Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

Кафедра информационных и управляющих систем

Работа допущена к защите

Зав. кафедрой ИУС

\_\_\_\_\_\_\_\_И.Г. Черноруцкий

«\_\_\_\_\_»\_\_\_\_\_\_\_\_ 2015 г.

**ВЫПУСКНАЯ РАБОТА БАКАЛАВРА**

Тема: Разработка протокола канального уровня с прямым исправлением ошибок

Направление**:** 230100 – Информатика и вычислительная техника

Санкт-Петербург

2015 г.

Содержание

[Введение 3](#_Toc420865417)

[1. Анализ методов передачи данных. 7](#_Toc420865418)

[1.1 Базовая эталонная модель OSI 7](#_Toc420865419)

[1.2 Канальный уровень базовой эталонной модели OSI 10](#_Toc420865420)

[1.3 Обзор проколов канального уровня 12](#_Toc420865421)

[1.3.1 Высокоуровневый протокол управления каналом(HDLC) 12](#_Toc420865422)

[1.3.2 Протокол стандарта IEEE 802.3 15](#_Toc420865423)

[Семейство протоколов стандарта IEEE 802.3 – это самое распространенное семейство протоколов в данный момент. 15](#_Toc420865424)

[В первых версиях стандарта указано, что в качестве передающий среды указан коаксиальный кабель, в дальнейшем появилась возможность использовать витую пару и оптический кабель. Таким образом данный стандарт предлагает крайне надежную физическую среду передачи данных. 15](#_Toc420865425)

[1.3.5 Постановка задачи 16](#_Toc420865426)

[2. Исследование существующих методов исправления ошибок 17](#_Toc420865427)

[**2.1. Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды)** 17](#_Toc420865428)

[**2.2. Коды Соломона Рида** 19](#_Toc420865429)

[**2.3.** **Набор исследуемых кодов** 20](#_Toc420865430)

[**2.4.** **Обеспечение требуемой вероятности приема** 20](#_Toc420865431)

[**2.5.** **Минимизация времени задержки приема** 20](#_Toc420865432)

[3. Разработка и программная реализация канальных протоколов 21](#_Toc420865433)

[3.1. Обобщённая структура кадра 21](#_Toc420865434)

[3.2. Выбор способа реализации канальных протоколов для исследования. 21](#_Toc420865435)

[3.3 Алгоритм оценочного моделирования 23](#_Toc420865436)

[3.4 Реализация инструмента оценочного моделирования 24](#_Toc420865437)

[3.4 Проверка инструмента оценочного моделирования. 25](#_Toc420865438)

[4. Разработка архитектуры и реализация протокола передачи данных с исправлением данных. 26](#_Toc420865439)

[**3.1. Модель p a** 26](#_Toc420865440)

[**3.3. Двоичный симметричный канал** 29](#_Toc420865441)

[**3.4. Параметры исследования протоколов канального уровня** 31](#_Toc420865442)

[5. Анализ результатов. 32](#_Toc420865443)

[5. Заключение 34](#_Toc420865444)

[6. Список литературы 35](#_Toc420865445)

[**7. Приложение 1 Исходный код** 36](#_Toc420865446)

[**Приложение 2 Файл результатов** 46](#_Toc420865447)

[**Приложение 3 Интерфейс программы** 47](#_Toc420865448)

# Введение

Каналом передачи информации (каналом связи)называют совокупность устройств, обеспечивающих передачу сигналов с определёнными свойствами с одного пункта к другому. При построении системы канал, как правило, является заданным звеном, с которым источники и получатели должны быть согласованы посредством передатчиков и приёмников.

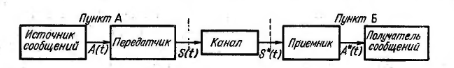


Рис. 1: система передачи информации от одного источника к одному получателю по одному каналу

A(t) – сообщение поступающее от источника.

S(t) – сигнал полученный после преобразования сообщения A(t) передатчиком.

S\*(t) – сигнал полученный на выходе канала (выходной или принимаемый).

A\*(t) – сообщение преобразованное из сигнала S\*(t) приёмником.

Каналы связи характеризуются по различным признакам:

* По используемым линиям связи:
  + Кабельные;
  + Радиорелейные;
  + Тропосферные;
  + И др.
* По полосе частот сигнала в линии:
  + Тональные;
  + Высокочастотные;
  + Коротковолновые;
  + Световые;
  + И др.
* По техническому характеру сигналов и назначению систем связи:
  + Телефонные;
  + Звукового вещания;
  + Телевизионные;
  + Телеграфные;
  + Передачи цифровой информации;
  + И др.

При математическом описании каналы различают, в первую очередь, по типу множеств которым принадлежат входные и выходные сигналы (непрерывные и дискретные), и по характеру изменения этих сигналов во времени (каналы непрерывного и дискретного времени).

Среди непрерывных каналов непрерывного времени можно выделить два наиболее типичных:

1. Непосредственно линия связи, часто с усилительным или переприёмным (ретрансляционным) оборудованием. Такой канал входит в состав всех других каналов связи.
2. Отличается от первого наличием модуляционного и демодуляционного оборудования (часто многоступенного) и полосой пропускания. Примерами таких каналов могут служить телевизионные каналы, тракты систем высокочастотного телефонирования и т.п.

Дискретный канал чаще всего дискретен и по времени. Он состоит из непрерывного канала и подключенных к нему формирователя сигналов (дискретного модулятора) и решающего устройства. Последние и обуславливают дискретность множества входных и выходных элементарных сигналов. В идеальном случае эти два устройства действуют синхронно.

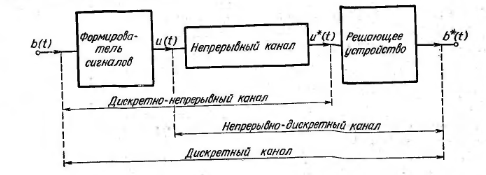


Рис.2: Дискретный канал

Дискретный канал связи предназначен для передачи дискретных сигналов (символов). При передаче по такому каналу сообщение S(t) представляется некоторой последовательностью элементарных дискретных сообщений X(t), принадлежащих конечному множеству. В результате кодирования последовательность X(t) заменяется другой последовательностью Y(t), которая ставится в соответствие E(t). Последовательность Y(t) подаётся на вход дискретного канала.

Основной характеристикой дискретного канала является вероятность того или иного изменения символа на данной позиции. Эта характеристика определяется теми изменениями, которые претерпевает символ при передаче по каналу:

* Смещение во времени;
* Отличие на некоторых позициях выходных символов от входных(аддитивные ошибки);
* Смещение номеров позиций выходной последовательности символов от позиций входной последовательности символов (ошибки синхронизации);
* Появление на некоторых ошибках символов стирания.

При действии рассмотренных факторов основная характеристика дискретного канала зависит от номера позиции, от значения, передаваемого и всех ранее переданных символов.

Поток ошибок представляет собой дискретный случайный процесс E. Каждая позиция E по определённому правилу складывается соответствующей позицией Y (сообщение на выходе).

Представляется логичным и достаточно удобным рассматривать поток ошибок дискретного канала связи как ступенчатый случайный процесс. Такой подход позволяет при исследовании каналов связи использовать многочисленные важные результаты, полученные для случайных процессов.

# Анализ методов передачи данных.

# Базовая эталонная модель OSI

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель OSI | | | | |
| Уровень | | Тип данных | Функции | Примеры |
| Host layers | Прикладной (applicaiton) | Данные | Доступ к сетевым службам | HTTP, FTP, SMTP |
| Представительский(представления) (presentation) | Представление и шифрование данных | ASCII, EBCDIC, JPEG |
| Сеансовый(session) | Управление сеансом связи | RPC, PAP |
| Транспортный(transport) | Сегменты | Прямая связь между конечными пунктами и надежность | TCP, UPD |
| Media layers | Сетевой(network) | Пакеты, дейтаграммы | Определение маршрута и логическая адресация | [IPv4](https://ru.wikipedia.org/wiki/IPv4), IPv6, IPsec, [AppleTalk](https://ru.wikipedia.org/wiki/AppleTalk) |
| Канальный(data link) | Биты, Кадры | Физическая адресация | [PPP](https://ru.wikipedia.org/wiki/Point-to-Point_Protocol), 802.2, L, [ARP](https://ru.wikipedia.org/wiki/ARP) |
| Физический(physical) | Биты | Работа со средой передачи, сигналами и двоичными данными | DSL, USB |

Сетевая модель OSI(англ. *open systems interconnection basic reference model*— базоваяэталонная модельвзаимодействия открытых систем, сокр.ЭМВОС; 1978 год) —сетевая модельстекасетевых протоколовOSI/ISO (ГОСТ Р ИСО/МЭК 7498-1-99).

Любой протокол модели OSI должен взаимодействовать либо с протоколами своего уровня, либо с протоколами на единицу выше и/или ниже своего уровня. Взаимодействия с протоколами своего уровня называются горизонтальными, а с уровнями на единицу выше или ниже — вертикальными. Любой протокол модели OSI может выполнять только функции своего уровня и не может выполнять функций другого уровня, что не выполняется в протоколах альтернативных моделей.

Каждому уровню с некоторой долей условности соответствует свой операнд — логически неделимый элемент данных, которым на отдельном уровне можно оперировать в рамках модели и используемых протоколов: на физическом уровне мельчайшая единица — бит, на канальном уровне информация объединена в кадры, на сетевом — в пакеты (датаграммы), на транспортном — в сегменты. Любой фрагмент данных, логически объединённых для передачи — кадр, пакет, датаграмма — считается сообщением. Именно сообщения в общем виде являются операндами сеансового, представительского и прикладного уровней. К базовым сетевым технологиям относятся физический и канальный уровни.

