

Università degli Studi di Trieste  
Facoltà di Ingegneria  
Corso di Diploma in Ingegneria Informatica  
Anno Accademico 2001 - 2002

**Progetto di una piccola rete ad indirizzi pubblici  
collegata ad un server web e ad un server di posta  
tramite un router protetto da access-list**

Corso di Sistemi di Elaborazione  
Docente: prof. Paolo Inchingolo  
Esercitatore: dott. Michele Bon

Esaminandi: Fedric Avian, Paolo Luca

## Introduzione

Il progetto presentato prevede lo sfruttamento di un router, di uno switch di tipo intelligente e di un certo numero (in verità variabile) di workstations collegate alla rete creata appositamente con i primi due elementi.

Lo scopo del progetto è quello di collegare una rete di dimensioni medio - piccole, quale per esempio quella di una piccola azienda, al server di un provider di servizi web, al server di un secondo provider di servizi di e-mail per il tramite di una stazione interna programmata in modo da agire essa stessa da server di e-mail, con le caselle di posta necessarie allo svolgimento della funzione di posta interna.

Il progetto è stato diviso in sei parti distinte:

1. protezione del router tramite programmazione di un'access-list idonea allo scopo prefisso
2. divisione dello switch in VLAN protette sulla base dell'indirizzo IP sorgente
3. programmazione dei server web e ftp, dei server pop3 e smtp
4. programmazione dei client interni e del server DNS
5. dettaglio del programma di serving web
6. collaudo generale

Ogni parte verrà trattata separatamente l'una dalle altre, in quanto si tratta di operazioni virtualmente indipendenti.

## Protezione del router

Il router è l'elemento di separazione tra la rete definita "interna" e il "resto del mondo": è stato pertanto protetto con un'access-list studiata per consentire il passaggio di informazioni in essenzialmente due modi: il primo, per consentire a qualsiasi stazione della rete definita "interna" di uscire dalla rete interna soltanto sui porti di interesse per il web e per il trasferimento di file soltanto in senso entrante; la connessione deve però essere scatenata dall'"interno" all'"esterno", e non viceversa; il secondo, per consentire l'ingresso e l'uscita della posta elettronica soltanto da alcune stazioni (il server interno) in modo indipendente e bidirezionale. Si è voluto inoltre permettere il passaggio dei pacchetti ICMP, anche questo in modo indipendente e bidirezionale.

L'access-list che risulta da questo lavoro è questa:

```
access-list 116 permit icmp 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
access-list 116 permit tcp 140.105.17.0 0.0.0.255 0.0.0.0 255.255.255.255 eq 80
access-list 116 permit tcp 140.105.17.0 0.0.0.255 0.0.0.0 255.255.255.255 eq 8080
access-list 116 permit tcp 140.105.17.0 0.0.0.255 0.0.0.0 255.255.255.255 eq 21
access-list 116 permit tcp 140.105.17.0 0.0.0.31 0.0.0.0 255.255.255.255 eq 110
access-list 116 permit tcp 140.105.17.0 0.0.0.31 0.0.0.0 255.255.255.255 eq 25
access-list 116 permit tcp 0.0.0.0 255.255.255.255 140.105.17.0 0.0.0.255 eq 20
access-list 116 permit tcp 0.0.0.0 255.255.255.255 140.105.17.0 0.0.0.31 eq 25
access-list 116 permit tcp 0.0.0.0 255.255.255.255 140.105.17.0 0.0.0.31 eq 110
access-list 116 permit tcp 0.0.0.0 255.255.255.255 140.105.17.0 0.0.0.255 established
access-list 116 permit tcp 140.105.17.0 0.0.0.255 0.0.0.0 255.255.255.255 established
access-list 116 deny tcp 0.0.0.0 255.255.255.255 140.105.17.250 0.0.0.0 eq 23
access-list 116 deny ip 0.0.0.0 255.255.255.255 140.105.17.0 0.0.0.255
access-list 116 deny ip 140.105.17.0 0.0.0.255 0.0.0.0 255.255.255.255
```

Come si può notare, si permette l'uscita sia sul porto 80 (web standard) che sul porto 8080 (web ausiliario), oltre che sul porto 21 (comandi del FTP) da tutte le stazioni; si permette inoltre l'entrata dall'esterno sul porto 20 (canale dati del FTP) e sul resto dei porti soltanto su connessione già stabilita e scatenata dall'interno; sui porti della posta invece si vuole permettere l'uscita e l'entrata soltanto alle macchine con numerazione corrispondente ai server. Tutto il resto viene bloccato, nei due sensi, dal router stesso, isolando la rete "interna" dal "resto del mondo" e rendendo virtualmente impossibile il transito uscente di ulteriori informazioni.

