CHƯƠNG 3: XÂY DỰNG ỨNG DỤNG MẠNG

3.1. Giao thức ICMP

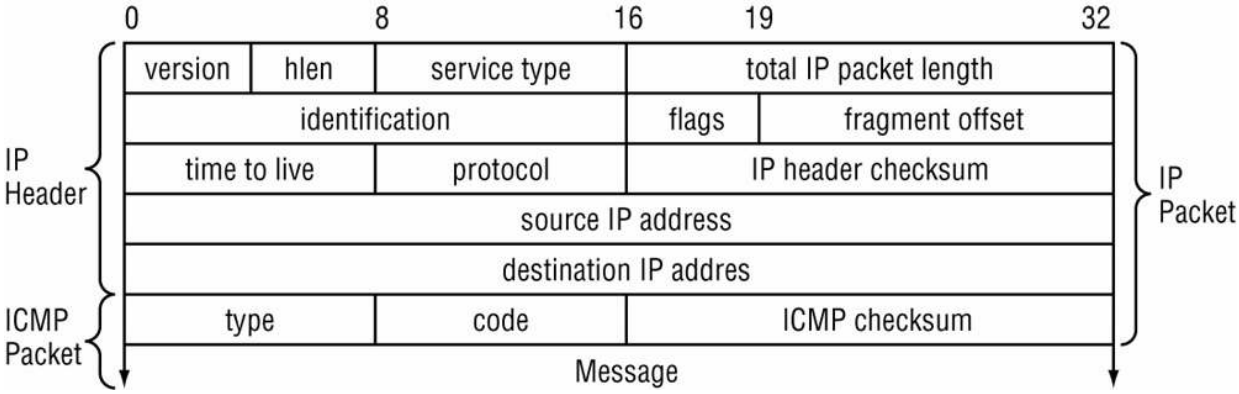
Giới thiệu giao thức ICMP (Internetwork Control Message Protocol)

- Giao thức ICMP hoạt động trên layer 2 - Internetwork trong mô hình TCP/IP hoặc layer 3 - Network trong mô hình OSI

Cho phép kiểm tra và xác định lỗi của Layer 3 Internetwork trong mô hình TCP/IP bằng cách định nghĩa ra các loại thông điệp có thể sử dụng để xác định xem mạng hiện tại có thể truyền được gói tin hay không.

Trong thực tế, ICMP cần các thành phần của mọi gói tin IP để có thể hoạt động được.

Cấu trúc của gói tin IP và ICMP



+ Type: có thể là một query hay một lỗi
+ Code: Xác định đây là loại query hay thông điệp lỗi
+ Checksum: Kiểm tra và sửa lỗi cho dữ liệu ICMP
+ Message: Tuỳ thuộc vào Type và Code

3.1.1. Sử dụng Raw Socket

Gói tin ICMP không sử dụng TCP hoặc UDP nên chúng ta không thể sử dụng các lớp được hỗ trợ như TcpClient hay UdpClient mà phải sử dụng một Raw Socket

Muốn tạo Raw Socket khi tạo ra Socket bạn sử dụng SocketType.Raw, giao thức ICMP

Tạo Raw Socket như sau

Socket sock = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);

Raw Socket Format

Value	Description
Ggp	Gateway-to-Gateway Protocol
Icmp	Internet Control Message Protocol

Idp	IDP Protocol
Igmp	Internet Group Management Protocol
IP	A raw IP packet
Ipx	Novell IPX Protocol
ND	Net Disk Protocol
Pup	Xerox PARC Universal Protocol (PUP)
Raw	A raw IP packet
Spx	Novell SPX Protocol
SpxII	Novell SPX Version 2 Protocol
Unknown	An unknown protocol
Unspecified	An unspecified protocol

- Gửi gói dữ liệu Raw
  - ICMP là giao thức không hướng kết nối
  - Sử dụng phương thức SendTo() của lớp Socket để gửi
  - Cổng trong giao thức ICMP không quan trọng

```
IPEndPoint iep = new IPEndPoint(IPAddress.Parse("192.168.1.2"), 0);
sock.SendTo(packet, iep);
```

- Nhận gói dữ liệu Raw
  - Sử dụng phương thức ReceiveForm cửa lớp Socket
  - Dữ liệu nhận về là một gói tin IP chúng ta phải tách ra để lấy gói tin ICMP

Raw Socket không tự động định dạng gói tin ICMP cho chúng ta. Chúng ta phải tự làm

Data Variable	**Size**	**Type**
Type	1 byte	Byte
Code	1 byte	Byte
Checksum	2 bytes	Unsigned 16-bit integer
Message	multibyte	Byte array

Định nghĩa lớp và phương thức khởi tạo mặc định

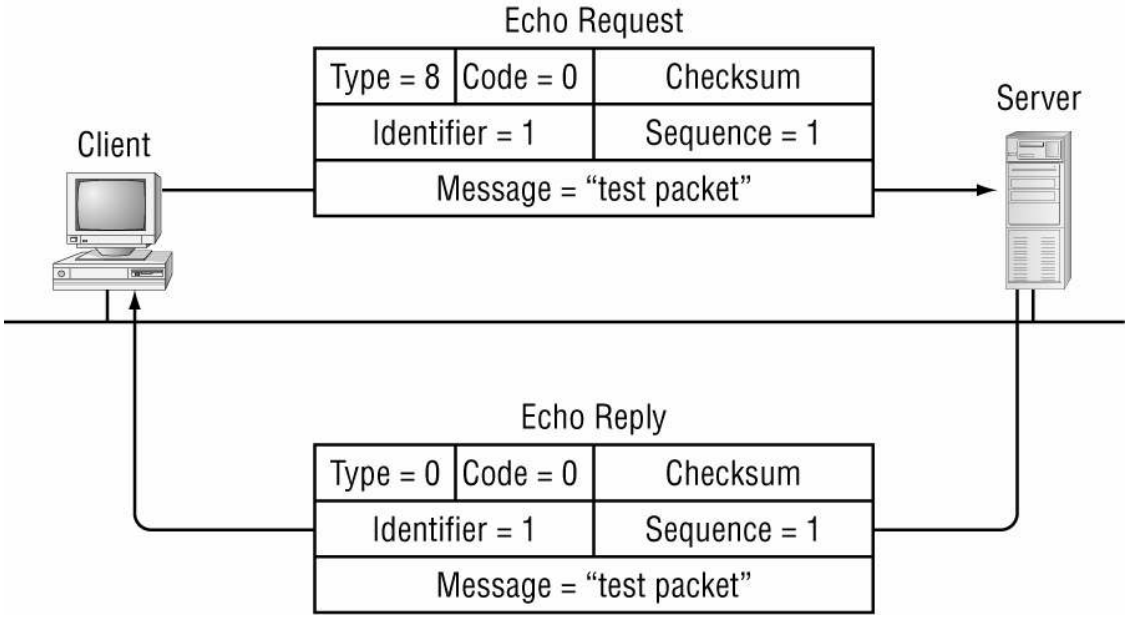
```
class ICMP {
   public byte Type;
   public byte Code;
   public UInt16 Checksum;
   public int Messagesize;
   public byte[] Message = new byte[1024];
   public ICMP() {
   }
}
```

- Tạo ra gói tin ICMP

```
ICMP packet = new ICMP();
```

```
            packet.Type = 0x08;
            packet.Code = 0x00;
            packet.Checksum = 0;
public ICMP(byte[] data, int size) {
        Type = data[20];
        Code = data[21];
        Checksum = BitConverter.ToUInt16(data, 22);
        MessageSize = size - 24;
        Buffer.BlockCopy(data, 24, Message, 0, MessageSize);
    }
public byte[] getBytes() {
        byte[] data = new byte[MessageSize + 9];
        Buffer.BlockCopy(BitConverter.GetBytes(Type), 0, data, 0, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Code), 0, data, 1, 1);
        Buffer.BlockCopy(BitConverter.GetBytes(Checksum), 0, data, 2, 2);
        Buffer.BlockCopy(Message, 0, data, 4, MessageSize);
        return data;
    }
public UInt16 getChecksum() {
        UInt32 chcksm = 0;
        byte[] data = getBytes();
        int packetsize = MessageSize + 8;
        int index = 0;
        while (index < packetsize) {
            chcksm += Convert.ToUInt32(BitConverter.ToUInt16(data, index));
            index += 2;
        }
        chcksm = (chcksm >> 16) + (chcksm & 0xffff);
        chcksm += (chcksm >> 16);
        return (UInt16)(~chcksm);
    }
```

3.1.2. Sử dụng giao thức ICMP và Raw Socket để xây dựng chương trình Ping




```
class Program {
   static void Main(string[] args) {
      byte[] data = new byte[1024];
      int recv;
      Socket host = new Socket(AddressFamily.InterNetwork, SocketType.Raw,
         ProtocolType.Icmp);
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse(argv[0]), 0);
      EndPoint ep = (EndPoint)iep;
      ICMP packet = new ICMP();
      packet.Type = 0x08;
      packet.Code = 0x00;
      packet.Checksum = 0;
      Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 0, 2);
      Buffer.BlockCopy(BitConverter.GetBytes((short)1), 0, packet.Message, 2, 2);
      data = Encoding.ASCII.GetBytes("test packet");
      Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);

packet.MessageSize = data.Length + 4;
      int packetsize = packet.MessageSize + 4;
      UInt16 chcksum = packet.getChecksum();
      packet.Checksum = chcksum;
      host.SetSocketOption(SocketOptionLevel.Socket,
         SocketOptionName.ReceiveTimeout, 3000);
      host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
      try {
         data = new byte[1024];
         recv = host.ReceiveFrom(data, ref ep);
      } catch (SocketException) {
         Console.WriteLine("No response from remote host");
         return;
      }
ICMP response = new ICMP(data, recv);
      Console.WriteLine("response from: {0}", ep.ToString());
      Console.WriteLine(" Type {0}", response.Type);
      Console.WriteLine(" Code: {0}", response.Code);
      int Identifier = BitConverter.ToInt16(response.Message, 0);
```

```
      int Sequence = BitConverter.ToInt16(response.Message, 2);
      Console.WriteLine(" Identifier: {0}", Identifier);
      Console.WriteLine(" Sequence: {0}", Sequence);
      string stringData = Encoding.ASCII.GetString(response.Message, 4,
         response.MessageSize - 4);
      Console.WriteLine(" data: {0}", stringData);
      host.Close();
   }
}
```

