Understanding Tensorflow 2 source code

D. Gueorguiev 4/27/20

Contents

[Tensorflow Internal structures , classes and interfaces 2](#_Toc40648886)

[C++ External and Internal API 2](#_Toc40648887)

[TF\_CAPI\_EXPORT directive 3](#_Toc40648888)

[TF\_VERSION string 3](#_Toc40648889)

[TF\_Buffer and functionality for manipulating it 3](#_Toc40648890)

[Global functions for manipulation of TF\_SessionOptions 5](#_Toc40648891)

[Classes Graph and GraphDef 6](#_Toc40648892)

[Protobuf interfaces and formats 7](#_Toc40648893)

[Core/Protobuf/Config.proto 7](#_Toc40648894)

[Exploring the C++ code examples 8](#_Toc40648895)

[Exploring //tensorflow/core/example 8](#_Toc40648896)

[Proto file for ***Feature*** 8](#_Toc40648897)

[Proto file for ***Example***: 10](#_Toc40648898)

[Building the tensorflow libraries and examples using Bazel 12](#_Toc40648899)

[Building tensorflow libraries 12](#_Toc40648900)

[Building tensorflow core components 12](#_Toc40648901)

[Building Example Code 12](#_Toc40648902)

[Build Folder structure 12](#_Toc40648903)

[Build first C++ Tensorflow app 14](#_Toc40648904)

[Third Party and External Packages 14](#_Toc40648905)

[**BoringSSL:** https://boringssl.googlesource.com/boringssl/ 14](#_Toc40648906)

[**FarmHash:** https://github.com/google/farmhash 15](#_Toc40648907)

[*Introducing FarmHash* 15](#_Toc40648908)

[**HighwayHash**: https://github.com/google/highwayhash 15](#_Toc40648909)

[Appendix A: Bazel tutorial for Tensorflow Builds 15](#_Toc40648910)

[Appendix B: Hashing in Tensorflow 15](#_Toc40648911)

[Bibliography 18](#_Toc40648912)

# Tensorflow Internal structures , classes and interfaces

## C++ External and Internal API

The files which define the C++ are:

[bazel-tensorflow/tensorflow/c/c\_api.h](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/c/c_api.h)

// --------------------------------------------------------------------------

// C API for TensorFlow.

//

// The API leans towards simplicity and uniformity instead of convenience

// since most usage will be by language specific wrappers.

//

// Conventions:

// \* We use the prefix TF\_ for everything in the API.

// \* Objects are always passed around as pointers to opaque structs

// and these structs are allocated/deallocated via the API.

// \* TF\_Status holds error information. It is an object type

// and therefore is passed around as a pointer to an opaque

// struct as mentioned above.

// \* Every call that has a TF\_Status\* argument clears it on success

// and fills it with error info on failure.

// \* unsigned char is used for booleans (instead of the 'bool' type).

// In C++ bool is a keyword while in C99 bool is a macro defined

// in stdbool.h. It is possible for the two to be inconsistent.

// For example, neither the C99 nor the C++11 standard force a byte

// size on the bool type, so the macro defined in stdbool.h could

// be inconsistent with the bool keyword in C++. Thus, the use

// of stdbool.h is avoided and unsigned char is used instead.

// \* size\_t is used to represent byte sizes of objects that are

// materialized in the address space of the calling process.

// \* int is used as an index into arrays.

// \* Deletion functions are safe to call on nullptr.

//

// Questions left to address:

// \* Might at some point need a way for callers to provide their own Env.

// \* Maybe add TF\_TensorShape that encapsulates dimension info.

//

// Design decisions made:

// \* Backing store for tensor memory has an associated deallocation

// function. This deallocation function will point to client code

// for tensors populated by the client. So the client can do things

// like shadowing a numpy array.

// \* We do not provide TF\_OK since it is not strictly necessary and we

// are not optimizing for convenience.

// \* We make assumption that one session has one graph. This should be

// fine since we have the ability to run sub-graphs.

// \* We could allow NULL for some arguments (e.g., NULL options arg).

// However since convenience is not a primary goal, we don't do this.

// \* Devices are not in this API. Instead, they are created/used internally

// and the API just provides high level controls over the number of

// devices of each type.

