«Муниципальное автономное образовательное учреждение муниципального образования «Северодвинск» Лицей №17»

«Компьютерные сети»

Проект выполнен учеником 9А класса
МАОУ «Лицей №17» муниципального
образования «Северодвинск»
Поповым Романом

Руководитель проекта:
Учитель информатики
МАОУ «Лицей 17» муниципального
образования «Северодвинск»
Орлов Сергей Владимирович

Оглавление

ВВЕДЕНИЕ

Нас окружают компьютерные сети. Многие не замечают этого, но это так. Например, Интернет. Многие ли знают, как работает Интернет? Многие ли знают, как реально выглядит адрес сайта? Не многие.

Актуальность нашего проекта заключается в повсеместном распространении Интернета и других видов компьютерных сетей. 67,9% населения земли подключены к Интернету. Люди, хоть и пользуются им, но не знают, как это работает. Для многих после нажатия кнопки отправки сообщения в приложении мессенджере происходит магия и сообщение собеседнику перемещается при помощи волшебства.

Цель нашего проекта — рассказать людям о том, как и по каким правилам их данные перемещаются в сети интернет, что есть такое интернет и по каким принципам он работает.

Задачи проекта:

1) узнать, что такое интернет и его историю

2) узнать, как работает интернет

3) изучить модель OSI

4) рассмотреть стек TCP/IP и его устройство

5) разработать свой TCP/IP сервер, в ходе работы с которым мы подробнее разберём его принцип работы

ГЛАВА I. Основные понятия

1.1. Интернет вокруг нас. История Интернета

Точной даты появления Интернета нет. Причин на то несколько. Одной из главных причин послужило то, что первые сети использовались военными. Даже в советском союзе была своя военная сеть. Пытались разработать так же и гражданскую сеть под названием ОГАС, но в силу бюрократизма страны, проект не был реализован.

Как и многие гражданские инновации того времени, Интернет зародился в США. NSFNET – так называлась первая общественная сеть, прародитель современного Интернета, созданная Национальным научным фондом США (NSF) в 1984г по образцу закрытой ARPANET. Изначально к этой сети было подключены только университеты и вычислительные центры. Так как подключение к NSFNET было довольно свободным, к 1992г к ней подключились более 7500 мелких сетей, включая 2500 за пределами США. Знакомый нам глобальный Интернет появился в начале 90-х, когда NSFNET перешла в коммерческое использование и присоединила к себе BITNET и другие небольшие сети. Всех их объединяло нечто общее. Стек протокола TCP/IP. Но что это такое?

TCP/IP — это свод правил, по которым происходит обмен данными по всей сети Интернет. Для большего понимания этих правил стоит заметить, что есть два вида архитектуры сети: одноранговая (peer-to-peer) и клиент/сервер (client/server). Что же означают эти понятия?

Одноранговая сеть, или же peer-to-peer, подразумевает, что два компьютера в сети имеют один и тот же уровень доступа. Такие сети могут использоваться в офисах или учебных организациях. Например, в нашем Кванториуме все компьютеры связаны между собой. С одного компьютера можно просматривать содержание жёсткого диска другого, управлять этим содержимым, короче, иметь одни права с пользователем непосредственно того компьютера, в память которого вы залезли. Но что же тогда есть архитектура клиент/сервер?

1.2. Архитектуры и протоколы

Клиент/сервер (client/server) – это вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Взаимодействие между клиентом и сервером осуществляется по определённому протоколу, своду правил, который регулирует различные процессы, от создания пакета данных, до их маршрутизации.

Весь Интернет держится на своде правил под названием TCP/IP. Так же есть и другие протоколы, например, RIP (Routing Information Protocol) или более эффективный OSPF (Open Shortest Path First), FTP (File Transport Protocol), HTTP (Hyper Text Transport Protocol), HTTPS (HTTP Secure), ICMP (Internet Control Message Protocol), POP (Post Office Protocol) и SMTP (Simple Mail Transport Protocol). Далее, мы разберём основные протоколы.

1) Допустим, вы зашли полистать ленту на сайте ВКонтакте. Вы вводите в адресную строку адрес сайта и нажимаете Enter. Так как маршрутизация в сети Интернет использует протокол IP, буквенный вид не устроит маршрутизатор. Для преобразования того, что вы ввели в адресную строку в IP адрес, используется специальный сервер, DNS (Domain Name System). Что же такое IP адрес? IP адрес это уникальный идентификатор устройства в компьютерной сети, работающей по протоколу IP.

