МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
«ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ ИМЕНИ П.О. СУХОГО»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине «Распределённые информационные системы»

на тему: «Клиент-серверная система обработки запросов на основе протокола *TCP-IP* для обработки изображения c использованием высокочастотного

фильтра»

Исполнитель: студент группы ИТИ-41

Буякевич Д. О.

Руководитель: доцент

Комраков В. В.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

По защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2024

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc180343217)

[1 Аналитический обзор средств для создания клиент-серверной системы 6](#_Toc180343218)

[1.1 Клиент-серверная модель 6](#_Toc180343219)

[1.2 Использование протокола *TCP/IP* 8](#_Toc180343220)

[1.3 Использование протокола *HTTP* 11](#_Toc180343222)

[1.4 Использование протокола *UDP* 13](#_Toc180343223)

[1.5 равнение основных протоколов 15](#_Toc180343224)

[Список используемых источников 17](#_Toc180343225)

# **ВВЕДЕНИЕ**

В современном мире существует множество приложений, разработанных на разных языках программирования для различных платформ и аудитории. Быстрое развитие информационных технологий и их интеграция в повседневную жизнь создают спрос на эффективные и надежные распределенные системы. Эти системы снижают нагрузку на устройства пользователей, что позволяет большему числу людей использовать такие приложения.

Сегодня многие приложения для обработки изображений работают непосредственно на устройствах пользователей, так как это упрощает процесс разработки. Разработчики фокусируются на создании логики для локальной обработки данных, не тратя ресурсы на создание распределенных систем. Однако, распределенные системы способны предложить значительные преимущества за счет переноса вычислительных задач на сервер, что снижает требования к устройствам пользователей.

Создание распределенной системы для обработки изображений, где клиент отправляет изображение на сервер для обработки и получает результат обратно, является актуальной задачей. Такая система позволит пользователям с маломощными устройствами эффективно работать с приложением, так как все ресурсоемкие процессы будут выполняться на сервере. Это расширяет целевую аудиторию и повышает доступность приложения. Клиент-серверная архитектура обеспечит высокую производительность, минимизирует задержки и повысит устойчивость системы при высокой нагрузке.

Основной проблемой локальных приложений для обработки изображений является ограничение производительности устройств пользователей. Мобильные и менее мощные устройства могут сталкиваться с долгими задержками, медленной обработкой и высоким энергопотреблением при выполнении сложных вычислений. Кроме того, обновление таких приложений требует значительных усилий для поддержки различных платформ и устройств. Использование распределенной системы стримится решить эти проблемы, перенося вычислительные задачи на сервер, обеспечивая высокую производительность и стабильность при работе с любыми устройствами.

В ходе разработки необходимо выбрать технологии и подходы, которые определят архитектуру системы. Важно также выбрать протокол передачи данных и применить архитектурные шаблоны, которые позволят легко масштабировать и модифицировать приложение в будущем. Это позволит не только поддерживать текущую функциональность, но и с минимальными затратами добавлять новые возможности.

Итогом разработки станет распределенная система для обработки изображений, доступная широкому кругу пользователей, не требующая мощного оборудования. Гибкость архитектуры позволит адаптировать систему под новые требования и использовать различные протоколы передачи данных в будущем.

**1 АНАЛИТИЧЕСКИЙ ОБЗОР СРЕДСТВ ДЛЯ СОЗДАНИЯ   
 КЛИЕНТ-СЕРВЕРНОЙ СИСТЕМЫ**

* 1. **Клиент-серверная модель**

Для построения распределённых систем широко используется клиент-серверная модель. В этой модели все процессы можно разделить на две основные группы: серверы и клиенты. Серверы – это центральный компонент системы, который обрабатывает запросы клиентов и предоставляет им доступ к ресурсам (данным, вычислительным мощностям или другим сервисам). Сервер может работать в многозадачном режиме, обслуживая сразу нескольких клиентов. Клиент – это программа или устройство, которое посылает запросы серверу для получения определённой информации или использования его ресурсов. Клиент инициирует соединение и взаимодействие с сервером. Такое взаимодействие, когда клиент посылает запрос, а сервер отвечает, часто называют моделью «запрос-ответ», и оно демонстрируется на рисунке 1.1.

