A Season with D: SAoC Report

A Season with Tales of a Symmetry Autumn of Code participant
Symmetry Autumn of Code

- Promoted by the D Language Foundation and Symmetry Investments
- Period of 4 months: September 2018 – January 2019
- One milestone each month
- 3 Participants, each one working on a different project

Objective?

Improving the D Ecosystem
About me

• CS student in Italy, 24 years old

• Experienced in C, Python, some Java. Met D while trying to expand my experience in programming languages

• Before SAoC: ~1 year of D experiments

• Fond of language theory, concurrent program verification, networking protocols

• Looking forward to contribute to an Open Source codebase
HTTP/2 in Vibe.d

• Work on the experimental *vibe-http* repository.

• Mentor: Sönke Ludwig

• Project idea found on Dlang’s wiki page for GSoC 2018.

• HTTP/2 implementation had to be written almost from scratch.

HTTP/2? What does that 2 stand for?
## HTTP/1 vs HTTP/2

<table>
<thead>
<tr>
<th>Stateless: One request per TCP Connection.</th>
<th>Stateful: Connection state is maintained between requests.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cleartext protocol:</strong> Headers and data are transmitted without further serialization.</td>
<td><strong>Binary Protocol:</strong> Frames are serialized using the HPACK algorithm and transmitted as binary.</td>
</tr>
<tr>
<td>Relies on TCP to implement flow control at the packet level.</td>
<td>Integrates flow control at the Frame level using a connection window.</td>
</tr>
<tr>
<td><strong>Ordered</strong> and blocking.</td>
<td><strong>Multiplexed</strong> using the concept of Streams.</td>
</tr>
</tbody>
</table>
Work Plan

HTTP/2 is composed roughly of 3 logic blocks:

- HTTP/1 to HTTP/2 Protocol switching
- The HPACK header compression algorithm
- Asynchronous handling of HTTP/2 Streams

Each block became a milestone in my work plan.
Milestones

Milestones acted as checkpoints to ensure us participants had a plan to follow.

- Planned in advance while proposing an application to SAoC, with the help of our respective mentors

- Progress on a milestone was reported month by month

- Proposed by applicants, reviewed by organizers and mentors
Working with a mentor

- **Discussion topics**: bugs found, implementation issues, high-level strategy.

- **Means of communication**: e-mails and GitHub PRs.

- **Difficulties**: adapting to the mentor’s development cycle.

- **Benefits**: constant communication improved focus and commitment towards a complete project realization.
Workflow

September – October 2018

Dedicated to:

• Studying the available resources regarding HTTP/2, mainly RFC 7540 and RFC 7541.

• Reading through vibe’s HTTP module to familiarize with it.

• Discussing an high-level strategy with my mentor.

Milestone: HTTP/1 to HTTP/2 Protocol Switching
Workflow

October – November 2018

Dedicated to:

• Implementing the HPACK header compression algorithm.

• Digging deep into Phobos modules, especially std.range, std.algorithm and std.traits.

• Testing the obtained HPACK implementation.

Milestone: HPACK header compression algorithm
Workflow

November – December 2018

Dedicated to:

• Devise an efficient strategy for asynchronous management of HTTP/2 Streams
• Compare performance and logic of different implementations in various languages (C, C++, Go)
• *Testing, bug fixing, testing, bug fixing,* ...

Milestone: HTTP/2 Streams and Multiplexing
Workflow

December 2018 - Ongoing

Work that is still being pursued:

- Complete the review of HTTP/2 Streams and Multiplexing so that it can be merged in *vibe-http*

- Profile web server performance and memory usage, trying to minimize dynamic allocations and remove as much *GC-dependent* code as possible.

- *Testing, bug fixing, testing, bug fixing, ...*
Accomplishments

It's Alive!
Accomplishments

The introduction of HTTP/2 in *vibe-http* is geared towards building a new HTTP module to be merged in Vibe.d.

Modifying the existing codebase is still going to be a long process and has to be carried out with patience and care.

Yet, the new HTTP server lives, and talks with HTTP/1 and HTTP/2 clients successfully...

...in an experimental way.
Accomplishments

Protocol Switching: HTTP/2 is enabled transparently for cleartext HTTP connections and requires minimal changes for HTTPS ones.

```cpp
void main() {
  /** Example of HTTP/2 webserver initialization using TLS **/
  HTTPServerSettings settings;
  settings.port = 8000;
  settings.bindAddresses = ["127.0.0.1"];

  /** TLS Context initialization **/
  settings.tlsContext = createTLSContext(TLSContextKind.server);
  settings.tlsContext.useCertificateChainFile("server.crt");
  settings.tlsContext.usePrivateKeyFile("server.key");

  settings.tlsContext.alpnCallback(http2Callback);  /** Setup ALPN to accept HTTP/2 **/

  listenHTTP!requestHandler(settings);
  runApplication();  /** start the event loop **/
}
```
Accomplishments

**HPACK module**: a simple interface based on `std.range` which can encode and decode arbitrarily long chunks of headers.