For Linux and MacOS the symbol TF\_CAPI\_EXPORT is defined as:

### TF\_CAPI\_EXPORT directive

#define TF\_CAPI\_EXPORT \_\_attribute\_\_((visibility("default")))

GCC-specific details on the visibility attribute can be found [here](https://gcc.gnu.org/wiki/Visibility). Basically, by judicious use of the visibility attribute we can decrease dramatically the load times of dynamically shared objects i.e. tensorflow library libtensorflow\_cc.so. Great [article](#UlrichDrepperSharedLibrary) about writing shared libraries by Ulrich Drepper can be found [here](https://akkadia.org/drepper/dsohowto.pdf).

### TF\_VERSION string

The first member of the API is the TF\_VERSION string :

// --------------------------------------------------------------------------

// TF\_Version returns a string describing version information of the

// TensorFlow library. TensorFlow using semantic versioning.

TF\_CAPI\_EXPORT extern const char\* TF\_Version(void);

TensorFlow follows Semantic Versioning 2.0 ([semver](https://semver.org/spec/v2.0.0.html)) for its public API. Each release version of TensorFlow has the form MAJOR.MINOR.PATCH. For example, TensorFlow version 1.2.3 has MAJOR version 1, MINOR version 2, and PATCH version 3. Changes to each number have the following meaning:

MAJOR: Potentially backwards incompatible changes. Code and data that worked with a previous major release will not necessarily work with the new release. However, in some cases existing TensorFlow graphs and checkpoints may be migratable to the newer release; see Compatibility of graphs and checkpoints for details on data compatibility.

MINOR: Backwards compatible features, speed improvements, etc. Code and data that worked with a previous minor release and which depends only on the non-experimental public API will continue to work unchanged. For details on what is and is not the public API, see What is covered.

PATCH: Backwards compatible bug fixes.

For example, release 1.0.0 introduced backwards incompatible changes from release 0.12.1. However, release 1.1.1 was backwards compatible with release 1.0.0.

### TF\_Buffer struct and functionality for manipulating it

The struct **TF\_Buffer** is defined next. Its purpose and usage are described in the comment lines:

// --------------------------------------------------------------------------

// TF\_Buffer holds a pointer to a block of data and its associated length.

// Typically, the data consists of a serialized protocol buffer, but other data

// may also be held in a buffer.

//

// By default, TF\_Buffer itself does not do any memory management of the

// pointed-to block. If need be, users of this struct should specify how to

// deallocate the block by setting the `data\_deallocator` function pointer.

typedef struct TF\_Buffer {

const void\* data;

size\_t length;

void (\*data\_deallocator)(void\* data, size\_t length);

} TF\_Buffer;

The next member is the global function **TF\_NewBufferFromString** which instantiates a new **TF\_Buffer** from read-only protobuf instances

// Makes a copy of the input and sets an appropriate deallocator. Useful for

// passing in read-only, input protobufs.

TF\_CAPI\_EXPORT extern TF\_Buffer\* TF\_NewBufferFromString(const void\* proto, size\_t proto\_len);

Here is an implementation for the function:

TF\_Buffer\* TF\_NewBufferFromString(const void\* proto, size\_t proto\_len) {

void\* copy = tensorflow::port::Malloc(proto\_len);

memcpy(copy, proto, proto\_len);

TF\_Buffer\* buf = new TF\_Buffer;

buf->data = copy;

buf->length = proto\_len;

buf->data\_deallocator = [](void\* data, size\_t length) {

tensorflow::port::Free(data);

};

return buf;

}

Follow three more global functions for manipulation of **TF\_Buffer**:

// Useful for passing \*out\* a protobuf.

TF\_CAPI\_EXPORT extern TF\_Buffer\* TF\_NewBuffer(void);

TF\_CAPI\_EXPORT extern void TF\_DeleteBuffer(TF\_Buffer\*);

TF\_CAPI\_EXPORT extern TF\_Buffer TF\_GetBuffer(TF\_Buffer\* buffer);

TF\_Buffer\* TF\_NewBuffer() { return new TF\_Buffer{nullptr, 0, nullptr}; }

void TF\_DeleteBuffer(TF\_Buffer\* buffer) {

if (buffer == nullptr) return;

if (buffer->data\_deallocator != nullptr) {

(\*buffer->data\_deallocator)(const\_cast<void\*>(buffer->data),

buffer->length);

}

delete buffer;

}

TF\_Buffer TF\_GetBuffer(TF\_Buffer\* buffer) { return \*buffer; }

### Global functions for manipulation of TF\_SessionOptions

// --------------------------------------------------------------------------

// TF\_SessionOptions holds options that can be passed during session creation.

typedef struct TF\_SessionOptions TF\_SessionOptions;

// Return a new options object.

