Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

Лабораторная работа № 3

по курсу «Компьютерные системы и сети»

«Протокол одноранговой сети»

Студент группы ИУ9-32Б Федуков А. А.

Преподаватель Посевин Д. П.

24 сентября 2024 г.

Цель работы

Целью данной работы является разработка одноранговой сетевой службы.

Задание

Краткое описание вариантов одноранговых сетевых служб, один из которых нужно раз-работать в ходе выполнения лабораторной работы, приведено в таблице с перечнем вариантов.

Основные требования к сетевой службе: 1. в качестве формата сообщений для протокола взаимодействия пиров нужно использовать JSON; 2. полная проверка данных, получаемых из сети; 3. устойчивость к обрыву соединения; 4. ведение подробного лога всех ошибок, а также других важных событий (установка и завершение соединения с соседним пиром, приём и передача сообщений, и т.п.).

Документация к протоколу должна быть оформлена в виде комментариев к структурам данных, описывающим сообщения, в исходном коде. Сетевая служба должна работать строго на боевых серверах.
Распределённый массив (кольцо) Топология: кольцевой список. Информация, известная пиру при запуске: его IP-адрес и порт, а также IP-адрес и порт следующего пира в кольцевом списке (следующий пир не обязан быть заранее запущен). Описание службы: каждый пир через стандартный поток ввода принимает команды – присвоить целочисленное значение элементу массива, вычислить сумму элементов массива на отрезке. Замечание: за каждым пиром должен быть закреплён фрагмент массива, за хранение которого пир отвечает.Замечание: за каждым пиром должен быть закреплён фрагмент массива, за хранение которого пир отвечает.

Реализация

Для создания пиринговой сети, контролируемой с дашборда, я создал несколько файлов: main.go (задание основных переменных и структур, запуск основных функций); server.go (серверная часть пирингого узла); client.go (клиентская часть пирингого узла); httpServer.go (общение с дашбордом по http);

socketServer.go (общение с дашбордом по websocket); sseServer.go (лог на дашборд); а также start.sh, который собирал код, а потом запускал его. После чего при помощи runNetwork.sh я загружал файлы на сервера и запускал, управляя ими с дашборда webClient.html

Код

Листинг 1: Файл main.go

```go
package main

import (
  "encoding/json"
  "fmt"
  "os"
  "time"
  "strconv"
)

type Request struct {
  Command string `json:"command"`

  Data *json.RawMessage `json:"data"`
}

type Response struct {
  Status string `json:"status"`

  Data *json.RawMessage `json:"data"`
}

type jsonInt struct {
  N          int `json:"n"`
  StartIndex int `json:"startindex"`
  EndIndex   int `json:"endindex"`
}

type jsonSum struct {
  N          int `json:"n"`
  M          int `json:"m"`
  StartIndex int `json:"startindex"`
}

var MyPort = ":1572"
var MySocketPort = ":1494"
var MyHttpPort = ":1491"
var MySsePort = ":1497"
var MyIP = os.Getenv("MyIP")
var NetworkList = [...]string{"185.104.251.226", "185.102.139.161", "
    185.102.139.168", "185.102.139.169"}
var MyIndex int
var NeigborIndex int
var ConnectionMode = "Socket"
var MyValue = 1
var RequestedValue int
var SocketCommand string

type WebFormRequestParsed struct {
  command string
  data    interface{}
}

var WebFormRequest WebFormRequestParsed

func sendToWebForm(message string, mode string) {
  switch mode {
  case "Socket":
    dataSocketChannel <- message
  case "SSE":
    dataSseChannel <- message
  }
}

// Run commands from "c" and write response into "out"
func parseFromForm(out *string, c ...string)  {
  var data interface{}

  switch c[0] {
  case "setValue":
    i1, _ := strconv.Atoi(c[1])
    i2, _ := strconv.Atoi(c[2])
    data = &jsonInt{EndIndex: i1, N: i2, StartIndex: MyIndex}
    WebFormRequest.command = c[0]
    WebFormRequest.data = data
    interactWithWebForm(Connection, out)
  case "getSum":
    i1, _ := strconv.Atoi(c[1])
    i2, _ := strconv.Atoi(c[2])
    data = &jsonSum{N: i1, M: i2, StartIndex: MyIndex}
    WebFormRequest.command = c[0]
    WebFormRequest.data = data
		interactWithWebForm(Connection, out)
	case "showValue":
		fmt.Println(MyValue)
		*out = *out + strconv.Itoa(MyValue)
	default:
		fmt.Println("Unknown command from WebForm")
		SocketCommand = ""
		*out = *out + "Bad command!"

	}

}
func main() {
	// Set IP
	for i, v := range NetworkList {
		if MyIP == v {
			MyIndex = i
			NeigborIndex = (i + 1) % len(NetworkList)
			break
		}
	}

	fmt.Println("MyIP:", NetworkList[MyIndex])
	fmt.Println("NeigborIP:", NetworkList[NeigborIndex])
	fmt.Println("Port:", MyPort)

	go startSocketServer()
	time.Sleep(1 * time.Second)
	go startHttpServer()
	time.Sleep(1 * time.Second)
	go startSseServer()
	time.Sleep(1 * time.Second)
	go startServer(MyIP + MyPort)
	time.Sleep(5 * time.Second)
	fmt.Println("Don't forget to reload a page!")
	go startClient(NetworkList[NeigborIndex] + MyPort)
	select {}

}
```

