New compact binary protocol

Description

There's been numerous discussion on how to implement a new more compact binary protocol. The discussions become hard to follow after a while so this page is intended to be used as an easy to use summary that can later be formalized into different options and finally become a specification. Help needed to fill this page with further details, suggestions and pros/cons for each suggestion.

Implementation suggestions

Encode i32 and i64 types saved as variable size integers

Suggestion	Pros	Cons
ZIP encoding (variable length encoding) for only positive values	save a max of 3 bytes for small ints	user has to specify the new type
Base 128 + zigzag, borrow from protocol buffers?		user has to specify whether zigzag needs to be used for efficiency

As the user knows best about his data he can choose whichever he wants and save bytes. This means we need more type modifiers for these types

Remove / reduce the size of field prefix tags

Suggestion	Pros	Cons
Reduce from 3 bytes per field to 1 byte, see mail	Retains versioning support	Only good for dense structs Breaks down if type modifiers/hints need to go into type field
1-byte type-and-modifier, variable length int for field id		
Drop field prefix altogether	saves tons of space	no versioning is possible
Use a per-struct variable length bitset to specify which all fields present . Preserve type info	Saves 1 bit/field and adds 1 byte/ 7 fields	Bad for sparse objects Implies fields must be ordered by id in encoding

Type changes

Suggestion	Pros	Cons
ZIP encoding (variable length encoding) for only positive values	save a max of 3 bytes for small ints	user has to specify the new type
Unsigned integers	Would alleviate need for separate zigzag type	Unsigned ints don't exist in all languages
Type annotations	Allows us to specify encoding details about the fields/types that the protocols may or may not use	
Variable ints for string, binary, and collection sizes	Will often shrink to one or two bytes	
Have two types BOOLEAN_TRUE and BOOLEAN_FALSE instead of type and value	Save a byte on every boolean	

Better usage of type byte

If we spent one whole byte for type it is quite a waste considering we have ~15 types . That is a wastage of almost 4 bits on EACH field. Let us have two types of types. One with extra information and one which does not . Let us take the 5 least significant bit (LSB) to represent them. Let us make use of the 3 most significant bits (MSB) for types with extra information

7	6	5	4	3	2	1	0

The 5 LSB (green) could be used for these types

- VOID
- STOP
- BOOLEAN_TRUE
- BOOLEAN_FALSE
- DOUBLE
- I16
- I32
- I64

The 3 MSB (red) can be used for a max 7 types. The 5 MSB can be used in these types for length ,value etc (depending on the type)

- STRING
- SET
- LIST
- MAP
- POSITIVE_I32
- STRUCT
- EXTERN_STRING

Information sources

2008 jan mail thread

jira ticket

TDenseProtocol