TF\_CAPI\_EXPORT extern TF\_SessionOptions\* TF\_NewSessionOptions(void);

// Set the target in TF\_SessionOptions.options.

// target can be empty, a single entry, or a comma separated list of entries.

// Each entry is in one of the following formats :

// "local"

// ip:port

// host:port

TF\_CAPI\_EXPORT extern void TF\_SetTarget(TF\_SessionOptions\* options,

const char\* target);

// Set the config in TF\_SessionOptions.options.

// config should be a serialized tensorflow.ConfigProto proto.

// If config was not parsed successfully as a ConfigProto, record the

// error information in \*status.

TF\_CAPI\_EXPORT extern void TF\_SetConfig(TF\_SessionOptions\* options,

const void\* proto, size\_t proto\_len,

TF\_Status\* status);

// Destroy an options object.

TF\_CAPI\_EXPORT extern void TF\_DeleteSessionOptions(TF\_SessionOptions\*);

TF\_SessionOptions\* TF\_NewSessionOptions() { return new TF\_SessionOptions; }

void TF\_DeleteSessionOptions(TF\_SessionOptions\* opt) { delete opt; }

void TF\_SetTarget(TF\_SessionOptions\* options, const char\* target) {

options->options.target = target;

}

void TF\_SetConfig(TF\_SessionOptions\* options, const void\* proto,

size\_t proto\_len, TF\_Status\* status) {

if (!options->options.config.ParseFromArray(proto, proto\_len)) {

status->status = InvalidArgument("Unparseable ConfigProto");

}

}

/// Configuration information for a Session.

struct TF\_SessionOptions {

tensorflow::SessionOptions options;

};

[tensorflow/core/public/session\_options.h](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/core/public/session_options.h#L28-L61)

struct SessionOptions {

/// The environment to use.

Env\* env;

/// \brief The TensorFlow runtime to connect to.

///

/// If 'target' is empty or unspecified, the local TensorFlow runtime

/// implementation will be used. Otherwise, the TensorFlow engine

/// defined by 'target' will be used to perform all computations.

///

/// "target" can be either a single entry or a comma separated list

/// of entries. Each entry is a resolvable address of the

/// following format:

/// local

/// ip:port

/// host:port

/// ... other system-specific formats to identify tasks and jobs ...

///

/// NOTE: at the moment 'local' maps to an in-process service-based

/// runtime.

///

/// Upon creation, a single session affines itself to one of the

/// remote processes, with possible load balancing choices when the

/// "target" resolves to a list of possible processes.

///

/// If the session disconnects from the remote process during its

/// lifetime, session calls may fail immediately.

std::string target;

/// Configuration options.

ConfigProto config;

SessionOptions();

};

bazel-tensorflow/tensorflow/c/c\_api.cc

bazel-tensorflow/tensorflow/c/c\_api\_internal.h

bazel-tensorflow/tensorflow/c/c\_api\_function.cc

bazel-tensorflow/tensorflow/c/eager/c\_api\_unified\_experimental\_graph.cc

bazel-tensorflow/tensorflow/c/c\_api\_experimental.cc

bazel-tensorflow/tensorflow/c/c\_api\_test.cc

bazel-tensorflow/tensorflow/c/c\_api\_function\_test.cc

bazel-tensorflow/tensorflow/c/while\_loop\_test.cc

bazel-tensorflow/tensorflow/c/c\_test\_util.cc

bazel-tensorflow/tensorflow/c/eager/c\_api\_experimental\_test.cc

## Classes Graph and GraphDef

The classes Graph (or Computation Graph) is a core concept of tensorflow to present computation. When first using TF, we first will create Computation Graph and pass the Graph to TF.

