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

```
1  package main
2
3  import (
4    "encoding/json"
5    "fmt"
6    "os"
7    "time"
8    "strconv"
9  )
10
11 type Request struct {
12   Command string `json:"command"`
13
14   Data *json.RawMessage `json:"data"`
15 }
16
17 type Response struct {
18   Status string `json:"status"`
19
20   Data *json.RawMessage `json:"data"`
21 }
22
23 type jsonInt struct {
24   N          int `json:"n"`
25   StartIndex int `json:"startindex"`
26   EndIndex   int `json:"endindex"`
27 }
28
29 type jsonSum struct {
30   N          int `json:"n"`
31   M          int `json:"m"`
32   StartIndex int `json:"startindex"`
33 }
34
35 var MyPort = ":1572"
36 var MySocketPort = ":1494"
```

```
37 var MyHttpPort = ":1491"
38 var MySsePort = ":1497"
39 var MyIP = os.Getenv("MyIP")
40 var NetworkList = [...]string{"185.104.251.226", "185.102.139.161", "
       185.102.139.168", "185.102.139.169"}
41 var MyIndex int
42 var NeigborIndex int
43 var ConnectionMode = "Socket"
44 var MyValue = 1
45 var RequestedValue int
46 var SocketCommand string
47
48 type WebFormRequestParsed struct {
49   command string
50   data    interface{}
51 }
52
53 var WebFormRequest WebFormRequestParsed
54
55 func sendToWebForm(message string, mode string) {
56   switch mode {
57   case "Socket":
58     dataSocketChannel <- message
59   case "SSE":
60     dataSseChannel <- message
61   }
62 }
63
64 // Run commands from "c" and write response into "out"
65 func parseFromForm(out *string, c ...string)  {
66   var data interface{}
67
68   switch c[0] {
69   case "setValue":
70     i1, _ := strconv.Atoi(c[1])
71     i2, _ := strconv.Atoi(c[2])
72     data = &jsonInt{EndIndex: i1, N: i2, StartIndex: MyIndex}
73     WebFormRequest.command = c[0]
74     WebFormRequest.data = data
75     interactWithWebForm(Connection, out)
76   case "getSum":
77     i1, _ := strconv.Atoi(c[1])
78     i2, _ := strconv.Atoi(c[2])
79     data = &jsonSum{N: i1, M: i2, StartIndex: MyIndex}
80     WebFormRequest.command = c[0]
81     WebFormRequest.data = data
```

```
82      interactWithWebForm(Connection, out)
83    case "showValue":
84      fmt.Println(MyValue)
85      *out = *out + strconv.Itoa(MyValue)
86    default:
87      fmt.Println("Unknown command from WebForm")
88      SocketCommand = ""
89      *out = *out + "Bad command!"
90
91    }
92
93  }
94  func main() {
95    // Set IP
96    for i, v := range NetworkList {
97      if MyIP == v {
98        MyIndex = i
99        NeigborIndex = (i + 1) % len(NetworkList)
100       break
101     }
102   }
103
104   fmt.Println("MyIP:", NetworkList[MyIndex])
105   fmt.Println("NeigborIP:", NetworkList[NeigborIndex])
106   fmt.Println("Port:", MyPort)
107
108   go startSocketServer()
109   time.Sleep(1 * time.Second)
110   go startHttpServer()
111   time.Sleep(1 * time.Second)
112   go startSseServer()
113   time.Sleep(1 * time.Second)
114   go startServer(MyIP + MyPort)
115   time.Sleep(5 * time.Second)
116   fmt.Println("Don't forget to reload a page!")
117   go startClient(NetworkList[NeigborIndex] + MyPort)
118   select {}
119
120 }
```

Листинг 2: Файл server.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "fmt"
```

```
6     "math/big"
7     "net"
8
9     log "github.com/mgutz/logxi/v1"
10 )
11
12 // Client - состояние клиента.
13 type Client struct {
14     logger log.Logger     // Объект для печати логов
15     conn   *net.TCPConn   // Объект TCP-соединения
16     enc    *json.Encoder // Объект для кодирования и отправки сообщений
17     sum    *big.Rat      // Текущая сумма полученных от клиента дробей
18     count  int64         // Количество полученных от клиента дробей
19 }
20
21 // NewClient - конструктор клиента, принимает в качестве параметра
22 // объект TCP-соединения.
23 func NewClient(conn *net.TCPConn) *Client {
24     return &Client{
25         logger: log.New(fmt.Sprintf("client %s", conn.RemoteAddr().String())
          ),
26         conn:   conn,
27         enc:    json.NewEncoder(conn),
28         sum:    big.NewRat(0, 1),
29         count:  0,
30     }
31 }
32
33 // serve - метод, в котором реализован цикл взаимодействия с клиентом.
34 // Подразумевается, что метод serve будет вызываться в отдельной go-прог
      рамме.
35 func (client *Client) serve() {
36     defer client.conn.Close()
37     decoder := json.NewDecoder(client.conn)
38     for {
39         var req Request
40         if err := decoder.Decode(&req); err != nil {
41             client.logger.Error("cannot decode message", "reason", err)
42             break
43         } else {
44             client.logger.Info("received command", "command", req.Command)
45             if client.handleRequest(&req) {
46                 client.logger.Info("shutting down connection")
47                 break
48             }
49         }
```

```
50    }
51  }
52
53  func goPeer(addrStr string, command string, data interface{}, args ...
        int) {
54    if addrStr == "next" {
55      addrStr = NetworkList[NeigborIndex] + MyPort
56    } else if addrStr == "back" {
57      addrStr = NetworkList[args[0]] + MyPort
58    }
59
60    if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
61      log.Error("cannot resolve addres to connect", "adress", addrStr, "
        reason", err)
62    } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
63      log.Error("cannot establish connection to")
64    } else {
65      log.Info("establish connection to", "address", conn.RemoteAddr().
        String())
66      encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
67      send_request(encoder, command, data)
68
69      // Получение ответа.
70      var resp Response
71      if err := decoder.Decode(&resp); err != nil {
72        fmt.Printf("error: %v\n", err)
73      }
74      // Вывод ответа в стандартный поток вывода.
75      switch resp.Status {
76      case "ok":
77        fmt.Printf("Data sended\n")
78      case "failed":
79        if resp.Data == nil {
80          fmt.Printf("error: while next peer data field is absent in
        response\n")
81        } else {
82          var errorMsg string
83          if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
84            fmt.Printf("error: malformed data field in response\n")
85          } else {
86            fmt.Printf("failed: %s\n", errorMsg)
87          }
88        }
89      default:
90        fmt.Printf("error: server reports unknown status %q\n", resp.
        Status)
```

