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 сети»

«Разработка кошелька»

Студент группы ИУ9-32Б Лавров Р. Д.

Преподаватель Посевин Д. П.

Москва 2025

1 Задание

Разработать распределённую сеть с возможность отправка "неназванных валют"между компьютерами в локальной сети

2 Результаты

Листинг 1: Кошелёк на golang (wallet.go)

```
package wallet

import "log"

type Wallet struct {
  ID            int
  UserName      string
  IPadressMask string
  ControllerIP  string
  IPadress      string
  Port          string
  Balance       int
  Block         bool
  wallets       map[int]string
  Balances      map[int]int
}

func NewWallet(id int, userName, IPadressMask, IPadress, port string, balance int) *Wallet {
  log.Println(" Initializing  wallet")
  return &Wallet{
    ID:            id,
    UserName:      userName,
    IPadressMask: IPadressMask,
    IPadress:      IPadress,
    ControllerIP: "",
    Port:          port,
    Balance:       balance,
    Block:         false,
    wallets:       make(map[int]string),
    Balances:      make(map[int]int),
  }
}

func (w *Wallet) GetWallets() map[int]string {
  return w.wallets
```

```
}

func (w *Wallet) AddWallet(id int, address string) {
  w.wallets[id] = address
  w.Balances[id] = 10
}

func (w *Wallet) UpdateBalance(id int, amount int) {
  w.Balances[id] = amount
}
```

