Ex04 Middleware

Setup

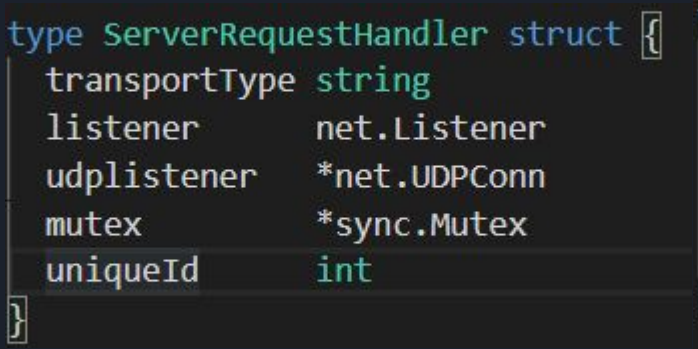
- Processador i7
- 8GB de RAM
- Windows 10 64 bits
- SSD
- Wi-fi: ligada
- VMWare - Ubuntu
  - 2Gb de ram
  - 1 cpu

Aplicação

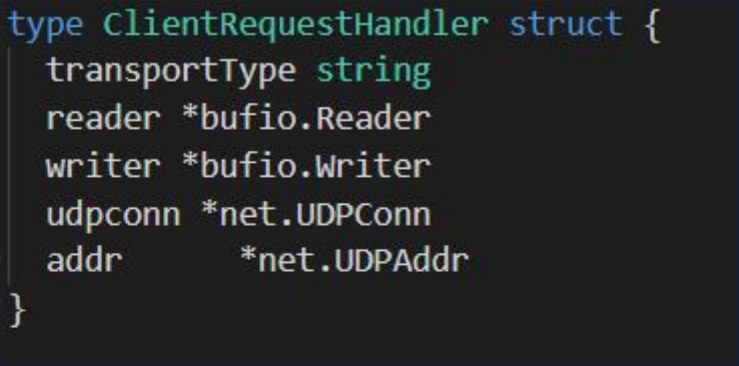
- Vanets
- Carros se registram na RSU mais proxima informando a ‘lane’ que eles estão. Podendo sempre registrar quando mudam de ‘lane’
- Caso um carro avise de algum evento naquela ‘lane’, todos os carros que estao naquela lane recebem o evento, e tomam alguma ação (Freiar, mudar de lane, mudar de rota, etc)

Experimento - Definição

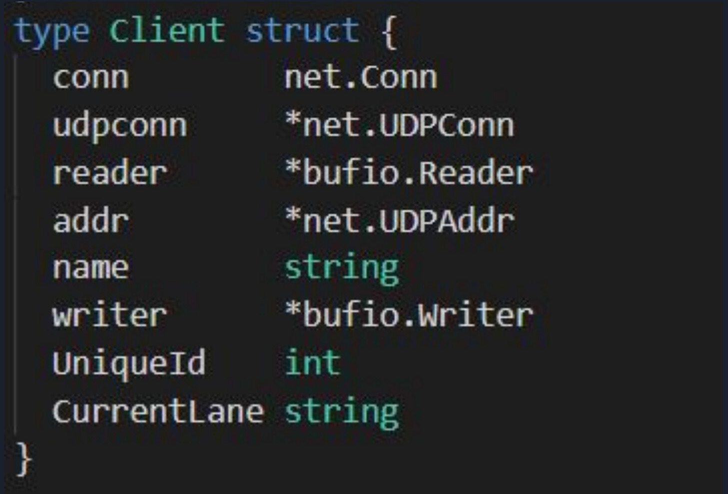
- Não abro e fecho a conexão para cada request
- Envio de uma mensagem



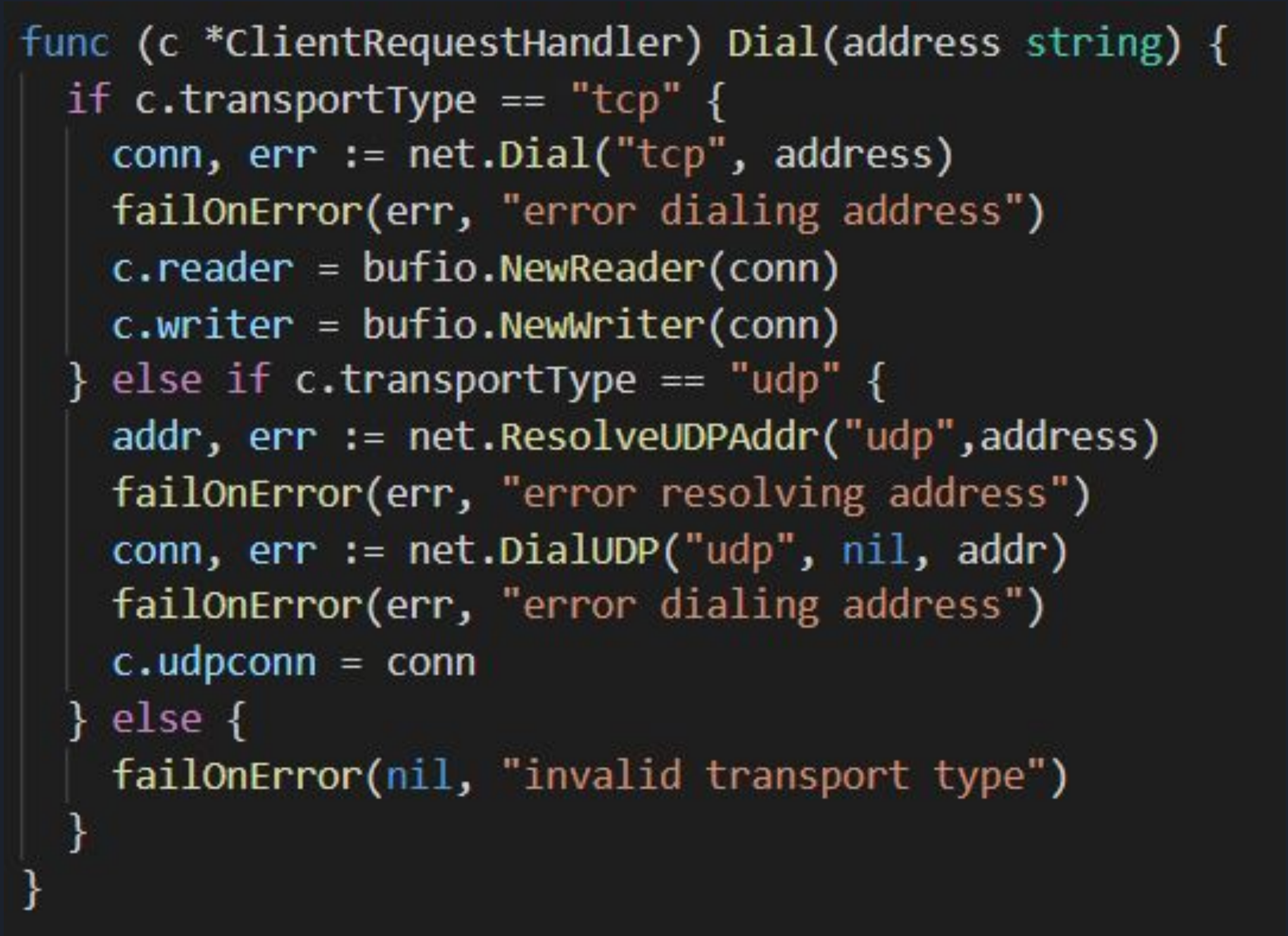
```go
type ServerRequestHandler struct {
	transportType string
	listener      net.Listener
	udplistener   *net.UDPConn
	mutex         *sync.Mutex
	uniqueId      int
}
```



