Web Servers

Timothy Daly

February 18, 2004

Abstract

This lecture covers the standards and concepts involved in Web Servers. We talk about the RFC standards in some detail. Then we present the source code for a full implementation of a standard Web Server and examine the network interactions. We demonstrate communicating with this Web Server using telnet by hand and using a standard web browser. Next we review ideas involved in using Apache, an open source web server.

Contents

1 Weekly News

Copyright for Computer Authors This must-read article popped up because it is rumored that Microsoft's Windows 2000 and Windows NT source code has been illegally released on the net.

The lawyers are coming It seems that companies will eventually have lawsuits brought against them due to "negligence". In particular, you could be held responsible for negligence by not applying security patches. This has already happened in two lawsuits. One was here in New York State [3]:

> Maine Public Utilities Commission v. Verizon. Verizon rents the use of Maine Public Utilities infrastructure, but Verizon went out of action due to the Slammer worm - revealing that Verizon's patch management process was not up to snuff. Verizon asked Maine for a refund for the time it was out of action. Maine refused to pay asserting that Verizon was negligent. The argument went to court and the verdict went to Maine.

IBM: Linux mainframe for German authority [**1**] Linux is making inroads at the Federal, state, and local levels in Germany.

> The German Federal Finance Office has implemented what its technology supplier, IBM Corp., is calling one of the largest Linux-based mainframe deployments in Europe.
>
> The Berlin-based authority has replaced more than 30 smaller servers with one mainframe computer, IBM's eServer z990, running the open-source Linux operating system, IBM said Tuesday.
>
> The Linux deployment is part of an agreement Big Blue struck in 2002 with the German Federal Ministry of the Interior to supply computers with Linux at a discount to federal, state, and local governments as well as other public authorities.

World of Ends: What the Internet is and how to stop mistaking it for something else [**2**] This article gives 10 quick guidelines about the internet. In a distant future the 10 rules will also come to dominate the IT working world. You will be one of the "ends" that contributes value to the IT net.

Linux Moves In On The Desktop [**5**]

> Market researcher IDC expects to announce within weeks that Linux PC market share in 2003 hit 3.2%, overtaking Apple Computer Inc.'s Macintosh software. And the researcher expects Linux to capter 6% of this market by 2007. That's still tiny compared with Microsoft's 94% share. But it's clear now that

> Linux is becoming a viable alternative to Windows on desktop and laptop PCs for companies willing to put up with the trouble of switching.
>
> Linux has made major strides in the past few months, In November, China declared it the operating system of choice. Starting on Jan. 1, the Israeli government plans gradually to replace desktop Windows with Linux. IBM CEO Samuel J. Palmisano last last year challenged his 319,000-employee company to move entirely to Linux PCs. And now, analysts say, dozens of major corporations in the U.S., Europe, and Japan are sizing up Linux. In a survey of corporate buyers by Merrill Lynch & Co., 43% said they would consider replacing Windows with desktops with Linux. "I had expected governments to be interested, but now it's on the radar of corporate chief information officers." says analyst Steven Mulunovich of Merrill Lynch.

The Microsoft Killers This is a discussion of the "coming of age" of open source effort. It talks about the benefits to the public good of open source efforts.

EU Poised to Attack P2P File-Sharers From Slashdot on Tuesday, February 17, 2004 [41]:

> Robin Gross of IP Justice writes "The EU is about to vote on a controversial piece of legislation [42] that targets P2P file-sharing and other non-commercial infringements. The EU Intellectual Property Rights Directive creates a 'nuclear weapons' of law enforcement tools for intellectual property holders. It combines the most extreme enforcement provisions found throughout Europe and imposes them collectively onto all of Europe, for example England's Anton Pillar orders that permit recording industry executives to raid and ransack the homes of alleged users of file-sharing software or it's Mareva injunctions that freeze a defendant's bank accounts without a hearing. The vote in the EU plenary will likely be March 11, 2004 - watch the CODE [43] site for developments."

2 The fundamental idea

Every program has a fundamental idea, the very essense of its existence. Web servers are a very simple idea which is that I want to listen for file requests and, when asked, send them.

Most programs that are widely used are based on a standard. Web servers rest on two standards: TCP/IP for network protocols and HTTP for file transfers. First we cover these two standards.

3 Standards

3.1 IP [19, 20]

The phone network is a circuit-switched network. When you make a phone call there is a continuous wire circuit that connects you to the other party thru dedicated switches. Thus the whole connection is set up when you dial, exists for the length of the call, and is torn down when you hang up.