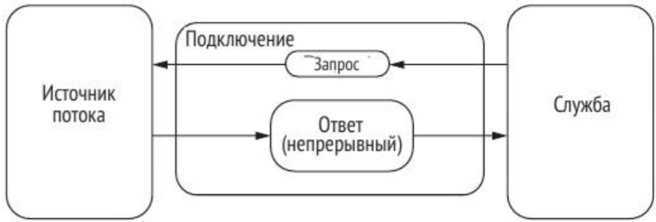


Рисунок 1.1 – Модель «Запрос-Ответ»

В основе работы клиент-серверной модели лежит обмен информацией по сети. Для этого используются различные сетевые протоколы, такие как TCP/IP, *HTTP* и другие. Наиболее распространённым является протокол с установкой соединения. В этом случае перед отправкой запроса клиент сначала устанавливает соединение с сервером, после чего отправляет запрос. Сервер отвечает через то же соединение, которое затем разрывается.

Использование такого протокола обеспечивает надёжность, поскольку каждое сообщение подтверждается сервером. Однако, в случае применения протоколов без установления соединения, таких как *UDP*, обмен данными происходит быстрее, но есть риск потери сообщений или их повреждения, что может негативно сказаться на работе системы.

Так, разработка таких приложений производится при помощи трех основных уровней клиент-серверной архитектуры. Первый уровень – это пользовательский интерфейс – этот уровень отвечает за взаимодействие с пользователем. Обычно он реализован на стороне клиента и включает в себя графические элементы, кнопки и формы, через которые пользователь взаимодействует с приложением. Например, в приложении для обработки изображений пользователь загружает картинку с помощью интерфейса на клиенте.

Второй, уровень обработки, здесь выполняется основная функциональность системы. Этот уровень обрабатывает запросы, полученные с клиентской стороны. В случае с приложением для обработки изображений, на уровне обработки происходит применение фильтров или других манипуляций с изображением.

И последний, уровень данных, предназначен для хранения и управления данными. Обычно, его принято реализовывать на сервере, где данные хранятся в реляционных базах данных или других системах управления данными. Сервер не только сохраняет информацию, но и отвечает за её целостность и безопасность.

Такая многослойная архитектура позволяет гибко распределять задачи между клиентом и сервером, в зависимости от потребностей системы. Например, в простых приложениях клиент может быть ответственен за выполнение большинства операций, а сервер будет обеспечивать только доступ к этим данным.

Также существуют несколько вариантов организации клиент-серверной архитектуры, каждый из которых может подойти для решения конкретных задач. В одной из возможных конфигураций клиент обрабатывает пользовательский интерфейс, а сервер выполняет все остальные задачи, включая обработку данных и их хранение. Такой подход называется тонким клиентом. В другом варианте, называемом толстым клиентом, клиентская часть берёт на себя больше работы, включая обработку и хранение некоторых данных, а сервер лишь обеспечивает доступ к определённым ресурсам. Принцип их работы представлен на рисунке 1.2.

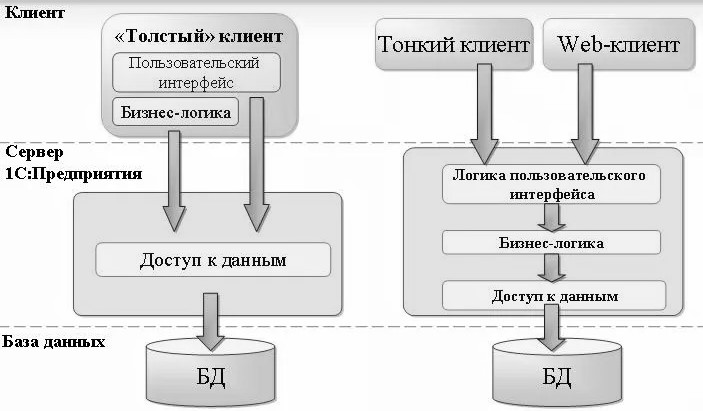


Рисунок 1.2 – Варианты архитектур клиент-серверной модели

В зависимости от задачи, разные уровни могут быть реализованы как на клиентской, так и на серверной стороне, или даже распределены между ними. Например, в веб-приложениях клиент может обрабатывать только графический интерфейс, а вся бизнес-логика и управление данными происходят на сервере. Такой подход позволяет оптимизировать распределение ресурсов и уменьшить нагрузку на клиента.

