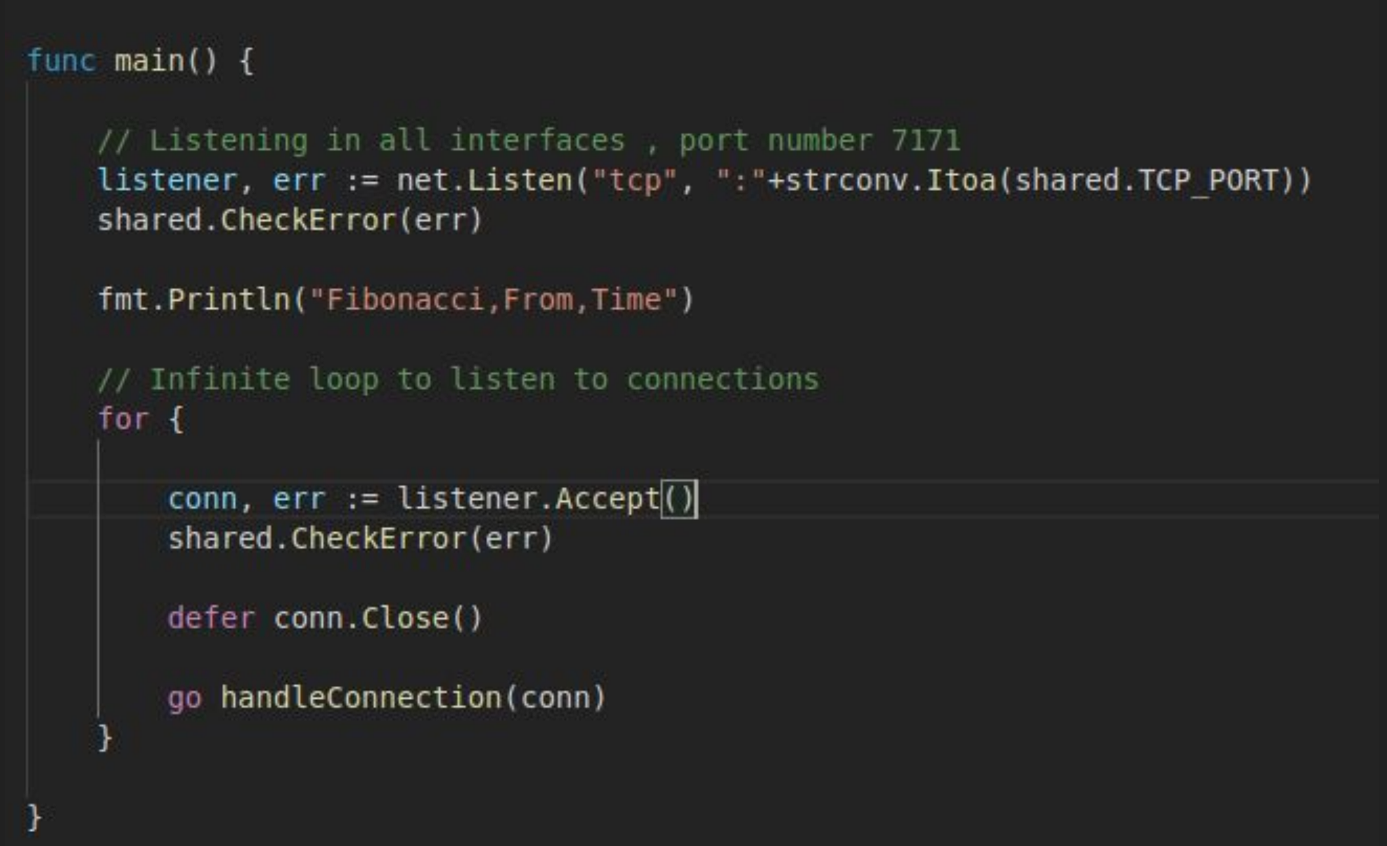
Exercício 1 - Socket

Aplicação e Avaliação de Desempenho

Código - Servidor TCP

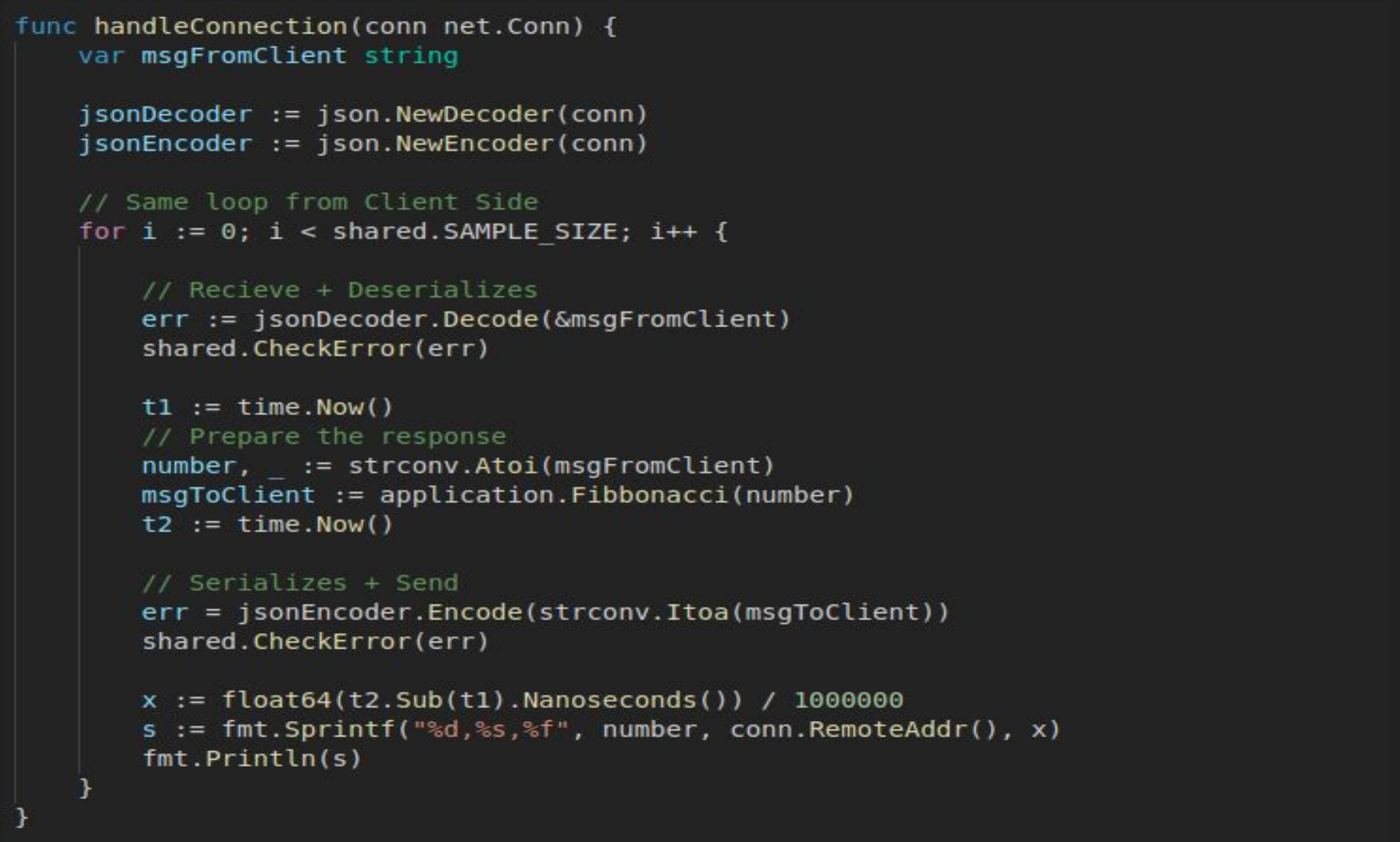
// Preparando servidor para escutar



```go
func main() {

    // Listening in all interfaces , port number 7171
    listener, err := net.Listen("tcp", ":"+strconv.Itoa(shared.TCP_PORT))
    shared.CheckError(err)

    fmt.Println("Fibonacci,From,Time")

    // Infinite loop to listen to connections
    for {

        conn, err := listener.Accept()
        shared.CheckError(err)

        defer conn.Close()

        go handleConnection(conn)
    }

}
```