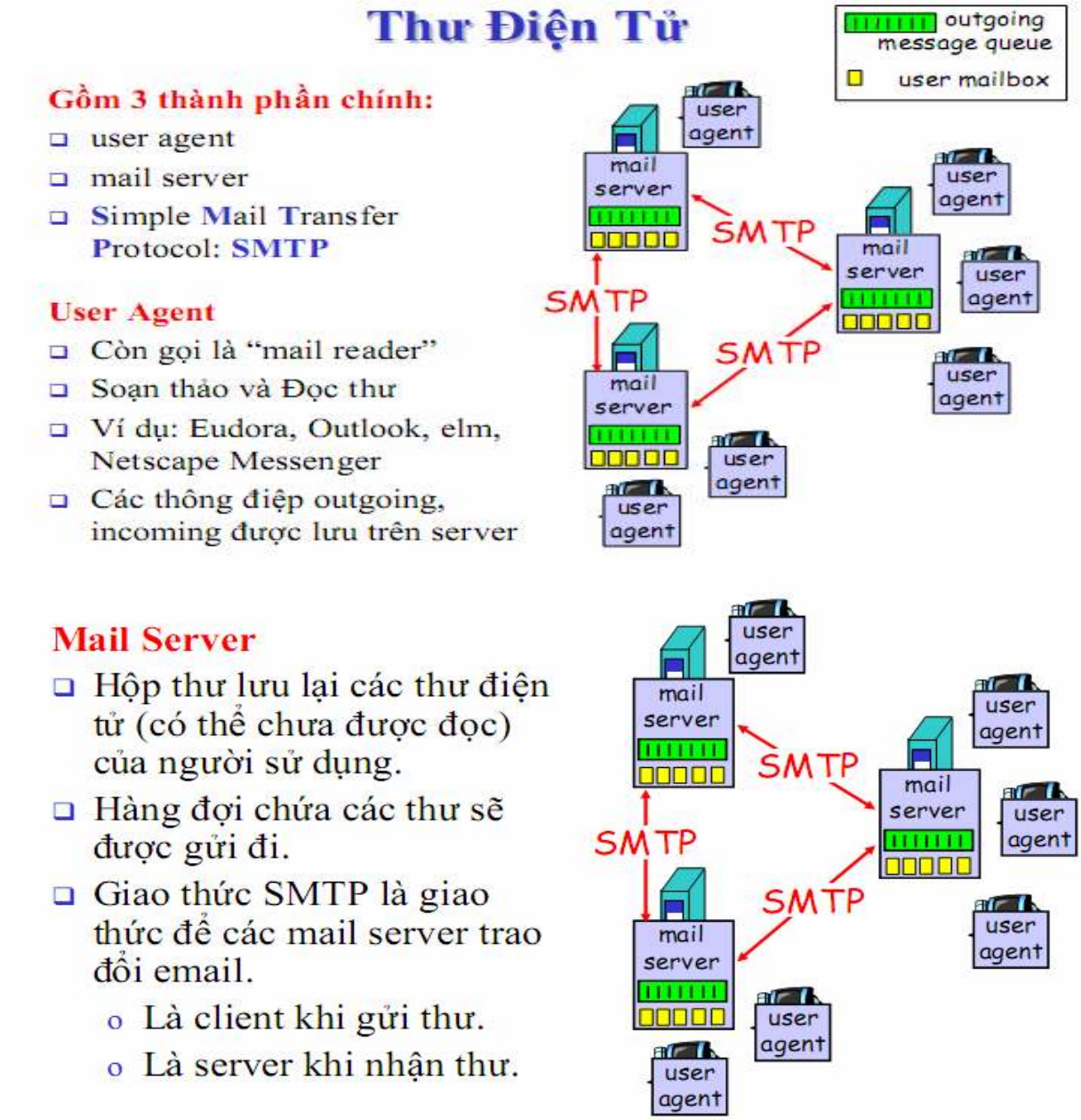
3.1.3. Sử dụng giao thức ICMP và Raw Socket để xây dựng chương trình TraceRoute

```
class TraceRoute {
   public static void Main(string[] argv) {
      byte[] data = new byte[1024];
      int recv, timestart, timestop;
      Socket host = new Socket(AddressFamily.InterNetwork,
           SocketType.Raw, ProtocolType.Icmp);
      IPHostEntry iphe = Dns.Resolve(argv[0]);
      IPEndPoint iep = new IPEndPoint(iphe.AddressList[0], 0);
      EndPoint ep = (EndPoint)iep;
      ICMP packet = new ICMP();
      packet.Type = 0x08;
      packet.Code = 0x00;
      packet.Checksum = 0;
      Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 0, 2);
      Buffer.BlockCopy(BitConverter.GetBytes(1), 0, packet.Message, 2, 2);
      data = Encoding.ASCII.GetBytes("test packet");
Buffer.BlockCopy(data, 0, packet.Message, 4, data.Length);
      packet.MessageSize = data.Length + 4;
      int packetsize = packet.MessageSize + 4;
      UInt16 chcksum = packet.getCchecksum();
      packet.Checksum = chcksum;
      host.SetSocketOption(SocketOptionLevel.Socket,
              SocketOptionName.ReceiveTimeout, 3000);
      int badcount = 0;
      for (int i = 1; i < 50; i++) {
         host.SetSocketOption(SocketOptionLevel.IP,
              SocketOptionName.IpTimeToLive, i);
         timestart = Environment.TickCount;
         host.SendTo(packet.getBytes(), packetsize, SocketFlags.None, iep);
try {
            data = new byte[1024];
            recv = host.ReceiveFrom(data, ref ep);
            timestop = Environment.TickCount;
            ICMP response = new ICMP(data, recv);
            if (response.Type == 11)
               Console.WriteLine("hop {0}: response from {1}, {2}ms",
                 i, ep.ToString(), timestop - timestart);
            if (response.Type == 0) {
               Console.WriteLine("{0} reached in {1} hops, {2}ms.",
                 ep.ToString(), i, timestop - timestart);
               break;
```

```
            }
badcount = 0;
        } catch (SocketException) {
            Console.WriteLine("hop {0}: No response from remote host", i);
            badcount++;
            if (badcount == 5) {
                Console.WriteLine("Unable to contact remote host");
                break;
            }
        }
      }
      host.Close();
   }
}
```

3.2. Giao thức SMTP, POP3

3.2.1. Cơ bản về hệ thống Mail và giao thức SMTP, POP3



*** Giao thức SMTP**

Một số lệnh cơ bản của giao thức SMTP:

Lệnh	Mô tả
HELO	Hello. Sử dụng để xác định người gửi điện. Lệnh này này đi kèm với tên của host gửi điện. Trong ESTMP (extended protocol), thì lệnh này sẽ là EHLO.
MAIL	Khởi tạo một giao dịch gửi thư. Nó kết hợp "from" để xác định người gửi thư.
RCPT	Xác định người nhận thư.
DATA	Thông báo bắt đầu nội dung thực sự của bức điện (phần thân của thư). Dữ liệu được mã thành dạng mã 128-bit ASCII và nó được kết thúc với một dòng đơn chứa dấu chấm (.).
RSET	Huỷ bỏ giao dịch thư
VRFY	Sử dụng để xác thực người nhận thư.
NOOP	Nó là lệnh "no operation" xác định không thực hiện hành động gì
QUIT	Thoát khỏi tiến trình để kết thúc
SEND	Cho host nhận biết rằng thư còn phải gửi đến đầu cuối khác.

Một số lệnh không yêu cầu phải có được xác định bằng RFC 821

SOML	Send or mail. Báo với host nhận thư rằng thư phải gửi đến đấu cuối khác hoặc hộp thư.
SAML	Send and mail. Nói với host nhận rằng bức điện phải gửi tới người dùng đầu cuối và hộp thư.
EXPN	Sử dụng mở rộng cho một mailing list.
HELP	Yêu cầu thông tin giúp đỡ từ đầu nhận thư.
TURN	Yêu cầu để host nhận giữ vai trò là host gửi thư.

- Mã trạng thái của SMTP

Khi một MTA gửi một lệnh SMTP tới MTA nhận thì MTA nhận sẽ trả lời với một mã trạng thái để cho người gửi biết đang có việc gì xẩy ra đầu nhận. Và dưới đây là bảng mã trạng thái của SMTP theo tiêu chuẩn RFC 821. Mức độ của trạng thái được

xác định bởi số đầu tiên của mã (5xx là lỗi nặng, 4xx là lỗi tạm thời, 1xx–3xx là hoạt động bình thường).

- Một số mã trạng thái của SMTP

211 Tình trạng hệ thống, hay reply giúp đỡ hệ thống

214 Thông điệp giúp đỡ.

220 <domain> dịch vụ sẵn sàng

221 <domain> dịch vụ đóng kênh giao chuyển

250 Hành động mail yêu cầu OK, hoàn thành

251 User không cục bộ; sẽ hướng đến <forward-path>

354 Khởi động việc nhập mail; kết thúc với <CLRF>. <CLRF>

421 <domain> dịch vụ không sử dụng được, đóng kênh giao chuyển

450 Không lấy hành động mail yêu cầu; mailbox không hiệu lực

451 Không nhận hành động được yêu cầu; lưu trữ của hệ thống không đủ.

500 Lỗi cú pháp; không chấp nhận lệnh

501 Lỗi cú pháp trong tham số hay đối số

502 Lệnh không được cung cấp

503 Dòng lệnh sai

504 Tham số của dòng lệnh không được cung cấp

550 Không nhận hành động được yêu cầu ; mailbox không hiệu lực

[như mailbox không tìm thấy hay không truy cập được]

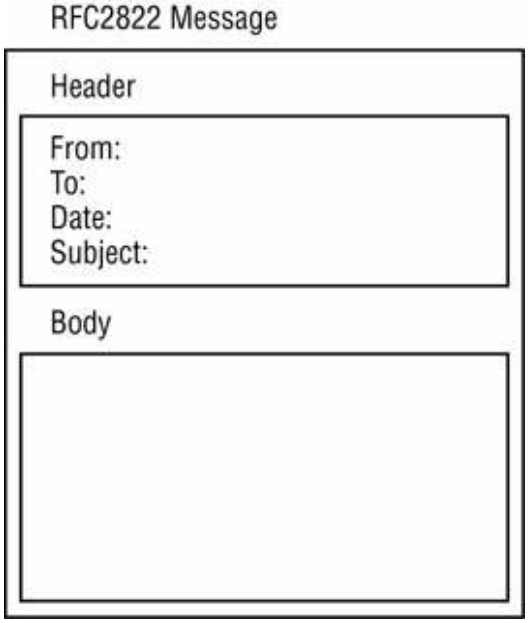
551 User không cục bộ; vui lòng thử <forward-path>

552 Bỏ qua hành động mà mail yêu cầu, vượt quá chỉ định lưu trữ

554 Không nhận hành động được yêu cầu; tên mailbox không được

chấp nhận. [như sai cú pháp mailbox] giao chuyển sai.

- Định dạng của một bức thư thông thường không có phần đính kèm như sau:



* Giao thức POP3

Giao thức dùng để lấy thư, POP3 Server lắng nghe trên cổng 110, mô tả trong RFC 1939

- Một số lệnh của POP3

USER Xác định username

PASS Xác định password

STAT Yêu cầu về trạng thái của hộp thư như số lượng thư và độ lớn của thư

LIST Hiện danh sách của thư

RETR Nhận thư

DELE Xoá một bức thư xác định

NOOP Không làm gì cả

RSET Khôi phục lại như thư đã xoá (rollback)

QUIT Thực hiện việc thay đổi và thoát ra

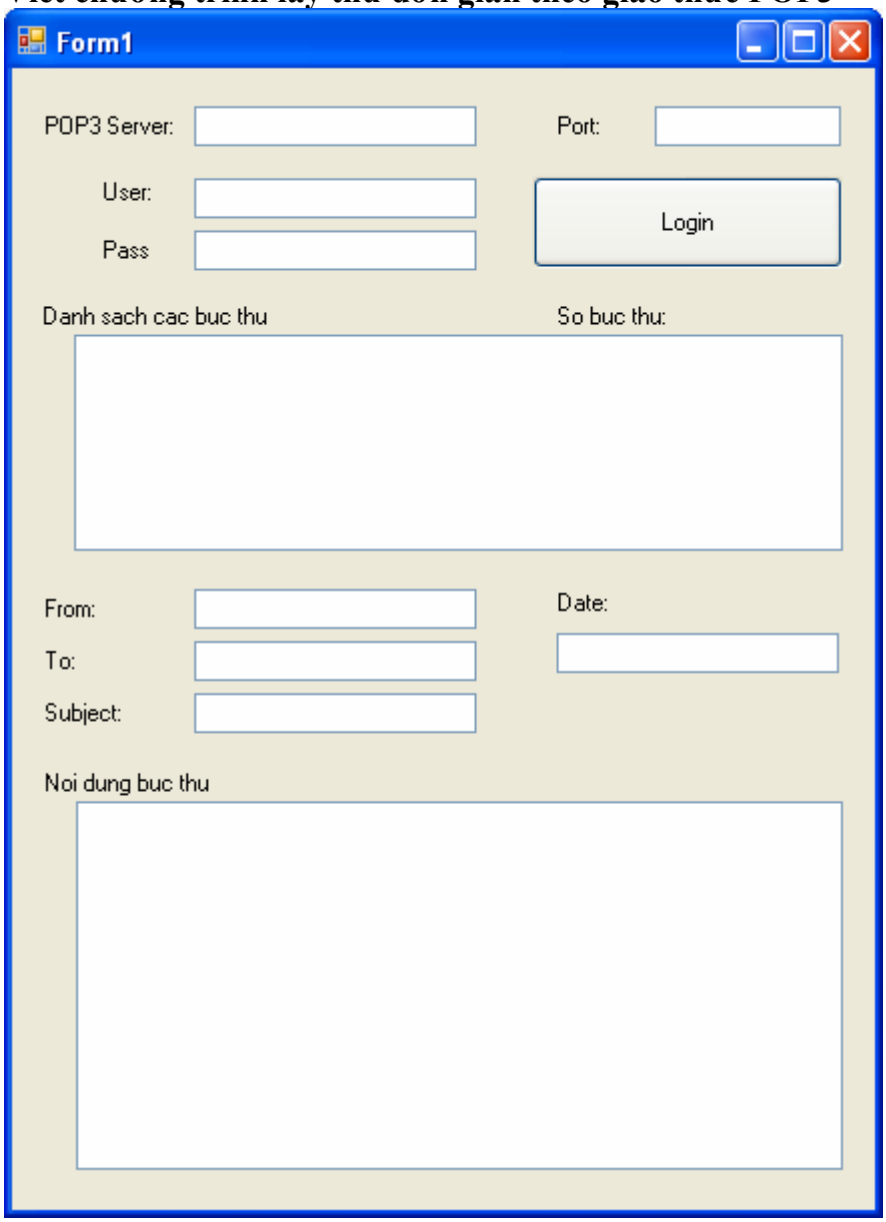
3.2.2. Cài đặt SMTP, POP3 Client/Server

Viết chương trình gửi Mail đơn giản theo giao thức SMTP

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
class Program {
   static void Main(string[] args) {
      string nguoigui, nguoinhan, tieude,body;
      string diachimaychu;
      int portmaychu;
      Console.Write("Nhap di chu SMTP Server:");
      diachimaychu = Console.ReadLine();
      Console.Write("Nhap cong cua may chu:");
      portmaychu = int.Parse(Console.ReadLine());
      Console.Write("Nhap dia chi nguoi gui:");
      nguoigui = Console.ReadLine();
      Console.Write("Nhap dia chi nguoi nhan:");
      nguoinhan = Console.ReadLine();
      Console.Write("Nhap tieu de buc thu:");
      tieude = Console.ReadLine();
      Console.Write("Nhap noi dung buc thu:");
      body= Console.ReadLine();
      //Tao Endpoit cua may chu
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse(diachimaychu), portmaychu);
      TcpClient client = new TcpClient();
      client.Connect(iep);
      string Data = "Helo";
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      sw.WriteLine(Data);
      sw.Flush();
      //Doc thong bao tu Server gui ve va xu ly neu can thiet
      Console.WriteLine(sr.ReadLine());
```

