УДК 004.72

РАЗРАБОТКА ОБЛАЧНОЙ ПЛАТФОРМЫ И ВИЗУАЛЬНОЙ ПРОГРАММНОЙ СИСТЕМЫ КОНФИГУРИРОВАНИЯ УСТРОЙСТВ ИНТЕРНЕТА ВЕЩЕЙ

Д. А. Перепелкин, д.т.н., доцент, декан факультета вычислительной техники РГРТУ, Рязань, Россия; orcid.org/0000-0003-4775-5745, e-mail: perepelkin.d.a@rsreu.ru
Д. Д. Ткачев, студент факультета вычислительной техники РГРТУ, Рязань, Россия; orcid.org/0000-0002-1999-1356, e-mail: fenix784@mail.ru

В настоящее время широко востребованными становятся решения на базе Интернета вещей. Концепция Интернета вещей подразумевает построение программно-конфигурируемой сети (ПКС) физических устройств с интегрированными в них механизмами взаимодействия как между собой, так и с облачной платформой, программной системой и объектами внешнего мира. ***Целью работы*** *является разработка облачной платформы и визуальной программной системы конфигурирования устройств Интернета вещей. В работе предложена четырехуровневая архитектура программно-конфигурируемой сети устройств Интернета вещей. Для агрегирования структуры сети разработана облачная платформа, позволяющая с помощью интерфейса REST API и сокетов осуществлять взаимодействие с программной системой и конечными устройствами Интернета вещей. Для конфигурирования ПКС устройств Интернета вещей разработана визуальная программная система IoT Map. Детальное описание архитектуры и процесса функционирования программной системы выполнено с помощью UML-диаграммы классов. Особое внимание в работе уделено взаимодействию между визуальной программной системой, облачной платформой и конечными устройствами Интернета вещей.*

Ключевые слова: *Интернет вещей, программно-конфигурируемые сети, облачная платформа, сканер устройств сети, устройства сбора параметров Интернета вещей, визуальная программная система, сетевая архитектура, протокол OpenFlow, UML-диаграмма классов.*

DOI: 10.21667/1995-4565-2022-82-73-88

Введение

Интернет вещей (Internet of Things, IoT) – это концепция построения системы взаимосвязанных друг с другом и внешней средой физических устройств, позволяющая собирать, передавать и обрабатывать данные практически без необходимости участия человека. В настоящее время концепция Интернета вещей имеет достаточно широкий спектр применения, автоматизируя процессы из различных сфер деятельности человека. Программные системы Интернета вещей позволяют отслеживать и визуализировать данные с подключенных устройств и управлять ими.

Для развертывания систем Интернета вещей используют программно-конфигурируемые сети. В простейшем случае архитектура программно-конфигурируемой сети может быть представлена в виде множества программных систем и аппаратных коммутаторов, соединенных с общим контроллером OpenFlow. Данная архитектура строится на основе трех уровней: уровень сетевых приложений, уровень управления, инфраструктурный уровень.

Инфраструктурный уровень содержит коммутаторы, каналы передачи и системы обработки информации и выполняет функции агрегирования и преобразования данных. Уровень управления представляет собой программные средства, реализующие различные механизмы управления устройствами инфраструктурного уровня. Уровень сетевых приложений включает в себя набор приложений, позволяющих гибко и эффективно визуализировать, анализировать данные и управлять сетевой инфраструктурой.

В последние годы массово набирают популярность исследования и разработки с применением концепции Интернета вещей и технологии программно-конфигурируемых сетей. В данной работе предложена четырехуровневая архитектура ПКС устройств Интернета вещей, разработаны сканер устройств Wi-Fi сети и облачная платформа для сбора, обработки данных и управления устройствами Интернета вещей в режиме реального времени, а также описана визуальная программная система конфигурирования ПКС устройств Интернета вещей.

Теоретические сведения

В работе [1] представлена детальная информация об экосистеме Интернета вещей, рассмотрена архитектура и технологии построения ПКС устройств Интернета вещей. Работа [2] содержит подробную информацию об основных принципах построения, проектирования и поддержки ПКС. Различные подходы к технической реализации концепции Интернета вещей, протоколы и технологии передачи данных рассмотрены в [3]. Создание новых устройств в рамках концепции Интернета вещей, их подключение с помощью технологии Wi-Fi и обмен данными с облачной платформой рассматриваются в [4]. Основы облачных вычислений, существующие модели построения и развертывания облачных платформ, технологии виртуализации и информационной безопасности облачной инфраструктуры описаны в [5]. Оптимизация потоков данных, конфигурирование сети с помощью программных приложений описаны в работах [6, 7]. В работе [8] рассматривается концептуальный подход динамического формирования трафика программно-конфигурируемых телекоммуникационных сетей с балансировкой нагрузки. Визуальная среда и программная инфраструктура для распределенной обработки данных представлены в работе [9]. Подходы к динамическому управлению трафиком в облачной инфраструктуре программно-конфигурируемых сетей и центров обработки данных рассматриваются в работах [10-12].

Таким образом, актуальной научной задачей является построение ПКС устройств Интернета вещей с поддержкой облачных вычислений и возможностью конфигурирования.

Архитектура ПКС устройств Интернета вещей

При разработке систем на основе концепции Интернета вещей в простейшую архитектуру программно-конфигурируемой сети добавляют еще один уровень – уровень устройств Интернета вещей, получая тем самым четырехуровневую архитектуру ПКС (рисунок 1).

Уровень управления представлен облачной платформой, которая, помимо выполнения функций по обработке и хранению данных, управлению сетью и конечными устройствами, реализует программные методы интеграции с визуальной программной системой.

Уровень устройств Интернета вещей включает в себя датчики, выполняющие сбор, обработку и передачу по сети информации об окружающем мире, и исполнительные устройства, осуществляющие сбор информации, механическую работу и управление процессами.

В качестве устройства сбора параметров Интернета вещей используются микроконтроллер ESP32 и подключенные к нему датчики. Микроконтроллер ESP32 представляет собой систему на чипе с интегрированным модулем Wi-Fi и Bluetooth.

