Оглавление

[Введение 2](#_Toc418044299)

[Анализ методов передачи данных. 5](#_Toc418044300)

[Исследование существующих методов исправления ошибок 5](#_Toc418044301)

[Коды Соломона Рида 5](#_Toc418044302)

[Коды Боуза—Чоудхури—Хоквингема (БЧХ-коды) 7](#_Toc418044303)

[Разработка архитектуры и реализация протокола передачи данных с исправлением данных. 10](#_Toc418044304)

[Модель p a 10](#_Toc418044305)

[Модель ОПП. 10](#_Toc418044306)

[Алгоритм проверки выбранных кодов БЧХ 12](#_Toc418044307)

[Дейтограммный протокол передачи данных 13](#_Toc418044308)

[Протокол с задержкой 13](#_Toc418044309)

[Протокол с возвращением на n шагов 13](#_Toc418044310)

[Анализ результатов. 14](#_Toc418044311)

[Приложения 15](#_Toc418044312)

[Приложение 1 Исходный код 15](#_Toc418044313)

[Приложение 1.1 Реализация модели канала типа ОПП. 15](#_Toc418044314)

[Приложение 1.2 Реализация модели канала типа p a. 17](#_Toc418044315)

[Приложение 1.3 Реализация общих функции для моделей каналов 18](#_Toc418044316)

# 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.

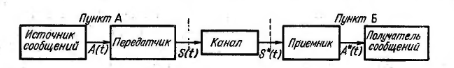


Рисунок Система передачи информации от одного источника к одному получателю по одному каналу

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.

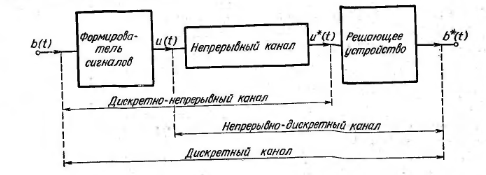


Рисунок Дискретный канал

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

### Коды Соломона Рида

**Коды Рида — Соломона**  — недвоичные циклические коды, позволяющие исправлять ошибки в блоках данных. Элементами кодового вектора являются не биты, а группы битов (блоки).

Коды Рида — Соломона являются важным частным случаем БЧХ-кода, корни порождающего полинома которого лежат в том же поле, над которым строится код (m=1). Пусть \alpha — элемент поля \textstyle GF(q) порядка \textstyle n. Если \alpha —*примитивный* элемент, то его порядок равен q-1, то есть \alpha^{q-1}=1,\quad \alpha^i \neq 1, 0<i<q-1. Тогда нормированный полином g(x) минимальной степени над полем \textstyle GF(q), корнями которого являются d-1 подряд идущих степеней \alpha^{l_0}, \alpha^{l_0+1},...,\alpha^{l_0+d-2} элемента \alpha, является порождающим полиномом кода Рида — Соломона над полем \textstyle GF(q):

g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1})\dots(x - \alpha^{l_0+d-2})

где l_0 — некоторое целое число (в том числе 0 и 1), с помощью которого иногда удается упростить кодер. Обычно полагается l_0 = 1. Степень многочлена \textstyle g(x) равна d-1.

Длина полученного кода n, минимальное расстояние d (минимальное расстояние d линейного кода является минимальным из всех расстояний Хемминга всех пар кодовых слов, см. Линейный код). Код содержит r=d-1=\deg (g(x)) проверочных символов, где \deg() обозначает степень полинома; число информационных символов k = n - r = n - d + 1. Таким образом \textstyle d = n - k + 1 и код Рида — Соломона является *разделимым кодом с максимальным расстоянием* (является оптимальным в смысле границы Синглтона).

Кодовый полином c(x) может быть получен из информационного полинома m(x), \deg m(x) \leqslant k-1, путем перемножения m(x) и g(x):

c(x) = m(x)g(x)

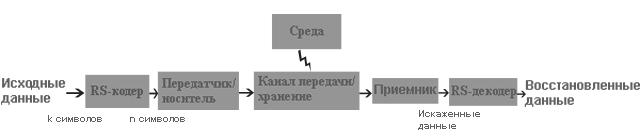
Код Рида — Соломона является одним из наиболее мощных кодов, исправляющих многократные пакеты ошибок. Применяется в каналах, где пакеты ошибок могут образовываться столь часто, что их уже нельзя исправлять с помощью кодов, исправляющих одиночные ошибки.