Листинг 2: Кошелёк на golang (wsServer.go)

```
    package wsserver

import (
  "encoding/json"
  "fmt"
  "log"
  "net/http"
  "strconv"
  "strings"
  "sync"
  "time"
  "wallet-app/wallet"

  "github.com/gorilla/websocket"
)

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

type JSONbalance struct {
  Balance string `json:"balance"`
  Name    string `json:"name"`
}

var myWallet *wallet.Wallet

func StartServer(w *wallet.Wallet) {
  myWallet = w
  http.HandleFunc("/", handleClient)
  go func() {
    addr := myWallet.IPadress + ":" + myWallet.Port
    log.Println("Starting WebSocket server...")
```

```
34      if err := http.ListenAndServe(addr, nil); err != nil {
35        log.Fatalf("ListenAndServe error: %v", err)
36      }
37    }()
38    go ScanNetwork()
39  }
40
41  func ScanNetwork() {
42    log.Println("Start scanning network")
43    var wg sync.WaitGroup
44
45    for i := 1; i <= 15; i++ {
46      wg.Add(1)
47      go func(i int) {
48        defer wg.Done()
49        ip := fmt.Sprintf("%s%d", myWallet.IPadressMask, i)
50        connection := getConnectionWithWallet(ip)
51        if connection == nil {
52          return
53        }
54        defer connection.Close()
55        connection.WriteMessage(websocket.TextMessage, []byte("wallet"))
56        connection.SetReadDeadline(time.Now().Add(time.Second))
57        _, message, err := connection.ReadMessage()
58        if err != nil {
59          return
60        }
61        id, err := strconv.Atoi(string(message))
62        if err == nil {
63          if id == -2 {
64            myWallet.ControllerIP = ip
65            log.Println("Discovered Controller with id", id)
66          } else {
67            log.Println("Discovered wallet with id", id)
68            myWallet.AddWallet(id, ip)
69          }
70        }
71      }(i)
72    }
73    wg.Wait()
74    log.Println("Finished scanning network")
75    updateBalances()
76    updateController()
77  }
78
79  func updateBalances() {
```

```
80    for _, address := range myWallet.GetWallets() {
81      connection := getConnectionWithWallet(address)
82      defer connection.Close()
83      connection.WriteMessage(websocket.TextMessage, []byte("balance"))
84      _, message, err := connection.ReadMessage()
85      if err != nil {
86        fmt.Println(err)
87        continue
88      }
89
90      var balances map[string]JSONbalance
91      if err := json.Unmarshal(message, &balances); err != nil {
92        fmt.Println("Error unmarshalling balances:", err)
93        continue
94      }
95      for balanceID, balance := range balances {
96        intBalance, _ := strconv.Atoi(balance.Balance)
97        intBalanceID, _ := strconv.Atoi(balanceID)
98        myWallet.UpdateBalance(intBalanceID, intBalance)
99        log.Printf("Updated balance for wallet %d: %d\n", intBalanceID, intBalance)
100     }
101   }
102 }
103
104 func updateController() {
105   if myWallet.ControllerIP != "" {
106     conn := getConnectionWithWallet(myWallet.ControllerIP)
107     if conn == nil {
108       fmt.Println("Failed to connect to", myWallet.ControllerIP)
109       return
110     }
111     defer conn.Close()
112     if err := conn.WriteMessage(websocket.TextMessage, []byte("money_balance")); err != nil {
113       log.Println("Write error:", err)
114       return
115     }
116     conn.SetReadDeadline(time.Now().Add(3 * time.Second))
117     _, reply, _ := conn.ReadMessage()
118     if string(reply) == "ok" {
119       b := createJSONBalance()
120       if err := conn.WriteMessage(websocket.TextMessage, b); err != nil {
121         log.Println("Write error:", err)
122         return
123       }
124     }
125   }
```

```
126 }
127
128 func SendMoney(id int, amount int) {
129     _, ok := myWallet.GetWallets()[id]
130     if !ok {
131         fmt.Println("Wallet not found:", id)
132         return
133     }
134     for walletID, address := range myWallet.GetWallets() {
135         conn := getConnectionWithWallet(address)
136         if conn == nil {
137             fmt.Println("Failed to connect to", address)
138             return
139         }
140         defer conn.Close()
141         msg := fmt.Sprintf("send:%d:%d:%d", myWallet.ID, id, amount)
142         if err := conn.WriteMessage(websocket.TextMessage, []byte(msg)); err != nil {
143             log.Println("Write error:", err)
144             return
145         }
146         conn.SetReadDeadline(time.Now().Add(3 * time.Second))
147         _, reply, _ := conn.ReadMessage()
148         if string(reply) == "ok" && walletID == id {
149             log.Printf("Sent %d to wallet %d\n", amount, id)
150         }
151     }
152     updateController()
153 }
154
155 func handleClient(w http.ResponseWriter, r *http.Request) {
156     conn, _ := upgrader.Upgrade(w, r, nil)
157     defer conn.Close()
158     _, msg, err := conn.ReadMessage()
159     if err != nil {
160         return
161     }
162     message := string(msg)
163     ip := strings.Split(r.RemoteAddr, ":")
164     log.Printf("Get message from %s: %s\n", ip[0], message)
165     switch {
166     case message == "wallet":
167         if ip[0] == myWallet.ControllerIP {
168             updateController()
169         } else {
170             conn.WriteMessage(websocket.TextMessage, []byte(fmt.Sprint(myWallet.ID)))
171         }
```

```
172     case message == "block":
173         myWallet.Block = true
174     case message == "unblock":
175         myWallet.Block = false
176     case strings.HasPrefix(message, "send:"):
177         parts := strings.Split(message, ":")
178         if len(parts) == 4 {
179             from, _ := strconv.Atoi(parts[1])
180             to, _ := strconv.Atoi(parts[2])
181             amount, _ := strconv.Atoi(parts[3])
182             myWallet.Balances[from] -= amount
183             myWallet.Balances[to] += amount
184             conn.WriteMessage(websocket.TextMessage, []byte("ok"))
185         }
186     case message == "balance":
187         b := createJSONBalance()
188         conn.WriteMessage(websocket.TextMessage, b)
189     default:
190         conn.WriteMessage(websocket.TextMessage, []byte("Unknown command"))
191     }
192 }
193
194 func getConnectionWithWallet(address string) *websocket.Conn {
195     ip := fmt.Sprintf("ws://%s:3000/ws", address)
196     connection, _, err := websocket.DefaultDialer.Dial(ip, nil)
197     if err != nil {
198         return nil
199     }
200     return connection
201 }
202
203 func createJSONBalance() []byte {
204     var balances = make(map[int]JSONbalance)
205     for id, balance := range myWallet.Balances {
206         strBalance := fmt.Sprintf("%d", balance)
207         name := ""
208         if id == myWallet.ID {
209             name = myWallet.UserName
210         }
211         balances[id] = JSONbalance{Balance: strBalance, Name: name}
212     }
213     b, _ := json.Marshal(balances)
214     return b
215 }
```

