МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Череповецкий государственный университет»

**Лабораторная работа № 1**

**«ПЕРЕДАЧА ДИСКРЕТНЫХ ДАННЫХ ПО ЛИНИЯМ СВЯЗИ»**

**Выполнил:**

студент гр. 1ИВТпб-01-31оп

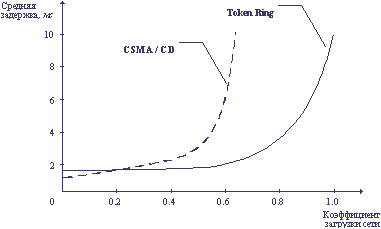
Климов А.Г.  
**Проверил:** преподаватель

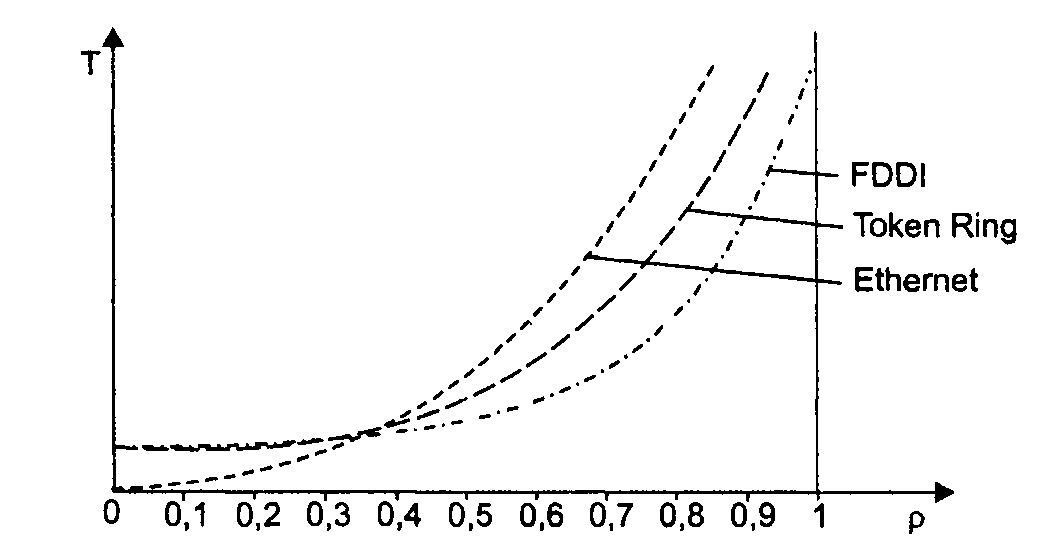
Селяничев О.Л.  
Отметка о зачете:

Череповец

2018 год

Цель работы: Закрепление теоретических знаний в области расчета производительности сети и определение влияющих на неё факторов. Изучение особенностей использования и применения протокола ICMP.

1. Построить экспериментальный график зависимости среднего времени ожидания доступа к среде для протокола Ethernet от коэффициента использования сети. Определить параметры, которые влияют на вид этой зависимости.  




где T – средняя задержка в мс,

ρ – коэффициент загрузки сети.

Как видно из рисунка, всем технологиям присуща качественно одинаковая картина экспоненциального роста величины задержек доступа при увеличении коэффициента использования сети. Однако их отличает порог, при котором наступает резкий перелом в поведении сети, когда почти прямолинейная зависимость переходит в крутую экспоненциальную. Для всего семейства технологий Ethernet — это 30-50 % (сказывается эффект коллизий), для технологии Token Ring — 60 %, а технологии FDDI - 70-80 %.

Количество узлов, при которых коэффициент использования сети начинает приближаться к опасной границе, зависит от типа функционирующих в узлах приложений. Если раньше для сетей Ethernet считалось, что 30 узлов — это вполне приемлемое число для одного разделяемого сегмента, то сегодня, в условиях, когда мультимедийные приложения передают по сети большие файлы данных, предельное число узлов может составлять 5-10.

Топологии = шина, кольцо, звезда.