```
        //Gui dia chi nguyoi gui
        Data = "MAIL FROM: <" + nguoigui + ">";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Gui dia chi nguyoi gui
        Data = "RCPT TO: <" + nguoinhan + ">";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Bat dau gui noi dung buc thu
        Data = "Data";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Gui noi dung buc thu
        Data = "SUBJECT:" + tieude + "\r\n" + body + "\r\n" + "." + "\r\n";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        //Ngat ket noi
        Data = "QUIT";
        sw.WriteLine(Data);
        sw.Flush();
        //Doc thong bao tu Server gui ve va xu ly neu can thiet
        Console.WriteLine(sr.ReadLine());
        sr.Close();
        sw.Close();
        client.Close();
        Console.ReadLine();
    }
}
```

Viết chương trình lấy thư đơn giản theo giao thức POP3




```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace MyPop3 {
   public partial class Form1 : Form {
      TcpClient popclient;
      StreamReader sr;
      StreamWriter sw;
      public Form1() {
         InitializeComponent();
         CheckForIllegalCrossThreadCalls = false;
```

```
    }

    private void btLogin_Click(object sender, EventArgs e) {
        IPEndPoint iep = new IPEndPoint(IPAddress.Parse(txtPOP.Text),
int.Parse(txtPort.Text));
        popclient = new TcpClient();
        popclient.Connect(iep);
        sr = new StreamReader(popclient.GetStream());
        sw = new StreamWriter(popclient.GetStream());
        sr.ReadLine();
        string data = "";
        data = "User " + txtUser.Text;
        sw.WriteLine(data);
        sw.Flush();
        sr.ReadLine();
        data = "PASS " + txtPass.Text;
        sw.WriteLine(data);
        sw.Flush();
        sr.ReadLine();
        data = "LIST";
        sw.WriteLine(data);
        sw.Flush();
        lstHeader.Items.Clear();
        string s = sr.ReadLine();
        char[] ch = { ' ' };
        string[] tam = s.Split(ch);
        //MessageBox.Show("so buc thu la:" + tam[1]);
        while ((s = sr.ReadLine()) != ".") {
            lstHeader.Items.Add(s);
        }
    }

    private void lstHeader_SelectedIndexChanged(object sender, EventArgs e) {
        int i = lstHeader.SelectedIndex + 1;
        //Lay buc thu ve va tien hanh phan tich
        string data = "RETR " + i.ToString();
        sw.WriteLine(data);
        sw.Flush();
        string s;
        //MessageBox.Show(sr.ReadLine());
        //Lay phan header
        while ((s = sr.ReadLine().Trim()) != null) {
            //MessageBox.Show(s);
            if (s.Length == 0) break;
            if (s.ToUpper().StartsWith("DATE")) {
                DateTime dt=DateTime.Parse(s.Substring(5, s.Length - 5));
                txtDate.Text = dt.ToShortDateString() +" " +dt.ToLongTimeString();
            }
            if (s.ToUpper().StartsWith("FROM"))
```

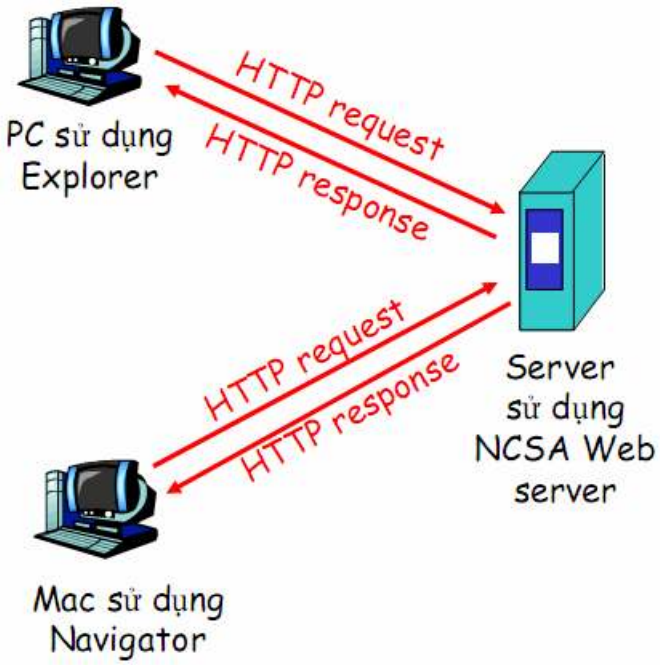
```
                txtFrom.Text = s.Substring(5, s.Length - 5);
            if (s.ToUpper().StartsWith("TO"))
                txtTo.Text = s.Substring(3, s.Length - 3);
            if (s.ToUpper().StartsWith("SUBJECT"))
                txtSubject.Text = s.Substring(8, s.Length - 8);
        }
        //Lay phan body
        textBox4.Clear();
        //MessageBox.Show("Lay body");
        while (!sr.EndOfStream) {
            s = sr.ReadLine().Trim();
            if (s.Equals(".")) break;
            textBox4.Text += s + "\r\n";
        }
        //MessageBox.Show("Het noi dung buc thu");
    }
  }
}
```

3.3. Giao thức HTTP

3.3.1. Cơ bản về giao thức HTTP

HTTP (Hypertext Transfer Protocol) giao thức truyền siêu văn bản. HTTP là giao thức tầng ứng dụng cho Web. Nó hoạt động theo mô hình client/server.

- Client: browser yêu cầu, nhận, hiển thị các đối tượng Web.
- Server: Web server gửi các đối tượng



Hai phiên bản của giao thức HTTP hiện được phổ biến là HTTP1.0 được đặc tả trong RFC 1945 và HTTP1.1 được đặc tả trong RFC 2068.

HTTP là giao thức "không trạng thái" server không lưu lại các yêu cầu của client.

HTTP sử dụng giao thức TCP của tầng giao vận. Các bước tiến hành từ khi client kết nối tới server sau đó gửi và nhận kết quả từ server gửi về như sau:

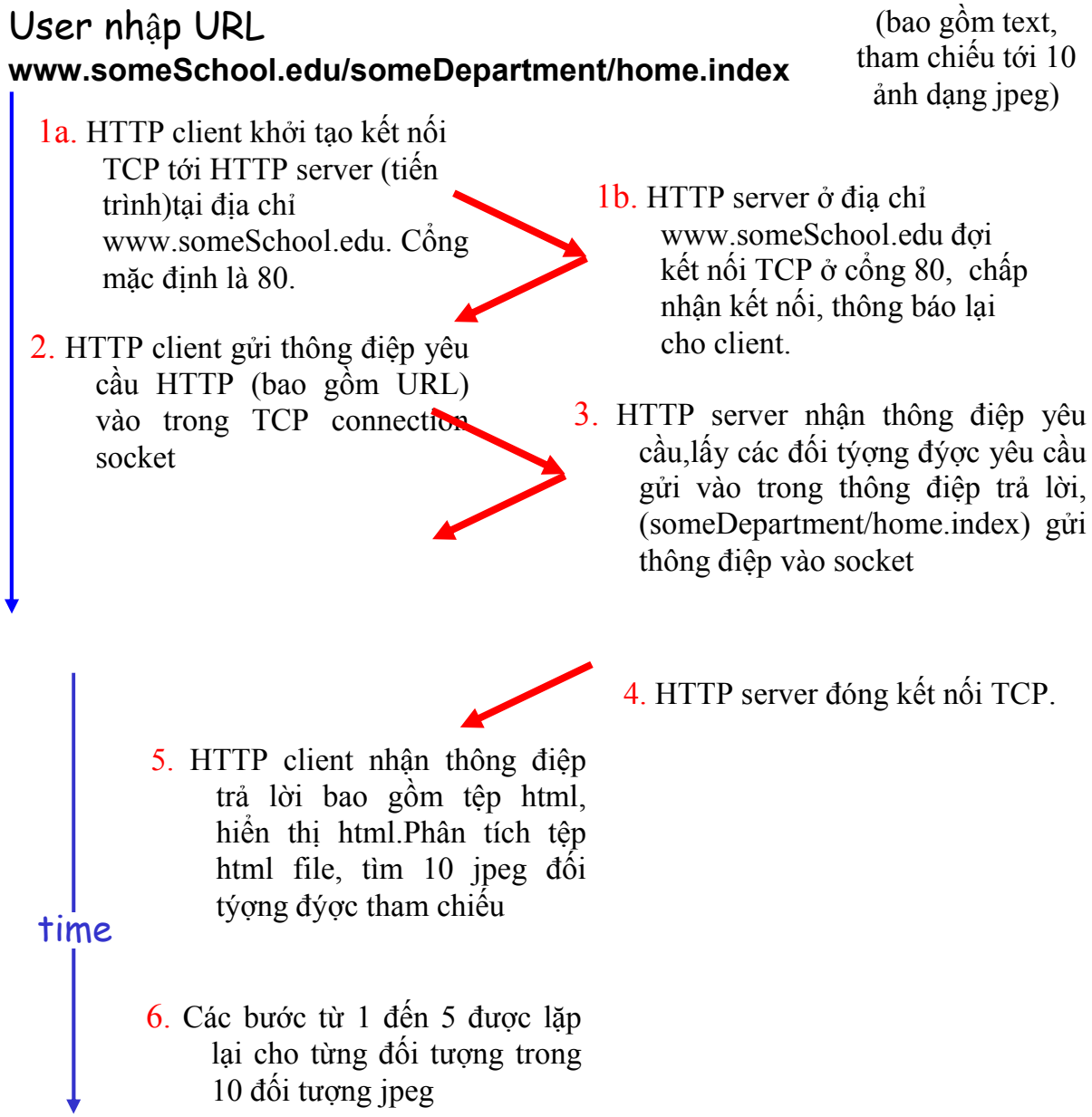
+ client khởi tạo kết nối TCP (tạo socket) với server, qua cổng 80

+ server chấp nhận kết nối TCP từ client

+ Các thông điệp HTTP (thông điệp tầng ứng dụng) được trao đổi giữa browser (HTTP client) và Web server (HTTP server)

+ Đóng kết nối TCP.

Chúng ta xem một ví dụ về quá trình trao đổi giữa browser và Web server như sau:



Có hai kiểu thông điệp HTTP là yêu cầu (Request) và trả lời (Response). Các thông điệp được định dạng kiểu mã ASCII.

Định dạng thông điệp yêu cầu HTTP

Dòng yêu cầu (lệnh GET, POST, HEAD)

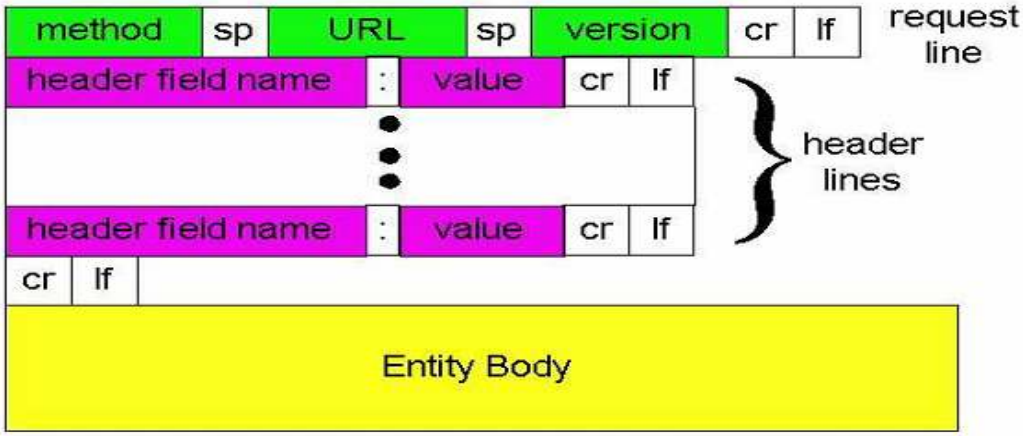
Những dòng header

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(CR,LF)

CR,LF,kí hiệu kết thúc thông điệp

Định dạng Thông điệp Yêu cầu HTTP



Định dạng thông điệp trả lời HTTP

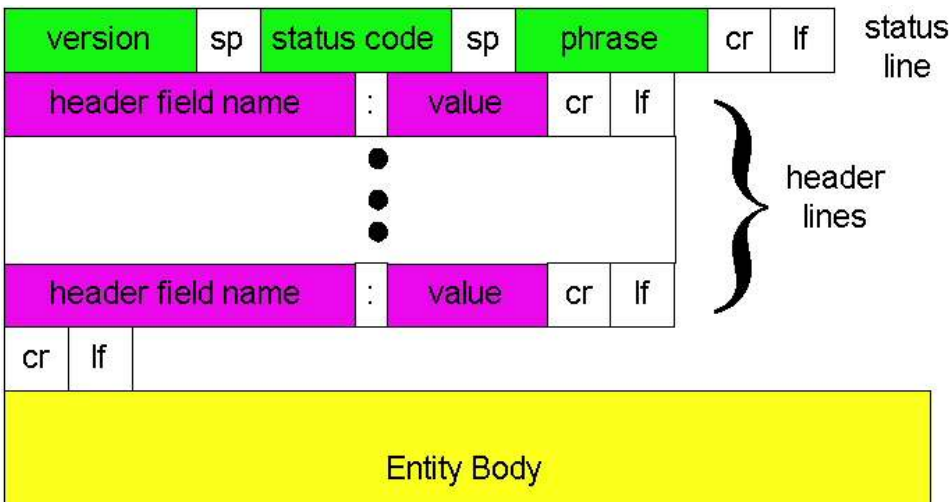
Dòng trạng thái (mã trạng thái)

Những dòng header

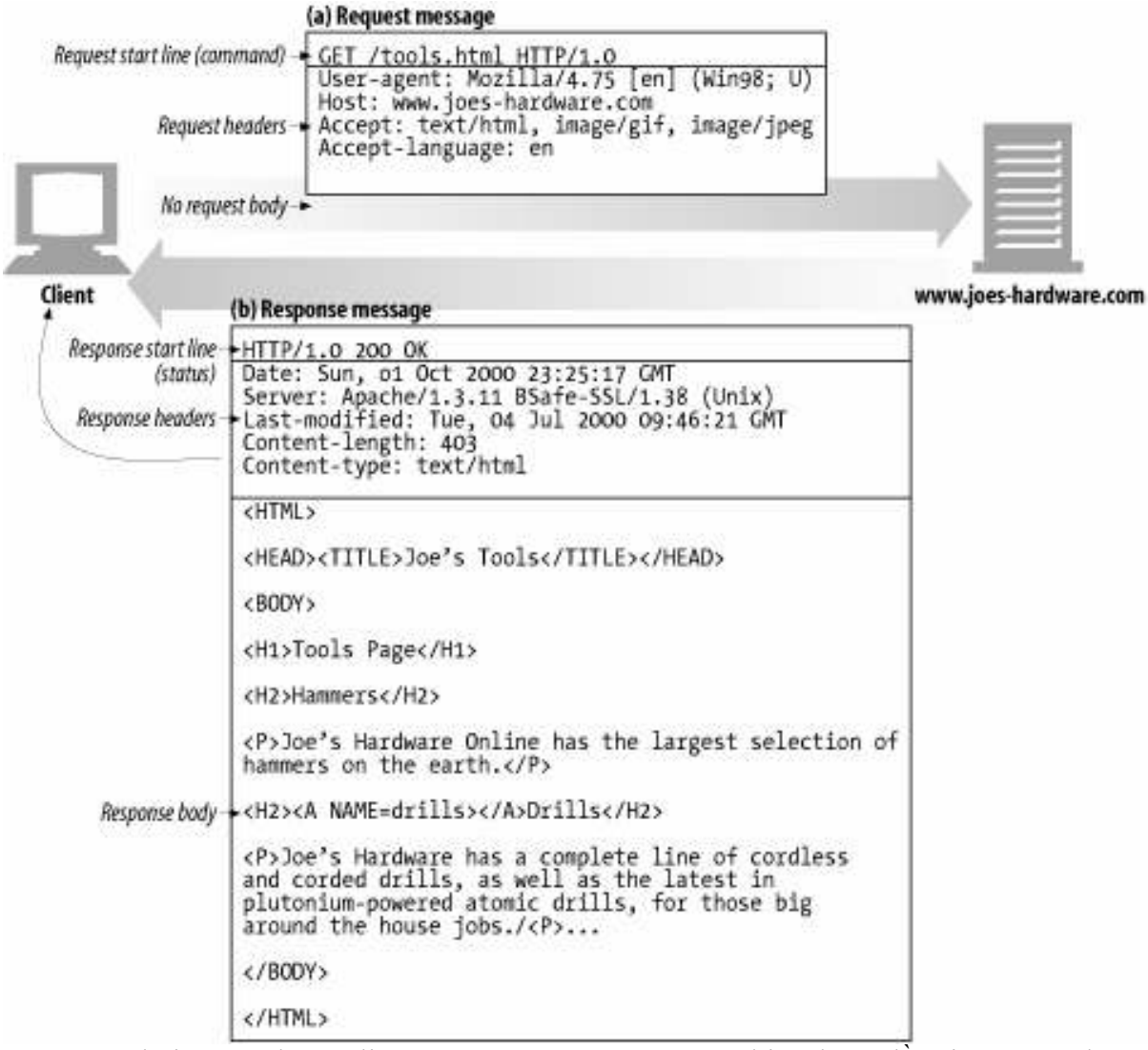
```
HTTP/1.0 200 OK
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