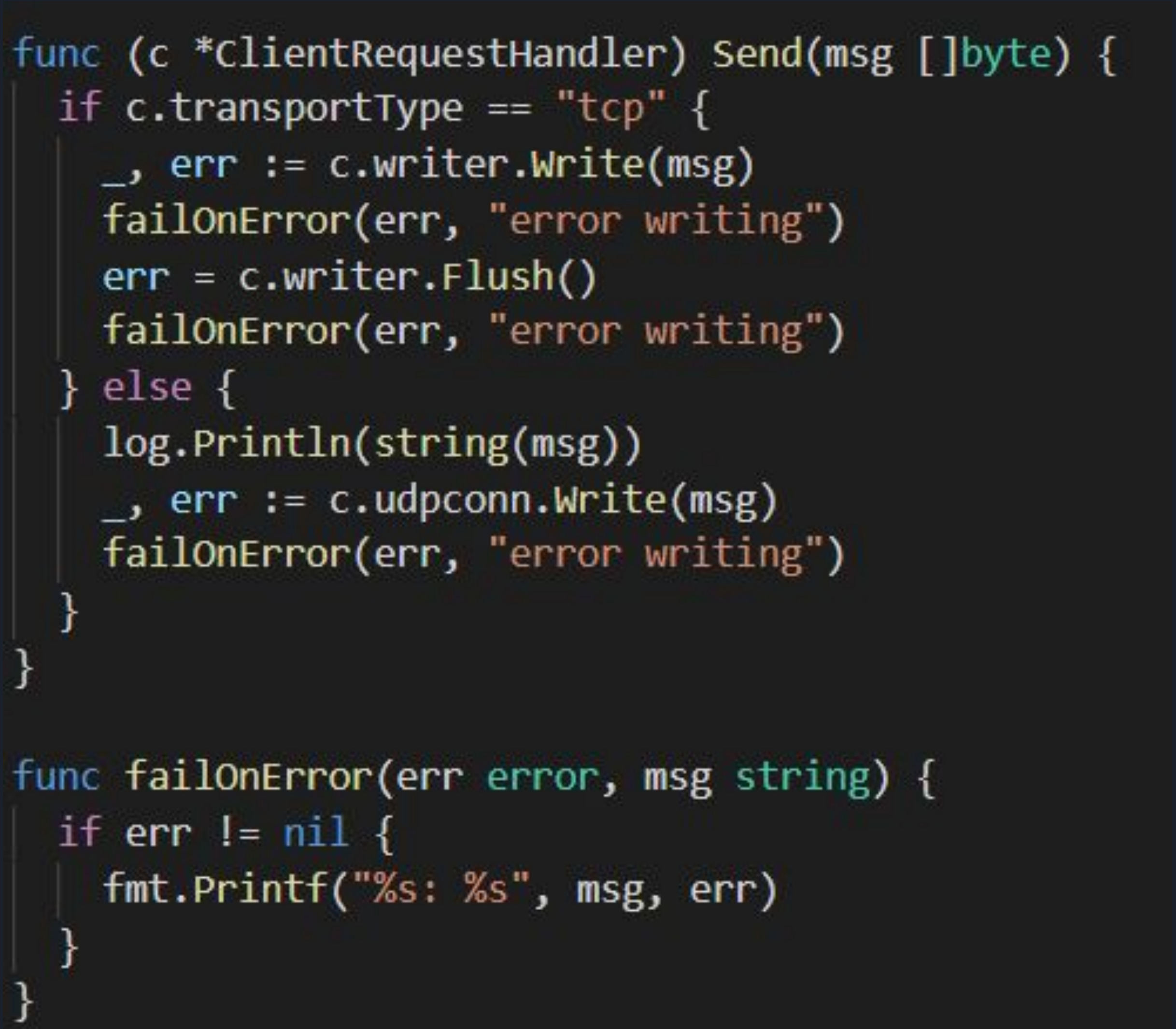
```go
type ClientRequestHandler struct {
	transportType string
	reader *bufio.Reader
	writer *bufio.Writer
	udpconn *net.UDPConn
	addr      *net.UDPAddr
}
```



