СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

TCP	Transmission Control Protocol (протокол управления передачей)
FTP	File Transfer Protocol (протокол передачи файлов)
HTTP	HyperText Transfer Protocol (протокол передачи гипертекста)
SMB	Server Message Block (блочные сообщения сервера)
UDP	User Datagram Protocol (протокол пользовательских датаграмм)
FEC	Forward error correction (упреждающая коррекция ошибок)
MAC	Media Access Control (надзор доступа к средам)
Метаданные	данные, относящиеся к дополнительной информации о содержимом или объекте

ВВЕДЕНИЕ

Во время развития сетевых технологий прошлых поколений в основном решались проблемы технического характера. В следствие чего проектируемые системы на основе сетевых протоколов и стандартов прошлого всегда имеют уязвимости, которые могут быть обнаружены и использованы злоумышленником впоследствии. Из этого следует возможность взлома, компрометации, изменения и несанкционированного доступа к информации.

Проблема хранения конфиденциальных данных возникает в любой организации, работающей с информацией, потеря, утечка или искажение которой может привести к значительным последствиям. Диапазон принимаемых мер, варьируется от установки систем противодействия утечкам до принятия концепции нулевого доверия.

В настоящее время передовые методы защиты данных подразумевают возможность компрометации любого участка защищенной системы. Несмотря на технический прогресс в области защиты информации и сетевых технологий, даже новейшая инфраструктура нуждается в регулярных обновлениях программно-технического комплекса и отслеживания новых методик противодействия атакам. При этом подобный комплекс мер не ограничивает возможность злоумышленнику, внедренному в организацию, распространить данные за пределы внутренней сети.

Изоляция сети – наиболее эффективный метод борьбы с утечками. Даже в случае полной компрометации внутренней сети, злоумышленник не сможет передать конфиденциальные данные за пределы локальной сети. К сожалению, даже полностью изолированная система нуждается в доступе во внешний мир для выполнения своих функций. Данную проблему можно решить посредством физических накопителей, однако подобное решение не дает гарантий того, что данный накопитель не станет хранилищем конфиденциальной информации из внутренней сети, создавая возможность потери данных.

Для решения проблемы невозможности работы полностью изолированной сети используют методы однонаправленной передачи данных. Термин «Диод данных»[1] означает систему, в которой данные могут передаваться только в одном направлении, полностью блокируя любые возможности обратной передачи данных. Таким образом, даже в случае полной компрометации внутренней сети, передать данные во внешний мир не представляется возможным, предотвращая возможную утечку данных.

Диоды данных могут быть выполнены в программном или аппаратном варианте. В случае аппаратной реализации, корпус содержит интерфейсы для

подключения принимающей и передающей сети, а также разъём питания. Недостатком подобных устройств является невысокая скорость их работы, а также необходимость использования специальных протоколов передачи данных, не нуждающихся в обратном канале связи.

Программный диод данных – это сетевое устройство, в котором ограничение на передачу информации определяется логикой работы прошивки или конфигурации. Данный фактор позволяет реализовывать однонаправленную сеть на уже существующей инфраструктуре. Недостатком подобной системы, является теоретическая возможность утечки информации через обратный канал.

Так как большинство современных протоколов передачи данных общего назначения требует наличие двунаправленной связи, диод данных не может работать напрямую с распространёнными протоколами TCP, FTP, HTTP и нуждается в программно-аппаратном комплексе.

Реализуется подобный комплекс на базе прокси серверов, которые эмулируют работу TCP, SMB и других стандартов передачи данных. Дополнительным достоинством данной системы является возможности контроля входных данных, их мониторинга и фильтрации.

Диоды данных могут использоваться не только для защиты конфиденциальных данных, но и для защиты устройств от несанкционированного доступа. В случае работы с производственной инфраструктурой, возникает задача защиты устройств от возможности изменения их конфигурации удалённо. Для этой цели диод данных передаёт данные от датчика или системы отслеживания во внешнюю сеть, однако предотвращает возможность получения доступа к конфигурации устройства.

Цель дипломной работы – исследование принципов работы однонаправленных сетей и реализация программного комплекса для работы однонаправленной ведомственной сети.

Задачи дипломной работы:

– сравнительный анализ существующих систем однонаправленной передачи данных.

– разработка системы однонаправленной передачи данных по техническому заданию

– проведение технико-экономических обоснований исследования и разработки системы однонаправленной передачи данных.

1 СРАВНИТЕЛЬНЫЙ АНАЛИЗ АНАЛОГИЧНЫХ СИСТЕМ ЗАЩИТЫ ДАННЫХ

1.1 Анализ существующих систем защиты данных

Однонаправленная передача данных в закрытую сеть является частным случаем системы разграничения доступа. Подобными свойствами обладают межсетевые экраны и системы контроля трафика внутри локальной вычислительной сети. Таким образом, корректно сравнение разрабатываемой системы с межсетевыми экранами и другими методиками разграничения доступа к данным по сети.

В [А.1] приведена система, в которой посредством внедрения между внешней сетью и защищаемой сетью специального узла «Модуль Администрирования» происходит контроль трафика. Пример предлагаемой в патенте системы представлен на рисунке 1.1.

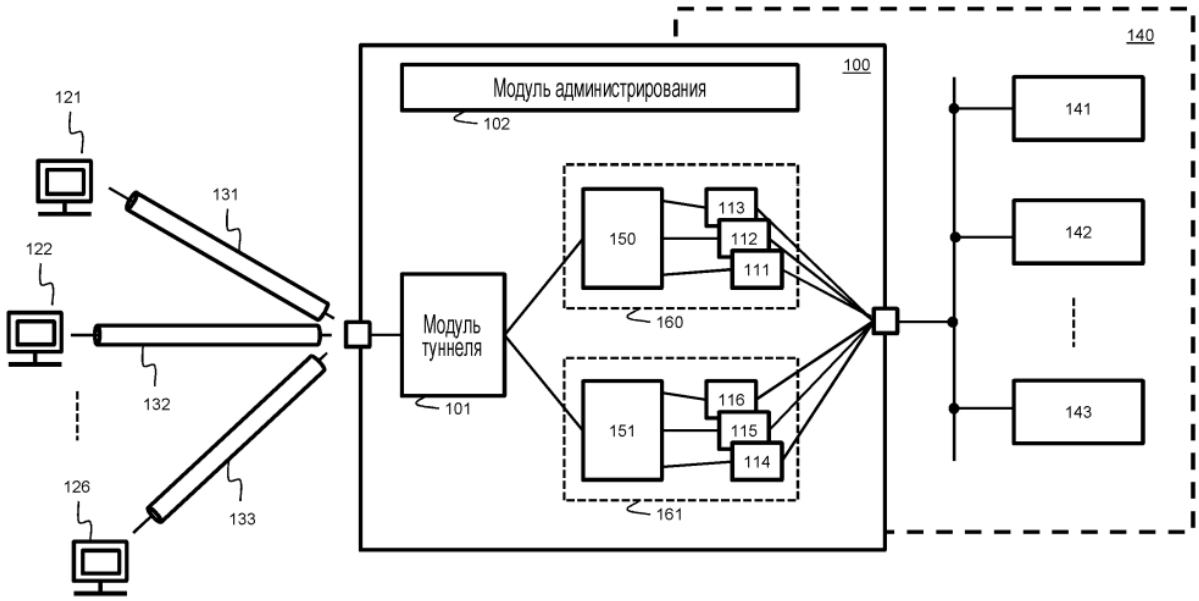

Рисунок 1.1 – Изображение, представленное авторами патента с моделью предлагаемой системы

В данном примере три сетевых устройства (серверы 141, 142 и 143 приложений) являются частью частной сети 140. Доступ к серверам 141-143 получается изнутри частной сети 140 через частный сетевой адрес. Другими словами, адресация серверов 141-143 приложений не может быть осуществлена посредством их частных сетевых адресов извне частной сети 140. Частная сеть 140 отделена от внешней сети шлюзом 100, тем самым

обеспечивая прохождение трафика между внешней сетью и сетью 140 управляемым образом.

Предложенная система может идентифицировать клиентов 121-126 в качестве «доверенных клиентов» с правами доступа к одному или более из серверов 141-143 приложений внутри частной сети 140 для того, чтобы использовать функционирующие на них службы.

Для того чтобы управлять доступом клиентов 121-126 к серверам 141-143 приложений, сетевые туннели 131-133 создаются между клиентами 121-126 и шлюзом 100. Таким образом, частная сеть 140 расширяется для клиентов 121-126. Вследствие этого, клиенту 121-126, несмотря на то, что физически он не находится в частной сети 140, предоставляется адрес частной сети в диапазоне частной сети 140, и может, следовательно, потенциально осуществлять доступ к всем серверам 141-143 приложений посредством их соответствующего адреса частной сети.

Процесс создания сетевого туннеля между сетями представлен на рисунке 1.2.

Рисунок 1.2 – Алгоритм создания сетевого туннеля между клиентским устройством и частной сетью

Посредством данного процесса, клиентское устройство 121-126 соединяется с частной сетью 140 через шлюз 100. На первом этапе 201, модуль 101 туннеля принимает первый запрос соединения от клиентского сетевого устройства 121, чтобы создать первое сетевое соединение со шлюзом 100. За

этим, сетевое соединение создается на этапе 202. Данное первое сетевое соединение используется, чтобы осуществлять обмен информацией управления между клиентом 121 и шлюзом 100, и, в частности, с модулем 102 администрирования, реализованным в шлюзе 100. Для того, чтобы знать, что соединение служит для целей управления, модуль туннеля может инспектировать первый пакет данных, обмен которым осуществляется через каждое вновь созданное сетевое соединение. Если пакет данных является пакетом данных управления, модуль 101 туннеля идентифицирует сетевое соединение в качестве соединения управления и будет перенаправлять все дальнейшие пакеты, принимаемые через данное соединение, модулю 102 администрирования.

Представленная в патенте система позволяет гибко настраивать правила передачи данных в сети, а также проводить анализ передаваемого трафика. Данная особенность позволяет реализовать системы защиты от утечек, гибко разграничивать возможность получения данных используя систему авторизации, а также вести мониторинг получения доступа к данным, для своевременного обнаружения попытки произвести утечку данных.

Несмотря на расширенные возможности контроля трафика в сети, данная система имеет ряд уязвимостей, которые позволяют произвести атаку на защищаемые данные с целью уничтожения, модификации или хищения. Наличие слоя между защищаемой сетью и внешним миром, не способно предотвратить попытку передачи конфиденциальных данных во внешний мир со стороны защищаемой сети. Также данная система нуждается в регулярных обновлениях программно-технического комплекса, что, впрочем, не означает абсолютную защиту от взлома одного или нескольких устройств внутренней сети.

Таким образом, данная система лучше подходит для организации работы удаленных сотрудников, так как предоставляет удобный способ доступа во внутреннюю сеть. По этой же причине, данная система не может предоставить абсолютную защиту конфиденциальных данных, и не может использоваться в сетях, где отсутствие возможности утечки, важней удобства доступа к защищаемой информации.

В [А.2] приведено устройство которое представляет собой межсетевой фильтр, включаемый между двумя компьютерными сетями таким образом, что весь обмен информацией между указанными сетями ограничивается с помощью правил фильтрации, при этом межсетевой фильтр содержит по меньшей мере два сетевых интерфейса для обмена данными между клиентами первой компьютерной сети и второй компьютерной сети из двух. Устройство дополнительно содержит узел обработки трафика, включающий устройство

управления, обеспечивающее ввод правил фильтрации трафика и хранение информации о правилах фильтрации, устройство анализа трафика, обеспечивающее проверку соответствия поступающей информации правилам фильтрации, а также коммутирующее устройство, через которое указанные сетевые интерфейсы соединены между собой и которое обеспечивает прохождение разрешенной правилами фильтрации информации между сетевыми интерфейсами и блокировку неразрешенной правилами фильтрации информации, при этом правила фильтрации запрещают транзитную передачу любых пакетов между указанными сетевыми интерфейсами кроме тех, которые имеют разрешенные признаки и параметры адресации в своих заголовках, форму информационной части пакета, соответствующую шаблону, хранящемуся в памяти межсетевого фильтра, а также параметры запроса или ответа, соответствующие множеству разрешенных значений, хранящихся в памяти межсетевого фильтра.

Схема подключения компьютерной сети к представленному устройству представлено на рисунке 1.3.

Рисунок 1.3 – Структурная схема подключения компьютерной сетей к устройству фильтрации трафика

Согласно разработке для управления процессами фильтрации трафика МСФ содержит специальный узел обработки трафика (УОТ), устройство управления которого информационно изолировано от сетевых интерфейсов, а взаимодействие с ним осуществляется через отдельный интерфейс управления. Все изменения программы фильтрации трафика, а также управление соединениями могут быть выполнены исключительно через интерфейс устройства управления УОТ, что полностью устраняет возможность несанкционированного доступа с МСФ со стороны сетей КС1 и КС2.

В [А.3] приведена система однонаправленной передачи данных из сети общего пользования в сеть недоступную для общего пользования, содержащая сетевой узел связи сети недоступной для общего пользования, к входу которого подключен выход приемника данных, вход которого через линию

связи соединен с первым выходом передатчика данных, первый вход которого соединен с выходом сетевого узла связи сети общего пользования, отличающаяся тем, что на передающей стороне указанной системы введен узел квитирования, первый вход которого соединен со вторым выходом передатчика данных, а второй вход соединен с третьим выходом передатчика данных, второй вход которого соединен с выходом узла квитирования, при этом узел квитирования выполнен с возможностью формирования сигнала готовности приемника данных, задержанного относительно сигнала завершения передачи данных, формируемого по сигналу разрешения передачи данных, на величину, которая определена эмпирически из условия передачи без потерь данных из сети общего пользования в сеть недоступную для общего пользования, для конкретных сетей.

Протокол однонаправленной передачи позволяет передавать данные из незащищенной сети в безопасную сеть. Незащищенный шлюз может получать данные и/или информацию, предназначенные для защищенной сети, от одного или нескольких устройств.

В [А.4] приведен протокол передачи данных в однонаправленной сети который может фрагментировать данные и/или информацию на более мелкие фрагменты и передавать фрагменты на защищенный шлюз по однонаправленному каналу связи. Конечный шлюз может проверять фрагменты, используя одно или несколько правил, и повторно собирать фрагменты после проверки данных. Собранные данные могут быть отправлены на конечное сетевое устройство. Протокол однонаправленной передачи может обеспечивать аппаратно-независимое решение для передачи данных по однонаправленному каналу связи.

[А.5] предлагает для защиты связи технологического предприятия через однонаправленный диод данных, публиковать данные через диод на приемное устройство на удаленном конце. Публикация различных данных зависит от контекста и информации. Передающим устройством через заданный интервал может производится повторная передача данных.

Конфигурация может производится промышленными протоколами конфигурации или любым удобным протоколом

1.2 Принципы построения однонаправленных сетей

Документ «Unidirectional Networking» [2] от GIAC (Global Information Assurance Certification) описывает процесс разработки и возможные сложности в процессе реализации однонаправленной системы передачи

данных. В данной работе предлагается рассматривать двунаправленное оптическое подключение, как пару отдельных однонаправленных каналов.

Путем отключения одного из каналов, компьютеры будут соединены одним каналом, в который только один из компьютеров сможет передавать данные, а другой только принимать.

Для работы через однонаправленную сеть, нужно использовать протокол без активного подключения, а каждый передаваемый пакет воспринимать независимо. UDP не смотря на то, что является протоколом без необходимости наличия открытого соединения, однако, это вовсе не означает, что на реальных устройствах он реализован как полностью однонаправленный. В случае некоторых операционных систем, отсутствие ответа на отправку UDP приведет к внутренней ошибке.

