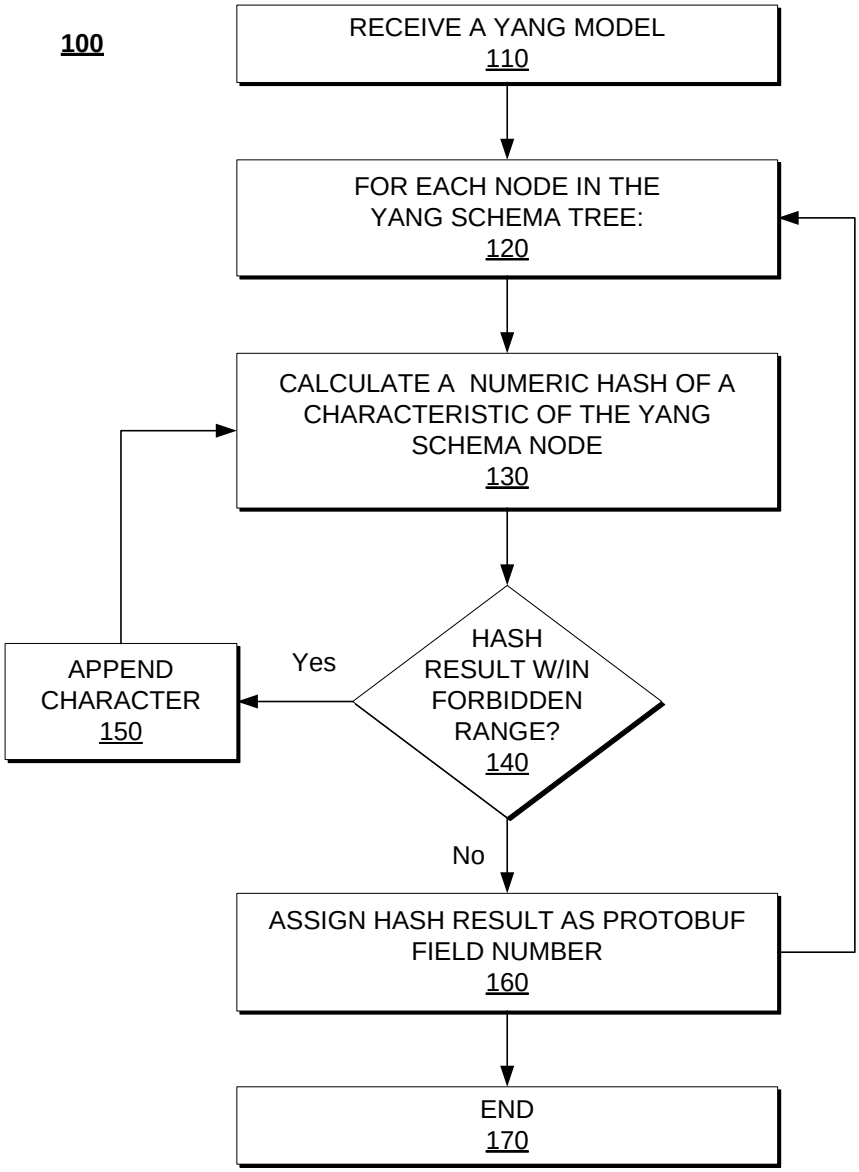
100
RECEIVE A YANG MODEL
110
FOR EACH NODE IN THE YANG SCHEMA TREE:
120
CALCULATE A NUMERIC HASH OF A CHARACTERISTIC OF THE YANG SCHEMA NODE
130
HASH RESULT W/IN FORBIDDEN RANGE?
140
Yes
APPEND CHARACTER
150
No
ASSIGN HASH RESULT AS PROTOBUF FIELD NUMBER
160
END
170

FIGURE 1

Various hash functions can be used to validate the above characteristics. For example:

- FNV hash
- A Go implementation of cityhash

as hash functions to generate 32-bit hashes of an input corpus. The generated uint32 may be converted to a valid protobuf field tag by:

- Masking the upper 3 bits to ensure the value is <= 2^{29}-1.
- In the case that the value is found to be in the range 19000-19999, then append ^ to the beginning of the input string and recompute the hash. Since ^ cannot begin a YANG identifier we know that this new string cannot be within the input corpus when considering YANG models.