Una particolarità di questa access-list: alla terz'ultima riga viene bloccato esplicitamente il transito di pacchetti Telnet dalla rete esterna all'interno dello switch (rinumerato per l'occasione come 140.105.17.250)

per impedire riconfigurazioni indebite, dal momento che si vuole che quest'ultimo faccia parte della rete "interna".

## Configurazione e protezione dello switch

Lo switch, di tipo "intelligente" e pertanto divisibile in VLAN indipendenti, è stato configurato in modo da creare cinque VLAN; alcune di queste sono state protette sulla base dell'indirizzo IP sorgente.

La VLAN id 1, la **default VLAN**, alla quale appartengono tutte le porte dello switch senza alcun tipo di classificazione, agisce da "cestino della spazzatura", in quanto non esce su alcuna porta.

La VLAN id 3, a nome **MailServerVLAN**, classifica come "suo" tutto ciò che entra dalle porte comprese tra la 9 e la 12 e che provenga da macchine con indirizzi IP facenti parte della rete 140.105.19.0 (netmask 255.255.255.0); inoltre esce sulla porta 17, alla quale è collegato il router il quale risponde con un indirizzo secondario che fa parte della di questa VLAN..

La VLAN id 4, la **WebServerVLAN**, è identica alla terza, tranne per il fatto che insiste sulle porte dalla 13 alla 16, uscendo anche sulla porta 17.

La VLAN id 6, a nome **dnsVLAN**, classifica come "suo" tutto quello che entra dalle porte dalla 1 alla 8 e dalla porta 18, e che provenga da stazioni il cui indirizzo sorgente faccia parte della rete 140.105.17.0 (net mask 255.255.255.0); inoltre può uscire sulla porta 17, alla quale è collegato il router, il quale deve poter rispondere con un indirizzo secondario che fa parte del *pool* specificato per la InnerNetVLAN. Inoltre la VLAN esce anche dalla porta 27, che identifica lo switch stesso: quest'ultimo può venir quindi riconfigurato da remoto tramite operazioni di Telnet (porto TCP 23), ma soltanto da una stazione della rete "interna"

La VLAN id 5, la **RouterVLAN**, è rovesciata rispetto alle altre: classifica come suo tutto quello che entra dalla porta 17, bloccata sulla rete determinata dall'indirizzo *primario* del router (anche in questo caso trattasi dell'indirizzo IP sorgente), e permette l'uscita dei pacchetti su tutte le porte appartenenti alle altre VLAN.

Come già anticipato, tutte le VLAN, tranne la prima, sono classificate sulla base dell'indirizzo IP sorgente, ossia accettano pacchetti IP immessi da una stazione il cui indirizzo sia compreso all'interno di un *pool* specificato e che sia collegata su porte di rete specificate. Ciò significa che in mancanza di questi due requisiti il pacchetto appartiene alla **Default VLAN**, e viene quindi cestinato dal momento che quest'ultima non ha porte in uscita.

## Programmazione delle stazioni server esterne

Le stazioni server definite "esterne" sono due: un server POP3 e SMTP collegato ad una qualsiasi delle porte tra la 9 e la 12 ed un server web e FTP collegato su una qualsiasi delle porte tra la 13 e la 16. Tali stazioni agiscono da "resto del mondo".