- Can be used as standalone library (outside `vibe-http`)
- The interface is `@safe` and *almost* `@nogc`, if it wasn’t for those exceptions…
- **Lightweight** on CPU and memory resources, protecting from possible DoS attacks which exploit the heavy computational cost of the encoding algorithm.
Accomplishments

HTTP/2 message exchanges:

- Handling multiple HTTP requests over the same connection by using asynchronous handlers based on Vibe.d’s fibers.

- Complying with flow control restrictions and manage data dispatch while responding to other requests, by performing each dispatch on a separate task.

- Enforcing protocol correctness by maintaining a connection state, enforcing compliance with protocol rules defined in RFC 7540.
Missing Bricks

- **Review** of the PR regarding stream multiplexing and flow control logic has yet to be completed, and will probably undergo some changes.

- **Server Push**, an additional feature of HTTP/2 which allows a server to send data without a previous request. Not a requirement for HTTP/2 functionality: postponed.

- *Testing, bug fixing, testing, bug fixing, …*
Lessons Learned

D for network protocols

Implementing HTTP/2 required application of some of the language features:

- Ranges as a serialization interface.
- Meta-Programming through mixins, CTFE and ADTs.
- Abstract concurrency paradigm based on Tasks.
Ranges
D for network protocols

Ranges proved to be a powerful abstraction to handle data serialization.

- HPACK interface: based on `std.range` and `std.algorithm`.
- Methods templatized on `InputRange` and `OutputRange`, usable outside of the HTTP/2 module.
- Support custom allocators by using `stdxallocator`.
- Could be `@nogc`...
import vibe.http.internal.http2.hpack : decodeHPACK;

void exampleDecode(IN,OUT)(IN encodedSource, OUT dst) @safe
if(isInputRange!IN && isOutputRange!OUT)
{
    const uint size = 4096;
    auto table = myCustomAllocator.make!(IndexingTable(size));
    decodeHPACK(encodedSource, dst, table, myCustomAllocator, size);
}
Meta-Programming

D for network protocols

HPACK algorithm uses an indexing table (HT), chaining:

• A static, immutable table which is generated at compile time.

• A dynamic table built around a fixed-length ring buffer.

The indexing table provides an unified address space for the stored data, represented as an algebraic data type.
immutable size_t ST_SIZE = 61;
static immutable H2TableField[ST_SIZE+1] staticTable;

static this() {  
    staticTable = [  
        H2TableField("hname","hvalue"),  
        ... ];  
}  

struct IndexingTable {  
    private {  
        DynamicTable dTable;  
    }  
    ...  

    // H2TableSize is a { name: ADT(integers, ubyte, string) }  
    H2TableField opIndex(const size_t idx) @safe  
    {  
        enforceHPACK(idx > 0 && idx < size);

        if (idx < ST_SIZE+1) return staticTable[idx];
        else return dTable[ dTable.index - (idx - ST_SIZE) + 1 ];
    }  
    ...  
}
Concurrency
D for network protocols

Vibe.d’s concurrency paradigm:

- Uses the asynchronous I/O model, spawning Tasks (AKA Fibers) using the internal event loop.
- Supports message passing between Tasks.
- Allows for lightweight multiplexing of incoming requests over HTTP/2 Streams.

HTTP/2 Connection and Request handling is built around Tasks.
Experience Analysis

A participant’s perspective
Why SAoC?

- A great occasion to learn from experienced programmers and to get to know them
- Lightens the burden of starting as a OSS contributor
- Gain insights about the dynamics of a Open Source Software community
Why SAoC?

• A great occasion to learn from experienced programmers and to get to know them

• Lightens the burden of starting as a OSS contributor

• Gain insights about the dynamics of a Open Source Software community

• All of that while being paid
Starting as a contributor

I want to start contributing to D, but:

1. I can’t decide what to work on.
2. I don’t know who to talk to.
3. I have no experience and I am afraid of not completing my work.

How can SAoC/GSoC help me?
Starting as a contributor

I can’t decide what to work on.

• The Dlang wiki provides idea pages, where community members listed possible useful projects. Mine: GSoC 2018 ideas page.