Прикладной уровень (уровень приложений; англ. *application layer*) — верхний уровень модели, обеспечивающий взаимодействие пользовательских приложений с сетью:

* позволяет приложениям использовать сетевые службы:
  + удалённый доступ к файлам и базам данных,
  + пересылка электронной почты;
* отвечает за передачу служебной информации;
* предоставляет приложениям информацию об ошибках;
* формирует запросы к уровню представления.

Представительский уровень (уровень представления; англ. *presentation layer*) обеспечивает преобразование протоколов и кодирование/декодирование данных. Запросы приложений, полученные с прикладного уровня, на уровне представления преобразуются в формат для передачи по сети, а полученные из сети данные преобразуются в формат приложений. На этом уровне может осуществляться сжатие/распаковка или шифрование/дешифрование, а также перенаправление запросов другому сетевому ресурсу, если они не могут быть обработаны локально. Обычно представляет собой промежуточный протокол для преобразования информации из соседних уровней. Это позволяет осуществлять обмен между приложениями на разнородных компьютерных системах прозрачным для приложений образом. Уровень представлений обеспечивает форматирование и преобразование кода. Форматирование кода используется для того, чтобы гарантировать приложению поступление информации для обработки, которая имела бы для него смысл. При необходимости этот уровень может выполнять перевод из одного формата данных в другой. Уровень представлений имеет дело не только с форматами и представлением данных, он также занимается структурами данных, которые используются программами. Таким образом, уровень 6 обеспечивает организацию данных при их пересылке.

Сеансовый уровень (англ. *session layer*) модели обеспечивает поддержание сеанса связи, позволяя приложениям взаимодействовать между собой длительное время. Уровень управляет созданием/завершением сеанса, обменом информацией, синхронизацией задач, определением права на передачу данных и поддержанием сеанса в периоды неактивности приложений.

Транспортный уровень (англ. *transport layer*) модели предназначен для обеспечения надёжной передачи данных от отправителя к получателю. При этом уровень надёжности может варьироваться в широких пределах. Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции (например, функции передачи данных без подтверждения приема), и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности, мультиплексируют несколько потоков данных, обеспечивают механизм управления потоками данных и гарантируют достоверность принятых данных. Например, UDP ограничивается контролем целостности данных в рамках одной датаграммы, и не исключает возможности потери пакета целиком, или дублирования пакетов, нарушение порядка получения пакетов данных; TCP обеспечивает надёжную непрерывную передачу данных, исключающую потерю данных или нарушение порядка их поступления или дублирования, может перераспределять данные, разбивая большие порции данных на фрагменты и наоборот склеивая фрагменты в один пакет.

Сетевой уровень (англ. *network layer*) модели предназначен для определения пути передачи данных. Отвечает за трансляцию логических адресов и имён в физические, определение кратчайших маршрутов, коммутацию и маршрутизацию, отслеживание неполадок и «заторов» в сети. Протоколы сетевого уровня маршрутизируют данные от источника к получателю. Работающие на этом уровне устройства (маршрутизаторы) условно называют устройствами третьего уровня (по номеру уровня в модели OSI).

Канальный уровень (англ. *data link layer*) предназначен для обеспечения взаимодействия сетей на физическом уровне и контроля за ошибками, которые могут возникнуть. Полученные с физического уровня данные, представленные в битах, он упаковывает в кадры, проверяет их на целостность и, если нужно, исправляет ошибки (формирует повторный запрос поврежденного кадра) и отправляет на сетевой уровень. Канальный уровень может взаимодействовать с одним или несколькими физическими уровнями, контролируя и управляя этим взаимодействием.

Физический уровень (англ. *physical layer*) — нижний уровень модели, который определяет метод передачи данных, представленных в двоичном виде, от одного устройства (компьютера) к другому. Составлением таких методов занимаются разные организации, в том числе: Институт инженеров по электротехнике и электронике, Альянс электронной промышленности, Европейский институт телекоммуникационных стандартов и другие. Осуществляют передачу электрических или оптических сигналов в кабель или в радиоэфир и, соответственно, их приём и преобразование в биты данных в соответствии с методами кодирования цифровых сигналов.

# Канальный уровень базовой эталонной модели OSI

Канальный уровень (англ. *Data Link layer*) — второй уровень сетевой модели OSI, предназначенный для передачи данных узлам, находящимся в том же сегменте локальной сети. Также может использоваться для обнаружения и, возможно, исправления ошибок, возникших на физическом уровне. Примерами протоколов, работающих на канальном уровне, являются: Ethernet для локальных сетей (много узловой), Point-to-Point Protocol (PPP), HDLC и ADCCP для подключений точка-точка (двух узловой).

Канальный уровень отвечает за доставку кадров между устройствами, подключенными к одному сетевому сегменту. Кадры канального уровня не пересекают границ сетевого сегмента. Функции межсетевой маршрутизации и глобальной адресации осуществляются на более высоких уровнях модели OSI, что позволяет протоколам канального уровня сосредоточиться на локальной доставке и адресации.

Заголовок кадра содержит аппаратные адреса отправителя и получателя, что позволяет определить, какое устройство отправило кадр и какое устройство должно получить и обработать его. В отличие от иерархических и маршрутизируемых адресов, аппаратные адреса одноуровневые. Это означает, что никакая часть адреса не может указывать на принадлежность к какой-либо логической или физической группе. Когда устройства пытаются использовать среду одновременно, возникают коллизии кадров. Протоколы канального уровня выявляют такие случаи и обеспечивают механизмы для уменьшения их количества или же их предотвращения.

Многие протоколы канального уровня не имеют подтверждения о приёме кадра, некоторые протоколы даже не имеют контрольной суммы для проверкицелостности кадра. В таких случаях протоколы более высокого уровня должны обеспечивать управление потоком данных, контроль ошибок, подтверждение доставки и ретрансляции утерянных данных.

В программировании доступ к этому уровню предоставляет драйвер сетевой платы. В операционных системах имеется программный интерфейс взаимодействия канального и сетевого уровней между собой, это не новый уровень, а просто реализация модели для конкретной ОС. Примеры таких интерфейсов: ODI, NDIS.

Длина пакета, формируемого протоколом канального уровня, ограничена сверху посредством MTU. MTU может быть изменено. Минимальная длина кадра прописывается в стандартах и не может быть изменена.

Спецификация IEEE 802 разделяет этот уровень на 2 подуровня. MAC (Media Access Control) регулирует доступ к разделяемой физической среде, LLC (Logical Link Control) обеспечивает обслуживание сетевого уровня.

Функции канального уровня:

1. Получение доступа к среде передачи. Обеспечение доступа — важнейшая функция канального уровня. Она требуется всегда, за исключением случаев, когда реализована полно связная топология (например, два компьютера, соединенных через кроссовер, или компьютер со свичом в полнодуплексном режиме).
2. Выделение границ кадра. Эта задача также решается всегда. Среди возможных решений этой задачи — резервирование некоторой последовательности, обозначающей начало или конец кадра.
3. Аппаратная адресация (или адресация канального уровня). Требуется в том случае, когда кадр могут получить сразу несколько адресатов. В локальных сетях аппаратные адреса (MAC-адреса) применяются всегда.
4. Обеспечение достоверности принимаемых данных. Во время передачи кадра есть вероятность, что данные исказятся. Важно это обнаружить и не пытаться обработать кадр, содержащий ошибку. Обычно на канальном уровне используются алгоритмы контрольных сумм, дающие высокую гарантию обнаружения ошибок.
5. Адресация протокола верхнего уровня. В процессе декапсуляции указание формата, вложенного PDU существенно упрощает обработку информации, поэтому чаще всего указывается протокол, находящийся в поле данных, за исключением тех случаев, когда в поле данных может находиться один-единственный протокол.

# Обзор проколов канального уровня

# Высокоуровневый протокол управления каналом(HDLC)

Во всех бит-ориентированных протоколах используется формат кадра, показанный на рис.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 8 | 8 | 8 | 10 | 16 | 8 |
| 0 1 1 1 1 1 1 0 | Адрес | Управляющее поле | Данные | Контрольная  сумма | 0 1 1 1 1 1 1 0 |

Поле *Address* (адрес) чрезвычайно важно для линий с несколькими терминалами, где оно используется для идентификации одного из терминалов. В двухточечных сетях это поле иногда используется, чтобы отличать команды от ответов.

Поле *Control* (управляющей информации) используется для хранения порядковых номеров, подтверждений и других служебных данных.

Поле Data(данные) может содержать произвольную информацию. Оно может быть любой длины, хотя эффективность контрольной суммы снижается с увеличением длины кадра из-за увеличения вероятности многочисленных пакетов ошибок.

Поле Checksum(контрольная сумма) является разновидностью циклического избыточного кода.

В качестве заголовка и концевика кадра используется флаговый байт (01111110).