```
91      }
92    }
93  }
94
95  // handleRequest - метод обработки запроса от клиента. Он возвращает
        true,
96  // если клиент передал команду "quit" и хочет завершить общение.
97  func (client *Client) handleRequest(req *Request) bool {
98    sendToWebForm("Server command: " + req.Command, "SSE")
99    switch req.Command {
100   case "quit":
101     client.respond("ok", nil)
102     return true
103   case "getValue":
104     // by id
105     errorMsg := ""
106     if req.Data == nil {
107       errorMsg = "data field is absent"
108     } else {
109       var s jsonSum
110       if err := json.Unmarshal(*req.Data, &s); err != nil {
111         errorMsg = "malformed data field"
112       } else {
113         if MyIndex == s.N {
114           fmt.Printf("\nReturned MyValue to index %d\n", s.StartIndex)
115           goPeer("back", "giveRequestedValue", &jsonInt{N: MyValue,
        StartIndex: MyIndex, EndIndex: s.StartIndex}, s.StartIndex)
116         } else {
117           fmt.Println("Going to next peer")
118           goPeer("next", "getValue", req.Data)
119         }
120
121         sendToWebForm(fmt.Sprintf("Geting value on to %d index", s.
        StartIndex), "SSE")
122       }
123     }
124
125     if errorMsg == "" {
126       client.respond("ok", nil)
127     } else {
128       client.logger.Error("addition failed", "reason", errorMsg)
129       client.respond("failed", errorMsg)
130     }
131
132   case "setValue":
133     // by id
```

```
134     errorMsg := ""
135     if req.Data == nil {
136       errorMsg = "data field is absent"
137     } else {
138       var s jsonInt
139       if err := json.Unmarshal(*req.Data, &s); err != nil {
140         errorMsg = "malformed data field"
141       } else {
142         if s.EndIndex == MyIndex {
143           MyValue = s.N
144           fmt.Printf("\nMyValue is updated to %d by index %d\n", MyValue
      , s.StartIndex)
145         } else {
146           goPeer("next", "setValue", req.Data)
147         }
148         sendToWebForm(fmt.Sprintf("Updating value on %d index to %d", s.
      EndIndex, s.N), "SSE")
149       }
150     }
151
152     if errorMsg == "" {
153       client.respond("ok", nil)
154     } else {
155       client.logger.Error("addition failed", "reason", errorMsg)
156       client.respond("failed", errorMsg)
157     }
158
159   case "giveRequestedValue":
160     // by id
161     errorMsg := ""
162     if req.Data == nil {
163       errorMsg = "data field is absent"
164     } else {
165       var s jsonInt
166       if err := json.Unmarshal(*req.Data, &s); err != nil {
167         errorMsg = "malformed data field"
168       } else {
169         RequestedValue = s.N
170         fmt.Printf("\nGot value %d from index %d\n", RequestedValue, s.
      StartIndex)
171         sendToWebForm(fmt.Sprintf("Give value %d", s.N), "SSE")
172       }
173     }
174
175     if errorMsg == "" {
176       client.respond("ok", nil)
```

```
177     } else {
178       client.logger.Error("addition failed", "reason", errorMsg)
179       client.respond("failed", errorMsg)
180     }
181
182   default:
183     client.logger.Error("unknown command")
184     client.respond("failed", "unknown command")
185   }
186   return false
187 }
188
189 // respond - вспомогательный метод для передачи ответа с указанным стату
      ком
190 // и данными. Данные могут быть пустыми (data == nil).
191 func (client *Client) respond(status string, data interface{}) {
192   var raw json.RawMessage
193   raw, _ = json.Marshal(data)
194   client.enc.Encode(&Response{status, &raw})
195 }
196
197 func startServer(addrStr string) {
198   if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
199     log.Error("address resolution failed", "address", addrStr)
200   } else {
201     log.Info("resolved TCP address", "address", addr.String())
202
203     // Инициация слушания сети на заданном адресе.
204     if listener, err := net.ListenTCP("tcp", addr); err != nil {
205       log.Error("listening failed", "reason", err)
206     } else {
207       // Цикл приёма входящих соединений и обработки запросов.
208       for {
209         if conn, err := listener.AcceptTCP(); err != nil {
210           log.Error("cannot accept connection", "reason", err)
211         } else {
212           log.Info("accepted connection", "address", conn.RemoteAddr().
      String())
213           // Запуск go-программы для обслуживания клиентов.
214           go NewClient(conn).serve()
215
216         }
217       }
218     }
219   }
220 }
```

Листинг 3: Файл client.go

```
1  package main
2
3  import (
4    "encoding/json"
5    "fmt"
6    "net"
7    "strconv"
8    "time"
9
10   log "github.com/mgutz/logxi/v1"
11   "github.com/skorobogatov/input"
12 )
13
14 var Connection *net.TCPConn
15
16 func interactWithWebForm(conn *net.TCPConn, webFormResponse *string) {
17   encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
18
19   switch WebFormRequest.command {
20   case "quit":
21     send_request(encoder, "quit", nil)
22     return
23   case "help":
24     *webFormResponse = *webFormResponse + "You should use client after
     connection to server!"
25     *webFormResponse = *webFormResponse + "showValue - prints current
     index value"
26     *webFormResponse = *webFormResponse + "setValue - define a value of
     given index"
27     *webFormResponse = *webFormResponse + "getSum - count sum from n to
     m indexes"
28   case "showValue":
29     fmt.Println(MyValue)
30     *webFormResponse = *webFormResponse + strconv.Itoa(MyValue)
31   case "setValue":
32     // by id
33     fmt.Println("Updating value by webForm")
34     send_request(encoder, "setValue", WebFormRequest.data)
35     *webFormResponse = *webFormResponse + "Sended setValue request"
36
37   case "getSum":
38     // from n to m
39     csum := 0
40     cstart := WebFormRequest.data.(*jsonSum).N
41     if WebFormRequest.data.(*jsonSum).N == MyIndex {
```

```
42          csum += MyValue
43          cstart = WebFormRequest.data.(*jsonSum).N + 1
44        }
45        fmt.Println("Requests for sum...")
46
47        for i := cstart; i <= WebFormRequest.data.(*jsonSum).M; i++ {
48          // Запрос значения
49          send_request(encoder, "getValue", &jsonSum{N: i, StartIndex:
      MyIndex})
50          // Получение ответа.
51          var resp Response
52          if err := decoder.Decode(&resp); err != nil {
53            fmt.Printf("error: %v\n", err)
54            break
55          } else {
56            if resp.Status != "ok" {
57              fmt.Printf("error: data field is absent in response\n")
58            } else {
59              fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
60              csum += RequestedValue
61            }
62          }
63        }
64        fmt.Printf("Sum is %d\n", csum)
65        *webFormResponse = *webFormResponse + fmt.Sprintf("Sum is %d\n",
      csum)
66
67      default:
68        fmt.Printf("error: unknown command\n")
69        *webFormResponse = *webFormResponse + "error: unknown command"
70      }
71      WebFormRequest = WebFormRequestParsed{}
72  }
73
74  // interact - функция, содержащая цикл взаимодействия с сервером.
75  func interact(conn *net.TCPConn) {
76    defer conn.Close()
77    encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
78    for {
79      // Чтение команды из стандартного потока ввода
80      fmt.Printf("\ncommand = ")
81      command := input.Gets()
82
83      fmt.Println("Send command to WebForm: ", command)
84      go sendToWebForm("Client command: " + command, "SSE")
85
```