(q^m - 1,  q^m -2 - 2t)-код Рида — Соломона над полем \textstyle GF(q^m) с кодовым расстоянием d = 2t + 1 можно рассматривать как ((q^m - 1)m,(q^m -1 - 2t)m)-код над полем \textstyle GF(q), который может исправлять любую комбинацию ошибок, сосредоточенную в t или меньшем числе блоков из m символов. Наибольшее число блоков длины m, которые может затронуть пакет длины l_i, где l_i \leqslant mt_i - (m-1), не превосходит t_i, поэтому код, который может исправить t блоков ошибок, всегда может исправить и любую комбинацию из p пакетов общей длины l, если l+(m-1) \leqslant mt.

Кодирование с помощью кода Рида — Соломона может быть реализовано двумя способами: систематическим и несистематическим.

При несистематическом кодировании информационное слово умножается на некий неприводимый полином в поле Галуа. Полученное закодированное слово полностью отличается от исходного и для извлечения информационного слова нужно выполнить операцию декодирования и уже потом можно проверить данные на содержание ошибок. Такое кодирование требует большие затраты ресурсов только на извлечение информационных данных, при этом они могут быть без ошибок.

При систематическом кодировании к информационному блоку из k символов приписываются 2t проверочных символов, при вычислении каждого проверочного символа используются все k символов исходного блока. В этом случае нет затрат ресурсов при извлечении исходного блока, если информационное слово не содержит ошибок, но кодировщик/декодировщик должен выполнить k(n - k) операций сложения и умножения для генерации проверочных символов. Кроме того, так как все операции проводятся в поле Галуа, то сами операции кодирования/декодирования требуют много ресурсов и времени. Быстрый алгоритм декодирования, основанный на быстром преобразовании Фурье, выполняется за время порядка  O({ln( {n}) }^2).

[](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB:%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%BF%D1%80%D0%B8%D0%BC%D0%B5%D0%BD%D0%B5%D0%BD%D0%B8%D1%8F_%D0%BA%D0%BE%D0%B4%D0%B0_%D0%A0%D0%B8%D0%B4%D0%B0-%D0%A1%D0%BE%D0%BB%D0%BE%D0%BC%D0%BE%D0%BD%D0%B0.gif)

При операции кодирования информационный полином умножается на порождающий многочлен. Умножение исходного слова S длины k на неприводимый полином при систематическом кодировании можно выполнить следующим образом:

* К исходному слову приписываются 2t нулей, получается полином \textstyle T = S x^{2t}.
* Этот полином делится на порождающий полином G, находится остаток R, \textstyle S x^{2t} = Q G + R, где Q — частное.
* Этот остаток и будет корректирующим кодом Рида — Соломона, он приписывается к исходному блоку символов. Полученное кодовое слово \textstyle C = S x^{2t} + R.

Кодировщик строится из сдвиговых регистров, сумматоров и умножителей. Сдвиговый регистр состоит из ячеек памяти, в каждой из которых находится один элемент поля Галуа.

Существует и другая **процедура** кодирования (более практичная и простая). Положим  a_{i} \in GF(q) , (i = 1,2,\ldots,k-1) , \alpha \in GF(q) - примитивный элемент поля GF(q), и пусть 
a = (a_0,a_1,\ldots,a_{k-1})
 - вектор информационных символов , а значит a(x) = a_0 + a_{1}x + \ldots + a_{k-1}x^{k-1} - информационный многочлен. Тогда вектор u = (a(1),a(\alpha),\ldots,a(\alpha^{q-2})) есть вектор кода Рида - Соломона , соответствующий информационному вектору a. Этот способ кодирования показывает , что для кода РС вообще не нужно знать порождающего многочлена и порождающей матрицы коды, достаточно знать разложение поля GF(q)по примитивному элементу \alpha и размерность кода k (длина кода в этом случае определяется как n = q - 1). Все дело в том , что за разностью n-k полностью скрывается порождающий многочлен g(x) и кодовое расстояние.