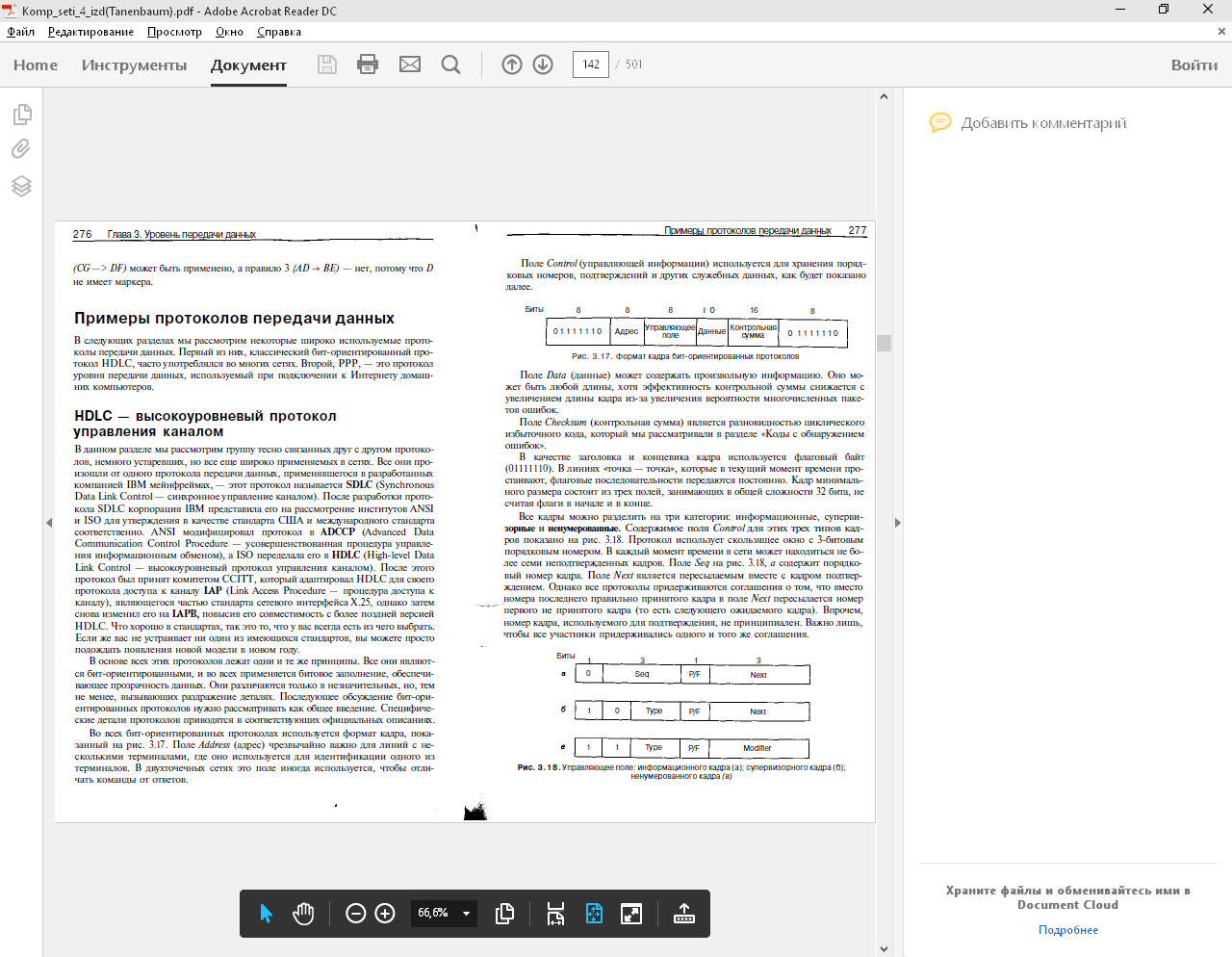
В линиях ≪точка — точка≫, которые в текущий момент времени проcтаивают, флаговые последовательности передаются постоянно. Кадр минимального размера состоит из трех полей, занимающих в общей сложности 32 бита, не считая флаги в начале и в конце.

Все кадры можно разделить на три категории: информационные, супервизорные и ненумерованные**.**

Содержимое поля *Control* для этих трех типов кадров показано на рис. 3.18. Протокол использует скользящее окно с 3-битовым порядковым номером. В каждый момент времени в сети может находиться не более семи неподтвержденных кадров. Поле *Seq* на рис. 3.18, *а* содержит порядковый номер кадра. Поле *Next* является пересылаемым вместе с кадром подтверждением. Однако все протоколы придерживаются соглашения о том, что вместо номера последнего правильно принятого кадра в поле *Next* пересылается номер

первого не принятого кадра (то есть следующего ожидаемого кадра). Впрочем,

номер кадра, используемого для подтверждения, не принципиален. Важно лишь,

чтобы все участники придерживались одного и того же соглашения. 

**Рис. . Управляющее поле: информационного кадра (а); супервизорного кадра (б);**

**ненумерованного кадра *(в)***

Бит *P/F* означает *Poll/Final* (Опрос/Финальный). Он используется, когда компьютер (или концентратор) опрашивает группу терминалов. В случае значения *Р* компьютер предлагает терминалу посылать данные. Во всех кадрах, кроме последнего, посылаемых терминалом, бит *P/F* устанавливается в Р. В последнем кадре этот бит устанавливается в *F.*

Некоторые протоколы используют бит *P/F,* чтобы заставить другую машину послать супервизорный кадр немедленно, не ожидая попутного потока данных. Этот бит также изредка используется в ненумерованных кадрах. Тип супервизорного кадра указывается с помощью значения поля *Туре.*

Если *Туре* = 0, значит, данный кадр является подтверждением. Он официально называется RECEIVE READY (к приему готов). Такой кадр сообщает номер следующего ожидаемого кадра и применяется при отсутствии попутного потока данных для передачи подтверждения.

*Туре* = 1 является признаком отрицательного подтверждения, официально называющегося REJECT (отказ). Он применяется для сообщения об обнаружении ошибки передачи. Поле *Next* в этом случае содержит номер первого неверно полученного кадра (то есть первого кадра, который следует переслать повторно). Отправитель должен переслать повторно все неподтвержденные кадры, начиная

с кадра с номером *Next.*

*Туре -* 2 означает RECEIVE NOT READY (к приему не готов). При этом, как и в случае RECEIVE READY, подтверждается прием всех кадров вплоть до *Next-i,* однако отправителю сообщается, что передачу следует приостановить. Сигнал неготовности к приему предназначен не для использования в качестве альтернативы схеме скользящих окон, а для обозначения наличия у получателя каких-либо временных проблем, например отсутствия свободной памяти в буферах. Когда получатель сможет продолжить работу, он пошлет сигнал готовности, отказа или другой управляющий кадр.

*Туре =* 3 означает SELECTIVE REJECT (выборочный отказ). Такой байт представ-

ляет собой запрос повторной передачи только указанных кадров. Наиболее полезен, когда

размер окна отправителя не превышает половины количества используемых порядковых номеров. Таким образом, если получатель хочет сохранить в буфере несвоевременные кадры для последующего использования, он может запросить повторную передачу любого кадра с помощью SELECTIVE REJECT.

Протоколы HDLC и ADCCP поддерживают этот тип кадра, а протоколы SDLC и LAPB — нет, то есть в этих протоколах нет команды выборочного отказа, а кадры типа 3 не используются.

Третий класс кадров составляют ненумерованные кадры. Иногда они применяются для служебных целей, но могут переносить и данные, когда требуется ненадежный, не требующий соединения сервис. В отличие от предыдущих двух классов, в которых различные бит-ориентированные протоколы были почти идентичными, в вопросе использования ненумерованных кадров они очень сильно различаются. Для обозначения типа кадра зарезервировано 5 бит, однако используются значительно меньше, чем 32 возможных комбинации.

Все протоколы поддерживают команду DISC (DISConnect — прервать связь),

позволяющую предупредить, что машина скоро будет выключена (например, для профилактического обслуживания). Также имеется команда, позволяющая машине, только что вернувшейся в подключенный режим (on-line), заявить о своем присутствии и принудительно обнулить все порядковые номера.

Эта команда называется SNRM (Set Normal Response Mode — установить нормальный режим ответа). К сожалению, этот ≪нормальный режим≫ является чем угодно, но не нормой. Это несбалансированный (то есть асимметричный) режим, при котором один конец линии является ведущим (master), а другой — ведомым (slave). Команда SNRM появилась еще в те времена, когда обмен данными означал общение примитивного терминала с компьютером, которое, конечно, было асимметричным. Чтобы лучше учитывать ситуацию равноправных партнеров, в протоколы HDLC и LAPB была добавлена команда SABM (Set Asynchronous Balanced Mode — установить асинхронный сбалансированный режим), которая инициализирует линию и объявляет равенство сторон. Кроме того, в этих протоколах имеются дополнительные команды SABME и SNRME, которые отличаются от SABM и SNRM только тем, что вводят расширенный формат кадров с 7-битовым порядковым номером вместо 3-битового.

Третьей командой, поддерживаемой всеми этими протоколами, является FRMR

(FRaMe Reject — отклонить кадр), применяющаяся, когда приходит кадр с верной контрольной суммой, но недопустимой семантикой. Например, супервизорный кадр типа 3 в протоколе LAPB, кадр длиной менее 32 бит, недопустимый управляющий кадр или подтверждение кадра, находящегося вне пределов окна и т. д. Данные включают управляющее поле неправильного кадра, параметры окна и набор битов, указывающих тип ошибки.

Управляющие кадры могут быть повреждены или потеряны так же, как и информационные кадры, поэтому им также нужны подтверждения. Для этой цели предназначен специальный служебный кадр, называемый UA (Unnumbered Acknowledgement — ненумерованное подтверждение). Поскольку неподтвержденным может быть только один управляющий кадр, то не возникает вопроса о том, какой именно служебный кадр подтверждается.

Остальные управляющие кадры занимаются инициализацией, опросом и сообщением состояния. Есть также управляющий кадр, который может содержать произвольную информацию, UI (Unnumbered Information). Эта информация не передается на сетевой уровень, но получается и обрабатывается самим уровнем передачи данных.

# 1.3.5 Постановка задачи

В данной работе основной задачей является разработка протокола канального уровня с прямым исправлением ошибок. Для решения данной задачи необходимо разработать методику синтеза протокола, программу моделирования и систему проедоставления полученных результатов в удобном для анализа виде. В ходе решения поставленной задачи так же необходимо разработать программы моделирования потока ошибок, алгоритм и программу моделирования прямого исправления ошибок.

# 2. Исследование существующих методов исправления ошибок

### **2.1. Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды)**

БЧХ-код является циклическим кодом, который задается порождающим полиномом. Для его нахождения в случае БЧХ-кода необходимо заранее определить длину кода n  и требуемое минимальное расстояние d \leqslant n. Найти порождающий полином можно следующим образом.