The Computation Graph is given by class TF\_Graph:

GraphDef is a serialization utility class which binds to Graph. The corresponding Protobuf interface is defined as:

// Represents the graph of operations

message GraphDef {

repeated NodeDef node = 1;

// Compatibility versions of the graph. See core/public/version.h for version

// history. The GraphDef version is distinct from the TensorFlow version, and

// each release of TensorFlow will support a range of GraphDef versions.

VersionDef versions = 4;

// Deprecated single version field; use versions above instead. Since all

// GraphDef changes before "versions" was introduced were forward

// compatible, this field is entirely ignored.

int32 version = 3 [deprecated = true];

// EXPERIMENTAL. DO NOT USE OR DEPEND ON THIS YET.

//

// "library" provides user-defined functions.

//

// Naming:

// \* library.function.name are in a flat namespace.

// NOTE: We may need to change it to be hierarchical to support

// different orgs. E.g.,

// { "/google/nn", { ... }},

// { "/google/vision", { ... }}

// { "/org\_foo/module\_bar", { ... }}

// map<string, FunctionDefLib> named\_lib;

// \* If node[i].op is the name of one function in "library",

// node[i] is deemed as a function call. Otherwise, node[i].op

// must be a primitive operation supported by the runtime.

//

//

// Function call semantics:

//

// \* The callee may start execution as soon as some of its inputs

// are ready. The caller may want to use Tuple() mechanism to

// ensure all inputs are ready in the same time.

//

// \* The consumer of return values may start executing as soon as

// the return values the consumer depends on are ready. The

// consumer may want to use Tuple() mechanism to ensure the

// consumer does not start until all return values of the callee

// function are ready.

FunctionDefLibrary library = 2;

}

# Protobuf interfaces and formats

## Core/Protobuf/Config.proto

<https://github.com/tensorflow/tensorflow/blob/master/tensorflow/core/protobuf/config.proto>

This interface contains various options:

1. for tuning the resources occupied by the GPU (see message [GPUOptions](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/core/protobuf/config.proto#L18-L194)).

Inside GPUOptions there are various experimental configuration options such as per virtual device memory limit, memory-specific options, kernel-specific timing and memory parameters (see message [Experimental](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/core/protobuf/config.proto#L100-L188)).

1. Optimizer tuning parameters (see message [OptimizerOptions](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/core/protobuf/config.proto#L197-L242)). For instance there is an option for selecting the level at which the Optimizer works where level L1 denotes common subexpression elimination and constant folding. a special option for turning on the internal Just-in-time compiler and selecting how aggressive the auto-compilation should be.
2. Various graph options (see message [GraphOptions](https://github.com/dimitarpg13/tensorflow/blob/master/tensorflow/core/protobuf/config.proto#L244-L289)). For instance when to build a cost model for the nodes in the graph in terms of memory and cpu resource consumption, parameters controlling various aspects of the graph construction and updates
3. Thread pool tuning parameters and options

# Exploring the C++ code examples

## Exploring //tensorflow/core/example

### Proto file for ***Feature***

This proto file contains protocol messages for describing features for machine learning model training or inference. There are three base ***Feature*** types:

* bytes
* float
* int64

A ***Feature*** contains Lists which may hold zero or more values. These lists are the base values ***BytesList***, ***FloatList***, ***Int64List***.

***Features*** are organized into categories by name. The ***Features*** message contains the mapping from the name to Feature. Here are example Features for a movie recommendation application:

*Feature {*

*key: “age”*

*value: { float\_list {*

*value: 29.0*

*}}*

*}*

*Feature {*

*key: “movie”*

*value: { bytes\_list {*

*value: “The Shawshank Redemption”*

*value: “Fight Clubs”*

*}}*

*}*

*Feature {*

*key: “movie\_ratings”*

*value: { float\_list {*

*value: 9.0,*

*value: 9.7*

*}}*

*}*

*Feature {*

*key: “suggestion”*

*value: { byte\_list {*

*value: “Inception”*

*}}*

*}*

*Feature {*

*key: “suggestion\_purchased”*

*value: { int64\_list {*

*value: 1*

*}}*

*}*

*Feature {*

*key: “purchase\_price”*

*value: { float\_list {*

*value: 9.99*

*}}*

*}*

|  |
| --- |
|  |

// containers to hold repeated fundamental values

message ByteList {

repeated bytes value = 1;