Наличие Wi-Fi модуля позволяет микроконтроллеру подключаться к Wi-Fi сети и отправлять информацию об окружающей среде в облако через сокет. Листинг подключения микроконтроллера ESP32 к сети WiFi представлен ниже.

```
#include <WiFi.h>


const char* ssid = "networkName";
const char* password = "networkPassword";

void setup() {
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(5000);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
  Serial.print("Hostname: ");
  Serial.println(WiFi.getHostname());
  Serial.print("ESP Mac Address: ");
  Serial.println(WiFi.macAddress());
  Serial.print("Gateway IP: ");
  Serial.println(WiFi.gatewayIP());
}
```

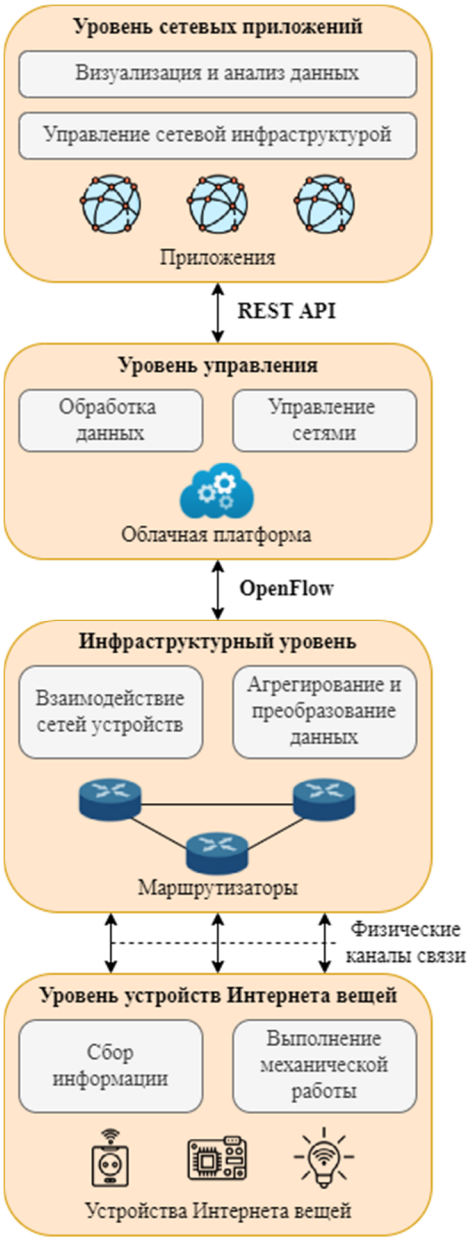

Рисунок 1 – Четырехуровневая архитектура ПКС устройств Интернета вещей
Figure 1 – Four-layer architecture of a SDN of Internet of Things devices

Программирование устройства сбора параметров Интернета вещей

В приведенном выше листинге в качестве глобальных переменных *ssid* и *password* объявлены учетные данные Wi-Fi сети, к которой требуется осуществить подключение. В методе *setup()* производится подключение к сети Wi-Fi с использованием ранее объявленных учетных данных. Поскольку Wi-Fi подключение является обязательным условием работы программы микроконтроллера, дальнейшее выполнение кода не будет продолжено, пока микроконтроллер не будет подключен к сети Wi-Fi. В случае успешного подключения к сети Wi-Fi программа микроконтроллера выводит соответствующую информацию, а также данные об IP и MAC-адресе микроконтроллера, его имени и шлюзе по умолчанию.

Для получения данных о температуре и влажности воздуха используется цифровой датчик DHT11. Листинг получения показаний температуры и влажности воздуха с датчика DHT11 приведен ниже.

```
#define DHTPIN 14
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
float h;
float t;

void setup() {
  Serial.println("DHT11 sensor!");
  dht.begin()
}

void loop() {
  h = dht.readHumidity();
  t = dht.readTemperature();
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.println(" *C ");
}
```

В представленном листинге переменная *DHTPIN* задает номер порта микроконтроллера, к которому подключен контакт *Data* датчика DHT11. Переменная *DHTTYPE* задает тип датчика влажности и температуры воздуха – DHT11. Далее создается объект *DHT*, методы которого позволяют получать данные с датчика. Для хранения значений влажности и температуры воздуха используются переменные с плавающей точкой *t* и *h*. В методе *setup()* осуществляется инициализация объекта *DHT* с помощью функции *begin()*. В основном цикле программы *loop()* происходит получение значений влажности и температуры воздуха из объекта *DHT*. Если значений параметров получить не удалось, то основной цикл программы начинается заново. После получения значений параметров программа осуществляет их печать.

Фоторезистор GL5537 позволяет получить информацию об уровне освещенности. Сопротивление фоторезистора GL5537 обратно пропорционально интенсивности света, поэтому, измерив его, можно получить значение уровня освещенности. Аналоговый вход микроконтроллера ESP32, к которому подключен фоторезистор GL5537, преобразует напряжение от 0 В до 3,3 В в целочисленное значение от 0 до 4095, называемое аналоговым значением. Аналоговое значение может быть считано с помощью функции *analogRead()*, а затем представлено в удобном для понимания человеком виде. Листинг получения значения уровня освещенности представлен ниже.

```
#define LIGHT_SENSOR_PIN 35

void loop() {
  float analogValue = analogRead(LIGHT_SENSOR_PIN);
  Serial.print("Analog Value = ");
  float avalogValuePercent = (float) analogValue / 4095.0 * 100.0;
  Serial.print(analogValue);
  if (analogValue < 40) {
    Serial.println(" => Dark");
  } else if (analogValue < 800) {
    Serial.println(" => Dim");
  } else if (analogValue < 2000) {
    Serial.println(" => Light");
  } else if (analogValue < 3200) {
    Serial.println(" => Bright");
  } else {
    Serial.println(" => Very bright");
  }
}
```

В приведенном листинге переменная *LIGHT_SENSOR_PIN* задает номер порта микроконтроллера, к которому подключен один из контактов фоторезистора GL5537. В основном цикле программы *loop()* осуществляются считывание аналогового значения порта, его дальнейшее преобразование в процентное соотношение и дискретный вид уровня освещенности, а также вывод преобразованных значений на печать.

Инфракрасный датчик движения HC-SR501 позволяет отслеживать движение. Принцип работы датчика HC-SR501 основан на изменении инфракрасного излучения на движущемся объекте. Пин *OUTPUT* датчика HC-SR501 выводит сигнал, соответствующий движению: LOW – движение не обнаружено, HIGH – движение обнаружено. Подключив вывод ESP32 к выводу *OUTPUT* датчика HC-SR501, можно считывать значение вывода *OUTPUT*, а затем определять движение. Листинг обнаружения движения приведен ниже.

```
#define timeSeconds 5

const int motionSensor = 27;
bool motionDetected = false;
unsigned long now = millis();
unsigned long lastTrigger = 0;
boolean startTimer = false;

void IRAM_ATTR detectsMovement() {
  Serial.println("Motion detected!");
  motionDetected = true;
  startTimer = true;
  lastTrigger = millis();
}

void setup() {
  pinMode(motionSensor, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);

void loop() {
  now = millis();
  if(startTimer && (now - lastTrigger > (timeSeconds*1000)) && (digitalRead(motionSensor) == LOW)) {
    Serial.println("Motion stopped...");
    startTimer = false;
    motionDetected = false;
  }
```

