БГУИР

Кафедра ЭВМ

Лабораторная работа №3

Методы защиты от ошибок, используемые в сетях

Выполнил Проверил:

Студент группы 750502 Глоба А.А

Никанов И.В.

Минск 2019

**Основные теоретические сведения:**

Код Хэмминга: возьмем строку “ha”.

В бинарном виде выглядит следующим образом: 0100010000111101.

Вставляем контр. биты(изначально 0): 000010000100001011101.

Вычисляем контр. биты:

1) Для 1-го контр. бита кол-во единиц получилось равным 5, значит бит становится единицей;

2) Для 2-го кол-во единиц = 4 => бит остается в нуле;

3) 4 бит – кол-во единиц =3 => бит становится единицей;

4) 8 бит - кол-во единиц =2 => бит остается в нуле;

5) 16 бит - кол-во единиц =4 => бит остается в нуле.

Получилась следующая последовательность: 100110000100001011101.

Предположим мы получили закодированное сообщение, но оно пришло к нас с ошибкой. К примеру 11-ый бит передался неправильно: 1000110000110001011101.

Теперь необходимо заново вычислить контр. биты:

1) 1-ый бит – поменялся на ноль;

2) 2-ый бит – поменялся на единицу;

3) 4-ый бит – остался единицей;

4) 8-ый бит – поменялся на единицу;

5) 16-ый бит – остался в нуле.

1, 2, 8 не совпадают с такими же контрольными битами, которые мы получили. Позицию ошибочного бита можно получить, просто сложив номера позиций неправильных контрольных бит (1 + 2 + 8 = 11). Теперь просто инвертировав его и отбросив контрольные биты, мы получим исходное сообщение.

CRC(Cyclic redundancy check) код: символ “h”.

Для каждого символа находится CRC код, который потом складывается

CRC=byte(1) + byte(2) + byte(3)+ ... + byte(N). + byte(N) и используется при проверке входных данных на приёмнике. Но можно пропустить ошибку, если при случайном сбое один байт увеличится на некоторое значение, а другой байт уменьшится на то же значение.

h – 00010110

Допустим, порождающий многочлен: x4+x3+1 или 11001

Тогда при выполнении сложения по модулю 2 00010110 и порождающего многочлена получим:

000101100000|11001

11001

11110

11001

11100

11001

1010 - CRC код

При проверке входных данных при подстановке должна выполняться операция XOR информационных бит с подставленным в конце кодом CRC и порождающий многочлен. При безошибочной передаче данных результат будет равен нулю.

**Код лабораторной работы:**

**Файл MyLib.cpp**

std::deque<bool> EncodeMessage(std::string message) //функция кодирования

{

if (message.length() != messageLengthInBytes)

{

throw std::exception("First argument must contains 2 symbols only");

}

std::deque<bool> result = BytesToBits(message);

//for main indexing from 1

/\*std::cout << "\n\njaja = ";

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

{

std::cout << result[i];

}

std::cout << "\n\n";\*/

result.push\_front(false);

//inserting control bits

for (auto i = 0; i < numOfControlBits; ++i)

{

//1 << i == pow(2, i)

result.insert(result.begin() + (1 << i), false);

}

/\*std::cout << "\n\njaja1 = ";

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

{

std::cout << result[i];

}

std::cout << "\n\n";\*/

for (auto i = 0; i < numOfControlBits; ++i)

{

//1 << i == pow(2, i)

result[1 << i] = CalculateControlBit(result, i);

std::cout << "Control bit[" << pow(2, i) << "] = " << result[1 << i];

std::cout << "\n";

}

//removing indexing from 1

result.pop\_front();

return result;

}

std::string DecodeMessage(std::deque<bool> message, int& numOfFoundErrors, int& posOfError) //функция раскодирования

{

//for main indexing from 1

message.push\_front(false);

for (auto i = 0; i < numOfControlBits; ++i)

{

//1 << i == pow(2, i)

int a = CalculateControlBit(message, i);

std::cout << "Control bit[" << pow(2, i) << "] after error = " << a;

std::cout << "\n";

//std::cin >> abc;

}

if (!FindAndRemoveErrorsIfCan(message, numOfFoundErrors, posOfError))

{

return "haha";

}

//removing control bits

for (auto i = numOfControlBits - 1; i >= 0; --i)

{

//1 << i == pow(2, i)

message.erase(message.begin() + (1 << i));

}

//removing indexing from 1

message.pop\_front();

return BitsToBytes(message);

}

bool FindAndRemoveErrorsIfCan(std::deque<bool>& message, int& numOfFoundErrors, int& posOfError) //функция, которая убирает ошибки

{

int errorPos = 0;

for (auto i = 0; i < numOfControlBits; ++i)

{

if (CalculateControlBit(message, i))

{

//1 << i == pow(2, i)

errorPos += (1 << i);

}

}

numOfFoundErrors = (errorPos) ? 1 : 0;

posOfError = errorPos - 1;

if (static\_cast<unsigned>(posOfError) >= message.size())

{

numOfFoundErrors = 2;

return false;

}

message[errorPos] = !message[errorPos];

return true;

}

bool CalculateControlBit(const std::deque<bool>& bits, const int calculatingLevel) //подсчет контр. битов

{

auto res = false;

const auto delta = 1 << calculatingLevel;

for (auto i = 0U; i < bits.size(); ++i)

{

//if i need to calculate

if (i & delta)

{

res ^= static\_cast<bool>(bits[i]);

//std::cout << "i = "<<i<<"\n";

//std::cout << "res = "<<res<<"\n";

}

}

return res;}