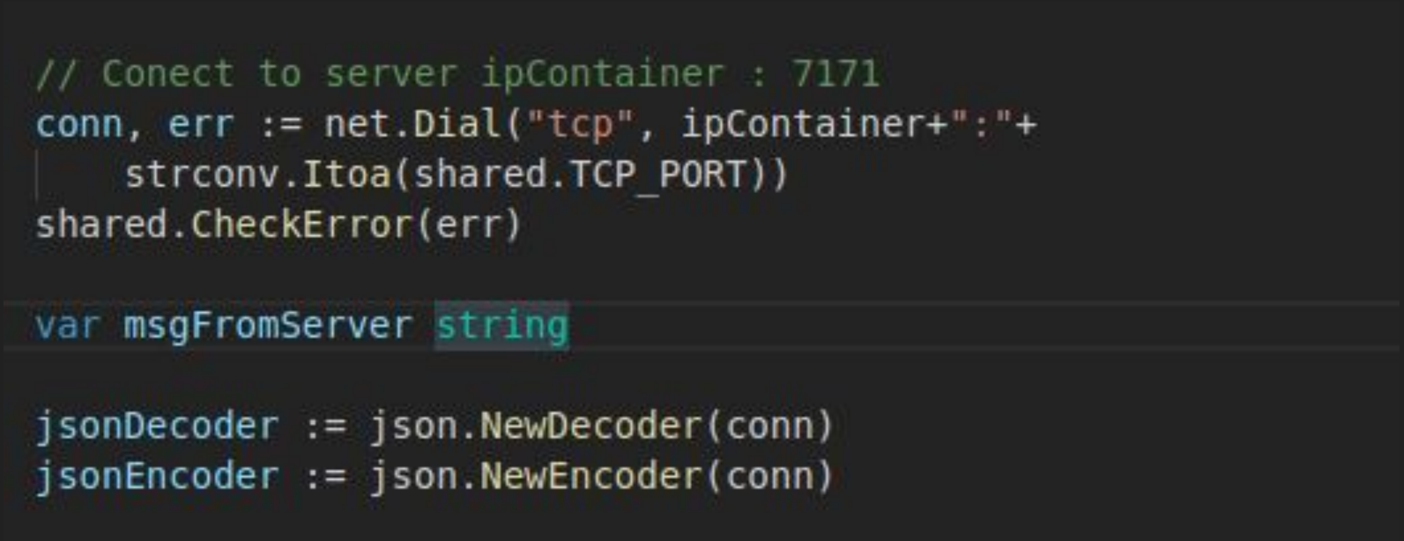
Código - Servidor TCP

// Conexão



```go
func handleConnection(conn net.Conn) {
    var msgFromClient string

    jsonDecoder := json.NewDecoder(conn)
    jsonEncoder := json.NewEncoder(conn)

    // Same loop from Client Side
    for i := 0; i < shared.SAMPLE_SIZE; i++ {

        // Recieve + Deserializes
        err := jsonDecoder.Decode(&msgFromClient)
        shared.CheckError(err)

        t1 := time.Now()
        // Prepare the response
        number, _ := strconv.Atoi(msgFromClient)
        msgToClient := application.Fibbonacci(number)
        t2 := time.Now()

        // Serializes + Send
        err = jsonEncoder.Encode(strconv.Itoa(msgToClient))
        shared.CheckError(err)

        x := float64(t2.Sub(t1).Nanoseconds()) / 1000000
        s := fmt.Sprintf("%d,%s,%f", number, conn.RemoteAddr(), x)
        fmt.Println(s)
    }
}
```