Листинг 3: Кошелёк на golang (main.go)

```go
package main

import (
  "fmt"
  "wallet-app/wallet"
  wsserver "wallet-app/wsServer"
)

func main() {
  wallet := wallet.NewWallet(3, "Rodion", "172.20.10.", "172.20.10.2", "3000", 10)

  go wsserver.StartServer(wallet)
  for {
    fmt.Println("1. Show wallets")
    fmt.Println("2. Show balances")
    fmt.Println("3. Send money")
    fmt.Println("4. Exit")
    var choice int
    fmt.Scan(&choice)
    switch choice {
    case 1:
      fmt.Println("Wallets:")
      for id, address := range wallet.GetWallets() {
        fmt.Printf("ID: %d, Address: %s\n", id, address)
      }
    case 2:
      fmt.Println("Balances:")
      for id, balance := range wallet.Balances {
        fmt.Printf("ID: %d, Balance: %d\n", id, balance)
      }
    case 3:
      var id, amount int
      fmt.Print("Enter wallet ID to send money to: ")
      fmt.Scan(&id)
      fmt.Print("Enter amount to send: ")
      fmt.Scan(&amount)
      wsserver.SendMoney(id, amount)
    case 4:
      fmt.Println("Exiting ... ")
      return
    default:
      fmt.Println("Invalid choice, please try again.")
    }
  }
}
```

