Using libRaptorQ library

December 11, 2017

Abstract

libRaptorQ is a C++11 implementation of the RaptorQ Forward Error Correction, as described in the RFC6330 .

The implementation was started as a university laboratory project, and will be later used and included in Fenrir, the maintainer's master thesis.

This implementation is quite short (the core is $\sim 3k$ lines), thanks to the chosen language and the use of external libraries for matrix handling (eigen3).

libRaptorQ is the only RaptorQ implementation in C++, include C hooks, and it is the only free (**LGPL3**) implementation of the rfc, except for the (apache2) java implementation, OpenRQ , which is much bigger ($\sim 46k$) and slower.

Contents

1 Contacts

The main development and dicussions on the project, along with bug reporting, happens on the main website.

Mailing Lists: Mailing lists are available at https://www.fenrirproject.org/lists The two mailing lists are for development and announcements, but due to the low traffic of the development mailing list, it can also be used by users for questions on the project.

IRC: Since there are not many developers for now, the main irc channel is **#fenrirproject** on freenode

2 Build & install

2.1 Get the source code

Although things seems to work, no stable release has been released yet, as of December 11, 2017.

This means you can only check this out with git.

To check out the repository:

```
$ git clone https://github.com/LucaFulchir/libRaptorQ.git
```

You can also get it from our main server:

```
$ git clone https://www.fenrirproject.org/Luker/libRaptorQ.git
```

GPG verification: Once you have cloned it, it's always a good thing to check the repository gpg signatures, so you can import my key with:

```
$ gpg --keyserver pgp.mit.edu --recv-key 7393DAD255BEB5751DBDA04AB11
```

Now you have the source, and the key, it's enough to check the signature of the last commit:

```
$ git log -n 1 --show-signature
```

The important part is that you get something like this:

```
gpg: Signature made Mon 11 Dec 2017 21:55:28 CET
gpg:                using RSA key 8F7474044095B405D0F042F0A2CCA134BC7C8572
   gpg: Good signature from "Luca Fulchir <luker@fenrirproject.org>" [unknown]
gpg:                 aka "Luca Fulchir <luca@fulchir.it>" [unknown]
 gpg: WARNING: This key is not certified with a trusted signature!
 gpg:          There is no indication that the signature belongs to the owner.
 Primary key fingerprint: 7393 DAD2 55BE B575 1DBD A04A B11C D823 BA27 8C85
```

And as long as you got the right key, and you find the **"gpg: Good signature"** string, you can be sure you have the right code.

2.2 Dependencies

libRaptorQ has 3 dependencies:

Eigen3 : This is used for matrix manipulation, which is a big part of RaptorQ.

lz4 : Used to compress cached precomputations.

git : This is used not only to get the source, but also by the build system. We get the last git commit id and feed it to clang or gcc as seed for their internal random number generator. This makes it possible to have reproducible builds.

All dependencies are included in the sources, so you do not need to download and compile them.
Eigen3 is only needed at build time, so there is no need to download it again. we use the 3.2.8 version.
LZ4 is included as a git submodule. if you do not have it, run:

```
$ git submodule init
$ git submodule update
```

and LZ4 will be build statically in the library (it will not be installed on the system)

2.3 Build & Install

libRaptorQ uses the cMake build system, so things are fairly standard:

```
$ cd libRaptorQ.git
$ mkdir build
$ cmake ../
$ make -j 4
```

By default, the libRaptorQ project tries to have deterministic builds. This means that if you compile things twice, or with two different computers, the hash of the resulting library will be the same, provided that the same compiler (clang, gcc 4.8, gcc 4.9 etc) was used. Currently the only exception is the clang compiler with the *PROFILING* option enabled, and this will not likely be solved.

There are lots of options, you can use in cmake. As always, you can change them by adding "**-Dcmake_option=cmake_value**" when calling cmake.

You can always use the cmake-gui or ccmake commands to have the list of possible options.

The ones we recognize are:

LTO : ON/OFF. Default:**ON**. Enables *Link Time Optimizatios* for clang or gcc. Makes libraries smaller and better optimized.