В качестве метода обеспечения однонаправленной передачи данных, рекомендуется разместить аппаратный диод данных между устройствами, для обеспечения гарантии, того, что данные не могут быть отправлены в обратном направлении.

Если сетевое устройство не может принимать данные, то невозможно удаленно выполнять инструкции. С точки зрения удаленного устройства, нет способов изменить данные на передающем устройстве, без наличия физического доступа.

С точки зрения безопасности, компьютер, который может только принимать данные, может считаться конфиденциальным для внешнего пользователя. Если злоумышленник не может получить никакой информации о удаленном компьютере, то не сможет провести эффективную атаку на защищенный компьютер. Единственный способ просматривать информацию с защищенного таким образом компьютера – получение физического доступа. Теоретически возможна атака, в случае если злоумышленник обладает полным знанием о приемной стороне, но практически невозможна в случае, если злоумышленник с ней не знаком. Однако даже в данном случае, произвести утечку данных не получится. Таким образом, для внешнего пользователя, данный компьютер полностью конфиденциален.

Возникает множество практических проблем, когда сеть работает только в одном направлении. При использовании оптических сетевых карт, сигнал несущей предается по линии передачи. В случае отсутствия несущей на линии приема, сетевая карта не будет передавать данные. Для того, чтобы оптическая сетевая карта производила передачу, одним из решений предлагается использование несущей из другой сетевой карты на стороне передачи. Схема подключения передающей стороны представлена на рисунке 1.4.

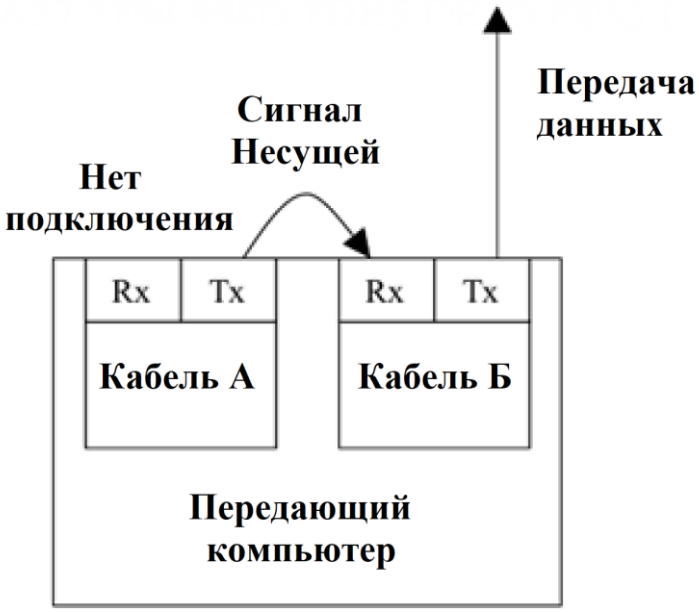

Рисунок 1.4 – Схема подключения передающей стороны для работы в однонаправленном режиме

Невозможно гарантировать отсутствие ошибок при передаче данных через однонаправленную сеть. Даже в случае, если принимающий компьютер может понять, какие данные повреждены или утеряны, нет способа, которым он мог бы об этом сообщить источнику.

Потеря данных может быть минимизирована использованием избыточности, через передачу данных больше, чем один раз, или с передачей дополнительной информации, такой как кода FEC (Forward Error Correction), для восстановления утерянных данных.

Для того, чтобы компьютер имел возможность принять пакет, он должен иметь MAC адрес на сетевой карте. MAC адрес это уникальный идентификатор выданный каждому сетевому устройству и используется для соединения данных на втором уровне OSI для идентификации каждой сетевой карты в сети.

Протокол определения адреса (ARP) используется устройствами для того, чтобы создать ассоциацию между MAC адресом и IP адресом устройства. Компьютер, который может производить обмен данными только в одном направлении не может использовать ARP для определения MAC адреса принимающего компьютера, так как не способен получить ответ на ARP запрос.

Для того, чтобы обойти данное ограничение можно воспользоваться несколькими путями:

– вручную настроить таблицу определения адреса на передающем узле;

– настроить сетевую карту так, что любая информация, полученная на нее, будет принята вне зависимости от адресата;

– убедится, что передающая сторона и приемная сторона находятся в одной подсети и использовать широковещательный адрес.

Таким образом, используя однонаправленное соединение можно подключить незащищенную сеть к защищенной и гарантировать конфиденциальность защищаемой сети. Использование однонаправленной передачи данных значительно ограничивает число возможных сетевых уязвимостей.

1.3 Анализ существующего рынка аппаратных диодов данных (однонаправленных шлюзов)

Диод данных предназначен для гарантированной однонаправленной передачи информации между защищённым сегментом сети и внешними сетевыми устройствами.

Системы, содержащие в себе диод данных можно разделить на несколько типов:

– для защиты от утечек данных;

для защиты конфигурации оборудования;

– для систем репликации.

В первом случае данные могут только поступать в защищенную сеть, не позволяя произвести утечку. Во втором случае, данные с датчиков или системы трекинга свободно проходят диод данных, однако для изменения конфигурации оборудования потребуется наличие физического доступа к устройству. В случае систем репликации, использование диода данных не позволит злоумышленнику получить доступ к исходному серверу.

Первые диоды данных появились ещё в конце прошлого века, однако широкое распространение такие устройства получили с ростом числа целевых кибератак на объекты критической инфраструктуры. Возможно, именно поэтому мировой рынок инструментов для однонаправленной передачи данных в последние годы демонстрирует стабильный рост. Вместе с тем объёмы продаж в сегменте диодов данных невелики по сравнению с другими сферами информационной безопасности.

Наибольшее число заказчиков диодов данных сконцентрировано в энергетическом секторе, нефтегазовых компаниях, государственных организациях и предприятиях, эксплуатирующих объекты критической инфраструктуры. Производственные и транспортные компании, использующие большое количество промышленных датчиков и

программируемых контроллеров, вкладываются в защиту таких устройств от направленных проникновений и кражи данных. Вопросы информационной безопасности устройств автоматизации с каждым годом приобретают всё большее значение. Главным драйвером мирового рынка диодов данных специалисты называют возросшую активность киберпреступников, атакующих нефтегазовый сектор. Подключение предприятий отрасли к технологически сложным решениям, таким как IoT, призвано увеличить производительность и снизить затраты, но одновременно делает их более уязвимыми к кибератакам. Действия злоумышленников способны остановить деятельность компании, что может привести к огромным финансовым, репутационным и – в некоторых случаях – человеческим потерям, а также к экологическим катастрофам [3].

Ключевыми вендорами, поставляющими диоды данных, являются компании: Advenica AB. BAE Systems. Belden. Deep Secure. Fibersystem. Forcepoint. Fox-IT. Garland Technology. Nexor. OPSWAT. Owl Cyber Defense. Siemens. ST Engineering. Waterfall.

В странах СНГ также существуют компании разрабатывающие и поставляющие диоды данных для внутреннего рынка. Большинство компаний работает в России.

Использование таких устройств в России предусматривается нормативными документами, регулирующими безопасность в государственных информационных системах и обработку персональных данных. Для организации доступа к информации, содержащей государственную тайну, оборудование должно быть сертифицировано ФСТЭК России с указанием уровня контроля.

В России диоды данных выпускают как минимум пять производителей: «АйТи БАСТИОН». «АМТ-Груп». «Ореол Секьюрити». «Росэлектроника». «СиЭйЭн». «Эшелон». «Центр безопасности информации».

Компания «АМТ-Груп» предлагает линейку аппаратных и программно-аппаратных решений для однонаправленной передачи данных под брендом InfoDiode[4]. Они предназначены для организации обмена данными со критически значимыми сегментами. Все системы сертифицированы ФСТЭК России по ТУ.

Решения «АМТ-Груп» поддерживают передачу как стандартных транспортных протоколов (FTP / FTPS, CIFS, SMTP, SFTP, StartTls, IPsec, UDP), так и специализированных для SCADA-систем и OPC-серверов промышленных протоколов (OPC UA, Modbus, MQTT). Диоды могут быть «из коробки» интегрированы с различными прикладными сервисами и

решениями, в том числе SNMP, Syslog, NTP, Active Directory. Заявленная пропускная способность диода данных – 1 Гбит/с.

Преимущества решений AMT InfoDiode:

– Интеграция с Active Directory, Syslog, SIEM-системами; формирование файла метаинформации для его анализа средствами DLP.

– Возможность передачи данных SCADA-систем и OPC-серверов, поддержка FTP / FTPS, CIFS, SMTP, SFTP и др., а также промышленных протоколов; приоритизация передачи данных и потоков.

– Поддержка сценариев репликации баз данных Microsoft SQL, PostgreSQL, сценариев передачи обновлений WSUS, антивирусов KPSN от Kaspersky, сценариев трансляции рабочего стола оператора за границу защищаемого сегмента.

– Помехоустойчивое кодирование, резервное копирование настроек.

Среди представленных на рынке устройств, доступны также диоды данных, получившие сертификацию Минобороны России, и может применяться в сетях, где обрабатывается информация составляющая государственную тайну

Комплект изделия «Рубикон-ОШ» [5], выпускаемого компанией «Эшелон», состоит изо двух полукомплектов (передатчик и приёмник), соединённых с использованием специализированных оптических плат. Таким образом обеспечивается полная гальваническая развязка передающего и принимающего полукомплектов, находящихся в сегментах разного уровня секретности, с невозможностью прохождения сетевых пакетов в обратном направлении на физическом уровне. «Рубикон-ОШ» может функционировать в следующих режимах:

– передача сетевых пакетов через однонаправленную связь посредством маршрутизации IP.

– односторонняя передача файлов с одного FTP-сервера, подключённого к передающему комплекту ОШ, на другой FTP-сервер, подключённый ко принимающему комплекту ОШ.

– может выполнять функции маршрутизатора (коммутатора уровня L3)

– объединять физические интерфейсы в сетевой мост (коммутатор уровня L2)

– работать как межсетевой экран и система обнаружения и предотвращения вторжений

Преимущества однонаправленного шлюза «Рубикон»:

– Производительность межсетевого экрана до 8,5 Гбит/с.

– Производительность системы обнаружения вторжений до 2,5 Гбит/с.

– Наличие накопителя информации объёмом 1 ТБ.

– Широкий выбор сетевых интерфейсов (6 медных портов RJ-45, 2 оптических порта 10G SFP+З

– Устройство сертифицировано Минобороны России и может применяться в сетях, где обрабатывается информация составляющая государственную тайну.

Рассмотрим зарубежные решения в области диодов данных.

Компания Owl Cyber Defense[8] является одним из крупнейших производителей диодов данных. В данный момент вендор предлагает широкую линейку устройств для однонаправленной передачи трафика, оптимизированных для различных задач. Флагманом модельного ряда является OPDS-1000. Система «всё в одном» в формфакторе 1U обеспечивает передачу данных со скоростью от 26 Мбит/с до 1 Гбит/с в зависимости от конфигурации. Устройство сертифицировано по критериям безопасности EAL4+ и обладает встроенной поддержкой протоколов UDP, TCP/IP, SNMP, SMTP, NTP, SFTP и FTP. Данное устройство изображено на рисунке 1.9.

Помимо этого, разработчик предлагает комплексные решения для однонаправленной передачи данных, состоящие из пар прокси-серверов на оборудовании Dell. Система позволяет организовать полноценное движение данных с возможностями расширенного управления электропитанием, резервного копирования и самозащиты устройств. Система Owl PaciT рассчитана на передачу «сырого» трафика.

1.4 Вывод

В результате проведенного анализа современного состояния науки и техники в области защиты информации посредством однонаправленной передачи данных, сделан вывод, что системы, содержащие в себе диод данных являются наиболее надежными для обеспечения конфиденциальности данных.

Также приведены основные принципы построения систем однонаправленной придачи данных и обозначены ограничения и технические сложности в процессе разработки подобной системы.

Таким образом, одной физической организации односторонней передачи данных недостаточно, нужен программно-аппаратный комплекс способный решать задачи по маршрутизации данных в сети, поддержанию современных протоколов передачи данных. Существующие аналоги поддерживают также распространенные протоколы TCP, FTP, HTTP, которые требуют наличие двунаправленной связи.

2 ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ТРЕБОВАНИЙ К ВЕДОМСТВЕННОЙ СЕТИ

Согласно техническому заданию, для маршрутизации пактов используется модель TCP/IP (Transmission Control Protocol/Internet Protocol).

Стек протоколов TCP/IP — сетевая модель, которая описывает процесс передачи цифровых данных. Регламентирует и описывает всю уровневую архитектуру и протоколы, входящие в стек, документ RFC 1122 [7]. В данной модели стандарт выделяет четыре уровня: канальный, межсетевой, транспортный и прикладной.

На канальном уровне модели TCP/IP на уровне сетевых устройств происходит обмен информацией, определяется как данные будут передаваться от одного устройства к другому. На данном уровне используется протокол Ethernet определенный в стандарте IEEE группы 802.3[8]. Отправляющее и принимающее устройство в сети имеют определенные уникальные идентификаторы – MAC-адреса. Такие идентификаторы вместе с типом передаваемых данных и самими данными инкапсулируются в Ethernet. В следствие чего составляется фрагмент данных, который называется фреймом или кадром.

Межсетевой уровень позволяет устройствам из разных сетей взаимодействовать между собой, объединить локальные сети. Взаимодействие между сетями осуществляют пограничные и магистральные маршрутизаторы. Маршрутизатор отправляет пакет напрямую при условии, что устройство назначения находится в той же подсети, что и отправляющее устройство. Для того, чтобы определить к какой подсети принадлежит устройство назначения, маршрутизатор использует протокол интернета IP (Internet Protocol), описанный в документе RFC 791[9]. Каждое сетевое устройство в глобальной сети имеет свой уникальный идентификатор – IP-адрес. Этот протокол необходим для определения и доставки данных к устройству назначения.

На транспортном уровне происходит передача пакетов между сетевыми устройствами с использованием протокола UDP (User Datagram Protocol). – очень быстрый протокол, поскольку в нем определен самый минимальный механизм, необходимый для передачи данных. UDP не требует открывать соединение, и данные могут быть отправлены сразу же, как только они подготовлены. UDP не отправляет подтверждающие сообщения. Этот протокол определен в RFC 768[10].

На прикладном уровне модели TCP/IP происходит предоставление услуг пользователю или обмен данными по уже установленным соединениям.

File Transfer Protocol (FTP) – протокол передачи файлов по сети, описанный в спецификации RFC 959[11]. Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером.

В качестве протокола динамической маршрутизации стека TCP/IP используется протокол OSPF (Open Shortest Path First) – основанный на технологии отслеживания состояния канала и использующий для нахождения кратчайшего пути Алгоритм Дейкстры. Открытый протокол маршрутизации не устанавливает отдельных требований к расчету метрики и оценки маршрутов. Его стандарт определяет стоимость каждого пути. В случае прохождения маршрута через несколько соединений их стоимость суммируется. Оптимальным признается маршрут с наименьшей стоимостью. Протокол OSPF определен в RFC 2328[12].

Данные через диод данных передаются с использованием технологии Fast Ethernet которая обеспечивает скорость передачи 100 Мбит/с. Данный стандарт описывается в IEEE 802.3 который содержит описание различных стандартов передачи данных посредством кабелей Ethernet. Для работы однонаправленного шлюза используются стандарты 100BASE-TX и 100BASE-FX описанные в стандарте IEEE 802.3u[13].

В качестве среды передачи 100BASE-TX применяются две витые пары. Одна линия используется для передачи данных, а вторая — для их приема. Спецификация содержит описания как экранированных, так и неэкранированных витых пар.

В сетях стандарта 100Base-FX используется волоконно-оптический, длиной сегмента до 412 метров. Стандарт определяет, что в кабеле имеются две жилы многомодового волокна – одна для передачи, а другая для приема данных.