```
86      // Отправка запроса.
87      switch command {
88      case "quit":
89        send_request(encoder, "quit", nil)
90        return
91      case "help":
92        fmt.Println("You should use client after connection to server!")
93        fmt.Println("showValue - prints current index value")
94        fmt.Println("setValue - define a value of given index")
95        fmt.Println("getSum - count sum from n to m indexes")
96        continue
97      case "showValue":
98        fmt.Println(MyValue)
99        continue
100     case "setValue":
101       // by id
102       fmt.Printf("Index = ")
103       if valInd, err := strconv.Atoi(input.Gets()); err != nil {
104         fmt.Println("\nMust be number!")
105         continue
106       } else {
107         fmt.Printf("Value = ")
108         if valN, err := strconv.Atoi(input.Gets()); err != nil {
109           fmt.Println("\nMust be number!")
110           continue
111         } else {
112           send_request(encoder, "setValue", &jsonInt{N: valN, StartIndex
      : MyIndex, EndIndex: valInd})
113         }
114       }
115
116     case "getSum":
117       // from n to m
118       fmt.Printf("From index = ")
119       if valN, err := strconv.Atoi(input.Gets()); err != nil {
120         fmt.Println("\nMust be number!")
121         continue
122       } else {
123         fmt.Printf("To index = ")
124         if valM, err := strconv.Atoi(input.Gets()); err != nil {
125           fmt.Println("\nMust be number!")
126           continue
127         } else {
128           csum := 0
129           cstart := valN
130           if valN == MyIndex {
```

```
131             csum += MyValue
132             cstart = valN + 1
133           }
134           fmt.Println("Requests for sum...")
135           for i := cstart; i <= valM; i++ {
136             // Запрос значения
137             send_request(encoder, "getValue", &jsonSum{N: i, StartIndex:
      MyIndex})
138             // Получение ответа.
139             var resp Response
140             if err := decoder.Decode(&resp); err != nil {
141               fmt.Printf("error: %v\n", err)
142               break
143             } else {
144               if resp.Status != "ok" {
145                 fmt.Printf("error: data field is absent in response\n")
146               } else {
147                 fmt.Printf("Recieved %d from index %d\n", RequestedValue
      , i)
148                 csum += RequestedValue
149               }
150             }
151           }
152           fmt.Printf("Sum is %d\n", csum)
153           continue
154         }
155       }
156
157     default:
158       fmt.Printf("error: unknown command\n")
159       // Quit for Socket interaction
160       continue
161     }
162
163     // Получение ответа.
164     var resp Response
165     if err := decoder.Decode(&resp); err != nil {
166       fmt.Printf("error: %v\n", err)
167       break
168     }
169
170     // Вывод ответа в стандартный поток вывода.
171     switch resp.Status {
172     case "ok":
173       fmt.Printf("ok\n")
174     case "failed":
```

```
175         if resp.Data == nil {
176           fmt.Printf("error: data field is absent in response\n")
177         } else {
178           var errorMsg string
179           if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
180             fmt.Printf("error: malformed data field in response\n")
181           } else {
182             fmt.Printf("failed: %s\n", errorMsg)
183           }
184         }
185       default:
186         fmt.Printf("error: server reports unknown status %q\n", resp.
      Status)
187       }
188
189     }
190 }
191
192 // send_request - вспомогательная функция для передачи запроса с указанн
      ой командой
193 // и данными. Данные могут быть пустыми (data == nil).
194 func send_request(encoder *json.Encoder, command string, data interface
      {}) {
195     var raw json.RawMessage
196     raw, _ = json.Marshal(data)
197     encoder.Encode(&Request{command, &raw})
198 }package main
199
200 import (
201     "encoding/json"
202     "fmt"
203     "net"
204     "strconv"
205     "time"
206
207     log "github.com/mgutz/logxi/v1"
208     "github.com/skorobogatov/input"
209 )
210
211 var Connection *net.TCPConn
212
213 func interactWithWebForm(conn *net.TCPConn, webFormResponse *string) {
214     encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
215
216     switch WebFormRequest.command {
217     case "quit":
```

```
218     send_request(encoder, "quit", nil)
219     return
220   case "help":
221     *webFormResponse = *webFormResponse + "You should use client after
      connection to server!"
222     *webFormResponse = *webFormResponse + "showValue - prints current
      index value"
223     *webFormResponse = *webFormResponse + "setValue - define a value of
      given index"
224     *webFormResponse = *webFormResponse + "getSum - count sum from n to
      m indexes"
225   case "showValue":
226     fmt.Println(MyValue)
227     *webFormResponse = *webFormResponse + strconv.Itoa(MyValue)
228   case "setValue":
229     // by id
230     fmt.Println("Updating value by webForm")
231     send_request(encoder, "setValue", WebFormRequest.data)
232     *webFormResponse = *webFormResponse + "Sended setValue request"
233
234   case "getSum":
235     // from n to m
236     csum := 0
237     cstart := WebFormRequest.data.(*jsonSum).N
238     if WebFormRequest.data.(*jsonSum).N == MyIndex {
239       csum += MyValue
240       cstart = WebFormRequest.data.(*jsonSum).N + 1
241     }
242     fmt.Println("Requests for sum...")
243
244     for i := cstart; i <= WebFormRequest.data.(*jsonSum).M; i++ {
245       // Запрос значения
246       send_request(encoder, "getValue", &jsonSum{N: i, StartIndex:
      MyIndex})
247       // Получение ответа.
248       var resp Response
249       if err := decoder.Decode(&resp); err != nil {
250         fmt.Printf("error: %v\n", err)
251         break
252       } else {
253         if resp.Status != "ok" {
254           fmt.Printf("error: data field is absent in response\n")
255         } else {
256           fmt.Printf("Recieved %d from index %d\n", RequestedValue, i)
257           csum += RequestedValue
258         }
```

```
259         }
260       }
261       fmt.Printf("Sum is %d\n", csum)
262       *webFormResponse = *webFormResponse + fmt.Sprintf("Sum is %d\n",
      csum)
263
264     default:
265       fmt.Printf("error: unknown command\n")
266       *webFormResponse = *webFormResponse + "error: unknown command"
267     }
268     WebFormRequest = WebFormRequestParsed{}
269 }
270
271 // interact - функция, содержащая цикл взаимодействия с сервером.
272 func interact(conn *net.TCPConn) {
273     defer conn.Close()
274     encoder, decoder := json.NewEncoder(conn), json.NewDecoder(conn)
275     for {
276       // Чтение команды из стандартного потока ввода
277       fmt.Printf("\ncommand = ")
278       command := input.Gets()
279
280       fmt.Println("Send command to WebForm: ", command)
281       go sendToWebForm("Client command: " + command, "SSE")
282
283       // Отправка запроса.
284       switch command {
285       case "quit":
286         send_request(encoder, "quit", nil)
287         return
288       case "help":
289         fmt.Println("You should use client after connection to server!")
290         fmt.Println("showValue - prints current index value")
291         fmt.Println("setValue - define a value of given index")
292         fmt.Println("getSum - count sum from n to m indexes")
293         continue
294       case "showValue":
295         fmt.Println(MyValue)
296         continue
297       case "setValue":
298         // by id
299         fmt.Printf("Index = ")
300         if valInd, err := strconv.Atoi(input.Gets()); err != nil {
301           fmt.Println("\nMust be number!")
302           continue
303         } else {
```