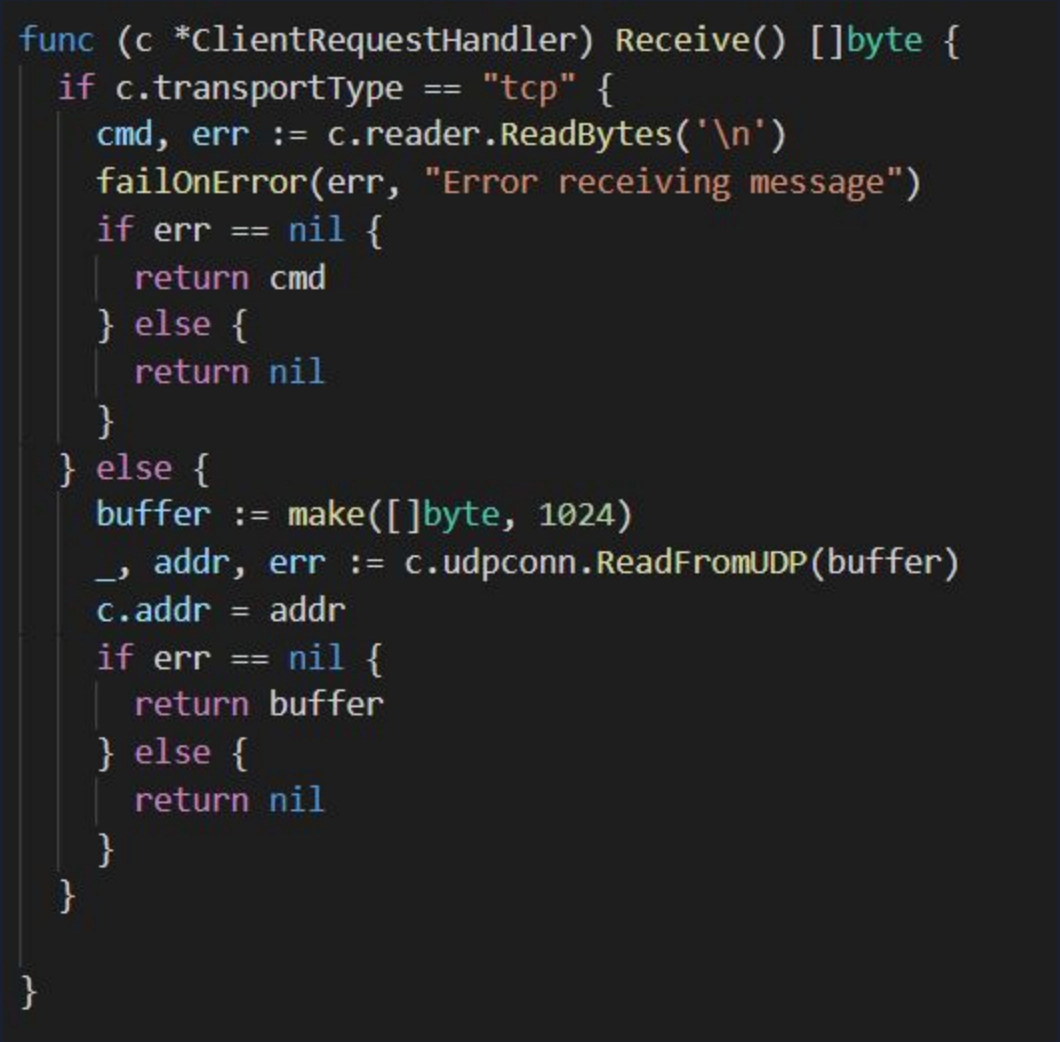
```go
type Client struct {
	conn        net.Conn
	udpconn     *net.UDPConn
	reader      *bufio.Reader
	addr        *net.UDPAddr
	name        string
	writer      *bufio.Writer
	UniqueId    int
	CurrentLane string
}
```



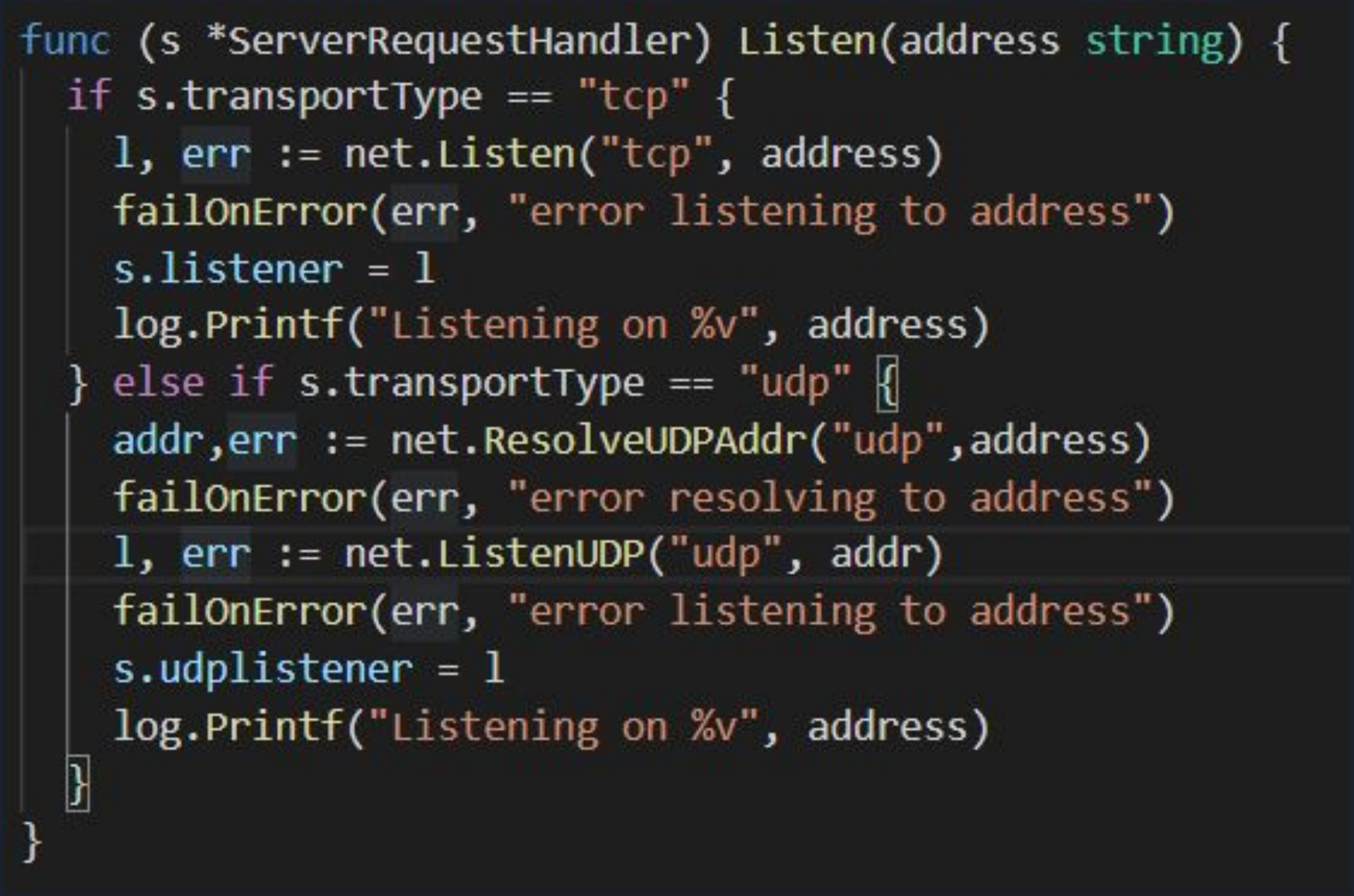
```go
func (c *ClientRequestHandler) Dial(address string) {
  if c.transportType == "tcp" {
    conn, err := net.Dial("tcp", address)
    failOnError(err, "error dialing address")
    c.reader = bufio.NewReader(conn)
    c.writer = bufio.NewWriter(conn)
  } else if c.transportType == "udp" {
    addr, err := net.ResolveUDPAddr("udp",address)
    failOnError(err, "error resolving address")
    conn, err := net.DialUDP("udp", nil, addr)
    failOnError(err, "error dialing address")
    c.udpconn = conn
  } else {
    failOnError(nil, "invalid transport type")
  }
}
```



