Politecnico di Milano - Department of Electronics, Information and Bioengineering

POLITECNICO
MILANO 1863

BarbequeRTRM
Google Protocol Buffer based communication

[Coding Project]

Student Fabio Codiglioni
ID 919897

Course Advanced Operating Systems
Academic Year 2018-2019

Advisor Giuseppe Massari
Professor William Fornaciari

August 29, 2019

Contents

1 Introduction

1.1 Problem statement

Refactoring of the BarbequeRTRM ⟷ RTLib communication infrastructure, basing it on *Google Protocol Buffer*. Possibly, make a performance comparison with the current FIFO-based implementation.

1.2 Summary of the work

- Definition of a `proto` file holding all the *Protocol Buffer* declarations.
- Declaration of all the additional configuration options.
- Refactorization of the pre-existing code and CMake files to accomodate such additional configuration.
- Separation of the pre-existing implementation and the future one into separate directories.
- Implementation of the RTLib side of the infrastructure.
- Implementation of the Barbeque side of the infrastructure.

2 Design and implementation

2.1 Messages declaration

The pre-existing code featured a series of native C-style `structs` used to represent each type of message together with some `enums` to limit the values of some fields. The adopted approach was to translate all the message type `structs` into a single protobuf `message`, in order to limit the number of types declared[1]. This was suggested also by the specific serialization technique of the adopted version 3 of the framework: scalar fields are only explicitly serialized if their value is different from the default one, so the unset fields are not actually transmitted. A separate `message` was used to translate the message header and the other message-specific `structs`, e.g., the constraints. The original C-style enums were left, and the corresponding fields were translated into simple 32-bits unsigned integers.

Since some response types carry additional fields compared to the *standard* response type, an additional `enum` field was added to the protobuf header in order to correctly translate these responses back and forth between `structs` and protobuf classes.

2.2 Implementation

In order to speedup implementation, a *helper* class composed of only static member functions was used. This class was originally conceived to hold all the conversion code from struct to protobuf. This idea has later been dropped, because contrary to expectations it was generating a lot of repeated code. The only member functions left in the class were the ones related to the header translation, as the header is present in almost everything that is serialized and deserialized on and from the wire. The actual translation code was implemented in place, where the pre-existing code was simply writing and reading `structs`.

The message structs described in Section 2.1 are not directly transmitted on the wire in neither of the implementations. Instead, a wrapper C-like struct was used to hold the message type, the size of the wrapper struct and of the wrapped one. This approach was left also in the new implementation, as the receiving side must know how much bytes are to be read from the channel. The only difference in the new implementation is that the payload is now represented by an array of `unsigned char` instead of a `struct`, in order to accomodate the serialized protobuf data.

3 Experimental evaluation

The performance evaluation was carried out with the standard UNIX tool `time`. Table 1 contains the experimental values and their average.

[1] Note that the type of the message is still clearly identifiable by one of the header fields.

Table 1: Collection of execution times taken with the standard UNIX tool `time`. The last row contains the average.

FIFO	Protobuf
85 ms	78 ms
83 ms	82 ms
80 ms	82 ms
90 ms	85 ms
75 ms	86 ms
72 ms	90 ms
81 ms	90 ms
79 ms	81 ms
77 ms	82 ms
74 ms	76 ms
80 ms	**82 ms**

3.1 Experimental setup

3.2 Results

4 Conclusions and Future Works

References