Технологии = Ethernet, Token Ring, FDDI.

CSMA/CD (Carrier Sense Multiple Access with Collision Detection — множественный доступ с прослушиванием несущей и обнаружением коллизий) — технология (IEEE 802.3) множественного доступа к общей передающей среде в локальной компьютерной сети с контролем коллизий. CSMA/CD относится к децентрализованным случайным (точнее, квазислучайным) методам. Он используется как в обычных сетях типа Ethernet, так и в высокоскоростных сетях (Fast Ethernet, Gigabit Ethernet).

**Метод доступа и кадры для сетей Ethernet**

На логическом уровне в Ethernet применяется топология шина:

* все устройства, подключённые к сегменту сети, равноправны, т. е. любая станция может начать передачу в любой момент времени, если передающая среда свободна,
* кадр, передаваемый одной станцией, одновременно анализируется всеми остальными станциями сегмента.

Стандарты Ethernet поддерживают метод доступа CSMA/CD (Carrier Sense Multiple Access / Collision Detection) и обеспечивают скорость передачи по шине 10 Мбит/с и 100 Мбит/с. По-русски этот метод доступа называется "Множественный доступ с контролем несущей и обнаружением коллизий". Ниже приводится краткое описание этого метода доступа.

Передача данных происходит следующим образом. Станция проверяет состояние среды передачи данных (шины). Если среда занята, то станция ожидает освобождения среды. Если среда свободна, то станция начинает передавать кадр данных, одновременно контролируя состояние среды (несущую частоту f1). В том случае, когда за время передачи кадра станция не обнаружила состояние коллизии в сети (т. е. частота f1 не изменилась), считается, что данные переданы успешно.

Если при передаче кадра произошла коллизия (т. е. несущая частота изменилась на величины f2(f1), то станция прекращает передавать данные и выдаёт специальную последовательность из 32 битов, которая позволяет всем станциям определить, что произошла коллизия. Затем станция переходит в состояние ожидания на небольшой случайный промежуток времени, по окончании которого она, проверив среду, пытается ещё раз передать по сети свой кадр. Если за 16 попыток станции не удается передать свои данные, то считается, что среда недоступна.

Следует отметить, что коллизия (конфликт) может произойти в том случае, если среду проверяют несколько станций одновременно. Выяснив, что среда свободна, они пытаются передать свои кадры. При одновременной передаче нескольких кадров несущая частота среды изменяется до величины f2, отличной от f1 (передача одного кадра). Коллизию обнаруживают все станции, которые пытаются одновременно передать свои данные. Каждая из этих станций выполняет действия, перечисленные выше.

**Метод доступа и кадры для сетей Token Ring**

Адаптеры Token Ring поддерживают метод доступа Token Ring (маркерное кольцо) и обеспечивают скорости передачи 4 Мбит/с или 16 Мбит/с. Ниже перечислены основные положения этого метода:

* станции подключаются к сети по топологии кольцо,
* все станции, подключённые к сети, могут передавать данные, только получив разрешение на передачу (маркер),
* в любой момент времени только одна станция в сети обладает таким правом.

В сетях Token Ring используются три основных типа кадров (рисунок 2.29):

* Data/Command Frame (кадр управления/данные),
* Token (маркер),
* Abort (кадр сброса).

**Технология Ethernet**

Ethernet – это самый распространенный на сегодняшний день стандарт локальных сетей.

Ethernet – это сетевой стандарт, основанный на экспериментальной сети Ethernet Network, которую фирма Xerox разработала и реализовала в 1975 году.

В 1980 году фирмы DEC, Intel и Xerox совместно разработали и опубликовали стандарт Ethernet версии II для сети, построенной на основе коаксиального кабеля, который стал последней версией фирменного стандарта Ethernet. Поэтому фирменную версию стандарта Ethernet называют стандартом Ethernet DIX, или Ethernet II, на основе которых был разработан стандарт IEEE 802.3.