The Internet is a packet-switched network. When you send or receive information it is broken up into packets, the packets are tagged with their source and destination and are sent out to the local router. The router reads the destination, decides which of its many connecting paths is the best path, and send the packet out to the next router. Eventually your packet reaches the router connected to the destination. That router sees that the address is locally attached and forwards the packet to the destination node. Each connection is made as needed and there is no continuous connection between the source and the destination.

IP, the Internet Protocol, specifies the contents of the packets.

3.2 TCP/IP [21, 40]

Packets are sent on a best-effort basis. There is no guarantee that your packets will be sent. There is no guarantee that your packets will arrive in the order they were sent (errors can cause packets to go the long way around while other packets make it thru the short way). There is no guarantee that your packets are not sent several times resulting in duplicates (this can happen if a routing error occurs and your packets get stored and routed over multiple paths). The default protocol is called UDP [39], which means "User Datagram Protocol".

TCP, which mean "Transmission Control Protocol" (telnet) is a **reliable** protocol. That is, TCP will deliver your packets, guarantee that they arrive, arrive in order, and arrive without duplicates. TCP does this by implementing a *handshake* protocol that allows the sending end to number the packets and the receiving end to acknowledge numbered packets, put them in numeric order, and discard duplicates. Lost packets can be explicitly requested.

Thus TCP builds a reliable protocol on top of an unreliable one. The advantage is clear. The disadvantage is cost in bits used for overhead and in time spent acknowledging packets. Over a reliable link TCP is much more expensive than UDP.

3.3 HTTP [8, 9]

So now we can send and receive packets reliably. There are many kinds of transfers we can set up once we have such a facility. We could build a message mechanism such as Instant Messaging (ICQ) [17, 18]. We could build a mail mechanism such as POP [15, 16]. We could build a file transfer mechanism such as FTP [10, 11, 12, 13, 14].

Each of these transfer mechanisms is a "protocol" built on top of TCP/IP. A protocol is just an agreement about what the sender and the receiver expect.

In this case we are building a protocol to transfer "pages". A page is really nothing more than a file. Some files conform to a particular format called HTML [22, 23, 24, 25, 26, 27].

The HTTP standard says that an HTTP server should be prepared to handle the following messages:

OPTIONS

> The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

GET

> The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

HEAD

> The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

POST

> The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI in the Request-Line. POST is designed to allow a uniform method to cover the following functions:
>
> - Annotation of existing resources;
>
> - Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;

- Providing a block of data, such as the result of submitting a form, to a data-handling process;
- Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the Request-URI. The posted entity is subordinate to that URI in the same way that a file is subordinate to a directory containing it, a news article is subordinate to a newsgroup to which it is posted, or a record is subordinate to a database.

PUT

The PUT method requests that the enclosed entity be stored under the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases.

DELETE

The DELETE method requests that the origin server delete the resource identified by the Request-URI. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

TRACE

The TRACE method is used to invoke a remote, application-layer loop- back of the request message. The final recipient of the request SHOULD reflect the message received back to the client

> as the entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request (see section 14.31). A TRACE request MUST NOT include an entity.
>
> TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field (section 14.45) is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.

CONNECT

> This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling [44]).

This is a very small list and one we could easily implement. The fundamental command is the **GET** command. The standard says:

> ... The methods GET and HEAD MUST be supported by all general-purpose servers. All other methods are OPTIONAL; however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in section 9.

The GET and HEAD commands take 2 arguments, the page to fetch and the name of the HTTP version to which the request conforms. Since HTTP is a very old protocol the original versions would accept 1 argument; the page to fetch. So servers will respond to single argument requests.

3.3.1 Universal Resource

> ... It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) [28], as a location (URL) [29] or name (URN) [30], for indicating the resource to which a method is to be applied. Messages are passed in a format similar to that used by Internet mail [31] as defined by the Multipurpose Internet Mail Extensions (MIME) [32].

The GET command will accept something of the general form:

```
protocol://user:pass@host:port/url
```

where:
protocol is one of http, ftp, https, etc.
user is a user name,
pass is a password,

host is a host name or IP address,
port is a socket number, and
url conforms to the URL standard [29].

Since HTTP uses TCP as its transport mechanism we can use the TCP program called telnet to connect to the server and request a page. We can, for instance, type:

```
telnet www.google.com 80
GET /
```

Notice that the GET must be uppercase and the GET line is not echoed by telnet so you are typing blind. What comes back from this request is the default file called index.html. The index.html file is assumed to be the default root page in a web server. So the GET command above is equivalent to:

```
GET /index.html
```

If we look at the first command there are several interesting things of note:

```
telnet www.google.com 80
```

As mentioned the *telnet* command uses TCP as its protocol which is the same protocol as HTTP uses.