Листинг 4: Кошелёк на golang (wallet.py)

```
    import asyncio
import websockets
import json

class Wallet:
    def __init__(self):
        self.server_id = 6
        self.name = "Вова 2"
        self.INIT_NUMBER = 10
        self.amount = self.INIT_NUMBER
        self.block = False
        self.found_wallets = {}
        self.balances = {}
        self.queue = []
        self.IP = '172.20.10.2'
        self.IP2 = '172.20.10.'
        self.ip_controller = ''

    async def handle_client(self, websocket):
        try:
            async for message in websocket:
                if message == "wallet":
                    await websocket.send(str(self.server_id))
                    ip, p = websocket.remote_address
                    if ip != self.IP:
                        self.queue.append(ip)
                    if self.ip_controller == ip:
                        await self.update_controller()
                    print(f"Sent ID {self.server_id} to {websocket.remote_address}")
                elif message == "block":
                    self.block = True
                    print("block")
                elif message == "unblock":
                    self.block = False
                    print("unblock")
                elif message.startswith("send:"):
                    print(f"Received: {message} from {websocket.remote_address}")
                    _, from_id, to_id, amount = message.split(":")
                    from_id, to_id, amount = int(from_id), int(to_id), int(amount)

                    if to_id == self.server_id:
                        self.amount += amount
                        self.balances[to_id] += amount
                        self.balances[from_id] -= amount
                        await websocket.send("ok")
```

```
                        print(f"Got {amount} from {from_id}. New_balance: {self.amount}")
                    else:
                        if from_id in self.balances:
                            self.balances[from_id] -= amount
                        if to_id in self.balances:
                            self.balances[to_id] += amount
                        await websocket.send("ok")
                elif message == "balance":
                    await websocket.send(self.create_json_balance())
                else:
                    await websocket.send("Unknown command")
        except Exception as e:
            print(f"Client error: {e}")

    async def start_server(self, host=None, port=3000):
        if host is None:
            host = self.IP
        try:
            server = await websockets.serve(self.handle_client, host, port)
            print(f"Wallet server {self.server_id} started on ws://{host}:{port}")
            return server
        except Exception as e:
            print(f"Failed to start server: {e}")
            return None

    async def scan_network(self, ip_range=None):
        if ip_range is None:
            ip_range = self.IP2
        print(f"Scanning network {ip_range}*")

        tasks = []
        for i in range(1, 255):
            ip = ip_range + str(i)
            task = self.check_wallet(ip, 3000)
            tasks.append(task)

        results = await asyncio.gather(*tasks, return_exceptions=True)
        found_count = 0
        id = 0
        for i, result in enumerate(results):
            ip = ip_range + str(i + 1)
            if isinstance(result, int):
                if result > 0:
                    self.found_wallets[result] = ip
                    found_count += 1
                    id = result
```

```
                    print(f"+ Found wallet: ID {result} at {ip}")
                elif result == -2:
                    self.ip_controller = ip
                    print(f"+ Found controller on ip: ({ip})")
        if found_count == 1:
            self.balances[id] = self.INIT_NUMBER
        return found_count

    async def check_wallet(self, host, port, timeout=1):
        if host in ['0.0.0.0', 'localhost', '127.0.0.1']:
            return -1
        try:
            #print(f"Trying to connect to {host}...")
            async with websockets.connect(f"ws://{host}:{port}") as ws:
                await ws.send("wallet")
                response = await asyncio.wait_for(ws.recv(), timeout=timeout)
                #response = await ws.recv()
                if int(response) > 0 or int(response) == -2:
                    #print(f"{host} Success")
                    return int(response)
                return -1
        except Exception:
            return -1

    def create_json_balance(self):
        d = {}
        for wallet_id, _ in self.found_wallets.items():
            d[str(wallet_id)] = {
                "balance": str(self.balances[wallet_id]),
                "name": self.name if wallet_id == self.server_id else f"Wallet_{wallet_id}"
            }
        return json.dumps(d)


    async def update_balance(self, timeout=0.5):
        try:
            for wallet_id, ip in self.found_wallets.items():
                if wallet_id != self.server_id:
                    money_data = await self.get_balance(ip)
                    for id_str, balance_str in money_data.items():
                        id_int = int(id_str)
                        balance_int = int(balance_str["balance"])
                        self.balances[id_int] = balance_int
                        if self.server_id == id_int:
                            self.amount = balance_int
            if not self.server_id in self.balances:
```

```
                self.balances[self.server_id] = self.INIT_NUMBER
                self.amount = self.INIT_NUMBER
        except Exception as e:
            print(f"error balance {e}")

    async def get_balance(self, ip, timeout=0.5):
        try:
            async with websockets.connect(f"ws://{ip}:3000") as ws:
                await ws.send("balance")
                response = await asyncio.wait_for(ws.recv(), timeout=timeout)
                balance_data = json.loads(response)
                return balance_data
        except Exception as e:
            print(f"Error getting balance from {ip}: {e}")
            print(balance_data)
            return {}

    async def send_money(self, id, send_money, timeout=0.5):
        if id not in self.found_wallets:
            print("Not such wallet!")
            return False
        if self.block:
            print("Someone is sending money!")
            return False
        if self.amount < send_money:
            print("Don't have enough money!")
            return False
        if send_money <= 0:
            print("Wrong sum!")
            return False
        if id == self.server_id:
            print("This is you!!!")
            return False
        #ip = self.found_wallets[id]
        try:
            for wallet_id, ips in self.found_wallets.items():
                if wallet_id != self.server_id:
                    try:
                        async with websockets.connect(f"ws://{ips}:3000") as ws:
                            await ws.send("block")
                    except Exception:
                        pass

            #send to needed address
            for wallet_id, ips in self.found_wallets.items():
                try:
```

```
                async with websockets.connect(f"ws://{ips}:3000") as ws:
                    await ws.send(f"send:{self.server_id}:{id}:{send_money}")
                    response = await ws.recv()
                    if response == "ok" and wallet_id == id:
                        self.amount -= send_money
                        #self.balances[self.server_id] -= send_money
                        #self.balances[id] += send_money
                        print(f"Sent {send_money} to {id}. New_balance: {self.amount}")
            except Exception:
                pass

        if self.ip_controller != '':
            try:
                async with websockets.connect(f"ws://{self.ip_controller}:3000") as ws:
                    await ws.send(f"money_balance")
                    response = await ws.recv()
                    if response == "ok":
                        await ws.send(self.create_json_balance())
            except Exception:
                print(f"Error notifying controller {self.controller_id}: {e}")


        for wallet_id, ips in self.found_wallets.items():
            if wallet_id != self.server_id:
                try:
                    async with websockets.connect(f"ws://{ips}:3000") as ws:
                        await ws.send("unblock")
                except Exception:
                    pass
        return True
    except Exception as e:
        print(f"Exception in sending: {e}")
        return False

async def update_controller(self):
    try:
        async with websockets.connect(f"ws://{self.ip_controller}:3000") as ws:
            await ws.send(f"money_balance")
            response = await ws.recv()
            if response == "ok":
                await ws.send(self.create_json_balance())
    except Exception as e:
        print(f"Error controller-update: {e}")
 # Запускаем обработку очереди в фоне
async def interactive(self):
# Запускаем обработку очереди в фоне
```

```
230         async def process_queue():
231             while True:
232                 if self.queue:
233                     ip = self.queue.pop(0)
234                     if ip not in self.found_wallets.values() and self.ip_controller != ip:
235                         try:
236                             async with websockets.connect(f"ws://{ip}:3000") as ws:
237                                 await ws.send("wallet")
238                                 response = await ws.recv()
239                                 response = int(response)
240                                 if response > 0:
241                                     if response not in self.found_wallets:
242                                         self.balances[response] = self.INIT_NUMBER
243                                     self.found_wallets[response] = ip
244                                     print(f"+ Added wallet {response} from queue")
245                                 elif response == -2:
246                                     self.ip_controller, _ = ws.remote_address
247                                     print(f"+ Added controller! on ip {self.ip_controller}")
248 
249                             if self.ip_controller != '' and ip == self.ip_controller:
250                                 await self.update_controller()
251 
252                         except Exception:
253                             pass
254                 await asyncio.sleep(1)
255 
256         queue_task = asyncio.create_task(process_queue())
257 
258         try:
259             while True:
260                 print("\n"+40*"=")
261                 print("1. Send money")
262                 print("2. Balance")
263                 print("3. Update Balance")
264                 print("4. Exit")
265                 print(self.queue)
266                 try:
267                     choice = await asyncio.get_event_loop().run_in_executor(
268                         None, input, "Choose option: "
269                     )
270 
271                     if choice == "1":
272                         self.print_status()
273                         id = input("ID for sending: ")
274                         money = input("Money: ")
275                         await self.send_money(int(id), int(money))
```

```
                    elif choice == "2":
                        self.print_status()
                    elif choice == "3":
                        await self.update_balance()
                    else:
                        break

                except KeyboardInterrupt:
                    break

        finally:
            queue_task.cancel()

    def print_status(self):
        print(f"\nOur wallet: {self.server_id} (balance: {self.amount})")
        #print(self.found_wallets)
        #print(self.balances)
        if self.found_wallets:
            print("Wallets:")
            for wallet_id, _ in self.found_wallets.items():
                balance = self.balances[wallet_id]
                print(f"\t{wallet_id}: {balance} coins")

async def main():
    wallet = Wallet()

    server = await wallet.start_server()
    if not server:
        return

    await asyncio.sleep(2)
    found = await wallet.scan_network()

    # Выводим результаты
    print(f"\n" + "="*50)
    print(f"SCAN COMPLETED!")
    print(f"Found {found} wallet(s) in network")
    print(f"Our server ID: {wallet.server_id}")

    if found > 0:
        print("\nDiscovered wallets:")
        for wallet_id, ip in wallet.found_wallets.items():
            print(f"  ID: {wallet_id} -> IP: {ip}")
    else:
        print("\nNo other wallets found in the network")

