CHAIR OF NETWORK ARCHITECTURES AND SERVICES

TECHNICAL UNIVERSITY OF MUNICH

Miterm Report

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

1 Change of assumptions

2 Architecture of your module

The module gossip has been developed using the Golang programming language. It relies heavily on Go features like goroutines, Go channels, and multiple Go libraries.

2.1 The whole picture

As the specification requires, the gossip module runs as two independent protocols: one API protocol and one P2P protocol. However, these two protocols share some data to fulfill the functionality of the module.

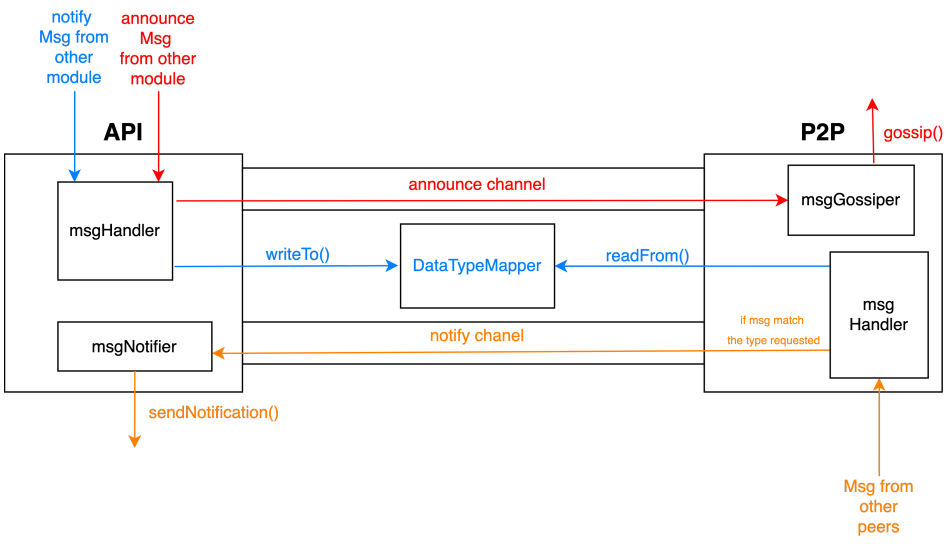

Figure 1: Structure of the gossip module

1. Announce messages Go channel

To make the announce functionality work, we need an **announce Go channel**[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.

```go
announceMsgChan := make(chan enum.AnnounceMsg)
```

[1]Marked in red in Figure 1

2. Datatype mapper

To make the notify functionality work, we need a **datatype mapper**[2] 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. This datatype mapper will, of course, own a mutex that guarantees there is no race condition between the two protocols.

```go
type DatatypeMapper struct {
    mutex sync.RWMutex
    data  map[net.Addr]map[enum.Datatype]bool
}
```

3. Notify messages Go channel

Thanks to the datatype mapper, 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. If that is the case, it sends this message through **notify message Go channel**[3]. 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.

2.2 API

The API is designed to facilitate a Gossip-based protocol in a distributed system, leveraging Go's robust features for concurrency and networking. At its core, the Server listens on a specified TCP address for incoming connections, using Go's net package to manage network communications. Once a connection is accepted, the Server hands it off to a Handler, which processes messages according to their type.

The Handler utilizes a custom logger for monitoring and error reporting, enhancing the system's reliability and debuggability. It reads incoming messages, verifies their size and type using the bytes and encoding/binary packages, and then routes them to the appropriate handler functions. These functions handle specific message types such as announcements or notifications, updating the datatypeMapper, or sending messages to the announceMsgChan channel as necessary.

This seamless interaction between the Server and Handler ensures the system can efficiently process and route messages, maintaining data integrity and system state across distributed nodes. The use of Go's concurrency primitives, like goroutines and channels, allows the API to handle multiple connections simultaneously, making it scalable and robust for real-time, distributed communication.

[2]Marked in blue in Figure 1
[3]Marked in orange in Figure 1

2.3 P2P

TODO: Trung

3 Security Measures

4 Specification of the peer-to-peer protocol that will be implemented

5 Future Work

6 Workload distributed

7 Effort spent for the project