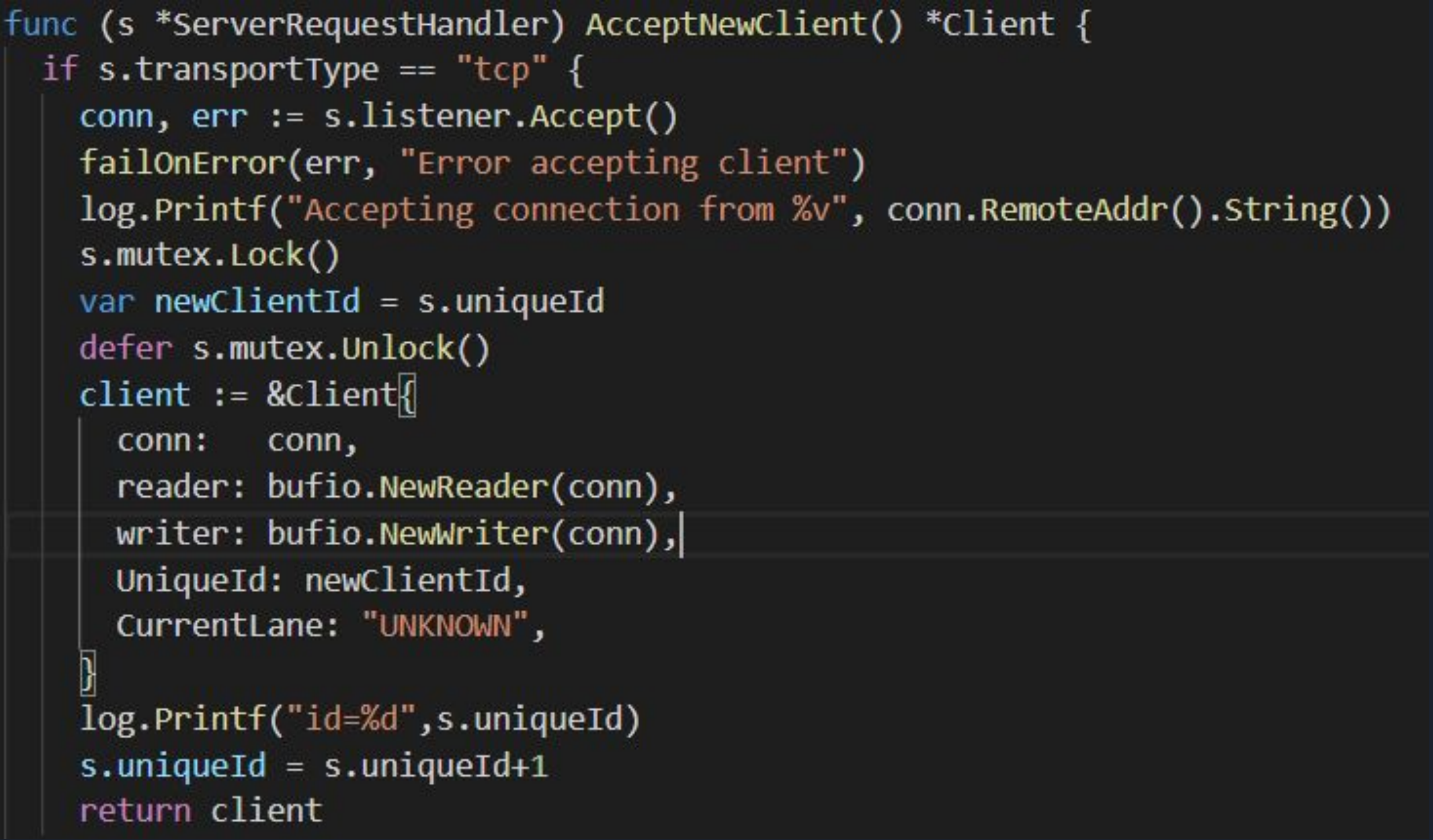
```go
func (c *ClientRequestHandler) Send(msg []byte) {
  if c.transportType == "tcp" {
    _, err := c.writer.Write(msg)
    failOnError(err, "error writing")
    err = c.writer.Flush()
    failOnError(err, "error writing")
  } else {
    log.Println(string(msg))
    _, err := c.udpconn.Write(msg)
    failOnError(err, "error writing")
  }
}

func failOnError(err error, msg string) {
  if err != nil {
    fmt.Printf("%s: %s", msg, err)
  }
}
```