Для определения движения в программе используется прерывание, которое вызывает функцию *detectsMovement()* при обнаружении изменения на выводе, исключая при этом необходимость постоянной проверки текущего значения вывода. Переменная *motionSensor* задает номер порта микроконтроллера, к которому подключен вывод *OUTPUT* датчика движения HC-SR501. Функция *digitalPinToInterrupt()* позволяет установить вывод как вывод прерывания. Для установки прерывания используется функция *attachInterrupt()*, которая принимает в качестве аргументов вывод микроконтроллера, имя вызываемой функции и режим. В качестве режима используется режим RISING, который будет вызывать функцию каждый раз, когда значение вывода переходит с LOW на HIGH. Переменная *motionDetected* принимает значение true в случае обнаружения движения. Чтобы информация о движения была доступна в течение определенного количества секунд после его обнаружения, необходимо использовать таймер. В основном цикле *loop()* программа вычитает из текущего времени (переменная *now*) переменную *lastTrigger*, которая хранит время последнего обнаружения движения. Если разность времени превышает значение таймера, то переменная *motionDetected* устанавливается в false. Переменная *startTimer* отвечает за новый отсчет таймера.

После сбора всех параметров окружающей среды микроконтроллер осуществляет отправку данных в облако. Листинг отправки данных в облако через сокет представлен ниже.

```
#include <ArduinoJson.h>

WiFiClient client;
const uint16_t port = 7842;
const char * host = "hostIP";

void setup() {
  while (!client.connect(host, port)) {
   Serial.println("Connection to host failed");
   delay(5000);
  }
  Serial.println("Host connected");
  client.print("esp32");
}

void loop() {
  if (client.connected()) {
    DynamicJsonDocument doc(2048);
    doc["TYPE"] = "ESP32";
    doc["MAC"] = WiFi.macAddress();
    doc["HUMIDITY"][0] = h;
    doc["HUMIDITY"][1] = "%";
    doc["HUMIDITY"][2] = 0;
    doc["HUMIDITY"][3] = 100;
    doc["HUMIDITY"][4] = 0.1;
    doc["TEMPERATURE"][0] = t;
    doc["TEMPERATURE"][1] = "°C";
    doc["TEMPERATURE"][2] = 0;
    doc["TEMPERATURE"][3] = 50;
    doc["TEMPERATURE"][4] = 0.1;
    doc["IP"] = WiFi.localIP();
    doc["MOTION"][0] = motionDetected;
    doc["MOTION"][1] = "";
    doc["MOTION"][2] = 0;
    doc["MOTION"][3] = 1;
    doc["MOTION"][4] = 1;
    doc["LIGHT"][0] = avalogValuePercent;
    doc["LIGHT"][1] = "%";
    doc["LIGHT"][2] = 0;
    doc["LIGHT"][3] = 100;
    doc["LIGHT"][4] = 1;
```

```
      char json[400];
      serializeJson(doc, json);
      client.print(json);
    }
    else {
      client.stop();
      while (!client.connect(host, port)) {
        Serial.println("Connection to host failed");
        delay(5000);
      }
      Serial.println("Host reconnected");
      client.print("esp32");
    }
}
```

В представленном листинге в качестве глобальных переменных *port* и *host* объявлены IP-адрес и порт серверной части сокета, которая реализована в облаке. Сначала необходимо объявить объект класса **WiFiClient**, который будет использоваться для установления соединения и отправки данных. Чтобы установить фактическое соединение с облаком, необходимо вызвать метод *connect()* у объекта WiFiClient, передав в качестве первого параметра IP-адрес облака (переменная *host*), а в качестве второго – порт (переменная *port*). Данный метод возвращает значение true, если соединение было успешно установлено, и false в противном случае. Это позволяет также использовать данный метод и для проверки ошибок. В случае сбоя соединения с облаком программа циклично с некоторой задержкой предпринимает попытки установить соединение снова до тех пор, пока оно не будет установлено. В случае успешного соединения микроконтроллер отправляет данные о своем типе в облако для идентификации сервером, что осуществляется путем вызова метода *print()* у объекта класса **WiFiClient** и передачи в качестве входных данных строки для отправки («esp32»). В основном цикле программы *loop()* формируется json объект класса **DynamicJsonDocument**, содержащий информацию о типе микроконтроллера, его MAC-адресе, а также данные, полученные со всех датчиков. Для каждого параметра окружающей среды микроконтроллер также передает его единицу измерения, максимальное и минимальное значения, шаг изменения. Если в какой-то момент времени соединение с облаком будет разорвано, текущее соединение будет завершено путем вызова метода *stop()* у объекта класса **WiFiClient** для освобождения ресурсов. После чего микроконтроллер циклически с некоторой задержкой будет осуществлять попытки установления нового соединения с облаком.

Разработка сканера устройств Wi-Fi сети

Сканер устройств Wi-Fi сети (сетевой сканер) осуществляет сканирование сети, к которой подключено устройство, и определяет все клиенты, подключенные к этой сети. Идентификация каждого клиента в сети происходит с помощью его IP и MAC-адреса. Для определения активных устройств используются утилиты ping и arp, основанные на протоколах ICMP и ARP. Утилита ping осуществляет отправку пакета эхо-запроса протокола межсетевых управляющих сообщений ICMP на указанный клиент и ожидает эхо-ответа ICMP. Утилита arp позволяет определить MAC-адрес клиента по известному IP-адресу.

Методы *get_router_ip()* и *get_server_ip()* сканера сети используются соответственно для получения локальных IP-адресов маршрутизатора и устройства, на котором запущен сканер. Метод *get_global_ip()* возвращает внешний (глобальный) IP-адрес всего узла. На основе локального IP-адреса маршрутизатора определяется идентификатор подсети и формируется пул доступных IP-адресов устройств в сети.

Для выполнения утилиты ping используется метод *ping_device()*, который принимает целевой IP-адрес в качестве входного параметра и определяет доступность устройства по этому IP-адресу. Данный метод позволяет также определить задержку передачи данных до устройства и его тип. Метод *connect_pyp100()* определяет факт того, является ли сканируемое

устройство умной розеткой TP-Link Tapo P100, и в случае успешного соединения с ней, возвращает текущее состояние розетки. Метод *get_devices()* реализует выполнение утилиты arp и возвращает словарь соответствий IP и MAC-адресов устройств в сети. Для ускорения сканирования устройств в сканере сети реализована многопоточность.

Поскольку в операционных системах Windows и Linux формат записи утилит ping и arp отличаются, сканер сети при запуске определяет тип операционной системы и осуществляет выбор необходимого формата для дальнейшего выполнения утилит.