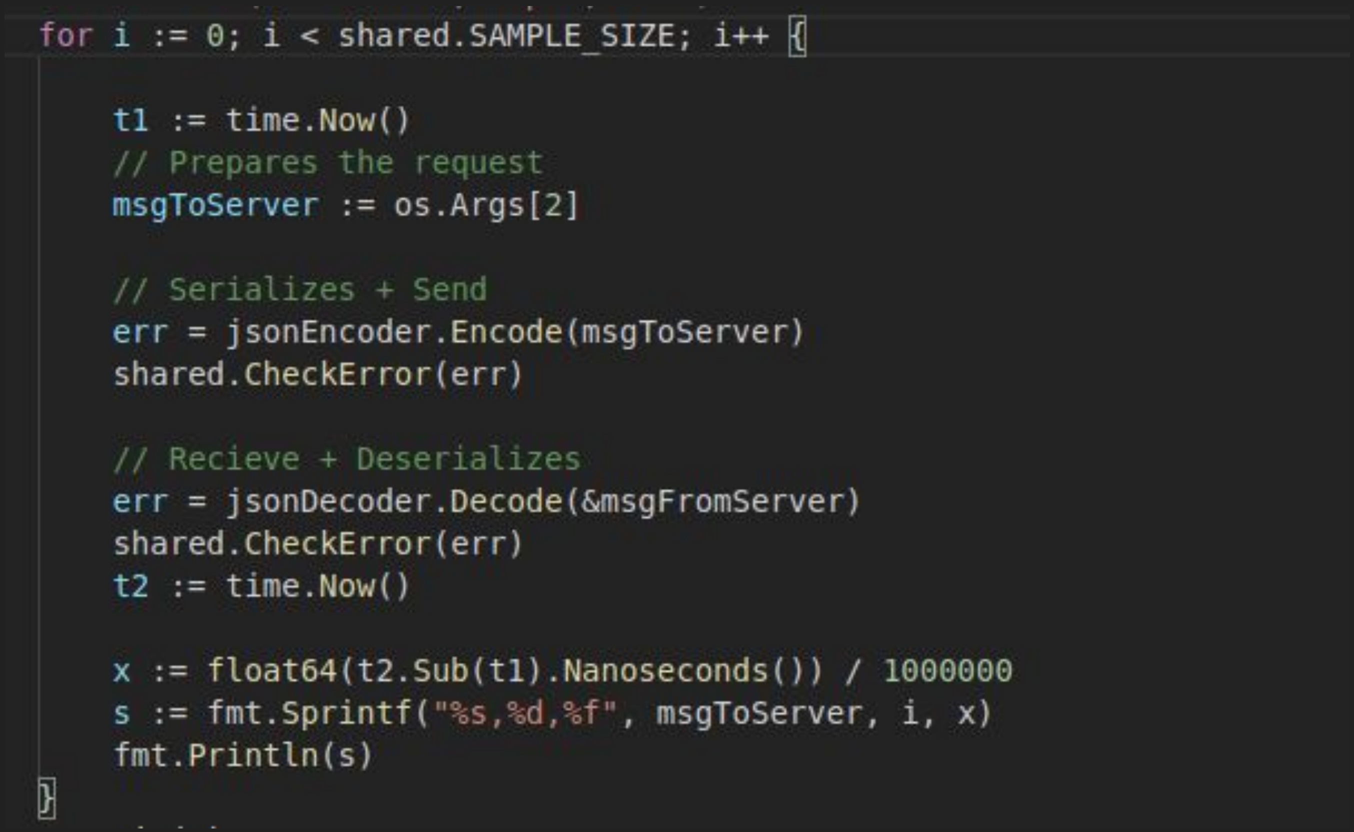
Código - Cliente TCP



```
// Conect to server ipContainer : 7171
conn, err := net.Dial("tcp", ipContainer+":"+
    strconv.Itoa(shared.TCP_PORT))
shared.CheckError(err)

var msgFromServer string

jsonDecoder := json.NewDecoder(conn)
jsonEncoder := json.NewEncoder(conn)
```