Листинг 2: Файл server.go

```go
package main

import (
	"encoding/json"
	"fmt"
	"math/big"
	"net"

	log "github.com/mgutz/logxi/v1"
)

// Client - состояние клиента.
type Client struct {
	logger log.Logger     // Объект для печати логов
	conn   *net.TCPConn   // Объект TCP-соединения
	enc    *json.Encoder  // Объект для кодирования и отправки сообщений
	sum    *big.Rat       // Текущая сумма полученных от клиента дробей
	count  int64          // Количество полученных от клиента дробей
}

// NewClient - конструктор клиента, принимает в качестве параметра
// объект TCP-соединения.
func NewClient(conn *net.TCPConn) *Client {
	return &Client{
		logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())
		),
		conn:   conn,
		enc:    json.NewEncoder(conn),
		sum:    big.NewRat(0, 1),
		count:  0,
	}
}

// serve - метод, в котором реализован цикл взаимодействия с клиентом.
// Подразумевается, что метод serve будет вызаваться в отдельной go-прог
	рамме.
func (client *Client) serve() {
	defer client.conn.Close()
	decoder := json.NewDecoder(client.conn)
	for {
		var req Request
		if err := decoder.Decode(&req); err != nil {
			client.logger.Error("cannot decode message", "reason", err)
			break
		} else {
			client.logger.Info("received command", "command", req.Command)
			if client.handleRequest(&req) {
				client.logger.Info("shutting down connection")
				break
			}
		}
  }
}

func goPeer(addrStr string, command string, data interface{}, args ...int) {
  if addrStr == "next" {
    addrStr = NetworkList[NeigborIndex] + MyPort
  } else if addrStr == "back" {
    addrStr = NetworkList[args[0]] + MyPort
  }

  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    log.Error("cannot resolve addres to connect", "adress", addrStr, "reason", err)
  } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
    log.Error("cannot establish connection to")
  } else {
    log.Info("establish connection to", "address", conn.RemoteAddr().String())
    encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
    send_request(encoder, command, data)

    // Получение ответа.
    var resp Response
    if err := decoder.Decode(&resp); err != nil {
      fmt.Printf("error: %v\n", err)
    }
    // Вывод ответа в стандартный поток вывода.
    switch resp.Status {
    case "ok":
      fmt.Printf("Data sended\n")
    case "failed":
      if resp.Data == nil {
        fmt.Printf("error: while next peer data field is absent in response\n")
      } else {
        var errorMsg string
        if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          fmt.Printf("failed: %s\n", errorMsg)
        }
      }
    default:
      fmt.Printf("error: server reports unknown status %q\n", resp.Status)
    }
  }
}

// handleRequest - метод обработки запроса от клиента. Он возвращает true,
// если клиент передал команду "quit" и хочет завершить общение.
func (client *Client) handleRequest(req *Request) bool {
  sendToWebForm("Server command: " + req.Command, "SSE")
  switch req.Command {
  case "quit":
    client.respond("ok", nil)
    return true
  case "getValue":
    // by id
    errorMsg := ""
    if req.Data == nil {
      errorMsg = "data field is absent"
    } else {
      var s jsonSum
      if err := json.Unmarshal(*req.Data, &s); err != nil {
        errorMsg = "malformed data field"
      } else {
        if MyIndex == s.N {
          fmt.Printf("\nReturned MyValue to index %d\n", s.StartIndex)
          goPeer("back", "giveRequestedValue", &jsonInt{N: MyValue, StartIndex: MyIndex, EndIndex: s.StartIndex}, s.StartIndex)
        } else {
          fmt.Println("Going to next peer")
          goPeer("next", "getValue", req.Data)
        }

        sendToWebForm(fmt.Sprintf("Geting value on to %d index", s.StartIndex), "SSE")
      }
    }

    if errorMsg == "" {
      client.respond("ok", nil)
    } else {
      client.logger.Error("addition failed", "reason", errorMsg)
      client.respond("failed", errorMsg)
    }

  case "setValue":
    // by id
    errorMsg := ""
    if req.Data == nil {
      errorMsg = "data field is absent"
    } else {
      var s jsonInt
      if err := json.Unmarshal(*req.Data, &s); err != nil {
        errorMsg = "malformed data field"
      } else {
        if s.EndIndex == MyIndex {
          MyValue = s.N
          fmt.Printf("\nMyValue is updated to %d by index %d\n", MyValue
    , s.StartIndex)
        } else {
          goPeer("next", "setValue", req.Data)
        }
        sendToWebForm(fmt.Sprintf("Updating value on %d index to %d", s.
    EndIndex, s.N), "SSE")
      }
    }

    if errorMsg == "" {
      client.respond("ok", nil)
    } else {
      client.logger.Error("addition failed", "reason", errorMsg)
      client.respond("failed", errorMsg)
    }

  case "giveRequestedValue":
    // by id
    errorMsg := ""
    if req.Data == nil {
      errorMsg = "data field is absent"
    } else {
      var s jsonInt
      if err := json.Unmarshal(*req.Data, &s); err != nil {
        errorMsg = "malformed data field"
      } else {
        RequestedValue = s.N
        fmt.Printf("\nGot value %d from index %d\n", RequestedValue, s.
    StartIndex)
        sendToWebForm(fmt.Sprintf("Give value %d", s.N), "SSE")
      }
    }

    if errorMsg == "" {
      client.respond("ok", nil)
```

```go
    } else {
      client.logger.Error("addition failed", "reason", errorMsg)
      client.respond("failed", errorMsg)
    }

  default:
    client.logger.Error("unknown command")
    client.respond("failed", "unknown command")
  }
  return false
}

// respond - вспомогательный метод для передачи ответа с указанным статусом
// и данными. Данные могут быть пустыми (data == nil).
func (client *Client) respond(status string, data interface{}) {
  var raw json.RawMessage
  raw, _ = json.Marshal(data)
  client.enc.Encode(&Response{status, &raw})
}

func startServer(addrStr string) {
  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    log.Error("address resolution failed", "address", addrStr)
  } else {
    log.Info("resolved TCP address", "address", addr.String())

    // Инициация слушания сети на заданном адресе.
    if listener, err := net.ListenTCP("tcp", addr); err != nil {
      log.Error("listening failed", "reason", err)
    } else {
      // Цикл приёма входящих соединений и обработки запросов.
      for {
        if conn, err := listener.AcceptTCP(); err != nil {
          log.Error("cannot accept connection", "reason", err)
        } else {
          log.Info("accepted connection", "address", conn.RemoteAddr().String())
          // Запуск go-программы для обслуживания клиентов.
          go NewClient(conn).serve()

        }
      }
    }
  }
}
```