Декодировщик, работающий по авторегрессивному спектральному методу декодирования, последовательно выполняет следующие действия:

* Вычисляет синдром ошибки
* Строит полином ошибки
* Находит корни данного полинома
* Определяет характер ошибки
* Исправляет ошибки

### 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 dназывается *конструктивным расстоянием* БЧХ-кода. Если n=q^m-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):

c(x)=m(x)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 n=(q^m-1)/s, 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

c(x)=m(x)g(x).

Оно может быть реализовано при помощи перемножения полиномов.

При систематическом кодировании кодовое слово формируется в виде информационного подблока и проверочного c(x) = [s(x)\;m(x)]

Пусть информационное слово образует старшие степени кодового слова, тогда

c(x)=x^rm(x) + s(x),r=n-k

Тогда из условия c(x)=x^rm(x) + s(x)=0\mod g(x), следует

s(x)=-x^r m(x)\mod g(x)

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

Основные методы декодирования кодов БЧХ:

* **Евклидов алгоритм** (ЕА). Из-за высокой регулярности структуры этого алгоритма его широко используют для аппаратной реализации декодеров БЧХ и кодов Рида-Соломона.

В основе этого метода лежит широко известный алгоритм Евклида по нахождению наибольшего общего делителя двух чисел (НОД), только в данном случае ищем НОК не двух чисел, а двух полиномов.

**Прямое решение** (алгоритм Питерсона — Горенстейна — Цирлера, ПГЦ). Исторически это первый метод декодирования, найденный Питерсоном для двоичного случая (q=2), затем Горенстейном и Цирлером для общего случая. Этот алгоритм находит коэффициенты *многочлена локаторов ошибок* прямым решением соответствующей системы линейных уравнений. В действительности, так как сложность этого алгоритма растет как куб минимального расстояния ~d, прямой алгоритм может быть использован только для малых значений ~d.

* **Алгоритм Берлекемпа-Мэсси** (BMA). Алгоритм поиска кратчайшего регистра сдвига с линейной обратной связью для поданной на вход бинарной последовательности. Также алгоритм позволяет найти минимальный многочлен поданной на вход линейной рекуррентной последовательности над произвольным полем.

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

## Модель p a

Основным понятием, положенным в основу данной модели является плотность ошибка порядка t. Это неслучайная функция от n и t , где - среднее число ошибок на блоке длинной n с t или большим количеством ошибок. Значение плотности порядка t подчиняется следующему условию . Значения функции v(t,n) не убывают с ростом t, то есть

, если выполняются условия больше p хотя бы в несколько раз. Параметр носит название показателя группирования ошибок, подчиняется условию . При получаем канал с независимыми ошибками, а при получаем канал с “жестким” пакетированием ошибок.

На практике применяют данное соотношение для задания модели (p, a). Параметр p – характеризует вероятность ошибки символа и рассчитывается по формуле ФОРМУЛА. Параметр а – находят из уравнения .

Достоинством применение данной модели для исследования каналов связи является то, что учитывается пакетирования ошибок, возможность единообразно описать разные типы каналов передачи данных. Так для кабельных каналов значение максимально (>0.5), а для радиоканалов применяют значение в промежутке от 0.3 до 0.45. У данного способа моделирования каналов связи есть недостаток, заключающийся в неполноте и вопрос модели на уровне блоков.

## Модель ОПП.

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

Т.е формула для P(k,n) сохраняется, но осуществляется усреднение по параметру.