Пусть ~\alpha — примитивный элемент поля ~GF(q^m) (то есть \alpha^{q^m-1}=1, \alpha^i \neq 1, i< q^m-1), пусть ~\beta=\alpha^s , — элемент поля ~GF(q^m) порядка ~n, \quad s = (q^m-1) / n . Тогда нормированный полином g(x) минимальной степени над полем GF(q), корнями которого являются ~d-1 подряд идущих степеней ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2}элемента ~\beta для некоторого целого ~l_0 (в том числе 0 и 1), является порождающим полиномом БЧХ-кода над полем ~GF(q) с длиной n и минимальный расстоянием ~d_0 \geqslant d. Поясним почему у получившегося кода будут именно такие характеристики (длина кода ~n , минимальное расстояние ~d_0). Действительно, как показано в [1] , длина БЧХ кода равна порядку элемента ~\beta, если ~d>2 и равна порядку элемента ~\beta^{l_0}, если ~d=2, тогда, так как случай ~d=2нам не интересен (такой код не может исправлять ошибки, только обнаруживать), то длина кода будет равна порядку элемента ~\beta ,то есть равна ~n. Минимальное расстояние ~d_0 может быть больше ~d, когда корнями минимальных функций(стр.83[2]) от элементов ~\beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} будут элементы расширяющие последовательность, то есть элементы ~\beta^{l_0+d-1},\beta^{l_0+d},\ldots,\beta^{l_0+d_0 - 2}.

Число проверочных символов r равно степени g(x), число информационных символов k = n - r, величина d называется конструктивным расстоянием БЧХ-кода. Если n = qm - 1, то код называется примитивным, иначе не примитивным.

Так же, как и для циклического кода, кодовый полином c(x) может быть получен из информационного полинома m(x), степени не больше k-1, путём перемножения m(x) и g(x):

Для нахождения порождающего полинома необходимо выполнить несколько этапов:

* выбрать q, то есть поле GF(q), над которым будет построен код;
* выбрать длину n кода из условия:

где m,s — целые положительные числа;

* задать величину d конструктивного расстояния;

1) построить циклотомические классы элемента \beta=\alpha^s поля GF(q^m) над полем GF(q), где \alpha — примитивный элемент GF(q^m);

2) поскольку каждому такому циклотомическому классу соответствует неприводимый полином над GF(q), корнями которого являются элементы этого и только этого класса, со степенью равной количеству элементов в классе, то выбрать \beta^{l_0}, \beta^{l_0+1},\ldots, \beta^{l_0+d-2} таким образом, чтобы суммарная длина циклотомических классов была минимальна; это делается для того, чтобы при заданных характеристиках кода ~n и ~d минимизировать количество проверочных символов ~k;

3) вычислить порождающий полином g(x)=f_1(x)f_2(x)\ldots f_h(x), где f_i(x) — полином, соответствующий i-ому циклотомическому классу; или вычислить g(x), как НОК минимальных функций от элементов \beta^{l_0}, \beta^{l_0+1},\ldots,\beta^{l_0+d-2} .

При несистематическом кодировании кодовое слово получается в виде произведения информационного полинома на порождающий:

Оно может быть реализовано при помощи перемножения полиномов.

При систематическом кодировании кодовое слово формируется в виде информационного подблока и проверочного

Пусть информационное слово образует старшие степени кодового слова, тогда

Тогда из условия  , следует

Главной идеей в декодировании БЧХ кодов является использование элементов конечного поля для нумерации позиций кодового слова (или, эквивалентно, в порядке коэффициентов ассоциированного многочлена).

### **2.2. Коды Соломона Рида**

Коды Рида-Соломона – недвоичные циклические коды, позволяющие исправлять ошибки в блоках данных. Элементами кодового вектора являются группы битов(блоки). Часто на практике используют варианты данного кода, работающего с байтами(октетами).

Коды Рида-Соломона являются частным случаем кодов БЧХ, для которых количество слов в символе m = m0 = 1. Данный вид кода характеризуется следующими параметрами:

* Длина блоков n = q -1, выраженная числом q-ичных символов;
* Количество информационных символов k от 1 до n – 1;
* Минимальное кодовое растояние dmin = n – k + 1.

Для задания кодов Рида-Соломона использую порождающий полином вида:

Поскольку данным полином имеет степень 2t, то возможно использование всего 2t проверочных символов для исправления пакетов из t ошибок. Согласно теореме о границе Рейгера, данный код является оптимальным с точки зрения соотношения длины пакета и возможности исправления ошибок.

Для декодирования кодов Рида-Соломона существуют эффективные алгоритмы декодирования жестких решений, что позволяет использовать относительно длинные коды во многих практических задачах, особенно в системах с q-ичной модуляций.

В настоящий момент коды Рида — Соломона используется при записи и чтении в контроллерах оперативной памяти, при архивировании данных, записи информации на жесткие диски, записи на CD/DVD диски. Даже если поврежден значительный объем информации, испорчено несколько секторов дискового носителя, то коды Рида — Соломона позволяют восстановить большую часть потерянной информации. Также используется при записи на такие носители, как магнитные ленты и штрих коды.

### **Набор исследуемых кодов**

**Таблица 1: параметры исследуемых кодов**

|  |  |  |
| --- | --- | --- |
| **Код** | **Длина кода n, Длина данных k** | **Корректирующая способность кода** |
| **БЧХ** | **(31,21)** | **2** |
| **БЧХ** | **(31,26)** | **3** |
| **БЧХ** | **(63,51)** | **2** |

**Таблица 2: параметры исследуемых кодов**

|  |  |  |
| --- | --- | --- |
| **Код** | **Длина кода n, Длина данных k** | **Корректирующая способность кода** |
| **Рида - Соломона** | **(23,17)** | **3** |
| **Рида - Соломона** | **(31,26)** | **3** |
| **Рида - Соломона** | **(63,51)** | **2** |

### **Обеспечение требуемой вероятности приема**

Помехоустойчивое кодирование позволяет повысить вероятность приема кадра в десятки раз, но повышение в большее количество раз коды, исправляющие ошибки обеспечить не могут. Для обеспечения необходимого уровня приема кадров применяют коды обнаруживающие ошибки. Избыточность таких кодов существенно меньше, чем у кодов исправляющие ошибки. Таким образом, используется комбинация двух кодов: коды обнаруживающие ошибки обеспечивают требуемую вероятность приема, а корректирующий код минимизацию времени задержки приема.

Вероятность не выявления ошибки в случае использования циклического избыточного кода, если ошибка на самом деле есть, в худшем случае равна , где r – степень образующего полинома. Таким образом, для обеспечения должно быть , т.е. . Такой циклический код позволит обнаружить пакеты ошибок длиной до r.

Таким образом выбираем код CRC-30 длиной 30 бит, задаваемый полиномом:

### **Минимизация времени задержки приема**

Использование помехоустойчивого кодирования способно значительно уменьшить время задержки приема данных в плохих каналах связи. Однако, имеется возможность дальнейшей минимизации задержки путем создания необходимой структуры кадра. Проверочные данные в структуре кадра можно передавать после всех блоков с информационными данными, включающих контрольные суммы этих блоков. В таком случае правильность приема данных можно определить до приема всего кадра. Если данные приняты верно, их можно передать на обработку, а проверочные данные игнорировать и таким образом минимизировать время задержки приема. Для реализации такой структуры кадра нужно использовать разделимые коды для помехоустойчивого кодирования.

# Разработка и программная реализация канальных протоколов

# 3.1. Обобщённая структура кадра

Предполагается к исследованию семейство канальных протоколов с потверждением приема кадра, с возможносьбю переспроса кадра. При повторной отправке кадра все последующие кадры, если они уже были переданы отбрасываются.

Кадр данного семейства канальных протоколов состоит из следующих частей:

* 1. Последовательность Голда длиной 63 бита – отделяет один кадр от другого и определяет количество исходных криптоблоков в кадре.
  2. Поле данных, длина которого определяется как l = (120+30)\*c, где 120 – длина криптоблока (задана), 30 – длина CRC, c – количество криптоблоков в сообщении.
  3. Поле проверочных кодов – длина, которых зависит от используемого кода.

Поле данных разделено на блоки, длина которых зависит от используемого кода. Если при приеме данных не выявлено ошибок, то декодер не принимает кадр до конца, а передает данные на обработку и отсылает подтверждение об успешном приеме.

Поле проверочных кодов, как и поле данных, разделено на блоки, длина которых зависит от используемого кода. Количество блоков в поле данных и в поле проверочных символов совпадает; при обнаружении ошибки в первом блоке данных для восстановления используется первый блок проверочных символов и т.д.

# 3.2. Выбор способа реализации канальных протоколов для исследования.

Самым очевидным способом исследованием системы цифровой передачи данных является создание такой системы с последующим ее испытанием в реальных условиях или в специально физически смоделированной среде. Однако, такой способ исследования является затратным. Для его реализации для каждого варианта протокола необходимо реализовать кодер и декодер.

В настоящий момент задачу исследования, могут облегчить системы математического моделирования, в которых существуют библиотеки компонентов, из которых можно создавать модели кодера и декодера, а также моделировать поток ошибок. Система математического моделирования MATLAB является одним из популярных средств для решения таких задач.

Главная особенность среды MATLAB – проработанность и отлаженность всего богатейшего арсенала средств и методов. Возможности MATLAB закрывают большую часть потребностей разработчиков все возможных систем. Обширный набор численных методов в сочетании с средствами графической визуализации делает MATLAB универсальным инструментов для инженерных расчетов и научных исследований.

Однако MATLAB основана на интерпретируемом языке программирования, что обуславливает ее главный недостаток – низкую скорость вычислений. Помимо этого в таких системах приходится создавать громоздкие многоуровневые блок-схемы, не отражающие естественной структуры моделируемой системе.