}

message FloatList {

repeated float value = 1 [packed = true];

}

message Int64List {

repeated int64 value = 1 [packed = true];

}

// Containers for non-sequential data.

message Feature {

// Each feature can be exactly one kind.

oneof kind {

BytesList bytes\_list = 1;

FloatList float\_list = 2;

Int64List int64\_list = 3;

}

}

message Features {

// Map from feature name to feature.

map<string, Feature> feature = 1;

}

// Containers for sequential data.

//

// A FeatureList contains lists of Features. These may hold zero or more

// Feature values.

//

// FeatureLists are organized into categories by name. The FeatureLists message

// contains the mapping from name to FeatureList.

//

message FeatureList {

repeated Feature feature = 1;

}

message FeatureLists {

// Map from feature name to feature list.

map<string, FeatureList> feature\_list = 1;

}

### Proto file for ***Example***:

An ***Example*** is a mostly-normalized data format for storing data for training and inference. It contains key-value store (features); where each key (string) maps to a Feature message which is one of packed BytesList, FloatList, or Int64List. This flexible and compact format allows the storage of large amounts of typed data, but it requires that the data shape and use be determined by the configuration files and parsers that are used to read and write that format. That is the ***Example*** is mostly not self-describing format. In TF, Examples are read in row-major format so any configuration that describes data with rank-2 or above should keep that in mind. For example, to store an M x N Matrix of Bytes, the BytesList must contain M\*N bytes with M rows of N contiguous values each. That is, the ByteList value must store the matrix as:

// .... row 0 .... .... row 1 .... // ........... // ... row M-1 ....

An Example for a movie recommendation application:

Features {

Feature {

key: "age"

value { float\_list {

value: 29.0

}}

}

Feature {

key: "movie"

value { bytes\_list {

value: "The Shawshank Redemption"

value: "Fight Club"

}}

}

Feature {

key: "movie\_ratings"

value { float\_list {

value: 9.0

value: 9.7

}}

}

Feature {

key: "suggestion"

value { bytes\_list {

value: "Inception"

}}

}

# Note that this feature exists to be used as a label in training.

# E.g., if training a logistic regression model to predict purchase

# probability in our learning tool we would set the label feature to

# "suggestion\_purchased".

Feature {

key: "suggestion\_purchased"

value { float\_list {

value: 1.0

}}

}

# Similar to "suggestion\_purchased" above this feature exists to be used

# as a label in training.

# E.g., if training a linear regression model to predict purchase

# price in our learning tool we would set the label feature to

# "purchase\_price".

// feature {

// key: "purchase\_price"

// value { float\_list {

// value: 9.99

// }}

// }

// }

//

# Building the tensorflow libraries and examples using Bazel

## Building tensorflow libraries

Building tensorflow libraries with debug symbols

bazel build --config=opt --verbose\_failures -c dbg --strip=never //tensorflow:libtensorflow\_cc.so

bazel build --config=opt --verbose\_failures -c dbg --strip=never //tensorflow:libtensorflow\_framework.so

Building tensorflow C++ api library using monolithic config

bazel build -c opt --config=monolithic //tensorflow:libtensorflow\_cc.so

## Building tensorflow core components

bazel build --config=opt //tensorflow/core:lib

exports the public non-test headers for:

//third\_party/tensorflow/core/platform: platform-specific code and external dependencies

lib/: Low-level libraries that are not TensorFlow-specific

bazel build --config=opt //tensorflow/core:framework

exports the public non-test headers for:

util/: General low-level TensorFlow-specific libraries

framework/: Support for adding new ops & kernels

example/: Wrappers to simplify access to Example proto

## Building Example Code

Building example code: parser configuration test

bazel build --config=opt //tensorflow/core/example:example\_parser\_configuration\_test

Building label\_image example code with debug symbols:

bazel build --config=opt --verbose\_failures -c dbg --strip=never tensorflow/examples/label\_image/...

Building label\_image example code with debug symbols and limited RAM resource (<2GB):

bazel build --config=opt --verbose\_failures -c dbg --strip=never --jobs 1 --local\_ram\_resources 2048 tensorflow/examples/label\_image/...

