Introducing Beast: HTTP and WebSockets C++ library

Vinnie Falco
Ripple

CppCon 2016
Sep/18/2016

What is it?

- HTTP and WebSockets using Boost.Asio
- Header-only, C++11 or later, Open source
- Emulates Boost.Asio style
- In the Boost incubator for review
- Full documentation, tests, and samples
- Running on production Ripple servers!
- Git repo https://github.com/vinniefalco/Beast

chat on gitter | build passing | codecov 98% | coverage 98% | documentation master | license boost

ripple

Why Do We Need This?

HTTP Request in JavaScript

```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open( "GET", theUrl, false );
xmlHttp.send( null );
return xmlHttp.responseText;
```

Why Do We Need This?

HTTP Request in JavaScript

```
var xmlHttp = new XMLHttpRequest();
xmlHttp.open( "GET", theUrl, false );
xmlHttp.send( null );
return xmlHttp.responseText;
```

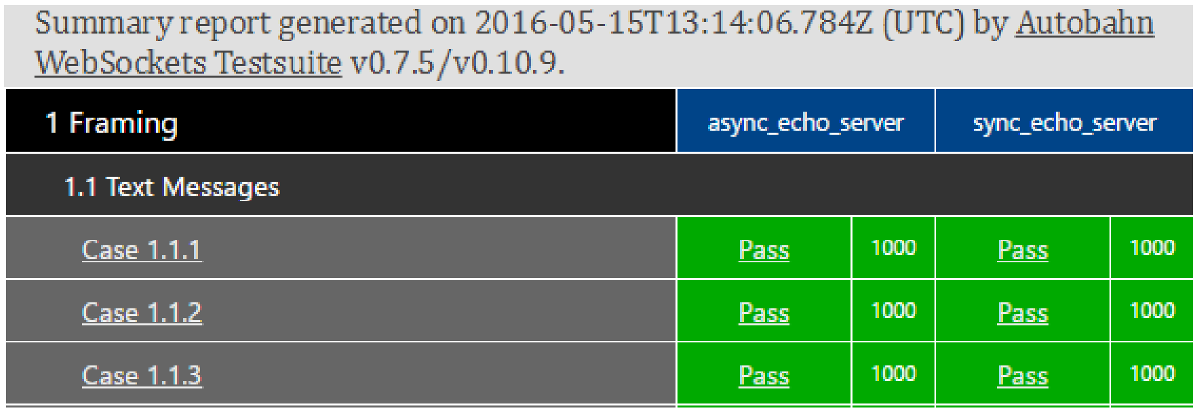
HTTP Request in C++

```
// ???
```

ripple

WebSocket Scope

- Establish WebSocket sessions
- Send and receive WebSocket messages
- Build clients or servers, sync or async
- Production-level performance
- Autobahn|Testsuite:



Summary report generated on 2016-05-15T13:14:06.784Z (UTC) by Autobahn WebSockets Testsuite v0.7.5/v0.10.9.

1 Framing	async_echo_server		sync_echo_server	
1.1 Text Messages				
Case 1.1.1	Pass	1000	Pass	1000
Case 1.1.2	Pass	1000	Pass	1000
Case 1.1.3	Pass	1000	Pass	1000

WebSocket Echo Example

- Connect to remote WebSocket echo server
- Handshake and send a message
- Receive and print echoed message

Connect to Remote Host

```
#include <beast/websocket.hpp>
#include <boost/asio.hpp>

using namespace boost::asio;

int main()
{
  auto host = "echo.websocket.org";
  io_service ios;
  ip::tcp::resolver r{ios};
  ip::tcp::socket sock{ios};
  connect(sock, r.resolve(
    ip::tcp::resolver::query{host, "80"}));
```

Handshake and Send a Message

```
// websocket::stream wraps your socket,
// SSL stream, or user defined type!
//
beast::websocket::stream<
  ip::tcp::socket&> ws{sock};

ws.handshake(host, "/");

ws.write(buffer("Hello, world!"));
```

Handshake and Send a Message

```cpp
// websocket::stream wraps your socket,
// SSL stream, or user defined type!
//
beast::websocket::stream<
  ip::tcp::socket&> ws{sock};

ws.handshake(host, "/");

ws.write(buffer("Hello, world!"));
```

Handshake and Send a Message

```
// websocket::stream wraps your socket,
// SSL stream, or user defined type!
//
beast::websocket::stream<
  ip::tcp::socket&> ws{sock};

ws.handshake(host, "/");

ws.write(buffer("Hello, world!"));
```

Receive and print echoed message

```
boost::asio::streambuf sb;
beast::websocket::opcode op;
ws.read(op, sb);

std::cout << to_string(sb.data());

// Send WebSocket close frame
ws.close(
  beast::websocket::close_code::normal);
```

ripple

Receive and print echoed message

```
boost::asio::streambuf sb;
beast::websocket::opcode op;
ws.read(op, sb);

std::cout << to_string(sb.data());

// Send WebSocket close frame
ws.close(
  beast::websocket::close_code::normal);
```

Receive and print echoed message

```
boost::asio::streambuf sb;
beast::websocket::opcode op;
ws.read(op, sb);

std::cout << to_string(sb.data());

// Send WebSocket close frame
ws.close(
  beast::websocket::close_code::normal);
```

Asynchronous Interfaces

```
boost::asio::streambuf sb;
beast::websocket::opcode op;

// Or use coroutines, std::future, or
// user defined types using Asio's
// `async_result` customization

ws.async_read(op, sb,
   [&](beast::error_code const& ec)
   {
     std::cout << to_string(sb.data());
   });
```

But Wait, There's More!