2) Но вернёмся к основной теме. Ваш запрос отправляется на сервер, где запущен сервер сайта. Стоит обратить внимание, что сервером называется как компьютер, обрабатывающий запросы, так и ПО, работающее с HTTP/HTTPS протоколом. Сервер обрабатывает ваш GET запрос и отправляет вам HTML страницу. Браузер её обрабатывает и отрисовывает.

3) Теперь, рассмотрим отличие протоколов HTTP и HTTPS. Протокол HTTP передаёт данные в исходном виде. То есть, любой, кто по ошибке или специально получит вашу страницу, без проблем сможет её расшифровать. Протокол HTTPS (S означает Secure) передаёт данные в зашифрованном виде, что не даёт так просто завладеть вашими данными.

4) Допустим, вы прочитали пост про интересные книги и захотели скачать эту книгу. Вы скачали её, прочитали и рассказали другу. Другу стало интересно и он попросил вас отправить ему файл. FTP протокол схож с HTTP своей сутью, но способен передавать большие объёмы файлов. Именно по протоколу FTP ваши компьютеры будут общаться.

5) Предположим, что вы решили написать своему другу электронное письмо. Для отправки электронных писем используется SMTP (Simple Mail Transport Protocol) протокол, а для получения — POP (Post Office Protocol).

6) Ранее, мы уже упоминали протокол IP. IP адрес – это уникальный идентификатор, который имеет каждое устройство, подключённое к сети, работающей с IP протоколом. Сам

протокол служит только маршрутизации. Взаимодействие же организует стек TCP. Если расшифровать эти аббревиатуры, мы получим Internet Protocol и Transmission Control Protocol соответственно. Чтобы лучше понять всю суть TCP/IP протокола, разберём, что такое ISO и OSI.

1.3. ISO и OSI

В начале 80-х, когда не было единого протокола для обмена данными, тогда международная организация по стандартизации (ISO – International Organization for Standardization) создала модель взаимодействия открытых систем (OSI – Open System Interconnection). Модель OSI так же имеет альтернативное название — семиуровневая модель взаимодействия открытых систем OSI. Эта модель подразумевает разделение архитектуры сети на 7 уровней: физический, канальный, сетевой, транспортный, сеансовый, представительный и прикладной. Эта модель может послужить наглядным примером принципа «разделяй и властвуй». Рассмотрим каждый из слоёв поподробнее.

Самый нижний уровень модели OSI – физический. Задача этого уровня передать вашу информацию по кабелю. Здесь определяются характеристики электрических сигналов: тип кодирования, скорость передачи сигналов. Так же к физическому уровню относят характеристики физических сред передачи данных: полоса пропускания, волновое сопротивление, помехозащищённость. Функции физического уровня реализуются сетевым адаптером или последовательным портом. Как пример протокола физического уровня, можно привести протокол 100Base-TX, или, его более известное название, Ethernet.

Чуть выше физического уровня находится канальный. Данный уровень отвечает за передачу данных между узлами в одной локальной сети. Узлом будет считаться любое устройство, подключённое к сети. Канальный уровень разделяет данные, которые вы хотите отправить, на фрагменты — кадры (frames). На этом уровне также будет происходить обмен кадрами между локальными устройствами и маршрутизация кадров при помощи MAC-адресов, которые производители вшивают в сетевые карты. Как пример, можно привести протоколы PPP и LAP-B.

Ещё выше находится сетевой уровень. Данный уровень служит для образования единой транспортной системы, которая объединяет несколько сетей. Важно понимать, что протоколы канального уровня обеспечивают транспортировку кадров только внутри одной локальной сети, а протоколы сетевого уровня обеспечивают межсетевое взаимодействие. На сетевом уровне работают несколько протоколов: маршрутизации и разрешения адреса. Протокол маршрутизации отвечает за выбор сети, в которую следует отправить кадр. Протокол разрешения адреса отвечает за преобразование IP-адресов в уникальные физические адреса хостов локальной сети. Примерами сетевых протоколов могут послужить IP и IPX протоколы.