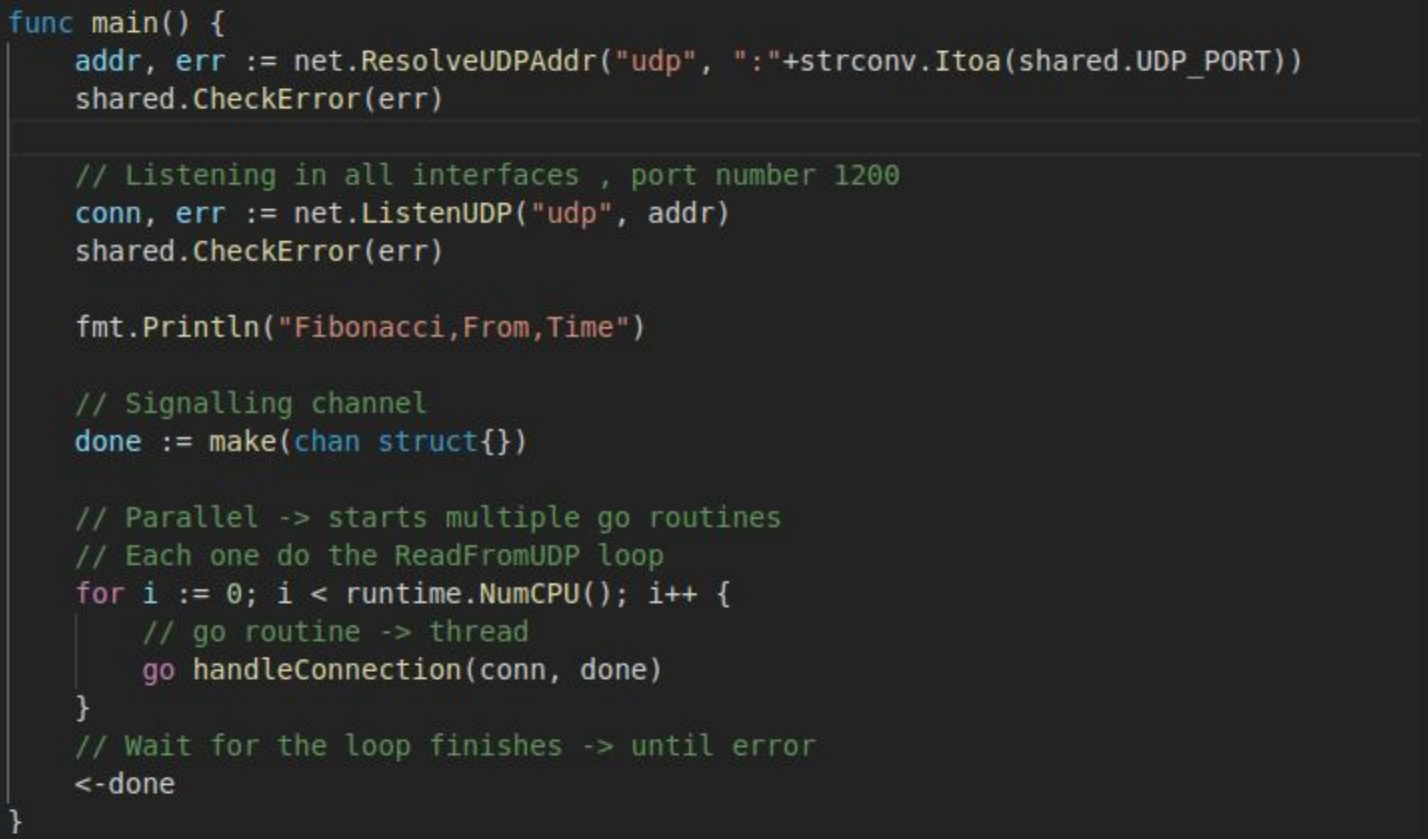
Código - Cliente TCP

// Loop para análise de desempenho



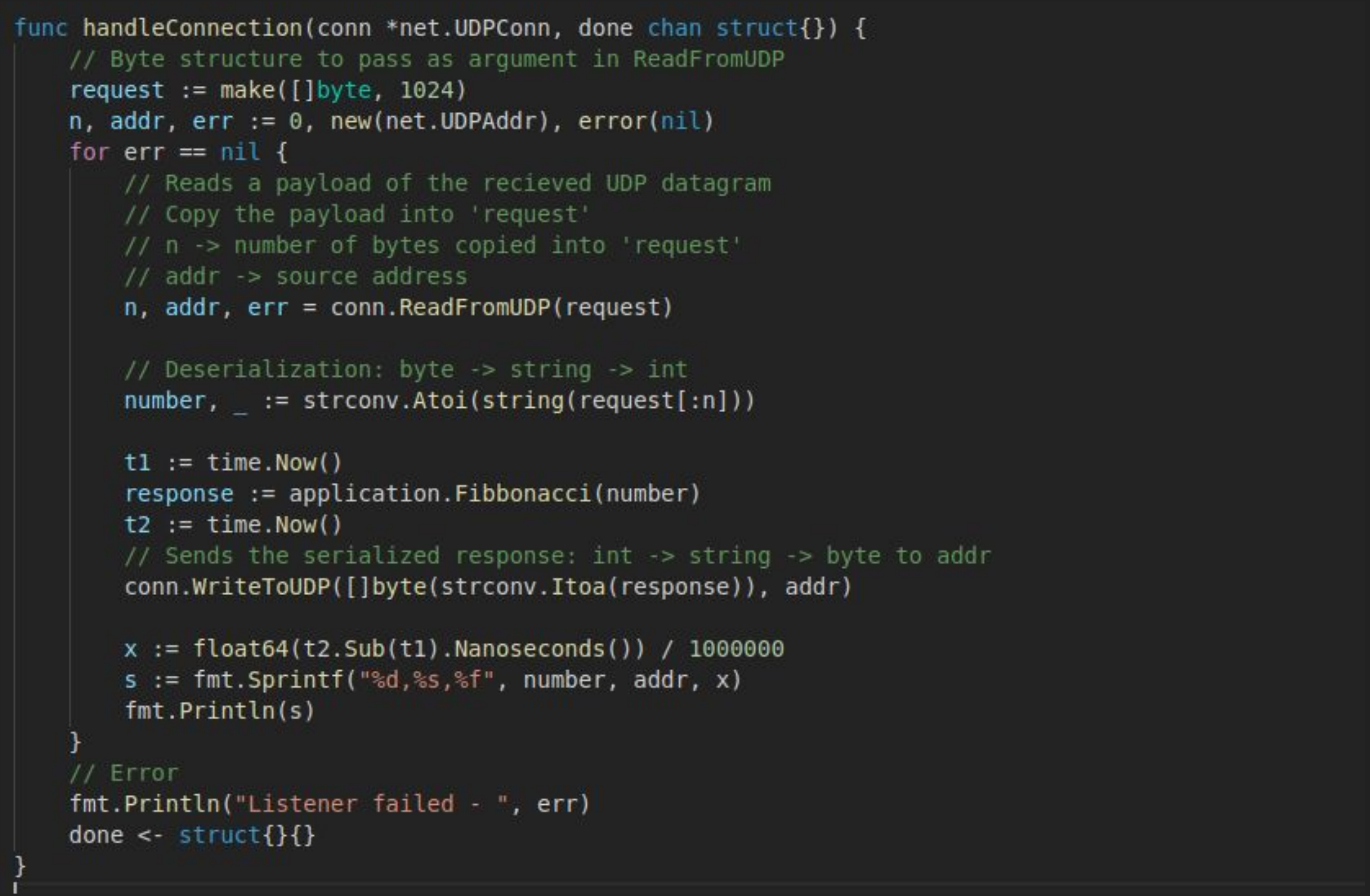
```go
for i := 0; i < shared.SAMPLE_SIZE; i++ {

    t1 := time.Now()
    // Prepares the request
    msgToServer := os.Args[2]

    // Serializes + Send
    err = jsonEncoder.Encode(msgToServer)
    shared.CheckError(err)

    // Recieve + Deserializes
    err = jsonDecoder.Decode(&msgFromServer)
    shared.CheckError(err)
    t2 := time.Now()

    x := float64(t2.Sub(t1).Nanoseconds()) / 1000000
    s := fmt.Sprintf("%s,%d,%f", msgToServer, i, x)
    fmt.Println(s)
}
```

Código - Servidor UDP



```go
func main() {
    addr, err := net.ResolveUDPAddr("udp", ":"+strconv.Itoa(shared.UDP_PORT))
    shared.CheckError(err)

    // Listening in all interfaces , port number 1200
    conn, err := net.ListenUDP("udp", addr)
    shared.CheckError(err)

    fmt.Println("Fibonacci,From,Time")

    // Signalling channel
    done := make(chan struct{})

    // Parallel -> starts multiple go routines
    // Each one do the ReadFromUDP loop
    for i := 0; i < runtime.NumCPU(); i++ {
        // go routine -> thread
        go handleConnection(conn, done)
    }
    // Wait for the loop finishes -> until error
    <-done
}
```