Основной задачей при проектировании клиент-серверной системы является выбор протокола передачи данных, который будет определять, как взаимодействуют клиент и сервер. Наиболее популярными протоколами для таких систем являются *TCP*, *UDP* и *HTTP*. Каждый из них обладает своими особенностями, которые делают его более или менее подходящим для конкретных задач, что будет детально рассмотрено в следующих разделах.

* 1. **Использование протокола *TCP/IP***

Протокол *TCP/IP* (*Transmission Control Protocol/Internet Protocol*) является основой большинства современных сетей, в том числе интернета. Он обеспечивает передачу данных и управление коммуникациями между устройствами в сети. *TCP/IP* представляет собой набор протоколов, который обеспечивает надёжную и последовательную передачу данных, а также их корректную обработку. Эта комбинация протоколов позволяет создавать надёжные системы передачи данных, что особенно важно для клиент-серверных приложений.

Протокол *TCP/IP* работает по принципу многоуровневой модели, которая состоит из четырёх основных уровней:

– прикладной уровень, отвечает за взаимодействие пользователя с программами. Здесь используются протоколы, такие как *HTTP*, *FTP*, *SMTP*, которые обеспечивают передачу данных между приложениями;

– транспортный уровень, он отвечает за надёжную передачу данных. Здесь работает протокол *TCP*, который управляет сегментацией данных, пересылкой и контрольными суммами для проверки целостности сообщений;

– сетевой уровень, отвечает за маршрутизацию данных между устройствами. Протокол *IP* определяет маршруты передачи пакетов и доставку их по сети;

– канальный уровень, управляет физическим подключением к сети и передачей данных по каналу связи. Этот уровень работает с конкретными физическими протоколами, например, *Ethernet* или *Wi-Fi*.

Все эти уровни имеют прямое отношение к модели взаимосвязи открытых систем (*OSI*), она служит основой для координации разработки стандартов, обеспечивая согласованность систем. В модели *OSI* коммуникация между системами разделена на семь уровней абстракции:

– прикладной,

– представительный,

– сеансовый,

– транспортный,

– сетевой,

– канальный,

– физический.

Модель описывает взаимодействие систем на каждом уровне, начиная с физической передачи битов по среде связи и заканчивая представлением данных на уровне распределённого приложения. Каждый уровень предоставляет функции для вышестоящего уровня и обслуживается нижестоящим. При разработке программного обеспечения используются протоколы связи для реализации классов функциональности на каждом уровне. На рисунке 1.3 представлено устройство модели уровней.

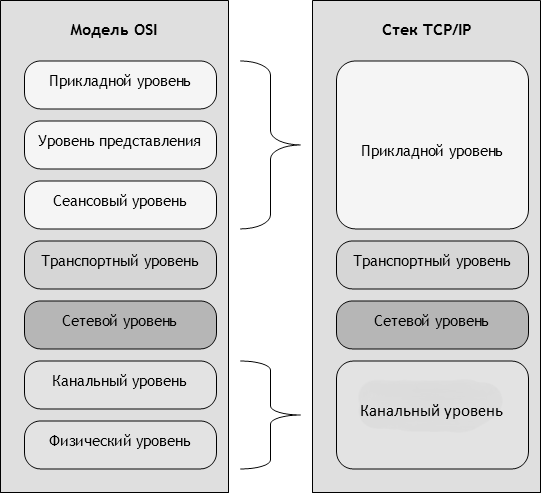
****

Рисунок 1.3 – Модель уровней TCP/IP

Эта многоуровневая структура позволяет *TCP/IP* эффективно управлять передачей данных между клиентами и серверами, обеспечивая надёжность и согласованность передачи на каждом этапе. Кроме того, каждый уровень независим, что упрощает внедрение изменений и добавление каких-либо новых технологий.

