```go
/*
 -----------------------------------------------------------------------------------
 Lab         : 02
 File        : network.go
 Authors     : François Burgener - Tiago P. Quinteiro
 Date        : 03.12.2019

 Goal        : Network layer for the algorithm of Carvalho et Roucairol
 -----------------------------------------------------------------------------------
*/

package network

import (
    "PRR-Labo2/labo2/config"
    "PRR-Labo2/labo2/utils"
    "bufio"
    "bytes"
    "io"
    "log"
    "net"
    "strconv"
)


/*************************************************
*                   INTERFACE                    *
*************************************************/
type Mutex interface {
    Req(stamp uint32, id uint16)
    Ok(stamp uint32, id uint16)
    Update(value uint32)
}

/*************************************************
*                   STRUCTURE                    *
*************************************************/
type Network struct {
    id uint16 //id of our processus
    nProc uint16 // Number of processus
    directory map[uint16]net.Conn // map of connection
    Done chan string // channel to say if the server initialisation is done
    mutex Mutex //Ref of our mutex
    Debug bool
}

/*************************************************
*                NETWORK METHOD                  *
*************************************************/

/**
 * Method of Network to send a MessageREQ message
 * @param stamp (logic clock) of the processus
 * @param id of the processus
 */
func (n *Network) REQ(stamp uint32, id uint16){
    msg := utils.InitMessage(stamp,n.id,[]byte(config.MessageREQ))
    //_, err := n.directory[id].Write(msg)
    mustCopy(n.directory[id], bytes.NewReader(msg))

    if n.Debug{
        log.Printf("Network: Send message type:%s stamp:%d id:%d \n", config.MessageREQ,stamp,id)
    }

}

/**
 * Method of Network to send a MessageREQ message
 * @param stamp (logic clock) of the processus
 * @param id of the processus
 */
func (n *Network) OK(stamp uint32, id uint16){
    msg := utils.InitMessage(stamp,n.id,[]byte(config.MessageOK))

    mustCopy(n.directory[id], bytes.NewReader(msg))

    if n.Debug{
        log.Printf("Network: Send message type:%s stamp:%d id:%d \n", config.MessageOK,stamp,id)
    }
}

/**
 * Method of Network to send a MessageUPDATE message
 * @param value to update
 */
func (n *Network) UPDATE(value uint32){
    for i:=0; i < len(n.directory) + 1; i++{
        if i != int(n.id){
            msg := utils.InitMessageUpdate(value,[]byte(config.MessageUPDATE))
            mustCopy(n.directory[uint16(i)], bytes.NewReader(msg))

            if n.Debug{
                log.Printf("Network: Send message Update P%d value: %d",i,value)
            }
        }
    }
}


/**
 * Method to init the server and get all connection between processus
 * @param id of the processus
 * @param N number of processus
 * @param mutex ref to mutex
 */
func (n *Network) Init(id uint16,N uint16, mutex Mutex) {
    log.Printf("Network: Initialisation ")
    n.directory = make(map[uint16]net.Conn,N)
    n.Done = make(chan string)
    n.mutex = mutex
    n.id = id
    n.nProc = N

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

    <- n.Done
}

// PRIVATE methods ---------------------------------

/**
 * Method to init all dial connection
 */
func (n *Network) initAllConn() {
    for i:=uint16(0) ; i < n.nProc; i++ {
        if i != uint16(n.id) {
            n.initConn(i)
        }
    }
}

/**
 * Method to init a dial connection
 * @param i id of the processus we want to connect
 */
func (n *Network)initConn(i uint16) {
    addr := utils.AddressByID(uint16(i))
    conn, err := net.Dial("tcp", addr)

    if err != nil {
        log.Printf("Network error : Connection refused with P%d",i)
    }else{
        n.directory[uint16(i)] = conn
        _, err := conn.Write([]byte(strconv.Itoa(int(n.id))))
        if err != nil{
            log.Fatal("Network error: Writing error:", err.Error())
        }

        if n.Debug{
            log.Printf("Network : Dial Connection between P%d and P%d\n", n.id, i)
        }

        go n.handleConn(conn)
    }
}


/**
 * Method to init a new Network
 */
func (n *Network) initServ(){
    addr := utils.AddressByID(n.id)
    listener, err := net.Listen("tcp", addr)
    if err != nil {
        log.Fatal("Network error: Listen error:", err.Error())
    }

    defer listener.Close()

    for {

        if len(n.directory) == int(n.nProc-1) {
            n.Done <- "done"
        }

        conn, err := listener.Accept()
        if err != nil {
            log.Fatal("Network error: Listen accept error:", err.Error())
        }


        tmp := make([]byte,128)
        l, err := conn.Read(tmp)
        if err != nil {
            log.Fatal("Network error: Reading error:", err.Error())
        }
        str := string(tmp[0:l])
        idConn, err := strconv.Atoi(str)
        if err != nil {
            log.Fatal("Network error: Cannot take the id of the processus:", err.Error())
        }

        log.Println("Network: Serv Connection between P" + strconv.Itoa(int(n.id)) + " and P" + strconv.Itoa(idConn))
        n.directory[uint16(idConn)] = conn

        go n.handleConn(conn)
    }
}

/**
 * Method to read message
 */
func (n *Network)handleConn(conn net.Conn) {
    for {
        // Make a buffer to hold incoming data.
        buf := make([]byte, 32)

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

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

        for s.Scan(){
            n.decodeMessage(s.Bytes())
        }

    }
}

func (n *Network) decodeMessage(bytes []byte) {

    _type := string(bytes[0:3])
    var stamp uint32
    var id uint16
    var value uint32

    if _type == config.MessageUPDATE {
        value = utils.ConverByteArrayToUint32(bytes[3:7])

        if n.Debug{
            log.Printf("Network: Decoded message type:%s value:%d",_type,value)
        }

    }else if _type == config.MessageOK || _type == config.MessageREQ {
        stamp = utils.ConverByteArrayToUint32(bytes[3:7])
        id = utils.ConverByteArrayToUint16(bytes[7:9])

        if n.Debug{
            log.Printf("Network: Decoded message type:%s stamp:%d id:%d",_type,stamp,id)
        }

    }


    switch _type {
    case config.MessageREQ:
        n.mutex.Req(stamp,id)
    case config.MessageOK:
        n.mutex.Ok(stamp,id)
    case config.MessageUPDATE:
        n.mutex.Update(value)
    default:
        log.Println("Network: Incorrect type message !")
    }
}

func mustCopy(dst io.Writer, src io.Reader) {
    if _, err := io.Copy(dst, src); err != nil {
        log.Fatal(err)
    }
}
```