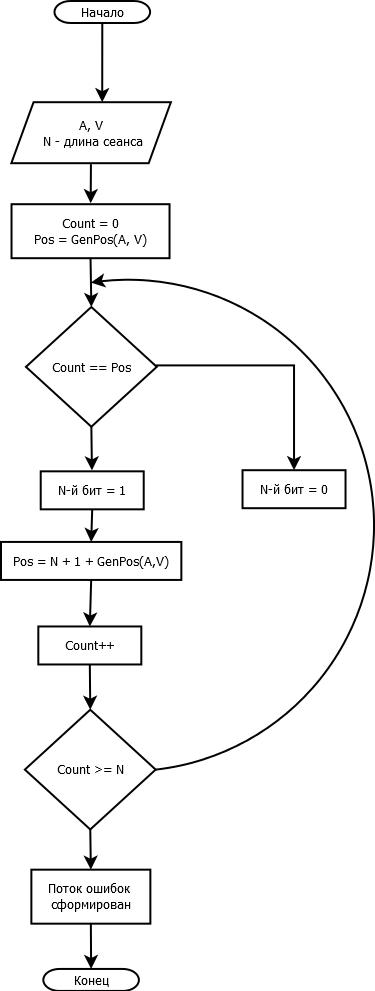
Поскольку вид и параметры закона распределения для реальных каналов обычно неизвестны, указанной выше формулой воспользоваться не удается.

Используя производящую функцию вероятностей:

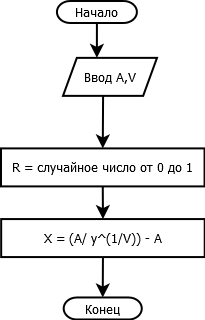
и

Таким образом, для ОПП, зная функцию распределения интервалов между ошибками или P0(t) вычисляются вероятности P(k,t), таким образом получаем первый способ задание потока.

На рисунках представлен алгоритм генерации потока по данной модели.



**Рисунок . Алгоритм генерации потока ошибок**

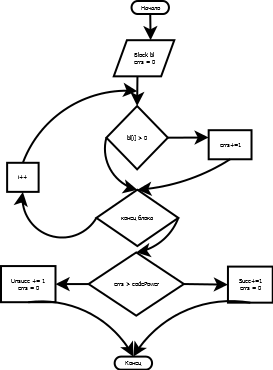


**Рисунок . Алгоритм функции генерации позиции ошибки в потоке битов.**

## 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



## Дейтограммный протокол передачи данных

## Протокол с задержкой

## Протокол с возвращением на n шагов

# Анализ результатов.

# Приложения

## Приложение 1 Исходный код

### Приложение 1.1 Реализация модели канала типа ОПП.

Файл opp.h

#include "api.hpp"

#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;

}

virtual ~opp() { }

void setCode(UINT codeLength, UINT errosCor) {

opp::code.errorsCorrection = errosCor;

opp::code.codeLegth = codeLength;

}

void setParams(double A, double V) {

opp::A = A;

opp::V = V;

}

void setProtocolType(UINT ProtocolType) {

opp::ProtocolType = ProtocolType;

}

void work();

private:

UINT BlockSize, SessionSize;

UINT Blocks, ProtocolType;

Code code;

std::vector<UINT> bytes, errorsPos;

std::vector<Block> bl;

double A,V;

UINT GenOppPos();

};

#endif //DIPLOM\_OPP\_H

Файл opp.cpp

#include "opp.h"

UINT opp::GenOppPos(){

double R, a = 0, b = 1;

generator(a,b,R);

double X = (A/pow(R,(1/V))) - A;

std::cout << "X = " << X << " R = " << R << std::endl ;

return static\_cast<int>(X);

}

void opp::work(){

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

bytes = makeSession(opp::SessionSize);

int Pos = GenOppPos();

for(int i = 0; i < bytes.capacity()-1; 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);

std::cout << "Blocks:" << std::endl;

printDb(bl);

switch (ProtocolType){

case 1:

datagrammProtocol(bl, code);

break;

case 2:

UINT latency;

std::cout << "Enter latency" << std::endl;

std::cin >> latency;

latencyProtocol(bl, code, latency);

break;

case 3:

UINT steps;

std::cout << "Enter steps for protocol" << std::endl;

std::cin >> steps;

backNsteps(bl, code, steps);

break;