```
304         fmt.Printf("Value = ")
305         if valN, err := strconv.Atoi(input.Gets()); err != nil {
306           fmt.Println("\nMust be number!")
307           continue
308         } else {
309           send_request(encoder, "setValue", &jsonInt{N: valN, StartIndex
    : MyIndex, EndIndex: valInd})
310         }
311       }
312
313     case "getSum":
314       // from n to m
315       fmt.Printf("From index = ")
316       if valN, err := strconv.Atoi(input.Gets()); err != nil {
317         fmt.Println("\nMust be number!")
318         continue
319       } else {
320         fmt.Printf("To index = ")
321         if valM, err := strconv.Atoi(input.Gets()); err != nil {
322           fmt.Println("\nMust be number!")
323           continue
324         } else {
325           csum := 0
326           cstart := valN
327           if valN == MyIndex {
328             csum += MyValue
329             cstart = valN + 1
330           }
331           fmt.Println("Requests for sum...")
332           for i := cstart; i <= valM; i++ {
333             // Запрос значения
334             send_request(encoder, "getValue", &jsonSum{N: i, StartIndex:
    MyIndex})
335             // Получение ответа.
336             var resp Response
337             if err := decoder.Decode(&resp); err != nil {
338               fmt.Printf("error: %v\n", err)
339               break
340             } else {
341               if resp.Status != "ok" {
342                 fmt.Printf("error: data field is absent in response\n")
343               } else {
344                 fmt.Printf("Recieved %d from index %d\n", RequestedValue
    , i)
345                 csum += RequestedValue
346               }
```

```
347             }
348           }
349           fmt.Printf("Sum is %d\n", csum)
350           continue
351         }
352       }
353
354     default:
355       fmt.Printf("error: unknown command\n")
356       // Quit for Socket interaction
357       continue
358     }
359
360     // Получение ответа.
361     var resp Response
362     if err := decoder.Decode(&resp); err != nil {
363       fmt.Printf("error: %v\n", err)
364       break
365     }
366
367     // Вывод ответа в стандартный поток вывода.
368     switch resp.Status {
369     case "ok":
370       fmt.Printf("ok\n")
371     case "failed":
372       if resp.Data == nil {
373         fmt.Printf("error: data field is absent in response\n")
374       } else {
375         var errorMsg string
376         if err := json.Unmarshal(*resp.Data, &errorMsg); err != nil {
377           fmt.Printf("error: malformed data field in response\n")
378         } else {
379           fmt.Printf("failed: %s\n", errorMsg)
380         }
381       }
382     default:
383       fmt.Printf("error: server reports unknown status %q\n", resp.
    Status)
384     }
385
386   }
387 }
388
389 // send_request - вспомогательная функция для передачи запроса с указанн
    ой командой
390 // и данными. Данные могут быть пустыми (data == nil).
```

```
391 func send_request(encoder *json.Encoder, command string, data interface
        {}) {
392   var raw json.RawMessage
393   raw, _ = json.Marshal(data)
394   encoder.Encode(&Request{command, &raw})
395 }
396
397 func startClient(addrStr string) {
398   // Установка соединения с сервером и
399   // запуск цикла взаимодействия с сервером.
400   if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
401     log.Error("cannot resolve addres to connect", "adress", addrStr, "
      reason", err)
402   } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
403     log.Error("cannot establish connection to")
404     log.Info("waiting 5 seconds and trying again")
405     time.Sleep(5 * time.Second)
406     go startClient(addrStr)
407   } else {
408     log.Info("establish connection to", "address", conn.RemoteAddr().
      String())
409     Connection = conn
410     go interact(conn)
411   }
412 }
413
414
415 func startClient(addrStr string) {
416   // Установка соединения с сервером и
417   // запуск цикла взаимодействия с сервером.
418   if addr, err := net.ResolveTCPAddr("tcp", addrStr); err != nil {
419     log.Error("cannot resolve addres to connect", "adress", addrStr, "
      reason", err)
420   } else if conn, err := net.DialTCP("tcp", nil, addr); err != nil {
421     log.Error("cannot establish connection to")
422     log.Info("waiting 5 seconds and trying again")
423     time.Sleep(5 * time.Second)
424     go startClient(addrStr)
425   } else {
426     log.Info("establish connection to", "address", conn.RemoteAddr().
      String())
427     Connection = conn
428     go interact(conn)
429   }
430 }
```

Листинг 4: Файл httpServer.go

```
1  package main
2
3  import (
4      "encoding/json"
5      "fmt"
6      "net/http"
7      "log"
8  )
9
10 type MyHttpResponse struct {
11     Message string `json:"message"`
12 }
13
14 type MyHttpPayload struct {
15     Command string `json:"command"`
16     I1 string `json:"i1"`
17     I2  string `json:"i2"`
18 }
19
20 var dataHttpGetResponse string
21 var dataHttpPostResponse string
22
23 func enableCors(next http.Handler) http.Handler {
24     return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request)
       {
25         w.Header().Set("Access-Control-Allow-Origin", "*") // Allow all
       origins
26         w.Header().Set("Access-Control-Allow-Methods", "GET, POST,
       OPTIONS")
27         w.Header().Set("Access-Control-Allow-Headers", "Content-Type")
28
29         if r.Method == "OPTIONS" {
30             w.WriteHeader(http.StatusOK)
31             return
32         }
33
34         next.ServeHTTP(w, r)
35     })
36 }
37
38 func getHandler(w http.ResponseWriter, r *http.Request) {
39     w.Header().Set("Content-Type", "application/json")
40
41     // Parse query parameters
42     comm := r.URL.Query().Get("command")
```

```
43      i1 := r.URL.Query().Get("i1")
44      i2 := r.URL.Query().Get("i2")
45
46    fmt.Println("Get request command:", comm)
47
48      dataHttpGetResponse = "Get: "
49      parseFromForm(&dataHttpGetResponse, comm, i1, i2)
50
51      // if i1 == "" || i2 == "" {
52      //      http.Error(w, "Missing query parameters", http.
        StatusBadRequest)
53      //      return
54      // }
55
56      response := MyHttpResponse{Message: dataHttpGetResponse}
57      json.NewEncoder(w).Encode(response)
58
59  }
60
61  func postHandler(w http.ResponseWriter, r *http.Request) {
62      var requestData MyHttpPayload
63
64      // Parse JSON body
65      err := json.NewDecoder(r.Body).Decode(&requestData)
66      if err != nil {
67          http.Error(w, "Invalid request body", http.StatusBadRequest)
68          return
69      }
70
71      fmt.Println("Post request command:", requestData.Command)
72
73      dataHttpPostResponse = "Post: "
74      parseFromForm(&dataHttpPostResponse, requestData.Command,
        requestData.I1, requestData.I2)
75
76      response := MyHttpResponse{Message: dataHttpPostResponse}
77      w.Header().Set("Content-Type", "application/json")
78      json.NewEncoder(w).Encode(response)
79  }
80
81  func startHttpServer() {
82      http.Handle("/get", enableCors(http.HandlerFunc(getHandler)))
83      http.Handle("/post", enableCors(http.HandlerFunc(postHandler)))
84
85
86      fmt.Println("HTTP Server started at", MyHttpPort)
```