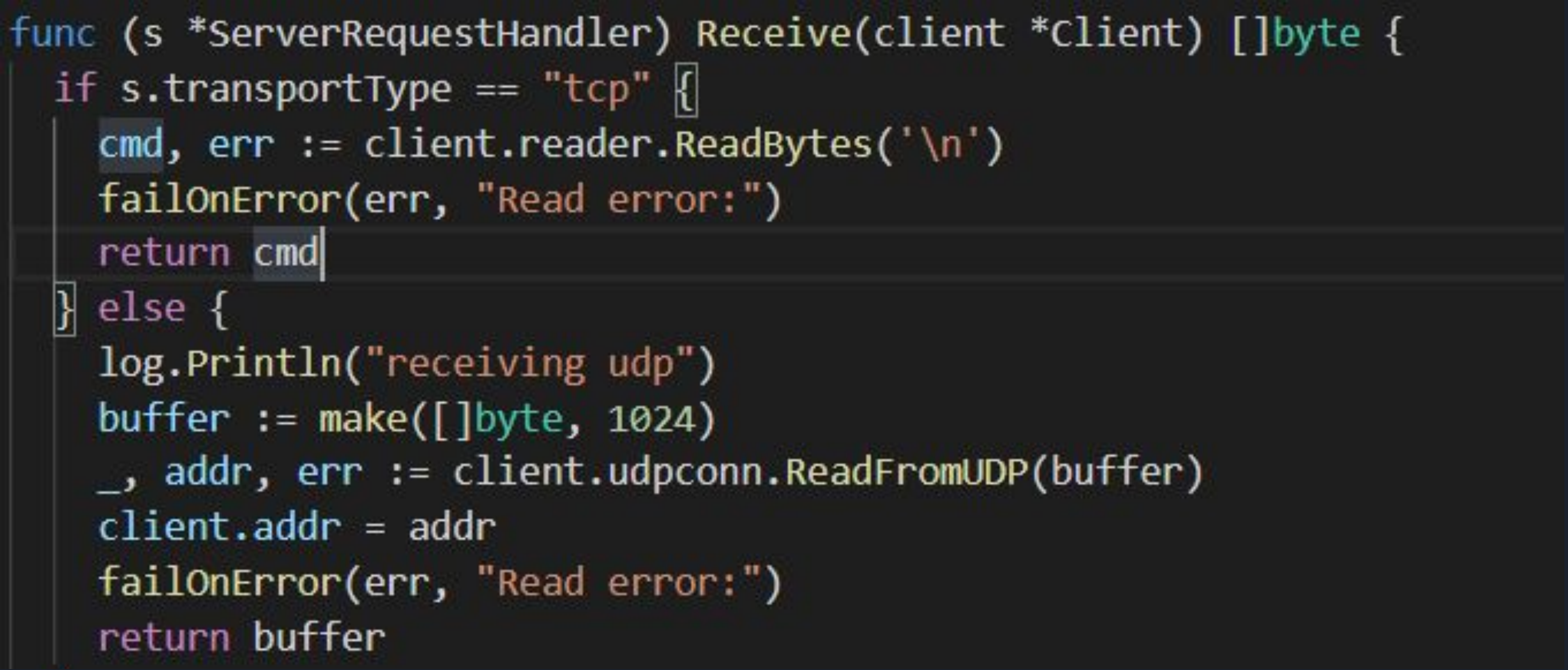
```go
func (c *ClientRequestHandler) Receive() []byte {
  if c.transportType == "tcp" {
    cmd, err := c.reader.ReadBytes('\n')
    failOnError(err, "Error receiving message")
    if err == nil {
      return cmd
    } else {
      return nil
    }
  } else {
    buffer := make([]byte, 1024)
    _, addr, err := c.udpconn.ReadFromUDP(buffer)
    c.addr = addr
    if err == nil {
      return buffer
    } else {
      return nil
    }
  }

}
```



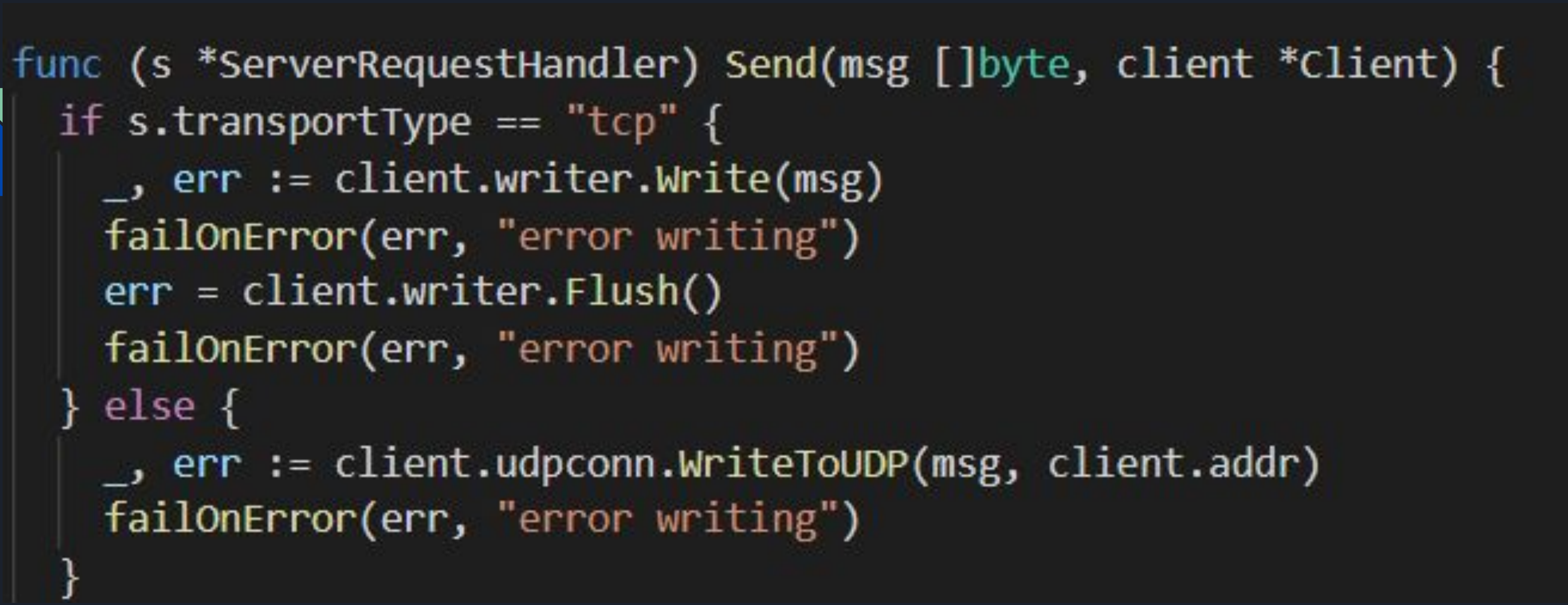
```go
func (s *ServerRequestHandler) Listen(address string) {
  if s.transportType == "tcp" {
    l, err := net.Listen("tcp", address)
    failOnError(err, "error listening to address")
    s.listener = l
    log.Printf("Listening on %v", address)
  } else if s.transportType == "udp" {
    addr,err := net.ResolveUDPAddr("udp",address)
    failOnError(err, "error resolving to address")
    l, err := net.ListenUDP("udp", addr)
    failOnError(err, "error listening to address")
    s.udplistener = l
    log.Printf("Listening on %v", address)
  }
}
```