PROFILING : ON/OFF. Default:**ON**. Profiling compiles everything once, then runs a test to see which code paths are used more, and then recompiles everything again, but making sure that the binary is optimized for those paths. Works with clang and gcc. Provides a slight speedup.

CMAKE_C_COMPILER : gcc and clang are directly supported. other should work, too. This is only used if you want to build the C example.

CMAKE_CXX_COMPILER : g++, clang++ are directly supported. other should work, too.

CLANG_STDLIB : use clang's libc++ standard library. Only available with clang.

CMAKE_BUILD_TYPE : Type of build. you can choose between "Debug", "Release", "MinSizeRel" or "RelWithDebInfo"

CMAKE_INSTALL_PREFIX : Default to */usr/local*. Change it to fit your distribution guidelines.

Then you can build everything by running:

```
$ make -j 4
```

Of course, you can configure the number of parallel jobs (the *-j* parameter) to be what you need.

Optional make targets: The following optional targets are also supported:

```
$ make docs tests examples
$ make everything
```

The "docs" target builds this document, but you need latex with the refman style. The tests are only useful to check perfromance of rfc compliance right now. "examples" compiles the C and C++ examples, which will not be installed.

Install: The installation process is very simple:

```
$ make install DESTDIR=...
```

You can change the *DESTDIR* parameter to fit your distribution guidelines.

3 Working with RaptorQ

3.1 Theory (you really need this)

To be able to work with liRaptorQ, you must first understand how the RaptorQ algorithms works. We won't go into the details, but just what you need to be able to tune the algorithm to your needs.

Fountain codes: Fountain codes are a special *Forward-Error-Correcting* code class, which characteristic is simple: if you want to send K packets, you actually send $K+X$ packets, and the receiver only needs to get any K packets to be able to reconstruct your data. The number X of overhead packets can be as big as you need (theoretically infinite), so you can tune it to be slightly higher than the expected packet loss.

Systematic codes: RaptorQ is also a systematic code. This means that those first K packets are the input *as-is* (**source symbols**), and the X packets (**repair symbols**) have the information needed to recover any of the lost source packets. This has the big advantage of avoiding any kind of time and memory consuming decoding if there is no packet loss during the transmission.

Complexity: The RaptorQ algorithm is often presented as having a linear time encoder and decoder. This is both false and misleading. Generating the source or repair symbols from the intermediate symbols has linear complexity. Generating the intermediate symbols has cubic complexity on the number of symbols. Which is a completely different thing. It is still very quick. On a core i7, 2.4Ghz, you need to wait *0.4ms* for 10 symbols, *280ms* for 1.000 symbols, but it can take an hour for 27.000 symbols. RaptorQ handles up to 56.403 symbols.

Caching libRaptorQ can work with big matrices that take a lot of time to compute. For this reason the matrices can be saved once they have been computed the first time. libRaptorQ uses a **shared** and a **local** cache. In both cases the matrix is compressed with LZ4.
Shared cache uses shared memory and is thus shared with all processes that use the save version of libRaptorQ. Local cache is only local to the same process.

3.2 Blocks & Symbols

To understand how to properly use and tune libRaptorQ, you first need to understand how RaptorQ handles its inputs, outputs, and what the time and memory constraints are.

Input sequencing: RaptorQ needs to have the whole input you want to send before it can start working.
This means that it might be a little more difficult to use in live-streaming contexts, or where you need real-time data, but libRaptorQ will have options to facilitate usage even in those contexts.

Once you have the whole input, RaptorQ divides it into **blocks**. Each block *is encoded and decoded independently* and will be divided into **symbols**. Each symbol *should* be transmitted separately in its own packet (if you are on the network).

Sizes: Each input can have up to *256 blocks*, each block can have up to 56.403 *symbols*, and each symbol can have up to $2^{16} - 1$ *bytes*long. This gives a maximum files size of almost 881 GB (946270874880 bytes to be exact)

Interleaving: An other feature of RaptorQ is to automatically provide some interleaving of the input data before transmitting it. This means that one symbol will not represent one sequential chunk of your input data, but more likely it's the concatenation of different **subsymbols**. The size of the subsymbol must thus be a fraction of the symbol size. This feature is not used if you set the size of the subsymbol to the size of symbol.