• Each project has an assigned mentor which can be contacted in case one wishes to start a collaboration.
Starting as a contributor

I don’t know who to talk to.

- Post on the D forum
- Ask on IRC
- Write a simple proposal and send it by e-mail to the maintainers of the project of your interest.

*Edit: See @wilzbach presentation*
Tackling a huge project

Issues

The first time on a well-established codebase might signify:

- Not knowing where to start.
- Little understanding of the internal logic of the project.
- Little familiarity with the development process.
- A lot of time to be spent reading.
Tackling a huge project

Useful Resources

Never be afraid of reading! My case:

• RFCs which detailed the structure of what I had to build.

• Code documentation for Phobos, Vibe.d, and particular projects.

• Similar projects: Webservers which already introduced support for HTTP/2.

• Hints: Forum posts, IRC discussions.
Tackling a huge project

Asking for help

Mentoring: key to a successful project kick-off.

What if my mentor cannot help?

• Write as many forum posts as you need
• Discuss the issues on IRC

Let your needs be heard!
Notes for the future
Benefits of SAoC

- Spreading D usage
- Pushing development of D features
- Fixing bugs thanks to fresh eyes and minds
- Bringing new people inside the community
Risks of SAoC

- Participants might drop early
- Work might require more time than planned
- Newly introduced features might be orphaned after the experience ends
Possible Improvements?

Garfield
Possible Improvements

Looking forward to future editions

• Increase communication with the participant using the community channels.

• Improve means of gathering ideas.

• Determine if project comparison is needed, and clarify a base for that.
Communication

Mentors might be busy!

- Causes delays on milestone delivery
- Need for someone to follow participants in need of help
- Need for participants to properly document their progress so that others can jump in
Communication

Helps quantify the impact of the project

• Encourage periodic reporting of the progress made to the whole community.

• Encourage forum posts and discussion during the experience.

Aim: motivate the participants
Fostering Ideas

wiki.dlang.org pages are often left to rot.
Fostering Ideas

RIP

- https://wiki.dlang.org/Project_Ideas
- https://wiki.dlang.org/Wish_list
- https://wiki.dlang.org/Language_design_discussions
- Project pages that should be updated or removed, e.g. https://wiki.dlang.org/Calypso
Fostering Ideas

Maintain one reference page

- Periodically refreshed with new / updated ideas
- Periodically cleansed from old projects
- Linked from the dlang.org website directly
- Makes contributors autonomous in finding a project
- Used as reference regarding planned and desired work
Project Comparison

How to compare work from different participants?

• Projects have different requirements

• Milestone design is left to each participant

• Communication channels are various (e-mail, forum, IRC, Slack...)

• Development process is not unique for all projects
Project Comparison

Objectively judging a project needs some sort of data quantification.

- LoC written?
- Pull Requests submitted?
- Communication activity?

I’ve gathered some data from Francesco Mecca: SAoC Participant, Fork-Based GC for DRuntime
Project Comparison

Fork-based GC

- Study of GC implementations, from Sociomantic to Phobos
- Design of test suites and benchmark suites
- Implementation of fork-based GC against the current DRuntime (~ 500 LoC)
- Study of the use cases suitable for a fork-based GC
- Multiple rebase cycles against upstream DRuntime
## Project Comparison

<table>
<thead>
<tr>
<th></th>
<th>Fork-based GC for DRuntime</th>
<th>HTTP/2 in Vibe.d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pull Requests opened</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>Pull Requests opened, reviewed and merged</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>Number of e-mails exchanged with respective mentors</td>
<td>128</td>
<td>24</td>
</tr>
<tr>
<td>Lines of Code written and submitted</td>
<td>1111</td>
<td>6563</td>
</tr>
<tr>
<td>Lines of Code written, reviewed and merged</td>
<td>572</td>
<td>4988</td>
</tr>
</tbody>
</table>
# Project Comparison

<table>
<thead>
<tr>
<th></th>
<th>Fork-based GC for DRuntime</th>
<th>HTTP/2 in Vibe.d</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pull Requests opened</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>Pull Requests opened, reviewed and merged</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>Number of e-mails exchanged with respective mentors</td>
<td>128</td>
<td>24</td>
</tr>
<tr>
<td>Lines of Code written and submitted</td>
<td>1111</td>
<td>6563</td>
</tr>
<tr>
<td>Lines of Code written, reviewed and merged</td>
<td>572</td>
<td>4988</td>
</tr>
</tbody>
</table>
Project Comparison

Numerical data is not sufficient.