Кроме того, сканер реализует отправку информации об устройствах данной сети в облачную платформу, используя сокет – двунаправленное соединение, по которому осуществляется передача данных. Метод *send_messages()* отправляет данные в облако через сокет, а метод *receive_messages()* отвечает за прослушивание входящих сообщений. В случае недоступности облака сканер циклически осуществляет попытки установления соединения с ним в методе *do_connection_attempt()* до момента, пока соединение не будет установлено. Листинг обмена данными с облачной платформой представлен ниже.

```
def send_messages(sock: socket) -> None:
    try:
        while True:
            network_scanning()
            devices[global_ip] = devices.pop(router_ip)
            devices_json = json.dumps(devices)
            sock.send(devices_json.encode())
    except ConnectionResetError:
        print('Connection is broken')
        sock.close()
        run_main_thread()

def receive_messages(sock: socket) -> None:
    try:
        while True:
            content = sock.recv(2048)
            if len(content) != 0:
                data = json.loads(content.decode())
                if data['DEVICE_TYPE'] == 'tapo_p100':
                    manage_tapo_p100(data['COMMAND'], data['IP'])
    except ConnectionResetError:
        print('Connection is broken')
        sock.close()
        run_main_thread()

def do_connection_attempt() -> socket.socket | bool:
    try:
        print('Attempt to connect to the cloud')
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect((cloud_ip, cloud_port))
        sock.send(f'server {global_ip}'.encode())
        print('Connection has been successfully established')
        return sock
    except socket.error:
        print('Connection to cloud failed')
        return False
```

Вся информация о сети отправляется в облако в виде json файла, содержащего информацию об IP-адресах, MAC-адресах, задержках и типах устройств данной сети.

Для управления умной розеткой TP-Link Tapo P100 сетевой сканер содержит методы *turn_on()* и *turn_off()*, которые отвечают за включение и выключение розетки соответственно. IP-адрес управляемой розетки и тип команды содержатся в json объекте, принимаемом сканером сети от облачной платформы. Если целевая розетка в конкретный момент времени не-

доступна, то сканер циклически продолжит осуществлять попытки подключения к ней, пока не будет установлено соединение и выполнена команда.

Разработка облачной платформы Интернета вещей

Облачная платформа Интернета вещей агрегирует данные об общей структуре сети, хранит информацию о конечных устройствах Интернета вещей, осуществляет выработку команд в зависимости от конфигурации сети и их передачу на исполнительное устройство, реализует внешние API интерфейсы для интеграции с визуальной программной системой.

Для хранения информации обо всех узлах и конечных устройствах программно-конфигурируемой сети устройств Интернета вещей облачная платформа использует базу данных. Схема базы данных представлена на рисунке 2.

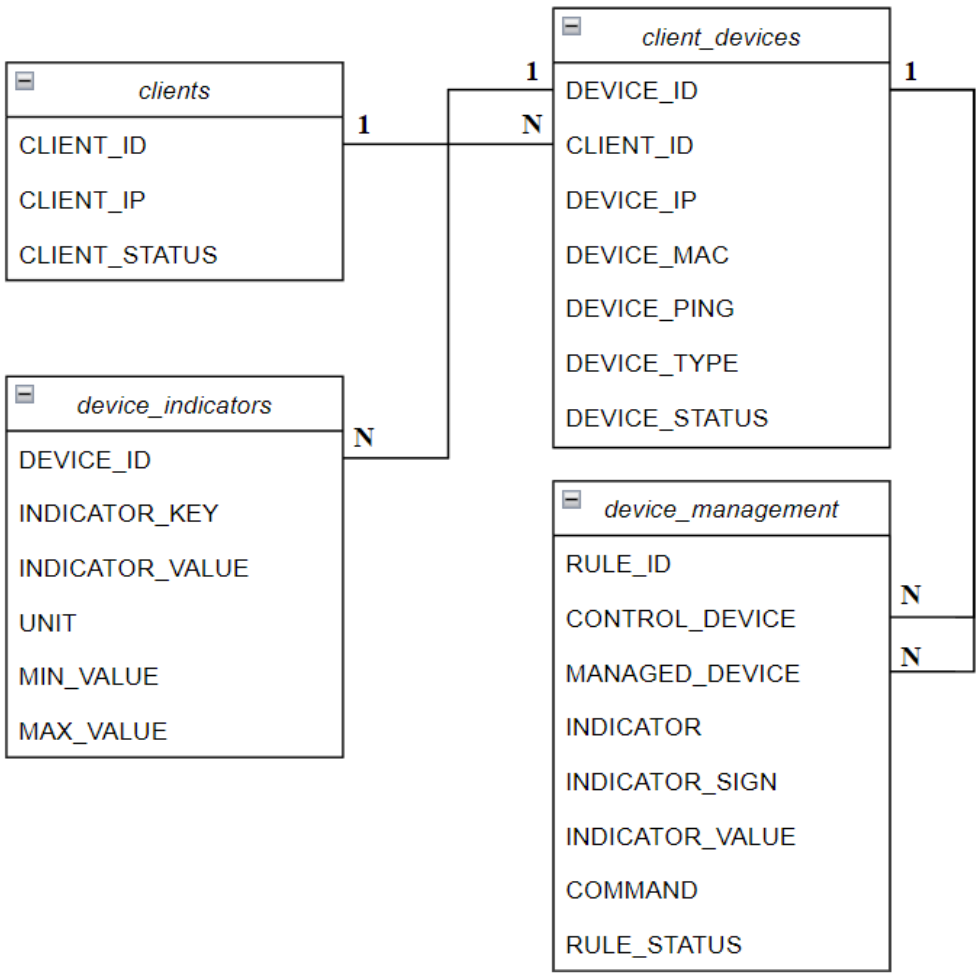

Рисунок 2 – Схема базы данных облачной платформы Интернета вещей
Figure 2 – Schema database of the Internet of Things cloud platform

В базе данных идентификация узлов программно-конфигурируемой сети осуществляется с помощью их глобального IP-адреса. Конечные устройства Интернета вещей идентифицируются с помощью MAC-адреса.

Для взаимодействия с базой данных в облачной платформе реализованы методы *select()* и *insert_or_update()*. Каждый из методов в качестве входных значений принимает параметризированную SQL-строку и кортеж параметров, необходимых для выполнения запроса. Метод *select()* используется для извлечения данных из базы данных и возвращает полученный после выполнения запроса кортеж значений. Метод *insert_or_update()* используется для выполнения операций вставки, обновления и удаления записей в базе данных. Для соединения с базой данных используется пул подключений, который осуществляет кэширование подключе-

ний в памяти для их повторного использования без необходимости создания нового соединения с базой данных с нуля. Такой подход обеспечивает значительное повышение производительности при работе с базой данных.