# Build Folder structure

ROOT\_FOLDER = /opt/tensorflow

Inside ROOT\_FOLDER/bazel-tensroflow/external :



By default tensorflow master as of 4/26/2020 installs the following external package dependencies:

*aws* *aws-c-common aws-c-event-stream*

*aws-checksums bazel\_tools boringssl*

*com\_google\_absl com\_google\_protobuf com\_googlesource\_code\_re2*

*com\_google\_grpc\_grpc curl double\_conversion*

*eigen\_archive farmhash\_archive fft2d*

*gif highwayhash jsoncpp\_git*

*libjpeg\_turbo local\_config\_cc local\_config\_cuda*

*local\_config\_git local\_config\_python local\_config\_rocm*

*local\_config\_sycl local\_config\_tensorrt nasm*

*nsync snappy zlib*

Some of the important libraries to build a C++ tensorflow app using the C++ tensorflow shared library libtensorflow\_cc.so:

The location of protobuf source: ROOT\_FOLDER/bazel-tensorflow/external/com\_google\_protobuf

![A screenshot of a cell phone

Description automatically generated]()

The location of eigen source: ROOT\_FOLDER/bazel-tensorflow/external/eigen\_archive

![A picture containing monitor

Description automatically generated]()

The location of grpc source: ROOT\_FOLDER/bazel-tensorflow/external/com\_github\_grpc\_grpc

![A screen shot of a monitor

Description automatically generated]()

The location of absl source: ROOT\_FOLDER/bazel-tensorflow/external/com\_google\_absl



The location of zlib source: ROOT\_FOLDER/bazel-tensorflow/external/zlib

![A close up of a screen

Description automatically generated]()

# Build first C++ Tensorflow app

1. Build the tensorflow C++ API library with

bazel build -c opt --config=monolithic //tensorflow:libtensorflow\_cc.so

or

bazel build --config=opt --verbose\_failures -c dbg --strip=never //tensorflow:libtensorflow\_cc.so

if we want to have DEBUG symbols available.

1. Build

# Third Party and External Packages

## **BoringSSL:** <https://boringssl.googlesource.com/boringssl/>

BoringSSL is a fork of OpenSSL that is designed to meet Google's needs.

Although BoringSSL is an open source project, it is not intended for general use, as OpenSSL is. We don't recommend that third parties depend upon it. Doing so is likely to be frustrating because there are no guarantees of API or ABI stability.

Programs ship their own copies of BoringSSL when they use it and we update everything as needed when deciding to make API changes. This allows us to mostly avoid compromises in the name of compatibility. It works for us, but it may not work for you.

BoringSSL arose because Google used OpenSSL for many years in various ways and, over time, built up a large number of patches that were maintained while tracking upstream OpenSSL. As Google's product portfolio became more complex, more copies of OpenSSL sprung up and the effort involved in maintaining all these patches in multiple places was growing steadily.

Currently BoringSSL is the SSL library in Chrome/Chromium, Android (but it's not part of the NDK) and a number of other apps/programs.

## **FarmHash:** <https://github.com/google/farmhash>

### *Introducing FarmHash*

#### Monday, March 31, 2014

*We’re pleased to announce the new [FarmHash](http://code.google.com/p/farmhash/" \t "_blank) family of hash functions for strings.  FarmHash is a successor to [CityHash](http://code.google.com/p/cityhash/" \t "_blank), and includes many of the same tricks and techniques, several of them taken from Austin Appleby’s [MurmurHash](https://code.google.com/p/smhasher/" \t "_blank).  
  
We’re heavily influenced by the types of CPUs that are common in Google’s datacenters, but FarmHash’s goals don’t end there. We want FarmHash to be fast and easy for developers to use in phones, tablets, and PCs too. So, yes, we’ve improved on CityHash64 and CityHash32 and so on.  But we’re also catering to the case where you simply want a fast, robust hash function for hash tables, and it need not be the same on every platform. To that end, we provide sample code that has one interface harboring multiple platform-specific implementations.  
  
Over time, we plan to expand FarmHash to include hash functions for integers, tuples, and other data. For now, it provides hash functions for strings, though some of the subroutines could be adapted to other uses.  
  