На основе стандарта Ethernet были приняты дополнительные стандарты: в 1995 году Fast Ethernet (дополнение к IEEE 802.3), в 1998 году Gigabit Ethernet (раздел IEEE 802.3z основного документа), которые во многом не являются самостоятельными стандартами.

Для передачи двоичной информации по кабелю для всех вариантов физического уровня технологии Ethernet, обеспечивающих пропускную способность 10 Мбит/с, используется манчестерский код (рис. 3.9).

**Технология FDDI**

Технология FDDI (Fiber Distributed Data Interface) – оптоволоконный интерфейс распределенных данных ‑ это первая технология локальных сетей, в которой средой передачи данных является волоконно-оптический кабель. Технология появилась в середине 80-х годов.

Технология FDDI во многом основывается на технологии Token Ring, поддерживая метод доступа с передачей маркера.

Сеть FDDI строится на основе двух оптоволоконных колец, которые образуют основной и резервный пути передачи данных между узлами сети. Наличие двух колец – это основной способ повышения отказоустойчивости в сети FDDI, и узлы, которые хотят воспользоваться этим повышенным потенциалом надежности, должны быть подключены к обоим кольцам.

В нормальном режиме работы сети данные проходят через все узлы и все участки кабеля только первичного (Primary) кольца, этот режим назван режимом Thru ‑ «сквозным», или «транзитным». Вторичное кольцо (Secondary) в этом режиме не используется.

В случае какого-либо вида отказа, когда часть первичного кольца не может передавать данные (например, обрыв кабеля или отказ узла), первичное кольцо объединяется со вторичным, вновь образуя единое кольцо. Этот режим работы сети называется Wrap, то есть «свертывание» или «сворачивание» колец. Операция свертывания производится средствами концентраторов и/или сетевых адаптеров FDDI.

**Технология Token Ring**

Сети Token Ring, как и сети Ethernet, характеризует разделяемая среда передачи данных, которая состоит из отрезков кабеля, соединяющих все станции сети в кольцо [. Кольцо рассматривается как общий разделяемый ресурс, и для доступа к нему требуется не случайный алгоритм, как в сетях Ethernet, а детерминированный, основанный на передаче станциям права на использование кольца в определенном порядке. Это право передается с помощью кадра специального формата, называемого маркером, или токеном (token).

Технология Token Ring был разработана компанией IBM в 1984 году, а затем передана в качестве проекта стандарта в комитет IEЕЕ 802, который на ее основе принял в 1985 году стандарт 802.5.

Каждый ПК работает в Token Ring согласно принципу «Ждать маркера, если необходимо послать сообщение, присоединить его к маркеру, когда он будет проходить мимо. Если проходит маркер, снять с него сообщение и отправить маркер дальше».

Сети Token Ring работают с двумя битовыми скоростями ‑ 4 и 16 Мбит/с. Смешение станций, работающих на различных скоростях, в одном кольце не допускается.

Технология Token Ring является более сложной технологией, чем Ethernet. Она обладает свойствами отказоустойчивости. В сети Token Ring определены процедуры контроля работы сети, которые используют обратную связь кольцеобразной структуры ‑ посланный кадр всегда возвращается в станцию-отправитель.

2. Рассчитать эффективную и реальную пропускную способность сети Ethernet 100BaseT при минимальной и максимальной длине кадра.

Служебная информация в кадрах технологии Ethernet составляет 18 байт (без преамбулы и стартового байта), а размер поля данных кадра меняется от 46 до 1500 байт. Сам размер кадра меняется от 46 + 18 = 64 байт до 1500 + 18 = 1518 байт. Поэтому для кадра минимальной длины полезная информация составляет всего лишь 46 / 64 = 0,72 от общей передаваемой информации, а для кадра максимальной длины 1500 / 1518 = 0,99 от общей информации.

Чтобы рассчитать полезную пропускную способность локальной сети для передаваемых кадров максимального и минимального размера, необходимо учесть различную частоту следования кадров передачи данных. Естественно, что, чем меньше размер передаваемых кадров, тем больше таких кадров будет проходить по сети за единицу времени, перенося с собой большее количество служебной информации.