Листинг 3: Файл client.go

```go
package main

import (
  "encoding/json"
  "fmt"
  "net"
  "strconv"
  "time"

  log "github.com/mgutz/logxi/v1"
  "github.com/skorobogatov/input"
)

var Connection *net.TCPConn

func interactWithWebForm(conn *net.TCPConn, webFormResponse *string) {
  encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)

  switch WebFormRequest.command {
  case "quit":
    send_request(encoder, "quit", nil)
    return
  case "help":
    *webFormResponse = *webFormResponse + "You should use client after connection to server!"
    *webFormResponse = *webFormResponse + "showValue - prints current index value"
    *webFormResponse = *webFormResponse + "setValue - define a value of given index"
    *webFormResponse = *webFormResponse + "getSum - count sum from n to m indexes"
  case "showValue":
    fmt.Println(MyValue)
    *webFormResponse = *webFormResponse + strconv.Itoa(MyValue)
  case "setValue":
    // by id
    fmt.Println("Updating value by webForm")
    send_request(encoder, "setValue", WebFormRequest.data)
    *webFormResponse = *webFormResponse + "Sended setValue request"

  case "getSum":
    // from n to m
    csum := 0
    cstart := WebFormRequest.data.(*jsonSum).N
    if WebFormRequest.data.(*jsonSum).N == MyIndex {
      csum += MyValue
      cstart = WebFormRequest.data.(*jsonSum).N + 1
    }
    fmt.Println("Requests for sum...")

    for i := cstart; i <= WebFormRequest.data.(*jsonSum).M; i++ {
      // Запрос значения
      send_request(encoder, "getValue", &jsonSum{N: i, StartIndex:
    MyIndex})
      // Получение ответа.
      var resp Response
      if err := decoder.Decode(&resp); err != nil {
        fmt.Printf("error: %v\n", err)
        break
      } else {
        if resp.Status != "ok" {
          fmt.Printf("error: data field is absent in response\n")
        } else {
          fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
          csum += RequestedValue
        }
      }
    }
    fmt.Printf("Sum is %d\n", csum)
    *webFormResponse = *webFormResponse + fmt.Sprintf("Sum is %d\n",
    csum)

  default:
    fmt.Printf("error: unknown command\n")
    *webFormResponse = *webFormResponse + "error: unknown command"
  }
  WebFormRequest = WebFormRequestParsed{}
}

// interact - функция, содержащая цикл взаимодействия с сервером.
func interact(conn *net.TCPConn) {
  defer conn.Close()
  encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
  for {
    // Чтение команды из стандартного потока ввода
    fmt.Printf("\ncommand = ")
    command := input.Gets()

    fmt.Println("Send command to WebForm: ", command)
    go sendToWebForm("Client command: " + command, "SSE")

    // Отправка запроса.
    switch command {
    case "quit":
      send_request(encoder, "quit", nil)
      return
    case "help":
      fmt.Println("You should use client after connection to server!")
      fmt.Println("showValue - prints current index value")
      fmt.Println("setValue - define a value of given index")
      fmt.Println("getSum - count sum from n to m indexes")
      continue
    case "showValue":
      fmt.Println(MyValue)
      continue
    case "setValue":
      // by id
      fmt.Printf("Index = ")
      if valInd, err := strconv.Atoi(input.Gets()); err != nil {
        fmt.Println("\nMust be number!")
        continue
      } else {
        fmt.Printf("Value = ")
        if valN, err := strconv.Atoi(input.Gets()); err != nil {
          fmt.Println("\nMust be number!")
          continue
        } else {
          send_request(encoder, "setValue", &jsonInt{N: valN, StartIndex
   : MyIndex, EndIndex: valInd})
        }
      }

    case "getSum":
      // from n to m
      fmt.Printf("From index = ")
      if valN, err := strconv.Atoi(input.Gets()); err != nil {
        fmt.Println("\nMust be number!")
        continue
      } else {
        fmt.Printf("To index = ")
        if valM, err := strconv.Atoi(input.Gets()); err != nil {
          fmt.Println("\nMust be number!")
          continue
        } else {
          csum := 0
          cstart := valN
          if valN == MyIndex {
```

```
          csum += MyValue
          cstart = valN + 1
        }
        fmt.Println("Requests for sum...")
        for i := cstart; i <= valM; i++ {
          // Запрос значения
          send_request(encoder, "getValue", &jsonSum{N: i, StartIndex: MyIndex})
          // Получение ответа.
          var resp Response
          if err := decoder.Decode(&resp); err != nil {
            fmt.Printf("error: %v\n", err)
            break
          } else {
            if resp.Status != "ok" {
              fmt.Printf("error: data field is absent in response\n")
            } else {
              fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
              csum += RequestedValue
            }
          }
        }
        fmt.Printf("Sum is %d\n", csum)
        continue
      }
    }

  default:
    fmt.Printf("error: unknown command\n")
    // Quit for Socket interaction
    continue
  }

  // Получение ответа.
  var resp Response
  if err := decoder.Decode(&resp); err != nil {
    fmt.Printf("error: %v\n", err)
    break
  }

  // Вывод ответа в стандартный поток вывода.
  switch resp.Status {
  case "ok":
    fmt.Printf("ok\n")
  case "failed":
      if resp.Data == nil {
        fmt.Printf("error: data field is absent in response\n")
      } else {
        var errorMsg string
        if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          fmt.Printf("failed: %s\n", errorMsg)
        }
      }
    default:
      fmt.Printf("error: server reports unknown status %q\n", resp.
    Status)
    }

  }
}

// send_request - вспомогательная функция для передачи запроса с указанн
    ой командой
// и данными. Данные могут быть пустыми (data == nil).
func send_request(encoder *json.Encoder, command string, data interface
    {}) {
  var raw json.RawMessage
  raw, _ = json.Marshal(data)
  encoder.Encode(&Request{command, &raw})
}package main

import (
  "encoding/json"
  "fmt"
  "net"
  "strconv"
  "time"

  log "github.com/mgutz/logxi/v1"
  "github.com/skorobogatov/input"
)

var Connection *net.TCPConn

func interactWithWebForm(conn *net.TCPConn, webFormResponse *string) {
  encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)

  switch WebFormRequest.command {
  case "quit":
    send_request(encoder, "quit", nil)
    return
  case "help":
    *webFormResponse = *webFormResponse + "You should use client after connection to server!"
    *webFormResponse = *webFormResponse + "showValue - prints current index value"
    *webFormResponse = *webFormResponse + "setValue - define a value of given index"
    *webFormResponse = *webFormResponse + "getSum - count sum from n to m indexes"
  case "showValue":
    fmt.Println(MyValue)
    *webFormResponse = *webFormResponse + strconv.Itoa(MyValue)
  case "setValue":
    // by id
    fmt.Println("Updating value by webForm")
    send_request(encoder, "setValue", WebFormRequest.data)
    *webFormResponse = *webFormResponse + "Sended setValue request"

  case "getSum":
    // from n to m
    csum := 0
    cstart := WebFormRequest.data.(*jsonSum).N
    if WebFormRequest.data.(*jsonSum).N == MyIndex {
      csum += MyValue
      cstart = WebFormRequest.data.(*jsonSum).N + 1
    }
    fmt.Println("Requests for sum...")

    for i := cstart; i <= WebFormRequest.data.(*jsonSum).M; i++ {
      // Запрос значения
      send_request(encoder, "getValue", &jsonSum{N: i, StartIndex: MyIndex})
      // Получение ответа.
      var resp Response
      if err := decoder.Decode(&resp); err != nil {
        fmt.Printf("error: %v\n", err)
        break
      } else {
        if resp.Status != "ok" {
          fmt.Printf("error: data field is absent in response\n")
        } else {
          fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
          csum += RequestedValue
        }
```

```
            }
        }
        fmt.Printf("Sum is %d\n", csum)
        *webFormResponse = *webFormResponse + fmt.Sprintf("Sum is %d\n",
      csum)

    default:
        fmt.Printf("error: unknown command\n")
        *webFormResponse = *webFormResponse + "error: unknown command"
    }
    WebFormRequest = WebFormRequestParsed{}
}

// interact - функция, содержащая цикл взаимодействия с сервером.
func interact(conn *net.TCPConn) {
    defer conn.Close()
    encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
    for {
        // Чтение команды из стандартного потока ввода
        fmt.Printf("\ncommand = ")
        command := input.Gets()

        fmt.Println("Send command to WebForm: ", command)
        go sendToWebForm("Client command: " + command, "SSE")

        // Отправка запроса.
        switch command {
        case "quit":
            send_request(encoder, "quit", nil)
            return
        case "help":
            fmt.Println("You should use client after connection to server!")
            fmt.Println("showValue - prints current index value")
            fmt.Println("setValue - define a value of given index")
            fmt.Println("getSum - count sum from n to m indexes")
            continue
        case "showValue":
            fmt.Println(MyValue)
            continue
        case "setValue":
            // by id
            fmt.Printf("Index = ")
            if valInd, err := strconv.Atoi(input.Gets()); err != nil {
                fmt.Println("\nMust be number!")
                continue
            } else {
```

```go
      fmt.Printf("Value = ")
      if valN, err := strconv.Atoi(input.Gets()); err != nil {
        fmt.Println("\nMust be number!")
        continue
      } else {
        send_request(encoder, "setValue", &jsonInt{N: valN, StartIndex: MyIndex, EndIndex: valInd})
      }
    }

  case "getSum":
    // from n to m
    fmt.Printf("From index = ")
    if valN, err := strconv.Atoi(input.Gets()); err != nil {
      fmt.Println("\nMust be number!")
      continue
    } else {
      fmt.Printf("To index = ")
      if valM, err := strconv.Atoi(input.Gets()); err != nil {
        fmt.Println("\nMust be number!")
        continue
      } else {
        csum := 0
        cstart := valN
        if valN == MyIndex {
          csum += MyValue
          cstart = valN + 1
        }
        fmt.Println("Requests for sum...")
        for i := cstart; i <= valM; i++ {
          // Запрос значения
          send_request(encoder, "getValue", &jsonSum{N: i, StartIndex: MyIndex})
          // Получение ответа.
          var resp Response
          if err := decoder.Decode(&resp); err != nil {
            fmt.Printf("error: %v\n", err)
            break
          } else {
            if resp.Status != "ok" {
              fmt.Printf("error: data field is absent in response\n")
            } else {
              fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
              csum += RequestedValue
            }
            }
          }
          fmt.Printf("Sum is %d\n", csum)
          continue
        }
      }

    default:
      fmt.Printf("error: unknown command\n")
      // Quit for Socket interaction
      continue
    }

    // Получение ответа.
    var resp Response
    if err := decoder.Decode(&resp); err != nil {
      fmt.Printf("error: %v\n", err)
      break
    }

    // Вывод ответа в стандартный поток вывода.
    switch resp.Status {
    case "ok":
      fmt.Printf("ok\n")
    case "failed":
      if resp.Data == nil {
        fmt.Printf("error: data field is absent in response\n")
      } else {
        var errorMsg string
        if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
          fmt.Printf("error: malformed data field in response\n")
        } else {
          fmt.Printf("failed: %s\n", errorMsg)
        }
      }
    default:
      fmt.Printf("error: server reports unknown status %q\n", resp.Status)
    }

  }
}

// send_request - вспомогательная функция для передачи запроса с указанной командой
// и данными. Данные могут быть пустыми (data == nil).
```

```go
func send_request(encoder *json.Encoder, command string, data interface{}) {
  var raw json.RawMessage
  raw, _ = json.Marshal(data)
  encoder.Encode(&Request{command, &raw})
}

func startClient(addrStr string) {
  // Установка соединения с сервером и
  // запуск цикла взаимодействия с сервером.
  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    log.Error("cannot resolve addres to connect", "adress", addrStr, "reason", err)
  } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
    log.Error("cannot establish connection to")
    log.Info("waiting 5 seconds and trying again")
    time.Sleep(5 * time.Second)
    go startClient(addrStr)
  } else {
    log.Info("establish connection to", "address", conn.RemoteAddr().String())
    Connection = conn
    go interact(conn)
  }
}


func startClient(addrStr string) {
  // Установка соединения с сервером и
  // запуск цикла взаимодействия с сервером.
  if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
    log.Error("cannot resolve addres to connect", "adress", addrStr, "reason", err)
  } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
    log.Error("cannot establish connection to")
    log.Info("waiting 5 seconds and trying again")
    time.Sleep(5 * time.Second)
    go startClient(addrStr)
  } else {
    log.Info("establish connection to", "address", conn.RemoteAddr().String())
    Connection = conn
    go interact(conn)
  }
}
```

