Final Report

Group 16 #Chili, RPS module, Alexandru Obada and Nicolas Frinker

1. Documentation for your software

a) What dependencies are required for your software

In order to list all the dependencies, including transitive ones run the following command from the root folder:
mvn dependency:tree

All the direct dependencies are listed in the /pom.xml file, in the **dependencies** section.

A short explanation of all the dependencies:
1. Lombok - syntactic sugar, increases code quality
2. Ini4j - *.ini parser, used for configuration parsing
3. commons-cli - command line parsing
4. cucumber - used for testing, enables Behaviour Driven Development
5. guava - in this project it is used mainly for byte to primitive types transformations and vice versa
6. Google Truth - really nice assertion library, used for testing
7. HikariCP - high performance, light weight database connection pool, used by hibernate to connect to H2
8. H2 - Known as in memory database, but it is used as a file based database
9. Netty - networking framework, all the networking is based on it.
10. Bouncy castle - Security framework, used it for *.pem file parsing
11. Snake yaml - Yaml parser, used in the file based bootstrapper implementation.

b) How to install and run your software

Before proceeding, make sure that the build machine has Java 1.8 or higher and maven 3.2. The target machine ("on which the software will be running") is required to have Java 1.8 or higher installed.

- Checkout repository
- Build software with Maven using "mvn package", this will generate multiple shippable artifacts:
  - random-peer-sampler-<version>.jar
  - random-peer-sampler-<version>.zip
  - random-peer-sampler-<version>.tar.gz
  - random-peer-sampler-<version>.tar.bz2
- Choose one of the archives, depending on the preferences (.zip, .tar.gz, .tar.bz2), move it to the target location and unpack

- The unpacked folder should contain the following:
    - config = configuration, bootstrap
    - docs = documentation
    - LICENSE
    - random-peer-sampler-<version>.jar
    - run.sh
- Run ./run.sh --c=<path to configuration file>
- Alternatively run “java -jar target/random-peer-sampler-1.0-SNAPSHOT.jar”

N.B. If the --c=<path> is not specified it defaults to ./config/config.ini
Also, the application populates the initial peer set from a bootstrap file.

The configuration is divided into two files:
Bootstrap.yml
- Contains a list of bootstrap peers for the RPS module

config.ini
- Configuration file as documented in the specification
- For the RPS module, there are the following settings:

```
api_address = 127.0.0.1:6003
listen_address = 127.0.0.1:7001
round_duration = 5000
sampler_num = 100
validation_rate = 300000
sampler_timeout = 30000
pull_ratio = 0.1
bootstrap_file=<path_to_bootstrap>
```

c) Any known issues while running your software

In a small network the peers are slowly propagating.
The logback configuration has to be adjusted (see logback.xml) to your needs, at the moment it is configured to write all the output to the console, and log level INFO which leads to excessive logging.

2. Protocols

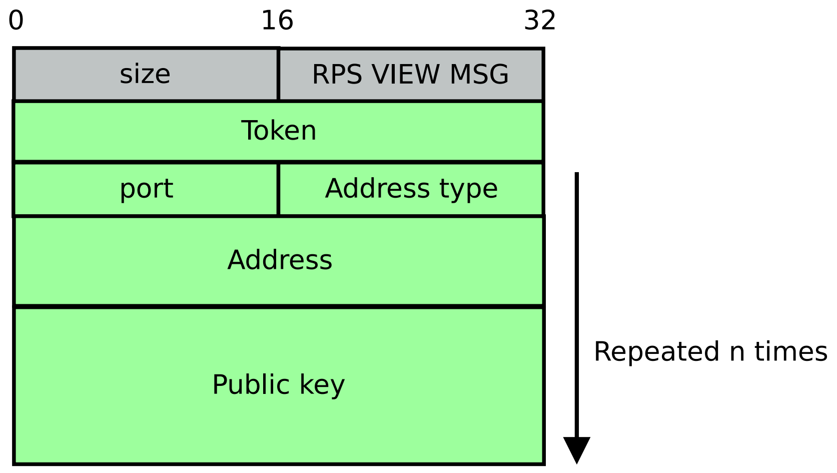
We have our own protocol on the P2P interface between RPS modules of different peers, where our module can send and receive directly addressed packets.

