Submission JSS 1313: Response to Reviewers' Comments

Dirk Eddelbuettel
Debian Project

Murray Stokely
Google, Inc

Jeroen Ooms
UCLA

December 15, 2014

Thank you for reviewing our manuscript, and for giving us an opportunity to rewrite, extend and and tighten both the paper and the underlying package.

We truly appreciate the comments and suggestions. Below, we have regrouped the sets of comments, and have provided detailed point-by-point replies. We hope that this satisfies the request for changes necessary to proceed with the publication of the revised and updated manuscript, along with the revised and updated package (which was recently resubmitted to CRAN as version 0.4.2).

Response to Reviewer #1

Comment 1*: Overall, I think this is a strong paper. Cross-language communication is a challenging problem, and good solutions for R are important to establish R as a well-behaved member of a data analysis pipeline. The paper is well written, and I recommend that it be accepted subject to the suggestions below.*
Reply: Thank you. We are providing a point-by-point reply below.

More big picture, less details

Comment 2*: Overall, I think the paper provides too much detail on relatively unimportant topics and not enough on the reasoning behind important design decisions. I think you could comfortably reduce the paper by 5-10 pages, referring the interested reader to the documentation for more detail.*
Reply: The paper was rewritten throughout and is now much tighter at just 23 pages.

Comment 3*: I'd recommend shrinking section 3 to 2 pages, and removing the subheadings. This section should quickly orient the reader to the RProtobuf API so they understand the big picture before learning more details in the subsequent sections. I'd recommend picking one OO style and sticking to it in this section - two is confusing.*
Reply: We followed this recommendation and reduced section 3 to about 2 1/2 pages.

Comment 3*: Section 4 dives into the details without giving a good overview and motivation. Why use S4 and not RC? How are the objects made mutable? Why do you provide both generic function and message passing OO styles? What does $ do in this context? What the heck is a pseudo-method? Spend more time on those big issues rather than describing each class in detail. Reduce class descriptions to a bulleted list giving a high-level overview, then encourage the reader to refer to the documentation for further details. Similarly, Tables 3-5 belong in the documentation, not in a vignette/paper.*
Reply: Done. TO BE EXPANDED

Comment 4*: Section 7 is weak. I think the important message is that RProtobuf is being used in practice at large scale for for large data, and is useful for communicating between R and Python. How can you make that message stronger while avoiding (for the purposes of this paper) the relatively unimportant details of the map-reduce setup?*
Reply: TBD

R to/from Protobuf translation

Comment 5: *The discussion of R to/from Protobuf could be improved. Table 9 would be much simpler if instead of Message, you provided a "vectorised" Messages class (this would also make the interface more consistent and hence the package easier to use).*
Reply: TBD

Comment 6: *Along these lines, I think it would make sense to combine sections 5 and 6 and discuss translation challenges in both direction simultaneously. At the minimum, add the equivalent for Table 9 that shows how important R classes are converted to their protobuf equivalents.*
Reply: TBD

Comment 7: *You should discuss how missing values are handled for strings and integers, and why enums are not equivalent to factors. I think you could make explicit how coercion of factors, dates, times and matrices occurs, and the implications of this on sharing data structures between programming languages. For example, how do you share date/time data between R and python using RProtoBuf?*
Reply: TBD

Comment 8: *Table 10 is dying to be a plot, and a natural companion would be to show how long it takes to serialise data frames using both RProtoBuf and R's native serialisation. Is there a performance penalty to using protobufs?*
Reply: TBD

RObjectTables magic

Comment 9: *The use of RObjectTables magic makes me uneasy. It doesn't seem like a good fit for an infrastructure package and it's not clear what advantages it has over explicitly loading a protobuf definition into an object.*
Reply: TBD

Comment 10: *Using global state makes understanding code much harder. In Table 1, it's not obvious where `tutorial.Person` comes from. Is it loaded by default by RProtobuf? This need some explanation. In Section 7, what does `readProtoFiles()` do? Why does `RProtobuf` need to be attached as well as `HistogramTools`? This needs more explanation, and a comment on the implications of this approach on CRAN packages and namespaces.*
Reply: TBD

Comment 11: *I'd prefer you eliminate this magic from the magic, but failing that, you need a good explanation of why.*
Reply: TBD

Code comments

Comment 12: *Using `file.create()` to determine the absolute path seems like a bad idea.*
Reply: TBD

Minor niggles

Comment 13: *Don't refer to the message passing style of OO as traditional.*
Reply: TBD

Comment 14: *In Section 3.4, if messages isn't a vectorised class, the default print method should use* `cat()` *to eliminate the confusing* `[1]`.
Reply: TBD

Comment 15: *The REXP definition would have been better defined using an enum that matches R's SEXP-TYPE "enum". But I guess that ship has sailed.*
Reply: TBD

Comment 16: *Why does* `serialize_pb(CO2, NULL)` *fail silently? Shouldn't it at least warn that the serialization is partial?*
Reply: TBD

Response to Reviewer #2

Comment 1: *The paper gives an overview of the RProtoBuf package which implements an R interface to the Protocol Buffers library for an efficient serialization of objects. The paper is well written and easy to read. Introductory code is clear and the package provides objects to play with immediately without the need to jump through hoops, making it appealing. The software implementation is executed well.*
Reply: Thank you.