Таким образом системы типа MATLAB имеет смысл использовать для проверки отдельных алгоритмов и отладки компонентов проектируемой системы передачи данных, но при необходимости выполнить большой объём статистического моделирования за короткое время, чтобы получить при использовании минимума ресурсов максимально быстро отбросить большое количество плохих вариантов, целесообразно использовать грубый метод оценочного моделирования.

Целью данного исследования является получения следующих характеристик для различных вариантов протокола и различных потоков ошибок:

* 1. Скорость передачи данных.
  2. Распределение вероятности доставки кадров по времени.

Скорость передачи данных – это отношение количества правильно принятых полезных бит к общему количеству переданных бит. Полезные биты – это сумма длины криптоблока и длины контрольной суммы этого криптоблока, умноженная на количество криптоблоков в кадре. Определив зависимость скорости передачи от количества полезных бит в кадре можно выбрать оптимальное количество криптоблоков на кадр для данного варианта протокола.

Вероятность доставки кадра за определенное время – это отношение успешно принятых кадров за данное время к общему количеству отправленных кадров за это же время.

Количество попыток ограничено, т.к. в некоторых условиях один кадр может постоянно приниматься неверно, даже с его переспросом. Из-за лимита попыток распределения будет достигать единицы в случае, если вероятность потери кадра равна нулю, в противном случае максимальная вероятность доставки равна:

# 3.3 Алгоритм оценочного моделирования

При оценочном моделировании эффективности протокола в качестве передаваемых данных используют нули, а единицы считают за ошибки.

Варианты протокола задаются следующими характеристиками:

1. Параметры кадра:
   1. Длина начальной флаговой последовательности (бит);
   2. Длина криптоблока (бит);
   3. Длина контрольной суммы для криптоблока (бит);
   4. Количество криптоблоков в кадре;
2. Параметры коды для прямого исправления ошибок:
   1. Длина кода n (бит);
   2. Длина информационной последовательности k (бит);
   3. Количество исправляемых кодов ошибок;
   4. Количество бит в слове.

При оценочном моделировании протоколов с прямым исправлением ошибок необходимо вести подсчет ошибок, приходящиеся на отдельные блоки кадра. Рассмотрим протокол, в котором используется Голея с 5 криптоблоками по 120 бит на кадр. Каждый криптоблок снабжается контрольной суммой длиной 30 бит, поэтому длина полезных данных равна 750 бит. Длина информационной последовательности для кода Голея равна 12 битам, поэтому для передачи данных необходимо 63 кодовых блока. Данный код обеспечивает исправление до 3 ошибок. При анализе потока ошибок нужно определить номер кадра и номер блока внутри кадра, на который приходится ошибка и увеличить на единицу. Если счетчик ошибок превысит порог ошибок для данного кода, то следует признать, что блок не будет восстановлен при приеме, и кадр будет принят неверно.

Данный алгоритм легко обощается для работы с недвоичными кодами. Для этого необходимо дополнительно определить слово в блоке, на которое приходиться ошибка. Если в слово попали еще ошибки, это ничего не меняет – декодер будет исправлять слово целиком.

# 3.4 Реализация инструмента оценочного моделирования

Инструмент оценочного моделирования представляет собой приложения с консольным интерфейсом. Программа выполнена на языке C++ и не зависит от программно.

Пользователь с помощью встроенного генератора смоделировать поток ошибок одним из алгоритмов: ДСК и ОПП.

Пользователь должен установить параметры моделирования:

1. Общее количество информации;
2. Размер блока и длина контрольной суммы для него;
3. Количество криптоблоков в кадре;
4. Установить параметры кода для прямого исправления ошибок(см. Раздел 3.3)
5. Установить модель потока ошибок и ее параметры.

Результатов работы программы является текстовый файл. Имя этого файла формируется из названия модели потока ошибок, ее параметра и параметров кода используемого для моделирования. Например,:

OPP\_model A = 1, V = 0.98 code(63, 57, 1, 1).txt,

Где

OPP\_model – тип потока ошибок;

A и V – параметры данного потока ошибок;

code(63, 57, 1 ,1) – параметры кода.

Пример файла приведен в приложении 2. Интерфейс программы приведен в приложении 3.

# 3.4 Проверка инструмента оценочного моделирования.

# Разработка архитектуры и реализация протокола передачи данных с исправлением данных.

## **4.1. Модель p a**

Основным понятием, положенным в основу данной модели, является плотность ошибка порядка t. Это неслучайная функция от n и t:

.

В числитель является средним числом ошибок на блоке длинной n, содержащих t или больше ошибок. Значения плотности порядка t ограничены снизу величиной t/n, а сверху единицей, т.е. . Значения v(t,n) не убывают с ростом t; . При величине плотности v(1,n) можно судить о степени группирования ошибок, если считать, что увеличение доли ошибок высших кратностей идентично увеличению степени группирования. Для многих каналов, было установлено:

*.*

Параметр α носит название показатель группирования . При значении α равного нулю получаем модель канала с независимыми ошибками, при значении α равного единице канал с “жестким” пакетированием ошибок.

Вероятность приема блока с t ошибками равна:

Использую приближение

Но, на практике обычно применяют более простое соотношение:

Это верхняя граница вероятности . При t/n < 0,3 точные значения близки к верхней границе.

Таким образом, модель(p, a) задается соотношением:

Параметр модели p – вероятность ошибки символа, находится как и для канала ДСК[Надо вставить метод вычисления].

Параметр a вычисляется из уравнения

Данная модель учитывает факт пакетирования ошибок, что возникает в большинстве реальных каналов, имеется возможность единообразно описывать различные типа каналов. Так значения a в кабельных каналах достигает максимального значения(>0.5), а в радиоканалах минимально(~0.3 – 0.45). Но, модель имеет недостаток, заключающейся в вопросе на уровне блоков.

**4.2. Модель ОПП**

Наблюдаемое пакетирование ошибок в каналах связи при предположении о пуассоновском характере потока можно объяснить, если считать параметр  не константой, а случайной величиной или процессом. Получающийся путем рандомизации  новый случайный процесс называют обобщенным пуассоновским . Будем считать  случайной величиной, закон распределения которой известен . Тогда канал задается как поток ошибок первым способом:



Поскольку вид и параметры закона распределения для реальных каналов обычно неизвестны, указанной выше формулой воспользоваться не удается.

По экспериментальным данным относительно легко можно найти закон распределения интервалов между ошибками – функцию Пальма-Хинчина , которая полностью определяет ОПП (второй способ здания потока).

Справедлива формула:

,

где - параметр потока,

 и 

 - вероятность отсутствия ошибок за время .

Таким образом, для ОПП, зная функцию распределения интервалов между ошибками или , вычисляются вероятности , т.е. приходим к конструктивному заданию потока первым способом. Для моделирования будем считать распределение интервалов, заданное обобщенной гиперболой:

, .

Тогда для параметра потока тогда получается:

.

 и

.

Для расчетов вероятностей наиболее удобна рекуррентная формула:



.

Неизвестные параметры  и  легко находятся, например, методом моментов, поскольку обобщенная гипербола для интервалов между ошибками приводит к гамма-распределению параметра .

К недостаткам этой модели можно отнести более трудоемкие формулы для расчета, чем у модели ДСК и тот факт, что не все каналы имеют обобщенную гиперболу в качестве закона распределения между ошибочных интервалов.

Алгоритмы, необходимые для генерации потока ошибок модели ОПП:

|  |  |
| --- | --- |
|  |  |
| **Рис. : Алгоритм генерации потока ошибок** | **Рис. : Алгоритм функции генерации позиции ошибки в потоке битов.** |

**4.3. Двоичный симметричный канал.**

В рамках данной модели дискретный канал связи может быть представлен как симметричный канал без памяти (ДСК), т.е. такой стационарный дискретный канал, в котором вероятности искажения любого из символов 0 или 1 одинаковы. В этом канале вероятность передачи не зависит от статистики передаваемой последовательности. Воздействие помехи можно представить, как позиционное суммирование входной последовательности символов, выдаваемых условным источником помехи, статистическая характеристика которой полностью определяет канал. В ДСК ошибки кратности t подчиняются биномиальному закону распределения, поток ошибок задается через вероятность ошибки бита *р.* Вероятность -кратной ошибки на блоке из символов равна:



Поток ошибок в ДСК без памяти является процессом восстановления с геометрическим распределением интервалов между ошибками .

Параметр  легко находится по экспериментальным данным

, здесь  - число ошибочных символов за сеанс связи,  - число символов переданных за этот сеанс.

К сожалению, число реальных каналов, ошибки которых описываются моделью ДСК весьма мало. Это обычно каналы высокого качества локальных сетей. Основное достоинство данной модели – простота и возможность оценки по ней потенциальных границ вероятностных характеристик качества доставки сообщений в системе.

Генерация потока ошибок для ДСК осуществляется по следующему алгоритму:

Начало

Задание длины сеанса связи N,

Вероятности ошибки символа p

//Счетчик

Counter = 0

R<p

//случайная величина от 0 до 1

R

Бит = 1

Бит = 0

Counter++

Counter<N

Возврат потока битов

Конец

Рисунок 1 Алгоритм генерации потока ошибок для ДСК

## **4.4. Параметры исследования протоколов канального уровня**

**Таблица : параметры потоков ошибок ОПП**

|  |  |  |
| --- | --- | --- |
| Средняя вероятность ошибки p | A | V |
| **0.00487** | **2.9** | **1.094** |
| **0.0132** | **4.5** | **1.1** |
| 0,0439 | **8** | **1.125** |

**Таблица : параметры потоков ошибок ДСК**

|  |
| --- |
| **0,001** |
| **0,01** |
| **0,05** |

# Анализ результатов.

Результатами моделирования являются две характеристики:

* 1. Скорость передачи данных
  2. Распределение вероятности доставки кадра по времени

Лучшим протоколом по критерию “скорость передачи данных” выбирается, такой протокол, который обеспечивает максимальное соотношение количества правильно принятых бит к общему числу отправленных бит.