Overall, we believe that FarmHash provides high-performance solutions to some classic problems. Please give it a try! Contributions and bug reports are most welcome.*

## **HighwayHash**: <https://github.com/google/highwayhash>

Hash functions are widely used, so it is desirable to increase their speed and security. This package provides two 'strong' (well-distributed and unpredictable) hash functions: a faster version of SipHash, and an even faster algorithm we call HighwayHash.

SipHash is a fast but 'cryptographically strong' pseudo-random function by Aumasson and Bernstein [<https://www.131002.net/siphash/siphash.pdf>].

HighwayHash is a new way of mixing inputs which may inspire new cryptographically strong hashes. Large inputs are processed at a rate of 0.24 cycles per byte, and latency remains low even for small inputs. HighwayHash is faster than SipHash for all input sizes, with 5 times higher throughput at 1 KiB. We discuss design choices and provide statistical analysis and preliminary cryptanalysis in <https://arxiv.org/abs/1612.06257>.

# Appendix A: Bazel tutorial for Tensorflow Builds

# Appendix B: Hashing in Tensorflow

UNDERSTANDING HASH FUNCTIONS

by Geoff Pike

Version 0.2 --- early draft --- comments and questions welcome!

References appear in square brackets.

1 INTRODUCTION

Hashing has proven tremendously useful in constructing various fast

data structures and algorithms. It is typically possible to simplify

the analysis of hash-based algorithms if one assumes that the relevant

hash functions are high quality. At the other extreme, if the

relevant hash functions were always to return the same value, many

hash-based algorithms become algorithms that are slower, simpler, but still well-known.

For example, a chaining hash table devolves into a linked list.

There are many possible definitions of hash function quality. For

example, one might want a list of keys and their hashes to provide no

pattern that would allow an opponent to predict anything about the

hashes of other keys. Although I cannot prove it, I think I can meet

this and many other definitions of quality with

f(s) = SHA-3(concatenation of z and s),

where z is some secret string known only to me. This well-known trick

provides, I think, more high-quality hash functions than anyone will

need, though greater computational power in the future may push us to

replace SHA-3 from time to time.

In short, discussions about choosing a hash function are almost always

discussions about speed, energy consumption, or similar. Concerns

about hash quality are easy to fix, for a price.

2 ANATOMY OF A HASH FUNCTION

Hash functions that input strings of arbitrary length are written in

terms of an internal state, S. In many cases the internal state is a

fixed number of bits and will fit in machine registers. One generic

sketch of a string hash is:

let S = some initial value

let c = the length of S in bits

while (input is not exhausted) {

let t = the next c bits of input (padded with zeroes if less than c remain)

S = M(S xor t)

}

let n = the number of bytes hashed

return F(S, n)

where M is a hash function that inputs and outputs c bits, and F is a

hash function that inputs c bits (plus, say, 64 for its second argument)

and outputs however many bits one needs to return. In some sense we have

reduced the string-hashing problem to two integer hashing problems.

2.1 INTEGER HASHING TECHNIQUES

A hash function that inputs and outputs the same number of bits, say,

32, can use reversible bit-twiddling operations, each of which is

"onto" in the mathematical sense. For example, multiplication by an

odd constant is reversible, as all odd numbers are relatively prime to

2^32. Other commonly used reversible operations include:

o Adding or xoring a constant

o Bitwise rotation or other bitwise permutations

o bit j = (bit j) xor (bit k) for unequal constants j and k

o "Shift mix": S = S xor (S >> k), where k is, say, 17

o Replacing a fixed-length bit string with its cyclic redundancy

checksum, perhaps via \_mm\_crc32\_u32(f, <some constant>) [Pike]

Each of the above is a "bad" hash function that inputs and outputs

the same number of bits. One can simply compose two or more of those

bad hash functions to construct a higher-quality hash function.

One common quality goal for integer hashing (and string hashing) is

that flipping the 19th bit, or any other small change, applied to

multiple input keys, causes a seemingly unpredictable difference each

time. Similarly, any change to an input should lead to a seemingly

unpredictable selection of the output bits to flip.