```
87      log.Fatal(http.ListenAndServe(MyIP + MyHttpPort, nil))
88  }
```

Листинг 5: Файл socketServer.go

```
1  package main
2
3  import (
4    "fmt"
5    "log"
6    "net/http"
7    "strings"
8
9    "github.com/gorilla/websocket"
10 )
11
12 var upgrader = websocket.Upgrader{
13   CheckOrigin: func(r *http.Request) bool {
14     return true
15   },
16 }
17 var dataSocketChannel = make(chan string)
18
19 func handleWebSocket(w http.ResponseWriter, r *http.Request) {
20   // Upgrade HTTP to WebSocket
21   conn, err := upgrader.Upgrade(w, r, nil)
22   if err != nil {
23     log.Println("Upgrade error:", err)
24     return
25   }
26   defer conn.Close()
27
28   // Waiting for socket reques
29   go func() {
30     for {
31       // Read a message from the client.
32       _, message, err := conn.ReadMessage()
33       if err != nil {
34         fmt.Println(err)
35         return
36       }
37       // Print the message to the console.
38       fmt.Println("Received command:", string(message))
39       SocketCommand = string(message)
40       c := strings.Split(SocketCommand, " ")
41
42       out := "Socket: "
```

```
43          parseFromForm(&out, c...)
44          conn.WriteMessage(websocket.TextMessage, []byte(out))
45        }
46      }()
47
48
49      // Waiting for dataSocketChannel to send to WebForm
50      for message := range dataSocketChannel {
51        // Send message to client
52        err := conn.WriteMessage(websocket.TextMessage, []byte(message))
53        if err != nil {
54          log.Println("Write error:", err)
55          return
56        }
57      }
58
59  }
60
61  func startSocketServer() {
62    http.HandleFunc("/", handleWebSocket)
63    fmt.Println("Socket server listening on", MySocketPort)
64    err := http.ListenAndServe(MyIP+MySocketPort, nil)
65    if err != nil {
66      log.Fatal("ListenAndServe:", err)
67    }
68  }
```

Листинг 6: Файл sseServer.go

```
1  package main
2
3  import (
4    "fmt"
5    "log"
6    "net/http"
7  )
8  var dataSseChannel = make(chan string)
9
10 // SSE handler function
11 func sseHandler(w http.ResponseWriter, r *http.Request) {
12     // Set CORS headers to allow cross-origin requests
13     w.Header().Set("Access-Control-Allow-Origin", "*") // Allow all
       origins
14     w.Header().Set("Access-Control-Allow-Methods", "GET") // Allow GET
       requests
15     w.Header().Set("Access-Control-Allow-Headers", "Content-Type") //
       Allow content-type headers
```

```
16
17      // Set headers for SSE
18      w.Header().Set("Content-Type", "text/event-stream")
19      w.Header().Set("Cache-Control", "no-cache")
20      w.Header().Set("Connection", "keep-alive")
21
22      // Create a channel for client disconnection
23      clientGone := r.Context().Done()
24
25      for {
26          select {
27              case message := <-dataSseChannel:
28                  _, err := fmt.Fprintf(w, "data: %s \n\n", MyIP + "$" +
    message)
29                  if err != nil {
30                      fmt.Println("Not sended")
31                      return
32                  }
33
34                  if f, ok := w.(http.Flusher); ok {
35                      f.Flush()
36                  } else {
37                      fmt.Println("Bad flush")
38                  }
39              case <-clientGone:
40                  fmt.Println("SSE client disconnected")
41                  return
42          }
43
44      }
45
46  }
47
48  func startSseServer() {
49      http.HandleFunc("/events", sseHandler)
50
51      fmt.Println("SSE server started at ", MySsePort)
52      log.Fatal(http.ListenAndServe(MyIP + MySsePort, nil))
53  }
```

Листинг 7: Файл start.sh

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

```
6  echo "Building..."
7  if [[ -e fedukov_lab3 ]]; then
8      rm fedukov_lab3
9  fi
10 go build -o fedukov_lab3
11 echo "Running..."
12 kill $(pgrep fedukov_lab3) > /dev/null 2>&1
13 ./fedukov_lab3
```

Листинг 8: Файл runNetwork.sh

```
1  #!/bin/bash
2  # Scipt that uploads files of lab and launches them on ssh hosts
3  # Requirements: "tmux" to control ssh, existing ssh-keys or "sshpass" to
       generate keys
4  # IHAVESSHKEY=false to generate keys
5
6  SESSION="MySSH"
7  tmux kill-session -t $SESSION
8  tmux new-session -d -s $SESSION
9
10 SCRIPT_DIR=$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev/null
       && pwd ) # Lab files dir. Default is script's dir
11 src=($SCRIPT_DIR/test.txt $SCRIPT_DIR/main.go) # Files to upload
12 path=/root/test/lab # Path to lab folder on server
13 startCommand="go run main.go" # Launch command
14 NetworkList=("yss1" "yss2" "yss3" "yss4") # List of ssh hosts (supposed
       to use ssh key)
15
16 # Options
17 clearMode=false # Remove lab dirs and exit
18 forceClear=false # Don't ask for confirmation before replacing lab files
19 justStart=false # Don't upload files, just execute $startCommand on
       server in tmux
20 IHAVESSHKEY=false # Set false and set IPS and PASSWORDS to generate ssh
       keys
21
22 # Read IPs and passwords into arrays
23 IPS=("185.104.251.226" "185.102.139.161" "185.102.139.168" "
       185.102.139.169")
24 PASSWORDS=("fMs0m69gIGQ3" "Up5b0A1wiLMQ" "gOsQ5p7FUJ9w" "w3Bt8hjge8oV")
25
26 function generateSSHKeys {
27
28     # SSH configuration file
29     SSH_CONFIG="$HOME/.ssh/config"
30     SSH_KEY="$HOME/.ssh/id_rsa_for_lab"
```

```
31
32      # Function to check if host is already in SSH config
33      function host_in_ssh_config() {
34          local host="$1"
35          if grep -q "Host $host" "$SSH_CONFIG"; then
36              return 0  # Found
37          else
38              return 1  # Not found
39          fi
40      }
41
42      # Check if SSH keys exist, generate if not
43      if [ ! -f "$SSH_KEY" ]; then
44          echo "Generating SSH key..."
45          ssh-keygen -t rsa -b 4096 -N "" -f "$SSH_KEY"
46      fi
47
48      # Make sure both lists have the same length
49      if [ "${#IPS[@]}" -ne "${#PASSWORDS[@]}" ]; then
50          echo "Error: IP list and password list must have the same number
        of entries."
51          exit 1
52      fi
53
54      useIpsAsHostNames=false
55      if [ "${#IPS[@]}" -ne "${#NetworkList[@]}" ]; then
56          echo "Not enough names in NetworkList. Using ip as a name"
57          useIpsAsHostNames=true
58      fi
59
60      # Backup existing SSH config file
61      if [ -f "$SSH_CONFIG" ]; then
62          local i=1
63
64          while [ -f "${SSH_CONFIG}.bak.$i" ]; do
65              i=$((i + 1))
66          done
67
68          cp "$SSH_CONFIG" "${SSH_CONFIG}.bak.$i"
69          echo "Backup of SSH config created at ${SSH_CONFIG}.bak.$i"
70      fi
71
72      # Create SSH config entries or skip if already exists
73      for i in "${!IPS[@]}"; do
74          IP="${IPS[$i]}"
75          PASSWORD="${PASSWORDS[$i]}"
```