### Il server di posta

Questo server è costituito da una stazione PentiumIII, temporaneamente distaccata dal laboratorio ex-Duinf di informatica ad uso studenti presso il DEEI, sulla quale sono installati sia Windows NT 4.0 Workstation (autoavviante) sia RedHat Linux 7.1 (avviabile tramite floppy di boot). Si è deciso infatti di utilizzare alcuni dei pacchetti software offerti dai sistemi Linux per tutte le stazioni server esterne. Nella fattispecie, su questa stazione verranno fatti girare i pacchetti di gestione dei porti 25 e 110 (SMTP e POP3) utilizzando la programmazione data dal sistema. Inoltre, l'interfaccia Ethernet è stata rinumerata come 140.105.19.1 per l'occasione, ma soltanto sul sistema operativo Linux (la partizione NTFS non è stata minimamente alterata).

### Il server web

Anche questo server è costituito da una stazione PentiumIII temporaneamente presa in prestito dallo stesso laboratorio dal quale è stata presa la stazione di cui sopra. Su questa, oltre che Windows NT 4.0 Workstation (autoavviante), è stato installato Mandrake Linux 8.1 (avviabile tramite il Boot Manager di Windows NT; è però stato fatto anche un floppy di avvio) provvisto di server FTP fornito con i pacchetti di sistema, e di interprete di linguaggio Java. Il "vero" server web è stato infatti scritto in questo linguaggio, e collaudato in laboratorio, dagli autori del progetto. Il testo dell'applicazione server è allegato in coda al progetto. Anche su

questa stazione l'interfaccia Ethernet è stata rinumerata come 140.105.18.1 per l'occasione, ma soltanto sul sistema operativo Linux (la partizione NTFS non è stata minimamente alterata).

## La rete interna

Questa rete è composta da quattro stazioni.

Due di queste sono degli i486 sui quali sono stati installati all'origine Windows95, Internet Explorer, i servizi di client posta e un certo numero di applicativi che consentono il download, l'upload e la modifica delle configurazioni di switch e router; l'indirizzo IP di queste è stato cambiato per l'occasione (gli indirizzi sono quelli della rete 140.105.17.0, netmask 255.255.255.0), in modo da presentare quei requisiti che non consentono l'uscita dal router per i servizi di posta.

La terza stazione è un i486 sulla quale è stato installato RedHat Linux 5.2, sulla quale sono attivi i servizi di Domain Name Server, e ciò per evitare di dover utilizzare direttamente gli indirizzi IP delle macchine.

Sulla rimanente stazione, un portatile Compaq provvisto di docking station, è stato installato all'origine Windows98; inoltre sono stati installati i servizi di client e di server di posta (protocolli POP3 e SMTP ai porti TCP 25 e 110) per le caselle postali "interne". Questa stazione è prevista come "ponte" per i servizi di posta tra l'"interno" e il "resto del mondo", e ciò per impedire l'uso di server di posta esterni a scopo di sicurezza e di evitare congestioni della linea.

## Il programma di serving web

Si è deciso di scrivere un programmino di server web in java per alcune valide ragioni:

- ❖ il linguaggio java è portabile in modo quasi indolore da una piattaforma all'altra: ciò significa che un programma scritto su macchine Linux può funzionare, sia se compilato da listato (file .java) sia se precompilato su un'altra piattaforma (file .class), su quasi qualsiasi stazione fornita di interprete (Java Virtual Machine)
- ❖ il programma stesso occupa poche risorse di macchina una volta messo in esecuzione: ciò significa che potrebbe "forzare" molte volte prima di esaurire le risorse disponibili sulla stazione su cui viene fatto girare, potendo pertanto servire molte richieste contemporaneamente, compatibilmente con la velocità del sistema
- ❖ le caratteristiche implementate nel programma lo rendono capace di discernere tra protocollo HTTP 1.0 e HTTP 1.1, permettono lo scaricamento di immagini in formati compatibili con Internet (piccole, fortemente compresse) e di pagine in formato documentale o HTML, nonché di eseguibili; tutto ciò partendo da alcuni esempi integrati insieme; inoltre fornisce un output sia a video sia su un file di log separato, gestito contemporaneamente

Tale programma è stato scritto e testato utilizzando una stazione Mandrake Linux 8.1 su cui è stato installato l'apposito pacchetto Java Software Development Kit (scaricato dall'Internet).