При передаче кадра по локальной сети максимального размера, который вместе с преамбулой имеет длину 1526 байт или 12208 бит, период следования составляет 12 208 bt + 96 bt = 12 304 bt, а частота передаваемых кадров при скорости передачи 100 Мбит/с составляет 1 / 123,04 мкс = 8127 кадр/с.

Так, для передачи кадра минимального размера, который вместе с преамбулой имеет длину 72 байта, или 576 бит, потребуется время, равное 576 bt, а если учесть межкадровый интервал в 96 bt то получим, что период следования кадров составит 672 bt. При скорости передачи в 100 Мбит/с это соответствует времени 6,72 мкс. Тогда частота следования кадров, то есть количество кадров, проходящих по сети за 1 секунду, составит 1/6,72 мкс = 148810 кадр/с.

Рассчитаем полезную пропускную способность сети :

Вp = Vp · 8 · f (бит/с),

где Вp - полезная пропускная способность,

f - частота следования кадров,

Vp - объем полезной информации.

Для кадра минимальной длины (46 байт) теоретическая полезная пропускная способность равна:

Вpmin = Vp · 8 · f= 148 810 кадр/с = 54,76 Мбит/с,

Для кадра максимального размера (1500 байт) полезная пропускная способность сети равна:

Вpmax = Vp · 8 · f =8127 кадр/с = 97,52 Мбит/с.

Таким образом, в сети Fast Ethernet полезная пропускная способность может меняться в зависимости от размера передаваемых кадров от 54,76 до 97,52 Мбит/с.

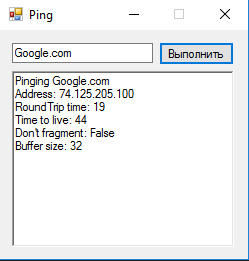
3. Ознакомиться с принципами работы протокола ICMP. Создать приложение, реализующее функции, аналогичные утилите ping с использованием библиотеки icmp.dll

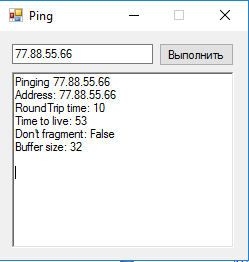
Принцип работы ping-а основан на использовании протокола ICMP - Internet Control Message Protocol (протокол управляющих, или контрольных, сообщений).

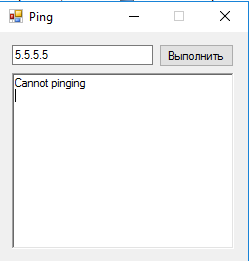
С помощью ICMP хосты в сети обмениваются различной служебной информацией (информацией о смене маршрута, уменьшении скорости передачи, недоступности какого-либо адреса и т.д.) В основе протокола ICMP лежит понятие сообщений. Сообщения ICMP протокола, как правило, оповещают об ошибках, возникающих при обработке датаграмм. ICMP использует основные свойства протокола IP, как если бы он являлся протоколом более высокого уровня. На самом же деле ICMP является составной частью IP.

Одним из типов сообщений протокола является т.н. "эхо-запрос". Получив "эхо-запрос" хост обязан ответить пославшему "эхо-ответом". По сути, "эхо-запрос" и "эхо-ответ" отличаются лишь адресами отправителя и получателя и кодом типа сообщения (тип 8 - "эхо-запрос", тип 0 - "эхо-ответ").

**Результаты тестирования**







192.168.214.210

Текст программы:

**MyForm.cpp**

#include "MyForm.h"

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

[STAThreadAttribute]

int Main(array<System::String ^> ^args)

{

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Project2::MyForm mainForm;

Application::Run(%mainForm);

return 0;

}

**MyForm.h**

#pragma once

#include <iostream>

#include <vector>

namespace Project2 {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Net::NetworkInformation;

using namespace System::Text;

/// <summary>

/// —водка дл¤ MyForm

/// </summary>

public ref class MyForm : public System::Windows::Forms::Form

{

public:

MyForm(void)

{

InitializeComponent();

//

//TODO: добавьте код конструктора

//

}

protected:

/// <summary>

/// ќсвободить все используемые ресурсы.