}

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

}

### Приложение 1.2 Реализация модели канала типа p a.

**Файл pa.h**

#include "api.hpp"

class pa{

public:

pa(UINT SessionSize, UINT BlockSize, UINT PacketSize){

pa::SessionSize = SessionSize;

pa::BlockSize = BlockSize;

pa::PacketSize = PacketSize;

pa::Blocks = this->SessionSize / this->BlockSize;

pa::Packets = this->Blocks / this->PacketSize;

}

~pa(){}

void work();

void setParams(double a, double p) {

pa::a = a;

pa::p = p;

}

void setCode(UINT codelenght, UINT correction){

pa::codelenght = codelenght;

pa::correction = correction;

}

void setProtocol(UINT protocol){

pa::protocol = protocol;

}

private:

UINT BlockSize, PacketSize, SessionSize;

UINT Blocks, Packets;

UINT codelenght, correction, protocol;

std::vector<UINT> bytes;

std::vector<Block> bl;

std::vector<Packet> pl;

double p, a;

};

#endif //DIPLOM\_PA\_H

**Файл pa.cpp**

#include "pa.h"

void pa::work(){

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

bytes = makeSession(SessionSize);

bl = makeBlocks(Blocks, BlockSize, bytes);

pl = makePackets(PacketSize, Packets, bl);

std::cout << "Analyze packets:" << std::endl;

UINT Succeful = 0, UnSucceful = 0;

checkPacketStream(pl, Succeful, UnSucceful);

double R = ((Succeful\*PacketSize) \* BlockSize)/SessionSize;

std::cout << "Packets in session: " << pl.capacity() << std::endl;

std::cout << "Succeful transmited packets: " << Succeful << std::endl;

std::cout << "Unsucceful transmited packets: " << UnSucceful << std::endl;

std::cout << "Result speed: " << R << std::endl;

std::cout << "End pa model" << std::endl;

}

### Приложение 1.3 Реализация общих функции для моделей каналов

**Файл api.hpp**

#ifndef DIPLOM\_API\_HPP

#define DIPLOM\_API\_HPP

#include <vector>

#include <iostream>

#include <algorithm>

#include <cstdlib>

#include <ctime>

#include <string>

#include <random>

typedef unsigned int UINT;

const UINT STRING\_LENGHT = 40;

typedef std::vector<UINT> Block;

typedef std::vector<Block> Packet;

struct Code{

UINT errorsCorrection;

UINT codeLegth;

};

void print(std::vector<UINT> cont);

void printDb(std::vector<std::vector<UINT>> cont);

bool checkPacket(Packet pk);

std::vector<UINT> makeSession(UINT SessionLenght);

std::vector<Block> makeBlocks(UINT Blocks, UINT BlockSize, std::vector<UINT>bytes);

std::vector<Packet> makePackets(UINT PacketSize, UINT Packets, std::vector<Block> v);

void checkPacketStream(std::vector<Packet> ps, UINT& Succeful, UINT& Unsucceful);

void generator(double a, double b, double& num);

std::vector<UINT> decodeBMA(std::vector<UINT>bytes);

void datagrammProtocol(std::vector<Block>bl, Code code);

void backNsteps(std::vector<Block>bl, Code code, UINT steps);

void latencyProtocol(std::vector<Block>bl, Code code, UINT latency);

#endif //DIPLOM\_API\_HPP

**Файл api.cpp**

#include "api.hpp"

void print(std::vector<UINT> cont){

UINT counter = 0;

for(auto value : cont){

if (counter == STRING\_LENGHT){

std::cout << std::endl;

counter = 0;

}

std::cout << value << " ";

counter++;

}

std::cout << std::endl;

}

void printDb(std::vector<std::vector<UINT>> cont){

for(auto value : cont){

print(value);

}

std::cout << std::endl;

}

bool checkPacket(Packet pk){

bool result = false;

UINT Succeful = 0, Unsucceful = 0;

for(auto value : pk){

for(auto val : value){

val == 0 ? Succeful += 1 : Unsucceful += 1;

}

}

result = Unsucceful == 0 ? true : false;

return result;