Для получения данных с устройств сбора параметров Интернета вещей и сетевых сканеров, а также отправки команд на конечные устройства в облаке реализован сокет-сервер. Для настройки сервера выполняется последовательность методов *socket()*, *bind()*, *listen()* и *accept()* объекта класса **socket**, встроенная в язык Python библиотеки *socket*. Вызов данных методов осуществляется в методе *launch_server()* облачной платформы. Листинг задания конфигурации сокет-сервера представлен ниже.

```
    def launch_server() -> None:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.bind(('0.0.0.0', port))
        sock.listen(10)
        while True:
            client, addr = sock.accept()
            client_info = client.recv(2048).decode().split()
            if client_info[0] == 'server':
                server_thread = threading.Thread(target=client_server,
args=[client, client_info[1]])
                server_thread.start()
            elif client_info[0] == 'esp32':
                esp32_thread = threading.Thread(target=client_esp32,
args=[client])
                esp32_thread.start()
```

После того, как входящее соединение принято сервером с помощью метода *accept()*, выполняется вызов метода, прослушивающего данные из соединения.

Для прослушивания данных от клиента, являющегося сканером устройств Wi-Fi сети, используется метод *client_server()*. Данный метод сначала сохраняет соединение со сканером сети в словаре *sockets* по ключу равному IP-адресу узла, а затем запускает цикл прослушивания данных от сканера сети, выполняющийся до тех пор, пока соединение не будет разорвано. Полученную от сетевого сканера через сокет строку данных метод *client_server()* дешифрирует и преобразует в формат json, после чего заносит в таблицу *client_devices* информацию об устройствах узла сети, содержащую данные о IP и MAC-адресе, типе, задержке и доступности устройств, а также обновляет значения параметров устройств Интернета вещей в таблице *device_indicators* базы данных облачной платформы Интернета вещей.

Метод *client_esp32()* вызывается, когда клиент является устройством сбора параметров Интернета вещей (микроконтроллером ESP32). Данный метод запускает цикл прослушивания данных от устройства сбора параметров, также выполняющийся до тех пор, пока соединение с микроконтроллером не будет разорвано. Из полученной через сокет строки после ее дешифрации и преобразования в формат json извлекаются и заносятся в базу данных новые значения параметров Интернета вещей.

Выполнение методов прослушивания данных от конечных устройств Интернета вещей осуществляется в отдельных потоках, что уменьшает время получения данных с различных устройств, ускоряет процесс обновления значения параметров в базе данных и механизм работы облачной платформы Интернета вещей в целом.

Для интеграции с визуальной программной системой и предоставления доступа к различным данным в облачной платформе реализован веб-сервер на основе архитектуры REST. Веб-сервер позволяет программной системе получать данные о структуре программно-конфигурируемой сети устройств Интернета вещей и задавать ее конфигурацию через интерфейс прикладного программирования (API). Облачная платформа отвечает на веб-запросы от программной системы с помощью представлений, которые имеют схожий с функциями синтаксис и сопоставляются с одним или несколькими URL запросами. Представление *send_cloud_mac()* веб-сервера возвращает MAC-адрес облачной платформы. Пред-

ставление *send_network_data()* используется для получения информации о структуре программно-конфигурируемой сети устройств Интернета вещей, ее узлах и конечных устройствах. Представление *send_esp32_data()* возвращает текущие значения параметров, полученные конкретным микроконтроллером ESP32. Для получения списка правил конфигурации сети используется представление *get_rules()*. Получение и дальнейшая отправка через сокет сканеру сети команд управления умной розеткой TP-Link Tapo P100, а также возврат статуса работы розетки осуществляются в представлении *manage_tapo_p100()*. Для получения от программной системы и сохранения в базе данных сформированных пользователем правил конфигурации сети используется представление *add_command()*.

Контроллер ПКС устройств Интернета вещей реализован в методе *controller()*. Контроллер получает из базы данных список всех правил конфигурации, сформированных пользователем, и проверяет их истинность. В случае, когда правило истинно, контроллер посылает команду, указанную в правиле, на соответствующий узел или конечное устройство.

Разработка визуальной программной системы конфигурирования устройств Интернета вещей

Разработанная визуальная программная система конфигурирования устройств Интернета вещей IoT Map предоставляет набор инструментов для управления сетевой инфраструктурой и устройствами Интернета вещей, а также визуализирует топологию сети и данные, получаемые с подключенных устройств.

Данная программная система представляет собой кроссплатформенное приложение, разработанное на языке программирования Python с использованием библиотеки для создания графического пользовательского интерфейса PyQt5. Программная система доступна для использования на операционных системах Windows и Mac OS.

Графический интерфейс разработанной программной системы представлен на рисунке 3.

Условно графический интерфейс можно разделить на следующие компоненты: графический редактор, окно информации об устройстве, окно функций устройства и программное меню. Графический редактор отображает топологию программно-конфигурируемой сети устройств Интернета вещей. Окно информации об устройстве, представленное классом **DeviceInfoFrame**, содержит сведения об IP-адресе, MAC-адресе, типе устройства, а также задержке передачи команд на устройство и получения данных с устройства в миллисекундах. Окно функций устройства, представленное классом **DeviceFunctionsFrame**, отображает список данных, получаемых с устройства, набор команд и функций, поддерживаемых данным устройством, а также позволяет провести конфигурирование сети.

На рисунке 4 представлена UML диаграмма классов визуальной программной системы IoT Map.

Класс **Device** является абстракцией сетевого устройства программно-конфигурируемой сети. Данный класс является базовым для всех классов устройств Интернета вещей. Все экземпляры данного класса являются элементами топологии сети, которые можно перемещать внутри графического редактора. Атрибуты *device_ip*, *device_mac*, *device_type*, *device_ping* хранят информацию об IP-адресе, MAC-адресе, типе устройства и значении задержки обмена пакетами данных с этим устройством соответственно. Атрибут *enabled* принимает значение «истина», если устройство доступно в настоящий момент, и «ложь» в противном случае. Каждый элемент имеет динамический размер, который определяется выбранным пользователем масштабом графического редактора и устанавливается с помощью метода *init_sizes*(). Метод *return_functions()* является абстрактным, и все подклассы класса **Device** предоставляют реализацию данного метода, содержащую элементы графического интерфейса для управления устройством и отображения данных с него.

Классы **Application, Cloud, Database, Server, Router, TapoP100, Unknown, ESP32** наследуются от класса **Device**.