Протокол *TCP* предназначен для установления соединений и надёжной передачи данных между узлами. Основным принципом работы *TCP* является механизм «трёхэтапного рукопожатия», при котором устанавливается надёжное соединение между клиентом и сервером перед началом передачи данных между ними.

В начале происходит установление соединения, клиент посылает запрос на установление соединения серверу (*SYN*), сервер отвечает подтверждением (*SYN-ACK*), после чего клиент отправляет подтверждение (*ACK*), завершая процесс установления соединения. При этом, процесс обеспечивает надёжность при передаче и позволяет гарантировать то, что обе стороны готовы начать обмен данными.

После этого следует новый этап, передача данных, когда соединение установлено данные разбиваются на небольшие сегменты, которые передаются от клиента к серверу. *TCP* следит за тем, чтобы каждый сегмент был получен и обработан в правильном порядке, используя контрольные суммы и номера последовательностей.

В конце происходит завершение соединения, после передачи данных соединение разрывается аналогичным образом с помощью финальных сообщений (*FIN-ACK*) [1, c. 120]. На рисунке 1.4 представлен устройство соединения *TCP* протокола.



Рисунок 1.4 – Процесс установления и закрытия соединения *TCP*

*TCP* обеспечивает надёжную доставку данных, что делает его идеальным для приложений, требующих точной передачи информации, таких как базы данных, веб-сервисы или системы обработки изображений.

Протокол *IP* (*Internet Protocol*) тесно связан с *TCP* и работает на сетевом уровне и отвечает за маршрутизацию пакетов данных между устройствами в сети. Он не гарантирует доставку данных, но обеспечивает адресацию и передачу пакетов от отправителя к получателю. Каждый компьютер в сети имеет уникальный *IP*-адрес, который используется для идентификации и доставки данных.

Протокол *IP* использует две основные версии, первая, это *IPv4* – наиболее распространённая версия протокола с длиной адреса в 32 бита, что позволяет идентифицировать более 4 миллиардов уникальных устройств. Вторая, *IPv6* – новая версия с длиной адреса 128 бит, которая обеспечивает значительно больший адресный диапазон для подключения новых устройств в сеть.

Работа протокола *IP* заключается в том, чтобы данные, которые были отправлены по *TCP*, прошли через сеть и попали к нужному узлу. В отличие от *TCP*, протокол *IP* не проверяет, дошли ли данные успешно, его задача – просто передать пакет на указанный адрес.

*TCP/IP* является основой взаимодействия между клиентом и сервером в большинстве современных приложений. Например, в системах обработки изображений клиент может отправлять серверу запрос на обработку изображения, используя *TCP*, а сервер, обработав запрос, отправит результат обратно. *TCP* гарантирует, что данные будут переданы в полном объёме и в правильном порядке, что особенно важно при передаче файлов или изображений, где важна точность.

Благодаря механизму проверки целостности данных, протокол *TCP/IP* минимизирует ошибки при передаче и предотвращает потерю данных. Этот аспект является важным для приложений, работающих с большими объёмами информации, таких как системы обработки изображений с применением фильтров или других методов анализа.

* 1. **Использование протокола *HTTP***

Протокол *HTTP* (*HyperText Transfer Protocol*) является одним из самых распространённых и широко используемых протоколов для передачи данных в интернете. Он предназначен для обмена информацией между клиентом и сервером в веб-приложениях, обеспечивая передачу гипертекста, мультимедиа и других ресурсов. *HTTP* является протоколом прикладного уровня, работающим поверх стека *TCP/IP*, что делает его надёжным и универсальным инструментом для передачи данных в клиент-серверных системах.

Протокол *HTTP* работает по принципу запрос-ответ: клиент (обычно это веб-браузер) отправляет запрос серверу, который в ответ передаёт необходимые данные. Этот процесс выглядит следующим образом, клиент, отправляет *http*-запрос, который содержит метод (например, *GET* или *POST*), который определяет, какое действие нужно выполнить, а также *URL*-адрес ресурса и заголовки, содержащие дополнительную информацию (например, о типе данных, которые клиент может принять) [2, c. 243].

