PSP05.- GENERACIÓN DE SERVICIOS EN RED

1.- INTRODUCCIÓN.

Los servicios son programas auxiliares utilizados en un sistema informático que permiten gestionar una colección de recursos y prestar su funcionalidad a los usuarios y aplicaciones. Por ejemplo, cuando se envía un documento a una impresora se está utilizando un servicio de impresión. Este servicio permite gestionar y compartir la impresora en red. El único acceso que se tiene al servicio está formado por el conjunto de operaciones que ofrece. Por ejemplo, un servicio de ficheros ofrece operaciones de lectura, escritura o borrado de ficheros.

Todos los servicios de Internet implementan una relación cliente-servidor. En esta unidad didáctica se estudiarán estos servicios y se usará Java para programar clientes de los servicios de Internet más frecuentemente usados.

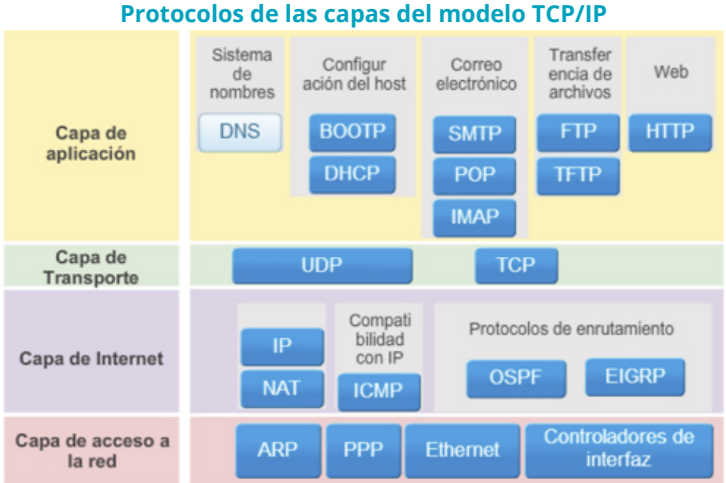
2.- PROTOCOLOS ESTÁNDAR DE COMUNICACIONES EN RED.

2.1.- EL MODELO TCP/IP.

El modelo TCP/IP está compuesto por 4 capas o niveles. La capa de aplicación maneja los protocolos de alto nivel que implementa servicios tales como:

- Conexión remota: Telnet.
- Correo electrónico: SMTP.
- Acceso a ficheros remotos: FTP, NFS, TFTP.
- Resolución de nombres de ordenadores: DNS, WINS.
- World Wide Web: HTTP.

Protocolos de las capas del modelo TCP/IP



Todas las aplicaciones que implementan TCP/IP se basan en un modelo cliente-sevidor:

- **Telnet** (Telecomunication Network): emulación de terminal. Permite a un usuario acceder a una máquina remota y manejarla como si estuviese conectado a ella. Este servicio ha sido sustituido por SSH, ya que envía la información entre cliente y el servidor en forma de texto plano y esto no es seguro. Usado frecuentemente para arreglar fallos de máquinas remotas o para realizar consultas a distancia.
- **SMTP** (Simple Mail Transfer Protocol): protocolo simple de transferencia de correo electrónico. Este estándar especifica el formato exacto de los mensajes que un cliente en una máquina debe enviar al servidor en otra. Administra la transmisión de correo electrónico a través de las redes informáticas.
- **FTP** (File Tranfer Protocol): protocolo de transferencia de ficheros. Permite la transferencia de ficheros entre un cliente y servidor a través de Internet. En los comienzos de Internet se utilizaban los servidores FTP públicos para la descarga de programas. Este servicio está orientado a conexión.
- **TFTP** (Trivial File Tranfer Protocol): protocolo trivial de transferencia de ficheros. A diferencia del protocolo FTP, no necesita autenticación para la transferencia de ficheros entre el cliente y el servidor. Se suele utilizar para transferir ficheros entre ordenadores en una red local en la que no es necesaria una autenticación. Es un servicio no orientado a conexión que utiliza el protocolo UDP.