Il collaudo è stato fatto utilizzando una stazione WinNT Workstation con Internet Explorer (scoprendo in un secondo momento che purtroppo questo usa HTTP 1.0), e chiamando il server con il suo indirizzo IP in formato numerico, dal momento che il DNS universitario potrebbe non gestire correttamente le chiamate dirette a duinfX.univ.trieste.it, ma accetta in ogni caso gli indirizzi IP della macchina Linux agente da server.

Il programma server è stato ulteriormente collaudato in laboratorio ANL utilizzando la rete creata per l'occasione, sempre utilizzando gli indirizzi IP delle stazioni utilizzate perché al momento del collaudo il server DNS non era ancora stato programmato, ma il router già era stato protetto, e così pure le VLAN dello switch. Il testo del programma, che gira in ogni caso in modalità testuale, con funzionalità di monitoring a video e di file di log, è accluso come allegato 1.

## Allegato 1

### Il testo dell'applicazione server

#### jserver.java

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

//Sintassi: jserverhttp nomedirfile portoascolto
//predefinito: ".", 80

public class jserver extends Thread{
    Socket theConnection;
    static File rootDocs;
    static String indexFile = "index.html";

    public jserver(Socket s) {theConnection=s;}

    public static void main(String[] args){
        int thePort;
        ServerSocket ss;

        // richiama la rootDocs
        try{rootDocs=new File(args[0]);} catch (Exception e) {rootDocs=new File("."); }

        // richiama il porto
        try{thePort=Integer.parseInt(args[1]);
        if (thePort<0 || thePort>65535) thePort=80;}
        catch (Exception e) {thePort=80;}

        try{
            ss=new ServerSocket(thePort);
            System.out.println("Si accettano connessioni sul porto "+ss.getLocalPort());
            System.out.println("La radice delle pagine web è "+rootDocs);
            while(true) {
                jserver j=new jserver(ss.accept());
                j.start();
            } //while
        } catch (IOException e) {
            System.err.println("Il server si è incasinato, fine del discorso.");
        } //trycatch
    } //jserver

    public String guessContentTypeFromName(String name){
        if (name.endsWith(".txt") || name.endsWith(".java")) return "text/plain";
        else if (name.endsWith(".htm") || name.endsWith(".html")) return "text/html";
        else if (name.endsWith(".jpg") || name.endsWith(".jpeg") || name.endsWith(".jpe")) return "image/jpeg";
        else if (name.endsWith(".tif") || name.endsWith(".tiff")) return "image/tiff";
        else if (name.endsWith(".gif")) return "image/gif";
        else if (name.endsWith(".class")) return "application/octet-stream";
        else return "text/plain";
    } //guessContent...

    public void run() {
        String action;
        String namedom;
        String method;
        String content = "";
        String version = "";
        File theFile;
        byte[] theData = null;
        try{
            PrintStream ostr=new PrintStream(theConnection.getOutputStream(), true);
            BufferedReader istr=new BufferedReader(new InputStreamReader(theConnection.getInputStream()));
            String get=istr.readLine();
            action=get;
            StringTokenizer str=new StringTokenizer(get);
            method=str.nextToken();
            boolean isDir=false;

            System.out.println(action);
            if (method.equals("GET")) {
                String file=str.nextToken();
                if (file.endsWith("/")) {
                    file+=indexFile;
                    isDir=true;
                }
                content=guessContentTypeFromName(file);
                if (str.hasMoreTokens()) version=str.nextToken();
                if (!(version.equals("HTTP/1.1")||version.equals("HTTP/1.0")))) {
                    print_nAcc(ostr);
                    ostr.println("<HTML><HEAD><TITLE>Not Acceptable</TITLE></HEAD>");
                }
            }
        }
    }
}
```