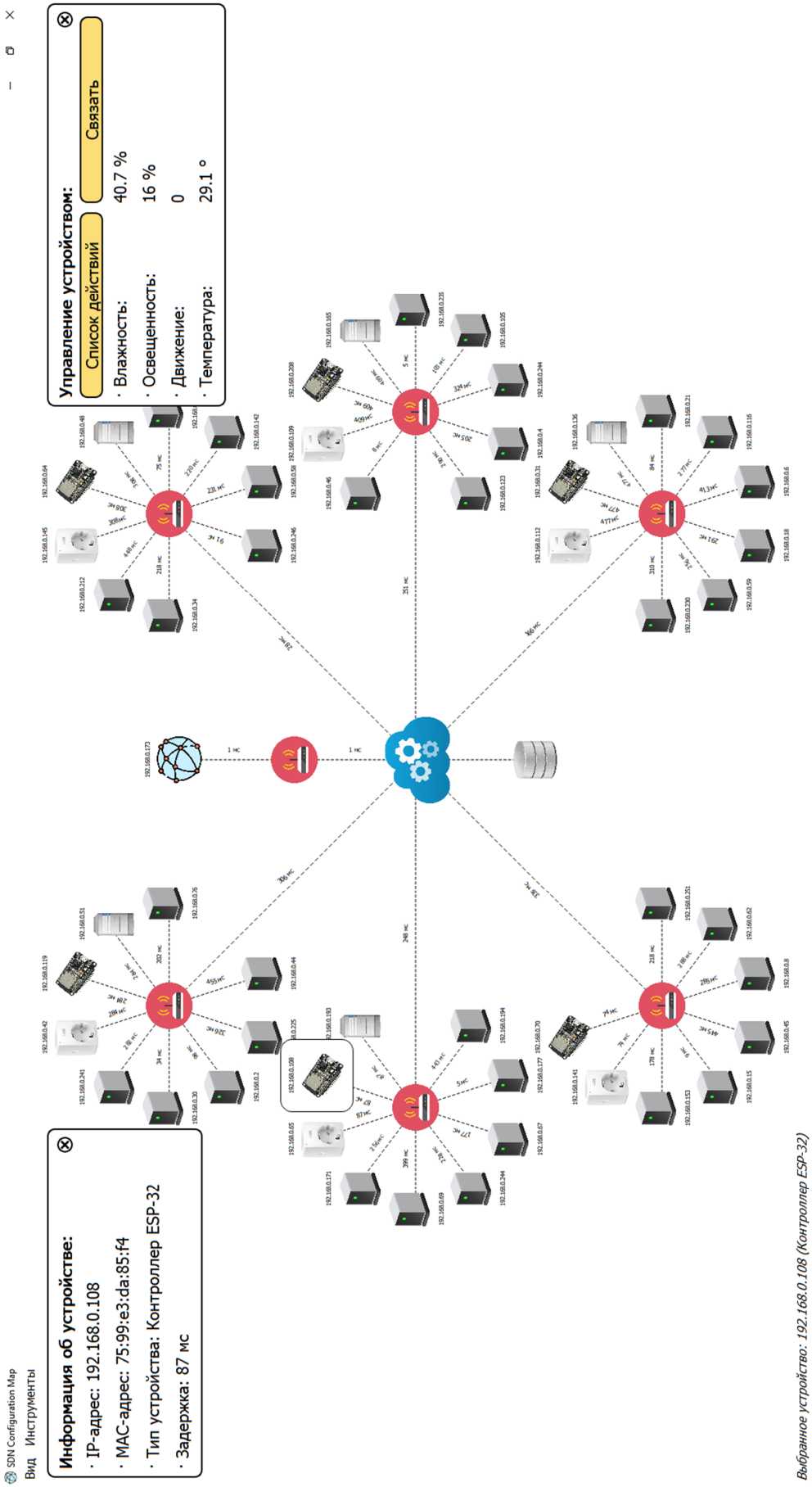

Рисунок 3 – Графический интерфейс программной системы IoT Map
Figure 3 – Graphical interface of the IoT Map software system

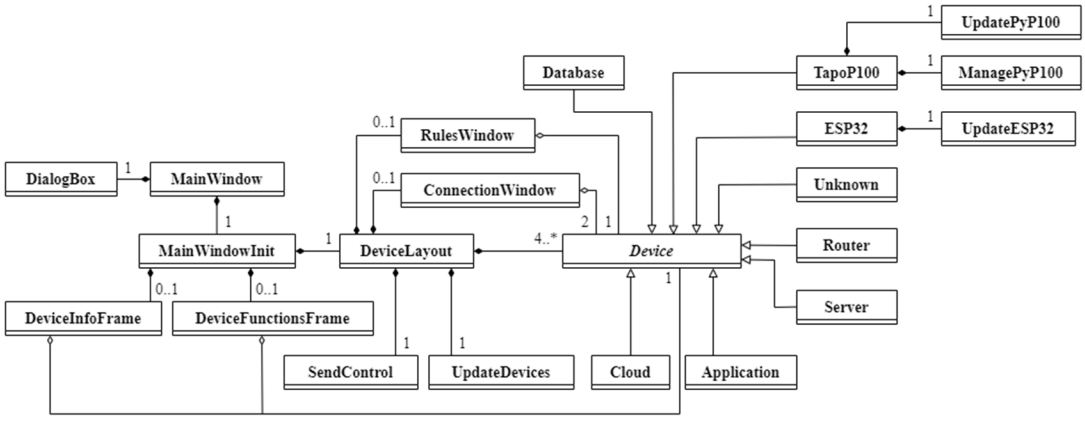

Рисунок 4 – Диаграмма классов визуальной программной системы IoT Map
Figure 4 – Class diagram of the visual software system IoT Map

Класс **ESP32** соответствует микроконтроллеру ESP32. Метод *add_indicator()* данного класса визуализирует в окне функций программной системы параметр Интернета вещей, получаемый с микроконтроллера. Экземпляр *thread_update* класса **UpdateESP32**, наследуемого от класса **QThread**, необходим для обращения к облачной платформе с целью получения новых значений параметров. Наследование класса **UpdateESP32** от класса **QThread** обеспечивает запуск параллельного цикла событий, благодаря чему графический интерфейс остается отзывчивым к взаимодействиям пользователя. Обращение к облачной платформе с целью получения данных в классе **UpdateESP32** осуществляется с помощью GET запроса. Полученные значения параметров передаются в метод *update_esp32_data()* класса **ESP32** с помощью механизма сигналов библиотеки PyQt5. Данный метод обновляет значения в окне функций программной системы на полученные и повторно вызывает метод *start()* экземпляра *thread_update* для получения новых значений.

Класс **TapoP100** соответствует умной розетке TP-Link Tapo P100. Для управления умной розеткой создается экземпляр *thread_manage* класса **ManagePyP100**, наследуемого от класса **QThread**. Обращение к облачной платформе для отправки команды управления умной розеткой в классе **ManagePyP100** осуществляется с помощью POST запроса. Атрибут *command* экземпляра *thread_manage* хранит тип отправляемой в облако команды управления. Экземпляр *thread_update* класса **UpdatePyP100**, наследуемого от класса **QThread**, необходим для обращения к облачной платформе с целью получения статуса работы розетки. Получение информации о статусе работы розетки в классе **UpdatePyP100** осуществляется с помощью GET запроса. Полученное значение статуса передается в метод *analyze_cloud_answer()* класса **TapoP100**, который обновляет доступность элементов управления в окне функций программной системы в зависимости от статуса работы умной розетки.