Листинг 4: Файл httpServer.go

```go
package main

import (
    "encoding/json"
    "fmt"
    "net/http"
    "log"
)

type MyHttpResponse struct {
    Message string `json:"message"`
}

type MyHttpPayload struct {
    Command string `json:"command"`
    I1 string `json:"i1"`
    I2  string `json:"i2"`
}

var dataHttpGetResponse string
var dataHttpPostResponse string

func enableCors(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request)
    {
        w.Header().Set("Access-Control-Allow-Origin", "*") // Allow all
    origins
        w.Header().Set("Access-Control-Allow-Methods", "GET, POST,
    OPTIONS")
        w.Header().Set("Access-Control-Allow-Headers", "Content-Type")

        if r.Method == "OPTIONS" {
            w.WriteHeader(http.StatusOK)
            return
        }

        next.ServeHTTP(w, r)
    })
}

func getHandler(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")

    // Parse query parameters
    comm := r.URL.Query().Get("command")
    i1 := r.URL.Query().Get("i1")
    i2 := r.URL.Query().Get("i2")

  fmt.Println("Get request command:", comm)

    dataHttpGetResponse = "Get: "
    parseFromForm(&dataHttpGetResponse, comm, i1, i2)

    // if i1 == "" || i2 == "" {
    //     http.Error(w, "Missing query parameters", http.
   StatusBadRequest)
    //     return
    // }

    response := MyHttpResponse{Message: dataHttpGetResponse}
    json.NewEncoder(w).Encode(response)

}

func postHandler(w http.ResponseWriter, r *http.Request) {
    var requestData MyHttpPayload

    // Parse JSON body
    err := json.NewDecoder(r.Body).Decode(&requestData)
    if err != nil {
        http.Error(w, "Invalid request body", http.StatusBadRequest)
        return
    }

    fmt.Println("Post request command:", requestData.Command)

    dataHttpPostResponse = "Post: "
    parseFromForm(&dataHttpPostResponse, requestData.Command,
   requestData.I1, requestData.I2)

    response := MyHttpResponse{Message: dataHttpPostResponse}
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(response)
}

func startHttpServer() {
    http.Handle("/get", enableCors(http.HandlerFunc(getHandler)))
    http.Handle("/post", enableCors(http.HandlerFunc(postHandler)))


    fmt.Println("HTTP Server started at", MyHttpPort)
    log.Fatal(http.ListenAndServe(MyIP + MyHttpPort, nil))
}
```