Memory and Time: Memory and time requirements are to be considered, though, as RaptorQ needs to run a cubic algorithm on matrix of size $K * K$, where K is the number of symbols in each block.
The algorithm needs to keep in memory two of these matrices, although most of the work is done on only one.
This is actually a lot. More benchmarks and optimizations will come later, for now remember that with 10 symbols it takes something like 0.4ms on a core i7 2.4GHZ, 280ms with 1000 symbols, and up to an hour with 27.000 symbols.

3.3 C++ interface

To use the C++ interface you only need to include the **RaptorQ.hpp** header, and link **libRaptorQ** and your threading library (usually *libpthread*).

To provide grater flexibility, the whole library uses iterators to read your data, and to write the data onto your data structures.
This means that a big part of the library is a template, which adapts to the alignment of the data in the data structures you use.

Templates There are two main classes you will use:

```
template <typename Rnd_It, typename Fwd_It>
class Encoder

template <typename In_It, typename Fwd_It>
class Decoder
```

As you might guess, the classes for the encoder and decoder take two template parameters.
For the **Encoder**, the first parameter *MUST* be a *random access iterator* to the input data, and the second parameter is an *forward iterator*. The random access iterator will be used to scan your input

data, and perform an interleaving (if you did not set the same size the symbol and to the subsymbol). The forward iterator will be used to write the data to your structure.
The same is done for the **Decoder**, but we do not need to do any interleaving on the input, so the first iterator can be just an input iterator, and nothing more.

3.4 Caching precomputations

libRaptorQ can now cache the most used matrices for a quicker reference.
There are two level of caching: *local* and *global*.

Global caching will cache precomputed matrices for all programs who use your version of libRaptorQ in the system. This uses shared memory to store the matrices. The shared memory is automatically increased to the maximum size any program requires, and automatically scaled down once the program is not being executed anymore.

Local caching instead works only in the program, and is not shared with any other application. Once the shared cache is checked, local caches is checked. The cache can be scaled up or down as needed dynamically.

shared_cache_size : **Input: const uint64_t shared_cache**
return: uint64_t
Set the maximum shared cache size for this process. Default: 0 (do not use shared cache)

local_cache_size : **const uint64_t local_cache**
return: bool
Set the local cache size. returns false on error.

get_shared_cache_size() : **return: uint64_t**
get the size of our shared memory requirement. *NOTE: the actual size might be larger, since other processes might have requested more*

get_local_cache_size() : **return: uint64_t**
get the size of our local cache

3.4.1 The Encoder

You can instantiate an encoder for example by doing:

```
std::vector<uint32_t> input, output;
...
using T_it = typename std::vector<uint32_t>::iterator;
RaptorQ::Encoder<T_it, T_it> enc (input.begin(),
                                  input.end(),
                                  4, 1444, 10000)
```

This will create an Encoder that works on vectors of unsigned 32 bit integers for both input and output, that will create symbols of size 1444 bytes, interleaving your input every 4 bytes, and try to work with big number of symbols per blocks (**TODO: explain memory requirements**)

The available methods for the encoder are the following:

operator bool() : **return:bool**
False if constructor parameters did not make sense. Else true.

OTI_Common() : **return: OTI_Common_Data**, aka *uint64_t*.
Keeps total **file size and symbol size**. You need to send this to the receiver, so that it will be able to properly decode the data.