Над сетевым уровнем находится транспортный уровень. Он отвечает за надёжность передачи пакетов. На транспортном уровне определены пять классов сервиса: срочность, восстановление прерванной связи, наличие средств мультиплексирования нескольких соединений, обнаружение ошибок и исправление ошибок. Обычно уровни модели OSI, начиная с транспортного уровня и выше, реализуются на программном уровне соответствующими компонентами операционных систем. Примерами транспортного уровня могут послужить TCP и UDP, SPX.

Представительный уровень изменяет форму передаваемой информации, но не изменяет её содержания. Например, это может быть изменение кодировки или шифровка и дешифровка информации. Пример протокола представительного уровня — SSL (Secure Socket Layer).

Прикладной уровень представляет собой набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к совместно используемым ресурсам. Единица данных называется сообщением. Примеры протоколов: HTTP, HTTPS, FTP, TFTP, SMTP, POP, SMB, NFS.

Для обозначения блоков данных, которыми обмениваются уровни используются следующие понятия: кадр (frame), пакет (packet), датаграмма (datagram), сегмент (segment). Все эти термины обозначают транспортируемые отдельно блоки данных и их можно считать синонимами.

1.4. архитектура стека TCP/IP

Так как же устроен стек TCP/IP? Протоколы семейства TCP/IP можно представить в виде модели, состоящей из четырёх уровней: прикладного, основного, межсетевого и сетевого. Каждый из этих уровней выполняет определённую задачу для организации надёжной и производительной работы сети. Рассмотрим эти уровни повнимательнее.

1.4.1. Уровень сетевого интерфейса

Данный уровень лежит в основании сей модели протоколов семейства TCP/IP. Уровень сетевого интерфейса отвечает за отправку в сеть и приём из сети кадров, которые содержат информацию.

1.4.2. Межсетевой уровень

Протоколы Интернет инкапсулируют блоки данных в пакеты (датаграммы) и обеспечивают необходимую маршрутизацию. К основным Интернет-протоколам относят: IP (отправка и маршрутизация пакетов), ARP (получение MAC-адресов), ICMP (отправление извещений и сообщений об ошибках при передаче пакетов), IGMP (используется маршрутизаторами при групповой передаче), RIP и OSPF(протоколы маршрутизации). На данном уровне реализуется передача пакетов без установки соединения — датаграммным способом. Основная задача межсетевого уровня — передача пакетов через составную сет, поэтому данный уровень также часто называют уровнем Интернет.

1.4.3. Транспортный (основной) уровень

Данный уровень обеспечивает сеансы связи между компьютерами. Существует два транспортных протокола: TCP (Transmission Control Protocol) и UDP (User Datagram Protocol). Протокол TCP ориентирован на договорную связь между компьютерами. Обычно по этому протоколу передают большие объёмы данных, для которых требуется подтверждение их приёма. Этот протокол использует большинство сетевых приложений, так как обеспечивает достаточную надёжность при передаче данных. Протокол UDP не ориентирован на соединение и не гарантирует доставку пакетов (датаграмм). Однако протокол UDP является более быстродействующим по сравнению с TCP. Обычно по этому протоколу передаются небольшие объёмы данных. Ответственность за доставку данных несёт сетевая программа.

1.4.4. Уровень приложения

Данный уровень является вершиной модели TCP/IP. На этом уровне работают практически все распространённые утилиты и службы: DNS, Telnet, WWW, Gopher, WAIS, SNMP, FTP, TFTP, SMTP, POP, IMAP. Если проводить аналогию между моделью OSI и TCP/IP, то получим, что 7 и 6 уровням OSI соответствует 1 уровень TCP/IP, 5 и 4 — 2, 3 — 3, 2 и 1 — 4. Мы подробно рассмотрели модель OSI и стек TCP/IP, время перейти к практической части. Сейчас мы создадим свой TCP/IP сервер и клиент на языке C++.

ГЛАВА II. Создание TCP/IP сервера

2.1. Подготовка к работе

В теоретической части мы рассмотрели, как устроено сетевое взаимодействие, самые распространённые архитектуры сети и основные протоколы. Так же мы более подробно разобрали стек протоколов TCP/IP.

Для практической части работы мы выбрали именно стек TCP/IP, так как он является самым распространённым. Так же, изначальной целью нашего проекта было понимание именно стека TCP/IP.

Мы напишем приложения сервера и клиента на языке программирования C++. Как говорилось в теоретической части, сервер — это провайдер услуг, а клиент — получатель этих услуг. Мы создадим простой сервер для обмена данными, который после можно масштабировать и интегрировать куда-либо.

