```go
/*
 -----------------------------------------------------------------------------------
 Lab         : 03
 File        : network.go
 Authors     : François Burgener - Tiago P. Quinteiro
 Date        : 10.12.19

 Goal        :  Network layer for the algorithm of chang and robert (bully)
 -----------------------------------------------------------------------------------
*/

package network

import (
    "bufio"
    "bytes"
    "log"
    "net"
    "prr-labo3/labo3/config"
    "prr-labo3/labo3/network/messages"
    "prr-labo3/labo3/utils"
    "time"
)

type Manager interface {
    SubmitNotification(notifMap map[uint16]uint16)
    SubmitResult(id uint16, resultMap map[uint16]bool)
}

type Network struct {
    id uint16
    N  uint16
    manager Manager
    Debug bool
}

/**
 * Method to init our Network
 * @param id of the processus
 * @param N number of processus
 */
func (n *Network) Init(id uint16, N uint16, manager Manager) {
    log.Println("Network : Initialization of the network")
    n.id = id
    n.N = N
    n.manager = manager

    go func() {
        n.initServ()
    }()
}

/**
 * Method to init our udp server
 */
func (n *Network) initServ() {
    addr := utils.AddressByID(n.id)
    conn, err := net.ListenPacket("udp", addr)
    if err != nil {
        log.Fatal("Network error: Initialisation failed",err)
    }
    defer conn.Close()

    n.handleConn(conn)
}

/**
 * Method to init our Network
 * @param id of the processus
 * @param N number of processus
 */
func (n *Network) handleConn(conn net.PacketConn) {
    buf := make([]byte, 1024)
    for {
        l, cliAddr, err := conn.ReadFrom(buf)
        if err != nil {
            log.Fatal("Network error: Reading error ",err)
        }
        s := bufio.NewScanner(bytes.NewReader(buf[0:l]))
        for s.Scan() {
            buf := s.Bytes()
            n.emitACK(conn,cliAddr)
            n.decodeMessage(buf)
        }
    }
}

/**
 * Method to emit a notification
 * @param _map with all processus and this aptitude
 */
```

```go
func (n *Network) EmitNotif(_map map[uint16]uint16){
    notif := messages.MessageNotif{_map}
    msg := utils.EncodeMessageNotif(notif)
    buf := utils.InitMessage([]byte(config.NotifMessage),msg)
    n.emit(buf)

    if n.Debug{
        log.Println("Network : Emit notification : ",_map)
    }
}

/**
 * Method to emit a result
 * @param id processus who is elected
 * @param _map of processus who send the result
 */
func (n *Network) EmitResult(id uint16,_map map[uint16]bool){
    result := messages.MessageResult{id,_map}
    msg := utils.EncodeMessageResult(result)
    buf := utils.InitMessage([]byte(config.ResultMessage),msg)
    n.emit(buf)

    if n.Debug{
        log.Println("Network : Emit result : id-",id," map-",_map)
    }
}

/**
 * Method to emit a ACK
 * @param conn conn of the client
 * @param cliAddr address of the client
 */
func (n *Network) emitACK(conn net.PacketConn, cliAddr net.Addr) {
    ack := messages.Message{n.id}
    msg := utils.EncodeMessage(ack)
    buf := utils.InitMessage([]byte(config.AckMessage),msg)

    if _, err := conn.WriteTo(buf, cliAddr); err != nil {
        log.Fatal("Network error: Writing error ",err)
    }

    if n.Debug{
        log.Println("Network : Emit ACK")
    }
}

/**
 * Method to emit an ECHO
 * @param id of the processus we want to send
 * @return true if we received an ACK, false otherwise
 */
func (n *Network) EmitEcho(id uint16) bool {
    channel := make(chan bool, 1) // channel to know if we received an ACK
    echo := messages.Message{n.id}
    msg := utils.EncodeMessage(echo)
    buf := utils.InitMessage([]byte(config.EchoMessage),msg)

    if n.Debug {
        log.Println("Network : Emit ECHO : ",n.id)
    }

    go n.emitById(buf,id,channel)

    select {
    case <-channel: //We received an ACK
        return true
    case <-time.After(config.TIME_OUT): // Timeout
        log.Println("Network : Timeout")
        return false
    }
}

/**
 * Method to emit an message of our next processus (Id + 1) with we can we try another (id + 2) ect
 * @param msg we want to send
 */
func (n *Network) emit(msg []byte) {

    for i:= n.id; i < n.N + n.id; i++{

        id := (i + 1) % n.N // id of the next processus
        channel := make(chan bool, 1) // channel to know if we received an ACK
        receivedACK := false //Boolean to stop the loop if we received an ACK


        //Emit message to the next processus
        n.emitById(msg,id,channel)

        select {
        case receivedACK = <-channel: //We received an ACK
        case <-time.After(config.TIME_OUT): // Timeout
```

```go
            log.Println("Network : Timeout")
            continue
        }

        //If we received an ACK, we stop the loop
        if receivedACK{
            break
        }
    }
}

/**
 * Method to emit an message
 * @param msg we want to send
 * @param id of the processus we want to send
 * @param channel to say if we received ACK
 */
func (n *Network) emitById(msg []byte,id uint16, channel chan bool) {
    add := utils.AddressByID(id)
    addr,err := net.ResolveUDPAddr("udp",add)
    if err != nil {
        log.Printf("The processus %d is not alive ",id)
    }

    conn,err := net.DialUDP("udp",nil,addr)
    if err != nil {
        log.Println("Network error: Error dial", err.Error())
    }

    _, err = conn.Write(msg)
    if err != nil {
        log.Fatal("Network error: Writing error ",err)
    }

    go n.readACK(conn,channel)

}


/**
 * Method to read an ACK message
 * @param conn to read the ack
 * @param channel to say if we received ACK
 */
func (n *Network) readACK(conn net.Conn, channel chan bool){
    // Make a buffer to hold incoming data.
    buf := make([]byte, 1024)

    // Read the incoming connection into the buffer.
    l, err := conn.Read(buf)
    if err != nil {
        log.Println("Network error: Error reading", err.Error()) //TODO Check
    }

    s := bufio.NewScanner(bytes.NewReader(buf[0:l]))

    for s.Scan(){
        buf := s.Bytes()
        if string(buf[0:3]) == config.AckMessage{
            msg := utils.DecodeMessage(buf[3:])

            channel <- true

            if n.Debug{
                log.Println("Decode : ",string(buf[0:3]),"-",msg.Id)
            }
        }
    }
}

/**
 * Method to read decode a message
 * @param buf  array of byte we want to decode
 */
func (n *Network) decodeMessage(buf []byte) {

    _type := string(buf[0:3])

    switch _type {
    case config.EchoMessage:
        msg := utils.DecodeMessage(buf[3:])

        if n.Debug{
            log.Println("Decode",_type,"-",msg.Id)
        }
    case config.ResultMessage:
        msg := utils.DecodeMessageResult(buf[3:])

        if n.Debug{
            log.Println("Decode",_type,"-",msg.Id,"-",msg.Map)
        }

        n.manager.SubmitResult(msg.Id,msg.Map)
    case config.NotifMessage:
        msg := utils.DecodeMessageNotif(buf[3:])

        if n.Debug{
            log.Println("Decode",_type,"-",msg.Map)
        }

        n.manager.SubmitNotification(msg.Map)
    default:
        log.Println("Network: Incorrect type message !")
    }
}



```