```

    ostr.println("<BODY><H1>HTTP Error 406: Not Acceptable</H1></BODY></HTML>");
    ostr.close();
    process_log(action, 406, 0);
    System.out.println("Not Acceptable");
    Exception e1 = new Exception();
    throw e1;
}
if (isDir) {
    action=method+" "+file+" "+version;
}

while ((get = istr.readLine()) != null) {
// header
    str=new StringTokenizer(get);
    if ((version.equals("HTTP/1.1"))&&(str.equals("Host:"))) {
        namedom=str.nextToken();
    }
    else
    {
        if (!(version.equals("HTTP/1.0")))
        {
            print_bad(ostr, version);
            ostr.println("<HTML><HEAD><TITLE>Bad Request</TITLE></HEAD>");
            ostr.println("<BODY><H1>HTTP Error 400: Bad Request</H1></BODY></HTML>");
            ostr.close();
            process_log(action, 400, 0);
            System.out.println("Bad Request");
            Exception e1 = new Exception();
            throw e1;
        }
    }

    if (get.trim().equals("")) break;
} //while

try {
    theFile=new File (rootDocs, file.substring(1, file.length()));
    FileInputStream fiStr=new FileInputStream(theFile);
    theData= new byte[(int) theFile.length()];

    fiStr.read(theData);
    fiStr.close();
    print_ok(ostr, content, theData.length, version);
    ostr.write(theData);
    ostr.close();
    process_log(action, 200, theData.length);
} //try
catch (IOException e) {
    if (version.startsWith("HTTP/")) {
        print_nf(ostr, version);
        ostr.println("<HTML><HEAD><TITLE>File Not Found</TITLE></HEAD>");
        ostr.println("<BODY><H1>HTTP Error 404: File Not Found</H1></BODY></HTML>");
        ostr.close();
        System.out.println("File Not Found");

        process_log(action, 404, 0);
    } //if
} //catch
} //if method equals get

else
if (method.equals("HEAD")) {
    String file=str.nextToken();
    if (file.endsWith("/"))
    {
        file+=indexFile;
        isDir=true;
    }
    content=guessContentTypeFromName(file);
    if (str.hasMoreTokens()) version=str.nextToken();
    if (!(version.equals("HTTP/1.1"))||version.equals("HTTP/1.0"))
    {
        print_nAcc(ostr);
        ostr.println("<HTML><HEAD><TITLE>Not Acceptable</TITLE></HEAD>");
        ostr.println("<BODY><H1>HTTP Error 406: Not Acceptable</H1></BODY></HTML>");
        ostr.close();
        process_log(action, 406, 0);
        System.out.println("Not Acceptable");
        Exception e1 = new Exception();
        throw e1;
    }

    if (isDir)
    {
        action=method+" "+file+" "+version;
    }
}

```

```

while ((get = istr.readLine()) != null) {
// header
str=new StringTokenizer(get);
if ((version.equals("HTTP/1.1"))&&(str.equals("Host:"))) {
    namedom=str.nextToken();
}
else
{
    if (!(version.equals("HTTP/1.0")))
    {
        print_bad(ostr, version);
        ostr.println("<HTML><HEAD><TITLE>Bad Request</TITLE></HEAD>");
        ostr.println("<BODY><H1>HTTP Error 400: Bad Request</H1></BODY></HTML>");
        ostr.close();
        process_log(action, 400, 0);
        System.out.println("Bad Request");
        Exception e1 = new Exception();
        throw e1;
    }
}
if (get.trim().equals("")) break;
} //while

try {
theFile=new File (rootDocs, file.substring(1, file.length()));
FileInputStream fiistr=new FileInputStream(theFile);
theData= new byte[(int) theFile.length()];

fiistr.read(theData);
fiistr.close();
print_ok(ostr, content, theData.length, version);
ostr.close();
process_log(action, 200, theData.length);
} //try
catch (IOException e) {
if (version.startsWith("HTTP/")) {
    print_nf(ostr, version);
    ostr.println("<HTML><HEAD><TITLE>File Not Found</TITLE></HEAD>");
    ostr.println("<BODY><H1>HTTP Error 404: File Not Found</H1></BODY></HTML>");
    ostr.close();
    process_log(action, 404, 0);
    System.out.println("File Not Found");
} //if
} //catch
} //if method equals head

