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 2](#_Toc419732915)

[1. Анализ методов передачи данных. 5](#_Toc419732916)

[2. Исследование существующих методов исправления ошибок 6](#_Toc419732917)

[2.1. Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды) 6](#_Toc419732918)

[2.2. Коды Соломона Рида 8](#_Toc419732919)

[2.3. Набор исследуемых кодов 9](#_Toc419732920)

[2.4. Обеспечение требуемой вероятности приема 9](#_Toc419732921)

[2.5. Минимизация времени задержки приема 9](#_Toc419732922)

[3. Разработка архитектуры и реализация протокола передачи данных с исправлением данных. 10](#_Toc419732923)

[3.1. Модель p a 10](#_Toc419732924)

[3.3. Двоичный симметричный канал 13](#_Toc419732925)

[3.4. Алгоритм проверки выбранных кодов БЧХ 15](#_Toc419732926)

[3.5. Дейтограммный протокол передачи данных 16](#_Toc419732927)

[3.6. Протокол с задержкой 16](#_Toc419732928)

[3.7. Протокол с возвращением на n шагов 16](#_Toc419732929)

[3.8. Параметры исследования протоколов канального уровня 16](#_Toc419732930)

[4. Анализ результатов. 17](#_Toc419732931)

[5. Заключение 18](#_Toc419732932)

[6. Список литературы 19](#_Toc419732933)

[7. Приложение 1 Исходный код 20](#_Toc419732934)

[Приложение 2 Полученные результаты в ходе работы 30](#_Toc419732935)

# 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.

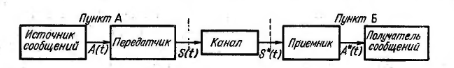


Рис. 1: система передачи информации от одного источника к одному получателю по одному каналу

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.

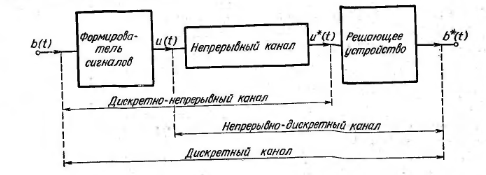


Рис.2: Дискретный канал

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.

# 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, символов | Длина данных k, символов | Корректирующая способность t, символов | Бит на символ |
| БЧХ | 31 | 11 | 5 | 1 |
| БЧХ | 31 | 16 | 3 | 1 |
| Рида-Соломона | 23 | 17 | 3 | 8 |
| Хемминга | 63 | 57 | 1 | 1 |
| Хемминга | 31 | 26 | 1 | 1 |
| БЧХ | 31 | 21 | 2 | 1 |

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

## **3.1. 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.

**3.2. 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.** |

**3.3. 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:



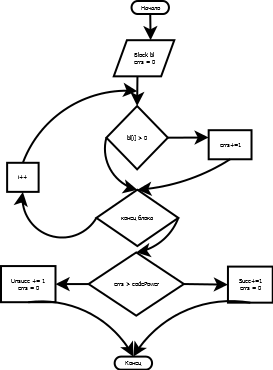
Рисунок 1 Алгоритм генерации потока ошибок для ДСК

## **Алгоритм проверки выбранных кодов БЧХ**

Проверку выбранных кодов БЧХ будем проведем по следующему алгоритму. Так как в нашей модели данные представляется в виде нулей, ошибки представляются в виде единиц, то можем сгенерировать следующей алгоритм проверки кодов БЧХ:

1. Задаем счетчик ошибок errs = 0
2. Проверяем каждый элемент блока на 1 bl[i] > 0
3. Если получаем true увеличиваем счетчик errs на единицу
4. Проверяем достигли конца блока, если не достигли то переходим к следующему элементу в блоке
5. Достигли конца блока и делаем проверку errs > codePower, если да то увеличиваем счетчик Unsucc на единицу, если нет то счетчик Succ на единицу
6. Обнуляем счетчик errs

Для наглядности блок-схема алгоритма приведена на рисунке ниже.