Листинг 5: Файл socketServer.go

```
package main

import (
  "fmt"
  "log"
  "net/http"
  "strings"

  "github.com/gorilla/websocket"
)

var upgrader = websocket.Upgrader{
  CheckOrigin: func(r *http.Request) bool {
    return true
  },
}
var dataSocketChannel = make(chan string)

func handleWebSocket(w http.ResponseWriter, r *http.Request) {
  // Upgrade HTTP to WebSocket
  conn, err := upgrader.Upgrade(w, r, nil)
  if err != nil {
    log.Println("Upgrade error:", err)
    return
  }
  defer conn.Close()

  // Waiting for socket reques
  go func() {
    for {
      // Read a message from the client.
      _, message, err := conn.ReadMessage()
      if err != nil {
        fmt.Println(err)
        return
      }
      // Print the message to the console.
      fmt.Println("Received command:", string(message))
      SocketCommand = string(message)
      c := strings.Split(SocketCommand, " ")

      out := "Socket: "
```

```
            parseFromForm(&out, c...)
            conn.WriteMessage(websocket.TextMessage, []byte(out))
        }
    }()


    // Waiting for dataSocketChannel to send to WebForm
    for message := range dataSocketChannel {
        // Send message to client
        err := conn.WriteMessage(websocket.TextMessage, []byte(message))
        if err != nil {
            log.Println("Write error:", err)
            return
        }
    }

}

func startSocketServer() {
    http.HandleFunc("/", handleWebSocket)
    fmt.Println("Socket server listening on", MySocketPort)
    err := http.ListenAndServe(MyIP+MySocketPort, nil)
    if err != nil {
        log.Fatal("ListenAndServe:", err)
    }
}
```

Листинг 6: Файл sseServer.go

```
package main

import (
    "fmt"
    "log"
    "net/http"
)
var dataSseChannel = make(chan string)

// SSE handler function
func sseHandler(w http.ResponseWriter, r *http.Request) {
      // Set CORS headers to allow cross-origin requests
      w.Header().Set("Access-Control-Allow-Origin", "*") // Allow all
    origins
      w.Header().Set("Access-Control-Allow-Methods", "GET") // Allow GET
    requests
      w.Header().Set("Access-Control-Allow-Headers", "Content-Type") //
    Allow content-type headers
```

```go

    // Set headers for SSE
    w.Header().Set("Content-Type", "text/event-stream")
    w.Header().Set("Cache-Control", "no-cache")
    w.Header().Set("Connection", "keep-alive")

    // Create a channel for client disconnection
    clientGone := r.Context().Done()

    for {
        select {
            case message := <-dataSseChannel:
                _, err := fmt.Fprintf(w, "data: %s \n\n", MyIP + "$" +
    message)
                if err != nil {
                    fmt.Println("Not sended")
                    return
                }

                if f, ok := w.(http.Flusher); ok {
                    f.Flush()
                } else {
                    fmt.Println("Bad flush")
                }
            case <-clientGone:
                fmt.Println("SSE client disconnected")
                return
        }

    }

}

func startSseServer() {
    http.HandleFunc("/events", sseHandler)

    fmt.Println("SSE server started at ", MySsePort)
    log.Fatal(http.ListenAndServe(MyIP + MySsePort, nil))
}
```

Листинг 7: Файл start.sh

```bash
#!/bin/bash
export LOGXI=*
export LOGXI_FORMAT=pretty,happy
MyIP=$(hostname -I | awk '{ print $1 }' | tr -d '[:space:]')
export MyIP
```

```
echo "Building..."
if [[ -e fedukov_lab3 ]]; then
    rm fedukov_lab3
fi
go build -o fedukov_lab3
echo "Running..."
kill $(pgrep fedukov_lab3) > /dev/null 2>&1
./fedukov_lab3
```