- **HTTP** (HyperText Transference Protocol): protocolo de transferencia de Hipertexto. Lo usan los navegadores web para realizar peticiones a los servidores web y para recibir las respuestas de ellos. Se trata de un protocolo que especifica los mensajes involucrados en un intercambio petición-respuesta, los métodos, argumentos, resultados y las reglas para representar todo ello en los mensajes.
- **NFS** (Network File System): sistema de ficheros de red que permite a los usuarios el acceso en línea a ficheros que se encuentran en sistemas remotos. De esta forma, el usuario accede a un fichero como si éste fuera un fichero local. Fue desarrollado por Sun Microsystems.
- **SNMP** (Simple Network Management Protocol): protocolo simple de administración de red. Protocolo utilizado para intercambiar información entre los diferentes dispositivos de una red. Permite a los administradores monitorizar, controlar y supervisar el funcionamiento de la red.
- **DNS** (Domain Name System): sistema de nombres de dominio. Sistema que usa servidores distribuidos a lo largo de la red para resolver el nombre de un host IP (nombre de ordenador + nombre de subdominio + nombre de dominio) en una dirección IP. De esta forma, no hay que recordar el nombre del dominio.

3.- COMUNICACIÓN CON UN SERVIDOR FTP.

FTP es una de las herramientas más útiles para el intercambio de ficheros entre diferentes ordenadores y es la forma más habitual de publicación en Internet.

Para usar FTP para transferir ficheros entre dos ordenadores, cada uno debe tener un papel; es decir, uno debe de ser el cliente FTP y el otro el servidor FTP. El cliente envía comandos al servidor (subir, bajar, borrar ficheros, crear un directorio) y el servidor los lleva a cabo. Es posible imaginarse al servidor como un gran contenedor en el que se pueden encontrar gran cantidad de ficheros y directorios.

Existen básicamente 2 formas de acceder a través de FTP:

- **Acceso anónimo:** cuando la conexión con la máquina servidora la realiza un usuario sin autentificar y con escasos privilegios en el servidor. En este caso el usuario es recluido a un directorio público donde solo se le permite descargar ficheros.
- **Acceso autorizado:** el usuario que realiza la conexión con la máquina servidora está registrado y tiene ciertos privilegios en el servidor. En este caso, y una vez autentificado, el usuario es recluido a su directorio personal donde puede subir y bajar ficheros; normalmente se le asigna una cuota de espacio.

FTP utiliza 2 **conexiones** TCP distintas: una **conexión de control** y **otra conexión de transferencia de datos.** La primera se encarga de iniciar y mantener la comunicación entre el cliente y el servidor. La segunda se encarga de enviar datos entre cliente y servidor y solo existe cuando hay datos que transmitir.

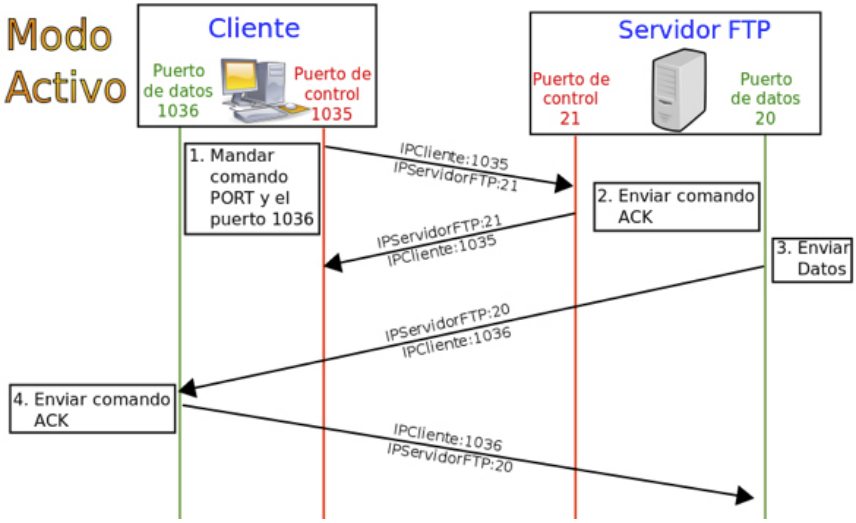
Cuando un cliente se conecta a un servidor FTP, el cliente emplea un puerto aleatorio pero el servidor se conecta al puerto 21. Para la transferencia de datos no se utilizan los mismos puertos, por lo que el cliente obtiene un nuevo puerto y el servidor suele usar el puerto 20.