Маршрутизация данных внутри каждой сети, реализуется на стандарте Gigabit Ethernet, который описан в документе IEEE 802.3ab[14]. Данный стандарт обеспечивает скорость передачи данных 1 Гбит/с.

Для реализации программы передачи и приема данных в сети используется язык программирования C++20, описанный в стандарте ISO/IEC 14882:2020[15].

3 РАЗРАБОТКА И ОБОСНОВАНИЕ СТРУКТУРНОЙ СХЕМЫ ПРОЕКТИРУЕМОЙ СЕТИ

3.1 Разработка схемы сети

Для выполнения требований технического задания необходимо разработать структурную схему проектируемой сети. На рисунке 3.1 представлена разработанная схема сети.

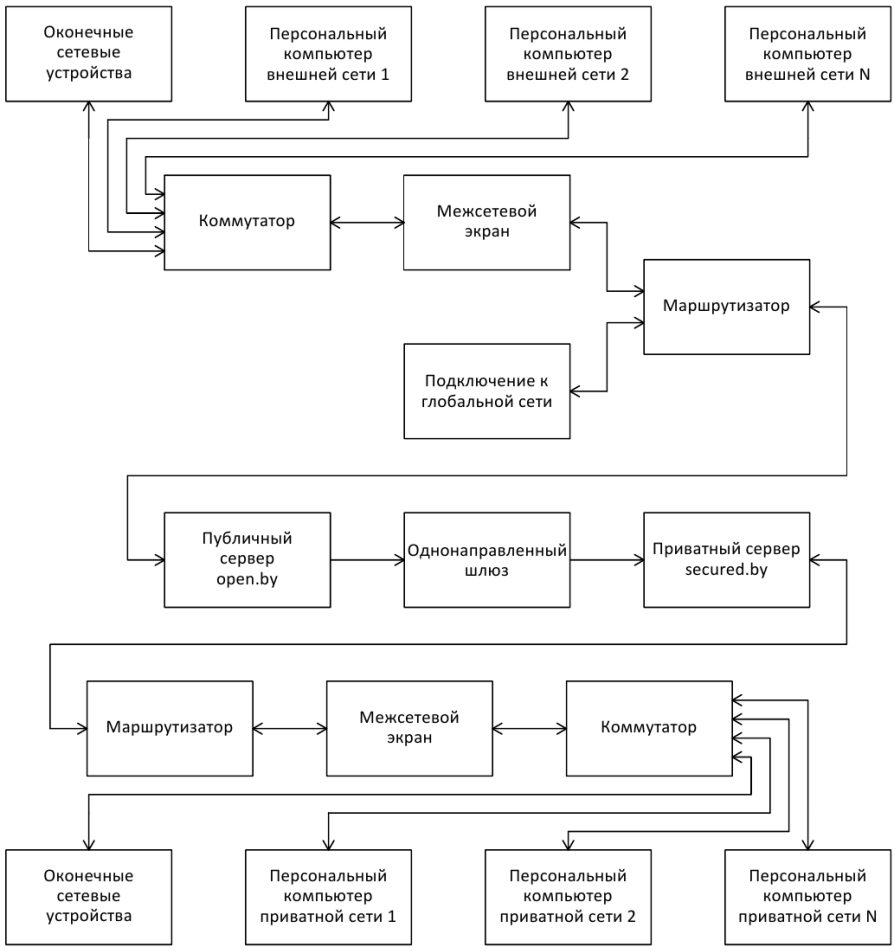

Рисунок 3.1 – Структурная схема сети

В представленной модели однонаправленной ведомственной сети важную роль играет диод данных, который позволяет осуществить фильтрацию трафика.

Согласно техническому заданию, в сети происходит односторонняя передача данных из публичной подсети в закрытую подсеть. Для передачи данных между подсетями в представленной модели ведомственной сети, используется FastEthernet, со скоростью работы до 100 Мбит\с, что соответствует техническому заданию.

Между публичной и закрытой сетью расположен программный однонаправленный шлюз, пропускающий данные только в направлении закрытой сети, предотвращая возможные утечки данных. В подсетях расположены два сервера. Сервер в публичной сети предназначен для передачи данных на сервер в закрытой сети. Публичная подсеть, изображена на рисунке 3.2.

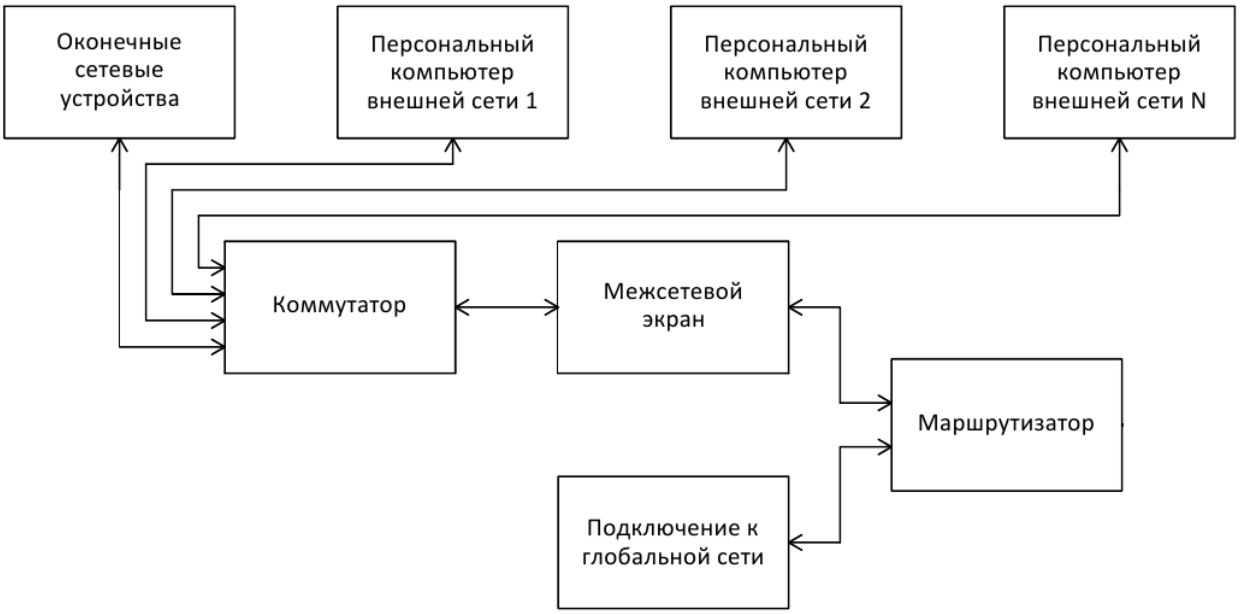

Рисунок 3.2 – Структурная схема публичной подсети

В открытой сети находится ftp-сервер open.by с ipv4-адресом 192.168.100.10/28, позволяющий хранить, обрабатывать и передавать информацию. Устройства, находящиеся в одной сети с сервером, могут беспрепятственно публиковать данные в специальный каталог, размещенный на сервере. Защищенная сеть, изображена на рисунке 3.3.

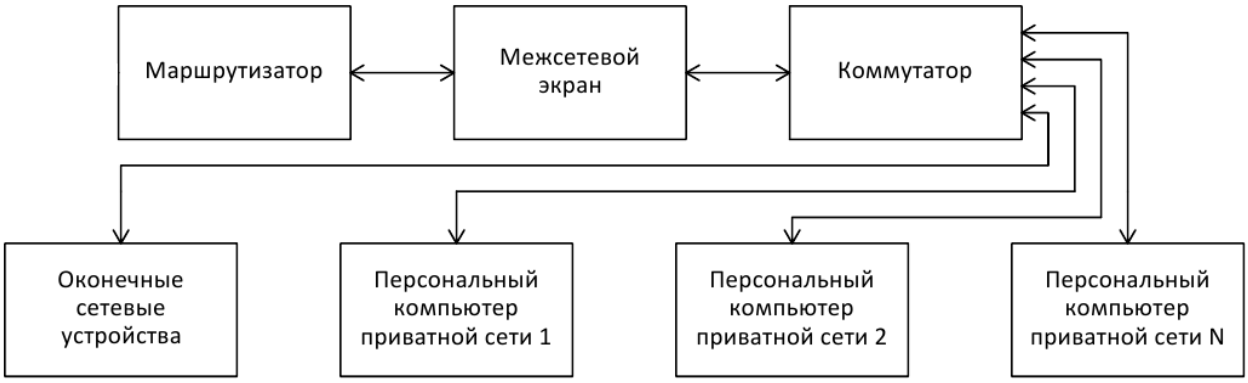

Рисунок 3.3 – Структурная схема закрытой подсети

В защищенной подсети расположен ftp-сервер secured.by с ipv4-адресом 192.168.50.10/28, который позволяет хранить информацию и передавать ее только в пределах своей сети.

Для демонстрации принципа работы сети, рассмотрим ситуацию, когда пользователю персонального компьютера внешней сети, расположенного в публичной сети, требуется передать данные на персональный компьютер внутренней сети, расположенный в приватной сети.

Для этого, на первом шаге, пользователь публичной сети, подключается к серверу open.by посредством протокола FTP и открывает каталог для отправки данных в защищенную сеть. Далее пользователь загружает необходимые данные на публичный сервер.

Программное обеспечение, размещенное на публичном сервере, в ходе запланированного сканирования каталога на предмет изменений или в ходе перехвата системного события записи файла в каталог, начинает отправку файла на удаленный сервер secured.by посредством пакетов UDP через диод данных. Движение трафика изображено на рисунке 3.4

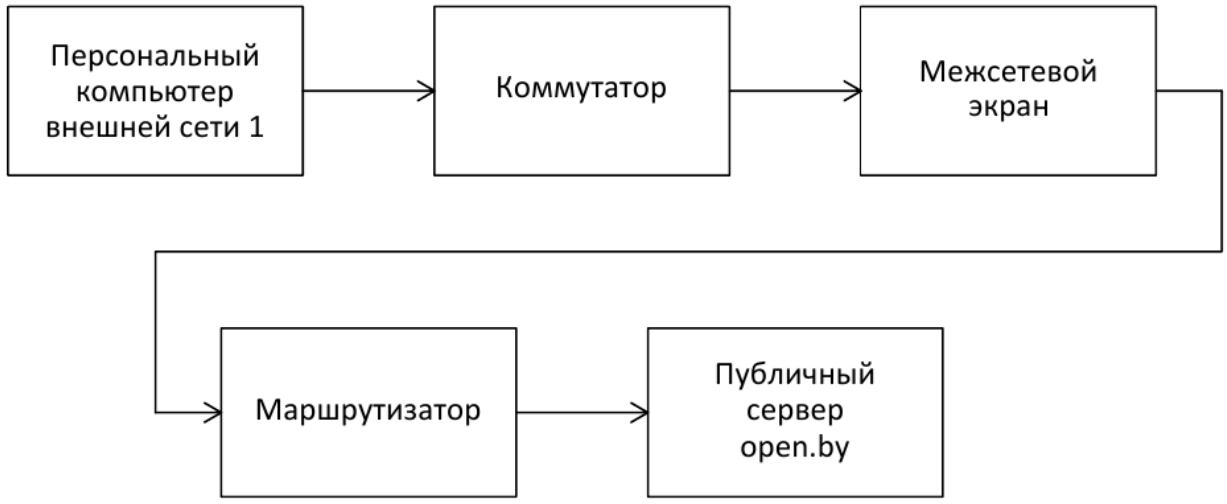

Рисунок 3.4 – Схема передачи данных на в публичной сети

В процессе передачи данных на сервере secured.by, создается, заполняется приходящими данными и проверяется на целостность приходящий файл. По завершению передачи пакетов, происходит копирование итогового файла из каталога «Downloading» в рабочий каталог.

После копирования, компьютер внутренней сети может получить доступ к каталогу на сервере secured.by. Схема передачи данных во внутренней сети описана на рисунке 3.5.

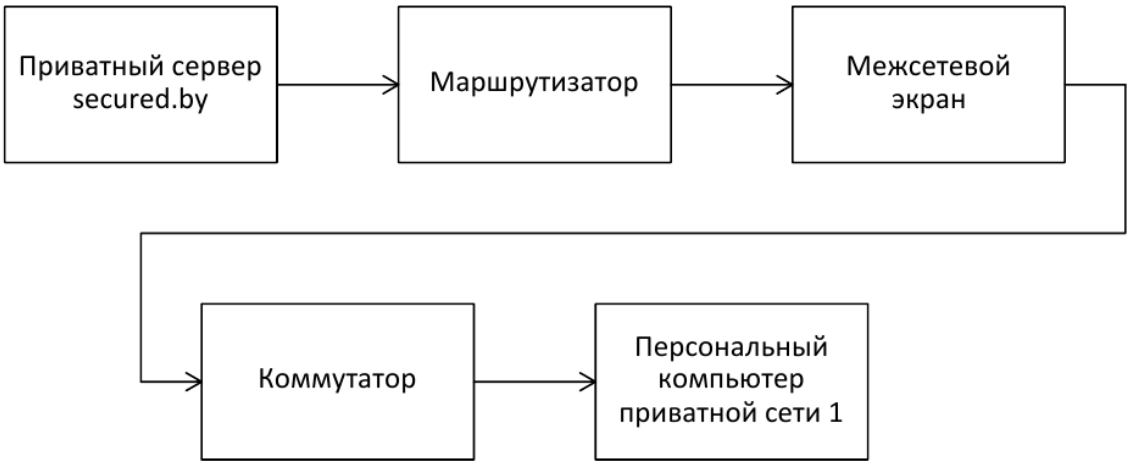

Рисунок 3.5 – Схема передачи данных в приватной сети

3.2 Выбор оборудования

В качестве маршрутизатора используется Cisco C1111-4P[16] – это современный высокопроизводительный беспроводной роутер, который входит в линейку оборудования Cisco 1100 Series Integrated Services Routers, которая ориентирована на филиалы, а также компании малого и среднего размера. Устройство имеет широкие функциональные возможности, и обеспечивает высокий уровень надёжности и сетевой безопасности.

Данный маршрутизатор обладает одним WAN портом Gigabit Ethernet и четырьмя портами LAN Gigabit Ethernet.

Коммутатор Cisco Catalyst 9200[17] – это стекируемый сетевой коммутатор корпоративного класса, предоставляющий расширенные функции безопасности, которые защищают целостность аппаратного и программного обеспечения, а также всех данных, проходящих через коммутатор.

Обладает 24 Ethernet портами с поддерживаемой скоростью до 1 Гигабит\с каждый. Таким образом, используя один коммутатор можно подключить большое число устройств к сети.

Сервер ProLiant DL180 Gen10 P35519-B21[18] – безопасный современный сервер. Отличается масштабируемостью, производительностью и надежностью, что делает его идеальной платформой для компаний, готовых к использованию локальных и гибридных облачных приложений.

В качестве однонаправленного шлюза используется СТРОМ-100[19], предназначенный для гарантированной однонаправленной передачи информации из открытых сетей в сети, в которых циркулирует информация ограниченного доступа. Помимо этого, возможно использовать диод данных

для защиты сети при передаче из нее информации в открытые сети, в том числе подключенные к сети Интернет. При соединении сетей через однонаправленный шлюз в первом случае гарантируется отсутствие утечек из конфиденциальной сети, во втором - невозможность воздействия из открытых сетей на защищаемую сеть.

Внешний интерфейс устройства Ethernet 100BASE-TX.

Внутренний интерфейс Ethernet 100BASE-FX.

4 РАЗРАБОТКА АЛГОРИТМА ПРОГРАММЫ

Для выполнения требований технического задания разработан алгоритм передачи данных из открытой сети в закрытую. Алгоритм работы программы состоит из двух блоков: приемного и передающего.