Comment 2: *There are, however, a few inconsistencies in the implementation and some issues with specific sections in the paper. In the following both issues will be addressed sequentially by their occurrence in the paper.*
Reply: TBD

Comment 3: *p.4 illustrates the use of messages. The class implements list-like access via* `[[` *and* `$`*, but strangely* `names()` *return NULL and* `length()` *doesn't correspond to the number of fields leading to startling results like the following:*

```
 > p
[1] "message of type 'tutorial.Person' with 2 fields set"
 > length(p)
[1] 2
 > p[[3]]
[1] ""
```

Reply: TBD

Comment 3 cont.: *The inconsistencies get even more bizarre with descriptors (p.9):*

```
 > tutorial.Person$email
[1] "descriptor for field 'email' of type 'tutorial.Person' "
 > tutorial.Person[["email"]]
Error in tutorial.Person[["email"]] : this S4 class is not subsettable
 > names(tutorial.Person)
NULL
 > length(tutorial.Person)
[1] 1
```

Reply: TBD

Comment 3 cont.: *It appears that there is no way to find out the fields of a descriptor directly (although the low-level object methods seem to be exposed as `$field_count()` and `$fields()` - but that seems extremely cumbersome). Again, implementing names() and subsetting may help here.*
Reply: TBD

Comment 4: *Another inconsistency concerns the `as.list()` method which by design coerces objects to lists (see `?as.list`), but the implementation for EnumDescriptor breaks that contract and returns a vector instead:*

```
 > is.list(as.list(tutorial.Person$PhoneType))
[1] FALSE
 > str(as.list(tutorial.Person$PhoneType))
  Named int [1:3] 0 1 2
  - attr(*, "names")= chr [1:3] "MOBILE" "HOME" "WORK"
```

Comment 4 cont: *As with the other interfaces, names() returns NULL so it is again quite difficult to perform even simple operations such as finding out the values. It may be natural use some of the standard methods like names(), levels() or similar. As with the previous cases, the lack of [[support makes it impossible to map named enum values to codes and vice-versa.*
Reply: TBD

Comment 5: *In general, the package would benefit from one pass of checks to assess the consistency of the API. Since the authors intend direct interaction with the objects via basic standard R methods, the classes should behave consistently.*
Reply: TBD

Comment 6: *Finally, most classes implement coercion to characters, which is not mentioned and is not quite intuitive for some objects. For example, one may think that as.character() on a file descriptor returns let's say the filename, but we get:*

```
 > cat(as.character(tutorial.Person$fileDescriptor()))
syntax = "proto2";

package tutorial;

option java_package = "com.example.tutorial";
option java_outer_classname = "AddressBookProtos";
[...]
```

Reply: TBD

Comment 7: *It is not necessary clear what java_package has to do with a file descriptor in R. Depending on the intention here, it may be useful to explain this feature.*
Reply: TBD

Other comments:

Comment 8: *p.17: "does not support ... function, language or environment. Such objects have no native equivalent type in Protocol Buffers, and have little meaning outside the context or R" That is certainly false. Native mirror of environments are hash tables - a very useful type indeed. Language objects are just lists, so there is no reason to not include them - they can be useful to store expressions that may not be necessary specific to R. Further on p. 18 your run into the same problem that could be fixed so easily.*
Reply: TBD

Comment 9: *The examples in sections 7 and 8 are somewhat weak. It does not seem clear why one would wish to unleash the power of PB just to transfer breaks and counts for plotting - even a simple ASCII file would do that just fine. The main point in the example is presumably that there are code generation methods for Hadoop based on PB IDL such that Hadoop can be made aware of the data types, thus making a histogram a proper record that won't be split, can be combined etc. – yet that is not mentioned nor a way presented how that can be leveraged in practice. The Python example code simply uses a static example with constants to simulate the output of a reducer so it doesn't illustrate the point - the reader is left confused why something as trivial would require PB while a savvy reader is not able to replicate the illustrated process. Possibly explaining the benefits and providing more details on how one would write such a job would make it much more relevant.*
Reply: TBD

Comment 10: *Section 8 is not very well motivated. It is much easier to use other formats for HTTP exchange - JSON is probably the most popular, but even CSV works in simple settings. PB is a much less common standard. The main advantage of PB is the performance over the alternatives, but HTTP services are not necessarily known for their high-throughput so why one would sacrifice interoperability by using PB (they are still more hassle and require special installations)? It would be useful if the reason could be made explicit here or a better example chosen.*
Reply: TBD