CHAIR OF NETWORK ARCHITECTURES AND SERVICES

TECHNICAL UNIVERSITY OF MUNICH

Project Documentation

Author: **Thua-Duc Nguyen & Duc-Trung Nguyen**

1 Architecture

The final architecture of our project is similar to the one we presented in our midterm project. As the specification requires, the `gossip` module runs as two independent protocols: one API protocol and one P2P protocol. These two protocols share some data to fulfill the functionality of the module.

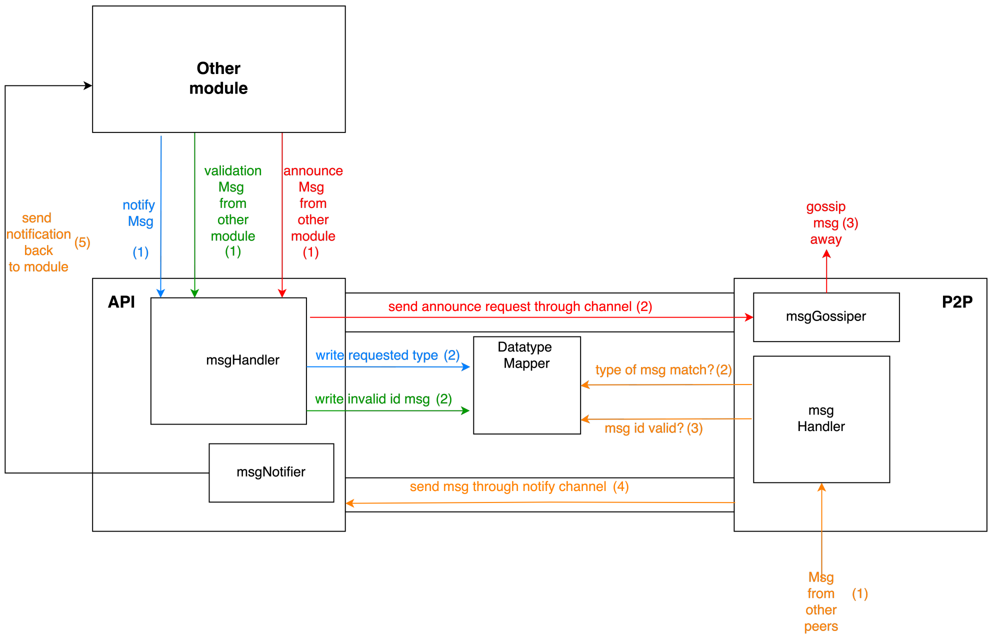

Figure 1: Structure of the gossip module

1.1 Shared data between API and P2P

AnnounceMsgChan

We have an announce `Go channel` [1] (marked in red in Figure 1) shared between the two protocols. Whenever the `API` protocol receives an announce message from another module, it processes the message immediately and sends it to the `P2P` protocol through this channel. The `P2P` protocol has an announce message handler running on a goroutine that always listens to this channel. When it receives an announce request, it will gossip this message away.

DatatypeMapper

We also need a `DatatypeMapper` (marked in black in Figure 1) shared between two protocols. Whenever API receives a notify message, it will write the message type that is valid into the mapper and hence should be propagated further. `DatatypeMapper` also contain a list of invalid message id, which is sent by other module through `Validation` endpoint. This list is being used by `P2P` to check for the validity of a message before forwarding it through `NotiyiMsgChan`

This datatype mapper will, of course, own a mutex that guarantees no race condition between the two protocols.

NotiyiMsgChan

Thanks to the `DatatypeMapper`, the `P2P` protocol can recognize which kind of message it should propagate. When it receives a new message, it will check if this message type was requested by any module by reading the datatype mapper and if the message is valid. If that is the case, it sends this message through `NotiyiMsgChan` (marked in orange in Figure 1). `API` protocol also has a running goroutine that constantly listens to this channel. It can get those messages from P2P and send corresponding notification messages to the module requesting them.

1.2 Security

We changed our security mechanism according to our midterm feedback. We integrated `POW` in every gossip message. The hardness of this challenge is defined in the config file. Depending on the security requirement of the project, we can adjust the hardness, also known as the number of leading zeros, accordingly.

2 Software Documentation

2.1 Dependency

2.2 How to install and run

2.3 Known Issues

3 Future Work

4 Workload distribution

Bibliography

[1] "Go channel". In: (). Accessed: 2024-09-08. URL: https://go.dev/tour/concurrency/2.