Листинг 8: Файл runNetwork.sh

```
#!/bin/bash
# Scipt that uploads files of lab and launches them on ssh hosts
# Requirements: "tmux" to control ssh, existing ssh-keys or "sshpass" to generate keys
# IHAVESSHKEY=false to generate keys

SESSION="MySSH"
tmux kill-session -t $SESSION
tmux new-session -d -s $SESSION

SCRIPT_DIR=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev/null && pwd ) # Lab files dir. Default is script's dir
src=($SCRIPT_DIR/test.txt $SCRIPT_DIR/main.go) # Files to upload
path=/root/test/lab # Path to lab folder on server
startCommand="go run main.go" # Launch command
NetworkList=("yss1" "yss2" "yss3" "yss4") # List of ssh hosts (supposed to use ssh key)

# Options
clearMode=false # Remove lab dirs and exit
forceClear=false # Don't ask for confirmation before replacing lab files
justStart=false # Don't upload files, just execute $startCommand on server in tmux
IHAVESSHKEY=false # Set false and set IPS and PASSWORDS to generate ssh keys

# Read IPs and passwords into arrays
IPS=("185.104.251.226" "185.102.139.161" "185.102.139.168" "185.102.139.169")
PASSWORDS=("fMs0m69gIGQ3" "Up5b0A1wiLMQ" "gOsQ5p7FUJ9w" "w3Bt8hjge8oV")

function generateSSHKeys {

    # SSH configuration file
    SSH_CONFIG="$HOME/.ssh/config"
    SSH_KEY="$HOME/.ssh/id_rsa_for_lab"
```

```

# Function to check if host is already in SSH config
function host_in_ssh_config() {
    local host="$1"
    if grep -q "Host $host" "$SSH_CONFIG"; then
        return 0  # Found
    else
        return 1  # Not found
    fi
}

# Check if SSH keys exist, generate if not
if [ ! -f "$SSH_KEY" ]; then
    echo "Generating SSH key..."
    ssh-keygen -t rsa -b 4096 -N "" -f "$SSH_KEY"
fi

# Make sure both lists have the same length
if [ "${#IPS[@]}" -ne "${#PASSWORDS[@]}" ]; then
    echo "Error: IP list and password list must have the same number
of entries."
    exit 1
fi

useIpsAsHostNames=false
if [ "${#IPS[@]}" -ne "${#NetworkList[@]}" ]; then
    echo "Not enough names in NetworkList. Using ip as a name"
    useIpsAsHostNames=true
fi

# Backup existing SSH config file
if [ -f "$SSH_CONFIG" ]; then
    local i=1

    while [ -f "${SSH_CONFIG}.bak.$i" ]; do
        i=$((i + 1))
    done

    cp "$SSH_CONFIG" "${SSH_CONFIG}.bak.$i"
    echo "Backup of SSH config created at ${SSH_CONFIG}.bak.$i"
fi

# Create SSH config entries or skip if already exists
for i in "${!IPS[@]}"; do
    IP="${IPS[$i]}"
    PASSWORD="${PASSWORDS[$i]}"
```

```
        if [ $useIpsAsHostNames = true ]; then
            HOST=$IP
        else
            HOST="${NetworkList[$i]}"
        fi

        if host_in_ssh_config "$HOST"; then
            echo "Host $HOST is already in SSH config, skipping..."
            continue
        fi

        echo "Setting up passwordless SSH for $HOST ($IP)..."

        # Copy SSH key to remote server using sshpass
        sshpass -p "$PASSWORD" ssh-copy-id -i "$SSH_KEY.pub" "root@$IP"

        # Add SSH config entry
        echo "Host $HOST" >> "$SSH_CONFIG"
        echo "  HostName $IP" >> "$SSH_CONFIG"
        echo "  User root" >> "$SSH_CONFIG"
        echo "  IdentityFile $SSH_KEY" >> "$SSH_CONFIG"
        echo "  IdentitiesOnly yes" >> "$SSH_CONFIG"
        echo "  StrictHostKeyChecking no" >> "$SSH_CONFIG"      # It
    causes "Warning: Permanently added"
        echo "  UserKnownHostsFile /dev/null" >> "$SSH_CONFIG"  # Don't
    write to known hosts
    done

    echo "SSH configuration updated at $SSH_CONFIG"
}

if [ $IHAVESSHKEY = false ]; then
    generateSSHKeys
fi

function addTmuxWindow {
    i=$1 # Vps id
    vps=$2 # Vps name
    path=$3 # Lab dir path

    # Launch program in tmux via ssh
    tmux new-window -t $SESSION:$((i + 1)) -n ${NetworkList[i]} ssh $vps
     "cd $path; source ~/.profile; $startCommand"

    echo "Tmux windows created"
}
```

```

function isDirEmpty {
    vps=$1
    path=$2
    [ $(ssh $vps "ls -1 $path | wc -l") -eq "0" ]
}

function doesDirExist {
    vps=$1
    path=$2
    ssh $vps "[ -e $path ]"
}

function removeDir {
    vps=$1
    path=$2

    if [ $forceClear = false ]; then
        echo "Clean folder?"
        echo "ls -a $path"
        echo "----"
        ssh $vps "ls -a $path"
        echo "----"
        select yn in "Yes" "No"; do
            case $yn in
                Yes) break;;
                No) echo "Skip"; return;;
            esac
        done

    fi

    ssh $vps "rm -rf $path/*"
    echo "Cleaned folder"
}

# Check src files existence
for file in ${src[*]}
do
    if [ ! -e $file ]; then
        echo -e "\nFile not found: $file\n"
    fi
done


for i in ${!NetworkList[*]}
```

```
do
    echo "Processing on ${NetworkList[i]} ..."
    vps=${NetworkList[i]}

    if [ $clearMode = true ]; then
        removeDir $vps $path
    else

        if ! doesDirExist $vps $path; then
            echo "$path not found"
            ssh $vps "mkdir -p $path"
            echo "Created $path"
        fi

        if [ $justStart = true ]; then
            if ! isDirEmpty $vps $path; then
                addTmuxWindow $i $vps $path
            else
                echo "No files to start. Skip"
            fi
        else
            removeDir $vps $path
            echo "Loading files"
            scp ${src[*]} $vps:$path

            addTmuxWindow $i $vps $path
        fi
    fi
done

if [ "$clearMode" = false ]; then
    echo "Starting tmux"
    # Launch tmux in gnome-terminal
    gnome-terminal -- tmux attach -t $SESSION
else
    echo "Killing tmux"
    tmux kill-session -t $SESSION
fi
```

Листинг 9: Файл webClient.html

```
<html>

<head>
  <title>DashBoard</title>
  <!-- <link rel="stylesheet" href="style.css" type="text/css"/> -->
  <meta charset="utf8">
```

```
<title>My dashboard</title>
<style>
  body {
    margin: 0;
    padding: 0;
    background-color: #f0f0f0;
    font-family: Arial, sans-serif;
    display: flex;

  }

  .container, .cont3 {
    text-align: center;
    background-color: #f5f5f5;
    align-items: center;
    padding: 10px;
    border: 1px solid #ccc;
    margin: 2.5vh;
    max-width: fit-content;
    margin-inline: auto; /*Delete?*/
    margin-left: 40%;
    min-height: 90vh;

  }

  .container {
    height: fit-content;
  }

  .cont3 {
    max-height: 90vh;

    position: relative;
    margin-left: 10%;
    width: 100%;

    border: 1px solid #ccc;
    background-color: #f9f9f9;
    padding: 10px;
    box-sizing: border-box;
    min-width: 50vh;


  }

  .scrollable {
    max-height: 75vh;
    overflow-y: auto;
    background: radial-gradient(white, #eee);
  }

  .text-box {
    border: 1px solid #ccc;
    background-color: #ffffff;
    padding: 10px;
    margin: 2vh 0;
    padding-bottom: 20px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
    border-radius: 8px;
    overflow-y: auto;
    display: grid;

  }

  .text-box:last-child,
  .text-box:first-child {
    margin: 0;
  }

  .text-box b {
    font-size: 1.5rem;
    color: #333;
    margin: 0;
  }

  .text-box input,
  select {
    padding: 10px;

    font-size: 1rem;
    margin-top: 10px;
    margin-bottom: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
    width: 80%;
    justify-self: center;

  }

  button {
    padding: 10px 20px;
    background-color: #444;
```