За создание экземпляров устройств и их размещение в графическом редакторе отвечает класс **DeviceLayout**. Атрибут *device_order* хранит в виде списка ссылки на экземпляры всех устройств в порядке их добавления в графический редактор. Атрибут *routers* содержит IP-адреса всех узлов программно-конфигурируемой сети устройств Интернета вещей. Атрибут *selected_device* хранит ссылку на текущее выбранное пользователем устройство. Атрибут *connected_device* хранит ссылку на устройство, которое является управляемым при конфигурировании сети. Атрибуты *connection_window* и *rules_window* содержат ссылки на окна конфигурирования и отображения конфигурации устройств соответственно. Метод *init_router()* инициализирует в программной системе новый узел сети, создавая экземпляры маршрутизатора и всех устройств данного узла. Метод *init_new_device()* отвечает за создание экземпляра устройства в зависимости от его типа. Методы *get_coordinates(), get_device_coordinates()*, *update_router_coordinates()* используются для формирования списка координат, по которым будут размещаться устройства внутри графического редактора. Для отрисовки линий связи между устройствами используется метод *draw_line()*. Для отображения значений задержки

передачи данных между устройствами на линиях связи используется метод *draw_text()*. Изменение масштаба графического редактора осуществляется в методе *wheelEvent()*, который срабатывает при прокрутке колесика мыши. Метод *create_window()* необходим для создания экземпляров класса **ConnectionWindow** и **RulesWindow**. Метод *select_device()* отображает доступные для конфигурирования подчиненные устройства. Метод *connect_devices()* вызывается при выборе пользователем в графическом редакторе подчиненного устройства и отображает в окне конфигурирования доступные функции управления данным устройством. Метод *save_connection()* собирает данные о сформированном пользователем правиле конфигурации сети и отправляет данную информацию в облако. Для отправки правила в облако используется экземпляр *send_control* класса **SendControl**, наследуемого от класса **QThread**. Обращение к облачной платформе для отправки правила конфигурации сети в классе **SendControl** осуществляется с помощью POST запроса.

Класс **ConnectionWindow** представляет собой окно конфигурирования устройств (рисунок 5), которое позволяет создавать пользователю новые правила конфигурации ПКС устройств Интернета вещей. Атрибут *device* хранит ссылку на текущее выбранное пользователем устройство. Атрибут *managed_device* хранит ссылку на управляемое устройство.

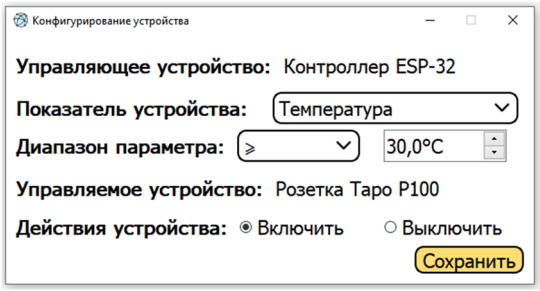

Рисунок 5 – Окно конфигурирования устройств
Figure 5 – Device configuration window

Класс **RulesWindow** представляет собой окно отображения конфигурации устройства (рисунок 6). Получение информации о конфигурации конкретного устройства осуществляется с помощью GET запроса к облачной платформе. Данная информация записывается в атрибут *rules* класса **RulesWindow** и в дальнейшем выводится в виде таблицы. Атрибут *device* также хранит ссылку на текущее выбранное пользователем устройство.

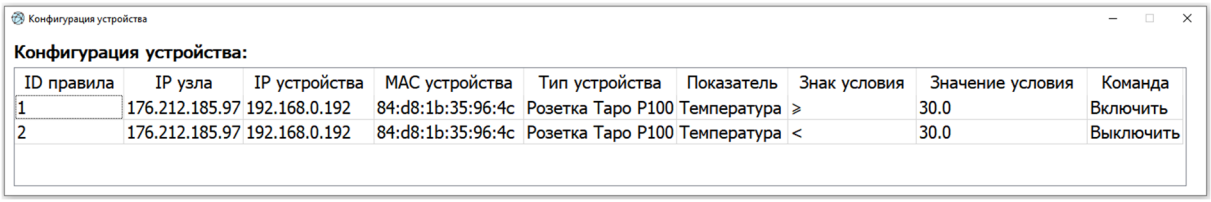

Рисунок 6 – Окно отображения конфигурации устройства
Figure 6 – Device configuration display window

В файле *adaptive_dimensions.py* хранятся размеры всех элементов программной системы, которые подстраиваются под размер экрана пользователя при запуске программы и динамически изменяются при масштабировании пользователем графического редактора с помощью метода *init_dimensions()*.

За обновление списков узлов и конечных устройств программно-конфигурируемой сети устройств Интернета вещей отвечает класс **UpdateDevices**, наследуемый от класса **QThread**. Получение информации о структуре сети, информации об устройствах осуществляется с помощью GET запроса к облачной платформе. Для добавления нового узла с помощью механизма сигналов вызывается метод *init_router()* класса **DeviceLayout**. Аналогичным образом вызываются методы *remove_routers()* для отображения информации о недоступности узла в данный момент и *add_device()* для добавления нового устройства.

Заключение

В статье разработана облачная платформа и визуальная программная система IoT Map для конфигурирования устройств Интернета вещей. Особое внимание в работе уделено организации сетевого взаимодействия между конечными устройствами Интернета вещей, визуальной программной системой и облачной платформой. Описание архитектуры программной системы IoT Map выполнено на основе UML-диаграммы классов.

Работа выполнена при финансовой поддержке гранта Президента РФ МД-3201.2022.1.6.

Библиографический список

1. **Ли П.** Архитектура Интернета вещей / пер. с англ. М. А. Райтмана. М.: ДМК Пресс, 2019. 454 с.

2. **Корячко В. П., Перепелкин Д. А.** Программно-конфигурируемые сети. Учебник для вузов. М.: Горячая линия-Телеком, 2020. 288 с.

3. **Росляков А. В.** Интернет вещей: учебное пособие [текст] / А. В. Росляков, С. В. Ваняшин, А. Ю. Гребешков. Самара: ПГУТИ, 2015. 200 с.

4. **Петин В. А.** Новые возможности Arduino, ESP, Raspberry Pi в проектах IoT. СПб.: БХВ-Петербург, 2022. 320 с. ил.

5. **Андреевский И. Л.** Технологии облачных вычислений: учебное пособие / И. Л. Андреевский. СПб.: Изд-во СПбГЭУ, 2018. 79 с.

6. **McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.** OpenFlow: Enabling Innovation in Campus Networks. Proc. ACM SIGCOMM Computer Communication Review. 2008, vol. 38, no. 2, pp. 69-74.