dữ liệu,e.g., tệp html đýợc yêu cầu



Quá trình trao đổi giữa Browser và Web Server có thể được minh họa như hình sau:



Mã trạng thái trong thông điệp HTTP Response: Được ghi ở dòng đầu tiên trong thông điệp response từ server về client.

Một số mã thường gặp:

200 OK

Yêu cầu thành công,các đối tượng được yêu cầu ở phần sau thông điệp.

301 Moved Permanently

Đối tượng được yêu cầu đã được chuyển và địa chỉ mới của đối tượng được đặt trong trường Location:

400 Bad Request

Server không hiểu được thông điệp yêu cầu

404 Not Found

Tài liệu được yêu cầu không có trong server

505 HTTP Version Not Supported

Server không hỗ trợ version của giao thức HTTP.

- Để kiểm tra các lệnh của HTTP bên phía Client chúng ta có thể thực hiện như sau:

+ Telnet tới Web server

`telnet www.eurecom.fr 80` Tạo kết nối TCP ở cổng 80 (cổng mặc định cho HTTP server) at www.eurecom.fr.

+ Lệnh GET trong thông điệp HTTP

`GET /~ross/index.html HTTP/1.0` Gửi thông điệp yêu cầu lấy tệp Index.html trong thư mục ~ross về client

+ Xem thông điệp response được gửi về từ Server

- Gõ các lệnh khác để kiểm tra.

3.3.2. Cài đặt HTTP Client/Server

a. Chương trình HTTP Client

```
using System;
using System.IO;
using System.Net.Sockets;
using System.Windows.Forms;

namespace HttpView {
   public partial class MainForm : Form
   {
      [STAThread]
      public static void Main(string[] args){
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new MainForm());
      }
      public MainForm(){
         //
         // The InitializeComponent() call is required for Windows Forms designer
support.
         //
         InitializeComponent();

         //
         // TODO: Add constructor code after the InitializeComponent() call.
         //
      }
      // Gửi yêu cầu và nhận về phản hồi từ web server
      void BtGoClick(object sender, EventArgs e){
         //Thêm tiền tố http:// nếu không có
         if (txtAddress.Text.StartsWith("http://", StringComparison.OrdinalIgnoreCase)
== false)
            txtAddress.Text = "http://" + txtAddress.Text;
         TcpClient client = new TcpClient();
         try{
            client.Connect(GetHost(txtAddress.Text), GetPort(txtAddress.Text));
         }
         catch{
```

```
            rtfBody.Text = "Không thể kết nối đến server";
            return;
        }
        StreamWriter OutStream = new StreamWriter(client.GetStream());
        StreamReader InpStream = new StreamReader(client.GetStream());
        // Gửi đi yêu cầu bằng phương thức GET
        OutStream.WriteLine("GET " + txtAddress.Text + " HTTP/1.0");
        OutStream.WriteLine("Content-Type:text/html");
        OutStream.WriteLine("Content-Language:en");
        OutStream.WriteLine();
        OutStream.Flush();
        //Lấy phần Header
        rtfHeader.Text = "";
        string s;
        while (null != (s = InpStream.ReadLine())){
            if (s.Equals("")) break;
            rtfHeader.Text += s;
        }
        //Lấy phần Body
        rtfBody.Text = "";
        int c;
        while (true){
            c = InpStream.Read();
            if (c == -1) break;
            rtfBody.Text += (char)c;
        }
        client.Close();
    }
    // Khi gõ Enter thì gửi đi yêu cầu
    void TxtAddressKeyPress(object sender, KeyPressEventArgs e){
        if ((int)e.KeyChar == 13)
            btGo.PerformClick();
    }
    // Lấy về tên máy trong URL
    internal string GetHost(string url){
        url = url.ToLower();
        Uri tmp = new Uri(url);
        return tmp.Host;
    }
    // Lấy về số hiệu cổng trong URL
    internal int GetPort(string url){
        Uri tmp = new Uri(url);
        return tmp.Port;
    }
  }
}
```

b. Chương trình HTTP Server

```
using System;
using System.IO;
```

```
using System.Net;
using System.Net.Sockets;

public class SimpleWebServer {
	public void StartListening(int port) {
		IPEndPoint LocalEndPoint = new IPEndPoint(IPAddress.Any, 8080);
		TcpListener server = new TcpListener(LocalEndPoint);
		server.Start();
		while (true) {
			TcpClient client = server.AcceptTcpClient(); // Cho doi ket noi
			processClientRequest(client);
		}
	}
	//Xử lý yêu cầu của mỗi máy khách
	private void processClientRequest(TcpClient client) {
		Console.WriteLine("May khach su dung cong: " +

(client.Client.RemoteEndPoint as IPEndPoint).Port + "; Dia chi IP may khach: " +

client.Client.RemoteEndPoint.ToString() + "\n");
		Console.WriteLine("Yeu cau: ");
		readRequest(client);
		sendResponse(client);
		client.Close();
	}
	//Đọc phần header của gói dữ liệu gửi từ máy khách
	private void readRequest(TcpClient client) {
		StreamReader request = new StreamReader(client.GetStream());
		String nextLine;

		while (null != (nextLine = request.ReadLine())) {
			if (nextLine.Equals("")) {
				break;
			} else {
				Console.WriteLine(nextLine);
			}
		}
		Console.WriteLine("----------------------------------------");
	}

	//Phản hồi yêu cầu cho máy khách
	private void sendResponse(TcpClient client) {
		StreamWriter response = new StreamWriter(client.GetStream());
		response.Write(_mainPage);
		response.Flush();
	}
	//Mội phản hồi về trình duyệt
	private const String _mainPage =
		"HTTP/1.0 200 OK\n" +
```

```
                "Content-type: text/html\n" +
                "Connection: close\n" +
                "\n<HTML>\n" +
                "<HEAD>\n" +
                "<TITLE>Hello World</TITLE>\n" +
                "</HEAD>\n" +
                "<BODY> <a href = \"localhost\"> Hello World </a></BODY>\n" +
                "</HTML> \n\n";
}

class TestWebServer {
        public static void Main(String[] args) {
                SimpleWebServer webServer1 = new SimpleWebServer();
                webServer1.StartListening(8080);
        }
}
```

3.4. Giao thức FTP

3.4.1. Cơ bản về giao thức FTP

Giao thức FTP là một giao thức trao đổi file khá phổ biến

Khái niệm và hoạt động cơ bản của FTP:

Những người phát triển giao thức FTP cần phải cân bằng nhu cầu giữa việc phát triển một bộ giao thức vừa có nhiều chức năng và vừa đơn giản trong triển khai. Do đó, FTP không đơn giản như "đứa em trai" của nó – là giao thức TFTP. Hoạt động của giao thức này có thể chia ra thành nhiều thành phần nhỏ, hoạt động cùng nhau để thực hiện các công việc như khởi tạo kết nối, truyền thông tin điều khiển và truyền lệnh.