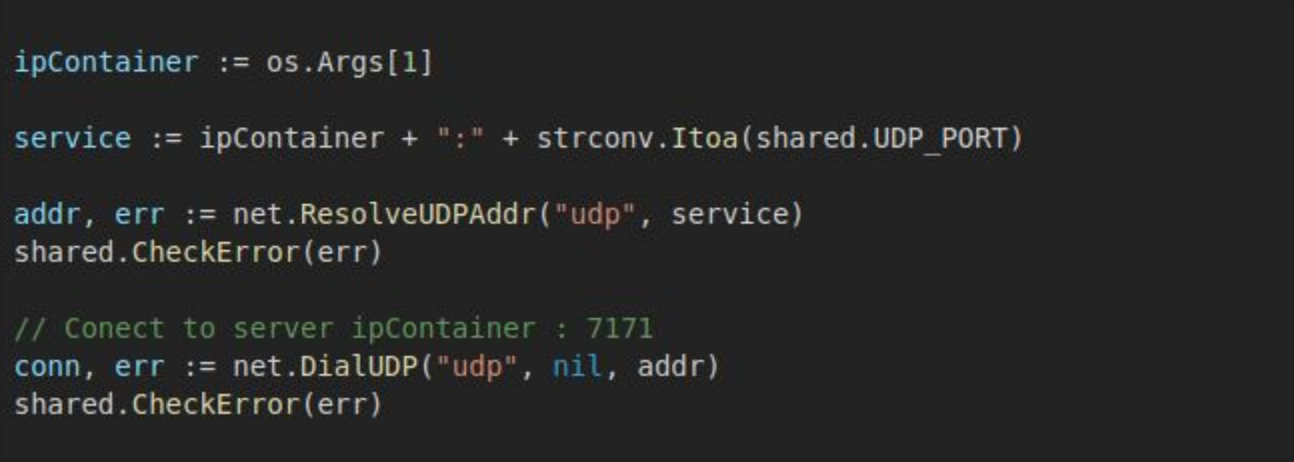
Código - Servidor UDP



```go
func handleConnection(conn *net.UDPConn, done chan struct{}) {
    // Byte structure to pass as argument in ReadFromUDP
    request := make([]byte, 1024)
    n, addr, err := 0, new(net.UDPAddr), error(nil)
    for err == nil {
        // Reads a payload of the recieved UDP datagram
        // Copy the payload into 'request'
        // n -> number of bytes copied into 'request'
        // addr -> source address
        n, addr, err = conn.ReadFromUDP(request)

        // Deserialization: byte -> string -> int
        number, _ := strconv.Atoi(string(request[:n]))

        t1 := time.Now()
        response := application.Fibbonacci(number)
        t2 := time.Now()
        // Sends the serialized response: int -> string -> byte to addr
        conn.WriteToUDP([]byte(strconv.Itoa(response)), addr)

        x := float64(t2.Sub(t1).Nanoseconds()) / 1000000
        s := fmt.Sprintf("%d,%s,%f", number, addr, x)
        fmt.Println(s)
    }
    // Error
    fmt.Println("Listener failed - ", err)
    done <- struct{}{}
}
```

Código - Cliente UDP

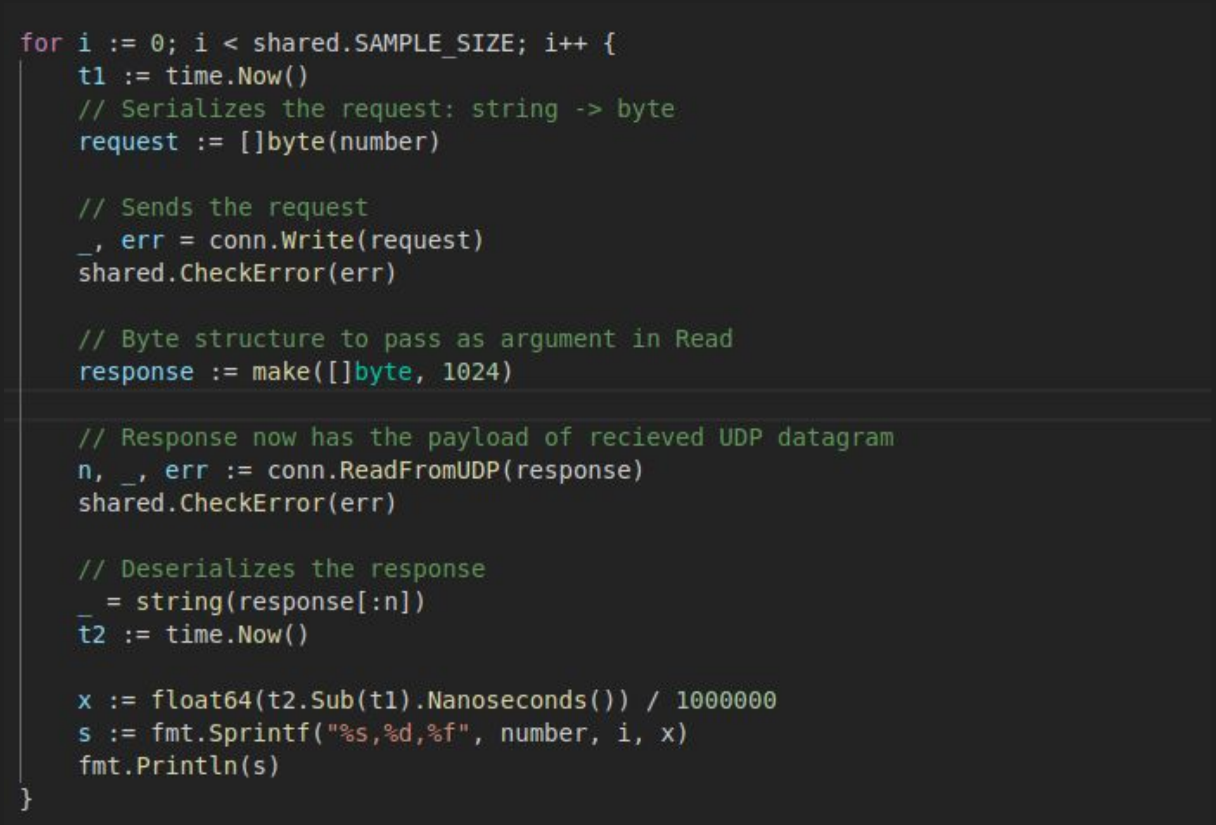
// Preparando cliente



```
ipContainer := os.Args[1]

service := ipContainer + ":" + strconv.Itoa(shared.UDP_PORT)

addr, err := net.ResolveUDPAddr("udp", service)
shared.CheckError(err)

// Conect to server ipContainer : 7171
conn, err := net.DialUDP("udp", nil, addr)
shared.CheckError(err)
```