WebSocket uses HTTP to perform the handshake

HTTP Scope

- A universal HTTP message container
- Send and receive HTTP/1 messages
- Build clients or servers, sync or async
- Works with SSL or any Stream concept
- Production-level performance
- For library developers, not end users
- Use Beast to build higher level abstractions:
  (e.g. build a better curl)

HTTP GET Example

- Connect to remote host
- Assemble and send HTTP GET request
- Receive and print HTTP Response

Connect to Remote Host

```
#include <beast/http.hpp>
#include <boost/asio.hpp>

using namespace boost::asio;

int main()
{
  auto host = "boost.org";
  io_service ios;
  ip::tcp::resolver r{ios};
  ip::tcp::socket sock{ios};
  connect(sock, r.resolve(
    ip::tcp::resolver::query{host,"http"}));
```

ripple

Send HTTP GET Request

```
beast::http::request_v1<
  beast::http::empty_body> req;

req.method = "GET";
req.url = "/";
req.version = 11;
req.headers.insert("User-Agent", "Me");

// (`sock` could be an SSL stream)
beast::http::prepare(req);
beast::http::write(sock, req);
```

Send HTTP GET Request

```
beast::http::request_v1<
  beast::http::empty_body> req;

req.method = "GET";
req.url = "/";
req.version = 11;
req.headers.insert("User-Agent", "Me");

// (`sock` could be an SSL stream)
beast::http::prepare(req);
beast::http::write(sock, req);
```

Send HTTP GET Request

```
beast::http::request_v1<
  beast::http::empty_body> req;

req.method = "GET";
req.url = "/";
req.version = 11;
req.headers.insert("User-Agent", "Me");

// (`sock` could be an SSL stream)
beast::http::prepare(req);
beast::http::write(sock, req);
```

ripple

Receive HTTP Response

```
beast::http::response_v1<
    beast::http::string_body> res;

// (`sock` could be an SSL stream)
boost::asio::streambuf sb;
beast::http::read(sock, sb, res);

std::cout << res;
```

ripple

Receive HTTP Response

```
beast::http::response_v1<
    beast::http::string_body> res;

// (`sock` could be an SSL stream)
boost::asio::streambuf sb;
beast::http::read(sock, sb, res);

std::cout << res;
```

Asynchronous Interfaces

```
beast::http::response_v1<
    beast::http::string_body> res;

// Or use coroutines, std::future, or
// user defined types using Asio's
// “async_result” customization

boost::asio::streambuf sb;
beast::http::async_read(sock, sb, res,
    [&](beast::error_code const& ec)
    {
        std::cout << res;
    });
```

ripple

Advanced HTTP Features

- Customize the message body
  - User defined type in message
  - Custom algorithm for serializing and deserializing
- Send incremental body data from coroutine
- Read-only message bodies
  (e.g. A body that streams from a file)
- HTTP/1 parser is zero alloc and self contained

ripple

Summary

- https://github.com/vinniefalco/Beast
- If Christopher Kohloff (Boost.Asio author) wrote an HTTP and WebSockets library, it would look like this!
- Any questions?

HTTP

```
request_v1<empty_body> req;

req.method = "GET";
req.url = "/";
req.version = 11;

prepare(req);
write(sock, req);
```

WebSockets

```
stream<socket&> ws{sock};

ws.handshake(host, "/");

ws.write(asio::buffer(
  "Hello, world!"));
```

ripple

HTTP Parser Performance

```
beast.http.parser_bench Parser speed test,
34377KB in 200000 messages
sizeof(request parser)  == 48
sizeof(response parser) == 48

nodejs_parser
Trial 1: 4111 ms
Trial 2: 4096 ms
Trial 3: 4091 ms

http::basic_parser_v1
Trial 1: 4510 ms
Trial 2: 4520 ms
Trial 3: 4527 ms

Longest suite times:
   26.2s beast.http.parser_bench
```

HTTP Message Container

```
template<
  bool isRequest,
  class Body,
  Class Headers = beast::http::headers
>
struct message
{
  Headers headers;

  // Trait controls this member's type!
  typename Body::value_type body;
};
```

HTTP Body Concept

```
struct string_body
{
  // The message::body data member type
  using value_type = std::string;

  // Algorithm for reading a string_body
  class reader;

  // Algorithm for writing a string_body
  class writer;
};
```