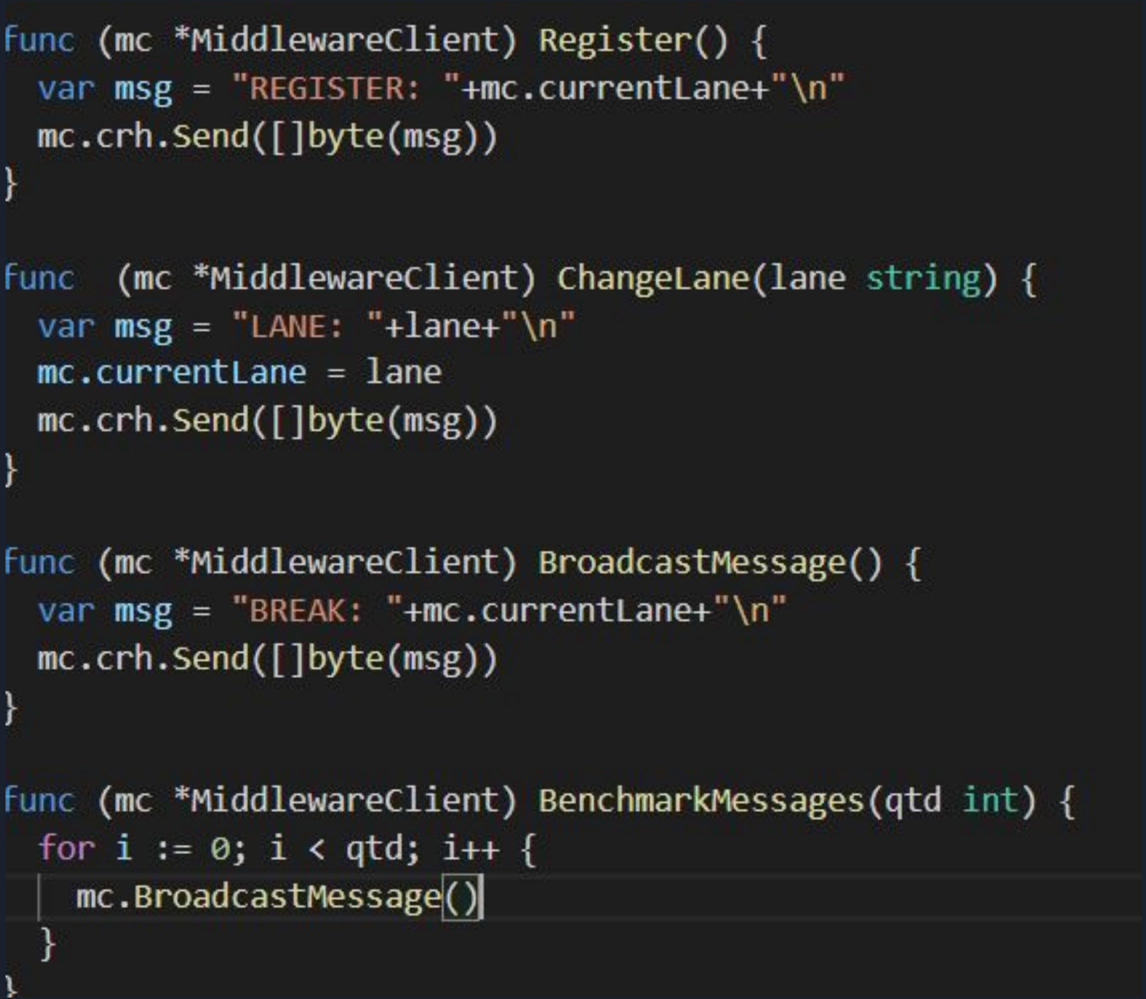
```go
func (s *ServerRequestHandler) AcceptNewClient() *Client {
  if s.transportType == "tcp" {
    conn, err := s.listener.Accept()
    failOnError(err, "Error accepting client")
    log.Printf("Accepting connection from %v", conn.RemoteAddr().String())
    s.mutex.Lock()
    var newClientId = s.uniqueId
    defer s.mutex.Unlock()
    client := &Client{
      conn:   conn,
      reader: bufio.NewReader(conn),
      writer: bufio.NewWriter(conn),
      UniqueId: newClientId,
      CurrentLane: "UNKNOWN",
    }
    log.Printf("id=%d",s.uniqueId)
    s.uniqueId = s.uniqueId+1
    return client
```