Лучшим протоколом по критерию “минимальная задержка при передаче” выбирается:

1. В случае, если для потока ошибок, существуют протоколы у которых максимально накопленная вероятности приема кадра превышает 0.95, то выбирается протокол, который достиг данную вероятность раньше других вариантов;
2. В случае , если ни один из вариантов протокола не достиг вероятность приема кадра большую 0.95, то выбирается протокол с максимально накопленной вероятностью.

В ходе моделирования, было получено, что лучший протокол по критерию “скорость передачи” редко совпадает с лучшим протоколом по критерию “минимальная задержка”.

Для потока ошибок, полученным по ОПП показывает код .

Для потока ошибок, полученным по ДСК показывает код .

Во многих случаях код , обеспечивает наивысшую скорость передачи, но при этом часто вероятность безошибочного приема кадра не превышает 0.95. Это озночает, что используя код с большей корректирующей способностью, можно сократить задержку, но такой код дает снижение скорости передачи.

Для протоколов с сильным группированием ошибок вполне может быть эффективным использованием кода БЧХ(63,57,1) – этот код обеспечивает высокую скорость передачи, исправляет редкие одиночные ошибки, а длинные группы ошибок в любом случае приведут к потере кадра, какой бы корректирующий способностью код не обладает.

Ниже приведены таблицы лучших протоколов с одним, одиннадцатью и шестнадцатью криптоблоками на кадр. Отсутствие результата в таблице, означает, что невозможно получить протокол, который будет обладать вероятностью доставки кадра минимум 0,5.

**Таблица : лучшие протоколы с одним криптоблоком на кадр по потокам ошибок**

|  |  |  |  |
| --- | --- | --- | --- |
| Тип потока ошибок | Средняя вероятность ошибки | Скорость передачи | Минимальная задержка |
| ДСК | 0,001 |  |  |
| 0,01 |  |  |
| 0,05 |  |  |
| ОПП | 0,0439 |  |  |
| 0.0132 |  |  |
| 0.00487 |  |  |

**Таблица : лучшие протоколы с одиннадцатью криптоблоками на кадр по потокам ошибок**

|  |  |  |  |
| --- | --- | --- | --- |
| Тип потока ошибок | Средняя вероятность ошибки | Скорость передачи | Минимальная задержка |
| ДСК | 0,001 |  |  |
| 0,01 |  |  |
| 0,05 |  |  |
| ОПП | 0,0439 |  |  |
| **0.0132** |  |  |
| **0.00487** |  |  |

**Таблица : лучшие протоколы с шестнадцатью криптоблоками на кадр по потокам ошибок**

|  |  |  |  |
| --- | --- | --- | --- |
| Тип потока ошибок | Средняя вероятность ошибки | Скорость передачи | Минимальная задержка |
| ДСК | 0,001 |  |  |
| 0,01 |  |  |
| 0,05 |  |  |
| ОПП | 0,0439 | БЧХ (31, 26, 1, 1) | БЧХ (31, 11, 5, 1) |
| 0.0132 |  |  |
| 0.00487 |  |  |

# 5. Заключение

В рамках данной работы для решения задачи разработки протокола канального уровня с прямым исправлением ошибок был осуществлен анализ существующих алгоритмов и способов помехоустойчивой передачи данных и предложена архитектура для семейства протоколов. В ходе решения поставленной задачи был разработан инструмент оценочного моделирования в виде компьютерной программы, позволяющий провести моделирование протокола при использовании различных потоков ошибок. С использованием программы был проведен анализ и для каждого потока ошибок были предложены оптимальные параметры протокола.

Так же в ходе данной работы была создана методика синтеза протокола, которую можно использовать повторно. Алгоритм синтеза описывается следующим образом:

1. Формулирование технического задания на разработку протокола (характеристики потоков ошибок; характеристики передаваемых данных; требования к скорости и надежности передачи данных).
2. Выбор необходимой архитектуры семейства протоколов. В данной работе, есть возможность выбора архитектуры протокола, что позволяет решать большое множество заданий.
3. На основе требований определить параметры протокола, которые однозначно определены.
4. Использую программу оценочного моделирования получить параметры протокола, которые невозможно определить однозначно.

# 6. Список литературы

**Р. Блейхут** Теория и практика кодов, контролирующих ошибки: Пер. с англ. [Книга]. - Москва : Мир, 1986.

**Э. Таненбаум** Компьютерные сети [Книга]. - СПб : Питер, 2003.

**Finneran, Michael F. WiMax versus Wi-Fi. A Comparison Technologies, Markets and Business Plans. 2004**

**Р. Блейхут** Теория и практика кодов, контролирующих ошибки: Пер. с англ. [Книга]. - Москва : Мир, 1986.

**Э. Таненбаум** Компьютерные сети [Книга]. - СПб : Питер, 2003.

**Finneran, Michael F. WiMax versus Wi-Fi. A Comparison Technologies, Markets and Business Plans. 2004.**

**Wikipedia.[В Интернете] http://ru.wikipedia.org/wiki/HDLC**

## **7. Приложение 1 Исходный код**

//  
// Created by Alexander on 12.05.2015.  
//  
#include "api.hpp"  
#ifndef DIPLOM\_PROTOCOL\_H  
#define DIPLOM\_PROTOCOL\_H  
  
class protocol {  
  
public:  
 protocol(const std::vector<Block> &bl, const Code &code) : bl(bl), code(code) {  
 protocol::blSize = (UINT) protocol::bl.begin()->size();  
 protocol::blocks = (UINT) protocol::bl.size();  
 }  
 virtual ~protocol() { }  
  
 const std::string &getResults() const {  
 return results;  
 }  
  
 const Plot &getPlot() const {  
 return plot;  
 }  
 const Plot &getDelProbPlot() const{  
 return delProbPlot;  
 }  
 void work(UINT type, UINT pkSize);  
private:  
 std::vector<Block>bl;  
 std::vector<Packet>pl;  
 std::string results;  
 Code code;  
 Plot plot, delProbPlot;  
 double speed = 0.0, delProbability = 0.0, singleTime = 0.0;  
 UINT blSize, blocks, pkSize, RecivedPackets = 0, SentPackets = 0;  
 UINT blErrors = 0, packetSize;  
  
 void datagramm();  
 void latency();  
 void Nstep();  
  
 bool isCorectable(Packet packet);  
 bool checkPacket(Packet packet);  
 UINT checkBlockErrors(Block bl);  
 bool isCorrectiableBlock(Block block);  
};  
  
  
#endif //DIPLOM\_PROTOCOL\_H

//  
// Created by Alexander on 12.05.2015.  
//  
//***TODO need implement Nstep and latency protocol***#include "protocol.h"  
#include <list>  
  
void protocol::work(UINT type, UINT pkSize) {  
 protocol::pkSize = pkSize;  
 UINT frames = blocks/pkSize;  
 pl = makePackets(pkSize,frames, bl);  
 packetSize = blSize\*pkSize;  
 switch (type){  
 case 1:  
 datagramm();  
 break;  
 case 2:  
 latency();  
 break;  
 case 3:  
 Nstep();  
 break;  
 default:  
 std::cout << "Set correct params" << std::endl;  
 break;  
 }  
 pl.erase(pl.begin(), pl.end());  
 RecivedPackets = 0; SentPackets = 0; delProbability = 0.0; speed = 0; singleTime = 0.0;  
}  
void protocol::datagramm() {  
 bool CoruptedPacket, CorrectablePacket;  
 for(auto packet : pl){  
 CoruptedPacket = checkPacket(packet);  
 if (CoruptedPacket){  
 CorrectablePacket = isCorectable(packet);  
 if(CorrectablePacket){  
 RecivedPackets+=1;  
 }  
 }  
 else{  
 RecivedPackets+=1;  
 }  
 SentPackets+=1;  
 }  
 std::cout << "Sent Packets = " << SentPackets << " RecivedPackets = " << RecivedPackets << std::endl;  
 delProbability = static\_cast<double >(RecivedPackets) / static\_cast<double>(SentPackets);  
 speed = delProbability\*RecivedPackets/SentPackets;  
 speed = speed\*packetSize;  
 plot.speed.emplace\_back(speed);  
 plot.FrameSize.emplace\_back(packetSize);  
 plot.code = protocol::code;  
 std::cout << "Deleviring probability: " << delProbability << std::endl <<  
 "Speed: " << speed << std::endl;  
}  
//***TODO need hard fix***void protocol::latency() {  
 bool CoruptedPacket, CorrectablePacket;  
  
