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.

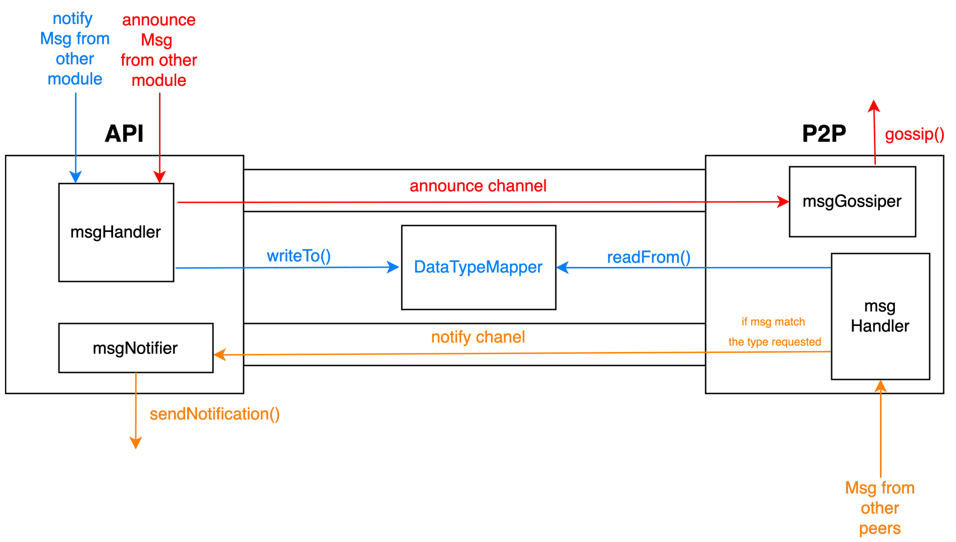

Figure 1: Structure of the gossip module

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.

```
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

Bootstrapping strategy:

Bootstrapping service is one of the important components of a P2P network that helps newly joined Node to get initial knowledge on current active peers in the network. In our current implementation, we use a static bootstrapping method to ensure that new nodes can join the network and connect to existing peers. The bootstrapping process involves the following steps:

1. **Registration**: When a new node starts, it registers itself with the bootstrapper server. The server maintains a list of all registered peers.
2. **Fetching Initial Peers**: After registration, the new node fetches an initial list of peers from the bootstrapper server. This list is used to establish initial connections and begin participating in the gossip protocol.

For security, we implement POW protocol when a node register to bootstrap (refer to section....)

Gossip Node:

The `GossipNode` implementation forms a critical component of our P2P VoIP application. It is designed to manage peer-to-peer communication, allowing nodes to join and leave the network, and to disseminate information efficiently. This report provides a detailed overview of the current functionalities and properties of the `GossipNode`.

Functionalities

The `GossipNode` implementation offers several key functionalities. Firstly, when a node starts, it joins the network by registering with the bootstrapper server to announce its presence. After this registration, the node fetches an initial list of peers from the bootstrapper server to begin establishing connections. When a node leaves the network, it announces its departure to its known peers, ensuring their peer lists are updated accordingly.

Additionally, the node spreads information using a gossip protocol, which involves disseminating data such as new peers joining or leaving, and application-related messages, to a random subset of known peers. The node processes incoming gossip messages and forwards them to other peers, ensuring widespread distribution of information. For peer list management, nodes periodically exchange peer lists to maintain up-to-date knowledge of the network. The node maintains a list of known peers, dynamically adding new peers and removing inactive ones as necessary.

Properties

The `GossipNode` has several important properties. Fanout refers to the number of peers a node gossips to during each gossip round. Currently, the fanout is set to 2, meaning each node sends gossip messages to 2 randomly selected peers. The gossip interval is the interval at which periodic gossip messages are sent; it is currently set to 5 seconds, ensuring regular dissemination of information.