Nội dung mà bài viết này hướng tới là đưa tới người đọc những khái niệm quan trọng nhất đằng sau giao thức FTP, cũng như giải thích tổng quan về nguyên lý hoạt động của nó.

1 - Mô hình hoạt động của FTP, các thành phần trong giao thức, và các thuật ngữ cơ bản

Giao thức FTP được mô tả một cách đơn giản thông qua mô hình hoạt động của FTP. Mô hình này chỉ ra các nguyên tắc mà một thiết bị phải tuân theo khi tham gia vào quá trình trao đổi file, cũng như về hai kênh thông tin cần phải thiết lập giữa các thiết bị đó. Nó cũng mô tả các thành phần của FTP được dùng để quản lý các kênh này ở cả hai phía – truyền và nhận. Do đó, mô hình này tạo cho ta một khởi điểm lý tưởng để xem xét hoạt động của FTP ở mức khái quát.

Tiến trình Server-FTP và User-FTP

FTP là một giao thức dạng client/server truyền thống, tuy nhiên thuật ngữ client thông thường được thay thế bằng thuật ngữ user – người dùng – do thực tế là người sử dụng mới là đối tượng trực tiếp thao tác các lệnh FTP trên máy clients. Bộ phần mềm FTP được cài đặt trên một thiết bị được gọi là một tiến trình. Phần mềm FTP được cài đặt trên máy Server được gọi là tiến trình Server-FTP, và phần trên máy client được gọi là tiến trình User-FTP.

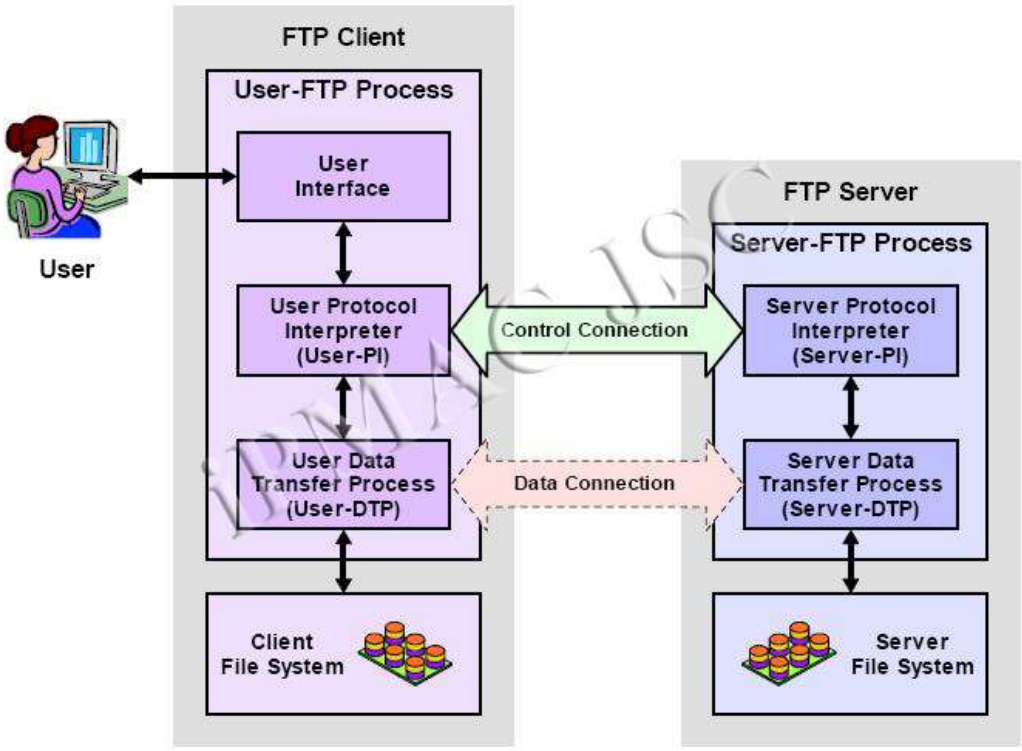
Kênh điều khiển và kênh dữ liệu trong FTP

Một khái niệm cốt yếu mà ta cần phải nắm về FTP là: mặc dù giao thức này sử dụng kết nối TCP, nhưng nó không chỉ dùng một kênh TCP như phần lớn các giao thức truyền thông khác. Mô hình FTP chia quá trình truyền thông giữa bộ phận Server với bộ phận client ra làm hai kênh logic:
- Kênh điều khiển: đây là kênh logic TCP được dùng để khởi tạo một phiên kết nối FTP. Nó được duy trì xuyên suốt phiên kết nối FTP và được sử dụng chỉ để truyền các thông tin điều khiển, như các lệnh và các hồi đáp trong FTP. Nó không được dùng để truyền file
- Kênh dữ liệu: Mỗi khi dữ liệu được truyền từ server tới client, một kênh kết nối TCP nhất định lại được khởi tạo giữa chúng. Dữ liệu được truyền đi qua kênh kết nối này – do đó nó được gọi là kênh dữ liệu. Khi file được truyền xong, kênh này được ngắt. Việc sử dụng các kênh riêng lẻ như vậy tạo ra sự linh hoạt trong việc truyền truyền dữ liệu – mà ta sẽ thấy trong các phần tiếp theo. Tuy nhiên, nó cũng tạo cho FTP độ phức tạp nhất định.

Các tiến trình và thuật ngữ trong FTP

Do các chức năng điều khiển và dữ liệu sử dụng các kênh khác nhau, nên mô hình hoạt động của FTP cũng chia phần mềm trên mỗi thiết bị ra làm hai thành phần logic tương ứng với mỗi kênh. Thành phần Protocol Interpreter (PI) là thành phần quản lý kênh điều khiển, với chức năng phát và nhận lệnh. Thành phần Data Transfer Process (DTP) có chức năng gửi và nhận dữ liệu giữa phía client với server. Ngoài ra, cung cấp cho tiến trình bên phía người dùng còn có thêm thành phần thứ ba là giao diện người dùng FTP - thành phần này không có ở phía server.

Do đó, có hai tiến trình xảy ra ở phía server, và ba tiến trình ở phía client. Các tiến trình này được gắn với mô hình FTP để mô tả chi tiết hoạt động của giao thức FTP. Dưới đây là hình đối chiếu các tiến trình vào trong mô hình FTP:



Các tiến trình phía server:

Các tiến trình phía server bao gồm hai giao thức:

- Server Protocol Interpreter (Server-PI): chịu trách nhiệm quản lý kênh điều khiển trên server. Nó lắng nghe yêu cầu kết nối hướng tới từ users trên cổng dành riêng. Khi kết nối đã được thiết lập, nó sẽ nhận lệnh từ phía User-PI, trả lời lại, và quản lý tiến trình truyền dữ liệu trên server.
- Server DataTransfer Process (Server-DTP): làm nhiệm vụ gửi hoặc nhận file từ bộ phận User-DTP. Server-DTP vừa làm nhiệm thiết lập kết nối kênh dữ liệu và lắng nghe một kết nối kênh dữ liệu từ user. Nó tương tác với server file trên hệ thống cục bộ để đọc và chép file.

Các tiến trình phía client:

- User Protocol Interpreter (User-PI): chịu trách nhiệm quản lý kênh điều khiển phía client. Nó khởi tạo phiên kết nối FTP bằng việc phát ra yêu cầu tới phía Server-PI. Khi kết nối đã được thiết lập, nó xử lý các lệnh nhận được trên giao diện người dùng, gửi chúng tới Server-PI, và nhận phản hồi trở lại. Nó cũng quản lý tiến trình User-DTP.
- User Data Transfer Process (User-DTP): là bộ phận DTP nằm ở phía người dùng, làm nhiệm vụ gửi hoặc nhận dữ liệu từ Server-DTP. User-DTP có thể thiết lập hoặc

lắng nghe yêu cầu kết nối kênh dữ liệu trên server. Nó tương tác với thiết bị lưu trữ file phía client.
- User Interface: cung cấp giao diện xử lý cho người dùng. Nó cho phép sử dụng các lệnh đơn giản hướng người dùng, và cho phép người điều khiển phiên FTP theo dõi được các thông tin và kết quả xảy ra trong tiến trình.

2 - Thiết lập kênh điều khiển và chứng thực người dùng trong FTP:

Mô hình hoạt động của FTP mô tả rõ các kênh dữ liệu và điều khiển được thiết lập giữa FTP client và FTP server. Trước khi kết nối được sử dụng để thực sự truyền file, kênh điều khiển cần phải được thiết lập. Một tiến trình chỉ định sau đó được dùng để tạo kết nối và tạo ra phiên FTP lâu bền giữa các thiết bị để truyền files.

Như trong các giao thức client/server khác, FTP server tuân theo một luật passive trong kênh điều khiển. Bộ phận Server Protocol Interpreter (Server-PI) sẽ lắng nghe cổng TCP dành riêng cho kết nối FTP là cổng 21. Phía User-PI sẽ tạo kết nối bằng việc mở một kết nối TCP từ thiết bị người dùng tới server trên cổng đó. Nó sử dụng một cổng bất kỳ làm cổng nguồn trong phiên kết nối TCP.

Khi TCP đã được cài đặt xong, kênh điều khiển giữa các thiết bị sẽ được thiết lập, cho phép các lệnh được truyền từ User-PI tới Server-PI, và Server-PI sẽ đáp trả kết quả là các mã thông báo. Bước đầu tiên sau khi kênh đã đi vào hoạt động là bước đăng nhập của người dùng (login sequence). Bước này có hai mục đích:
- Access Control - Điều khiển truy cập: quá trình chứng thực cho phép hạn chế truy cập tới server với những người dùng nhất định. Nó cũng cho phép server điều khiển loại truy cập như thế nào đối với từng người dùng.
- Resource Selection - Chọn nguồn cung cấp: Bằng việc nhận dạng người dùng tạo kết nối, FTP server có thể đưa ra quyết định sẽ cung cấp những nguồn nào cho người dùng đã được nhận dạng đó.

Trình tự truy cập và chứng thực FTP

Quy luật chứng thực trong FTP khá đơn giản, chỉ là cung cấp username/password. Trình tự của việc chứng thực như sau:
1 - người dùng gửi một username từ User-PI tới Server-PI bằng lệnh USER. Sau đó password của người dùng được gửi đi bằng lệnh PASS.
2 - Server kiểm tra tên người dùng và password trong database người dùng của nó. Nếu người dùng hợp lệ, server sẽ gửi trả một thông báo tới người dùng rằng phiên kết

nối đã đ.c mở. Nếu người dùng không hợp lệ, server yêu cầu người dùng thực hiện lại việc chứng thực. Sau một số lần chứng thực sai nhất định, server sẽ ngắt kết nối.

Giả sử quá trình chứng thực đã thành công, server sau đó sẽ thiết lập kết nối để cho phép từng loại truy cập đối với người dùng được cấp quyền. Một số người dùng chỉ có thể truy cập vào một số file nhất định, hoặc vào một số loại file nhất định. Một số server có thể cấp quyền cho một số người dùng đọc và viết lên server, trong khi chỉ cho phép đọc đối với những người dùng khác. Người quản trị mạng có thể nhờ đó mà đáp ứng đúng các nhu cầu truy cập FTP.

Một khi kết nối đã được thiết lập, server có thể thực hiện các lựa chọn tài nguyên dựa vào nhận diện người dùng. Ví dụ: trên một hệ thống nhiều người dùng, người quản trị có thể thiết lập FTP để khi có bất cứ người dùng nào kết nối tới, anh ta sẽ tự động được đưa tới "home directory" của chính anh ta. Lệnh tùy chọn ACCT (account) cũng cho phép người dùng chọn một tài khoản cá nhân nào đó nếu như anh ta có nhiều hơn một tài khoản.

Mở rộng về bảo mật FTP

Giống như phần lớn các giao thức cũ, phương pháp đăng nhập đơn giản của FTP là một sự kế thừa từ những giao thức ở thời kỳ đầu của Internet. Ngày nay, nó không còn bảo đảm tính an toàn cần thiết trên môi trường Internet toàn cầu vì username và password được gửi qua kênh kết nối điều khiển dưới dạng clear text. Điều này làm cho các thông tin đăng nhập có thể bị nghe lén. Chuẩn RFC 2228 về các phần mở rộng cho bảo mật FTP đã định ra thêm nhiều tùy chọn chứng thực và mã hóa phức tạp cho những ai muốn tăng thêm mức độ an toàn vào trong phần mềm FTP của họ.