• Some projects might require more design than actual keyboard work

• Some might require more benchmarks or testing

• Some might depend on the code submission method
Project Comparison

HTTP/2 in Vibe.d

Actual implementation

Discussion, design, benchmarks...
Project Comparison

Fork-based GC for DRuntime

Actual implementation

Discussion, design, benchmarks...
Project Comparison

What’s the point in comparing projects then?

Possible reasons:

• Strengthen competition between participants to enhance work done

• Award one or more participants as “winner” as an additional benefit

• ? ← Needs more discussion

Not needed for actual project completion!
Project Comparison

How to ensure project completion?

- Careful planning: give all the necessary time for milestones to be laid down
- Ensure mentors are available and willing
- When selecting participants, consider interest and proposal over technical preparation and CV
- Require a significant amount of documentation, so that work doesn’t get lost! 
Conclusions

Symmetry Autumn of Code

• Is a great occasion to introduce new people into the D community

• Can be an alternative and stimulating approach to starting as a contributor

• Needs more guarantees: use communication, gathering of idea, project evaluation.
Thank you!
Questions?
Acknowledgements

Heartfelt thanks to:

- The D Language Foundation and Symmetry Investments for making SAoC possible.
- Mike Parker and everyone who worked to organize and manage the experience.
- Sönke Ludwig, who helped me through the process as my mentor.
- The D community who created invaluable resources through discussion on the forum, the blog, all the IRC channels… Keep it up!
- Francesco Mecca, who helped me as a participant of SAoC and long-time fellow developer.
- Andrea, who’s been by my side the whole time.
References

- **SAoC Project Repository**
  - vibe-hpack: https://github.com/gallafrancesco/vibe-hpack (merged in vibe-http)


- **[RFC 7541] HPACK: Header Compression for HTTP/2**

- **GSoC 2018 Ideas Page**
  https://wiki.dlang.org/GSOC_2018_Ideas

- **SAoC 2018 Ideas Page**
  https://wiki.dlang.org/SAOC_2018_ideas

- **DLang's Blog posts about SAoC**
  - Presentation, https://dlang.org/blog/symmetry-autumn-of-code/
Appendix: HTTP/2

- Stateful, connection-aware protocol
- Server and client communicate through HTTP requests embedded in Frames.
- A request / response cycle of Frames is called a Stream.
- Streams can be multiplexed over the same TCP connection.
- Exploits Huffman coding to compress Frame headers through the HPACK algorithm.
HTTP/2

A stream is a request / response cycle, has finite lifetime and cannot be reused.
HTTP/2

Client Webpage

TCP Connection

HTTP/2 Server

GET / HTTP/2 […]

HTTP/2 200 OK […]

[optional] DATA Frames

Stream #1

Stream #2

Stream #3
HTTP/2

GET / HTTP/2 […]
HTTP/2 200 OK […]
[optional] DATA Frames

Stream #1
Stream #2
Stream #3

Client Webpage

TCP Connection

HTTP/2 Server

main.css

my_cat.png
HTTP/2

No ordering forced: Stream responses can interleave!

GET / HTTP/2 [...]
HTTP/2 200 OK [...]  
[optional] DATA Frames  

Stream #1  
Stream #2  
Stream #3  

Client Webpage
TCP Connection
HTTP/2 Server

main.css
my_cat.png
Reasons behind HTTP/2

Well-known FreeBSD developer: "The protocol has [...] layering violations, inconsistencies, needless complexity, bad compromises" [1]

Is he right? Yes:

• HTTP/2 maintains stateful connections over TCP, which is already connection-aware.

• Encodes all headers as binary, using more memory and computing power than HTTP/1.1

• Not intrinsically safer: web browser only implement it over TLS.

[1] HTTP/2.0 - The IETF is Phoning It In, Poul-Henning Kamp, acmqueue vol. 13 issue 2, 2015
Reasons behind HTTP/2

What the hell did you work on then?

HTTP/2 is controversial yet HTTP/1.1 is not built to scale efficiently when faced with:

- Large requests (high number of header fields): lots of noise for a simple GET

- Dynamic webpages which require more than one HTTP request. HTTP/1.1 sends each request on a separate TCP connection (often destroying TCP flow control mechanism)

HTTP/2 is useful for “heavy” HTTP servers.
Reasons behind HTTP/2

A great number of existing web servers support HTTP/2.

- Not all implementations are equal (some details of stream prioritization and flow control can be different)
- Not all websites need HTTP/2: my blog doesn’t, your online platformer might.
- Having HTTP/2 in Vibe.d means being able to compete with existing web servers as a standalone application.

Benefits: increased development surface, larger test base: solid framework in the long run.