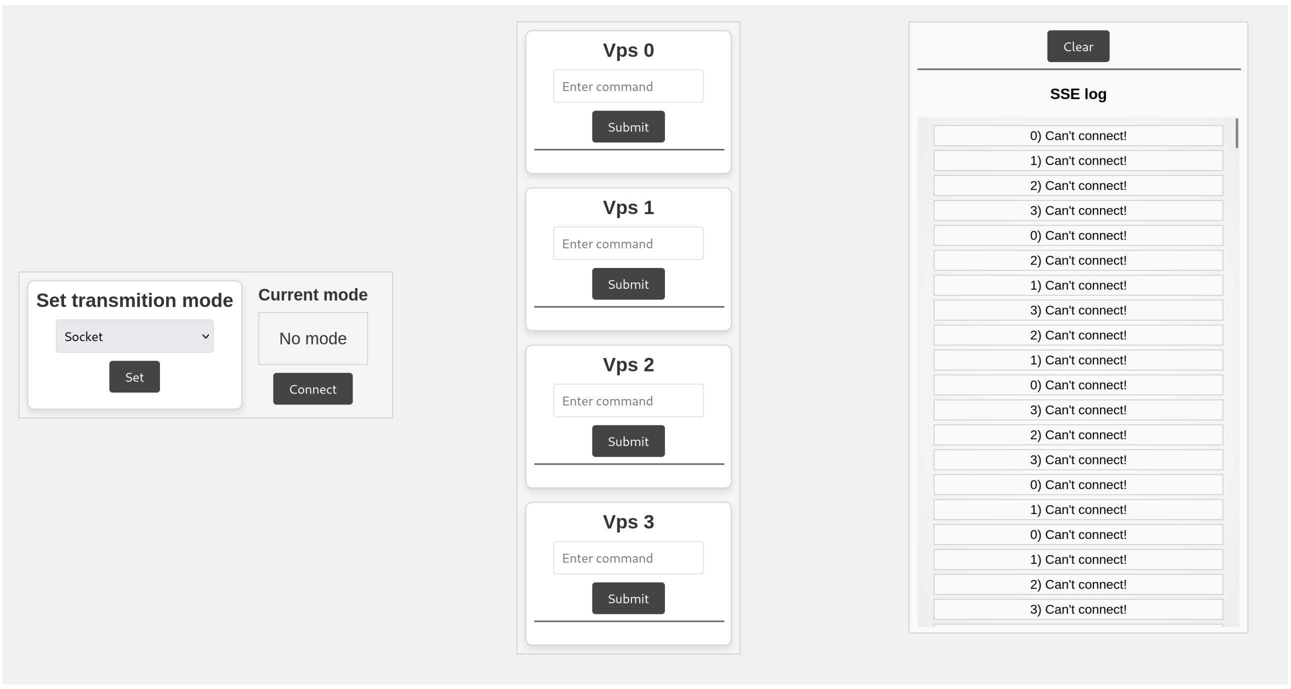
```
      color: white;
      border: none;
      border-radius: 4px;
      font-size: 1rem;
      cursor: pointer;
      transition: background-color 0.3s ease;
      justify-self: center;
    }

    button:hover {
      background-color: #000;
    }

    .box3 button {
      margin-top: 10px;
      margin-left: auto;
      margin-right: auto;
    }

    .cont2 {
      position: fixed;
      text-align: left;
      height: auto;
      transform: translateY(-50%);
      top: 50%;
      left: 20px;
      background-color: #f5f5f5;

      display: flex;
      justify-content: center;
      align-items: center;
      padding: 10px;
      border: 1px solid #ccc;
    }

    .box3 {
      padding: 20px;
      font-size: 1.3rem;

      color: #333;
      margin: 0;
      text-align: center;

      display: flex;
      flex-direction: column;
      height: 150px;
      box-sizing: border-box;
      justify-content: center;

      flex: 1;

    }

    .box3 span {
      border: 1px solid #ccc;
      padding: 20px;
      height: auto;
      display: flex;
      justify-content: center;
      flex-grow: 1;
    }

    .box3 strong {
      padding-bottom: 10px;
    }

    .box2 {
      text-align: center;
    }

    .box2 b {
      font-size: 1.5rem;
    }

    #messageContainer {
    display: grid;
    grid-template-columns: 1fr;
    gap: 5px;
    padding-left: 20px;
    padding-right: 20px;
    }

    .log_toremove {
      border: 1px solid #ccc;
      padding: 3px;
      max-width: 400px;
      background-color: #f9f9f9;
    }
    .log_toremove:first-child {
      margin-top: 10px;
    }
    .log_toremove:last-child {
      margin-bottom: 10px;
    }

  </style>
  <script>

    const ips = ["185.104.251.226", "185.102.139.161", "185.102.139.168"
    , "185.102.139.169"]
    const wsPort = 1494
    const httpPort = 1491
    const ssePort = 1497
    let msgCounters = [0, 0, 0, 0]
    let transmition = ""
    const psblTr = ["Get", "Post", "Socket"]

    let sockets = []


    function writeData(i, message) {
      if (msgCounters[i] > 5) {
        document.getElementById(i + " message").remove()
        msgCounters[i]--;
      }

      let messageDiv = document.createElement("div");
      messageDiv.id = i + " message"
      messageDiv.className = "toremove"
      messageDiv.textContent = message;

      // Display the message in the #messages div
      document.getElementById("id" + i).append(messageDiv);

      msgCounters[i]++;
    }

    function startSocket(i) {

      sockets[i] = new WebSocket("ws://" + ips[i] + ":" + wsPort)
        const socket = sockets[i];

        socket.onopen = function (e) {
          console.log("Socket opened")
        };

        socket.onmessage = function (event) {
          console.log(`Socket mesage to ${i}: ${event.data}`)
          // Remove old messages
          writeData(i, event.data)

        };

        socket.onclose = function (event) {
          console.log("closed")
          if (event.wasClean) {
            //alert(`[close] Соединение закрыто чисто, код=${event.code}
    причина=${event.reason}`);
            // socket.send("UPDATE");

          } else {
            console.log("by server")
            // например, сервер убил процесс или сеть недоступна
            // обычно в этом случае event.code 1006
            //alert('[close] Соединение прервано');
          }
        };

        socket.onerror = function (error) {
          alert(`[error] ${error.message}`);
        };
      }

    function updateQuery(ip_index) {
      const params = getInput(ip_index).split(" ")
      const urlParams = new URLSearchParams();
      urlParams.append("vps_id", ip_index)
      urlParams.append('command', params[0]);

      switch (params[0]) {
        case "getSum":
          urlParams.append('i1', params[1]);
          urlParams.append('i2', params[2]);
          break;
        case "setValue":
          urlParams.append('i1', params[1]);
          urlParams.append('i2', params[2]);
          break;
        default:
          break;
      }

    // Update the URL with the new query parameters
```

```
    const newUrl = `${window.location.pathname}?${urlParams.toString()
   }`;
    window.history.pushState({}, '', newUrl);

    console.log("Updated query")

    }

    async function sendGetRequest() {
      // Get WebForm query params
      const urlParams = new URLSearchParams(window.location.search);
      let ip_index = parseInt(urlParams.get("vps_id"))

      // Create a URL with query parameters
      const url = new URL('http://' + ips[ip_index] + ":" + httpPort +
   '/get' + `?${urlParams.toString()}`);

      try {
        const response = await fetch(url);
        const data = await response.json();
        writeData(ip_index, data.message)
      } catch (error) {
        alert("Http Get error")
      }
    }
    // Function to send a POST request
    async function sendPostRequest(ip_index) {
      try {
        const params = getInput(ip_index).split(" ")
        console.log(params)
        const response = await fetch('http://' + ips[ip_index] + ":" +
   httpPort + '/post', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify({
            "command": params[0],
            "i1": params[1],
            "i2": params[2],
          })
        });

        const data = await response.json();
        writeData(ip_index, data.message)

      } catch (error) {
        alert("Http Post error")

      }
    }

    function clearVpsLog(index) {
      let v = document.getElementById("inp" + index).value
      if (v == "clear") {
        els = document.getElementsByClassName("toremove")
        Array.from(els).forEach((el) => {
          el.remove()
        });

        msgCounters = [0, 0, 0, 0]
        console.clear()
        return true
      }
      return false
    }

    function getInput(index) {
      let v = document.getElementById("inp" + index).value
      if (v == "clear") {
        els = document.getElementsByClassName("toremove")
        Array.from(els).forEach((el) => {
          el.remove()
        });

        msgCounters = [0, 0, 0, 0]
        console.clear()
        return ""
      } else {
        console.log("Command: \"" + v + "\" sended to vps " + index)
        return v
      }
    }

    function sendToSocket(index) {
      sockets[index].send(getInput(index))
    }

    function send(ind) {
      if (!clearVpsLog(ind)){
        switch (transmition) {