Сейчас мы расскажем про принцип работы TCP/IP сервера. Сначала, мы создаём сокет, затем мы «биндим» сокет (связываем дальнейшие вызовы с сокетом), прослушиваем соединения и принимаем в случае наличия запроса и обслуживаем соединение.

Принцип работы клиента же немного отличается. Мы создаём сокет, затем мы запрашиваем у сервера подключение. После же просто пользуемся услугами сервера.

Если рассматривать процесс обслуживания и использования, то принцип таков: клиент отправляет на сервер какие-либо данные, сервер отправляет некий ответ, что бы отчитаться о получении пакета и цикл повторяется. При разрыве соединения, мы очищаем буферы как на стороне сервера, так и на стороне клиента во избежания дальнейших ошибок.

Рассмотрим заголовочные файлы: <iostream> - стандартная библиотека ввода-вывода, <stdlib.h> - из этой библиотеки нам понадобится системный вызов exit(), <sys/tepes.h> и <sys/socket.h> - это библиотеки для работы с сокетами, поэтому я не буду расписывать, что нам от них надо, <netinet/in.h> - в этой библиотеке находятся вспомогательные типы, <unistd.h> - ещё одна вспомогательная библиотека. Данные заголовочные файлы будут присутствовать как в приложении сервера, так и в приложении клиента.

2.2. Приложение сервера

Приступим к написанию кода приложения сервера. Сразу после подключения заголовочных файлов, объявим несколько констант (советую использовать constexpr вместо #define). Первая константа — это номер порта, с которого мы будем прослушивать соединения, вторая — максимальный размер буфера и третья — стоп-символ.

```
1   #include <iostream>
2   #include <netinet/in.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <sys/socket.h>
6   #include <sys/types.h>
7   #include <unistd.h>
8
9   constexpr int DEFAULT_PORT{8080};
10  constexpr int MAX_BUFFER_SIZE{4096};
11  constexpr char EXIT_SYMBOL{'#'};
```

Рис. 1.1. заголовки и константы приложения сервера

Далее, создадим сокет. Socket(AF_INET, SOCK_STREAM, 0). AF_INET – это семейство сокетов IPv4, SOCK_STREAM – это константа, которая обозначает сокет потока. Не будем углубляться в дальнейшие подробности, ведь основная суть практической части — понять принцип создания и работы сервера и клиента.

```
20      client = socket(AF_INET, SOCK_STREAM, 0);
21      if (client < 0) {
22        std::cout << "SERVER ERROR: establishing socket error." << std::endl;
23        exit(0);
24      }
```

Рис. 1.2. сокет приложения сервера

После создадим объект server_addres структуры sockaddr_in. Заполним в нём поля sin_port, sin_family, sin_addr.s_addr значениями hton(PORT), AF_INET и hton(INADDR_IN) соответственно.

```
27      server_addres.sin_port = htons(DEFAULT_PORT);
28      server_addres.sin_family = AF_INET;
29      server_addres.sin_addr.s_addr = htons(INADDR_ANY);
```

Рис. 1.3. заполнение полей структуры sockaddr_in

«Биндим» сокет методом bind(client, reinterpret_cast<struct sockaddr *>(&server_addres), sizeof(server_addres)).

```
31      int ret = bind(client, reinterpret_cast<struct sockaddr *>(&server_addres),
32                     sizeof(server_addres));
33
34      if (ret < 0) {
35        std::cout << "SERVER ERROR: binding connection. Socket has already been "
36                     "establishing."
37                  << std::endl;
38        return -1;
39      }
```

Рис. 1.4. привязка функций к сокету

Далее, ставим сокет на прослушивание. Если приходит запрос, принимаем соединение.

```
44      listen(client, 1);
45
46      server = accept(client, reinterpret_cast<struct sockaddr *>(&server_addres),
47                      &size);
48      if (server < 0) {
49        std::cout << "SERVER ERROR: can't accepting client." << std::endl;
50      }
```