7. **Hilmi E. E.**, Adaptive Video Streaming over OpenFlow Networks with Quality of Service. Thesis for Degree of Master Science in Electrical and Electronics Engineering, Koc University, July 2012. 91 p.

8. **Перепелкин Д. А.** Концептуальный подход динамического формирования трафика программно-конфигурируемых телекоммуникационных сетей с балансировкой нагрузки // Информационные технологии. 2015. Т. 21. № 8. С. 602-610.

9. **Корячко В. П., Перепелкин Д. А., Иванчикова М. А., Бышов В. С., Цыганов И. Ю.** Программная инфраструктура и визуальная среда распределенной обработки потоков данных в программно-конфигурируемых сетях // Вестник Рязанского государственного радиотехнического университета. 2018. № 65. С. 44-54. DOI: 10.21667/1995-4565-2018-65-3-44-54.

10. **Никульчев Е. В., Паяин С. В., Плужник Е. В.** Динамическое управление трафиком программно-конфигурируемых сетей в облачной инфраструктуре // Вестник Рязанского государственного радиотехнического университета. 2013. № 3 (45). С. 54-57.

11. **Леохин Ю. Л., Фатхулин Т. Д.** Оценка возможности предоставления гарантированной скорости передачи данных в программно-конфигурируемой оптической сети // Вестник Рязанского государственного радиотехнического университета. 2020. № 71. С. 45-59. DOI: 10.21667/1995-4565-2020-71-45-59.

12. **Ушакова М. В., Ушаков Ю. А.** Исследование сети виртуальной инфраструктуры центра обработки данных с гибридной программно-конфигурируемой коммутацией // Вестник Рязанского государственного радиотехнического университета. 2021. № 75. С. 34-43. DOI: 10.21667/1995-4565-2021-75-34-43.

UDC 004.72

DEVELOPMENT OF CLOUD PLATFORM AND VISUAL SOFTWARE SYSTEM FOR CONFIGURING INTERNET OF THINGS DEVICES

D. A. Perepelkin, Dr. Sc. (Tech.), associate professor, Dean of Computer Engineering Faculty, RSREU, Ryazan, Russia;
orcid.org/0000-0003-4775-5745, e-mail: perepelkin.d.a@rsreu.ru
D. D. Tkachev, Computer Engineering Faculty, student,, RSREU, Russia;
orcid.org/0000-0002-1999-1356, e-mail: fenix784@mail.ru

Currently, solutions based on the Internet of Things are becoming widely in demand. The concept of the Internet of Things implies the construction of a software defined network (SDN) of physical devices with

inte-grated mechanisms of interaction both among themselves and with a cloud platform, a software system and objects of the outside world. ***The aim of the work*** *is to develop an architecture, a cloud platform and a visual software system for configuring Internet of Things devices. The paper proposes a four-level architecture of a software defined network of Internet of Things devices. To aggregate the network structure, a cloud platform has been developed that allows using the REST API interface and sockets to interact with the software system and end devices of the Internet of Things. A visual software system IoT Map has been developed to configure the IoT devices. A detailed description of the architecture and the process of software system functioning is performed using a UML class diagram. Special attention is paid to the interaction between visual software system, a cloud platform and the end devices of the Internet of Things.*

Key words: *Internet of Things, software defined networks, cloud platform, network device scanner, Internet of Things parameter collection devices, visual software system, network architecture, OpenFlow protocol, UML class diagrams.*

DOI: 10.21667/1995-4565-2022-82-73-88

References

1. **Li P.** *Arhitektura interneta veshchej.* Moscow: DMK Press, 2019. 454 p. (in Russian).

2. **Koryachko V. P., Perepelkin D. A.** *Programmno-konfiguriruemye seti.* Uchebnik dlya vuzov. Moscow: Goryachaya liniya-Telekom, 2020. 288 p. (in Russian).

3. **Roslyakov A. V.** *Internet veshchej: uchebnoe posobie* [tekst] / A. V. Roslyakov, S. V. Vanyashin, A. Yu. Grebeshkov. Samara: PGUTI, 2015. 200 p. (in Russian).

4. **Petin V. A.** *Novye vozmozhnosti Arduino, ESP, Raspberry Pi v proektah IoT*. Saint Petersburg: BHV-Peterburg, 2022. 320 p.: il (in Russian).

5. **Andreevskij I. L.** *Tekhnologii oblachnyh vychislenij: uchebnoe posobie* / I.L. Andreevskij. Saint Petersburg: Izd-vo SPbGEU, 2018. 79 p. (in Russian).

6. **McKeown N., Anderson T., Balakrishnan H., Parulkar G., Peterson L., Rexford J., Shenker S., Turner J.** OpenFlow: Enabling Innovation in Campus Networks. Proc. *ACM SIGCOMM Computer Communication Review*. 2008, vol. 38, no. 2, pp. 69-74.

7. **Hilmi E. E.**, Adaptive Video Streaming over OpenFlow Networks with Quality of Service. *Thesis for Degree of Master Science in Electrical and Electronics Engineering, Koc University*, July 2012. 91 p.

8. **Perepelkin D. A.** Konceptual'nyj podhod dinamicheskogo formirovaniya trafika programmno-konfiguriruemyh telekommunikacionnyh setej s balansirovkoj nagruzki. *Informacionnye tekhnologii*. 2015, vol. 21, no. 8, pp. 602-610 (in Russian).

9. **Koryachko V. P., Perepelkin D. A., Ivanchikova M. A., Byshov V. S., Cyganov I. Yu.** Programmnaya infrastruktura i vizual'naya sreda raspredelennoj obrabotki potokov dannyh v programmno-konfiguriruemyh setyah. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta*. 2018, no. 65, pp. 44-54. DOI: 10.21667/1995-4565-2018-65-3-44-54 (in Russian).

10. **Nikul'chev E. V., Payain S. V., Pluzhnik E. V.** Dinamicheskoe upravlenie trafikom programmno-konfiguriruemyh setej v oblachnoj infrastructure. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta*. 2013, no. 3 (45), pp. 54-57 (in Russian).

11. **Leohin Yu. L., Fathulin T. D.** Ocenka vozmozhnosti predostavleniya garantirovannoj skorosti peredachi dannyh v programmno-konfiguriruemoj opticheskoj seti. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta.* 2020, no. 71, pp. 45-59. DOI: 10.21667/1995-4565-2020-71-45-59 (in Russian).

12. **Ushakova M. V., Ushakov Yu. A.** Issledovanie seti virtual'noj infrastruktury centra obrabotki dannyh s gibridnoj programmno-konfiguriruemoj kommutaciej. *Vestnik Ryazanskogo gosudarstvennogo radiotekhnicheskogo universiteta.* 2021, no. 75, pp. 34-43. DOI: 10.21667/1995-4565-2021-75-34-43 (in Russian).