3 - Quản lý kênh dữ liệu FTP, kết nối kênh dữ liệu dạng chủ động (mặc định) và bị động cùng với việc sử dụng cổng

Kênh điều khiển được tạo ra giữa Server-PI và User-PI sử dụng quá trình thiết lập kết nối và chứng thực được duy trì trong suốt phiên kết nối FTP. Các lệnh và các hồi đáp được trao đổi giữa bộ phận PI (Protocol Interpreter) qua kênh điều khiển, nhưng dữ liệu thì không.

Mỗi khi cần phải truyền dữ liệu giữa server và client, một kênh dữ liệu cần phải được tạo ra. Kênh dữ liệu kết nối bộ phận User-DTP với Server-DTP. Kết nối này cần thiết

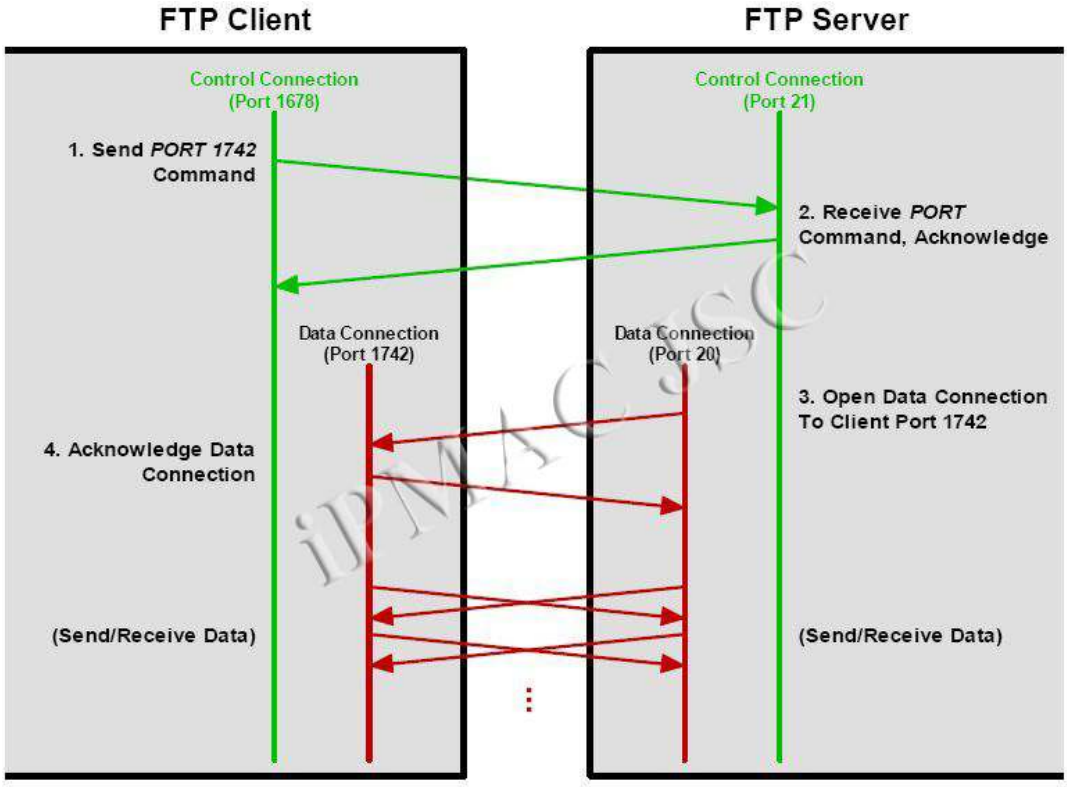
cho cả hoạt động chuyển file trực tiếp (gửi hoặc nhận một file) cũng như đối với việc truyền dữ liệu ngầm, như là yêu cầu một danh sách file trong thư mục nào đó trên server.
Chuẩn FTP chỉ định hai phương thức khác nhau để tạo ra kênh dữ liệu. Khác biệt chính của hai phương thức đó là ở mặt thiết bị: phía client hay phía server là phía đã đưa ra yêu cầu khởi tạo kết nối. Điều này nghe qua có vẻ khá đơn giản, nhưng kỳ thực nó lại khá quan trọng.

Kết nối kênh dữ liệu dạng chủ động

Phương thức đầu tiên đôi khi còn được gọi là kết nối kênh dữ liệu dạng thông thường (vì nó là phương pháp mặc định) và đôi khi được gọi là kết nối dạng chủ động (để đối chiếu với dạng kết nối bị động mà ta sẽ xét ở phần sau). Trong dạng kết nối này, phía Server-DTP khởi tạo kệnh dữ liệu bằng việc mở một cổng TCP cho phía User-DTP. Phía server sử dụng cổng được dành riêng, là cổng 20 cho kênh dữ liệu. Trên máy client, một giá trị cổng được chọn theo mặc định chính là cổng được sử dụng đối với kênh điều khiển, tuy nhiên phía client sẽ luôn chọn hai cổng riêng biệt cho hai kênh này.

Giả sử phía User-PI thiết lập một kết nối điều khiển từ cổng bất kỳ của nó là 1678 tới cổng điều khiển trên server là cổng 21. Khi đó, để tạo một kênh dữ liệu cho việc truyền dữ liệu, phía Server-PI sẽ báo cho phía Server-DTP khởi tạo một kênh kết nối TCP từ cổng 20 tới cổng 1678 của phía client. Sau khi phía client chấp nhận kênh được khởi tạo, dữ liệu sẽ được truyền đi.

Thực tế, việc sử dụng cùng một cổng cho cả kênh dữ liệu và kênh điều khiển không phải là một ý hay, nó làm cho hoạt động của FTP trở nên phức tạp. Do đó, phía client nên chỉ định sử dụng một cổng khác bằng việc sử dụng lệnh PORT trước khi truyền dữ liệu. Ví dụ: giả sử phía client chỉ định cổng 1742 với lệnh PORT. Phía Server-DTP sau đó sẽ tạo ra một kết nối từ cổng 20 của nó tới cổng 1742 phía client thay vì cổng 1678 như mặc định. Quá trình này được mô tả trong hình dưới đây.



Thông thường, đối với kênh dữ liệu FTP, phía server sẽ khởi tạo việc truyền dữ liệu bằng cách mở kết nối dữ liệu tới client.

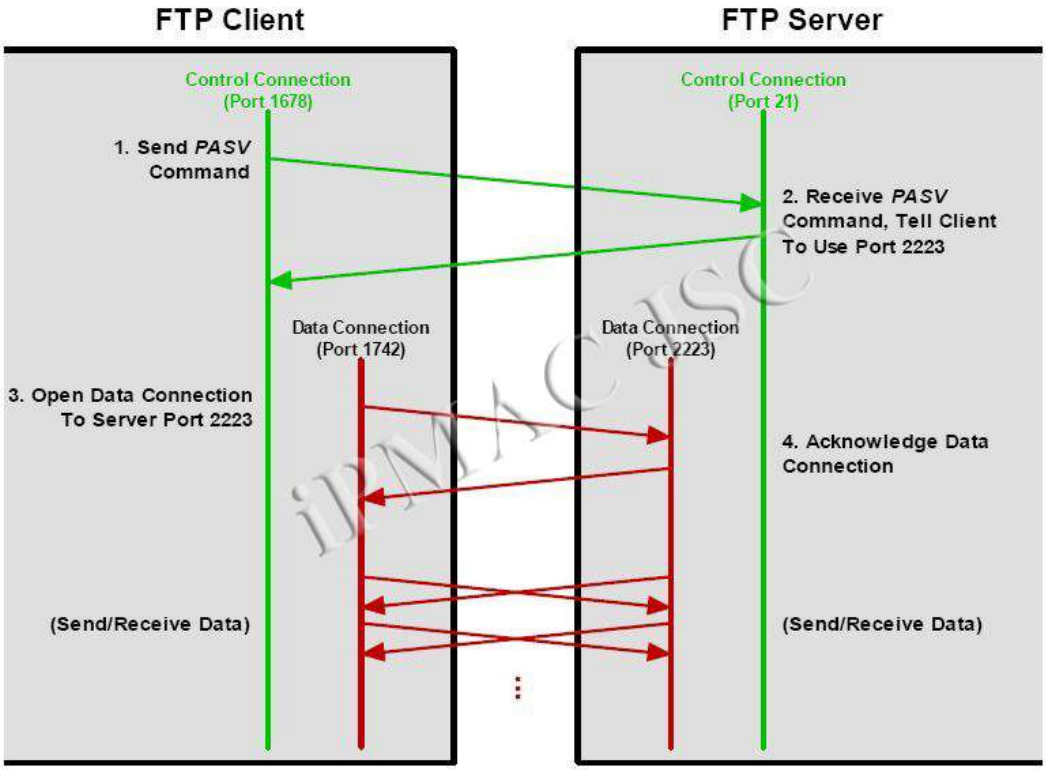
Trong trường hợp trên, phía client trước tiên sẽ đưa ra lệnh PORT để yêu cầu server sử dụng cổng 1742. Sau đó, server sẽ mở kết nối kênh dữ liệu từ cổng 20 mặc định của nó tới cổng 1742 phía client. Dữ liệu sau đó sẽ được truyền giữa các thiết bị qua các cổng này.

Kết nối kênh dữ liệu dạng bị động

Phương pháp kế tiếp được gọi là kết nối dữ liệu dạng bị động. Phía client sẽ nhận server là phía bị động, làm nhiệm vụ chấp nhận một yêu cầu kết nối kênh dữ liệu được khởi tạo từ phía client. Server trả lời lại phía client với địa chỉ IP cũng như địa chỉ cổng mà nó sẽ sử dụng. Phía Server-DTP sau đó sẽ lắng nghe một kết nối TCP từ phía User-DTP trên cổng này.

Mặc định, phía client sử dụng cùng một cổng đối với cả hai kênh điều khiển và dữ liệu như trong trường hợp kết nối chủ động ở trên. Tuy nhiên, ở đây, một lần nữa phía client có thể chọn sử dụng một giá trị cổng khác cho kênh dữ liệu. Ta sẽ xét lại ví dụ ở trên một lần nữa, với cổng điều khiển phía client là 1678 tới cổng 21 phía server.

Nhưng lần này truyền dữ liệu theo phương thức kết nối bị động, như mô tả trong hình dưới đây:



Phía client sẽ sử dụng lệnh PASV để yêu cầu server rằng nó muốn dùng phương thức điều khiển dữ liệu bị động. Phía Server-PI sẽ trả lời lại phía client với một giá trị cổng mà client sẽ sử dụng, từ cổng 2223 trên nó. Sau đó phía Server PI sẽ hướng cho phía Server-DTP lắng nghe trên cổng 2223. Phía User-PI cũng sẽ hướng cho phía User-DTP tạo một phiên kết nối từ cổng 1742 phía client tới cổng 2223 phía server. Sau khi Server chấp nhận kết nối này, dữ liệu bắt đầu được truyền đi.

Các vấn đề về tính hiệu quả và tính bảo mật trong việc chọn một phương thức kết nối

Vấn đề phía nào là phía khởi tạo kết nối kênh dữ liệu đưa ra một câu hỏi: sự khác nhau giữa hai phương thức là gì? Điều này cũng giống như việc hỏi ai đã thực hiện một cuộc điện thoại nội bộ. Câu trả lời là sự bảo mật. Việc FTP sử dụng nhiều hơn một kết nối TCP có thể giải quyết các vấn đề về phần mềm cũng như về phần cứng mà người dùng cần phải có để đảm bảo sự an toàn cho hệ thống của họ.

Khi xem xét việc gì sẽ xảy ra trong trường hợp kênh dữ liệu chủ động như trong ví dụ phía trên:

Đối với phía client, có một kênh kết nối điều khiển được thiết lập từ cổng 1678 client tới cổng 21 server. Nhưng kênh dữ liệu lại được khởi tạo từ phía server. Do đó, client sẽ nhận được một yêu cầu kết nối tới cổng 1678 (hoặc cổng nào khác). Một số client sẽ nghi ngờ về việc nhận được những kết nối tới như vậy, vì trong tình huống thông thường, client mới là phía khởi tạo kết nối chứ không phải đáp trả kết nối. Do các kênh kết nối TCP hướng tới có thể mang theo những mối đe dọa nhất định, một số client có thể sẽ ngăn chặn các luồng kết nối hướng tới bằng việc sử dụng tường lửa.