Рис. 1.5. установка прослушивания запросов

Далее, опишем принцип работы основного цикла. Создадим буфер и специальный флаг. Пока соединение активно, выполняем цикл. Отправляем сообщение об успешном подключении к серверу. Затем, создаём ещё один цикл, условием выполнения которого является функция, которая будет описана ниже. В новом цикле мы получаем строку из стандартного потока ввода и отправляем её клиенту, затем ждём ответа клиента, выводим сообщение на экран и завершаем итерацию цикла.

```
52      char buffer[MAX_BUFFER_SIZE];
53      bool isExit = false;
54      while (server > 0) {
55        strcpy(buffer, "=> Server connected!\n");
56        send(server, buffer, MAX_BUFFER_SIZE, 0);
57        std::cout << "=> Connected to the client #1" << std::endl
58                  << "Enter " << EXIT_SYMBOL << " to end the connection\n\n";
59
60        std::cout << "Client: ";
61        recv(server, buffer, MAX_BUFFER_SIZE, 0);
62        std::cout << buffer << std::endl;
63        if (is_client_connection_closed(buffer)) {
64          isExit = true;
65        }
66
67        while (isExit) {
68          std::cout << "Server: ";
69          std::cin.getline(buffer, MAX_BUFFER_SIZE);
70          send(server, buffer, MAX_BUFFER_SIZE, 0);
71          if (is_client_connection_closed(buffer)) {
72            break;
73          }
74
75          std::cout << "Client: ";
76          recv(server, buffer, MAX_BUFFER_SIZE, 0);
77          std::cout << buffer << std::endl;
78          if (is_client_connection_closed(buffer)) {
79            break;
80          }
81        }
82        std::cout << "\nGoodbue...\n" << std::endl;
```

Рис. 1.6. основная функция севера

Функция is_client_connection_closed имеет следующий вид:

```
86    bool is_client_connection_closed(const char *msg) {
87      for (size_t i = 0; i < strlen(msg); i++) {
88        if (msg[i] == EXIT_SYMBOL) {
89          return true;
90        }
91      }
92      return false;
93    }
```

2.3. Приложение клиента

Начало у приложения клиента похоже на начало приложения сервера.

```
1   #include <iostream>
2   #include <netinet/in.h>
3   #include <stdlib.h>
4   #include <string.h>
5   #include <sys/socket.h>
6   #include <sys/types.h>
7   #include <unistd.h>
8
9   #define SERVER_IP "127.0.0.1"
10
11  constexpr int DEFAULT_PORT{8080};
12  constexpr char EXIT_CHAR{'#'};
13  constexpr int MAX_BUFFER_SIZE{4096};
```

Рис. 2.1. заголовки приложения клиента

Отличие начинается на этапе заполнения полей структуры sockaddr_in.

```
18      int client;
19
20      struct sockaddr_in server_addres;
21      client = socket(AF_INET, SOCK_STREAM, 0);
22      if (client < 0) {
23        std::cout << "CLIENT ERROR: establishing socket error.";
24        exit(0);
25      }
26
27      server_addres.sin_port = htons(DEFAULT_PORT);
28      server_addres.sin_family = AF_INET;
29      inet_pton(AF_INET, SERVER_IP, &server_addres.sin_addr);
```

Рис. 2.2. сокет и sockaddr_in

Отправляем запрос на подключение серверу:

```
33      if (connect(client, reinterpret_cast<const struct sockaddr *>(&server_addres),
34                  sizeof(server_addres)) == 0) {
35        std::cout << "=> Connection server " << inet_ntoa(server_addres.sin_addr)
36                  << "with port number " << DEFAULT_PORT << "\n";
37      }
```

Рис. 2.3. отправление запроса на подключение к серверу

Тело главного цикла приложения клиента очень схоже с телом цикла приложения сервера:

```
39      char buffer[MAX_BUFFER_SIZE];
40      std::cout << "Waiting for server confirmation...\n";
41      recv(client, buffer, MAX_BUFFER_SIZE, 0);
42      std::cout << "=> Connection established.\n"
43                << "Enter " << EXIT_CHAR << " for close the connection.\n";
44
45      while (true) {
46        std::cout << "Server: ";
47        std::cin.getline(buffer, MAX_BUFFER_SIZE);
48        send(client, buffer, MAX_BUFFER_SIZE, 0);
49        if (is_server_connection_closed(buffer)) {
50          break;
51        }
52
53        std::cout << "Server: ";
54        recv(client, buffer, MAX_BUFFER_SIZE, 0);
55        std::cout << buffer;
56        if (is_server_connection_closed(buffer)) {
57          break;
58        }
59        std::cout << std::endl;
60      }
61      close(client);
62      std::cout << "\nGoodbue...\n" << std::endl;
```