Перед началом работы вне зависимости от функционального назначения программы, происходит вызов программы инициализации, представленный на рисунке 4.1.

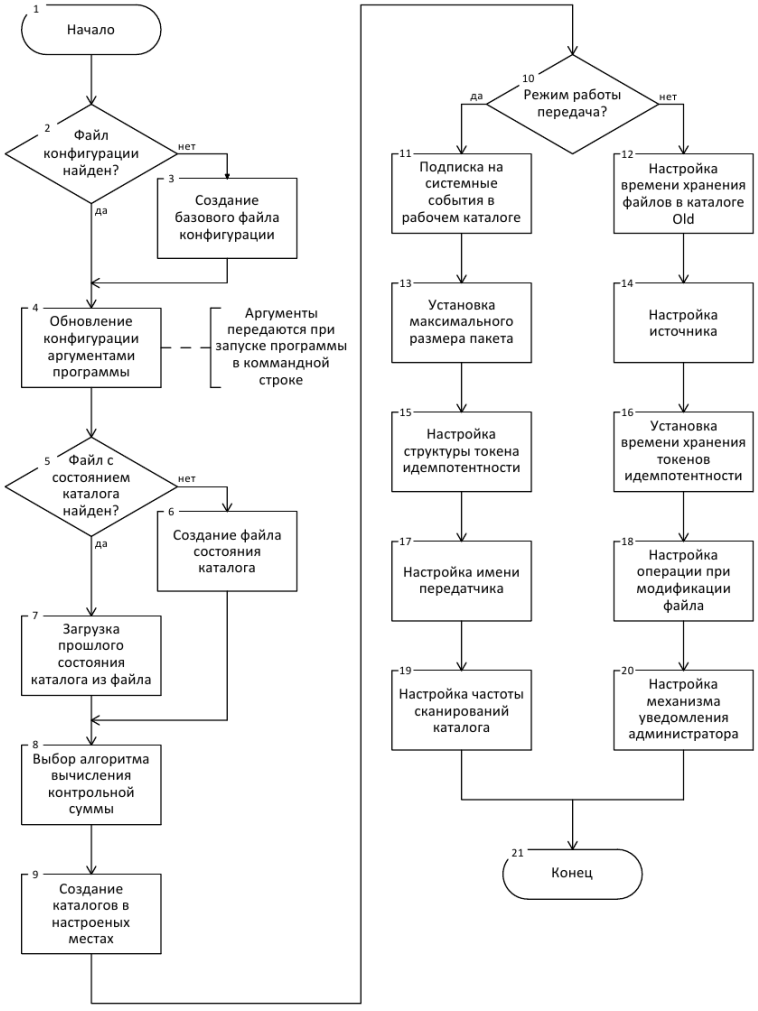

Рисунок 4.1 – Алгоритм инициализации программы

Программа пытается найти файл конфигурации, и в случае, если она его не находит, создает новый файл конфигурации в каталоге после чего заполняет его стандартной конфигурацией.

Для настройки программы можно использовать аргументы командной строки или изменять файл конфигурации. В случае использования аргументов командной строки, программа самостоятельно обновит файл конфигурации.

Общие настройки для передающего и принимающего режимов работы программы:

– выбор алгоритма вычисления контрольной суммы;

– настройка местонахождения рабочих каталогов.

Для передающего режима доступны:

– настройка адреса и порта для отправки;

– возможность подписаться на системные события в рабочем каталоге, для мгновенной обработки новых изменений;

– установка максимального размера пакета при передаче;

– настройка структуры токена идемпотентности;

– настройка имени передатчика для опознания на приемной стороне;

– настройка частоты сканирований каталога на изменения.

Для принимающего режима доступны:

– выбор операции при удалении и модификации файла;

– настройка времени хранения удаленных и измененных файлов в каталоге «Old»;

– настройка имени источника;

– установка времени хранения токенов идемпотентности;

– возможность вызова произвольной команды при наступлении ошибки для уведомления администратора.

После загрузки конфигурации происходит поиск файла с текущим состоянием каталога. В случае если он не был обнаружен, создается новый файл состояния каталога. По завершению настройки, программа приступает к работе в основном режиме.

Рассмотрим работу передающей части программы. Алгоритм ее работы приведен на рисунке 4.2.

После инициализации происходит ожидание следующего периода синхронизации, после чего, каталог сканируется на предмет изменений, путем сравнения с прошлым его состоянием.

Для этого, на каждый файл, находящийся в каталоге, создается запись с его именем, временем последнего изменения, и прочими метаданными. В случае изменения файла, он добавляется в очередь операций. По завершению сканирования начинается процесс обработки очереди операций.

При обработке каждого изменения, перед началом отправки происходит ожидание завершения работы с файлом, для предотвращения ситуации отправки неполного документа.

По завершению модификации файла пользователем, происходит его блокировка на запись, для предотвращения изменения файла в процессе отправки.

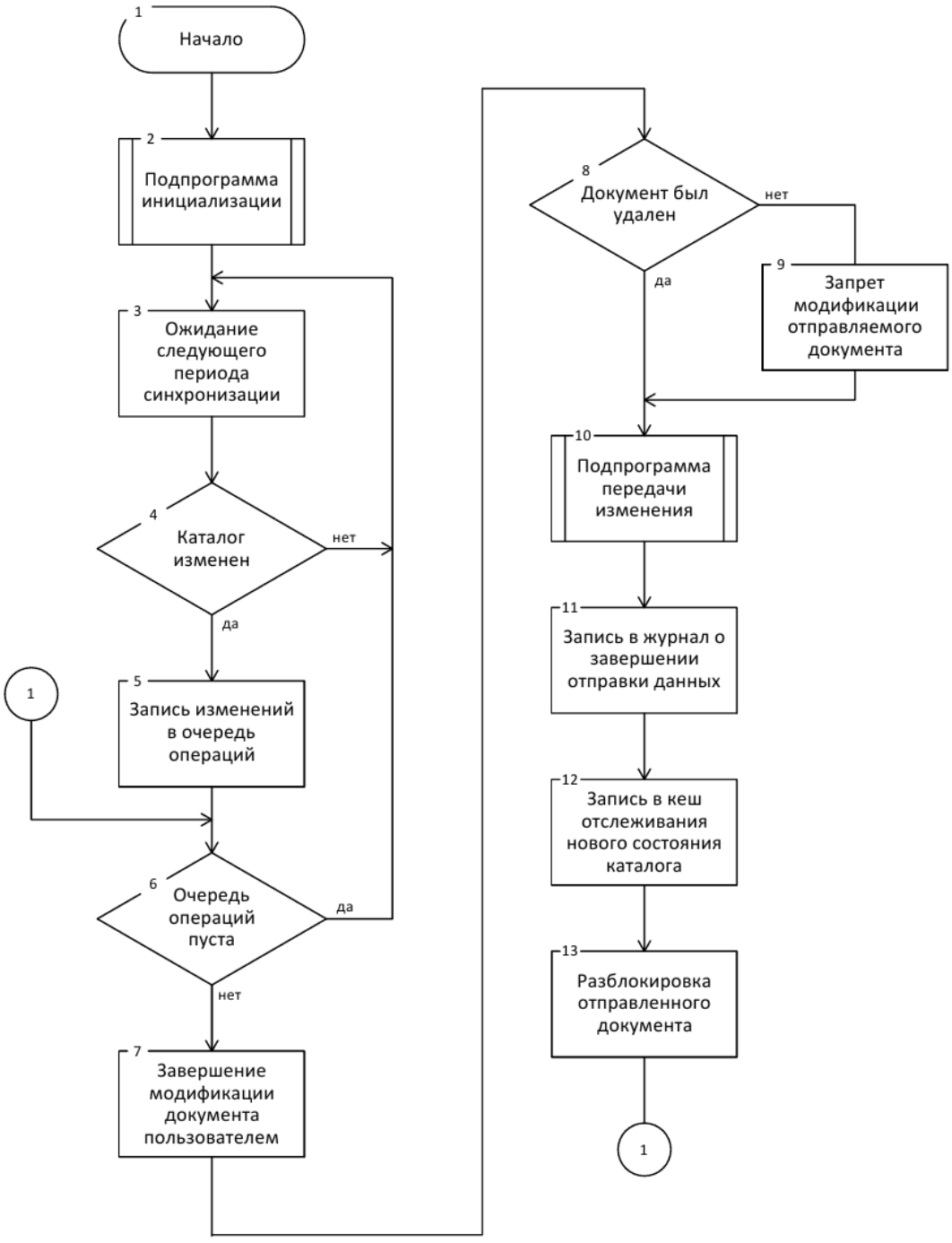

Рисунок 4.2 – Алгоритм отправки изменения

Далее происходит вызов подпрограммы отправки изменения, алгоритм которой приведен на рисунке 4.3.

Перед отправкой происходит сбор метаданных файла, подсчет его контрольной суммы, сохранение времени начала отправки файла, после чего файл разделяется на части, и они отправляются в очередь отправки.

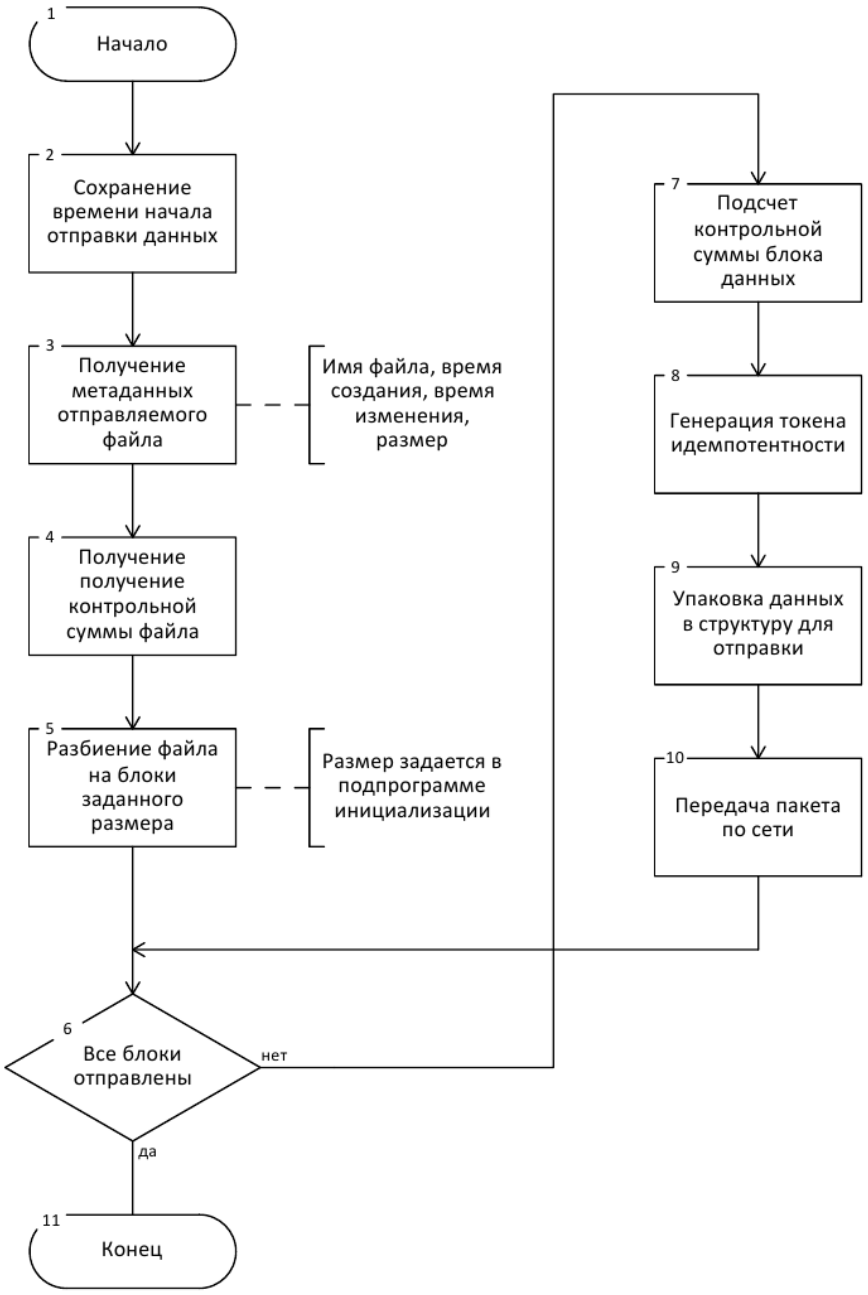

Рисунок 4.3 – Алгоритм передачи данных

Когда программа начинает обрабатывать очередь отправки, происходит подсчет контрольной суммы части файла, генерируется токен идемпотентности, происходит упаковка файла в структуру для отправки данных.

В данной структуре содержится информация:

– имя источника;

– имя файла;

– тип изменения;

– время изменения файла;
– номер блока;
– контрольная сумма файла;
– контрольная сумма блока;
– токен-идемпотентности;
– длинна данных;
– массив данных.

После упаковки происходит отправка пакета данных по сети, после чего, алгоритм проделывает тоже самое с каждым блоком файла.

По завершению отправки файла в сеть, происходит запись в журнал о завершении отправки файла, обновляется состояние каталога, снимается ограничение на модификацию с отправляемого документа.

Рассмотрим процесс приема данных. Алгоритм подпрограммы приема данных представлен на рисунке 4.4.

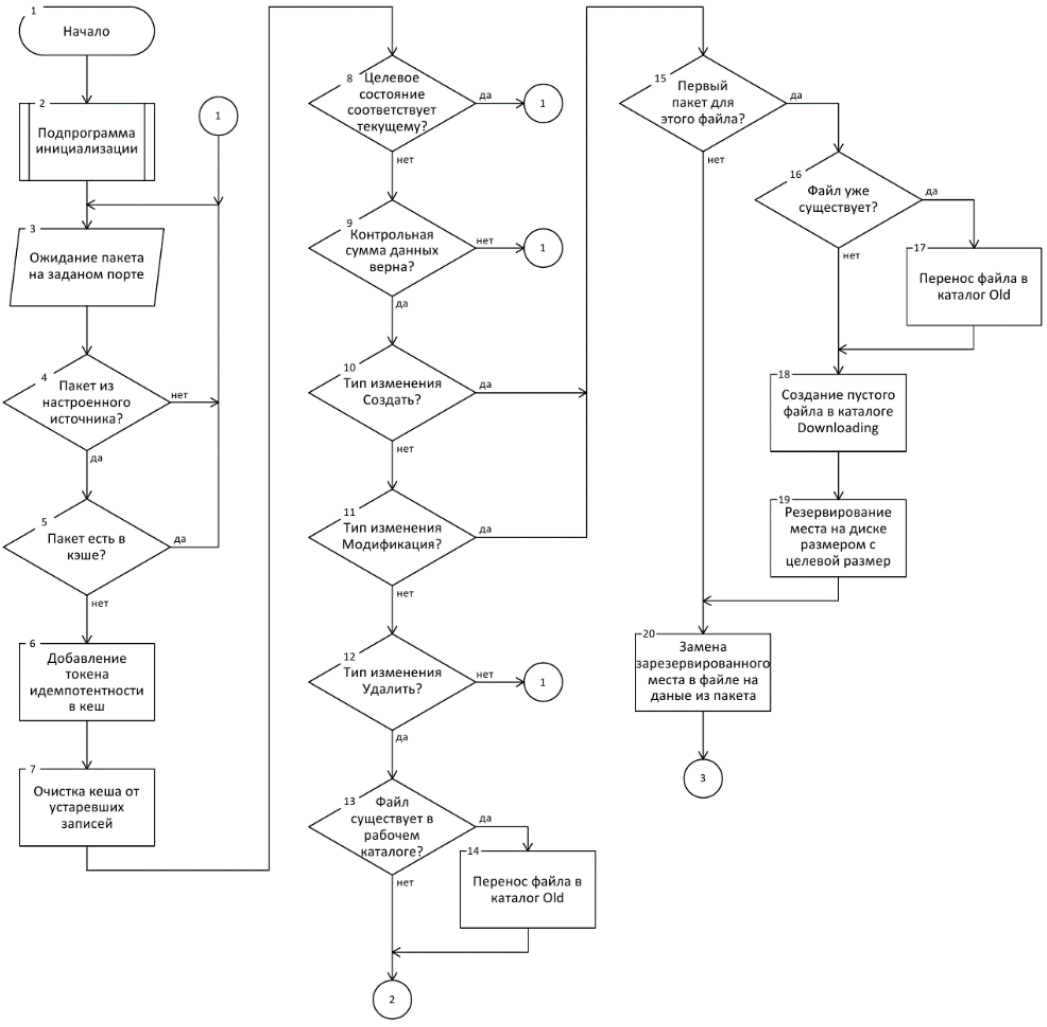

Рисунок 4.4 – Алгоритм приема данных

После инициализации, программа ожидает получение пакета на заданном порте. Когда пакет приходит, происходит проверка на соответствие

источника пакета, заданному источнику в файле конфигурации. Если имена не совпадают, пакет сбрасывается.