Therefore, if we want a high-quality hash function that inputs c bits

and outputs fewer than c bits, we can simply truncate the output of a

high-quality hash function that inputs and outputs c bits.

To give a concrete example, here is Bob Jenkins' mix(), published in

1996 [Jenkins]. Its input is 96 bits in three 32-bit variables, and its output

is 96 bits. However, one may use a subset of the output bits, as every

output bit is affected by every non-empty subset of the input bits.

Input: a, b, and c

Algorithm:

a -= b; a -= c; a ^= (c>>13);

b -= c; b -= a; b ^= (a<<8);

c -= a; c -= b; c ^= (b>>13);

a -= b; a -= c; a ^= (c>>12);

b -= c; b -= a; b ^= (a<<16);

c -= a; c -= b; c ^= (b>>5);

a -= b; a -= c; a ^= (c>>3);

b -= c; b -= a; b ^= (a<<10);

c -= a; c -= b; c ^= (b>>15);

Output: a, b, and c

2.2 VARIATIONS ON STRING HASHING

There are three variations on our initial sketch worth noting.

First, for speed, one can special-case short inputs, as the CityHash

and FarmHash algorithms do. The number of special cases can be

reduced by using loads that may overlap: for example, a hash of a 9-

to 16-byte string can be implemented by a hash that inputs two 8-byte

values (the first 8 and last 8 bytes of the input string) and the string

length [CityHash, FarmHash].

Second, one may choose different means of incorporating input bits

into the internal state. One example: the mixing of S and input bits

may be interleaved with the mixing of parts of S and other parts of S.

Another example: the input bits processed in a loop iteration might be

xor'ed into multiple places in S, rather than just one, or might be

hashed with each other before touching S [Murmur]. The advantages and

disadvantages of these are unclear.

Third, one may repeatedly "squeeze information" from S, by remixing it with

itself and then revealing a subset of S. This is convenient when one would

like a family of hash functions with different output lengths. A special

case of the idea, called the "sponge construction," has been well studied and

adopted by the authors of Keccak and others [SHA-3].

3 HASH FUNCTIONS FOR HASH TABLES

It isn't hard to find real-life examples where hash tables or the hash

functions for them take more than 5% of a program's CPU time.

Improvements to hash tables and their hash functions are therefore a

classic example of software performance tuning. Unfortunately, the

best choice may be platform-dependent, so to avoid writing your own

collection of #ifdefs, please consider selecting something like the

FarmHash family of hash functions, that supply decent

platform-dependent logic for you.

To tune a program, often one will replace an existing hash function with a

faster, lower-quality hash function, despite the increased chance of unlucky

or pathological performance problems. Clever algorithms can mitigate this

risk. For example, hash tables can start with one hash function and then

switch to another if things seem to be going poorly. Therefore, one should

rarely plan to spend much CPU time on a secure hash function (such as SHA-3)

or a near-universal hash function (such as VHASH) when seeking the best

possible performance from a hash table. Against that, those types of hash

functions can limit the risk of pathological performance problems when one is

designing around typical hash-based algorithms that stick with a single hash

function no matter how it behaves on the data at hand.

4 REFERENCES

[Murmur] Appleby, Austin. https://code.google.com/p/smhasher,

https://sites.google.com/site/murmurhash/

[SMHasher] Appleby, Austin. https://code.google.com/p/smhasher

[SHA-3] Bertoni, Guido, et al. http://keccak.noekeon.org/

[Jenkins] Jenkins, Bob. http://burtleburtle.net/bob/hash/doobs.html

[VHASH] Krovetz, Ted. Message authentication on 64-bit architectures. In

Selected Areas of Cryptography – SAC 2006. Springer-Verlag, 2006.

[CityHash] Pike, Geoff and Alakuijala, Jyrki. https://code.google.com/p/cityhash

[FarmHash] Pike, Geoff. https://code.google.com/p/farmhash

[Pike] Pike, Geoff. <http://www.stanford.edu/class/ee380/Abstracts/121017-slides.pdf>

# Bibliography

1. *How to write shared libraries*, Ulrich Drepper, [drepper@gmail.com](mailto:drepper@gmail.com), Dec 10, 2011
2. *Semantic Versioning v2.0.0* <https://semver.org/spec/v2.0.0.html>