OTI_Scheme_Specific_Data() : **return: OTI_Scheme_Specific_Data**, aka *uint32_t*.
Keeps number of **source blocks, sub blocks, and alignment**. As for the OTI_Common_Data, you need to send this to the receiver to be able to properly decode the data.

encode : **Input: Fwd_It &output, const Fwd_It end, const uint32_t esi, const uint8_t sbn**.
return:uint64_t.
Take as input the iterators to the data structure into where we have to save the encoded data, the **Encoding Symbol Id** and the **Source Block Number**. As you are writing in C++, you probably want to use the iterators begin/end, though. Returns the number of written iterators (**NOT** the bytes)

encode : **Input: Fwd_It &output, const Fwd_It end, const uint32_t id**.
return:uint64_t.
Exactly as before, but the **id** contains both the *source block number* and the *encoding symbol id*

begin() : **return: Block_Iterator¡Rnd_It, Fwd_It¿**
This returns an iterator to the blocks in which RaptorQ divided the input data. See later to understand how to use it.

end() : **return: const Block_Iterator¡Rnd_It, Fwd_It¿**
This returns an iterator to the end of the blocks in which RaptorQ divided the input data. See later to understand how to use it.

precompute : **Input:const uint8_t threads, const bool background**
return: void
Do the work of computing all different blocks in multithread.

If *background* is true, then return immediately, else return only when the job is done.
If *threads* is 0, try to guess the maximum threads from the number of available cpus.

precompute_max_memory : **return: size_t**
Each precomputation can take a lot of memory, depending on the configuration, so you might want to limit the number of precomputations run in parallel depending on the memory used. This method returns the amount of memory taken by **ONE** precomputation.

free : **Input: const uint8_t sbn**
return: void
Each block takes some memory, (a bit more than $symbols * symbol_size$), so once you are done sending source and repair symbols for one block, you might want to free the memory of that block.

blocks() : **return: uint8_t** The number of blocks.

block_size() : **Input: const uint8_t sbn**
return: uint32_t
The block size, in bytes. Each block can have different symbols and thus different size.

symbol_size() : **return: uint16_t** The size of a symbol.

symbols : **Input:uint8_t sbn**
return: uint16_t
The number of symbols in a specific block. different blocks can have different symbols.

max_repair : **Input: const uint8_t sbn)**
return: uint32_t
The maximum amount of repair symbols that you can generate. Something less than 2^{24}, but the exact number depends on the number of symbols in a block

3.4.2 Blocks

With the *begin()/end()* calls you get *Input iterators* to the blocks. a Block is has the following type:

```
template <typename Rnd_It, typename Fwd_It>
class Block
```

and exposes the 4 following methods:

begin_source : **return: Symbol_Iterator** end_source : **return: Symbol_Iterator**

begin_repair : **return: Symbol_Iterator**

end_repair : **Input: const uint32_t max_repair**
return: Symbol_Iterator

max_repair : **return:uint32_t**

symbols : **return: uint16_t**

block_size : **return: uint32_t**

As the names explain, you will get an iterator to the symbols in the block. As the number of repair symbols can vary, for now you get two separate begin/ends, so that you can check when you sent the source symbols, and how many repair symbols you send.
The other functions are helpers for the details of the block.

3.4.3 Symbols

Finally, through the *Symbol_Iterator Input Iterator* we get the **Symbol** class:

```
template <typename Rnd_It, typename Fwd_It>
class Symbol
```

which exposes the 2 methods we need to get the symbol data:

operator* : **Input:Fwd_It &start, const Fwd_It end**
return: uint64_t
takes a forward iterator, and fill it with the symbol data. returns the number of written iterators.

id() : **return: uint32_t**
return the id ($sbn + esi$) of this symbol, that you need to include in every packet you send, before the symbols.

3.4.4 The Decoder

The decoder is a bit simpler than the encoder.

Theere are two constructors for the Decoder:

```
std::vector<uint32_t> input, output;
using T_it = typename std::vector<uint32_t>::iterator;
RaptorQ::Decoder<T_it, T_it> dec (
                    const OTI_Common_Data common,
                    const OTI_Scheme_Specific_Data scheme)

RaptorQ::Decoder<T_it, T_it> dec (uint64_t size,
                                  uint16_t symbol_size,
                                  uint16_t sub_blocks,
                                  uint8_t blocks)
```

Which should be pretty self-explanatory, once you understand how the encoder works.