Если имена совпали, происходит проверка на наличие токена идемпотентности в кэше. Если токен присутствует, то значит, пакет уже обрабатывался, и его нужно сбросить.

В случае отсутствия токена в кэше, происходит его добавление, проверка кэша на наличие устаревших записей и последующая очистка.

После этого, происходит проверка, соответствует ли текущее состояние каталога, целевому. Если без обработки пакета, каталог будет соответствовать целевому, то пакет будет сброшен.

Далее проверяется контрольная сумма массива данных, если контрольная сумма не верна, пакет сбрасывается.

После этого, происходит обработка пакета в зависимости от типа изменения.

В случае если пакет имеет тип «Создать» или «Модификация» происходит проверка на то, существует ли такой файл в рабочем каталоге. Если существует происходит его перенос в каталог «Old» или удаление в зависимости от конфигурации.

Далее проверяется первый ли это пакет для файла, если да, происходит создание нового файла в каталоге «Downloading» где создается пустой файл, который записывается нулями до целевого размера.

После проверок происходит замена нулей в файле на данные из пришедшего пакета.

По завершению замены, проверяется, все ли части файла пришли, если да, происходит проверка контрольной суммы и в случае успеха, файл переносится в рабочий каталог, где с ним могут взаимодействовать пользователи приватной сети.

Текущее состояние каталога обновляется в кэше, после чего, происходит очистка каталогов «Old» и «Downloading» от устаревших записей.

После этого, алгоритм принимающей стороны возвращается к ожиданию новых пакетов.

По-особому в данной цепочке обрабатывается тип пакета «Удалить». После всех проверок, происходит перенос файла в каталог «Old» или удаление в зависимости от конфигурации.

В случае логической ошибки, например, попытка удаления отсутствующего файла или модификация еще не существующего файла, можно настроить желаемое поведение, а также уведомить администратора о случившемся.

5 РАЗРАБОТКА ПРОГРАММЫ

5.1 Установка инструментов разработки программы

Для разработки программного обеспечения выбран язык C++20. Использование современного стандарта позволяет затрачивать меньше усилий на реализацию популярного функционала, а также позволяет использовать продвинутые функции языка. При разработке современного программного обеспечения рекомендуется использовать последние версии инструментов разработки.

Язык C++ требует компилятор, стандартную библиотеку и текстовый редактор для обеспечения минимальной возможности создания программ.

Целесообразно использовать систему сборки, берущую на себя функции генерации инструкций для компилятора. На данный момент существует множество подобных систем. Одной из наиболее популярных на данный момент является система сборки CMake[20]. Инструкция по установке находится на официальном сайте программы.

Для сборки программы необходим компилятор. Так как целевая программа разрабатывается для наиболее популярной операционной системы Windows, необходимо использовать компилятор и инструменты сборки совместимые с данной операционной системой.

WinLibs[21] это оптимизированная для создания программ связка утилит целевой платформой которых является Windows. Данный набор подпрограмм содержит в себе различные компиляторы, инструменты отладки и обработчики скриптов полностью совместимые с их аналогами для других операционных систем. Подобный подход позволяет упросить процесс адаптации исходного кода программы, так как изменяется только зависимая от целевой платформы часть.

Git[22] это система контроля версий. Несмотря на то, что к разработке программы Git напрямую не относится, он является стандартным инструментом в процессе разработки, позволяя контролировать изменения программного кода.

При разработке программного кода, следует следовать стандартам оформления исходного кода, для упрощения его чтения и последующего анализа. Наиболее популярным на данный момент является «Google C++ Style Guide»[23]. В данном руководстве описаны рекомендации по названию функций и переменных, часто совершаемые ошибки при проектировании программного обеспечения и инструкции для решения возникающих проблем.

5.2 Описание библиотек

При разработке программного обеспечения целесообразно использовать готовый программный код.

Boost[23] – набор библиотек, использующих функциональность языка C++ и предоставляющих удобный кроссплатформенный высокоуровневый интерфейс для лаконичного кодирования различных повседневных подзадач программирования.

В программе используется библиотека asio из набора boost. Данная библиотека представляет функции асинхронной постановки задач, работы с сетевыми устройствами.

FMT[24] – библиотека для форматирования строк, предоставляющая удобный интерфейс при разработке программы. Используется для оформления отладочных сообщений, генерации токенов идемпотентности.

Json[25] – библиотека для работы с файлами JSON. Используется для работы с файлами конфигурации.

5.3 Настройка системы сборки

Для того что бы собрать проект средствами CMake, необходимо в корне дерева исходников разместить файл CMakeLists.txt, хранящий правила и цели сборки.

Сначала необходимо задать версию файла конфигурации, каталог с исходным кодом программы. Конфигурация проекта представлена на рисунке 5.1.

CMakeLists.txt

```
1   cmake_minimum_required(VERSION 3.14)
2   project(one-way-sync VERSION 0.1.0)
3
4   set(CMAKE_CXX_STANDARD 20)
5
6   include(FetchContent)
7
8   set (source_dir "${PROJECT_SOURCE_DIR}/src/")
9
10  file (GLOB source_files
11      "${source_dir}/*.cpp"
12      "${source_dir}/config/type/*.cpp"
13      "${source_dir}/config/*.cpp"
14      "${source_dir}/helpers/*.cpp"
15      "${source_dir}/models/*.cpp"
16      "${source_dir}/modules/*.cpp"
17      "${source_dir}/networking/*.cpp"
18  )
19
20  include_directories(
21      "${source_dir}/"
22  )
23
```

Рисунок 5.1 – Конфигурация проекта

Далее необходимо сконфигурировать библиотеки: указать каталог с исходниками, используемый стандарт языка, название переменных окружения. Конфигурация библиотек представлена на рисунке 5.2.

CMakeLists.txt

```
24  #Boost
25  set(BOOST_ROOT "C:/CLI_STUFF/mingw64/boost_1_78_0/")
26  set(CMAKE_INCLUDE_PATH ${CMAKE_INCLUDE_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/")
27  set(CMAKE_LIBRARY_PATH ${CMAKE_LIBRARY_PATH} "C:/CLI_STUFF/mingw64/boost_1_78_0/lib/")
28
29  set(Boost_USE_STATIC_LIBS ON)
30  find_package(Boost COMPONENTS system)
31
32  include_directories(${Boost_INCLUDE_DIR})
33
34  #CryptoPP
35  set(CryptoPP-header "C:/CLI_STUFF/mingw64/cryptopp860/")
36  set(CryptoPP-src "C:/CLI_STUFF/mingw64/cryptopp860/cryptopp/")
37
38  add_library(CryptoPP STATIC ${CryptoPP-src})
39  set_target_properties(CryptoPP PROPERTIES LINKER_LANGUAGE CXX)
40
41  include_directories(${CryptoPP-header})
42
43  #Threads
44  find_package(Threads REQUIRED)
45
46  #FMT
47  FetchContent_Declare(fmt
48    GIT_REPOSITORY https://github.com/fmtlib/fmt.git
49    GIT_TAG master
50  )
51  FetchContent_MakeAvailable(fmt)
52
53  #JSON-parsing
54  FetchContent_Declare(nlohmann_json
55    GIT_REPOSITORY https://github.com/nlohmann/json
56    GIT_TAG master
57  )
58  FetchContent_MakeAvailable(nlohmann_json)
```

Рисунок 5.2 – Конфигурация библиотек

Далее конфигурируются флаги компиляции, параметры выходного файла, методика подключения библиотек к итоговой программе. Конфигурация компилятора представлена на рисунке 5.3.

CMakeLists.txt

```
60  add_compile_options(-Wall -Wextra -pedantic -Werror -pthread)
61
62  add_executable(one-way-sync ${source_files})
63
64  target_link_libraries(one-way-sync ws2_32)
65  target_link_libraries(one-way-sync Threads::Threads)
66  target_link_libraries(one-way-sync CryptoPP)
67  target_link_libraries(one-way-sync fmt::fmt-header-only)
68  target_link_libraries(one-way-sync nlohmann_json::nlohmann_json)
69
```

Рисунок 5.3 – Конфигурация компилятора

При сборке программы, используется статическая линковка, что позволяет использовать программу на системах, в которых не установлены библиотеки. В ином случае, необходимо установить библиотеку на каждый компьютер с программой.

5.3 Структура программы

Программа разработана в файлах «.hpp» и «.cpp» расположенных в каталоге «src». В зависимости от функционального назначения файла, он размещается в различных каталогах. Структура каталога с исходным кодом представлена на рисунке 5.4.

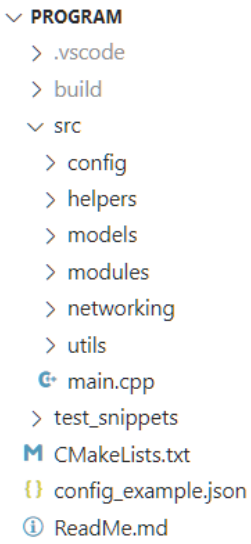

Рисунок 5.4 – Структура файлов в каталоге исходного кода программы

Каталог «config» содержит подпрограмму загрузки конфигурации, структуры содержащие настройки, используемые прочими модулями программы. Структура файлов в каталоге «config» представлена на рисунке 5.5.

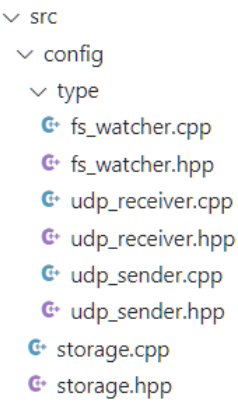

Рисунок 5.5 – Структура файлов в каталоге «config»

Каталог «helpers» содержит подпрограммы вспомогательного характера. Структура файлов в каталоге «helpers» представлена на рисунке 5.6.

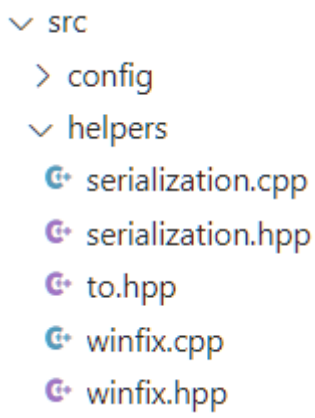

Рисунок 5.6 – Структура файлов в каталоге «helpers»

Файл «winfix.hpp» и «winfix.cpp» содержат настройку консоли Windows позволяя вводить и выводить текст на любом языке. Исходный код функции представлен на рисунке 5.7

src > helpers > winfix.cpp > ...

```cpp
#include "winfix.hpp"

#include <windows.h>
#include <winnls.h>

namespace helpers::winfix {

void FixWinConsole(){
    SetConsoleOutputCP(CP_UTF8);
}
}
```

Рисунок 5.7 – Исходный код файла «winfix.cpp»

Современные компиляторы по умолчанию работают в кодировке UTF8, однако, консоль Windows работает в режиме совместимости со старыми системами, таким образом для корректного вывода в консоль текста содержащего кириллицу необходимо передать команду в консоль.

Функция отправки команды в консоль расположена внутри специального пространства имен «helpers::winfix», что позволяет избежать неоднозначности при работе из других файлов. Каждый уровень пространства имен назван с соответствии с каталогом и названием файла.

Каталог «models» содержит описания структур данных и вспомогательных общих модулей. Структура файлов в каталоге «models» представлена на рисунке 5.8.

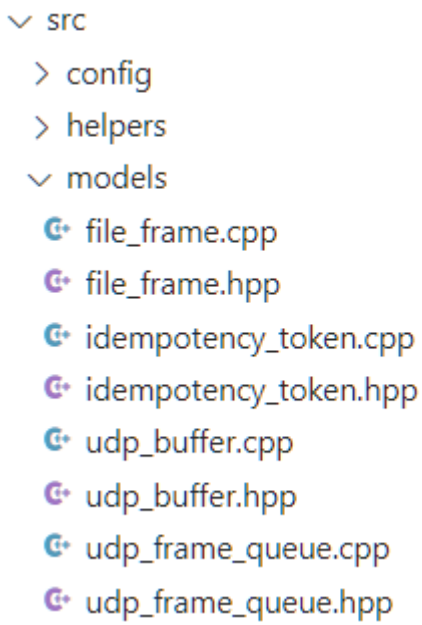

Рисунок 5.8 – Структура файлов в каталоге «models»

Каталог «modules» содержит описание обособленных функций программы. Структура файлов в каталоге «modules» представлена на рисунке 5.9.

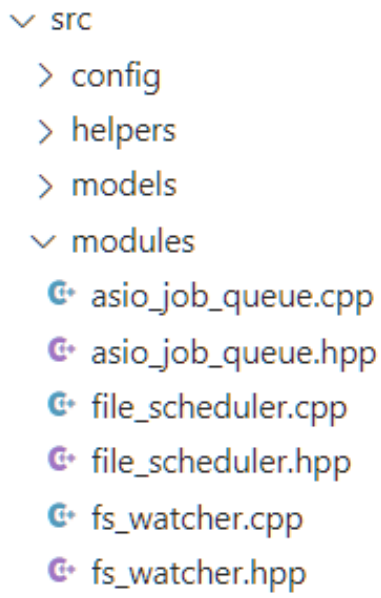

Рисунок 5.9 – Структура файлов в каталоге «modules»

Структуры из данных файлов, в отличии от файлов в каталоге «models», которые описывают структуры данных направленны на обработку и генерацию данных.

Каталог «networking» содержит модули отправки данных по сети. Структура файлов в каталоге «networking» представлена на рисунке 5.10.

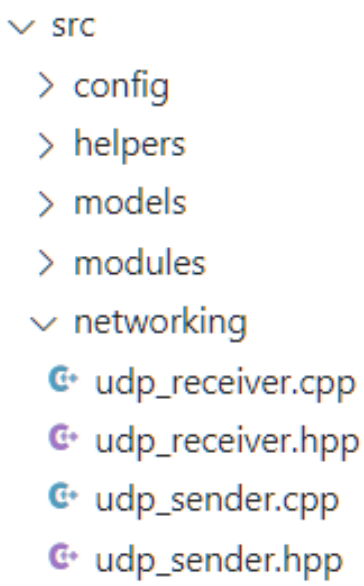

Рисунок 5.10 – Структура файлов в каталоге «networking»

Каталог «utils» содержит вспомогательные инструменты. Структура файлов в каталоге «utils» представлена на рисунке 5.11.