Tại sao người ta lại không làm cho phía client luôn chấp nhận kết nối từ một chỉ số port được dùng trong kênh điều khiển? Vấn đề ở đây là vì client thường dùng các cổng khác nhau cho mỗi phiên kết nối bằng việc sử dụng câu lệnh PORT. Và tại sao điều này lại được thực hiện? Vì theo luật TCP: sau khi một kết nối được đóng lại , có một khoảng thời gian trống trước khi cổng đó có thể được sử dụng lại – điều này để ngăn ngừa tình trạng các phiên kết nối liên tiếp bị lẫn với nhau. Điều này sẽ tạo ra độ trễ khi gửi nhiều file – do đó phía client thường dùng các giá trị cổng khác nhau cho mỗi kết nối. Điều này rất hiệu quả nhưng cũng dẫn tới việc firewall của client sẽ hỏi có chấp nhận phiên kết nối tới với nhiều giá trị cổng không ổn định hay không.

Việc dùng kết nối kiểu kênh gián tiếp sẽ giảm thiểu vấn đề này một cách hiệu quả. Phần lớn các tường lửa có nhiều vấn đề liên quan tới kết nối hướng về với các giá trị cổng bất kỳ, hơn là gặp vấn đề với các kết nối hướng đi. Ta có thể xem chi tiết hơn về vấn đề này trong chuẩn RFC 1579. Chuẩn này khuyến nghị rằng phía client nên sử dụng kết nối kiểu bị động làm dạng mặc định thay vì sử dụng kiểu kết nối dạng chủ động cùng với lệnh PORT, để ngăn chặn tình trạng block theo cổng. Tất nhiên, phương thức kết nối kiểu bị động không hoàn toàn giải quyết được vấn đề, chúng chỉ đẩy vấn đề về phía server mà thôi. Phía server, giờ đây phải đối mặt với việc có nhiều kênh kết nối hướng về trên hàng loạt các cổng khác nhau. Tuy nhiên việc xử lý các vấn đề bảo mật trên một nhóm nhỏ server vẫn dễ hơn nhiều so với việc phải đối mặt với một lượng lớn các vấn đề từ nhiều client. FTP server phải được cấu hình chấp nhận phương thức truyền bị động từ client, do đó cách thông thường để thiết lập trên server là thiết lập chấp nhận một số cổng kết nối hướng về trên server trong khi vẫn khóa các yêu cầu kết nối hướng về trên các cổng khác.

4 - Các phương thức truyền dữ liệu trong FTP

Khi kênh dữ liệu đã được thiết lập xong giữa Server-DTP với User-DTP, dữ liệu sẽ được truyền trực tiếp từ phía client tới phía server, hoặc ngược lại, dựa theo các lệnh

được sử dụng. Do thông tin điều khiển được gửi đi trên kênh điều khiển, nên toàn bộ kênh dữ liệu có thể được sử dụng để truyền dữ liệu. (Tất nhiên, hai kênh logic này được kết hợp với nhau ở lớp dưới cùng với tất cả các kết nối TCP/UDP khác giữa hai thiết bị, do đó điều này không hẳn đã cải thiện tốc độ truyền dữ liệu so với khi truyền trên chỉ một kênh – nó chỉ làm cho hai việc truyền dữ liệu và điều khiển trở nên độc lập với nhau mà thôi)
FTP có ba phương thức truyền dữ liệu, nêu lên cách mà dữ liệu được truyền từ một thiết bị tới thiết bị khác trên một kênh dữ liệu đã được khởi tạo, đó là: stream mode, block mode, và compressed mode

Stream mode

Trong phương thức này, dữ liệu được truyền đi dưới dạng các byte không cấu trúc liên tiếp. Thiết bị gửi chỉ đơn thuần đầy luồng dữ liệu qua kết nối TCP tới phía nhận. Không có một trường tiêu đề nhất định được sử dụng trong phương thức này làm cho nó khá khác so với nhiều giao thức gửi dữ liệu rời rạc khác. Phương thức này chủ yếu dựa vào tính tin cậy trong truyền dữ liệu của TCP. Do nó không có cầu trúc dạng header, nên việc báo hiệu kết thúc file sẽ đơn giản được thực hiện việc phía thiết bị gửi ngắt kênh kết nối dữ liệu khi đã truyền xong.
Trong số ba phương thưc, stream mode là phương thức được sử dụng nhiều nhất trong triển khai FTP thực tế. Có một số lý do giải thích điều đó. Trước hết, nó là phương thức mặc định và đơn giản nhất, do đó việc triển khai nó là dễ dàng nhất. Thứ hai, nó là phương pháp phổ biến nhất, vì nó xử lý với các file đều đơn thuần như là xử lý dòng byte, mà không để ý tới nội dung của các file. Thứ ba, nó là phương thức hiệu quả nhất vì nó không tốn một lượng byte "overload" để thông báo header.

Block mode

Đây là phương thức truyền dữ liệu mang tính quy chuẩn hơn, với việc dữ liệu được chia thành nhiều khối nhỏ và được đóng gói thành các FTP blocks. Mỗi block này có một trường header 3 byte báo hiệu độ dài, và chứa thông tin về các khối dữ liệu đang được gửi. Một thuật toán đặc biệt được sử dụng để kiểm tra các dữ liệu đã được truyền đi và để phát hiện, khởi tạo lại đối với một phiên truyền dữ liệu đã bị ngắt.

Compressed mode

Đây là một phương thức truyền sử dụng một kỹ thuật nén khá đơn giản, là "run-length encoding" – có tác dụng phát hiện và xử lý các đoạn lặp trong dữ liệu được truyền đi

để giảm chiều dài của toàn bộ thông điệp. Thông tin khi đã được nén, sẽ được xử lý như trong block mode, với trường header. Trong thực tế, việc nến dữ liệu thường được sử dụng ở những chỗ khác, làm cho phương thức truyền kiểu compressed mode trở nên không cần thiết nữa. Ví dụ: nếu bạn đang truyền đi một file qua internet với modem tương tự, modem của bạn thông thường sẽ thực hiện việc nén ở lớp 1; các file lớn trên FTP server cũng thường được nén sẵn với một số định dạng như ZIP, làm cho việc nén tiếp tục khi truyền dữ liệu trở nên không cần thiết.

3.4.2. Cài đặt FTP Client/Server

Trên cơ sở giao thức FTP chúng ta thực hiện cài đặt FTP Client/Server để minh họa cho giao thức

Chương trình Simple FTP Server:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
class Program {
   static void Main(string[] args) {
      string rootDir = "C:/MyFTP";
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
      TcpListener server = new TcpListener(iep);
      server.Start();
      TcpClient client = server.AcceptTcpClient();
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      sw.WriteLine("220 Chao mung ket noi toi MyFTP");
      sw.Flush();
      while (true) {
         string request = sr.ReadLine();
         string command="";
         if(request.Length!=0)         command  = request.Substring(0, 4);
         switch (command.ToUpper().Trim()) {
            case "USER": {
                  sw.WriteLine("331. Nhap pass vao");
                  sw.Flush();
                  //sw.Close();
                  Console.WriteLine(request);
                  break;
               }
            case "PASS": {
                  sw.WriteLine("230. Dang nhap thanh cong");
                  sw.Flush();
                  Console.WriteLine(request);
                  break;
               }
```

```
            case "MKD": {
                string folderName = request.Substring(4, request.Length - 4);
                folderName = rootDir + "/" + folderName.Trim();
                try {
                   Directory.CreateDirectory(folderName);
                   sw.WriteLine("150 Tao thu muc thanh cong");
                   sw.Flush();
                }catch(IOException){
                   sw.WriteLine("550 Tao thu muc co loi");
                   sw.Flush();
                }
                break;
              }
            case "RETR": {
                string fileName = request.Substring(4, request.Length - 4);
                fileName = rootDir + "/" + fileName.Trim();
                try {
                   if (File.Exists(fileName)) {
                      //Gui noi dung file ve cho client xu ly
                      sw.WriteLine("150 Truyen File thanh cong");
                      sw.Flush();
                      FileStream fs = new FileStream(fileName, FileMode.Open);
                      long totalLenght = fs.Length;
                      byte[] data = new byte[totalLenght];
                      fs.Read(data, 0, data.Length);
                      sw.Write(totalLenght);
                      char[] kt = Encoding.ASCII.GetChars(data);
                      sw.Write(kt,0,data.Length);
                      sw.Flush();
                      fs.Close();
                   } else {
                      sw.WriteLine("550 File khong ton tai tren server");
                      sw.Flush();
                   }
                } catch (IOException) {
                   sw.WriteLine("550 Khong truyen duoc file");
                   sw.Flush();
                }
                break;
              }
            case "STOR": {
                string fileName =
request.Substring(request.LastIndexOf("/"),request.Length-request.LastIndexOf("/"));
                fileName = rootDir + "/" + fileName.Trim();
                try {
                   FileStream fs = new FileStream(fileName, FileMode.CreateNew);
                   long totalLength=sr.Read();
                   byte[] data = new byte[totalLength];
                   char[] kt = Encoding.ASCII.GetChars(data);
```

```
                    int sobyte = sr.Read(kt, 0, data.Length);
                    fs.Write(data, 0, sobyte);
                    fs.Close();
                    sw.WriteLine("150 Up file thanh cong");
                    sw.Flush();
                } catch (IOException) {
                    sw.WriteLine("550 Khong truyen duoc file");
                    sw.Flush();
                }
                break;
            }
        case "QUIT": {
                client.Close();
                break;
            }
        default: {
                sw.WriteLine("Sai lenh");
                sw.Flush();
                break;
            }
      }
    }
  }
}
```

Chương trình Simple FTP Client:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Net;
using System.Net.Sockets;
namespace FtpClient {
  class Program {
    static void Main(string[] args) {
      IPEndPoint iep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 2121);
      TcpClient client = new TcpClient();
      client.Connect(iep);
      StreamReader sr = new StreamReader(client.GetStream());
      StreamWriter sw = new StreamWriter(client.GetStream());
      Console.WriteLine(sr.ReadLine());
      string input;
      string command = "";
      Console.WriteLine("Dang nhap bang USER Ten user, PASS Ten password");
      Console.WriteLine("Tao thu muc bang MKD ten thu muc can tao");
      Console.WriteLine("Upload bang cach STOR tenfile");
      Console.WriteLine("Download bang cach RETR tenfile");
      while (true) {
        input = Console.ReadLine();
```

```
command = input.Substring(0, 4).Trim().ToUpper();
switch (command) {
   case "STOR": {
         //Doc file gui cho server
         sw.WriteLine(input);
         sw.Flush();
         FileInfo fl=null;
         try {
            fl = new FileInfo(input.Substring(4, input.Length - 4).Trim());
         } catch (IOException) {
            Console.WriteLine("File khong ton tai");
         }
         long totalLength = fl.Length;
         FileStream fs = fl.OpenRead();
         sw.Write(totalLength);
         byte[] data = new byte[totalLength];
         int bytes = fs.Read(data, 0, data.Length);
         char[] kt = Encoding.ASCII.GetChars(data);
         sw.Write(kt, 0, data.Length);
         sw.Flush();
         fs.Close();
         Console.WriteLine(sr.ReadLine());
         break;
      }
   case "RETR": {
         sw.WriteLine(input);
         sw.Flush();
         string s = sr.ReadLine();
         Console.WriteLine(s);
         if (s.Substring(0, 3).Equals("150")) {
            Console.Write("Nhap vao noi luu tep:");
            string filename = Console.ReadLine();
            FileStream fs = new FileStream(filename, FileMode.CreateNew);
            //Doc tep ve;
            long totalLength = sr.Read();
            byte[] data = new byte[totalLength];
            char[] kt= new char[data.Length] ;
            int sobyte = sr.Read(kt, 0, data.Length);
            data=Encoding.ASCII.GetBytes(kt);
            fs.Write(data, 0, data.Length);
            fs.Close();
         }
         break;
      }
   default: {
         sw.WriteLine(input);
         sw.Flush();
         Console.WriteLine(sr.ReadLine());
         break;
```

```
                }
            }
            if (input.ToUpper().Equals("QUIT")) break;
        }
        sr.Close();
        sw.Close();
        client.Close();
    }
  }
}
```