 for(auto packet : pl){  
 CoruptedPacket = checkPacket(packet);  
 if(CoruptedPacket){  
 if(code.errorsCorrection > 0) {  
 CorrectablePacket = isCorectable(packet);  
 if (CorrectablePacket) {  
 RecivedPackets += 1;  
 }  
 }  
 }  
 else{  
 RecivedPackets += 1;  
 }  
 SentPackets += 1;  
 }  
 UINT CodeBlockCount = packetSize/code.DataLength;  
 double frameSize = blSize\*pkSize + (code.DataLength\*code.codeLength);  
 delProbability = static\_cast<double >(RecivedPackets) / static\_cast<double>(SentPackets);  
 //speed = (static\_cast<double>(RecivedPackets)/ static\_cast<double>(SentPackets))\*frameSize;  
 speed = static\_cast<double>(RecivedPackets\*packetSize)/  
 static\_cast<double>(SentPackets\*(blSize\*pkSize+  
 CodeBlockCount\*(code.codeLength-code.DataLength)\*code.bitsWord));  
 singleTime = static\_cast<double>(SentPackets\*SentPackets)/ static\_cast<double>(RecivedPackets\*frameSize);  
 std::stringstream ss;  
 ss <<"Код (" << to\_str(code.codeLength) << ", " << to\_str(code.DataLength) << ", " << to\_str(code.errorsCorrection) << ", " << to\_str(code.bitsWord) << ")**\n**"  
 << "Размер блока, бит = " << blSize << "**\n**Количество блоков в кадре = " << pkSize  
 << "**\n**Отправлено кадров = " << to\_str(SentPackets)  
 << "**\n**Принято кадров = " << to\_str(RecivedPackets)  
 << "**\n**Скорость передачи = " << to\_str(speed)  
 << "**\n**Вероятность доставки кадра = " << to\_str(delProbability)  
 << " **\n**Среднее время задержки = " << to\_str(singleTime);  
 results = ss.str();  
 plot.speed.emplace\_back(speed);  
 plot.FrameSize.emplace\_back(packetSize);  
 plot.code = protocol::code;  
 delProbPlot.speed.emplace\_back(delProbability);  
 delProbPlot.FrameSize.emplace\_back(packetSize);  
 delProbPlot.code = protocol::code;  
}  
void protocol::Nstep() {  
}  
  
bool protocol::isCorectable(Packet packet) {  
 UINT corupted = 0;  
 for(auto block : packet) if(isCorrectiableBlock(block)) corupted += 1;  
 return corupted == 0;  
}  
bool protocol::checkPacket(Packet packet) {  
 UINT corupted = 0;  
 for(auto block : packet) if (checkBlockErrors(block) > 0) corupted+=1;  
 return corupted > 0;  
}  
UINT protocol::checkBlockErrors(Block bl) {  
 UINT errors = 0;  
 for(auto value : bl) if(value == 1) errors+=1;  
 return errors;  
}  
bool protocol::isCorrectiableBlock(Block block) {  
 UINT ipos = 0, epos = code.bitsWord;  
 blErrors = 0;  
 std::list<std::list<UINT> > words;  
 for (auto i = 0; i < block.size() / code.bitsWord; i++) {  
 std::list<UINT> word;  
 for (auto j = ipos; j < epos; j++) {  
 word.emplace\_back(block.at(j));  
 }  
 words.emplace\_back(word);  
 ipos += code.bitsWord;  
 epos += code.bitsWord;  
 }  
 for (auto word : words) {  
 for (auto value : word) {  
 if (value == 1) blErrors+=1;  
 }  
 }  
 if (code.errorsCorrection == 0){  
 return blErrors <= 0;  
 }  
 return blErrors <= code.errorsCorrection;  
}

#include "api.hpp"  
#include "protocol.h"  
#ifndef DIPLOM\_DSK\_H  
#define DIPLOM\_DSK\_H  
  
class dsk  
{  
private:  
public:  
 dsk(UINT BlockSize, UINT SessionSize) : BlockSize(BlockSize), SessionSize(SessionSize) {  
 dsk::Blocks = dsk::SessionSize / dsk::BlockSize;  
 dsk::bytes = makeSession(dsk::SessionSize);  
 }  
 virtual ~dsk() { }  
 void setP(double p) {  
 dsk::p = p;  
 }  
 void setCode(UINT codeleght, UINT correction, UINT data, UINT bitsWord){  
 dsk::code.codeLength = codeleght;  
 dsk::code.DataLength = data;  
 dsk::code.errorsCorrection = correction;  
 dsk::code.bitsWord = bitsWord;  
 }  
 void setProtocol(UINT protocol, UINT PacketSize) {  
 dsk::ProtocolType = protocol;  
 dsk::PacketSize = PacketSize;  
 }  
  
 void setCode(const Code &code) {  
 dsk::code = code;  
 }  
 const Plot &getPlot() const{  
 return plot;  
 }  
 const Plot &getProbPlot() const{  
 return delProbPlot;  
 }  
 void work();  
private:  
 UINT BlockSize, SessionSize, Blocks, ProtocolType, PacketSize;  
 double p;  
 Code code;  
 Plot plot, delProbPlot;  
 protocol\* pr;  
 std::vector<UINT>bytes, errors;  
 std::vector<Block>bl;  
 void saveToFile(std::string str);  
};  
  
#endif //DIPLOM\_DSK\_H

#include "dsk.hpp"  
  
void dsk::work() {  
 std::cout << "======Begin dsk model======" << std::endl;  
 double a = 0, b = 1, r;  
  
 for(auto i = 0; i < bytes.size(); i++){  
 generator(a,b,r);  
 r >= p ? bytes.at(i) = 0:1;  
 }  
 /\*  
 for(auto value : bytes){  
 std::cout << value << "";  
 }\*/  
 std::cout << std::endl;  
// std::cout << "Errors:" << std::endl;  
// print(errors);  
  
 bl = makeBlocks(Blocks, BlockSize, bytes);  
 pr = new protocol(bl,code);  
 pr->work(dsk::ProtocolType, dsk::PacketSize);  
 std::stringstream res;  
 for(UINT i = 1; i <= dsk::PacketSize; i+=5){  
 pr->work(dsk::ProtocolType, i);  
 res << pr->getResults() << "**\n**";  
 }  
 saveToFile(res.str());  
 dsk::plot = pr->getPlot();  
 dsk::delProbPlot = pr->getDelProbPlot();  
 std::cout << "======End dsk model======" << std::endl;  
}  
void dsk::saveToFile(std::string str) {  
std::stringstream ostr;  
 ostr << "Модель канала ДСК p = " << p <<"**\n**"  
 << str << "**\n**";  
 std::cout << ostr.str() << std::endl;  
 std::ofstream fout;  
 std::stringstream fname;  
 fname << "DSK\_model p = " << p << " code(" << code.codeLength << ", " << code.DataLength <<  
 ", " << code.errorsCorrection << ", " << code.bitsWord << ")" << ".txt";  
 fout.open(fname.str());  
 fout.write(ostr.str().c\_str(), sizeof(char)\*ostr.str().size());  
 fout.close();  
}

//  
// Created by Alexander on 28.04.2015.  
//  
#include "api.hpp"  
#include "protocol.h"  
#ifndef DIPLOM\_OPP\_H  
#define DIPLOM\_OPP\_H  
  
  
class opp {  
public:  
 opp(UINT BlockSize, UINT SessionSize):BlockSize(BlockSize), SessionSize(SessionSize) {  
 opp::Blocks = opp::SessionSize / opp::BlockSize;  
 opp::bytes = makeSession(opp::SessionSize);  
 }  
 virtual ~opp() { }  
 void setCode(UINT codeLength, UINT errosCor, UINT dataLength, UINT bitsWord) {  
 opp::code.errorsCorrection = errosCor;  
 opp::code.codeLength = codeLength;  
 opp::code.DataLength = dataLength;  
 opp::code.bitsWord = bitsWord;  
 }  
 void setParams(double A, double V) {  
 opp::A = A;  
 opp::V = V;  
 }  
 void setProtocolType(UINT ProtocolType, UINT PacketSize) {  
 opp::ProtocolType = ProtocolType;  
 opp::PacketSize = PacketSize;  
 }  
 void work();  
  
 void setCode(const Code &code) {  
 opp::code = code;  
 }  
  
 const Plot &getPlot() const {  
 return plot;  
 }  
 const Plot &getDelProbPlot() const{  
 return delProbPlot;  
 }  
private:  
 UINT BlockSize, SessionSize, Blocks, ProtocolType, PacketSize;  
 double A,V;  
 Code code;  
 Plot plot, delProbPlot;  
 std::vector<UINT> bytes, errorsPos;  
 std::vector<Block> bl;  
 protocol\* pr;  
  
 UINT GenOppPos();  
 void saveToFile(std::string str);  
};  
  
  
#endif //DIPLOM\_OPP\_H

//  
// Created by Alexander on 28.04.2015.  
//  
#include "opp.h"  
  
UINT opp::GenOppPos(){  
 double R, a = 0, b = 1;  
 generator(a,b,R);  
 double X = (A/pow(R,(1/V))) - A;  
 return static\_cast<UINT>(X);  
}  
void opp::work()  
{  
 std::cout << "======Begin OPP model======" << std::endl;  
 int Pos = GenOppPos();  
  
 for(auto i = 0; i < bytes.capacity(); i++) {  
 if(Pos == i){  
 bytes[i] = 1;  
 errorsPos.emplace\_back(Pos);  
 Pos = i + 1 + GenOppPos();  
 }  
 else if( Pos != i){  
 bytes[i] = 0;  
 }  
 }  
  
// std::cout << std::endl << "Errors" << std::endl;  
// print(errorsPos);  
  
 bl = makeBlocks(Blocks, BlockSize, bytes);  
 pr = new protocol(bl,code);  
 std::stringstream res;  
 //pr->work(ProtocolType, PacketSize);  
 for(UINT i = 1; i <= PacketSize; i+=5){  
 pr->work(ProtocolType, i);  
 res << pr->getResults() << "**\n**";  
 }  
 saveToFile(res.str());  
 opp::plot = pr->getPlot();  
 opp::delProbPlot = pr->getDelProbPlot();  
 std::cout << "======End OPP model======" << std::endl;  
}  
void opp::saveToFile(std::string str) {  
 std::stringstream ostr;  
 ostr << "Модель канала ОПП с параметрами A = " << A << ", V = " << V <<"**\n**"  
 << str << "**\n**";  
 std::cout << ostr.str() << std::endl;  
 std::ofstream fout;  
 std::stringstream fname;  
 fname << "OPP\_model A = " << A << ", V = " << V << " code(" << code.codeLength << ", " << code.DataLength <<  
 ", " << code.errorsCorrection << ", " << code.bitsWord << ")" <<".txt";  
 fout.open(fname.str());  
 fout.write(ostr.str().c\_str(), sizeof(char)\*ostr.str().size());  
 fout.close();  
}