The remaining methods are:

decode : **Input: Fwd_It &start, const Fwd_It end**
return:uint64_t
Write **all** the blocks into the iterator. refuses to write if the input has not been completely received. Return the number of iterators written.

decode : **Input: Fwd_It &start, const Fwd_It end, const uint8_t sbn**
return:uint64_t
Write a specific block into the iterator. Refuses to write if the input for that block has not been completely received.

add_symbol : **In_It &start, const In_It end, const uint32_t esi, const uint8_t sbn**
return: bool
Add one symbol, while explicitly specifying the symbol id and the block id.

add_symbol : **In_It &start, const In_It end, const uint32_t id**
return: bool
Same as before, but extract the block id and the symbol id from the *id* parameter

free : **Input: const uint8_t sbn**
return: void
You might have stopped using a block, but the memory is still there. free it.

blocks() : **return: uint8_t** The number of blocks.

block_size() : **Input: const uint8_t sbn**
return: uint32_t
The block size, in bytes. Each block can have different symbols and thus different size.

symbol_size() : **return: uint16_t** The size of a symbol.

symbols : **Input:uint8_t sbn**
return: uint16_t
The number of symbols in a specific block. different blocks can have different symbols.

3.5 C interface

The C interface looks a lot like the C++ one.
You need to include the **cRaptorQ.h** header, and link the *libRaptorQ* library.

Static linking

If you are working with the static version of libRaptorQ remember to link the C++ standard library used when compiling the library (*libstdc++* for gcc or maybe *libc++* for clang), your threding library (usually *libpthread*), and the C math library (*libm*).

As for the C++11 interface, you can get and set the shared and local memory cache.
Shared cache uses shared memory, while local cache can be used only be the current process.

Cache settings

```
uint64_t RaptorQ_shared_cache_size(
                                    const uint64_t shared_cache);
bool RaptorQ_local_cache_size (
                                    const uint64_t local_cache);
uint64_t RaptorQ_get_shared_cache_size ();
uint64_t RaptorQ_get_local_cache_size ();
```

To use libRaptorQ you need to build the encoder or the decoder.

The C interface is just a wrapper around the C++ code, so you still have to specify the same things as before. A quick glance at the constructors should give you all the information you need:

C Constructors

```
typedef enum { RQ_NONE = 0, RQ_ENC_8 = 1, RQ_ENC_16 = 2,
               RQ_ENC_32 = 3, RQ_ENC_64 = 4, RQ_DEC_8 = 5,
               RQ_DEC_16 = 6, RQ_DEC_32 = 7, RQ_DEC_64 = 8}
                 RaptorQ_type;

struct RaptorQ_ptr
{
  void *ptr = nullptr;
  const RaptorQ_type type;
};

RaptorQ_ptr* RaptorQ_Enc (const RaptorQ_type type,
                          void *data,
                          const uint64_t size,
                          const uint16_t min_subsymbol_size,
                          const uint16_t symbol_size,
                          const size_t max_memory);

RaptorQ_ptr* RaptorQ_Dec (const RaptorQ_type type,
                   const RaptorQ_OTI_Common_Data common,
                   const RaptorQ_OTI_Scheme_Specific_Data
                                                   scheme);
```

The encoder and decoder must have a specific alignment. in C++ you can also have different alignments for the input and output, while in C things are a bit more strict as we have to enumerate all the possible cases. So you only get the same data alignment for both input and output.
Still, you don't lose anything in performance.

3.5.1 Common functions for (de/en)coding

These functions are used by both the decoder and the decoder, and will be helpful in tracking how much memory you will need to allocate, or in general in managing the encoder and decoder.

The **ptr** must be a valid encoder or decoder. The names for now are self-explanatory. Blocks can have different symbols, so the size of a block and the number of symbols in a block depend on which block we are talking about.

symbols, blocks, memory

```
uint16_t RaptorQ_symbol_size (struct RaptorQ_ptr *ptr);
uint8_t RaptorQ_blocks (struct RaptorQ_ptr *ptr);
uint32_t RaptorQ_block_size (struct RaptorQ_ptr *ptr,
                                     const uint8_t sbn);
uint16_t RaptorQ_symbols (struct RaptorQ_ptr *ptr,
                                     const uint8_t sbn);

void RaptorQ_free (struct RaptorQ_ptr **ptr);
void RaptorQ_free_block (struct RaptorQ_ptr *ptr,
                                     const uint8_t sbn);
```

Finally, when you are done working with a block, you can free the memory associated with the single block and just free the whole (en/de)coder when you are done. Freeing the whole (en/de)coder will obviously free also all the blocks.