The *"www.google.com"* is a name that gets resolved into an IP address by a program called BIND [38], the Berkeley Internet Name Domain, which is an implementation of the DNS Domain Name System [33, 34, 35, 36, 37] protocols.

4 A Trivial Web Server

This is a trivial web server. It is composed of two classes. The first class, called WebServer, has only a *main* method. The **main** method will create a new copy of the *getRequest* method to handle HTTP GET requests.

A web server is only required to implement GET requests. All of the other HTTP commands are optional.

You can run this code by typing:

```
javac WebServer.java
java WebServer
```

From another terminal on the same machine you can ask for files thus:

```
telnet 127.0.0.1 80
GET /WebServer.java
```

The telnet command establishes a TCP/IP connection to the local machine (127.0.0.1 is the default IP address of "this machine") and connects to port 80, the HTTP server port.

The "GET /WebServer.java" command says that we want to get the file called "WebServer.java" in the current directory.

The **main** method creates a new *getRequest* class as a new thread to handle this request. The *getRequest* routine changes the filename to read "./WebServer.java" so we look in the current directory. The console trace on the WebServer machine will look like (lines with "==>" are from you, lines with "<==" are what WebServer typed):

```
New connection accepted /127.0.0.1:49250
==>GET /WebServer.java
<==HTTP/1.0 200 OK
<==WebServer
<==Content-type: text/plain
<==Content-Length: 4045
<==
<== 200: file sent: ./WebServer.java
```

We can also use a web browser to connect by typing:

```
http://127.0.0.1/WebServer.java
```

in the location input box of your browser. Note that the browser sends all kinds of additional information besides the GET command.

```
New connection accepted /127.0.0.1:49250
==>GET /WebServer.java HTTP/1.1
<==HTTP/1.0 200 OK
<==WebServer
<==Content-type: text/plain
<==Content-Length: 4045
<==
<== 200: file sent: ./WebServer.java
==>Host: 127.0.0.1
==>User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.2.1) Gecko/20030225
==>Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,v
==>Accept-Language: en-us, en;q=0.50
==>Accept-Encoding: gzip, deflate, compress;q=0.9
==>Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
==>Keep-Alive: 300
==>Connection: keep-alive
==>
```

Note the glaring security hole. We could ask for something like

```
GET /../filename
```

The WebServer will let you fetch any file you can reach. Don't use this code as a real web server. In fact, this is a standard attack on Microsoft web servers. You simply ask for files anywhere on the machine. A lot of the servers are badly configured and will download any file on your Microsoft machine.

```
import java.io.*;
import java.lang.*;
import java.net.*;
import java.util.*;

public class WebServer
{

 public static void main(String args[])
 { ServerSocket sock;
   try
   { sock = new ServerSocket(80);
     System.out.println("WebServer started");
     while(true)
     { Socket socket = sock.accept();
       System.out.println("New connection accepted " +
           socket.getInetAddress() + ":" + socket.getPort());
       try
       { getRequest request =  new getRequest(socket);
         Thread thread = new Thread(request);
         thread.start();
       }
       catch(Exception e)
       { e.printStackTrace();
       }
     }
   }
   catch (Exception e)
   { e.printStackTrace();
   }
 }
}

class getRequest implements Runnable
{ String CRLF = "\r\n";
  Socket sock;
  InputStream input;
  OutputStream output;
  BufferedReader bin;

  public getRequest(Socket s) throws Exception
  { this.sock = s;
    this.input = s.getInputStream();
    this.output = s.getOutputStream();
    this.bin= new BufferedReader(new InputStreamReader(s.getInputStream()));
  }
```

```
public void run()
{ while(true)
  { try
    { String GETline = bin.readLine();
      System.out.println("==>"+GETline);
      if(GETline.equals(CRLF) || GETline.equals("")) break;
      StringTokenizer s = new StringTokenizer(GETline);
      String temp = s.nextToken();
      if(temp.equals("GET"))
      { String filename = s.nextToken();
        filename = "." + filename ;
        FileInputStream inFile = null ;
        boolean fileExists = true ;
        try
        { inFile = new FileInputStream( filename ) ;
        }
        catch ( FileNotFoundException e )
        { fileExists = false ;
        }
        String serverln = "WebServer"+CRLF;
        String statusln = null;
        String contentTypeln = null;
        String FourOFour = null;
        String contentLengthln = "error";
        String mime="text/html";
        if (!(filename.endsWith(".html")))
          mime="text/plain";
        if (fileExists)
        { statusln = "HTTP/1.0 200 OK" + CRLF ;
          contentTypeln = "Content-type: "+mime+CRLF;
          contentLengthln = "Content-Length: " +
             (new Integer(inFile.available())).toString() + CRLF;
        }
        else
        { statusln = "HTTP/1.0 404 Not Found" + CRLF ;
          contentTypeln = "Content-type: " + "text/html" + CRLF;
          FourOFour =
           "<HTML>" +
           " <HEAD>" +
           "  <TITLE>" +
           "   404 Not Found" +
           "  </TITLE>" +
           " </HEAD>" +
           " <BODY>"+
           "  404 Not Found "+filename+
```

```
            " </BODY>"+
            "</HTML>";
          contentLengthln = "Content-Length: " +
             (new Integer(FourOFour.length())).toString() + CRLF;
        }
        output.write(statusln.getBytes());
        System.out.print("<=="+statusln);
        output.write(serverln.getBytes());
        System.out.print("<=="+serverln);
        output.write(contentTypeln.getBytes());
        System.out.print("<=="+contentTypeln);
        output.write(contentLengthln.getBytes());
        System.out.print("<=="+contentLengthln);
        output.write(CRLF.getBytes());
        System.out.print("<=="+CRLF);
        System.out.flush();
        if (fileExists)
        { byte[] buffer = new byte[1024] ;
          int bytes = 0 ;
          while ((bytes = inFile.read(buffer)) != -1 )
          { output.write(buffer, 0, bytes);
          }
          inFile.close();
          System.out.println("<== 200: file sent: "+filename);
        }
        else
        { output.write(FourOFour.getBytes());
          System.out.println("<== 404: not found: "+filename);
        }
      }
    }
    catch(Exception e)
    { e.printStackTrace();
    }
  }
  try
  { output.close();
    bin.close();
    sock.close();
  }
  catch(Exception e) {}
 }

}
```