Рис. 2.4. основная функция приложения клиента

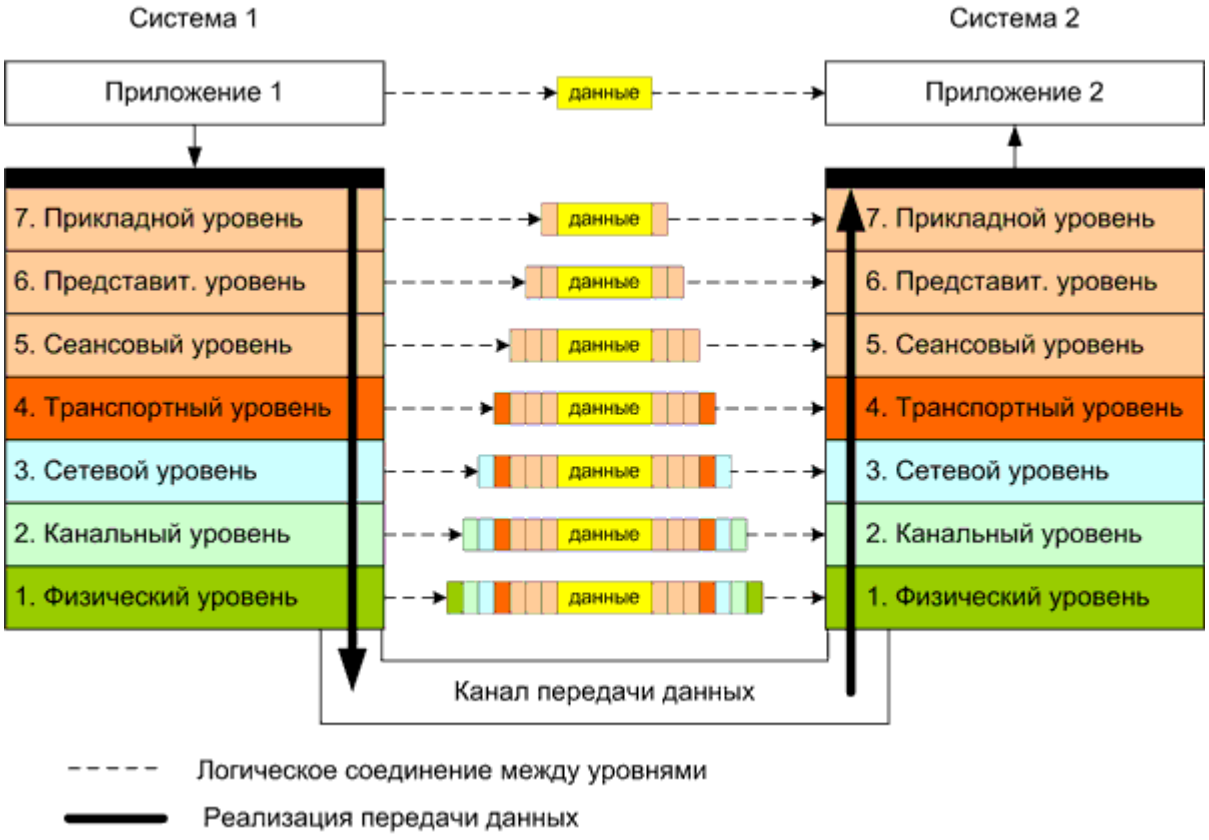
ЗАКЛЮЧЕНИЕ

Подводя итоги, мы разобрались в том, как работает интернет, узнали его историю, убедились в том, что даже при простом обмене данными существует сложная схема метаморфозы. Мы узнали, что такое архитектура OSI и как она повлияла на интернет в целом. Мы разобрали уровни архитектуры OSI и самый распространённый стек протоколов TCP/IP. Для закрепления пройденного материала мы создали свой собственный TCP/IP сервер, в ходе разработки которого мы узнали принцип его работы.

Справочные материалы

1. книга Ярошенко А. А. «Хакинг на C++», 2022
2. Ссылка на GitHub автора: https://github.com/DestructibleCoder/tcp_ip_for_raspberry/
3. https://ru.wikipedia.org/

Приложение



3.1. модель OSI

3.2. стек TCP/IP