3.5.2 Encoding

OTI Data

First, we need to tell the receiver all the parameters that the encoder is using, and for that two functions are provided:

```
typedef uint64_t RaptorQ_OTI_Common_Data;
typedef uint32_t RaptorQ_OTI_Scheme_Specific_Data;

RaptorQ_OTI_Common_Data RaptorQ_OTI_Common (
                                 struct RaptorQ_ptr *enc);
RaptorQ_OTI_Scheme_Specific_Data RaptorQ_OTI_Scheme (
                                 struct RaptorQ_ptr *enc);
```

Encoding

```
// maximum number of repair symbol in a block
uint32_t RaptorQ_max_repair (RaptorQ_ptr *enc,
                                        const uint8_t sbn);
//estimate bytes of ram used in the precomputation
// of one block
size_t RaptorQ_precompute_max_memory (
                                    struct RaptorQ_ptr *enc);
// do the precomputation.
void RaptorQ_precompute (struct RaptorQ_ptr *enc,
                                      const uint8_t threads,
                                      const bool background);

// encode one symbol. source block number and
// symbol id are in the ``id'' field.
// returns number of alignments written.
uint64_t RaptorQ_encode_id (struct RaptorQ_ptr *enc,
                                       void **data,
                                       const uint64_t size,
                                       const uint32_t id);
// encode one symbol. same as before
uint64_t RaptorQ_encode (struct RaptorQ_ptr *enc,
                                       void **data,
                                       const uint64_t size,
                                       const uint32_t esi,
                                       const uint8_t sbn);
// build an ``id'' field out of an esi and sbn field.
uint32_t RaptorQ_id (const uint32_t esi,
                                       const uint8_t sbn);
```

As for the C++ version, everything is thread-safe.

You can start the precomputation in background and not worry about it.
If you request repair symbols before the computation is finished, the call will block until the data is available.

The **encode** functions are the same, and will encode **one** symbol. They work for both source symbols and repair symbols, just keep increasing the *esi* field

3.5.3 Decoding

Decoding

```
// return the total size of the data that will be
// decoded
uint64_t RaptorQ_bytes (struct RaptorQ_ptr *dec);

// decode all blocks.
// returns the number of written alignments.
// returns 0 if decoding of *everything* is not possible
uint64_t RaptorQ_decode (struct RaptorQ_ptr *dec,
                                    void **data,
                                    const size_t size);
// decode only one block (if possible)
// returns 0 if decoding of the block is not possible.
uint64_t RaptorQ_decode_block (struct RaptorQ_ptr *dec,
                                    void **data,
                                    const size_t size,
                                    const uint8_t sbn);

// add a received symbol to the structure.
// either by using the 'id' field
bool RaptorQ_add_symbol_id (struct RaptorQ_ptr *dec,
                                    void **data,
                                    const uint32_t size,
                                    const uint32_t id);
// or by explicitely declaring esi and sbn
bool RaptorQ_add_symbol (struct RaptorQ_ptr *dec,
                                    void **data,
                                    const uint32_t size,
                                    const uint32_t esi,
                                    const uint8_t sbn);
```

4 GNU Free Documentation License

GNU Free Documentation License
Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the

Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover

Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified

Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this

License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
    Copyright (c)  YEAR  YOUR NAME.
    Permission is granted to copy, distribute and/or modify this document
    under the terms of the GNU Free Documentation License, Version 1.3
    or any later version published by the Free Software Foundation;
    with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
    A copy of the license is included in the section entitled "GNU
    Free Documentation License".
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
    with the Invariant Sections being LIST THEIR TITLES, with the
    Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Index