Código - Cliente UDP

// Realizando operações de escrita e leitura



```go
for i := 0; i < shared.SAMPLE_SIZE; i++ {
    t1 := time.Now()
    // Serializes the request: string -> byte
    request := []byte(number)

    // Sends the request
    _, err = conn.Write(request)
    shared.CheckError(err)

    // Byte structure to pass as argument in Read
    response := make([]byte, 1024)

    // Response now has the payload of recieved UDP datagram
    n, _, err := conn.ReadFromUDP(response)
    shared.CheckError(err)

    // Deserializes the response
    _ = string(response[:n])
    t2 := time.Now()

    x := float64(t2.Sub(t1).Nanoseconds()) / 1000000
    s := fmt.Sprintf("%s,%d,%f", number, i, x)
    fmt.Println(s)
}
```

Avaliação de Desempenho

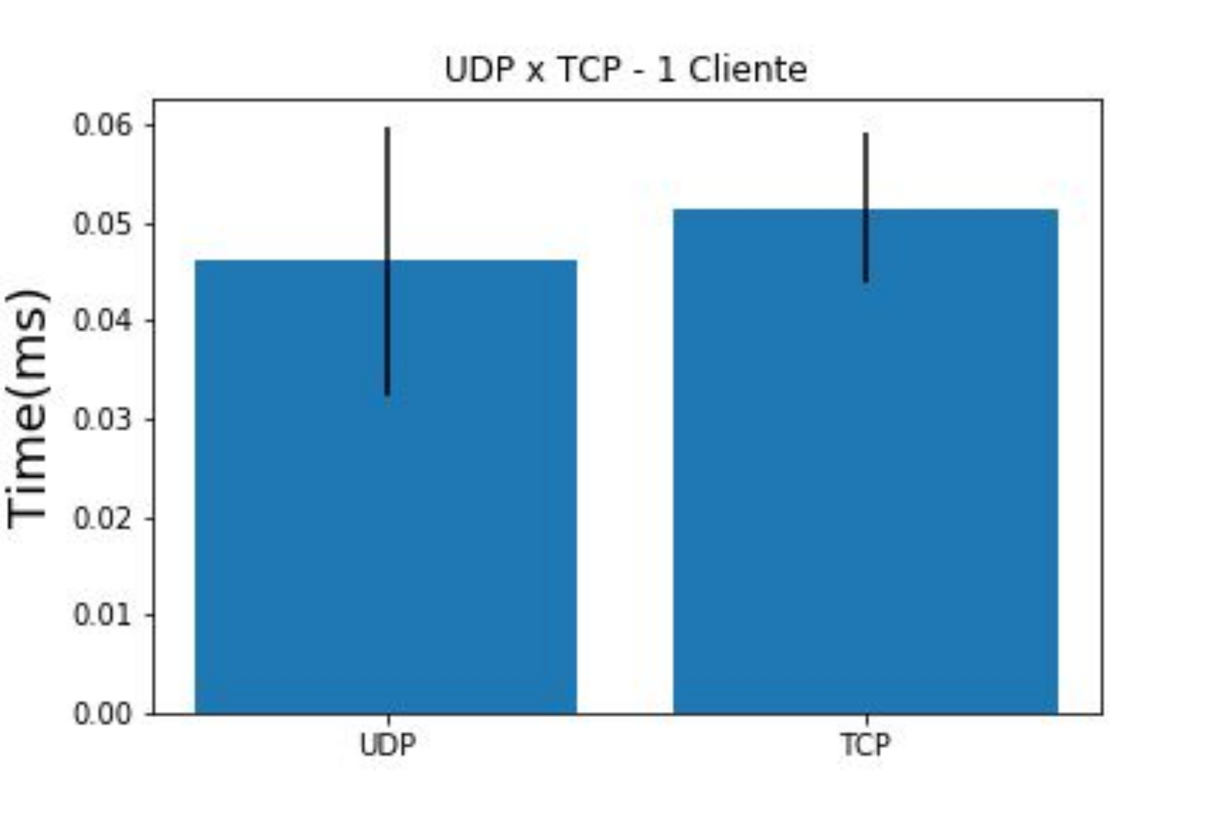
Máquina:

- Memória: 7,7 GiB
- Desktop
- OS: Ubuntu 18.04.2 LTS
- Processador: Intel® Core™ i7-3770 CPU @ 3.40GHz × 8