//  
// Created by Alexander on 27.05.2015.  
//  
  
#ifndef DIPLOM\_GRAPH\_H  
#define DIPLOM\_GRAPH\_H  
#include "GLWindow.hpp"  
#include <GL/glu.h>  
#include <GL/gl.h>  
#include <iostream>  
#include "api.hpp"  
#include <list>  
  
class Graph : public GLWindow {  
public:  
  
 Graph() {  
 Graph::mode = GLUT\_RGBA | GLUT\_SINGLE;  
 Graph::subwindows = 0;  
 }  
 virtual ~Graph() { }  
  
 void setPls(const std::list<Plot> &pls) {  
 Graph::pls = pls;  
 Graph::plot = Graph::pls.front();  
 }  
  
private:  
 std::list<Plot>pls;  
 Plot plot;  
 void text(char\* str);  
protected:  
 void resize(int w, int h);  
 void initGL(void);  
 void createsubwindows(void){};  
 void render(void);  
 void keyboard(unsigned char key, int x, int y){};  
 void mouse(unsigned int button, int state, int x, int y){};  
 void motion(int x, int y){};  
 void passmotion(int x, int y){};  
 void idlefunc(void){};  
};  
  
  
#endif //DIPLOM\_GRAPH\_H

//  
// Created by Alexander on 27.05.2015.  
//  
  
#include "Graph.h"  
void Graph::resize(int w, int h) {  
  
}  
void Graph::initGL(void) {  
 //glClearColor(1.0 , 1.0 , 1.0 , 1.0); ///цвет фона окна  
 glMatrixMode(GL\_PROJECTION);  
 glLoadIdentity();  
 glOrtho(  
 0, plot.FrameSize.back()+300,  
 0, plot.speed.back()+30, -1.0,1.0);  
}  
  
void Graph::render(void) {  
 std::string t;  
 glClear(GL\_COLOR\_BUFFER\_BIT);  
 glPushMatrix();  
 glBegin(GL\_LINES); ///начнём рисовать первую координатную ось - OY. Будем это делать при помощи линий  
  
  
 glColor3f(0,0,1);  
 glVertex2f(20, 0);  
 glVertex2f(20, plot.speed.back()\*3000); ///вторая точка оси  
 glEnd();  
 //OX  
 glColor3f(0,1,0);  
 glBegin(GL\_LINES); ////теперь проделаем тоже для оси  
 glVertex2f(0,plot.speed.front()+.5);  
 glVertex2f(plot.FrameSize.back(),plot.speed.back()+.5);  
 glEnd();  
 //some names  
 glColor3f(1.0, 0.0, 0.0);  
 glRasterPos2f(20, 25);  
 t = "Transfer Rate";  
 for(int i = 0; i<t.size(); i++) glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, t.at(i));  
 glColor3f(1.0, 0.0, 0.0);  
 glRasterPos2f(plot.FrameSize.back(), 2);  
 t = "FrameSize";  
 for(int i = 0; i<t.size(); i++) glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, t.at(i));  
 //params on axis  
 for(auto i = 0.01; i < 1.0; i+=0.02){  
 t = to\_str(i);  
 glRasterPos2f(20, i\*30);  
 for(int j = 0; j<t.size(); j++) glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, t.at(j));  
 }  
 for(auto i = 100; i <= plot.FrameSize.back(); i+=100){  
 t = to\_str(i);  
 glRasterPos2f(i, 30\*0.01);  
 for(int j = 0; j<t.size(); j++) glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_10, t.at(j));  
 }  
 glPopMatrix();  
 glFlush();  
  
 GLdouble r,g,b;  
 for(auto plots : pls) {  
 generator(0,1,r); generator(0,1,g); generator(0,1,b);  
 //glColor3f(1.0, 0.0, 0.0); ///выберем для неё красный цвет  
 glColor3f(r,g,b);  
 glBegin(GL\_LINE\_STRIP);  
 for (int i = 0; i < plot.speed.size(); i++) {  
 glColor3f(r,g,b);  
 glVertex2d(plots.FrameSize[i], plots.speed[i]\*30);  
 }  
 std::stringstream ss;  
 ss << "Code(" << plots.code.codeLength << ", " << plots.code.DataLength << ", " <<  
 plots.code.errorsCorrection << ", " << plots.code.bitsWord << ")";  
 (ss.str()).data();  
 glEnd();  
  
 glPushMatrix();  
 glRasterPos2f(1250+1, \*(std::max\_element(plots.speed.begin(), plots.speed.end())) \*30);  
 for (int i = 0; i < ss.str().size(); i++) glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, ss.str().at(i));  
 glPopMatrix();  
 glFlush();  
 }  
  
}  
  
void Graph::text(char\* str) {  
 char \*p;  
 for (p = str; \*p; p++)  
 {  
 glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*p);  
 }  
}

//  
// Created by Alexander on 28.04.2015.  
//  
  
#ifndef DIPLOM\_MAIN\_H  
#define DIPLOM\_MAIN\_H  
  
#include "opp.h"  
#include "dsk.hpp"  
#include "pa.h"  
#include "Graph.h"  
#endif //DIPLOM\_MAIN\_H

//  
// Created by Alexander on 28.04.2015.  
//  
#include "main.h"  
#include <list>  
int main(){  
 unsigned int SessionLenght = 0, BlockSize = 0, model = 0, protocol = 0, CRC;  
 unsigned int codeLenght = 0, dataLength = 0, errorsCorrection = 0, PacketSize = 0, bitsWord = 0;  
 double A = 0, V = 0, p = 0.0, a = 0.0;  
 Plot cur;  
 std::list<Plot>spdpls, delprobpls;  
  
 setparam(protocol, "Chose protocol type**\n**"  
 "Enter 1 - datagramm, 2 - with latency, 3 - backNstep");  
 setparam(model, "Выберите поток ошибок**\n**"  
 "Enter 1 - ДСК,, 2 - PA, 3 - ОПП");  
 set2params(SessionLenght, BlockSize, "Установите длину сеанса связи и размер криптоблока");  
 set2params(CRC,PacketSize, "Установите длину CRC последовательности и количество криптоблоков в кадре");  
 BlockSize += CRC;  
  
 set2params(codeLenght, errorsCorrection, "Параметры кода:**\n**Установите длину кода n(бит) и"  
 " Количество исправляемых ошибок");  
 set2params(dataLength, bitsWord, "Установите длину информационной последовательности k(бит) и длину слова(бит)");  
  
 dsk\* dk = new dsk(BlockSize, SessionLenght);  
 pa\* pk = new pa(SessionLenght, BlockSize);  
 opp\* op = new opp(BlockSize, SessionLenght);  
 Graph\* gr = new Graph();  
 gr->setinitPosition(0,0);  
 gr->setwidthheight(1024,768);  
  
 switch(model)  
 {  
 case 1:  
 setparam(p, "Enter p for single bit:");  
 dk->setP(p);  
 dk->setCode(codeLenght, errorsCorrection, dataLength, bitsWord);  
 dk->setProtocol(protocol, PacketSize);  
 dk->work();  
 cur = dk->getPlot();  
 spdpls.emplace\_back(cur);  
 cur = dk->getProbPlot();  
 delprobpls.emplace\_back(cur);  
 break;  
 case 2:  
 set2params(p, a, "Установите p и a:");  
 pk->setCode(codeLenght, errorsCorrection, dataLength, bitsWord);  
 pk->setProtocol(protocol, PacketSize);  
 pk->setParams(a,p);  
 pk->work();  
 cur = pk->getSpeedPlot();  
 spdpls.emplace\_back(cur);  
 cur = pk->getTimePlot();  
 delprobpls.emplace\_back(cur);  
 break;  
 case 3:  
 set2params(A, V, "Установите A и V:");  
 op->setProtocolType(protocol, PacketSize);  
 op->setCode(codeLenght, errorsCorrection, dataLength, bitsWord);  
 op->setParams(A, V);  
 op->work();  
 cur = op->getPlot();  
 spdpls.emplace\_back(cur);  
 cur = op->getDelProbPlot();  
 delprobpls.emplace\_back(cur);  
 break;  
 default:  
 std::cout << "Введены некоректные параметры" << std::endl;  
 break;  
 }  
 switch (protocol){  
 case 1:  
 gr->setPls(spdpls);  
 break;  
 case 2:  
 gr->setPls(delprobpls);  
 break;  
 }  
 gr->show();  
 return EXIT\_SUCCESS;  
}

## **Приложение 2 Файл результатов**

Модель канала ОПП с параметрами A = 12, V = 1.094

Код (31, 20, 2, 1)

Размер блока, бит = 150

Количество блоков в кадре = 1

Отправлено кадров = 6666

Принято кадров = 6666

Скорость передачи = 0.660793

Вероятность доставки кадра = 1

Среднее время задержки = 8.65714

Код (31, 20, 2, 1)

Размер блока, бит = 150

Количество блоков в кадре = 6

Отправлено кадров = 1111

Принято кадров = 1111

Скорость передачи = 0.645161

Вероятность доставки кадра = 1

Среднее время задержки = 0.730921

Код (31, 20, 2, 1)

Размер блока, бит = 150

Количество блоков в кадре = 11

Отправлено кадров = 606

Принято кадров = 606

Скорость передачи = 0.646552

Вероятность доставки кадра = 1

Среднее время задержки = 0.26696

Код (31, 20, 2, 1)

Размер блока, бит = 150

Количество блоков в кадре = 16

Отправлено кадров = 416

Принято кадров = 416

Скорость передачи = 0.645161

Вероятность доставки кадра = 1

Среднее время задержки = 0.137748

## **Приложение 3 Интерфейс программы**