          case "Get":
            updateQuery(ind)
            break;
          case "Post":
            sendPostRequest(ind)
            break;
          case "Socket":
            sendToSocket(ind)
            break;
          default:
            console.error("Choose right transmition option!")
            alert("No mode")
            return
        }
        console.log("By", transmition)
      }
    }


    function setTrMode() {
      let mode = document.getElementById("trmode").value
      updatePushButton()

      if (psblTr.includes(mode)) {
        transmition = mode
        document.getElementById("currentMode").innerHTML = transmition
        console.log("Switching to transmission mode: " + mode)

      } else {
        alert("Bad mode!")
      }
    }

    function connect() {
      let mode = document.getElementById("currentMode").innerHTML

      // Secon connect by get is send
      if (mode == "Get") {
          sendGetRequest()
          return
      }

      if (mode == "Socket") {
        // let btn = document.getElementById("btn_connect")
        // btn.disabled = true;
        // btn.style.opacity = 0.5
        return
      }

    }

    function writeToLog(message, ip_index) {
      const messageContainer = document.getElementById('messageContainer
   ');
      if (ip_index != undefined){
        ip_index += ") "
      } else {
        ip_index = ""
      }

      // Create a new div for each message
      const newMessageDiv = document.createElement('div');
      newMessageDiv.className = "log_toremove"
      newMessageDiv.innerText = ip_index + message;

      messageContainer.appendChild(newMessageDiv);
    }

    function clearLog() {
      els = document.getElementsByClassName("log_toremove")
        Array.from(els).forEach((el) => {
          el.remove()
          return ""
        });
      console.log("SSE log cleared")
    }
   // Open an SSE connection to the server
   function startSSE(ip_index) {
      const eventSource = new EventSource('http://' + ips[ip_index] + ":
   " + ssePort + '/events');

      // Listen for messages from the server
      eventSource.onmessage = function(event) {
          const message = event.data;
          let splitSymbolInd = message.indexOf("$")
          let ip = message.slice(0, splitSymbolInd)
          let odata = message.slice(splitSymbolInd + 1)
          console.log(`SSE message from ${ip}: ${odata}`)
          writeToLog(odata, ips.indexOf(ip))
      };

      // Handle SSE errors
      eventSource.onerror = function(event) {
          console.error(`SSE error on ${ips[ip_index]}`);
          writeToLog("Can't connect!", ip_index)
          setTimeout(startSSE(ip_index), 2000);
      };

      eventSource.onopen = function(event){
        console.log("SSE connected")
        writeToLog("SSE connected!", ip_index)
      }

  }


  function updatePushButton() {
    const selectedValue = document.getElementById("trmode").value;
        let btn = document.getElementById("btn_connect")
        btn.disabled = false;
        btn.style.opacity = 1
        switch (selectedValue) {
          case "Get":
            btn.innerHTML = "Send request"
            break;
          case "Post":
              btn.disabled = true;
              btn.innerHTML = "Automatically"
              btn.style.opacity = 0.5
              break
          case "Socket":
            btn.innerHTML = "Connect"
            btn.disabled = true;
            btn.style.opacity = 0.5
          default:
            break;
        }

  }

  window.onload = () => {
    for (let i = 0; i < ips.length; i++) {
      startSSE(i)
      startSocket(i)
    }
  }


//    // Test data for logs
//    window.addEventListener('load', function() {
//      for (let index = 0; index < 100; index++) {
//        writeToLog(index)
//        writeData(3, index)
//        writeData(0, index)
//      }
// });

  </script>

</head>

<body>

  <div class="cont2">
    <div class="text-box box2">
      <b>Set transmition mode</b>
      <select name="transMode" id="trmode">
        <option value="Socket">Socket</option>
        <option value="Get">Get</option>
        <option value="Post">Post</option>
      </select>

      <button onclick="setTrMode()">Set</button>
    </div>
    <div class="box3">
      <strong>Current mode</strong>
      <span id="currentMode">No mode</span>
      <button onclick="connect()" id="btn_connect">Connect</button>
    </div>
  </div>

  <div class="container">
    <div id="id0" class="text-box">
      <b> Vps 0 </b>
      <input id="inp0" type="text" placeholder="Enter command">
      <button onclick="send(0)">Submit</button>
      <hr style="width: 100%;">
    </div>
    <div id="id1" class="text-box">
      <b> Vps 1 </b>
      <input id="inp1" type="text" placeholder="Enter command">
      <button onclick="send(1)">Submit</button>
      <hr style="width: 100%;">
    </div>
    <div id="id2" class="text-box">
```

```
        <b> Vps 2 </b>
        <input id="inp2" type="text" placeholder="Enter command">
        <button onclick="send(2)">Submit</button>
        <hr style="width: 100%;">
      </div>
      <div id="id3" class="text-box">
        <b> Vps 3 </b>
        <input id="inp3" type="text" placeholder="Enter command">
        <button onclick="send(3)"> Submit</button>
        <hr style="width: 100%;">
      </div>
    </div>
    <div class="cont3">
      <div>
        <button type="reset" onclick="clearLog()">Clear</button>
      </div>
      <hr style="width: 100%;">
      <div style="position: relative;">
        <h3>SSE log</h3>
        <div>
          <div class="scrollable">
              <div id="messageContainer" style="overflow-y: auto;"></div>
          </div>
        </div>
      </div>
    </div>
</body>

</html>
```

Вывод программы

После запуска runNetwork.sh, загрузки файлов и создания сети, при помощи webClient.html можно управлять узлами пиринговой сети.



```
0) Can't connect!
1) Can't connect!
2) Can't connect!
3) Can't connect!
0) Can't connect!
2) Can't connect!
1) Can't connect!
3) Can't connect!
2) Can't connect!
1) Can't connect!
0) Can't connect!
3) Can't connect!
2) Can't connect!
3) Can't connect!
0) Can't connect!
1) Can't connect!
0) Can't connect!
1) Can't connect!
2) Can't connect!
3) Can't connect!
```

Вывод

Я научился использовать tcp протокол, продолжил познавать http, а также поработал с websocket на Go. Я изучил модель пиринговой сети, передавал данные с одного сервера на другой.