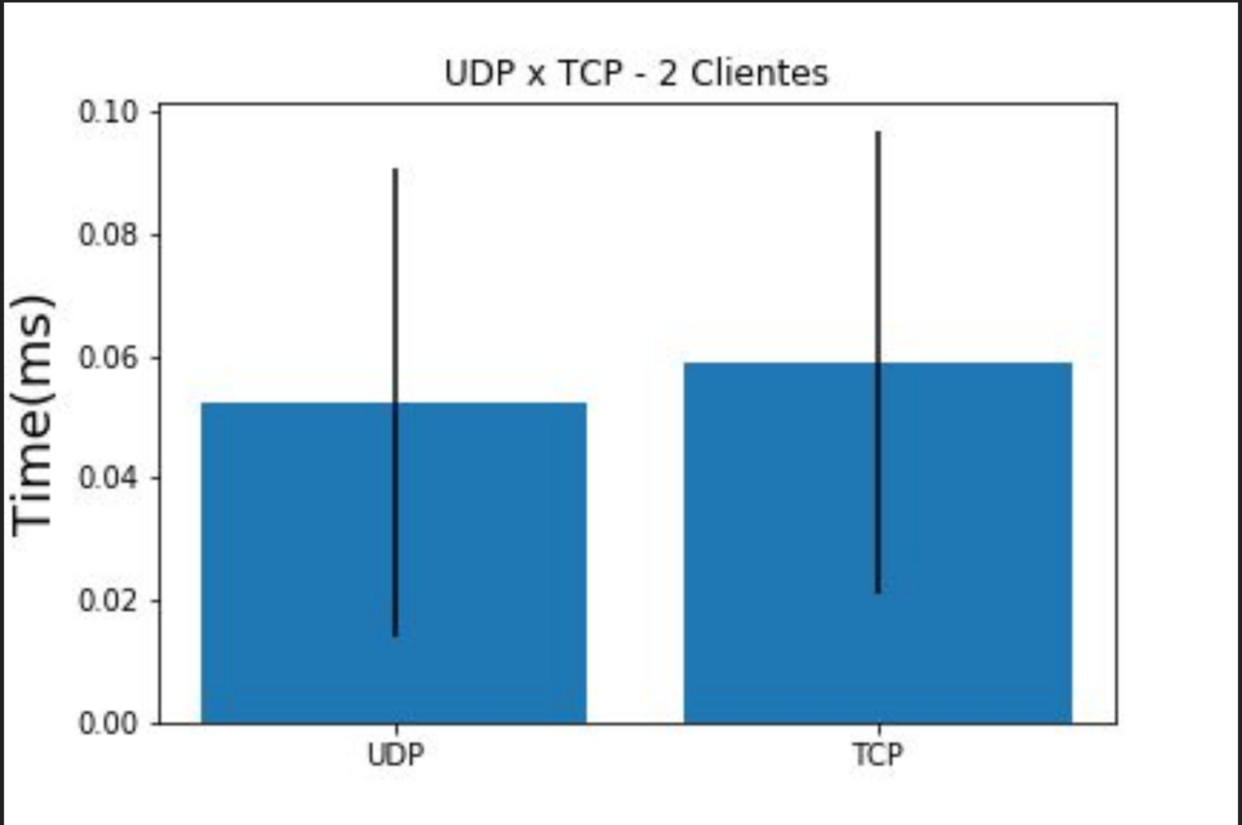
Preparação de Ambiente

- Máquina recém inicializada
- Docker (Imagem: golang + repositório)