The implemented RPS protocol consists of the following messages:

- RPS View - Exchange dynamic view
- RPS Push - Introduce the node to the network
- RPS Ping - Monitor the availability of a sampled peer

The Push messages is periodically sent with frequency $1/R$ where R is the round duration, this parameter can be configured in the *.ini config file. The delivery of the RPS Push message is delegated to the Gossip module.
When a node will receive an RPS Push message it will reply with an RPS View Message, this message contains the local Dynamic View, (all the peers learned by the peer), this message is sent via the RPS P2P interface. The RPS View Message is not deterministic, a random variable is evaluated first, the reply probability can be tweaked via the *pull_ratio* value, it is expected to be a value in [0, 1] interval, higher the ratio higher the probability to reply with a view message. The ratio is depending on the network size the NSE module is estimating.
The RPS Ping messages serve as a heartbeat for the sampled peers, as soon as a peer stops replying to the ping the sampling units containing it will reinitialize.

RPS View Message

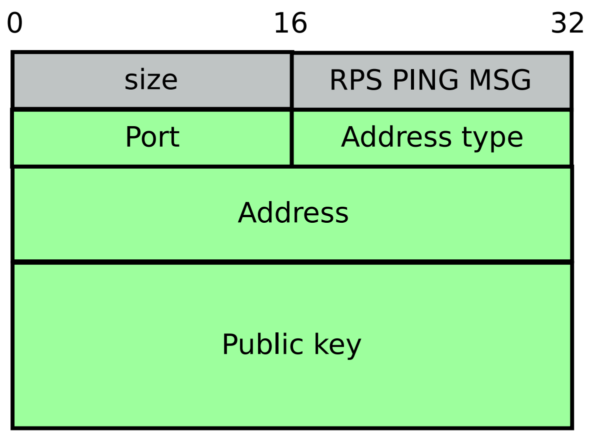


RPS VIEW MESSAGE has a type value of 542.
Each message contains the common header (size and message type) followed by a list of peers.
The first peer is the source peer, all following peers are peers from the originator's view.

Each peer consists of the listening address (might be IPv4 or IPv6) and port and the public key of the peer. The identity can be derived from the public key.

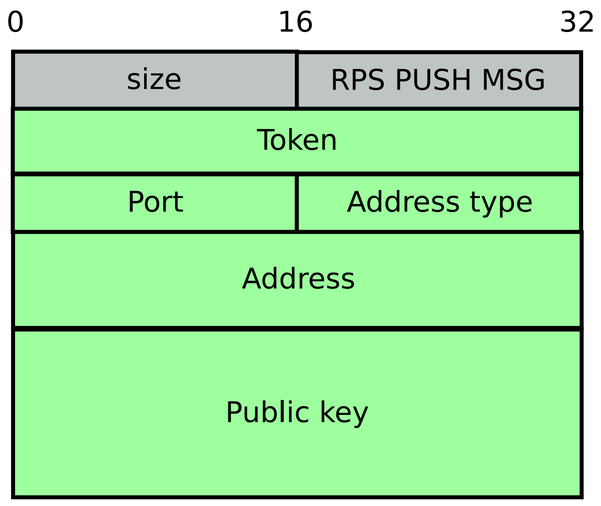
The address type contains the number (integer) of the used version of the IP protocol (currently either 4 or 6).

RPS Ping Message



RPS PING MESSAGE has a type value of 544.
Each message contains the common header (size and message type) followed by a single peer.

RPS Push Message



RPS PUSH MESSAGE has a type value of 543.
Each message contains the common header (size and message type) followed by a secret token and a single peer.
The token is echoed by receiving peers in the RPS View Message so the peer who sent the push can be sure the sender of the view is a valid peer in the network (otherwise he would not have received the token in the push message and therefore, could not answer with a valid one).

3. Future Work

Features/enhancements we you wanted to include in your software but couldn't:

The connections should be encrypted to make the token confidential. This would it make impossible for an attacker to send us valid view messages that could spoil our dynamic view.

4. How was the work divided in your team — who did what?

More or less weekly meeting (in person or remotely) where we synced, discussed and made up single tasks we distributed among us fairly. We have tried to keep the tasks small enough, in order to let every member of the team to work on different topics and not concentrate on just specific aspects, as a result all the team has a good knowledge of the implementation details as well as design decisions of all the software components. To even further enforce that aspect, we did Pull Reviews, every piece of code that went into the master branch had to be reviewed by the other team member.

5. Effort spent for the project

The project went for about 4-5 months. We had a meeting every week for about 1 hour, where we synced and distributed the tasks. In order to equally distribute the effort, the tasks had been relatively small, normally we tried to take a workload of around 8 hours per week per team member, which would include the time for deeper understanding of the tasks, the implementation, review and merging.