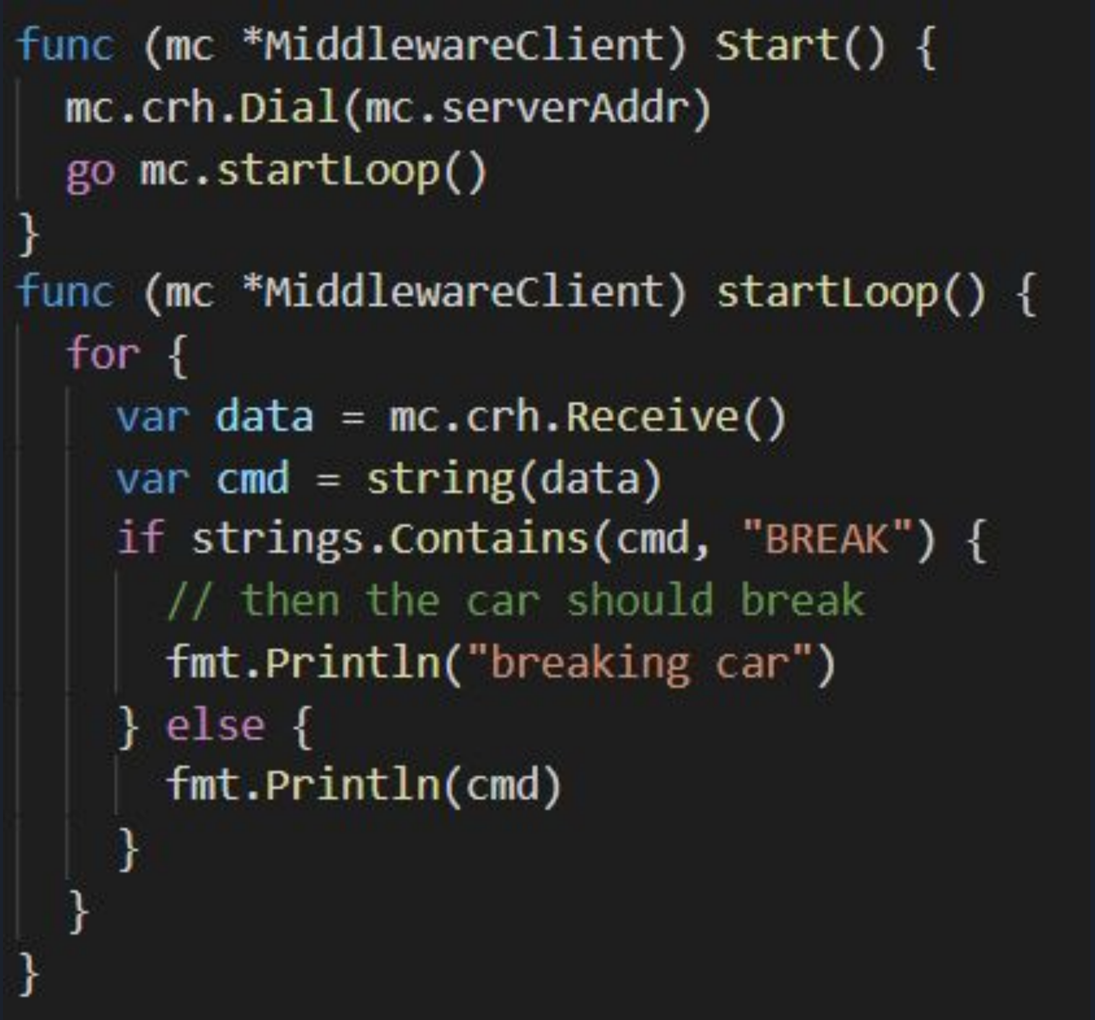
```go
func (s *ServerRequestHandler) Receive(client *Client) []byte {
  if s.transportType == "tcp" {
    cmd, err := client.reader.ReadBytes('\n')
    failOnError(err, "Read error:")
    return cmd
  } else {
    log.Println("receiving udp")
    buffer := make([]byte, 1024)
    _, addr, err := client.udpconn.ReadFromUDP(buffer)
    client.addr = addr
    failOnError(err, "Read error:")
    return buffer
```



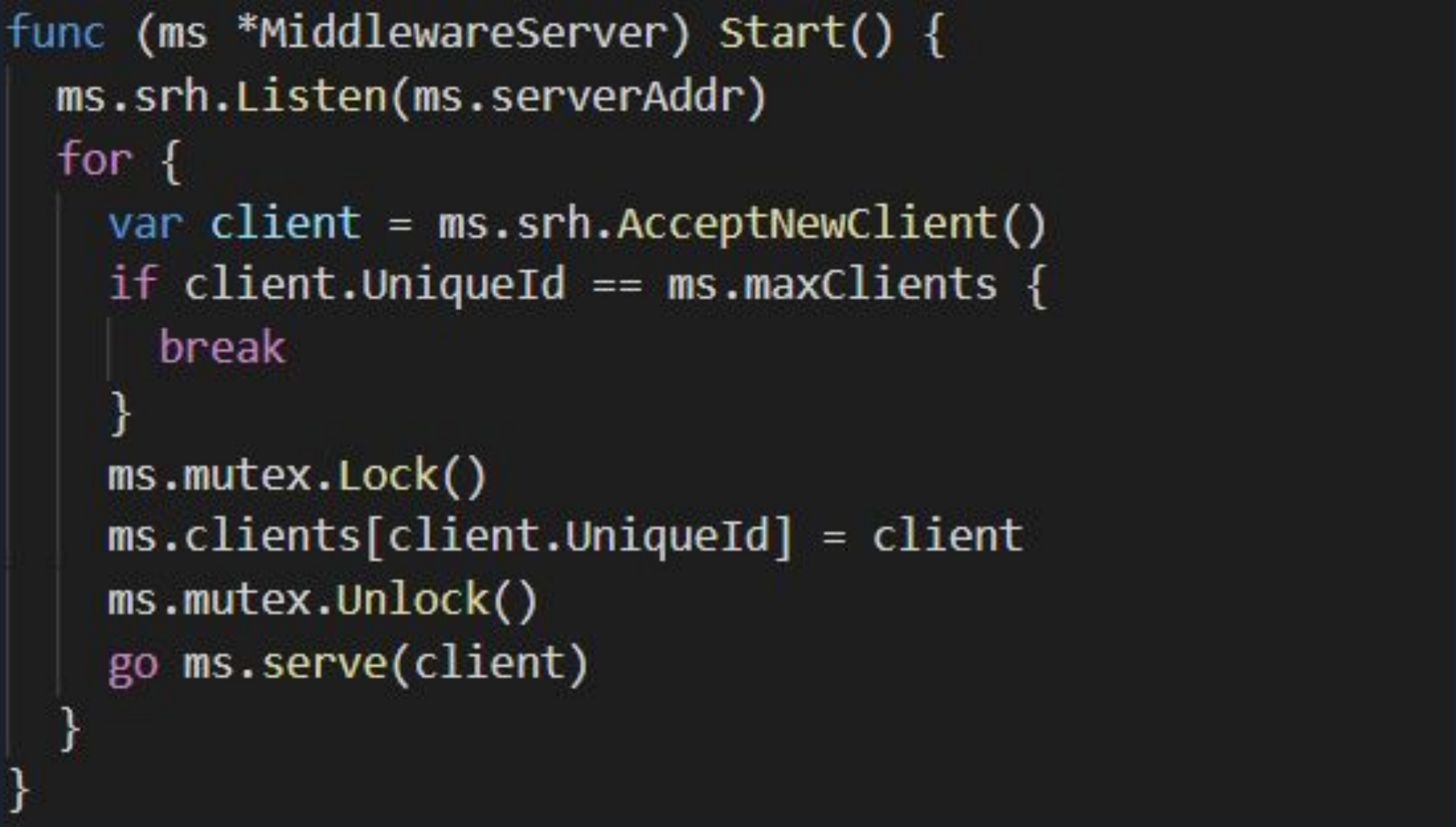
```go
func (s *ServerRequestHandler) Send(msg []byte, client *Client) {
  if s.transportType == "tcp" {
    _, err := client.writer.Write(msg)
    failOnError(err, "error writing")
    err = client.writer.Flush()
    failOnError(err, "error writing")
  } else {
    _, err := client.udpconn.WriteToUDP(msg, client.addr)
    failOnError(err, "error writing")
  }
```