```

```
322     await wallet.update_balance()
323     if wallet.ip_controller != '':
324         await wallet.update_controller()
325
326     await wallet.interactive()
327
328 if __name__ == "__main__":
329     asyncio.run(main())
```

Листинг 5: Кошелёк на golang (controller.py)

Листинг 6: Кошелёк на golang (page.html)

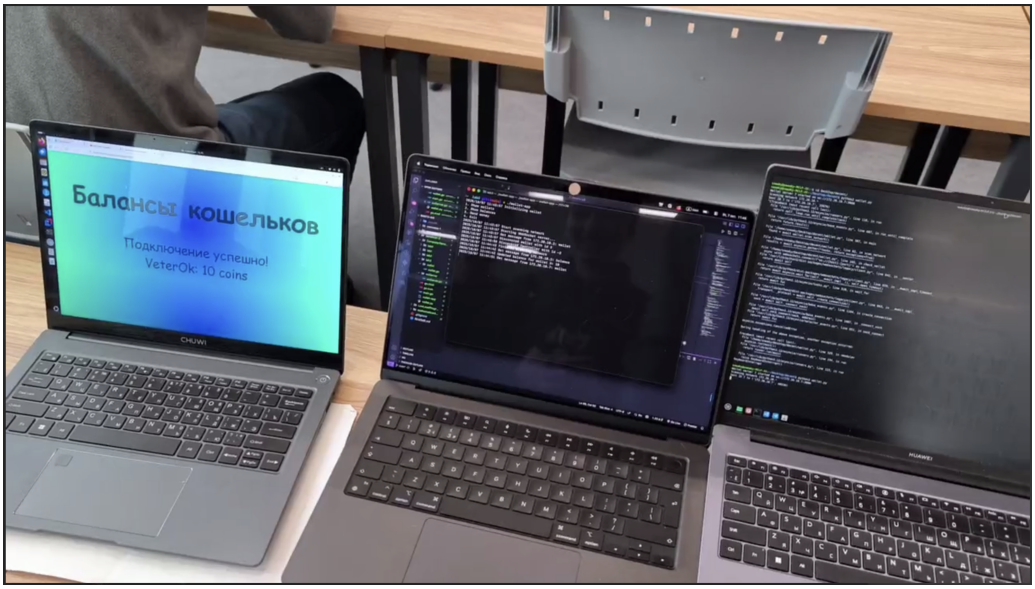

Рис. 1 — Результат работы

3 Вывод

Данная невероятно интересная лабораторная работа расширила мои знания компьютерных сетей и языка Golang