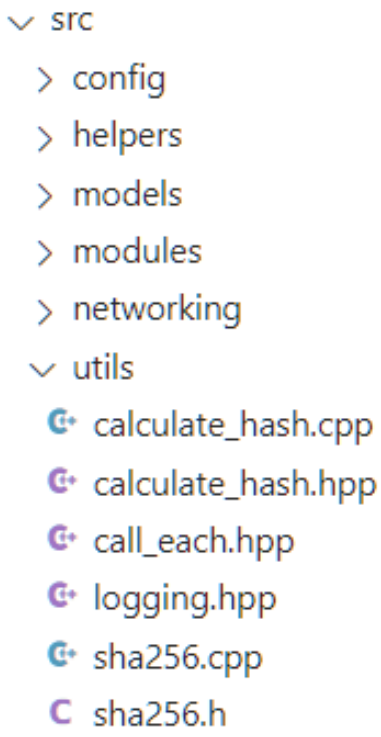

Рисунок 5.11 – Структура файлов в каталоге «networking»

5.4 Описание подпрограммы приема данных

Исходный код модуля приема данных состоит из заголовочного файла «udp_receiver.hpp» и файла реализации «udp_receiver.cpp».

Для создания объекта необходимо передать в его конструктор необходимые параметры. Для начала приема на заданном порте, необходимо вызвать метод «Start()». Объект в случае принятия сетевого пакета, проводит базовые проверки корректности пришедших данных и вызывает метод переданный в конструктор, передавая туда указатель на буфер с данными.

Структура класса модуля отправки данных представлена на рисунке 5.12.

```
16  class UdpReceiver {
17   public:
18    using HandlerType = std::function<void(models::udp_buffer::DataBuffer&)>;
19
20    UdpReceiver(const config::udp_receiver::Configuration& config,
21                const HandlerType handler)
22        : started_(false),
23          thread_count_(config.receive_threads),
24          ip_version_(config.ip_version),
25          buffer_queue_(config.buffer_count, config.datagram_max_size),
26          socket_(io_service_),
27          receiver_endpoint_(boost::asio::ip::address_v4::any(),
28                             config.target_port),
29          expected_source_ip_(config.target_ip),
30          handler_(handler){};
31
32    void Start();
33
34    ~UdpReceiver() {
35      io_service_.stop();
36      socket_.close();
37
38      for (auto& t : receive_threads_) {
39        //End threads and supress exceptions if present;
40        try {
41          t.join();
42        } catch (const std::exception&) {}
43      }
44    }
45
46   private:
47    void WaitReceive();
48    void Handle(models::udp_buffer::DataBuffer& data_buffer,
49                ip::udp::endpoint& udp_source,
50                const boost::system::error_code& error, size_t bytes_transferred);
51
52    bool started_;
53    int thread_count_;
54    ip::udp ip_version_;
55    models::udp_buffer::BufferQueue buffer_queue_;
56    boost::asio::io_service io_service_;
57    ip::udp::socket socket_{io_service_};
58    boost::asio::io_service::work work_{io_service_};
59    ip::udp::endpoint receiver_endpoint_;
60    std::optional<ip::address> expected_source_ip_;
61    std::vector<std::thread> receive_threads_;
62    HandlerType handler_;
63  };
```

Рисунок 5.12 – Листинг структуры класса «UdpReceiver»

Для модуля отправки данных в теле класса определены наследники стандартных объектов исключений для обработки различных ситуаций. Они представлены на рисунке 5.13.

```
34    class ReceiveFailed : std::runtime_error {
35      using std::runtime_error::runtime_error;
36    };
37
38    class BufferCorrupted : std::runtime_error {
39      using std::runtime_error::runtime_error;
40    };
41
42    class BufferSizeLessThanRequired : std::runtime_error {
43      using std::runtime_error::runtime_error;
44    };
```

Рисунок 5.13 – Листинг объектов исключений модуля отправки

Функция «WaitReceive()» является асинхронной функции постановки задач для потоков обрабатывающих прием. В её теле происходит получение буфера данных для последующего приема и постановка задачи принятия данных на заданном порте. Листинг функции «WaitReceive» представлен на рисунке 5.14.

```
41  void UdpReceiver::WaitReceive() {
42    auto buffer = buffer_queue_.AquireBuffer();
43    auto endpoint = std::make_unique<ip::udp::endpoint>();
44
45    socket_.async_receive_from(
46        boost::asio::buffer(buffer->buffer, buffer->buffer.size()), *endpoint,
47        [data_buffer = std::move(buffer), udp_source = std::move(endpoint), this](
48            const boost::system::error_code& error,
49            size_t bytes_transferred) mutable {
50          if (!data_buffer) {
51            throw UdpReceiver::BufferCorrupted("Data buffer ptr is NULL");
52          }
53          if (!udp_source) {
54            throw UdpReceiver::BufferCorrupted("UdpSource lost");
55          }
56
57          RAIIBufferContainer secured_buffer(std::move(data_buffer),
58                                             buffer_queue_);
59
60          Handle(*secured_buffer.container, *udp_source, error,
61                 bytes_transferred);
62        });
63  }
```

Рисунок 5.14 – Листинг функции «WaitReceive»

Для возврата буфера в очередь буферов даже в случаях обработки исключительных ситуаций и отсутствия необходимости отслеживать все дерево ветвления, используется контейнер технологии RAII (Resource Acquisition Is Initialization, Получение ресурса есть инициализация) представленный на рисунке 5.15.

```
11  struct RAIIBufferContainer {
12    RAIIBufferContainer(
13        std::unique_ptr<models::udp_buffer::DataBuffer> buffer_to_return,
14        models::udp_buffer::BufferQueue& queue_to_return)
15        : container(std::move(buffer_to_return)), queue(queue_to_return) {}
16    ~RAIIBufferContainer() { queue.ReleaseBuffer(std::move(container)); }
17    std::unique_ptr<models::udp_buffer::DataBuffer> container;
18    models::udp_buffer::BufferQueue& queue;
19  };
```

Рисунок 5.15 – Листинг защищенного контейнера

При обработке пакета, после записи данных, происходит вызов функции «Handle» которая обрабатывает ошибки и подготавливает пакет к обработке конечной функцией, заданной в переменной «handle_». Листинг функции «Handle» представлен на рисунке 5.16.

```
65  void UdpReceiver::Handle(models::udp_buffer::DataBuffer& data_buffer,
66                           ip::udp::endpoint& udp_source,
67                           const boost::system::error_code& error,
68                           size_t bytes_transferred) {
69    static const std::unordered_set<int> ignored_errors{
70        boost::asio::error::operation_aborted};
71
72    if (error) {
73      if (!ignored_errors.contains(error.value())) {
74        LOG_WARNING() << fmt::format("Receive failed: {} | Val: {} | Cat: {}\n",
75                                     error.message(), error.value(),
76                                     error.category().name());
77      }
78      return;
79    }
80
81    if (expected_source_ip_) {
82      if (expected_source_ip_ != udp_source.address()) {
83        LOG_INFO() << "Receive blocked for ip: " << udp_source.address() << "\n";
84        return;
85      }
86    }
87
88    if (data_buffer.buffer.size() < bytes_transferred) {
89      throw UdpReceiver::BufferSizeLessThanRequired("Data buffer ptr is NULL");
90    }
91    data_buffer.last_datagram_size = bytes_transferred;
92
93    try {
94      handler_(data_buffer);
95    } catch (std::runtime_error& e) {
96      LOG_INFO() << "Caught untyped exception: " << e.what() << "\n";
97    }
98  }
```

Рисунок 5.15 – Листинг защищенного контейнера

У данного объекта есть зависимость от объекта, предоставляющего буферы данных. Исходный код очереди буферов представлен на рисунке 5.4.4 и рисунке 5.16.

src > models > udp_buffer.hpp > ...

```cpp
#pragma once

#include <atomic>
#include <exception>
#include <memory>
#include <mutex>
#include <vector>

namespace models::udp_buffer {

struct DataBuffer {
  DataBuffer(int _last_datagram_size, std::vector<char>&& _buffer)
      : last_datagram_size(_last_datagram_size), buffer(_buffer) {}
  int last_datagram_size;
  std::vector<char> buffer;
};

class BufferQueue {
 public:
  class AllBuffersLocked : std::runtime_error {
    using std::runtime_error::runtime_error;
  };

  BufferQueue(int queue_size, int buffer_size);

  std::unique_ptr<DataBuffer> AquireBuffer();
  void ReleaseBuffer(std::unique_ptr<DataBuffer> buffer_to_release);

 private:
  int buffer_size_;
  std::vector<std::unique_ptr<DataBuffer>> free_buffers_;
  std::mutex mutex_;
};

}  // namespace models::udp_buffer
```

Рисунок 5.16 – Листинг заголовочного файла «udp_buffer.hpp»

Буфер реализован на базе «std::vector» содержащий в себе «std::unique_ptr». Данный подход позволяет расширять буфер данных в случае недостатка, а также избавляет от необходимости копировать данные в обработчик.

Использование «std::uniquie_ptr» позволяет гарантировать отсутствие утечек памяти в случае получения исключений или отсутствия возврата

буфера в очередь. Листинг реализации очереди буферов представлен на рисунке 5.17.

src > models > udp_buffer.cpp > ...

```cpp
#include "udp_buffer.hpp"

#include <utils/logging.hpp>

namespace models::udp_buffer {


BufferQueue::BufferQueue(int queue_size, int buffer_size) {
  buffer_size_ = buffer_size;

  free_buffers_.reserve(queue_size);

  for (int i = 0; i < queue_size; i++) {
    free_buffers_.emplace_back(
      std::make_unique<DataBuffer>(0, std::vector<char>(buffer_size_))
    );
  }
}

std::unique_ptr<DataBuffer> BufferQueue::AquireBuffer() {
  std::lock_guard lock(mutex_);

  if(free_buffers_.size() == 0){
    buffer_size_+=1;
    LOG_INFO() << "Added extra buffer, now total: " << buffer_size_ << '\n';

    free_buffers_.emplace_back(
      std::make_unique<DataBuffer>(0, std::vector<char>(buffer_size_))
    );
  }

  auto buffer = std::move(free_buffers_.back());
  free_buffers_.pop_back();

  return buffer;
}

void BufferQueue::ReleaseBuffer(std::unique_ptr<DataBuffer> buffer_to_release){
  std::lock_guard lock(mutex_);

  free_buffers_.push_back(std::move(buffer_to_release));
}

}  // namespace models::udp_buffer
```

Рисунок 5.17 – Листинг файла реализации «udp_buffer.cpp»

Буфер содержит в себе «std::mutex» являющийся инструментом синхронизации очереди. Таким образом, в случае возврата буфера в очередь одним потоком, второй поток дождется завершения модификации очереди перед получением нового буфера.

Внешний пользователь использует метод «AquireBuffer()» для получения буфера.

После приема, буфер с данными, передается в «FrameQueue». Заголовочный файл с описанием объекта, представлен на рисунке 5.18.

```
12  class FrameQueue {
13  public:
14    FrameQueue(int workers_count) : strand_(io_service_), pool_(workers_count) {}
15
16    void Add(models::udp_buffer::DataBuffer& frame_buffer);
17
18  private:
19    boost::asio::io_service io_service_;
20    boost::asio::io_service::strand strand_;
21    modules::asio_job_queue::AsioJobQueue pool_;
22    std::mutex mutex_;
23  };
```

Рисунок 5.18 – Листинг очереди фреймов «FrameQueue»

Тут полученные данные, передаются в очередь обработки, и поток приема данных освобождается, для последующего получения данных. По завершению обработки, буфер возвращается в очередь, для последующей выдачи.

Разделение программы на принимающую сетевую часть и конвейеры необходимо для предотвращения потери данных. Несмотря на то, что приемом данных занимается несколько потоков, необходимо как можно раньше их освобождать. Ситуация при котором сетевой поток ожидает записи в файл приведет к потере данных, так как запись и чтения с диска занимает значительное время. Данная очередь также защищена от гонок потоков посредством «std::mutex» и «asio::io_service».

У обработчика фреймов есть зависимость от общего компонента «AsioJobQueue», который непосредственно управляет рабочими потоками и предоставляет вычислительные мощности для решения задач. Листинг кода очереди задач приведен на рисунке 5.19.

Конструктор объекта запускает группу потоков, обрабатывающих модуль асинхронной работы io_service. Пользователь запускает задачи посредством функции «Shedule(Task task)». Тип «Task» является шаблонным типом, и к нем предъявляется требование наличия функции-оператора «void()».

Данный подход позволяет использовать как свободные функции без зависимостей, так и прокси-классы содержащие в себе изменяемые переменные.

```
class AsioJobQueue {
 public:
  AsioJobQueue(std::size_t pool_size) : work_(io_service_) {
    for (std::size_t i = 0; i < pool_size; ++i) {
      threads_.push_back(std::thread([this] { io_service_.run(); }));
    }
  }

  ~AsioJobQueue(){
    io_service_.stop();
    for (auto& t : threads_) {
      //End threads and supress exceptions if present;
      try {
        t.join();
      } catch (const std::exception&) {}
    }
  }

  template <typename Task>
  void Shedule(Task task) {
    const std::lock_guard<std::mutex> lock(mutex_);

    io_service_.post([this, &task]() { Wrapper(std::function<void()>(task)); });
  }

 private:
  void Wrapper(std::function<void()> task);

  boost::asio::io_service io_service_;
  boost::asio::io_service::work work_;
  std::vector<std::thread> threads_;
  std::mutex mutex_;
};
```

Рисунок 5.19 – Листинг класса файла «AsioJobQueue»

Так как в процессе обработки задач могут происходить исключительные ситуации используется функция-обертка, исходный код которой представлен на рисунке 5.20.

```
 void AsioJobQueue::Wrapper(std::function<void()> task) {
   try {
     task();
   } catch (const std::exception& e) {
     std::cout << "Exception caught in wrapper" << e.what() << "\n";
   } catch (const std::runtime_error& e) {
     std::cout << "Exception caught in wrapper" << e.what() << "\n";
   }
 }
```

Рисунок 5.20 – Листинг функции обертки «Wrapper»

Конфигурация модуля приема производится посредством настройки ключей в файле конфигурации. При запуске программы, конфигурация считывается в структуры, которые впоследствии передаются в объекты.