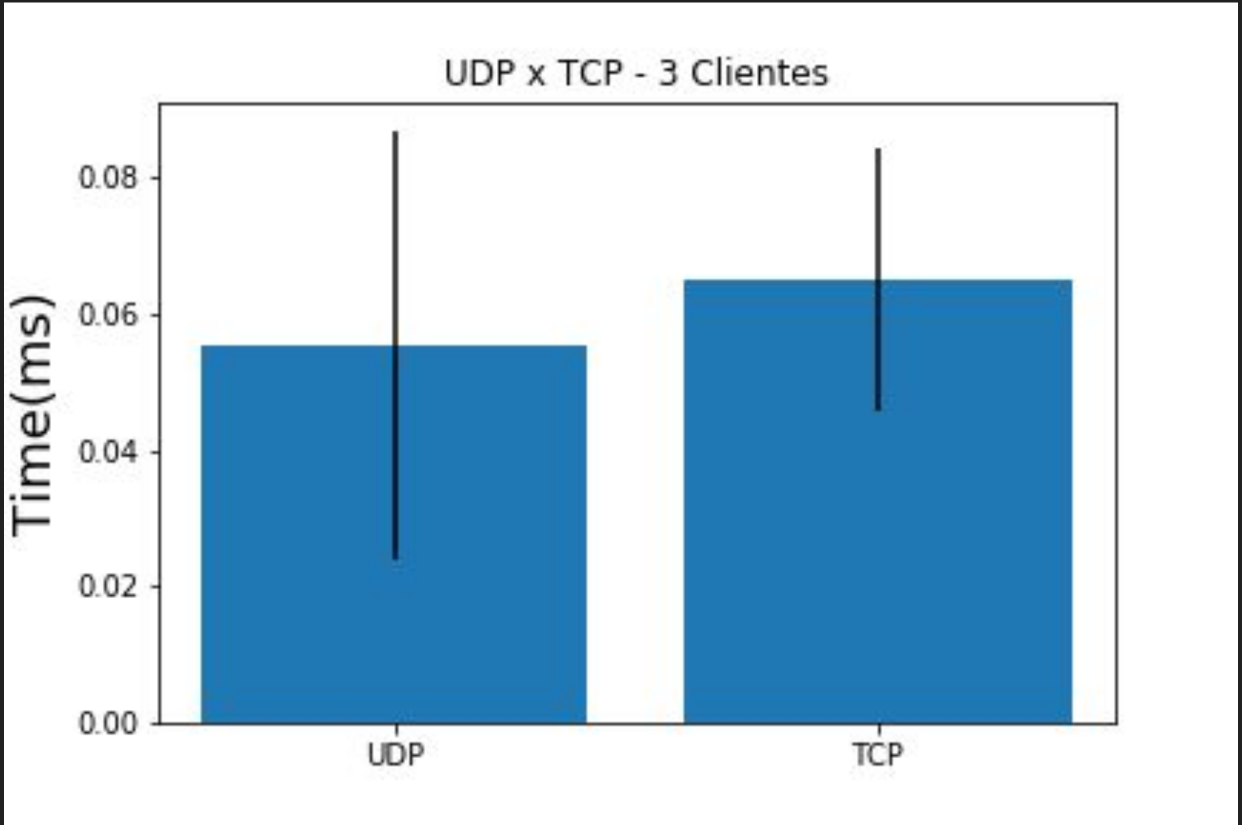
Único Cliente



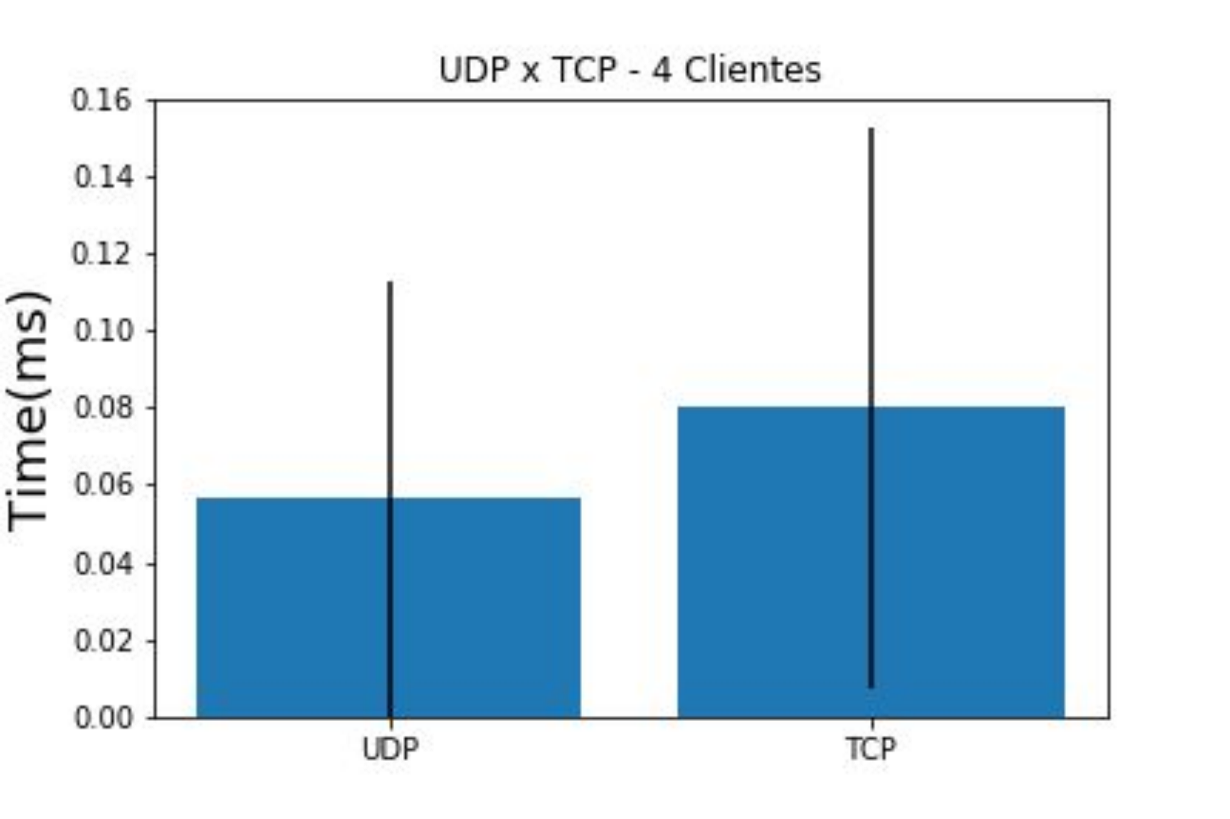
2 Clientes Simultâneos



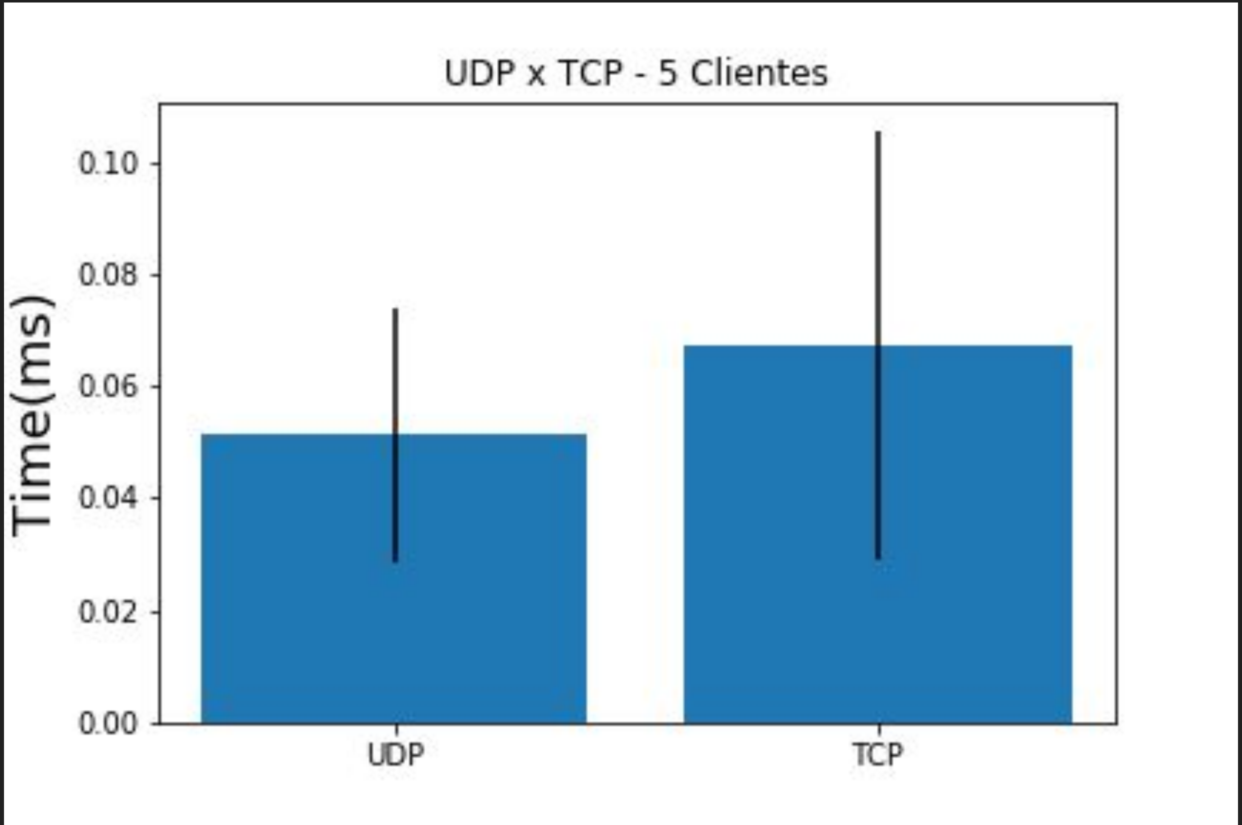
3 Clientes Simultâneos



4 Clientes Simultâneos



5 Clientes Simultâneos


UDP x TCP - 5 Clientes
Time(ms)
UDP
TCP