Each of the characteristics required above may be validated using this algorithm by:

- Repeating the hashing function N times, ensuring that the same output was received for each input.
- Repeating the hashing function with a subset of the input, and ensured that for each hashed input, we received the same hashed value.
- Monitoring each iteration through a known input corpus to determine the number of collisions.
    - In the case where the input data set was YANG paths, we determined whether the two colliding paths fell within the same container or list entry (i.e., they have the same schema parent). In the case that they did, then this resulted in an error in the generated protobuf.

The following input datasets may be used for validation:

- EnglishWords: 466,544 English words sourced from https://github.com/dwyl/english-words, including those starting with numeric values.
- EnglishAlpha: 370,100 English words sourced from https://github.com/dwyl/english-words, including only alphabetical characters.
- OpenConfigPaths: 7,956 paths from the OpenConfig models from https://github.com/openconfig/public (using the full YANG schema path).
- CiscoPaths: 120,706 paths from the Cisco YANG models using the full YANG schema path.
- JuniperPaths: 364,645 paths from the Juniper YANG model using the full YANG schema path.
- VendorModelPaths: 12,145 paths from a realistic input corpus corresponding to a real vendor's models.

Table 1 shows example outputs for validation experiments using each corpus.

Corpus	**Test**	**Collisions**		**Collisions Causing Protobuf Failure**	
		cityhash	**fnv**	**cityhash**	**fnv**
EnglishWords	Repeatability, n=100	212	190	212	190
EnglishWords	40% subset after first iteration	31	30	31	30
EnglishAlpha	Repeatability, n=100	108	121	108	121
EnglishAlpha	40% subset after first iteration	15	16	15	16
OpenConfigPaths	Repeatability, n=100	0	0	0	0
OpenConfigPaths	40% subset after first iteration	0	0	0	0

VendorModelPaths	Repeatability, n=100	0	0	0	0
VendorModelPaths	40% subset after first iteration	0	0	0	0
CiscoPaths	Repeatability, n=100	15	10	0	0
CiscoPaths	40% subset after first iteration	0-4	0-4	0	0
JuniperPaths	Repeatability, n=100	116	126	0	0
JuniperPaths	40% subset after first iteration	12-22	16-24	0	0

Observations:

- The number of collisions for any test that involves all the corpus may be constant, with no range is observed. This suggests repeatability of both cityhash and fnv as expected.
- There may be more collisions within the English alphanumeric and words corpuses than any YANG path dataset. This may be due to the fact that the input line length is greater for the YANG paths, compared to the input for English.
- None of these datasets appear to cause a collision that is fatal in the Protobuf generation.
- The difference in hashing performance of fnv vs. cityhash may be trivial. The cityhash function may be computationally more efficient, however, given that protobuf generation is only repeated when a model changes.

Notes On Encoding Efficiency

Since protobuf field tags are stored as variants on the wire, then the idiomatic convention to start field tags at 1 and increase them monotonically in the message also results in efficiencies on the wire such that field tag 1 results in 1-byte being used to store the tag, where field tag 2^29-1 takes 5 bytes. This results in a maximum overhead of 4 bytes per field for the tag number that is auto-generated.

In OpenConfig, typical containers have less than 20 leaves within them, such that it would generally be possible to encode all tags within 1 byte, therefore if we consider the worst case that every path within the existing models is set at the same time (unlikely, given that this would mean paths that are incompatible with each other, as well as those that correspond to entirely different devices - optical ROADMs vs. Ethernet switches for example), then this would result in 7,956*4 = 31.824kB of overhead for the encoding compared to the theoretical minimum.

However, since the alternative to using protobuf encoding is using JSON encoding, then this theoretical minimum is not - in practice - a fair comparison. If instead we compare the length of field names as strings, then we find that - the total number of characters to set all fields within the schema would be 93,131, assuming that zero overhead were experienced for the encoding - such that the alternative is 93,131 * 1 bytes (for Unicode characters) = 93.131kB - such that this encoding (merely on field tag numbers) represents a significant reduction. This does not include any overhead of encoding to JSON, of which we know that there is at least six bytes more per field for the quotes around the field name, and the colon following it.

This inefficiency is worth mentioning, however, when compared to the alternative encodings, and the actual data carried in each message, it appears to be relatively insignificant, given the advantage of the overall approach.

Conclusion

Based on this analysis, it is proposed that we adopt the hashing approach described above for generating protobuf field tags from a YANG schema. To ensure adequate input data, each tag

will be assigned a value generated from using the full path to each element being assigned a tag as a hash input.

The validation against real-world YANG models, especially those with many more entities within them than OpenConfig currently has, indicates that it is unlikely that issues will be experienced in the future in the OpenConfig schema.