The Promise and Pitfalls of TLV Serialization

Brian Petersen, Ethernovia, Inc

IEEE SA Ethernet & IP @ Automotive Technology Day
São Paulo, Brazil
September 2023
.Serialization
structured data

byte string
structured data

```c
struct myStruct {
    int anInteger;
    int anotherInteger;
    char aString[80];
};
```
Semantics preservation
Architectural independence
Language independence
Message evolution
Type safety
Efficient setters/getters
On-the-wire efficiency
Remote Procedure Calls (RPCs)
Remote Procedure Calls (RPCs)

$$a = fn(b)$$
Building Blocks
Serialization Categories

Human Readable
Memory Copy
TLV Tagged
Human Readable
int8 x = 42 → “theAnswer”: 42”
Human Readable

JSON

XML
Memory Copy
Memory Copy

```
int64 x = 42 → 0x0000_0000_0000_002a
```
Memory Copy

SOME/IP
Cap’n Proto
Flat Buffers
TLV Tagged

- **Application**
- **Setters**
- **Data**
- Structured Data
- **Serializer**
- Ethernet
- **Deserializer**
- Structured Data
- **Getters**
- **Application**

- **Type**
- **ID**
- **Length**
TLV Tagged

SOME/IP (TLV)

Protocol Buffers
On-The-Wire Efficiency

Tags take space 😞

Exact-fit arrays 😊

Suppression of optional values 😊

Lightweight scalar compression 😊
Protocol Buffers Performance

```java
syntax = "proto3";

message BottomLevelMessage {
  string bottomString1 = 1;
  string bottomString2 = 2;
  repeated sint64 arrayOfCompressibleInts = 3;
}

message MidLevelMessage {
  repeated BottomLevelMessage firstBottom = 1;
  repeated string arrayOfStrings = 2;
  string midString = 3;
  bool yesOrNo = 4;
}

message TopLevelMessage {
  MidLevelMessage firstMid = 1;
  MidLevelMessage secondMid = 2;
  string topString = 3;
  repeated double arrayOfIncompressibleDoubles = 4;
}
```
Protocol Buffers Performance

AMD EPYC 7702P @ ~2 GHz
serialized length = 102 Mbytes
serialization time = 2.17 secs = 394 Mbits/s
deserialization time = 1.78 secs = 480 Mbits/s
combined rate = 216 Mbits/s
Protocol Buffers Performance

6 \times 10 \text{ Gbits/s Ethernet links}

216 \text{ Mbits/s offered load}

0.36\% \text{ network utilization}
Why So Slow?
Why So Slow?

Compression/decompression
Length determination
Variable-ID mapping
Varint Compression/Decompression

\[ x: \text{Int64} = 50,036 \ // 0xc374 \]

\[
\begin{array}{cccccccc}
00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\
00000000000000000000000000000000 & 00000000000000000000000000000000 & 0000000011000011101110100 & c3 & 74 \\
\end{array}
\]

\[ \text{moreBytes} \]

\[
\begin{array}{cccccccc}
000000111100011101110100 & 000000111100011101110100 & 11110100 & 00001100 & 00000111 \\
f4 & 86 & 03 & \text{byte transmission order: f4, 86, 03} & & & &
\end{array}
\]

\[ \text{varintX: Int24} = 16,025,091 \ // 0xf4_8603 \]
## Varint Compression/Decompression

<table>
<thead>
<tr>
<th>Signed Original</th>
<th>Encoded As</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>-2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>-3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>etc.</td>
<td>etc.</td>
</tr>
</tbody>
</table>
Length Determination
Variable-ID Mapping

```protobuf
syntax = "proto3";

message BottomLevelMessage {
  string bottomString1 = 1;
  string bottomString2 = 2;
  repeated sint64 arrayOfCompressibleInts = 3;
}

message MidLevelMessage {
  repeated BottomLevelMessage firstBottom = 1;
  repeated string arrayOfStrings = 2;
  string midString = 3;
  bool yesOrNo = 4;
}

message TopLevelMessage {
  MidLevelMessage firstMid = 1;
  MidLevelMessage secondMid = 2;
  string topString = 3;
  repeated double arrayOfIncompressibleDoubles = 4;
}
```
What Can Be Done?
What Can Be Done?
What Can Be Done?
Concluding Thoughts

Increasing structured data communications

- Fully-featured serialization

- Modern software

- Must reduce CPU overhead

- Benefit to future Ethernet networks
Thank You