5 Apache

The Apache Web Server [7] is a large, free, open source program to handle Web Services. In concept it rests on the ideas already discussed. In practice it integrates a large number of other web service technologies.

5.1 Simple Configuration

5.2 A Simple Web Site

Websites are trivial. Remember that the primary function of a web server is to deliver static files. If we do not specify a particular file from a directory the web server will default to a file called "index.html". So we create a this file:

```
<HTML>
 <HEAD>
  <TITLE>
   Hello, World
  </TITLE>
 </HEAD>
 <BODY>
  Hello, World
 </BODY>
</HTML>
```

Apache keeps raw web pages in the *html* directory so we move this file there:

```
cp index.html /var/www/html
```

and then we ask for the code by going to a web browser (or using telnet) at the url:

```
http://127.0.0.1/index.html
```

or, since this is the default page we can just say:

```
http://127.0.0.1/
```

Once we have set up the first default page we can use HTML to construct pages as high, wide, and deep at we want. HTML programming is not covered here.

5.3 Running Programs

The "Hello, World" program is the canonical first C program. Apache can run such a program and sen the output results back to a web browser. Such a program is called a "cgi-bin" program. First, we need a simple C program.

5.3.1 Hello, World

Hello, World is the standard first C program. It usually looks like this:

```
#include <stdio.h>

int main(int argc, char *argv)
{ printf("Hello, World\n");
}
```

However, a cgi-bin program (that is, a program run by a web server in response to a request) needs to output some header information and html code. We do this here. In a file called "hello.c" we put the following code:

```
#include <stdio.h>

int main(int argc, char *argv)
{
  char *result="<HTML><HEAD></HEAD><BODY>Hello, World\n</BODY></HTML>\n";
  printf("Content-type: text/html\n");
  printf("Content-Length: %d\n",strlen(result));
  printf("\n");
  printf("%s",result);
  return(0);
}
```

Next, we compile this code:

```
gcc -o hello hello.c
```

Since the compiled code doesn't depend on anything we can just run it and look at the result:

```
Content-type: text/html
Content-Length: 53

<HTML><HEAD></HEAD><BODY>Hello, World
</BODY></HTML>
```

Since it outputs valid html we copy it to the cgi-bin directory:

```
cp hello /var/www/cgi-bin
```

and then we ask for the code by going to a web browser (or using telnet) at the url:

```
http://127.0.0.1/cgi-bin/hello
```

References

[1] German Federal Linux Project,
"IBM: Linux mainframe for German authority"
"http://open.itworld.com/4917/040210ibmmainframe/page_1.html"

[2] Searls, Doc, and Weinberger, David,
"World of Ends: What the Internet is and how to stop mistaking it for something else" **"http://www.worldofends.com"**

[3] The Register,
"The lawyers are coming",
"http://www.theregister.co.uk/content/55/35513.html"