Existen dos **modos de conexión** de los clientes FTP a los servidores FTP:

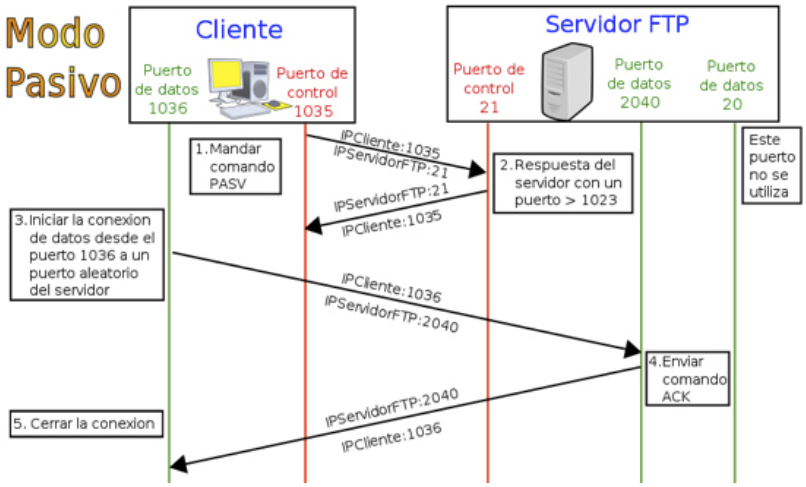
- **Modo activo:** el servidor siempre crea la conexión de datos en su puerto 20, mientras que en el lado del cliente la conexión de datos se asocia a un puerto aleatorio mayor que el 1024. Para ello, el cliente manda un comando PORT al servidor por la conexión de control, indicándole ese número de puerto, de manera que el servidor pueda abrirle en el puerto especificado una conexión de datos por donde se transferirán los archivos. Esto tiene un problema de seguridad, ya que la máquina cliente debe estar dispuesta a aceptar cualquier conexión de entrada en un puerto superior al 1024, con los problemas que ello implica si se tiene el equipo conectado a una red insegura como Internet. De hecho, los cortafuegos que se instalen en el equipo para evitar ataques seguramente rechazarán esas conexiones aleatorias. Para solucionar esto está el modo pasivo.
- **Modo pasivo:** cuando el cliente envía un comando PASV sobre la conexión de control, el servidor FTP le indica por la conexión de control el puerto (mayor a 1024 del servidor; por ejemplo, el puerto 2040) al que debe conectarse el cliente. El cliente inicia una conexión desde el puerto siguiente al puerto de control (ejemplo: 1036) hacia el puerto del servidor especificado anteriormente (ejemplo: 2040).

Antes de cada nueva transferencia, tanto en el modo Activo como en el Pasivo, el cliente debe enviar otra vez un comando de control (PORT o PASV, según el modo en el que haya conectado), y el servidor recibirá esa conexión de datos en un nuevo puerto (aleatorio si es en modo pasivo o por el puerto 20 si es en modo activo).

Modo de conexión activo de un cliente



Modo de conexión pasivo de un cliente



3.1.- C# PARA COMUNICAR CON SERVIDOR FTP.

Para comunicarnos con un servidor FTP vamos a utilizar la clase FTPClient del paquete FluentFTP. Es recomendable no utilizar directamente la clase FluentFTP en el proyecto. Lo ideal sería crear una clase que implemente una interfaz con las funciones que necesitemos (conectar, subir, descargar ...). De esta manera, si en futuro hay cambios en el paquete FluentFTP o si hay que utilizar otro paquete, únicamente habrá que cambiar la implementación en la nueva clase.