Исходный код структуры конфигурации для модуля приема представлен на рисунке 5.21.

```
10  namespace config::udp_receiver {
11
12  namespace ip = boost::asio::ip;
13
14  struct Configuration {
15    ip::udp ip_version; // udp::v4()
16    std::optional<ip::address> target_ip; //ip::address_v4::any()
17    ip::port_type target_port;
18    int datagram_max_size;
19    int buffer_count;
20    int receive_threads;
21  };
22
23  Configuration Get(const ::config::storage::Storage& storage);
24
25  Configuration Default();
26
27  }
```

Рисунок 5.21 – Листинг структуры конфигурации «udp_receiver.hpp»

Стандартная конфигурация может быть получена посредством вызова функции Default представленной на рисунке 5.22.

```
 9  Configuration Default(){
10    return {
11      ip::udp::v4(),
12      std::nullopt,
13      ip::port_type(2022),
14      1'048'576 /*datagram_max_size = 1 MiB*/,
15      8 /*buffer_count*/,
16      4 /*receive_threads*/
17    };
18  }
```

Рисунок 5.22 – Листинг структуры конфигурации «udp_receiver.hpp»

5.5 Описание подпрограммы передачи данных

Исходный код класс модуля передачи данных представлен на рисунке 5.23. Для создания объекта вызывается его конструктор, в который передается структура конфигурации. После создания объекта, для начала работы, необходимо вызвать метод «Start()».

src > networking > udp_sender.hpp > ...

```
13  class UdpSender {
14   public:
15    struct Package {
16      std::vector<char> data;
17      int retries;
18    };
19
20    class DataIncorrectState : std::runtime_error {
21      using std::runtime_error::runtime_error;
22    };
23
24    UdpSender(const config::udp_sender::Configuration& config)
25        : receiver_endpoint_(config.target_ip, config.target_port),
26          retry_count_(config.retry_count) {
27      socket_.open(config.ip_version);
28    }
29
30    void Start();
31
32    void Send(std::unique_ptr<Package> data_to_send);
33
34    ~UdpSender() {
35      io_service_.stop();
36      socket_.close();
37
38      try {
39        sender_thread_->join();
40      } catch (const std::exception&) {
41      }
42    }
43
44   private:
45    ip::udp::endpoint receiver_endpoint_;
46    boost::asio::io_service io_service_;
47    boost::asio::io_service::work work_{io_service_};
48    ip::udp::socket socket_{io_service_};
49    std::unique_ptr<std::thread> sender_thread_;
50    int retry_count_;
51  };
```

Рисунок 5.23 – Исходный код класс модуля передачи данных

Посредством передачи в метод «Send» массива данных, происходит передача на приемную часть. Поток, публикующий задачу о передаче, не задерживается в методе, а передает задачу по отправке местному потоку. Листинг реализации метода «Send» представлен на рисунке 5.24.

src > networking > udp_sender.cpp > ...

```
9   void UdpSender::Start() {
10    sender_thread_ =
11        std::make_unique<std::thread>(std::thread([this] { io_service_.run(); }));
12  }
13
14  void UdpSender::Send(std::unique_ptr<UdpSender::Package> data_to_send) {
15    if (!data_to_send) {
16      throw DataIncorrectState("Data package handles nothing");
17    }
18
19    socket_.async_send_to(
20        boost::asio::buffer(data_to_send->data), receiver_endpoint_,
21        [data_buffer = std::move(data_to_send), this](
22            const boost::system::error_code& error,
23            size_t bytes_transferred) mutable {
24          if (!error && data_buffer->data.size() == bytes_transferred) {
25            /*Send succeed*/
26            return;
27          }
28
29          LOG_WARNING() << fmt::format(
30              "Send failed: {} \ndata.size()::{} / actually_sent::{}\n",
31              error.what(), data_buffer->data.size(), bytes_transferred);
32
33          if (data_buffer->retries > retry_count_) {
34            LOG_WARNING() << "Data sent failed after retries: "
35                          << data_buffer->retries;
36            return;
37          }
38
39          data_buffer->retries += 1;
40          Send(std::move(data_buffer));
41        });
42  }
```

Рисунок 5.24 – Листинг реализации метода «Send»

Поток отправки один, однако публикацией данных могут заниматься множество потоков. Ограничение на один рабочий поток при отправке, не влияет на скорость работы, так как ограничением является работа с сетевым устройством, а не процессорные мощности.

При передаче данных, они упаковываются в специальные структуры, описанные на рисунке 5.25.

src > models > file_frame.hpp > ...

```
11  namespace models::file_frame {
12
13  using Payload = std::vector<char>;
14
15  enum class Action {
16    kCreate,
17    kModify,
18    kDelete
19  };
20
21  std::string ToString(const Action&);
22  Action Convert(const std::string&, helpers::to::To<Action>);
23
24  struct FileHeader {
25    std::string path;
26    uintmax_t size;
27    Action action;
28    std::filesystem::file_time_type last_write_time;
29    std::string sha256_hash;
30  };
31
32  struct FragmentHeader {
33    uint64_t part;
34    uint64_t parts;
35    std::string part_sha256_hash;
36    uint64_t write_position;
37  };
38
39  struct FileFragment {
40    FileHeader file_header;
41    FragmentHeader fragment_header;
42    std::string idempotency_token;
43    Payload payload;
44  };
45
46  }  // namespace models::file_frame
47
48  namespace helpers::serialization {
49
50  SerializedContainer Serialize(
51      const models::file_frame::FileFragment& to_serialize);
52
53  models::file_frame::FileFragment Deserialize(
54      SerializedContainer::const_iterator& start_position,
55      to::To<models::file_frame::FileFragment>);
56
57  }  // namespace helpers::serialization
```

Рисунок 5.25 – Листинг структур файлов используемых при передаче данных

Сериализация это процесс преобразования структуры данных в последовательный поток байт, который удобно передавать через сеть. Структура данных «FileFragment» содержит в себе подструктуры «FileHeader» и «FragmentHeader». Для этих целей необходимо произвести сериализацию структур данных. Функции прямой и обратной конвертации перечисления «Action» представлены на рисунке 5.26.

```cpp
namespace models::file_frame {

std::string ToString(const Action& action) {
  switch (action) {
    case Action::kCreate:
      return "create";
    case Action::kModify:
      return "modify";
    case Action::kDelete:
      return "delete";
  }
}
Action Convert(const std::string& from, helpers::to::To<Action>) {
  static const std::unordered_map<std::string, Action> convert{
      {"create", Action::kCreate},
      {"modify", Action::kModify},
      {"delete", Action::kDelete}};
  auto find = convert.find(from);
  if (find == convert.end()) {
    throw helpers::to::ConvertError(std::string("Can't convert Action from: ") +
                                    from);
  }
  return find->second;
}

}  // namespace models::file_frame
```

Рисунок 5.26 – Функции конвертации перечисления «Action»

При прямой сериализации данные из структур данных преобразуются в поток байт. При обратной сериализации данные извлекаются из массива байт и размещаются в структуре данных.

Данный подход позволяет применять строгую типизацию данных для работы внутри программы. Функция сериализации для заголовка «FileFragment» представлена на рисунке 5.27, десериализации на рисунке 5.28.

```cpp
SerializedContainer Serialize(
    const models::file_frame::FileHeader& to_serialize) {
  SerializedContainer out;

  auto write_time = to_serialize.last_write_time.time_since_epoch().count();

  auto path_part = Serialize(to_serialize.path);
  auto size_part = Serialize(to_serialize.size);
  auto action_part = Serialize(ToString(to_serialize.action));
  auto last_write_part = Serialize(write_time);
  auto sha_part = Serialize(to_serialize.sha256_hash);

  out.reserve(path_part.size() + size_part.size() + action_part.size() +
              last_write_part.size() + action_part.size());

  out.insert(out.end(), path_part.begin(), path_part.end());
  out.insert(out.end(), size_part.begin(), size_part.end());
  out.insert(out.end(), action_part.begin(), action_part.end());
  out.insert(out.end(), last_write_part.begin(), last_write_part.end());
  out.insert(out.end(), sha_part.begin(), sha_part.end());

  return out;
}
```

Рисунок 5.27 – Функции сериализации для заголовка «FileHeader»

```
60  models::file_frame::FileHeader Deserialize(
61      SerializedContainer::const_iterator& start_position,
62      to::To<models::file_frame::FileHeader>){
63
64    auto path = Deserialize(start_position, to::To<std::string>());
65    auto size = Deserialize(start_position, to::To<uintmax_t>());
66    auto action_str = Deserialize(start_position, to::To<std::string>());
67    auto last_write_epoch = Deserialize(start_position, to::To<int64_t>());
68    auto sha256 = Deserialize(start_position, to::To<std::string>());
69
70    std::chrono::time_point<std::chrono::system_clock, std::chrono::milliseconds>
71      last_write_tp{std::chrono::milliseconds{last_write_epoch}};
72
73    return models::file_frame::FileHeader{
74      path,
75      size,
76      Convert(action_str, helpers::to::To<models::file_frame::Action>()),
77      std::chrono::file_clock::from_sys(last_write_tp),
78      sha256
79    };
80  }
```

Рисунок 5.28 – Функции десериализации для заголовка «FileHeader»

Заголовок файла содержит информацию о размере итогового файла, типе и времени изменения, контрольную сумму файла.

Функция сериализации для заголовка «FragmentHeader» представлена на рисунке 5.29, десериализации на рисунке 5.30.

```
82  SerializedContainer Serialize(
83      const models::file_frame::FragmentHeader& to_serialize) {
84    SerializedContainer out;
85
86    auto part = Serialize(to_serialize.part);
87    auto parts = Serialize(to_serialize.parts);
88    auto part_sha256_hash = Serialize(to_serialize.part_sha256_hash);
89    auto write_position = Serialize(to_serialize.write_position);
90
91    out.reserve(part.size() + parts.size() + part_sha256_hash.size() +
92                write_position.size());
93
94    out.insert(out.end(), part.begin(), part.end());
95    out.insert(out.end(), parts.begin(), parts.end());
96    out.insert(out.end(), part_sha256_hash.begin(), part_sha256_hash.end());
97    out.insert(out.end(), write_position.begin(), write_position.end());
98
99    return out;
100 }
```

Рисунок 5.29 – Функции сериализации для заголовка «FragmentHeader»

```cpp
models::file_frame::FragmentHeader Deserialize(
    SerializedContainer::const_iterator& start_position,
    to::To<models::file_frame::FragmentHeader>){

  auto part = Deserialize(start_position, to::To<uint64_t>());
  auto parts = Deserialize(start_position, to::To<uint64_t>());
  auto part_sha256_hash = Deserialize(start_position, to::To<std::string>());
  auto write_position = Deserialize(start_position, to::To<uint64_t>());

  return models::file_frame::FragmentHeader{
    part,
    parts,
    part_sha256_hash,
    write_position
  };
}
```

Рисунок 5.30 – Функции десериализации для заголовка «FragmentHeader»

Комплексные типы, реализуются посредством комбинирования методов сериализации и десериализации базовых типов. Перед передачей данных происходит их сериализация, а после приема десериализация. Таким образом программа всегда работает с типизированными данными, что снижает сложность программы.

Исходный код шаблонных сериализаторов для различных числовых типов данных представлены на рисунке 5.31

```cpp
template<typename IntType>
SerializedContainer SerializeInt(const IntType& to_serialize) {
  SerializedContainer out;
  out.reserve(sizeof(IntType));

  for (int offset = sizeof(IntType) * 8 - 8; offset >= 0; offset -= 8) {
    out.push_back(to_serialize & (0xFFULL << (offset)));
  }
  return out;
}

template<typename IntType>
IntType DeserializeInt(SerializedContainer::const_iterator& start_position) {
  IntType out = 0;

  const auto end = start_position + sizeof(IntType);

  for (auto it = start_position; it != end; it++) {
    out = (out << 8) | *it;
  }

  start_position = end;
  return out;
}
```

Рисунок 5.31 – Функции сериализации для числовых типов

Исходный код шаблонных сериализаторов для различных контейнерных типов данных представлены на рисунке 5.32

```
32  template<typename Container>
33  SerializedContainer SerializeContainer(const Container& to_serialize) {
34    SerializedContainer container;
35    container.reserve(to_serialize.size() + kSizeBytes);
36
37    auto size_part = Serialize(to_serialize.size());
38    container.insert(container.begin(), size_part.begin(), size_part.end());
39    container.insert(container.end(), to_serialize.begin(), to_serialize.end());
40
41    return container;
42  }
43
44  template<typename Container>
45  Container DeserializeContainer(SerializedContainer::const_iterator& start_position){
46    const auto size = Deserialize(start_position, to::To<std::uint64_t>());
47    const auto string_start = start_position + kSizeBytes;
48
49    start_position = string_start + size;
50
51    Container out(string_start, string_start + size);
52    return out;
53  }
54  }
```

Рисунок 5.32 – Функции сериализации для контейнерных типов

Ключевой особенностью современного написания программ на C++ является широкое использование стандартной библиотеки. Для различных стандартных типов данных существуют функции итерирования, позволяющие получить доступ к данным непосредственно внутри контейнера. Подобный подход позволяет использовать различные алгоритмы обработки данных, которые не зависят от типа используемого контейнера.

Шаблонные функции представленные выше работают как с «std::vector» так и с «std::string», позволяя извлекать как строки так и бинарные данные.

При сериализации данных контейнерного типа, в начало данных кладется размер данных, после чего данные копируются из контейнера не сериализованных данных в буфер вывода после размера.

При десериализации метод получает итератор на неизменяемые данные, по которому он производит чтение в объект. Десериализаторы нижних уровней в результате чтения изменяют итератор, таким образом изменяя место чтения данных для последующих этапов.

Каждый десериализатор индивидуально занимается извлечением данных из контейнера. В результате их работы, данные попадают в структуру, где и используются впоследствии.

5.6 Описание подпрограммы сканирования каталога

Класс модуля отслеживания каталога представлены на рисунке 5.33. При конструировании объекта задается отслеживаемый каталог, который сканируется и записывается в кэш состояния каталога.

```
12  namespace modules::fs_watcher {
13
14  enum class FileAction { kCreated, kModified, kDeleted };
15
16  struct FileDiscription {
17    std::filesystem::path path;
18    uintmax_t size;
19    std::filesystem::file_time_type last_write_time;
20
21    bool operator==(const FileDiscription& other) const {
22      return path == other.path && size == other.size &&
23             last_write_time == other.last_write_time;
24    }
25  };
26
27  class FileWatcher {
28   public:
29    using FileUpdateHandler =
30        std::function<void(const FileDiscription&, FileAction)>;
31
32    FileWatcher(config::filesystem_watcher::Configuration& config,
33                std::optional<FileUpdateHandler> handler)
34        : folder_path_(config.folder_path), handler_(handler) {
35      namespace fs = std::filesystem;
36
37      for (auto& file : fs::recursive_directory_iterator(folder_path_)) {
38        if (fs::is_regular_file(file)) {
39          cache_[file.path().string()] =
40              std::make_unique<FileDiscription>(FileDiscription{
41                  file.path(), file.file_size(), file.last_write_time()});
42        }
43      }
44
45      if (handler) {
46        caller_ = std::make_unique<utils::call_each::CallEach>(
47            [this]() { UpdateFolder(); }, config.check_folder_each);
48      }
49    }
50
51    void UpdateFolder();
52
53    void List(std::function<void(const FileDiscription&)> info_cb);
54
55   private:
56    using FileWatcherCache =
57        std::unordered_map<std::string, std::unique_ptr<FileDiscription>>;
58    std::string folder_path_;
59    std::optional<FileUpdateHandler> handler_;
60    std::unique_ptr<utils::call_each::CallEach> caller_;
61    std::mutex mutex_;
62    FileWatcherCache cache_;
63  };
64
65  }  // namespace modules::filesystem_watcher
```

Рисунок 5.33 – Листинг класса модуля отслеживания каталога

Через интервалы заданные в файле конфигурации происходит вызов метода «UpdateFolder()». Листинг функции «UpdateFolder()» представлен на рисунке 5.34.

В процессе обработки, прошлое состояние кеша сравнивается с текущим состоянием каталога и на каждое изменение вызывается метод обратного вызова, заданный в конструкторе.

```
5   void FileWatcher::UpdateFolder() {
6     namespace fs = std::filesystem;
7
8     const std::lock_guard<std::mutex> lock(mutex_);
9
10    FileWatcherCache prev_cache_state_;
11    cache_.swap(prev_cache_state_);
12
13    for (auto& file :
14        | | std::filesystem::recursive_directory_iterator(folder_path_)) {
15      if (!fs::is_regular_file(file)) {
16        continue;
17      }
18      auto current_file_last_write_time = std::filesystem::last_write_time(file);
19      auto file_size = file.file_size();
20
21      auto prev_file_state = prev_cache_state_.find(file.path().string());
22
23      auto file_discription =
24          | | FileDiscription{file.path(), file_size, current_file_last_write_time};
25
26      if (prev_file_state == prev_cache_state_.end()) {
27        // New file
28        if (handler_) {
29          handler_.value()(file_discription, FileAction::kCreated);
30        }
31        cache_[file.path().string()] =
32            | | std::make_unique<FileDiscription>(std::move(file_discription));
33
34      } else {
35        // Exists
36        if (!(*prev_file_state->second == file_discription)) {
37          // Has changes
38          if (handler_) {
39            handler_.value()(file_discription, FileAction::kModified);
40          }
41        }
42        prev_cache_state_.erase(prev_file_state);
43      }
44
45      cache_[file.path().string()] =
46          | | std::make_unique<FileDiscription>(std::move(file_discription));
47    }
48
49    for(const auto& [path, description] : prev_cache_state_){
50      if (handler_) {
51        handler_.value()(*description, FileAction::kDeleted);
52      }
53    }
54  }
```

Рисунок 5.34 – Листинг функции «UpdateFolder()»

5.7 Вывод

Для работы однонаправленной ведомственной сети было разработано программное обеспечение на языке C++. Проведено тестирование программного обеспечения путем отправки и приема данных через loopback адрес.