[4] Field, Thomas,
"Copyright for Computer Authors",
"http://www.piercelaw.edu/tfield/copySof.htm"

[5] BusinessWeek Online,
"Linux Moves In On The Desktop",
"http://story.news.yahoo.com/news?tmpl=story&u=/bw/20040213/bs_bw/b3871118mz063"

[6] Azhar, Azeem,
"The Microsoft Killers",
"http://www.prospect-magazine.co.uk/ArticleView.asp?accessible=yes&P_Article=12404"

[7] Apache,
"http://httpd.apache.org"

[8] Hypertex Transfer Protocol – HTTP/1.0,
"http://asg.web.cmu.edu/rfc/rfc1945.txt"

[9] Hypertex Transfer Protocol – HTTP/1.1,
"http://asg.web.cmu.edu/rfc/rfc2616.txt"

[10] The File Transfer Protocol,
"http://asg.web.cmu.edu/rfc/rfc0265.txt"

[11] The File Transfer Protocol,
"http://asg.web.cmu.edu/rfc/rfc0354.txt"

[12] The File Transfer Protocol,
"http://asg.web.cmu.edu/rfc/rfc0542.txt"

[13] The File Transfer Protocol,
"http://asg.web.cmu.edu/rfc/rfc0765.txt"

[14] The File Transfer Protocol,
"http://asg.web.cmu.edu/rfc/rfc0959.txt"

[15] Post Office Protocol,
"http://asg.web.cmu.edu/rfc/rfc0918.txt"

[16] Post Office Protocol: Version 2,
"http://asg.web.cmu.edu/rfc/rfc0937.txt"

[17] A Model for Presence and Instant Messaging,
"http://asg.web.cmu.edu/rfc/rfc2778.txt"

[18] Instant Messaging / Presence Protocol Requirements,
"http://asg.web.cmu.edu/rfc/rfc2779.txt"

[19] DoD standard Internet Protocol,
"http://asg.web.cmu.edu/rfc/rfc0760.txt"

[20] Internet Protocol,
"http://asg.web.cmu.edu/rfc/rfc0791.txt"

[21] Telnet Protocol specification,
"http://asg.web.cmu.edu/rfc/rfc0764.txt"

[22] Telnet Protocol Specification,
"http://asg.web.cmu.edu/rfc/rfc0864.txt"

[23] Form-based File Upload in HTML,
"http://asg.web.cmu.edu/rfc/rfc1867.txt"

[24] HTML Tables,
"http://asg.web.cmu.edu/rfc/rfc1942.txt"

[25] A Proposed Extension to HTML: Client-Side Image Maps,
"http://asg.web.cmu.edu/rfc/rfc1980.txt"

[26] Internationalization of the Hypertext Markup Language,
"http://asg.web.cmu.edu/rfc/rfc2070.txt"

[27] The 'text/html' Media Type,
"http://asg.web.cmu.edu/rfc/rfc2854.txt"

[28] Universal Resource Identifiers in WWW,
"http://asg.web.cmu.edu/rfc/rfc1630.txt"

[29] Uniform Resource Locators (URL),
"http://asg.web.cmu.edu/rfc/rfc1738.txt"

[30] Functional Requirements for Uniform Resource Names,
"http://asg.web.cmu.edu/rfc/rfc1737.txt"

[31] Standard for The Format of ARPA Internet Text Messages,
"http://asg.web.cmu.edu/rfc/rfc0822.txt"

[32] Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,
"http://asg.web.cmu.edu/rfc/rfc2045.txt"

[33] Distributed system for Internet name service,
"http://asg.web.cmu.edu/rfc/rfc0830.txt"

[34] Domain adminstrators guide,
"http://asg.web.cmu.edu/rfc/rfc1032.txt"

[35] Domain adminstrators operations guide,
"http://asg.web.cmu.edu/rfc/rfc1033.txt"

[36] Domain names - concepts and facilities,
"http://asg.web.cmu.edu/rfc/rfc1034.txt"

[37] Domain names - implementation and specification,
"http://asg.web.cmu.edu/rfc/rfc1035.txt"

[38] Internet Systems Consortium,
"http://www.isc.org"

[39] User Datagram Protocol,
"http://asg.web.cmu.edu/rfc/rfc0768.txt"

[40] Hypertext Markup Language - 2.0,
"http://asg.web.cmu.edu/rfc/rfc1866.txt"

[41] EU Poised to Attack P2P File-Sharers,
"http://slashdot.org/"

[42] EU Poised to Attack P2P File-Sharers,
"http://www.ipjustice.org/code/021404.html"

[43] CODE: Campaign for an Open Digital Environment,
"http://www.ipjustice.org/code"