Initial Report – GOSSIP

Daniel Fomin, Deniz Etkar

May 23, 2020

1. Team

Team name: International Gossipers

Team members:

- Daniel Fomin
- Deniz Etkar

2. Module

Module & group number: Gossip – 6

Module implementation: Byzantine resilient random membership sampling (BRAHMS)

3. Programming Environment

Programming language: Go

Operating System: Windows 10

Reasons for choosing Go: Provides extensive support for networking, cryptography and parallel programming (goroutines). Full cross-compatibility support for FreeBSD, Linux, macOS and Windows.

Reasons for choosing Windows 10: It is the OS that was already in use by the team members and they are more comfortable coding in it. Windows Subsystem for Linux (WSL) could not be used due to a well-known issue related to debugging Go language in WSL1. Windows 10 is chosen without loss of cross compatibility thanks to Go language.

See: https://github.com/microsoft/vscode-go/issues/2505#issuecomment-566600059

Build system: *go install* command provided by Go language tools. *makefile* will also be available as an alternative build system.

4. **Testing & Code Quality**

- Specifications for both P2P and API will be documented. Unit tests for checking P2P functionality and integration tests for checking API functionality will be prepared according to the relevant specifications.
- 'gopls' language server is used for formatting and diagnostics to ensure code readability. 'govet' is used for checking the correctness of the code to ensure code quality.
- Wireshark for packet sniffing and network debugging.

5. Available Libraries

As we want our module to be particularly stable and secure, we will use a number of go libraries.
The most important library is "net" which implements an interface for network I/O such as TCP/UDP, domain resolution and Unix domain sockets. We plan to use it most of all for the API communication and P2P communication.
To read files such as the configuration file we will use the "io" library.
To read and write the data that is transferred between different nodes, we will also need some kind of encoding, the implementation for which we will use from the library "encoding".
To secure this transfer, we also need the library "crypto", which has implementations for example for the SHA512, AES or different elliptic curve algorithms.
Finally we will need some kind of serialization for the transfer of data between the nodes, for which we will use the "protobuf" (https://developers.google.com/protocol-buffers) library from Google. It allows to specify a structure for the transferred data and has integration to many programming languages.

6. License

As we want our module to be open sourced, we chose the GNU license for making the source code available to everyone. This allows the module to be further refined by the coding communication or to distribute the knowledge so that other projects can build upon the work we have already done.

7. Relevant Experience

Our experience in the programming language go can be summarized as basic. Deniz Etkar was already able to gain some basic knowledge in this language but did not have any real-world projects related with it. Daniel Fomin has also only some basic understanding about the language that he gained from reading a handful of open source go projects.
Both of us did not have any lectures related to distributed systems so we are eagerly learning everything we need from this lecture.
Deniz Etkar had lectures in networking but did not have any experience with practical network programming. Daniel Fomin did a small amount of networking projects such as building a traceroute application or implementing a novel Interface to the Routing System protocol that allows nodes in a software defined network to be controlled by a central controller.

8. Workload

We will distribute our workload by dividing the module in three parts. The first part is API communication which will deal with the interface to the other modules. The second part is the P2P communication which will be the implementation of the communication between different nodes within the module. The last part is the control logic which will be the implementation of the gossip protocol itself.
Deniz Etkar will start with the control logic part and Daniel Fomin will start with the API communication part. After the first team member is done with his part, we will distribute the last part between the two members.