La clase FTPClient encapsula toda la funcionalidad necesaria para almacenar y recuperar ficheros de un servidor FTP. Esta clase se encarga de todos los detalles de bajo nivel de la interacción con un servidor FTP. Para poder utilizarla primero es necesario realizar la conexión al servidor, realizar las operaciones de transferencia y, cuando termine el proceso, cerrar la conexión.

FTPClient cuenta con diferentes contructores, entre los que se encuentran los siguientes:

- ```
  public FtpClient(string host)
  ```
- ```
  public FtpClient(string host, string user, string pass)
  ```
- ```
  public FtpClient(string host, int port, string user, string pass)
  ```

Algunos de los métodos de esta clase aparecen a continuación:

- **Connect():** método para establecer la conexión con el servidor FTP.
- **Disconnect()** – método para realizar la desconexión del servidor.
- **Execute()** – método para ejecutar comandos que son soportados por FTPClient.
- **IsConnected** - método para comprobar si la conexión está establecida
  ```

- **GetListing()** - método para obtener el listado de ficheros de un directorio. Existe la posibilidad de utilizar FtpListOption.Recursive para poder listar los subdirectorios del directorio.
- **GetNameListing()** -método para obtener los nombres de los ficheros de un directorio
- **UploadFile()** – método para subir un archivo del sistema de archivos local al servidor. Devuelve verdadero si tuvo éxito, falso si falló o el archivo no existe. Se lanzan excepciones para errores críticos. Admite archivos muy grandes ya que carga datos en fragmentos. Opcionalmente, verifica el hash de un archivo y vuelve a intentar la transferencia si el hash no coincide.
- **DownloadFile()** – método para descargar un archivo del servidor al sistema de archivos local. Devuelve verdadero si tuvo éxito, falso si falló o el archivo no existe. Se lanzan excepciones para errores críticos. Admite archivos muy grandes ya que descarga datos en fragmentos. Los directorios locales se crean si no existen. Opcionalmente, verifica el hash de un archivo y vuelve a intentar la transferencia si el hash no coincide.
- **UploadFiles()** – método para realizar la subida de más de un archivo local al servidor. Devuelve el número de archivos subidos. Returns the number of files uploaded. Skipped files are not counted. User-defined error handling for exceptions during file upload (ignore/abort/throw). Opcionalmente, verifica el hash de un archivo y vuelve a intentar la transferencia si el hash no coincide. Es más eficiente utilizar UploadFiles() que llamar varias veces a UploadFile().
- **DownloadFiles()** – método para descargar más de un archivo del servidor al sistema de archivos local. Devuelve el número de archivos descargados. Opcionalmente, verifica el hash de un archivo y vuelve a intentar la transferencia si el hash no coincide.
- **GetWorkingDirectory()** – método que devuelve la ruta absoluta del directorio de trabajo.
- **SetWorkingDirectory()** – método que permiter establecer el directorio de trabajo.
- **DirectoryExists()**
- **CreateDirectory()**
- **DeleteDirectory()** – método para borrar un directorio en el servidor. Si el directorio no está vacio se realiza un borrado recursivo.
- **MoveDirectory()** – método para mover un directorio en el servidor. Si se utiliza FtpExists.Overwrite se sobrescribe el contenido del directorio destino.
- **FileExists()**
- **DeleteFile()**
- **MoveFile()** – método para mover un fichero de una carpeta a otra dentro del servidor. Si se utiliza FtpExists.Overwrite se borra el fichero en la carpeta destino antes de mover la el fichero.
- **Rename()** – método para cambiar el nombre de ficheros y directorios en el servidor. En un método de bajo nivel, por lo que no debería utilizarse en la mayoría de los casos. El preferible utilizar MoveFile() o MoveDirectory().

3.2.- EJMPLO SUBIR FICHEROS AL SERVIDOR.

En los siguientes ejemplos se asume que se dispone del servidor FTP debidamente instalado y configurado.

En general, y para subir ficheros a los servidores FTP suele ser necesario disponer de un usuario, su clave, un espacio en el servidor así como los permisos necesarios para dicho espacio. Además, dentro del servidor es necesario definir un directorio donde se van a subir los archivos y, en origen, la ubicación y nombre de los ficheros que se pretenden subir al servidor.