**Рис. : Алгоритм проверки выбранных БЧХ кодов.**

## **3.5. Дейтограммный протокол передачи данных**

## **3.6. Протокол с задержкой**

## **3.7. Протокол с возвращением на n шагов**

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

**Таблица : параметры потоков ошибок ОПП**

|  |  |  |
| --- | --- | --- |
| Средняя вероятность ошибки p | A | V |
|  |  |  |
|  |  |  |
|  |  |  |

**Таблица : параметры потоков ошибок ДСК**

# 4. Анализ результатов.

|  |  |  |  |
| --- | --- | --- | --- |
| Тип потока ошибок | Средняя вероятность ошибки | Скорость передачи | Минимальная задержка |
| ДСК |  |  |  |
|  |  |  |
|  |  |  |
| ОПП |  |  |  |
|  |  |  |
|  |  |  |

# 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.

## **7. Приложение 1 Исходный код**

#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 work();

private:

UINT BlockSize, SessionSize, Blocks, ProtocolType, PacketSize;

double p;

Code code;

protocol\* pr;

std::vector<UINT>bytes, errors;

std::vector<Block>bl;

};

#endif //DIPLOM\_DSK\_H

#include "dsk.hpp"

void dsk::work() {

std::cout << "======Begin dsk model======" << std::endl;

double a = 0, b = 1, r;

auto expr1 = [this](UINT index){

bytes.at(index) = 0;

};

auto expr2 = [this](UINT index){

bytes.at(index) = 1;

errors.emplace\_back(index);

};

for(auto i = 0; i < bytes.size(); i++){

generator(a,b,r);

r >= p ? expr1((UINT) i) : expr2((UINT) i);

}

bl = makeBlocks(Blocks, BlockSize, bytes);

pr = new protocol(bl,code);

pr->work(dsk::ProtocolType, dsk::PacketSize);

std::cout << "======End dsk model======" << std::endl;

}

// 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();

private:

UINT BlockSize, SessionSize, Blocks, ProtocolType, PacketSize;

double A,V;

Code code;

std::vector<UINT> bytes, errorsPos;

std::vector<Block> bl;

protocol\* pr;

UINT GenOppPos();

};

#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);

pr->work(ProtocolType, PacketSize);

std::cout << "======End OPP model======" << std::endl;

}

// 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::attems = 0;

protocol::blSize = (UINT) protocol::bl.begin()->size();

protocol::blocks = (UINT) protocol::bl.size();

}

virtual ~protocol() { }

void setSpeed(double speed) {

protocol::speed = speed;

}

void work(UINT type, UINT pkSize);

private:

std::vector<Block>bl;

std::vector<Packet>pl;

Code code;

double speed = 0.0, PolBits = 0.0, OverallBits = 0.0, percent = 0.0;

double delProbability = 0.0;

UINT blSize, blocks, pkSize, RecivedPackets = 0, SentPackets = 0, attems = 0;

UINT blErrors;

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.

#include "protocol.h"

void protocol::work(UINT type, UINT pkSize) {

protocol::pkSize = pkSize;

pl = makePackets(protocol::pkSize,blocks/protocol::pkSize, bl);

switch (type){

case 1:

datagramm();

break;

case 2:

latency();

break;

case 3:

Nstep();

break;

default:

std::cout << "Set correct params" << std::endl;

break;

}

}

void protocol::datagramm() {

std::cout << "!\*\*\*\*\*\*Datagramm protocol begin\*\*\*\*\*\*!" << std::endl;

std::cout << "!\*\*\*\*\*\*Datagramm protocol end\*\*\*\*\*\*!" << std::endl;

}

void protocol::latency() {

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\*pkSize\*blSize))/SentPackets\*blSize\*pkSize;

double singleTime = RecivedPackets\*pkSize\*blSize / speed;

std::cout << "Deleviring probability: " << delProbability << std::endl <<

"Speed: " << speed << std::endl <<

"Time for work on single packet: " << singleTime << std::endl;

}

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) {

auto errors = checkBlockErrors(block);

blErrors += errors;

return errors <= code.errorsCorrection;

}

## **Приложение 2 Полученные результаты в ходе работы**