После этого сервер получает запрос, обрабатывает его и подготавливает ответ. Например, если клиент запрашивает веб-страницу, сервер ищет нужный файл и формирует *HTTP*-ответ. Следующий шаг, сервер отправляет ответ клиенту, он содержит статус обработки (например, код 200 – «*ОК*», если запрос выполнен успешно), заголовки с информацией о типе данных и, собственно, сам контент (например, *HTML*-код страницы). И в конце клиент получает и отображает ответ, будь то веб-страница, изображение или любой другой ресурс. Схему взаимодействия можно увидеть на рисунке 1.5.

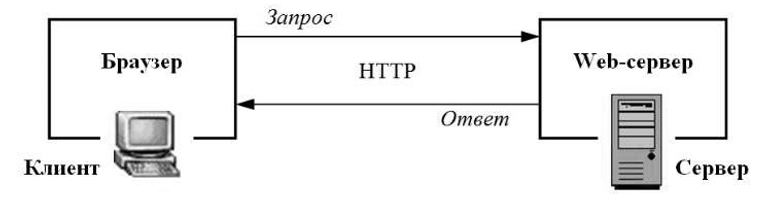


Рисунок 1.5 – Схема взаимодействия клиента и сервера через HTTP

*HTTP* поддерживает методы для работы с запросами, которые определяют тип действия, выполняемого клиентом при запросе к серверу, основными из них являются:

– метод *GET*, запрашивает данные с сервера. Это наиболее часто используемый метод, с помощью которого клиент запрашивает веб-страницы, изображения, файлы и другие ресурсы.;

– метод *POST*, используется для отправки данных на сервер, например, при отправке формы. *POST*-запросы передают информацию в теле сообщения, что делает их подходящими для передачи больших объёмов данных;

– метод *PUT*, загружает файл или данные на сервер, используется для обновления ресурса;

– метод *DELETE*, удаляет указанный ресурс на сервере;

– метод *HEAD*, работает как *GET*, но возвращает только заголовки ответа без тела данных, что полезно для проверки состояния ресурса.

Изначально *HTTP* не обеспечивал никаких механизмов безопасности, что делало его уязвимым для атак, таких как перехват данных или манипуляция содержимым запросов и ответов. Для решения этих проблем был разработан ***HTTPS*** – расширение *HTTP* с использованием *SSL/TLS* для шифрования данных. *HTTPS* работает по тому же принципу, что и *HTTP*, но добавляет слой безопасности, который шифрует данные при передаче между клиентом и сервером. Шифровка происходит так, что обеспечивает конфиденциальность и защищает от вмешательства третьих лиц [3, c. 75].

Протокол *HTTP* является фундаментом для большинства веб-приложений. Взаимодействие клиента и сервера через *HTTP* позволяет пользователю получать доступ к различным веб-ресурсам, таким как страницы, изображения, видео и другие мультимедийные элементы. Веб-браузеры автоматически отправляют *HTTP*-запросы каждый раз, когда пользователь переходит по ссылке или вводит *URL*-адрес в строке поиска.

Важной характеристикой *HTTP* является его статусная природа. Каждый запрос обрабатывается отдельно, без сохранения состояния между запросами, что делает его простым и эффективным для передачи небольших порций данных. Однако для приложений, которые требуют сохранения состояния (например, системы авторизации или онлайн-магазины), *HTTP* работает в связке с технологиями сессий или куки-файлов для поддержания информации о пользователе между запросами.

*HTTP* широко используется в клиент-серверных системах, где обмен данными осуществляется через веб-приложения. Протокол особенно полезен в тех случаях, когда требуется передача текстовых данных, изображений, мультимедиа или взаимодействие с различными *API*. Например, в системе обработки изображений клиент может загружать изображения на сервер через *POST*-запросы, а сервер в ответ передаёт обработанные результаты через протокол *HTTP*.

*HTTP* также активно используется для интеграции с другими сервисами через *REST* *API*, что делает его ключевым инструментом для создания распределённых систем. *REST* (*Representational State Transf*er) – это архитектурный стиль, в котором *HTTP*-запросы используются для выполнения различных операций над ресурсами, что позволяет легко создавать масштабируемые веб-приложения.