```
76          if [ $useIpsAsHostNames = true ]; then
77              HOST=$IP
78          else
79              HOST="${NetworkList[$i]}"
80          fi
81
82          if host_in_ssh_config "$HOST"; then
83              echo "Host $HOST is already in SSH config, skipping..."
84              continue
85          fi
86
87          echo "Setting up passwordless SSH for $HOST ($IP)..."
88
89          # Copy SSH key to remote server using sshpass
90          sshpass -p "$PASSWORD" ssh-copy-id -i "$SSH_KEY.pub" "root@$IP"
91
92          # Add SSH config entry
93          echo "Host $HOST" >> "$SSH_CONFIG"
94          echo "  HostName $IP" >> "$SSH_CONFIG"
95          echo "  User root" >> "$SSH_CONFIG"
96          echo "  IdentityFile $SSH_KEY" >> "$SSH_CONFIG"
97          echo "  IdentitiesOnly yes" >> "$SSH_CONFIG"
98          echo "  StrictHostKeyChecking no" >> "$SSH_CONFIG"      # It
    causes "Warning: Permanently added"
99          echo "  UserKnownHostsFile /dev/null" >> "$SSH_CONFIG"  # Don't
    write to known hosts
100     done
101
102     echo "SSH configuration updated at $SSH_CONFIG"
103 }
104
105 if [ $IHAVESSHKEY = false ]; then
106     generateSSHKeys
107 fi
108
109 function addTmuxWindow {
110     i=$1 # Vps id
111     vps=$2 # Vps name
112     path=$3 # Lab dir path
113
114     # Launch program in tmux via ssh
115     tmux new-window -t $SESSION:$((i + 1)) -n ${NetworkList[i]} ssh $vps
      "cd $path; source ~/.profile; $startCommand"
116
117     echo "Tmux windows created"
118 }
```

```
119
120 function isDirEmpty {
121     vps=$1
122     path=$2
123     [ $(ssh $vps "ls -1 $path | wc -l") -eq "0" ]
124 }
125
126 function doesDirExist {
127     vps=$1
128     path=$2
129     ssh $vps "[ -e $path ]"
130 }
131
132 function removeDir {
133     vps=$1
134     path=$2
135
136     if [ $forceClear = false ]; then
137         echo "Clean folder?"
138         echo "ls -a $path"
139         echo "----"
140         ssh $vps "ls -a $path"
141         echo "----"
142         select yn in "Yes" "No"; do
143             case $yn in
144                 Yes) break;;
145                 No) echo "Skip"; return;;
146             esac
147         done
148
149     fi
150
151     ssh $vps "rm -rf $path/*"
152     echo "Cleaned folder"
153 }
154
155 # Check src files existence
156 for file in ${src[*]}
157 do
158     if [ ! -e $file ]; then
159         echo -e "\nFile not found: $file\n"
160     fi
161 done
162
163
164 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">
```

```
7   <title>My dashboard</title>
8   <style>
9     body {
10      margin: 0;
11      padding: 0;
12      background-color: #f0f0f0;
13      font-family: Arial, sans-serif;
14      display: flex;
15
16    }
17
18    .container, .cont3 {
19      text-align: center;
20      background-color: #f5f5f5;
21      align-items: center;
22      padding: 10px;
23      border: 1px solid #ccc;
24      margin: 2.5vh;
25      max-width: fit-content;
26      margin-inline: auto; /*Delete?*/
27      margin-left: 40%;
28      min-height: 90vh;
29
30    }
31
32    .container {
33      height: fit-content;
34    }
35
36    .cont3 {
37      max-height: 90vh;
38
39      position: relative;
40      margin-left: 10%;
41      width: 100%;
42
43      border: 1px solid #ccc;
44      background-color: #f9f9f9;
45      padding: 10px;
46      box-sizing: border-box;
47      min-width: 50vh;
48
49
50    }
51
52    .scrollable {
```

```
53        max-height: 75vh;
54        overflow-y: auto;
55        background: radial-gradient(white, #eee);
56      }
57
58      .text-box {
59        border: 1px solid #ccc;
60        background-color: #ffffff;
61        padding: 10px;
62        margin: 2vh 0;
63        padding-bottom: 20px;
64        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
65        border-radius: 8px;
66        overflow-y: auto;
67        display: grid;
68
69      }
70
71      .text-box:last-child,
72      .text-box:first-child {
73        margin: 0;
74      }
75
76      .text-box b {
77        font-size: 1.5rem;
78        color: #333;
79        margin: 0;
80      }
81
82      .text-box input,
83      select {
84        padding: 10px;
85
86        font-size: 1rem;
87        margin-top: 10px;
88        margin-bottom: 10px;
89        border: 1px solid #ddd;
90        border-radius: 4px;
91        width: 80%;
92        justify-self: center;
93
94      }
95
96      button {
97        padding: 10px 20px;
98        background-color: #444;
```

```
99        color: white;
100       border: none;
101       border-radius: 4px;
102       font-size: 1rem;
103       cursor: pointer;
104       transition: background-color 0.3s ease;
105       justify-self: center;
106     }
107
108     button:hover {
109       background-color: #000;
110     }
111
112     .box3 button {
113       margin-top: 10px;
114       margin-left: auto;
115       margin-right: auto;
116     }
117
118     .cont2 {
119       position: fixed;
120       text-align: left;
121       height: auto;
122       transform: translateY(-50%);
123       top: 50%;
124       left: 20px;
125       background-color: #f5f5f5;
126
127       display: flex;
128       justify-content: center;
129       align-items: center;
130       padding: 10px;
131       border: 1px solid #ccc;
132     }
133
134     .box3 {
135       padding: 20px;
136       font-size: 1.3rem;
137
138       color: #333;
139       margin: 0;
140       text-align: center;
141
142       display: flex;
143       flex-direction: column;
144       height: 150px;
```

```
145         box-sizing: border-box;
146         justify-content: center;
147 
148         flex: 1;
149 
150       }
151 
152       .box3 span {
153         border: 1px solid #ccc;
154         padding: 20px;
155         height: auto;
156         display: flex;
157         justify-content: center;
158         flex-grow: 1;
159       }
160 
161       .box3 strong {
162         padding-bottom: 10px;
163       }
164 
165       .box2 {
166         text-align: center;
167       }
168 
169       .box2 b {
170         font-size: 1.5rem;
171       }
172 
173       #messageContainer {
174       display: grid;
175       grid-template-columns: 1fr;
176       gap: 5px;
177       padding-left: 20px;
178       padding-right: 20px;
179       }
180 
181       .log_toremove {
182         border: 1px solid #ccc;
183         padding: 3px;
184         max-width: 400px;
185         background-color: #f9f9f9;
186       }
187       .log_toremove:first-child {
188         margin-top: 10px;
189       }
190       .log_toremove:last-child {
```