3.5. DNS (Domain Name Server)

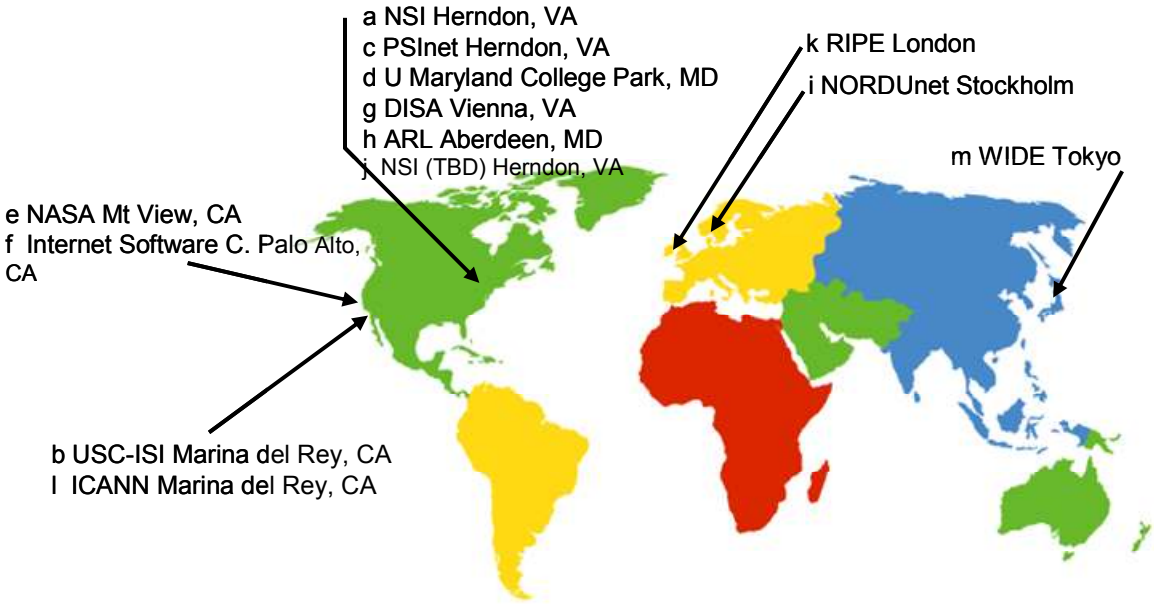
3.5.1. Vấn đề phân giải tên miền

Domain Name System:

- *Là hệ cơ sở dữ liệu phân tán hoạt động có thứ bậc bởi các name servers*
- *Là giao thức tầng ứng dụng :* host, routers yêu cầu tới name servers để xác định tên miền (ánh xạ địa chỉ <->tên miền)
  - Note : là một chức năng của Internet, hoạt động như là giao thức tầng ứng dụng
  - Rất phức tạp.

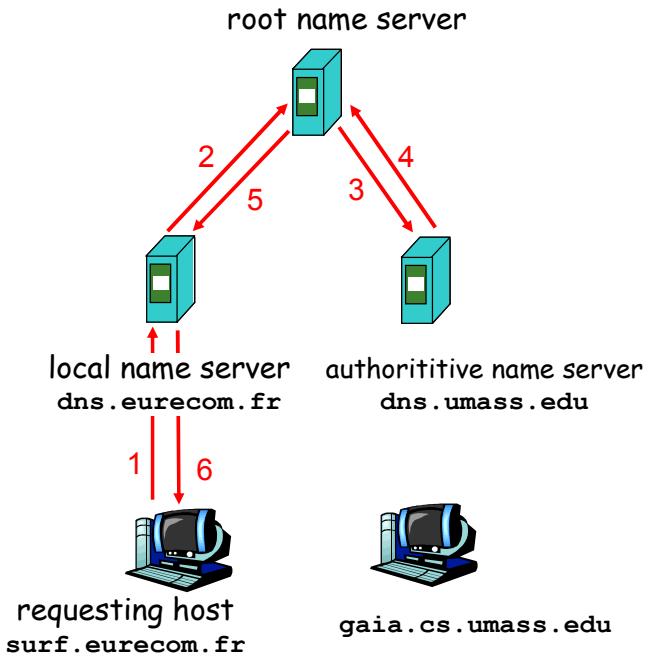
Q: Ánh xạ giữa địa chỉ IP và tên?

- **Tại sao không tập trung sự kiểm soát của DNS ?**
  - Điểm hỏng duy nhất - nếu name-server "chết" thì cả mạng Internet sẽ "chết" theo.
  - Tốn đường truyền.
  - Cơ sở dữ liệu tập trung sẽ "xa" với đa số vùng
  - Bảo trì phức tạp.
  - Phải chia để trị *!*
  - Không có server nào có thể lưu toàn bộ được tên miền và địa chỉ IP tương ứng
- **local name servers:**
  - Mỗi ISP,công ty có *local (default) name server*
  - Câu hỏi truy vấn của host về DNS sẽ được chuyển tới local name server
- **Chức năng của name server:**
  - Đối với host: lưu địa chỉ IP và tên miền tương ứng của host
  - Có thể tìm tên miền ứng với địa chỉ IP và ngược lại
- Được yêu cầu bởi các local name server không thể xác định được tên.
- root name server:
  - Được yêu cầu nếu có authoritative name server không xác định.
  - Nhận và xử lý mapping
  - Trả về mapping cho local name server



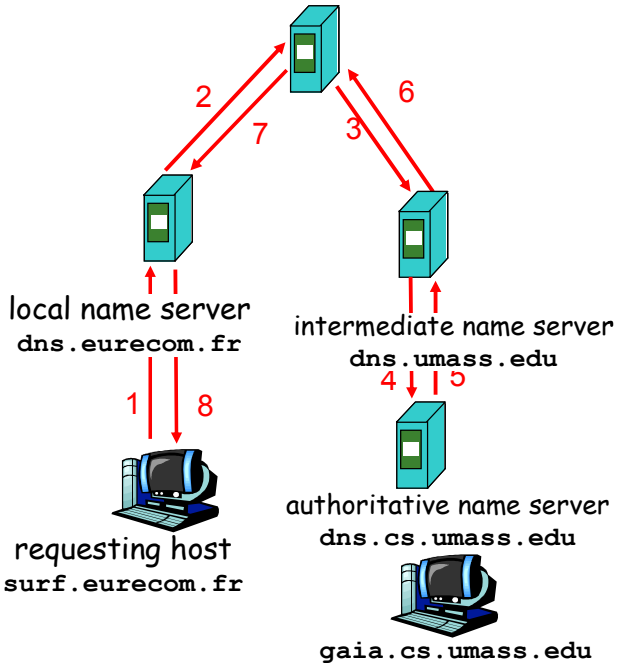
host **surf.eurecom.fr** muốn biết địa chỉ IP của **gaia.cs.umass.edu**

1. Yêu cầu tới local DNS server, **dns.eurecom.fr**
2. **dns.eurecom.fr** yêu cầu tới root name server nếu cần thiết
3. root name server yêu cầu authoritative name server, **dns.umass.edu,** nếu cần thiết.



Root name server:

- Có thể không biết authoritative name server
- Có thể biết *name server trung gian* ,nhờ đó có thể yêu cầu tìm authoritative name server



DNS example

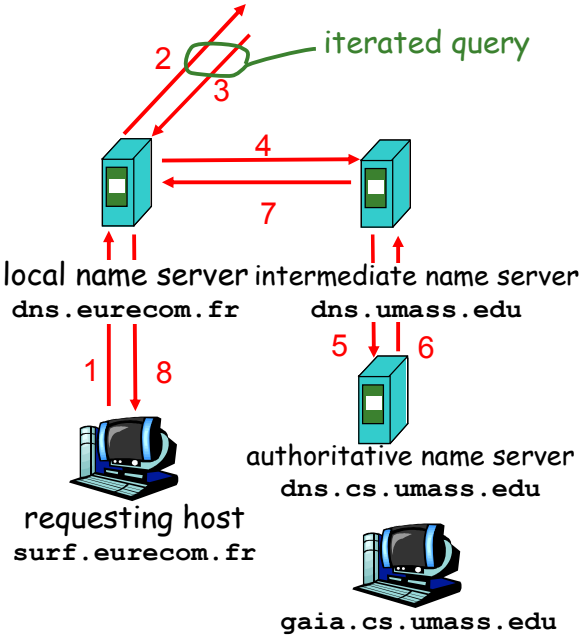
- Truy vấn trong DNS được chia thành các loại như sau:

Truy vấn đệ quy query:

- Name server là nơi phân gi ải địa chỉ/tên.Nếu nó không phân giải trong nội bộ,nó sẽ gửi yêu cầu đến name server khác.
- Công việc của name server liệu có quá nặng?

Truy vấn tương tác:

- Nếu không phân giải được địa chỉ IP/name,name server sẽ gửi trả thông đi ệp rằng "Tôi không biết,hãy thử hỏi anh bạn cạnh tôi là A".A là đ ịa chỉ IP của name server kế tiếp nó.



- Cấu trúc bản ghi DNS như sau:

DNS: cơ sở dữ liệu phân tán lưu các bản ghi nguồn (RR)

Định dạng của RR : **(name, value, type, ttl)**

- **Type=A**
  - **name** : hostname
  - **value** : IP address
- **Type=NS**
  - **name** : domain (e.g. foo.com)
  - **value** : địa chỉ IP authoritative name server cho tên miền đó
- **Type=CNAME**
  - **name** : tên bí danh cho một tên thực nào đó : e.g www.ibm.com là tên bí danh của servereast.backup2.ibm.com
  - **value** : là tên thực
- **Type=MX**
  - **value** : tên của mailserver

3.5.2. Triển khai DNS MX (Mail Exchange)

Chúng ta đi viết chương trình cho phép lấy về thông tin của mail server

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
public partial class Form1 : Form {
   public Form1() {
      InitializeComponent();
   }
   private void btFind_Click(object sender, EventArgs e) {
      byte[] DNSQuery;
      byte[] DNSReply;
      UdpClient dnsClient = new UdpClient(tbServer.Text, 53);
      DNSQuery = makeQuery(DateTime.Now.Millisecond *
      60, tbDomain.Text);
      dnsClient.Send(DNSQuery, DNSQuery.GetLength(0));
      IPEndPoint endpoint = null;
      DNSReply = dnsClient.Receive(ref endpoint);
      this.tbStatus.Text = makeResponse(DNSReply, tbDomain.Text);
   }
   public byte[] makeQuery(int id, string name) {
      byte[] data = new byte[512];
      byte[] Query;
      data[0] = (byte)(id >> 8);
      data[1] = (byte)(id & 0xFF);
      data[2] = (byte)1; data[3] = (byte)0;
      data[4] = (byte)0; data[5] = (byte)1;
```

```
        data[6] = (byte)0; data[7] = (byte)0;
        data[8] = (byte)0; data[9] = (byte)0;
        data[10] = (byte)0; data[11] = (byte)0;
        string[] tokens = name.Split(new char[] { '.' });
        string label;
        int position = 12;
        for (int j = 0; j < tokens.Length; j++) {
           label = tokens[j];
           data[position++] = (byte)(label.Length & 0xFF);
           byte[] b = System.Text.Encoding.ASCII.GetBytes(label);
           for (int k = 0; k < b.Length; k++) {
              data[position++] = b[k];
           }
        }
        data[position++] = (byte)0; data[position++] = (byte)0;
        data[position++] = (byte)15; data[position++] = (byte)0;
        data[position++] = (byte)1;
        Query = new byte[position + 1];
        for (int i = 0; i <= position; i++) {
           Query[i] = data[i];
        }
        return Query;
     }
     public string makeResponse(byte[] data, string name) {
        int qCount = ((data[4] & 0xFF) << 8) | (data[5] & 0xFF); int aCount = ((data[6]
& 0xFF) << 8) | (data[7] & 0xFF);
        int position = 12;
        for (int i = 0; i < qCount; ++i) {
           name = "";
           position = proc(position, data, ref name);
           position += 4;
        }
        string Response = "";
        for (int i = 0; i < aCount; ++i) {
           name = "";
           position = proc(position, data, ref name);
           position += 12;
           name = "";
           position = proc(position, data, ref name);
           Response += name + "\r\n";
        }
        return Response;
     }
     private int proc(int position, byte[] data, ref string name) {
        int len = (data[position++] & 0xFF);
        if (len == 0) {
           return position;
        }
        int offset;
```

```
        do {
            if ((len & 0xC0) == 0xC0) {
                if (position >= data.GetLength(0)) {
                    return -1;
                }
                offset = ((len & 0x3F) << 8) | (data[position++] &
                  0xFF);
                proc(offset, data, ref name);
                return position;
            } else {
                if ((position + len) > data.GetLength(0)) {
                    return -1;
                }
                name += Encoding.ASCII.GetString(data, position, len);
                position += len;
            }
            if (position > data.GetLength(0)) {
                return -1;
            }
            len = data[position++] & 0xFF;
            if (len != 0) {
                name += ".";
            }
        }
        while (len != 0);
        return position;
    }
}
```

3.6 Thảo luận về các ứng dụng khác thường gặp

3.7 Bài tập áp dụng