* 1. **Использование протокола *UDP***

Протокол *UDP* (*User Datagram Protocol*) является одним из основных транспортных протоколов, работающих поверх *IP*, наряду с *TCP*. В отличие от *TCP*, *UDP* не гарантирует доставку данных и не использует механизм установления соединения, что делает его более быстрым, но менее надёжным. Протокол *UDP* широко применяется в тех случаях, когда скорость передачи важнее гарантированной целостности данных, например, в мультимедийных приложениях, системах реального времени или онлайн-играх.

Протокол *UDP* предоставляет простой механизм передачи данных между устройствами в сети, не требующий установления соединения перед началом передачи. Это делает его менее ресурсоёмким и более подходящим для приложений, где важна минимальная задержка [4].

Основные характеристики UDP включают, первое, *UDP* не требует предварительного «рукопожатия» для начала передачи данных, что сокращает время на установку соединения. Второе, протокол не проверяет, дошли ли данные до получателя, и не пересылает их в случае потерь, что снижает нагрузку на сеть, но увеличивает вероятность потерь данных. Также, пакеты данных могут приходить в любом порядке, так как *UDP* не использует механизмы управления последовательностью пакетов, как это делает *TCP*. И последнее, меньшие накладные расходы, благодаря отсутствию механизмов подтверждения и управления соединением, *UDP* требует меньше ресурсов для обработки и передачи данных.

*UDP*-пакет состоит из заголовка и полезной нагрузки (данных). Заголовок *UDP* прост и состоит всего из четырёх полей, что позволяет сократить накладные расходы на передачу:

– порт отправителя (16 бит), указывает порт, с которого отправлен пакет;

– порт получателя (16 бит), указывает порт, на который должен быть доставлен пакет;

– длина (16 бит), общая длина заголовка и данных;

– контрольная сумма (16 бит), используется для проверки целостности данных.

Такой минимальный заголовок делает *UDP* очень лёгким и быстрым протоколом для передачи данных, что особенно полезно в приложениях с высокой скоростью передачи и минимальными требованиями к надёжности, структура сегмента представлена на рисунке 1.6.



Рисунок 1.6 – Структура UDP-сегмента

Принцип работы *UDP* заключается в отправке пакетов данных (датаграмм) без предварительного установления соединения. Клиент может отправить данные серверу или другому клиенту, указав его IP-адрес и порт. Сервер, в свою очередь, может обрабатывать полученные данные без подтверждения успешной доставки или повторной отправки в случае ошибок.

Этот подход позволяет *UDP* работать быстрее *TCP*, особенно в условиях, когда важно минимизировать задержки. Например, в потоковых мультимедийных приложениях небольшая потеря данных (несколько пакетов) не влияет критически на качество воспроизведения видео или аудио, но задержка в передаче может негативно сказаться на пользовательском опыте.

Протокол UDP находит применение в ряде систем, где скорость передачи данных является приоритетом над их надёжностью.

*UDP* активно используется для потоковой передачи видео и аудио в реальном времени. Поскольку задержки неприемлемы, потеря небольших пакетов может быть компенсирована без значительного ухудшения качества. В сетевых играх *UDP* используется для передачи данных в реальном времени, таких как положение игроков на карте или другие изменения в игровом процессе. Здесь крайне важна скорость, а не гарантированная доставка каждого пакета. Протокол *UDP* используется для обработки запросов системы доменных имён (*DNS*). *DNS*-запросы и ответы, как правило, представляют собой небольшие сообщения, где скорость важнее надёжности. Это только ряд вещей, где используется данный протокол.

**1.5 Сравнение основных протоколов**

Сравнительный анализ протоколов *TCP/IP*, *HTTP* и *UDP*, которые используются для передачи данных в клиент-серверных системах, является важной частью работы. Каждый из протоколов обладает уникальными особенностями, преимуществами и недостатками, что делает их более или менее подходящими для различных задач и сценариев использования.