```
191         margin-bottom: 10px;
192       }
193
194     </style>
195     <script>
196
197       const ips = ["185.104.251.226", "185.102.139.161", "185.102.139.168"
          , "185.102.139.169"]
198       const wsPort = 1494
199       const httpPort = 1491
200       const ssePort = 1497
201       let msgCounters = [0, 0, 0, 0]
202       let transmition = ""
203       const psblTr = ["Get", "Post", "Socket"]
204
205       let sockets = []
206
207
208       function writeData(i, message) {
209         if (msgCounters[i] > 5) {
210           document.getElementById(i + " message").remove()
211           msgCounters[i]--;
212         }
213
214         let messageDiv = document.createElement("div");
215         messageDiv.id = i + " message"
216         messageDiv.className = "toremove"
217         messageDiv.textContent = message;
218
219         // Display the message in the #messages div
220         document.getElementById("id" + i).append(messageDiv);
221
222         msgCounters[i]++;
223       }
224
225       function startSocket(i) {
226
227         sockets[i] = new WebSocket("ws://" + ips[i] + ":" + wsPort)
228           const socket = sockets[i];
229
230           socket.onopen = function (e) {
231             console.log("Socket opened")
232           };
233
234           socket.onmessage = function (event) {
235             console.log(`Socket mesage to ${i}: ${event.data}`)
```

```
236           // Remove old messages
237           writeData(i, event.data)
238
239         };
240
241         socket.onclose = function (event) {
242           console.log("closed")
243           if (event.wasClean) {
244             //alert(`[close] Соединение закрыто чисто, код=${event.code}
      причина=${event.reason}`);
245             // socket.send("UPDATE");
246
247           } else {
248             console.log("by server")
249             // например, сервер убил процесс или сеть недоступна
250             // обычно в этом случае event.code 1006
251             //alert('[close] Соединение прервано');
252           }
253         };
254
255         socket.onerror = function (error) {
256           alert(`[error] ${error.message}`);
257         };
258       }
259
260     function updateQuery(ip_index) {
261       const params = getInput(ip_index).split(" ")
262       const urlParams = new URLSearchParams();
263       urlParams.append("vps_id", ip_index)
264       urlParams.append('command', params[0]);
265
266       switch (params[0]) {
267         case "getSum":
268           urlParams.append('i1', params[1]);
269           urlParams.append('i2', params[2]);
270           break;
271         case "setValue":
272           urlParams.append('i1', params[1]);
273           urlParams.append('i2', params[2]);
274           break;
275         default:
276           break;
277       }
278
279     // Update the URL with the new query parameters
```

```
280     const newUrl = `${window.location.pathname}?${urlParams.toString()
        }`;
281     window.history.pushState({}, '', newUrl);
282
283     console.log("Updated query")
284
285     }
286
287     async function sendGetRequest() {
288       // Get WebForm query params
289       const urlParams = new URLSearchParams(window.location.search);
290       let ip_index = parseInt(urlParams.get("vps_id"))
291
292       // Create a URL with query parameters
293       const url = new URL('http://' + ips[ip_index] + ":" + httpPort +
        '/get' + `?${urlParams.toString()}`);
294
295       try {
296         const response = await fetch(url);
297         const data = await response.json();
298         writeData(ip_index, data.message)
299       } catch (error) {
300         alert("Http Get error")
301       }
302     }
303     // Function to send a POST request
304     async function sendPostRequest(ip_index) {
305       try {
306         const params = getInput(ip_index).split(" ")
307         console.log(params)
308         const response = await fetch('http://' + ips[ip_index] + ":" +
        httpPort + '/post', {
309           method: 'POST',
310           headers: {
311             'Content-Type': 'application/json',
312           },
313           body: JSON.stringify({
314             "command": params[0],
315             "i1": params[1],
316             "i2": params[2],
317           })
318         });
319
320         const data = await response.json();
321         writeData(ip_index, data.message)
322
```

```
323       } catch (error) {
324         alert("Http Post error")
325
326       }
327     }
328
329     function clearVpsLog(index) {
330       let v = document.getElementById("inp" + index).value
331       if (v == "clear") {
332         els = document.getElementsByClassName("toremove")
333         Array.from(els).forEach((el) => {
334           el.remove()
335         });
336
337         msgCounters = [0, 0, 0, 0]
338         console.clear()
339         return true
340       }
341       return false
342     }
343
344     function getInput(index) {
345       let v = document.getElementById("inp" + index).value
346       if (v == "clear") {
347         els = document.getElementsByClassName("toremove")
348         Array.from(els).forEach((el) => {
349           el.remove()
350         });
351
352         msgCounters = [0, 0, 0, 0]
353         console.clear()
354         return ""
355       } else {
356         console.log("Command: \"" + v + "\" sended to vps " + index)
357         return v
358       }
359     }
360
361     function sendToSocket(index) {
362       sockets[index].send(getInput(index))
363     }
364
365     function send(ind) {
366       if (!clearVpsLog(ind)){
367         switch (transmition) {
368           case "Get":
```

```
369             updateQuery(ind)
370             break;
371           case "Post":
372             sendPostRequest(ind)
373             break;
374           case "Socket":
375             sendToSocket(ind)
376             break;
377           default:
378             console.error("Choose right transmition option!")
379             alert("No mode")
380             return
381         }
382         console.log("By", transmition)
383       }
384     }
385
386
387     function setTrMode() {
388       let mode = document.getElementById("trmode").value
389       updatePushButton()
390
391       if (psblTr.includes(mode)) {
392         transmition = mode
393         document.getElementById("currentMode").innerHTML = transmition
394         console.log("Switching to transmission mode: " + mode)
395
396       } else {
397         alert("Bad mode!")
398       }
399     }
400
401     function connect() {
402       let mode = document.getElementById("currentMode").innerHTML
403
404       // Secon connect by get is send
405       if (mode == "Get") {
406           sendGetRequest()
407           return
408       }
409
410       if (mode == "Socket") {
411         // let btn = document.getElementById("btn_connect")
412         // btn.disabled = true;
413         // btn.style.opacity = 0.5
414         return
```