else {
if (version.startsWith("HTTP/")) {
    print_nimp(ostr, version);
    ostr.println("<HTML><HEAD><TITLE>Not Implemented</TITLE></HEAD>");
    ostr.println("<BODY><H1>HTTP Error 501: Not Implemented</H1></BODY></HTML>");
    ostr.close();
    process_log(action, 501, 0);
    System.out.println("Not Implemented");
} //end if
} //if method doesn't equal get or head
} //try connection
catch (Exception e1) {
} // do nothing

try {theConnection.close();} catch (IOException e) {} // e se si pianta qui?
} //run

void print_ok(PrintStream str,String cont,int len,String ver)
{
Date now=new Date();
if (ver.equals("HTTP/1.1"))
{
    str.print ("HTTP/1.1 200 OK\n\r");
}
else
{
    str.print ("HTTP/1.0 200 OK\n\r");
}
str.print ("Date: "+now+"\n\r");
str.print ("Server: jserverhttp 1.0\r\n");
str.print ("Content-length: "+len+"\r\n");
str.print ("Content-type: "+cont+"\r\n");
if (ver.equals("HTTP/1.1"))
{
    str.print ("Connection: close\r\n");
}
str.print ("\r\n");
}

```

```

void print_nimp(PrintStream str, String ver)
{
    Date now=new Date();
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("HTTP/1.1 501 Not Implemented\n\r");
    }
    else
    {
        str.print ("HTTP/1.0 501 Not Implemented\n\r");
    }
    str.print ("Date: "+now+"\n\r");
    str.print ("Server: jserverhttp 1.0\r\n");
    str.print ("Content-type: text/html\r\n");
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("Connection: close\r\n");
    }
    str.print ("\r\n");
}

```

```

void print_nf(PrintStream str, String ver)
{
    Date now=new Date();
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("HTTP/1.1 404 File Not Found\n\r");
    }
    else
    {
        str.print ("HTTP/1.0 404 File Not Found\n\r");
    }
    str.print ("Date: "+now+"\n\r");
    str.print ("Server: jserverhttp 1.0\r\n");
    str.print ("Content-type: text/html\r\n");
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("Connection: close\r\n");
    }
    str.print ("\r\n");
}

```

```

void print_bad(PrintStream str, String ver)
{
    Date now=new Date();
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("HTTP/1.1 400 Bad Request\n\r");
    }
    else
    {
        str.print ("HTTP/1.0 400 Bad Request\n\r");
    }
    str.print ("Date: "+now+"\n\r");
    str.print ("Server: jserverhttp 1.0\r\n");
    str.print ("Content-type: text/html\r\n");
    if (ver.equals("HTTP/1.1"))
    {
        str.print ("Connection: close\r\n");
    }
    str.print ("\r\n");
}

```