The message cache stores recently seen messages to prevent redundant processing and forwarding. It uses a map data structure to store message identifiers, ensuring quick lookup and efficient memory usage. The peers list is a record of known peers that a node maintains to facilitate communication. This list is updated dynamically as the node learns about new peers or detects the departure of existing peers. Finally, the bootstrap URL is the address of the bootstrapper server used for initial registration and fetching peers. Nodes use this URL to register themselves and to obtain an initial list of peers when they join the network.

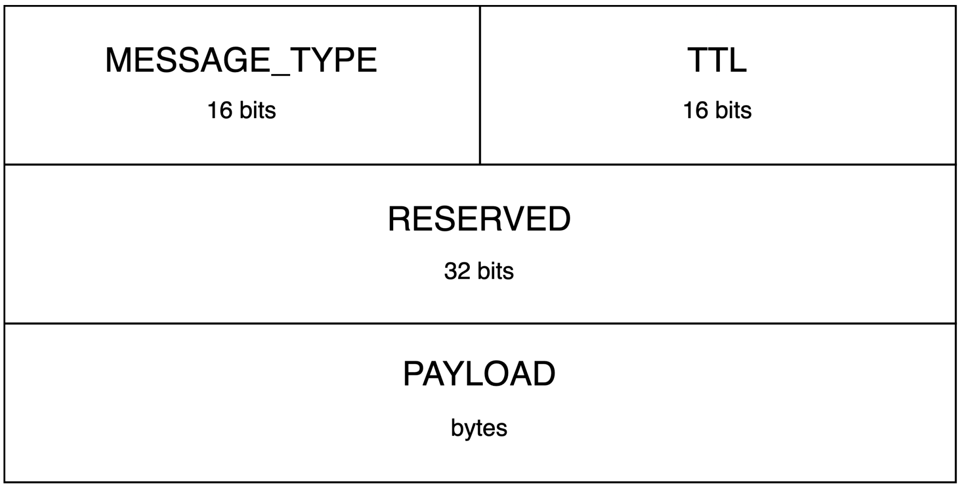

Figure 2: Gossip Message

3 Security Measures

3.1 Bootstrapper Proof-Of-Work (POW):

For security of Bootstrapper, we implement POW mechanism for the registering process. To avoid sybil attack, we define target difficulty, which is the number of leading zeros of a hash that a peer should calculate in order to register successfully. Details of the process are belows:

BOOTSTRAP INIT

When a new peer opens a connection to Bootrapping Server, the server sends an INIT message to the peer. This message contains a challenge and its target difficulty which this peer needs to use to calculate the hash.

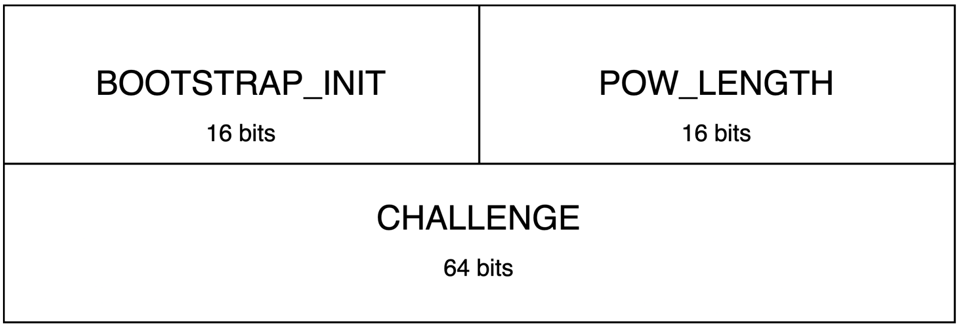

Figure 3: Bootstrap Init Message

BOOTSTRAP REGISTER

The peer calulate the SHA256 hash of the concatination of nonce (random number) and the challenge. If the resulting hash has the first n bits set to zeros (wheras n = target difficulty), then the peer can register by sending the BOOTSTRAP REGISTER message. If not, the nonce has to be changed to another random value and retried until one is found which gives one of the required SHA256 values. Note that SHA256 is considered a pseudo-random function. This means that there is no other possible way to find such values apart from randomly retrying them.

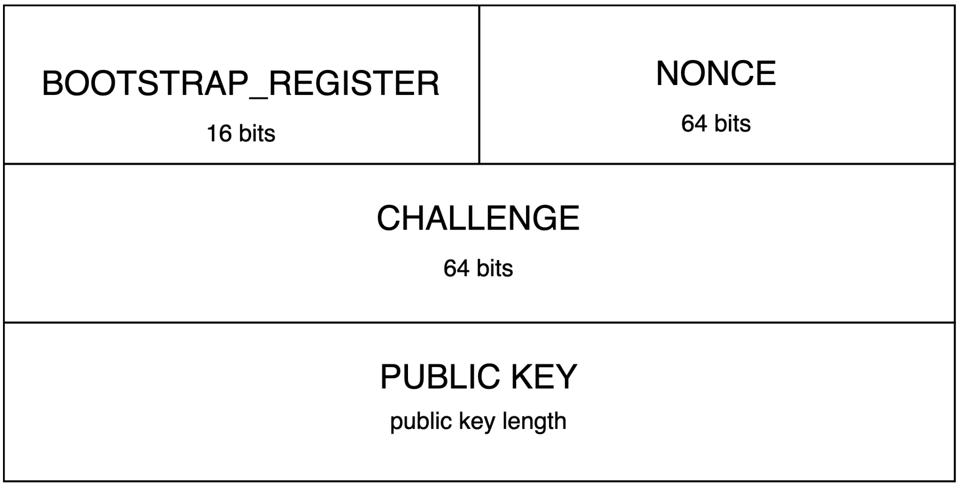

Figure 4: Bootstrap Register Message

BOOTSTRAP SUCCESS

If registeration is successful, meta data of the new peer got saved in local memory of the bootstrapping service and a BOOTSTRAP SUCCESS message is returned to the peer. The message also contains initial peer list.

BOOTSTRAP_REGISTER 16 bits	NONCE 64 bits
CHALLENGE 64 bits	

Figure 5: Bootstrap Success Message

BOOTSTRAP FAILURE

If registeration is unsuccesful, meta data of the new peer got saved in local memory of the bootstrapping service and a BOOTSTRAP FAILURE message is returned to the peer with error message.

BOOTSTRAP_FAILURE 16 bits	SIZE 16 bits
FAILURE_REASON bytes	

Figure 6: Bootstrap Failure Message

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

4.1 Message Types

500-503

504: BOOTSTRAP INIT

505: BOOTSTRAP REGISTER

506: BOOTSTRAP SUCCESS

507: BOOTSTRAP FAILURE

508: place holder

509: place holder

510: DATA_SPREAD

511: PEER_ANNOUNCE

512: PEER_LEAVE

513: PEER_LIST_REQUEST

514: PEER_LIST_REQUEST

5 Future Work

6 Workload distributed

7 Effort spent for the project