```
415       }
416
417     }
418
419     function writeToLog(message, ip_index) {
420       const messageContainer = document.getElementById('messageContainer
      ');
421       if (ip_index != undefined){
422         ip_index += ") "
423       } else {
424         ip_index = ""
425       }
426
427       // Create a new div for each message
428       const newMessageDiv = document.createElement('div');
429       newMessageDiv.className = "log_toremove"
430       newMessageDiv.innerText = ip_index + message;
431
432       messageContainer.appendChild(newMessageDiv);
433     }
434
435     function clearLog() {
436       els = document.getElementsByClassName("log_toremove")
437         Array.from(els).forEach((el) => {
438           el.remove()
439           return ""
440         });
441       console.log("SSE log cleared")
442     }
443     // Open an SSE connection to the server
444     function startSSE(ip_index) {
445       const eventSource = new EventSource('http://' + ips[ip_index] + ":
      " + ssePort + '/events');
446
447       // Listen for messages from the server
448       eventSource.onmessage = function(event) {
449           const message = event.data;
450           let splitSymbolInd = message.indexOf("$")
451           let ip = message.slice(0, splitSymbolInd)
452           let odata = message.slice(splitSymbolInd + 1)
453           console.log(`SSE message from ${ip}: ${odata}`)
454           writeToLog(odata, ips.indexOf(ip))
455       };
456
457       // Handle SSE errors
458       eventSource.onerror = function(event) {
```

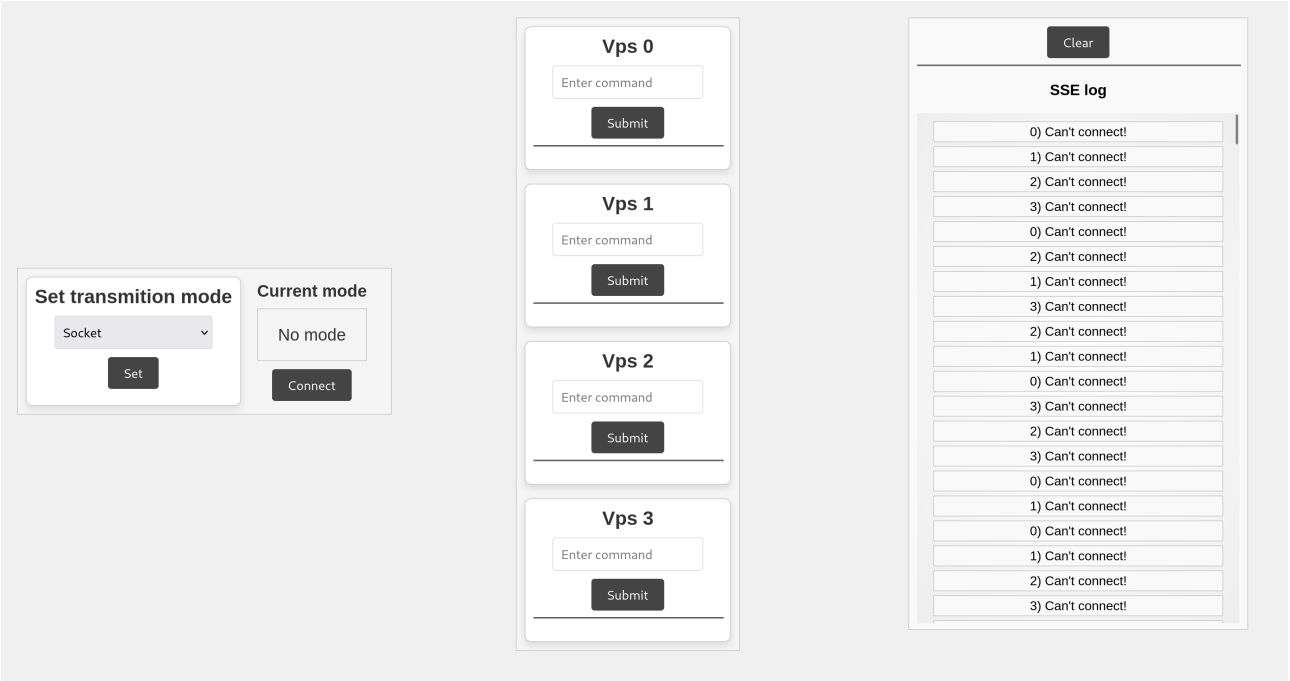
```
459             console.error(`SSE error on ${ips[ip_index]}`);
460             writeToLog("Can't connect!", ip_index)
461             setTimeout(startSSE(ip_index), 2000);
462         };
463
464         eventSource.onopen = function(event){
465           console.log("SSE connected")
466           writeToLog("SSE connected!", ip_index)
467         }
468
469     }
470
471
472     function updatePushButton() {
473       const selectedValue = document.getElementById("trmode").value;
474           let btn = document.getElementById("btn_connect")
475           btn.disabled = false;
476           btn.style.opacity = 1
477           switch (selectedValue) {
478             case "Get":
479               btn.innerHTML = "Send request"
480               break;
481             case "Post":
482                 btn.disabled = true;
483                 btn.innerHTML = "Automatically"
484                 btn.style.opacity = 0.5
485                 break
486             case "Socket":
487               btn.innerHTML = "Connect"
488               btn.disabled = true;
489               btn.style.opacity = 0.5
490             default:
491               break;
492           }
493
494     }
495
496     window.onload = () => {
497       for (let i = 0; i < ips.length; i++) {
498         startSSE(i)
499         startSocket(i)
500       }
501     }
502
503
504 //    // Test data for logs
```

```
505 //    window.addEventListener('load', function() {
506 //      for (let index = 0; index < 100; index++) {
507 //        writeToLog(index)
508 //        writeData(3, index)
509 //        writeData(0, index)
510 //      }
511 // });
512
513   </script>
514
515 </head>
516
517 <body>
518
519   <div class="cont2">
520     <div class="text-box box2">
521       <b>Set transmition mode</b>
522       <select name="transMode" id="trmode">
523         <option value="Socket">Socket</option>
524         <option value="Get">Get</option>
525         <option value="Post">Post</option>
526       </select>
527
528       <button onclick="setTrMode()">Set</button>
529     </div>
530     <div class="box3">
531       <strong>Current mode</strong>
532       <span id="currentMode">No mode</span>
533       <button onclick="connect()" id="btn_connect">Connect</button>
534     </div>
535   </div>
536
537   <div class="container">
538     <div id="id0" class="text-box">
539       <b> Vps 0 </b>
540       <input id="inp0" type="text" placeholder="Enter command">
541       <button onclick="send(0)">Submit</button>
542       <hr style="width: 100%;">
543     </div>
544     <div id="id1" class="text-box">
545       <b> Vps 1 </b>
546       <input id="inp1" type="text" placeholder="Enter command">
547       <button onclick="send(1)">Submit</button>
548       <hr style="width: 100%;">
549     </div>
550     <div id="id2" class="text-box">
```

```
551         <b> Vps 2 </b>
552         <input id="inp2" type="text" placeholder="Enter command">
553         <button onclick="send(2)">Submit</button>
554         <hr style="width: 100%;">
555       </div>
556       <div id="id3" class="text-box">
557         <b> Vps 3 </b>
558         <input id="inp3" type="text" placeholder="Enter command">
559         <button onclick="send(3)"> Submit</button>
560         <hr style="width: 100%;">
561       </div>
562     </div>
563     <div class="cont3">
564       <div>
565         <button type="reset" onclick="clearLog()">Clear</button>
566       </div>
567       <hr style="width: 100%;">
568       <div style="position: relative;">
569         <h3>SSE log</h3>
570         <div>
571           <div class="scrollable">
572               <div id="messageContainer" style="overflow-y: auto;"></div>
573           </div>
574         </div>
575       </div>
576     </div>
577 </body>
578
579 </html>
```

Вывод программы

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



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.