Протокол *TCP* является одним из самых распространенных транспортных протоколов. Его основное преимущество заключается в надежности передачи данных. *TCP* использует механизмы подтверждений (*ACK*), что гарантирует успешную доставку данных на другой конец соединения. Если пакет теряется или повреждается, *TCP* автоматически инициирует его повторную передачу, что особенно важно в приложениях, где потеря даже одного байта может привести к искажению данных, таких как изображения. Также стоит отметить, что TCP обеспечивает упорядоченную передачу данных, что критично для последовательной обработки информации.

Механизмы управления потоком позволяют избежать переполнения буфера на принимающей стороне, что делает *TCP* идеальным выбором для систем, требующих высокой стабильности и надежности. Однако использование *TCP* также имеет свои недостатки. В частности, он может вводить дополнительную задержку из-за необходимости установления соединения и обработки подтверждений. Более того, реализация *TCP* требует больше ресурсов и усилий, что может увеличивать время разработки.

Протокол *HTTP* представляет собой стандартный протокол для передачи данных в Интернете, который работает поверх *TCP*. Его простота использования и понимания делает его популярным выбором для веб-приложений и *API*. *HTTP* работает на основе принципа запросов и ответов, что позволяет легко интегрировать его в различные системы.

Однако, несмотря на преимущества, *HTTP* также имеет недостатки. Во-первых, он может быть неэффективным для передачи больших объемов данных, таких как изображения, особенно если не используется кэширование. Во-вторых, *HTTP* является протоколом без состояния, что создает дополнительные трудности при необходимости поддержания состояния между клиентом и сервером.

Протокол *UDP* отличается от *TCP* и *HTTP* тем, что он обеспечивает низкую задержку и высокую скорость передачи данных. UDP не требует установления соединения и не включает механизмы подтверждения, что уменьшает накладные расходы на передачу данных и делает его идеальным для приложений, таких как видеоконференции или онлайн-игры, где важна скорость. Тем не менее, *UDP* имеет серьезные недостатки. В отличие от *TCP*, *UDP* не предоставляет гарантии доставки данных, что может привести к потере пакетов. Также пакеты, отправленные по *UDP*, могут приходить в любом порядке, что может вызывать проблемы при обработке информации [5, c. 213].

С учетом вышеизложенных характеристик, для реализации клиент-серверной системы обработки изображений с использованием высокочастотного фильтра был выбран протокол *TCP-IP*. Это решение обусловлено несколькими ключевыми факторами. Во-первых, надежность является критически важным аспектом при передаче изображений, где потеря даже одного байта может привести к серьезным искажениям.

Протокол *TCP*, обеспечивая надежную доставку и контроль ошибок, идеально подходит для этой задачи. Во-вторых, упорядоченность передачи данных является важным требованием в рамках нашего проекта. *TCP* гарантирует, что все данные будут доставлены в правильном порядке, что предотвращает искажения изображений и обеспечивает корректность их обработки. В-третьих, управление потоком, которое предоставляет *TCP*, позволяет регулировать скорость передачи данных, снижая риск переполнения буферов и увеличивая стабильность работы системы.

В связи с этими факторами, протокол *TCP-IP* был выбран в качестве основного средства передачи данных для реализации нашего проекта, обеспечивая надежность, стабильность и высокое качество передачи данных, что делает его оптимальным для задач, связанных с обработкой изображений.

**Список используемых источников**

1. Хант, К. *TCP*/*IP.* Сетевое администрирование, 3-е изд. – СПб.: Символ-Плюс, 2004. – 816 с.

2. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы / В. Г. Олифер, Н. А. Олифер. – СПб.: Питер, 2006. – 958 с.

3. Таненбаум, Э. Распределённые системы. Принципы и парадигмы / Э. Таненбаум, М. ван Стеен. – СПб.: Питер, 2003. – 877 с.

4. Использование сокетов для работы с *UDP* в *C*# и .*NET: Metanit. –* Электронн. Данные. – Режим доступа: <https://metanit.com/sharp/net/3.3.php>. – Дата доступа: 19.10.2024.

5. Таненбаум, Э. Компьютерные сети. 5-е изд. / Э. Таненбаум, Д.   
Уэзеролл. – СПб.: Питер, 2012. – 960 с.