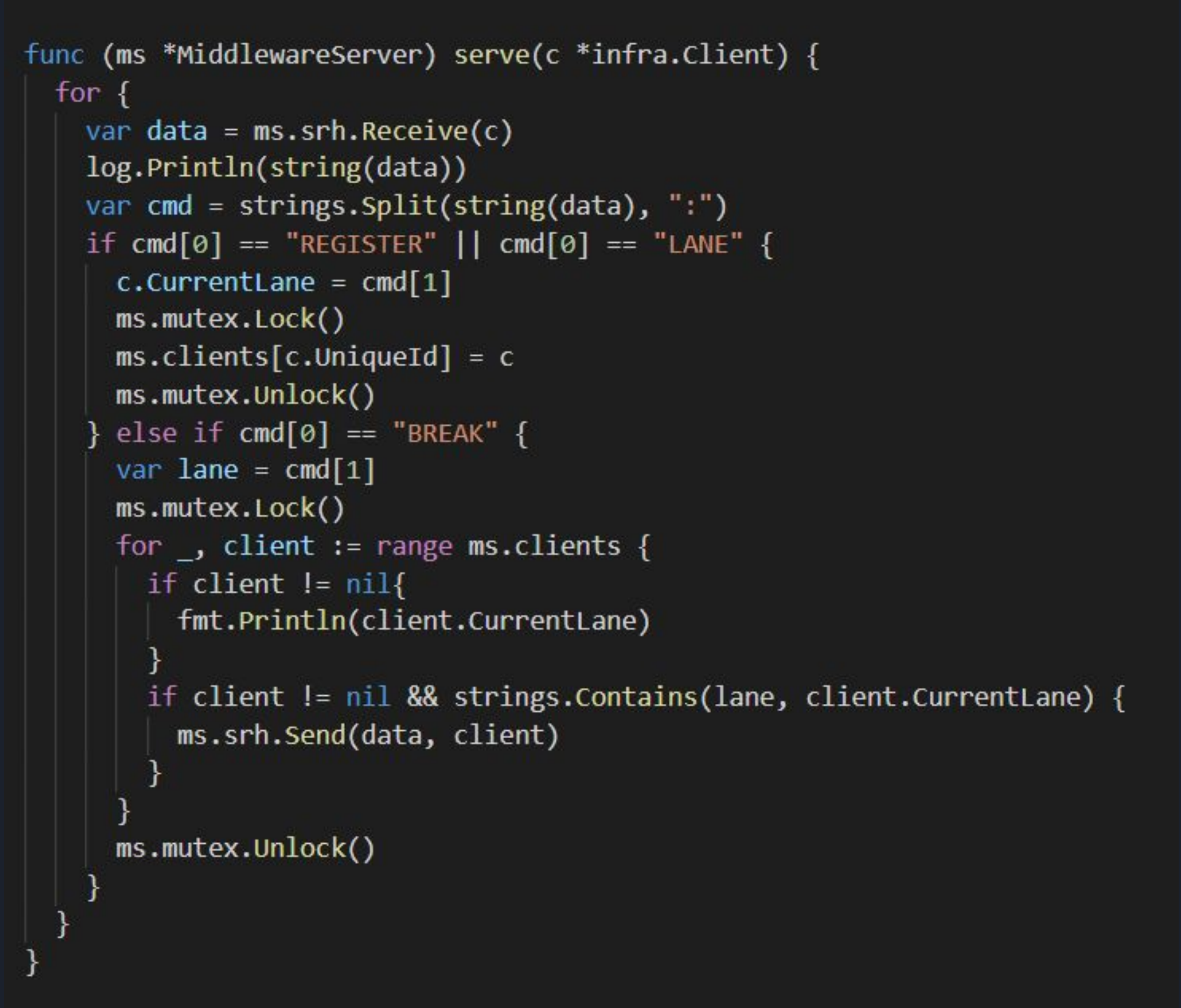
```
func (mc *MiddlewareClient) Register() {
  var msg = "REGISTER: "+mc.currentLane+"\n"
  mc.crh.Send([]byte(msg))
}

func  (mc *MiddlewareClient) ChangeLane(lane string) {
  var msg = "LANE: "+lane+"\n"
  mc.currentLane = lane
  mc.crh.Send([]byte(msg))
}

func (mc *MiddlewareClient) BroadcastMessage() {
  var msg = "BREAK: "+mc.currentLane+"\n"
  mc.crh.Send([]byte(msg))
}

func (mc *MiddlewareClient) BenchmarkMessages(qtd int) {
  for i := 0; i < qtd; i++ {
    mc.BroadcastMessage()
  }
}
```



```go
func (mc *MiddlewareClient) Start() {
  mc.crh.Dial(mc.serverAddr)
  go mc.startLoop()
}
func (mc *MiddlewareClient) startLoop() {
  for {
    var data = mc.crh.Receive()
    var cmd = string(data)
    if strings.Contains(cmd, "BREAK") {
      // then the car should break
      fmt.Println("breaking car")
    } else {
      fmt.Println(cmd)
    }
  }
}
```



```go
func (ms *MiddlewareServer) Start() {
	ms.srh.Listen(ms.serverAddr)
	for {
		var client = ms.srh.AcceptNewClient()
		if client.UniqueId == ms.maxClients {
			break
		}
		ms.mutex.Lock()
		ms.clients[client.UniqueId] = client
		ms.mutex.Unlock()
		go ms.serve(client)
	}
}
```



```go
func (ms *MiddlewareServer) serve(c *infra.Client) {
  for {
    var data = ms.srh.Receive(c)
    log.Println(string(data))
    var cmd = strings.Split(string(data), ":")
    if cmd[0] == "REGISTER" || cmd[0] == "LANE" {
      c.CurrentLane = cmd[1]
      ms.mutex.Lock()
      ms.clients[c.UniqueId] = c
      ms.mutex.Unlock()
    } else if cmd[0] == "BREAK" {
      var lane = cmd[1]
      ms.mutex.Lock()
      for _, client := range ms.clients {
        if client != nil{
          fmt.Println(client.CurrentLane)
        }
        if client != nil && strings.Contains(lane, client.CurrentLane) {
          ms.srh.Send(data, client)
        }
      }
      ms.mutex.Unlock()
    }
  }
}
```