В случае необходимости, программа может быть доработана с целью внедрения новых функций. Использование стандарта оформления позволяет ускорить процесс адаптации сторонних разработчиков, таким образом снижая расходы на внедрение новых функций и поддержку существующего функционала.

6 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ В ЭКСПЛУАТАЦИЮ СИСТЕМЫ ОДНОНАПРАВЛЕННОЙ ПЕРЕДАЧИ ДАННЫХ

6.1 Характеристика системы обеспечения безопасности

Однонаправленная ведомственная сеть, это программно-аппаратный комплекс, исключающий возможность утечки данных и позволяющий передавать данные из публичной сети в закрытую используя пару серверов и однонаправленный шлюз.

Данная система может опираться на уже существующую инфраструктуру ведомств или организаций, а также быть применена при первичном проектировании сети. Прежде всего данная система целесообразна для пользователей, работающих с данными, утечка которых недопустима.

Однонаправленная сеть предотвращает ряд возможных угроз информационной безопасности, делая невозможной утечку данных и делает невозможной анализ защищенной сети извне.

Система основана на взаимодействии сетевых устройств, таких как маршрутизаторы, коммутаторы, межсетевые экраны и однонаправленные шлюзы. Используется два сервера, один из которых является отправителем данных, а второй занимается приемом данных по сети. Программный комплекс занимается синхронизацией состояния каталога приватной сети изменениями из публичной. Такой подход позволяет использовать любой способ доставки данных на устройство в публичной сети.

Для применения технологии однонаправленной ведомственной сети можно задействовать силы администраторов предприятия, в котором происходит внедрение или использовать внешнего подрядчика.

После внедрения данной системы, возможность утечки данных через компьютерную сеть из закрытой сети исключена, однако, связь с внешним миром сохраняется, что приводит к исключению физического носителя из цепочек передачи информации.

Разработка системы безопасности, как правило, осуществляется сторонней организацией по заказу предприятия, на котором она будет устанавливаться. При таком подходе, учитываются пожелания заказчика по возможностям доставки данных на публичный сервер, а также происходит проектирование сети для удовлетворения нужд заказчика.

6.2 Расчет инвестиций на проектирование и внедрение в эксплуатацию системы обеспечения безопасности

Затраты на внедрение системы безопасности в эксплуатацию в значительной степени зависят от пожеланий заказчика, наличия у него оборудования, текущего состояния сети предприятия.

Целесообразно производить расчет стоимости проектирования сети и стоимости введения в эксплуатацию отдельно.

6.2.1 Расчет стоимости проектирования сети

Инвестиции в проектирование системы обеспечения безопасности включают заработную плату разработчиков, которая определяется по формуле

$$З_о = К_{пр} \cdot \sum_{i=1}^{n} З_{дi} \cdot t_i \qquad (6.1)$$

где $К_{пр}$ – коэффициент премий и иных стимулирующих выплат; n – категории исполнителей, занятых проектированием системы обеспечения безопасности; $З_{дi}$ – дневная заработная плата исполнителя i-й категории, занятого проектированием системы обеспечения безопасности, р.; t_i – трудоемкость работ по проектированию, выполняемых исполнителем.

Таблица 6.1 – Расчет основной заработной платы разработчиков

Категория исполнителя	Численность исполнителей, чел.	Месячный оклад (тарифная ставка), р.	Дневной оклад (тарифная ставка), р.	Трудоемкость, дней	Сумма, руб
1. Главный инженер	1	2520,0	120,0	8	960,0
2. Инженер-проектировщик	1	1260,0	60,0	10	600,0
Итого					1560,0
Премия и другие стимулирующие выплаты					640,0
Всего основная заработная плата разработчиков ($З_о^{пр}$)					2200,0

Данный расчет не учитывает расходы на работу административного ресурса организации занимающейся проектированием системы.

6.2.2 Расчет стоимости оборудования и материалов

В данном разделе производится расчет затрат на закупку сетевого оборудования и вспомогательных материалов. Реальные расходы зависят от наличия у заказчика своего оборудования и его совместимости с внедряемой системой.

Таблица 6.2 – Расчет затрат на материалы и оборудование

Наименование	Единица измерения	Норма расхода	Отпускная цена, руб	Сумма, руб
1. Кабели витая пара UTP Категория 5е	м	2000	0,7	1400,0
2. Кабель-канал 100х40 мм	м	200	18,0	3600,0
3. Патч-корд UTP категория 5е	шт	30	9,5	285,0
4. Коннектор RJ45	шт	5	1,6	8,0
5. Телекоммуникационный шкаф напольный	шт	1	1147,0	1147,0
6. Блок розеток	шт	2	44,31	88,62
7. Патч-панель FTP Cat 5e на 24 порта	шт	4	73,57	294,28
8. Гофрированная труба	м	6	9,19	55,14
9. Компьютерная розетка 1xRJ45	шт	16	44,22	707,52
10. Маршрутизатор CISCO C1111-4P	шт	2	4165,05	8330,1
11. Коммутатор Cisco Catalyst 9200	шт	2	2659,3	5318,6
12. Сервер HPE ProLiant DL180 Gen10 P35519-B21	шт	2	7186,0	14372,0
12. Сетевой шлюз СТРОМ-100	шт	1	1150,0	1150,0
Итого				36156,26
Транспортно-заготовительные расходы				1837,81
Всего				38594,1

6.2.3 Расчет стоимости монтажа оборудования

Заказчик может прибегнуть к использованию внешнего подрядчика, или использовать свой персонал для проведения работ по монтажу сетевого оборудования и его настройки.

Таблица 6.3 – Расчет основной заработной платы при монтаже системы

Категория исполнителя	Численность исполнителей, чел.	Месячный оклад (тарифная ставка), р.	Дневной оклад (тарифная ставка), р.	Трудоемкость, дней	Сумма, руб
1. Руководитель проекта	1	2100,0	100,0	9	900,0
2. Кабельщик	2	1050,0	50,0	9	450,0
3. Электромонтер	1	1470,0	70,0	5	350,0
2. Системный администратор	1	1680,0	80,0	5	400,0
Итого					2100,0
Премия и иные стимулирующие выплаты					840,0
Всего основная заработная плата ($З_о^м$)					2940,0

Источником данных о зарплатах сотрудников компаний, занимающихся разработкой и монтажом сетей является агрегатор объявлений rabota.by[27].

6.2.4 Сметная стоимость проектирования и монтажа однонаправленной ведомственной сети

Таблица 6.4 – Расчет основной заработной платы при монтаже системы

Показатель	Расходы
1. Затраты на материалы и оборудование	38594,1
2. Основная заработная плата, всего ($З_о$)	5140,0
В том числе: 2.1 Основная заработная плата на проектирование 2.2 Основная заработная плата на монтаж	 2100,0 2940,0
3. Дополнительная заработная плата	210,0
4. Отчисления на социальные нужды	1851,1
5. Накладные затраты	2570,0
6. Всего затрат ($З_{сб}$)	53405,2
7. Плановая прибыль ($П_п$)	21362,08
8. Сметная стоимость ($Ц_{сб}$)	74767,28

6.3 Расчет экономического эффекта от проектирования и внедрения в эксплуатацию

Экономическим эффектом для организации разработчика является прирост чистой прибыли. Исходя из специфики внедряемой системы, каждый случай внедрения является уникальным, и цена устанавливается в процессе переговоров между разработчиком и заказчиком и рассчитывается по формуле

$$\Delta П_ч = (Ц_д - З_{сб})\left(1 - \frac{Н_п}{100}\right), \quad (6.2)$$

где $Ц_д$ – договорная цена системы обеспечения безопасности (без НДС), р.; $З_{сб}$ – общая сумма затрат на проектирование и монтаж системы обеспечения безопасности организацией-разработчиком, р; $Н_п$ – ставка налога на прибыль согласно действующему законодательству, %.

В случае если договорная цена равна сумме затрат на проектирование, внедрение системы для организации разработчика будет равна

$$\Delta П_ч = (74767{,}28 - 53405{,}2)\left(1 - \frac{12}{100}\right) = 18798{,}63 \text{ р.} \quad (6.3)$$

Экономический эффект для заказчика является строго индивидуальным и может рассчитываться по различным методикам оценки рисков информационной безопасности. Внедрение системы не ведет к увеличению чистой прибыли, а только влияет на вероятность кризисной ситуации и затрат в случае реализации угрозы.

Так как внедряемая система исключает ряд угроз и сводит вероятность реализации некоторых к минимуму, заказчик может оценить потенциальную эффективность внедрения системы используя методики управления рисками. После внедрения системы, угрозы типа «утечка данных» и «несанкционированный доступ во внутреннюю сеть через сеть интернет» можно исключить из рисков информационной безопасности и рассчитать эффективность внедрения данной системы.

Так как значительная часть расходов заказчика – это закупка оборудования, стоимость внедрения значительно варьируется в зависимости от наличия существующей инфраструктуры.

6.4 Вывод

В результате обоснования разработки системы однонаправленной ведомственной сети, была получена себестоимость разработки решения для заказчика. В результате оценки расходов на заработную плату организации разработчика, получена стоимость разработки полной сети, которая составила 2200 рублей. Расходы на зарплаты сотрудников при монтаже оборудования составили 2940 рублей. Таким образом, наибольшую часть расходов составляет оборудование, покупаемое заказчиком которое составило 38594 рублей 10 копеек. Исходя из того, что организации, которые работают с данными утечку которых нельзя допустить, зачастую имеют свое оборудование и в случае, если однонаправленная сеть внедряется в уже существующую, можно избежать значительной части расходов.

Экономический эффект для организации разработчика напрямую зависит от договорной цены. Наибольшую прибыль в случаях, когда заказчик согласен с ценой по смете, приносят новые проекты, а не модернизация уже существующих.

Для организации заказчика, нет прямого экономического эффекта, так как утечка персональных данных не несет прямого ущерба. Однако, потенциальное уменьшение рисков позволяет сделать вывод о целесообразности внедрения системы.

ЗАКЛЮЧЕНИЕ

В ходе дипломной работы, был проведен анализ существующих систем однонаправленной передачи данных. Проведен патентный анализ, приведены документы с рекомендациями и принципами создания однонаправленных шлюзов из оптических сетевых карт, а также анализ рынка аппаратных диодов данных. В результате был обнаружен широкий спектр доступных на рынке решений, каждое из которых предлагало свое решение проблемы передачи данных в ситуации, когда ответа нет.

В результате анализа было принято решение о разработке универсальной системы, которая позволила бы вне зависимости от производителя аппаратного однонаправленного шлюза передавать данные в закрытую сеть. Решение с использованием синхронизируемого каталога, позволяет использовать любой метод передачи данных на сервер как в публичной сети, так и в приватной.

В разделе обоснование технических требований ведомственной сети, приведены ссылки на стандарты и принцип работы сетевых технологий в данной области. Описан принцип маршрутизации пакетов в локальных сетях.

В разделе разработка и обоснование структурной схемы проектируемой сети, приведена ведомственная сеть, состоящая из двух подсетей, разработана структурная схема проектируемой сети.

В разделе описание алгоритма программы описан принцип работы программы, даны ссылки на блок схемы алгоритмов.

В разделе экономическое обоснование разработки и внедрения в эксплуатацию системы однонаправленной передачи данных приведен расчет стоимости разработки и внедрения системы обеспечения безопасности, посчитана смета для заказчика. Дано обоснование как для организации разработчика, так и для заказчика.

В результате дипломной работы был разработан программно-аппаратный комплекс, который позволяет передавать данные из публичной сети в приватную, полностью блокируя возможность отправлять данные из приватной в публичную. Доступ к данным производится по протоколу FTP в соответствии с техническим заданием.

Список использованных источников

[1] Диод данных [Электронный ресурс] – Режим доступа: https://www.securitylab.ru/

[2] Unidirectional Networking [Электронный ресурс]: GIAC Security Essential Certification Practical Assignment Version 1.4b – Режим доступа: unidirectional-networking_2848.pdf

[3] Обзор рынка диодов данных [Электронный ресурс] – Режим доступа: https://www.anti-malware.ru/

[4] AMT InfoDiode [Электронный ресурс] – Режим доступа: https://infodiode.ru/

[5] РУБИКОН-ОШ [Электронный ресурс] – Режим доступа: https://npo-echelon.ru/

[6] Owl Cyber Defense [Электронный ресурс] – Режим доступа: https://owlcyberdefense.com/

[7] Requirements for Internet Hosts [Электронный ресурс] – Режим доступа: https://datatracker.ietf.org/doc/html/rfc1122

[8] IEEE Standard for Ethernet [Электронный ресурс] – Режим доступа: https://standards.ieee.org/ieee/802.3/7071/

[9] Internet protocol [Электронный ресурс] – Режим доступа: https://datatracker.ietf.org/doc/html/rfc791

[10] User Datagram Protocol [Электронный ресурс] – Режим доступа: https://datatracker.ietf.org/doc/html/rfc768

[11] File Transfer Protocol (FTP) [Электронный ресурс] – Режим доступа: https://datatracker.ietf.org/doc/html/rfc959

[12] Open Shortest Path First (OSPF) TCP/IP internet routing protocol [Электронный ресурс] – Режим доступа: https://www.rfc-editor.org/rfc/rfc2328

[13] IEEE 802.3u Ethernet [Электронный ресурс] https://isocpp.org/– Режим доступа: https://standards.ieee.org/ieee/802.3u/1079/

[14] IEEE 802.3ab Ethernet [Электронный ресурс] – Режим доступа: https://standards.ieee.org/ieee/802.3ab/1086/

[15] ISO/IEC 14882:2020 Programming languages – C++ [Электронный ресурс] – Режим доступа: https://www.iso.org/standard/79358.html

[16] Маршрутизатор CISCO C1111-4P [Электронный ресурс] – Режим доступа: https://www.unibelus.by/

[17] Коммутатор Cisco Catalyst 9200 (C9200L-24T-4G-A) [Электронный ресурс] – Режим доступа: https://www.unibelus.by

[18] Сервер HPE ProLiant DL180 Gen10 P35519-B21 [Электронный ресурс] – Режим доступа: https://hpserver.by/

[19] СТРОМ-100 [Электронный ресурс]. – Режим доступа: https://www.cryptoex.ru/

[20] CMake build system [Электронный ресурс]. – Режим доступа: https://cmake.org/

[21] Git [Электронный ресурс]. – Режим доступа: https://git-scm.com/

[22] WinLibs standalone build of GCC [Электронный ресурс]. – Режим доступа: https://winlibs.com/

[23] Boost provides free peer-reviewed portable C++ source libraries [Электронный ресурс]. – Режим https://www.boost.org/

[24] fmt A modern formatting library [Электронный ресурс]. – Режим доступа: https://fmt.dev/latest/index.html

[25] A modern JSON library [Электронный ресурс]. – Режим доступа: https://github.com/nlohmann/json

[26] Google C++ Style Guide [Электронный ресурс]. – Режим доступа: https://google.github.io/styleguide/cppguide.html

[27] Работа по профессиям в Минске [Электронный ресурс] – Режим доступа: https://rabota.by/