}

std::vector<UINT> makeSession(UINT SessionLenght){

std::vector<UINT> ve;

ve.resize(SessionLenght);

std::fill(ve.begin(), ve.end(), 0);

std::cout << "Bytes in session: " << ve.capacity() << std::endl;

return ve;

}

std::vector<Block> makeBlocks(UINT Blocks, UINT BlockSize, std::vector<UINT>bytes){

std::vector<Block> ve;

UINT IPOS = 0, EPOS = BlockSize;

ve.reserve(Blocks);

for(UINT i = 0; i < Blocks; i++){

Block bk;

for(auto i = IPOS; i < EPOS; i++){

bk.emplace\_back(bytes.at(i));

}

ve.emplace\_back(bk);

IPOS += BlockSize;

EPOS += BlockSize;

}

return ve;

}

std::vector<Packet> makePackets(UINT PacketSize, UINT Packets, std::vector<Block> v){

std::vector<Packet>ve;

UINT IPOS = 0, EPOS = PacketSize;

ve.reserve(Packets);

for(UINT i = 0; i < Packets; i++){

Packet pk;

for(auto i = IPOS; i < EPOS; i++){

pk.emplace\_back(v.at(i));

}

ve.emplace\_back(pk);

IPOS += PacketSize;

EPOS += PacketSize;

}

std::cout << "Packets created: " << ve.capacity() << std::endl;

return ve;

}

void checkPacketStream(std::vector<Packet> ps, UINT& Succeful, UINT& Unsucceful){

for(auto value : ps){

checkPacket(value) ? Succeful += 1 : Unsucceful += 1;

}

}

void generator(double a, double b, double& num){

std::random\_device rd;

std::mt19937 gen(rd());

std::uniform\_real\_distribution<> dis(a, b);

num = dis(gen);

}

std::vector<UINT> decodeBMA(std::vector<UINT>bytes){

std::vector<UINT> b,c,t,s;

int N, L, m, d;

b.resize(bytes.capacity());

t.resize(bytes.capacity());

c.resize(bytes.capacity());

s.resize(bytes.capacity());

std::fill(b.begin(), b.end(), 0);

std::fill(t.begin(), t.end(), 0);

std::fill(c.begin(), c.end(), 0);

std::fill(s.begin(), s.end(), 0);

b[0] = c[0] = 1;

N = L = 0;

m = -1;

while(N < (int)s.capacity()){

d = 0;

for(int i = 0; i <= L; i++){

d += s[N-i] \* c[i];

}

d = d % 2;

if (d != 0){

t = c;

for(int i = 0; i <= s.capacity() + m - 1 - N; i++){

c[N - m + i] = c[N - m + i] ^ b[i];

}

if( L <= (N/2)){

L = N + 1 - L;

m = N;

b = t;

}

}

N++;

}

return c;

}

void datagrammProtocol(std::vector<Block>bl, Code code){

UINT Succeful = 0, Unsucceful = 0, errsCounter = 0;

UINT BlockSize = bl.at(0).capacity();

std::vector<Block>ble;

//TODO implement datagramm protocol

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

for(auto value : bl){

for(auto val : value){

if(val == 1){

errsCounter++;

}

}

if(errsCounter > code.errorsCorrection){

Unsucceful++;

ble.emplace\_back(value);

}

else{

Succeful++;

}

errsCounter = 0;

}

std::cout << "Blocks with errors" << std::endl;

printDb(ble);

double speed = (Succeful\*BlockSize)/bl.capacity();

std::cout << "Blocks in session: " << bl.capacity() << std::endl <<

"Succeful blocks: " << Succeful << std::endl <<

"Unsucceful blocks: " << Unsucceful << std::endl <<

"Percent succeful: " << (Succeful\*100)/bl.capacity() << std::endl <<

"Result speed: " << speed << std::endl;

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

}

void backNsteps(std::vector<Block>bl, Code code, UINT steps){

//TODO implement back n steps protocol

}

void latencyProtocol(std::vector<Block>bl, Code code, UINT latency){

//TODO implement protocol with latency

}