/// </summary>

~MyForm()

{

if (components)

{

delete components;

}

}

private: System::Windows::Forms::TextBox^ textBox1;

protected:

private: System::Windows::Forms::Button^ button1;

private: System::Windows::Forms::RichTextBox^ richTextBox1;

private:

/// <summary>

/// “ребуетс¤ переменна¤ конструктора.

/// </summary>

System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

/// <summary>

/// ќб¤зательный метод дл¤ поддержки конструктора - не измен¤йте

/// содержимое данного метода при помощи редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->textBox1 = (gcnew System::Windows::Forms::TextBox());

this->button1 = (gcnew System::Windows::Forms::Button());

this->richTextBox1 = (gcnew System::Windows::Forms::RichTextBox());

this->SuspendLayout();

//

// textBox1

//

this->textBox1->Location = System::Drawing::Point(12, 13);

this->textBox1->Name = L"textBox1";

this->textBox1->Size = System::Drawing::Size(141, 20);

this->textBox1->TabIndex = 0;

//

// button1

//

this->button1->Location = System::Drawing::Point(159, 12);

this->button1->Name = L"button1";

this->button1->Size = System::Drawing::Size(75, 23);

this->button1->TabIndex = 1;

this->button1->Text = L"Выполнить";

this->button1->UseVisualStyleBackColor = true;

this->button1->Click += gcnew System::EventHandler(this, &MyForm::button1\_Click);

//

// richTextBox1

//

this->richTextBox1->Location = System::Drawing::Point(12, 41);

this->richTextBox1->Name = L"richTextBox1";

this->richTextBox1->Size = System::Drawing::Size(222, 176);

this->richTextBox1->TabIndex = 2;

this->richTextBox1->Text = L"";

//

// MyForm

//

this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->ClientSize = System::Drawing::Size(248, 229);

this->Controls->Add(this->richTextBox1);

this->Controls->Add(this->button1);

this->Controls->Add(this->textBox1);

this->MaximizeBox = false;

this->MaximumSize = System::Drawing::Size(264, 268);

this->MinimumSize = System::Drawing::Size(264, 268);

this->Name = L"MyForm";

this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;

this->Text = L"MyForm";

this->Load += gcnew System::EventHandler(this, &MyForm::MyForm\_Load);

this->ResumeLayout(false);

this->PerformLayout();

}

#pragma endregion

private: System::Void MyForm\_Load(System::Object^ sender, System::EventArgs^ e) {

}

private: System::Void button1\_Click(System::Object^ sender, System::EventArgs^ e) {

Ping ^ pingSender = gcnew Ping;

PingOptions ^ options = gcnew PingOptions();

// Use the default Ttl value which is 128,

// but change the fragmentation behavior.

options->DontFragment = true;

// Create a buffer of 32 bytes of data to be transmitted.

String^ data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";

array<Byte>^buffer = Encoding::ASCII->GetBytes(data);

int timeout = 120;

String^ adress = "google.com";

if (textBox1->Text->Length > 0)

adress = textBox1->Text;

PingReply ^ reply = pingSender->Send(adress, timeout, buffer, options);

if (reply->Status == IPStatus::Success)

{

richTextBox1->Text += "Pinging " + adress + "\n";

richTextBox1->Text += "Address: " + reply->Address->ToString() + "\n";

richTextBox1->Text += "RoundTrip time: " + reply->RoundtripTime + "\n";

richTextBox1->Text += "Time to live: " + reply->Options->Ttl + "\n";

richTextBox1->Text += "Don't fragment: " + reply->Options->DontFragment + "\n";

richTextBox1->Text += "Buffer size: " + reply->Buffer->Length + "\n\n";

}

else

{

richTextBox1->Text += "Cannot pinging" + "\n";

}

}

};

}