```

void print_nAcc(PrintStream str)
{
    Date now=new Date();
    str.print ("HTTP/1.0 406 Not Acceptable\n\r");
    str.print ("Date: "+now+"\n\r");
    str.print ("Server: jserverhttp 1.0\r\n");
    str.print ("Content-type: text/html\r\n");
    str.print ("\r\n");
}

```

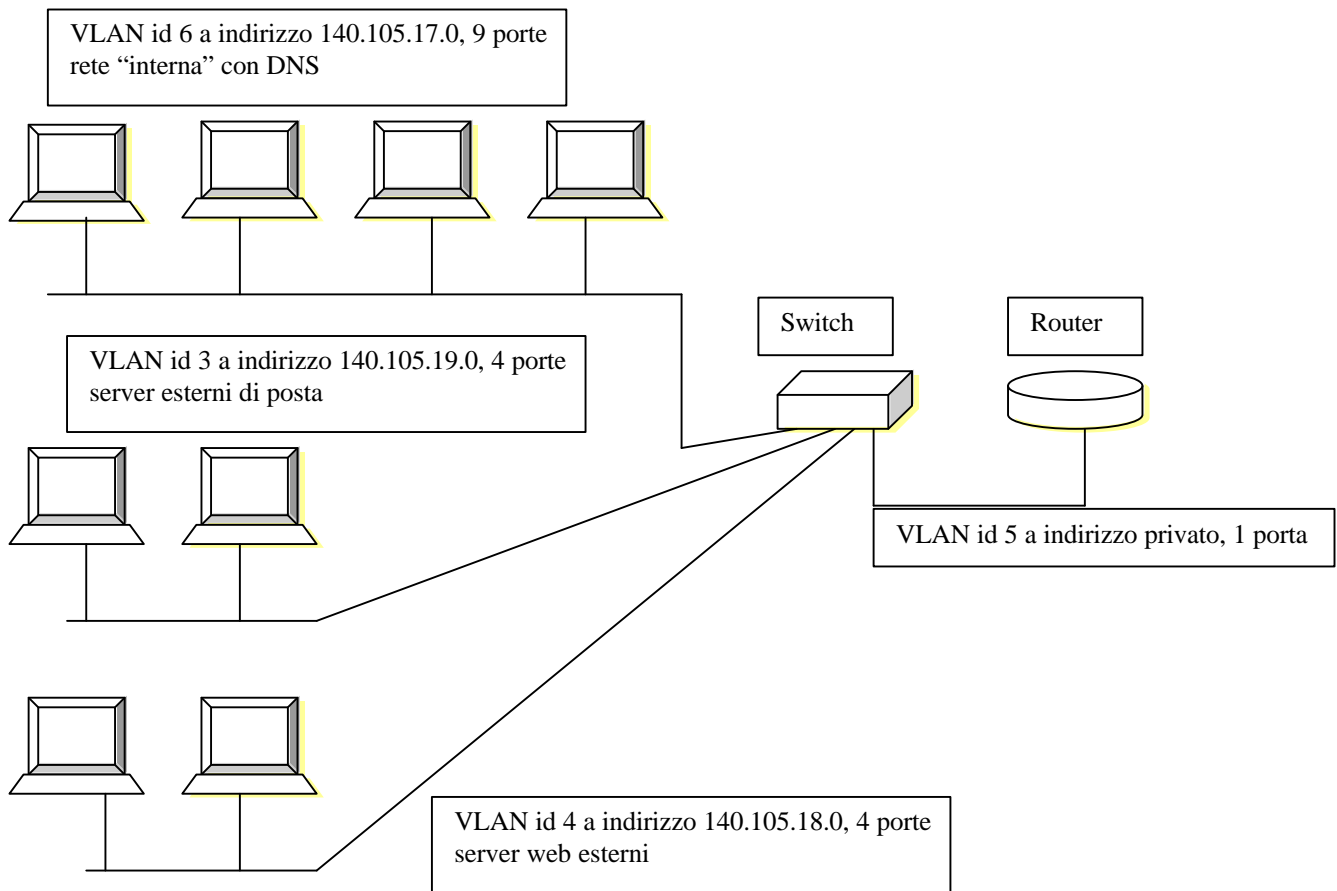
```

public void process_log(String action,int status,int length)
{
    try{
        InetAddress client=theConnection.getInetAddress();
        Date now=new Date();
        PrintStream log=null;
        String FileLog=rootDocs.toString()+"/jhttplog.txt";

```

```
log=new PrintStream(new FileOutputStream(FileLog, true), true);
synchronized(log){
    log.print(client);
    log.print(' ');
    log.print(now);
    log.print(' ');
    log.print(action);
    log.print(' ');
    log.print(status);
    log.print(' ');
    log.println(length);
}
log.close();
}
catch (IOException e){
    System.err.println("Non si riesce a gestire il file di log");
}
}
} //class
```

Allegato 2  
Disegni e diagrammi



Disegno 1: il layout della rete, eseguita con i materiali a disposizione

Tabella 1  
Le VLAN dello switch

nome della VLAN	ID della VLAN	indirizzo di base	porte di appartenenza	porte di uscita	funzione della VLAN
Default VLAN	1	(nessuno)	tutte tranne la 17	nessuna	cestino spazzatura
MailServerVLAN	3	140.105.19.0	9-12	9-12, 17	ospita i server mail esterni
WebServerVLAN	4	140.105.18.0	13-16	13-16, 17	ospita i server web
RouterVLAN	5	(privato)	17	1-16, 17, 18	ospita il router
dnsVLAN	6	140.105.17.0	1-8, 18	1-8, 17, 18	rete "interna" con dns