```
using System;
using System.Net;
using System.Threading.Tasks;
using FluentFTP;

namespace FTP
{

    class Program
    {
        public static async Task Main()
        {
            FtpClient client = new FtpClient("192.168.33.10");

            client.Credentials = new NetworkCredential("user123", "user123");

            client.Connect();

            //no espero a que se realice la subida
            client.UploadFileAsync("prueba.txt","pruebaAsincrona.txt");
            //espero a que se realice la subido, la llamada no es asyn
            bool subido = client.UploadFile("prueba.txt", "pruebaNoAsinc.txt", existsMode: FtpExists.Overwrite, creat
```

```
            subido = await client.UploadFileAsync("prueba.txt", "pruebaNoAsinc.txt", existsMode: FtpExists.Overwrite,

            client.Disconnect();


        }
    }
}
```

3.3.- EJMPLO RENOMBRAR Y BORRAR FICHEROS.

```
using System;
using System.Net;
using System.Threading.Tasks;
using FluentFTP;

namespace FTP
{

    class Program
    {
        public static async Task Main()
        {
            FtpClient client = new FtpClient("192.168.33.10", "user123", "user123");

            client.Connect();

            client.MoveFile("subido.txt", "nombre_cambiado.txt");

            client.DeleteFile("nombre_cambiado.txt");

            client.Disconnect();


        }
    }
}
```

3.4.- EJMPLO LISTAR ARCHIVOS DEL DIRECTORIO.

```
using System;
using System.Net;
using System.Threading.Tasks;
using FluentFTP;

namespace FTP
{

    class Program
    {
        public static async Task Main()
        {
            FtpClient client = new FtpClient("192.168.33.10", "user123", "user123");

            client.Connect();

            var listado = client.GetListing();

            foreach(var item in listado)
            {
                Console.WriteLine(item);
            }

            client.Disconnect();


        }
    }
}
```

El resultado en mi caso es el siguiente:

```
DIR    cliente      Modified : 09/08/2019 21:29:00
FILE   nuevo_nombre.txt      (23 bytes)      Modified : 01/10/2019 8:23:00
DIR    servidor      Modified : 09/08/2019 12:38:00
FILE   subida.txt      (26 bytes)      Modified : 30/09/2019 8:40:00
FILE   subido.txt      (32 bytes)      Modified : 30/09/2019 8:26:00
```

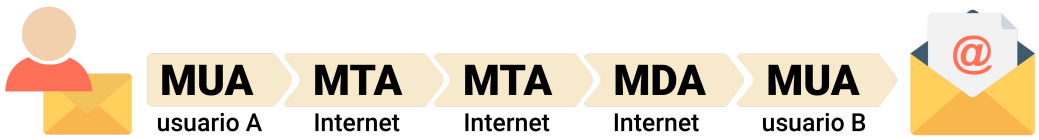
4.- COMUNICACIÓN CON UN SERVIDOR SMTP.

SMTP (Simple Mail Transfer Protocol) es el protocolo estándar de Internet para el intercambio de correo electrónico. Funciona con comandos de texto que se envían al servidor SMTP (por defecto, usa el puerto 25). A cada comando que envía el cliente le sigue una respuesta del servidor, compuesto por un número y un mensaje descriptivo. Las especificaciones de este protocolo se definen en la RFC 2821.

4.1.- INSTALACIÓN DE UN SEVIDOR DE CORREO ELETRÓNICO.

En el servicio de correo electrónico se distinguen 3 agentes diferentes que intervienen en el proceso de envío, recepción y lectura de los mensajes:

- **MUA:** agente de usuario de correo.
- **MTA:** agente de transporte de correo.
- **MDA:** agente de entrega de correo.



Autor *(link: https://www.google.es)* (Licencia) *(link: https://www.google.es)* Procedencia *(link: https://www.google.es)*

El MTA entrega el correo al **MDA**, el cual almacena el correo en el buzón que el usuario tiene en el servidor. Los protocolos que usan los MDAs suelen ser POP3 o IMAP.

Finalmente, el usuario B accede mediante su cliente de correo electrónico o MUA a su buzón de correo electrónico y visualiza o descarga el correo electrónico enviado por el usuario A.

Los servidores SMTP son programas que permiten enviar correo electrónico a otros servidores SMTP. El servidor de correo **Argosoft** permite la instalación de un servidor de correo SMTP (MTA) y también de un servidor de correo POP3 (MDA). En la web http://argosoft-mail-server.uptodown.com/ está disponible un archivo para poder realizar la instalación de este servidor de correo.

A continuación, se va a realizar una instalación en local del servidor de correo Argosot, la cual se simplificará desactivando momentáneamente el Firewall del sistema operativo. Después de la instalación, se habrá instalado, entre otras cosas, un servidor SMTP y otro POP3.

Una vez instalado, se procederá a la configuración de ciertas opciones desde Tools/Options/General. Estas opciones pueden ser las siguientes:

- Servidor de DNS: 8.8.8.8 (por ejemplo)
- Automatically Start the Server
- Permitir Relay
- Local Host: 172.20.105.120

En la siguiente figura se muestra esta configuración:

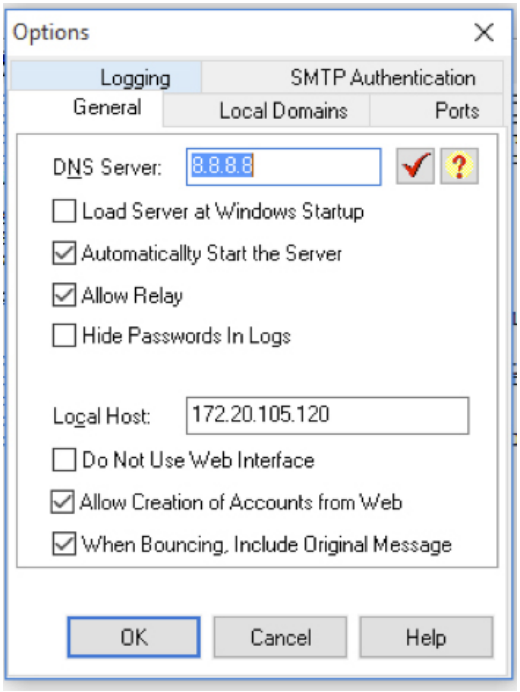


Figura 5.1 – Configuración básica del servidor de correo Argosoft
(link: https://www.google.es)

Una configuración adicional se consigue activando las siguientes opciones desde la opción Tools/Options/Logging del servidor Argosoft:

- Log SMTP commands
- Log SMTP Conversations with Exchangers
- Log to File

A continuación se dan de alta en el servidor dos usuarios: pepe y juan, desde la opción Tools/Users de Argosoft. La contraseña de cada uno de ellos coincide con el nombre del usuario.

Normalmente, cuando se crea una cuenta de correo en un proveedor de servicios de internet, el proveedor proporciona los datos del servidor POP3 (o IMAP) y del servidor SMTP. Estos son necesarios para configurar clientes de correo como Microsoft Outlook, Mozilla Thunderbird, Eudora, etc. El primero se usa para recibir los mensajes (configuración del correo entrante) y el segundo para enviar los mensajes (configuración del correo saliente). Se puede usar el servidor SMTP recién instalado de Argosoft para enviar correos, en vez de utilizar el proporcionado por el proveedor de internet.

4.2.- USO DE TELNET PARA COMUNICAR CON UN SERVIDOR SMTP.

A continuación, se muestra cómo, usando Telnet, se puede enviar un correo electrónico de forma manual a través del puerto 25 del servidor SMTP de Argosoft.

Algunos de los comandos que se van a usar aparecen en la siguiente tabla:

Comando	Función
HELO o EHLO	Se utiliza para abrir una sesión con el servidor
MAIL FROM: origen	Indica quién envía el correo electrónico
RCPT TO: destino	Indica a quién se envía el correo electrónico
DATA mensaje	Representa el mensaje a enviar. Se puede añadir un Subject: para indicar el título. Los datos terminarán cuando se escriba un punto “.” en una línea
QUIT	Cierra la sesión

HELP	Muestra la lista de comandos SMTP que el servidor admite

Estando en la línea de comandos de MS-DOS, se introduce el siguiente comando:

```
telnet 172.20.105.120 25
```

El servidor responde con la siguiente línea:

```
220 172.20.105.120 ArgGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
```

Las respuestas del servidor Argosoft suelen ser un número de 3 dígitos, donde cada uno tiene un significado especial:

- Los que empiezan por 2 (220, 250, ...) indican que la acción se ha completado con éxito.
- Los que empiezan por 3 (354) indican que el comando ha sido aceptado, pero la acción solicitada está suspendida a la espera de recibir más información; en este caso se usa un grupo de secuencias DATA.
- Los que empiezan por 4 indican que el comando no ha sido aceptado, pero se puede volver a escribir de nuevo.
- Los que empiezan por 5 indican que el comando no ha sido aceptado y la acción no se ha realizado.

A continuación se puede empezar a escribir los comandos referenciados en la tabla anterior que sean necesarios. Primero se abre la sesión con el comando HELO, luego se escribe el origen (MAIL FROM:) y el destino del mensaje (RCPT TO). Por cada comando que se le envía al servidor, éste va respondiendo. A continuación, mediante el comando DATA se envía el mensaje. Para finalizar el mensaje hay que escribir una línea con solo un punto. Para terminar la sesión, se introduce el comando QUIT.

En el siguiente ejemplo se ve la secuencia de comandos (los comandos aparecen en negrita) con los que se consigue enviar un mensaje desde la cuenta de correo electrónico dm2prosciudadjardin@gmail.com hacia el usuario pepe del servidor Argosoft.

```
HELO
250 Welcome [172.20.105.120], pleased to meet you
MAIL FROM:dm2prosciudadjardin@gmail.com
250 Sender "dm2prosciudadjardin@gmail.com" OK...
RCPT TO: pepe
250 Recipient "pepe" OK...
DATA
354 Enter mail, end with "." on a line by itself
Esto es un correo electronico
enviado desde un Telnet.
.
250 Message accepted for delivery. <igk3nk5pbe14j3b.291220151022@ulhi-PC>
QUIT
221 Aba he
Connection closed by foreign host.
```

4.3.- USO DE TELNET PARA COMUNICAR CON UN SERVIDOR POP3.

A continuación, se muestra cómo, usando Telnet, se puede conectar a través del puerto 110 con el servidor POP3 de Argosoft para leer el correo electrónico de un determinado usuario.

Algunos de los comandos que se van a usar aparecen en la siguiente tabla:

Comando	Función
USER login	Se escribe el login de la cuenta de usuario
PASS password	Se indica la contraseña del usuario
STAT	Muestra el número de mensajes en la cuenta
LIST	Lista los mensajes. Número y tamaño total del mensaje
RETR numero-	Visualiza el mensaje indicado en numero-mensaje

mensaje	
DELE numero-mensaje	Borra el mensaje indicado en numero mensaje. Esto tendrá efecto la próxima vez que salgamos y volvamos a entrar.
TOP numero-mensaje n	Muestra las n primeras lineas del mensaje numero-mensaje
QUIT	Cierra la sesión

En primer lugar, el usuario pepe del servidor Argosoft va a enviar al usuario juan un mensaje, para posteriormente comprobar que juan puede leerlo. En la línea de comandos de MS-DOS se introduciría:

```
telnet 172.20.105.120 25
```

La secuencia de comandos para el envío del mensaje sería:

```
HELO
250 Welcome [172.20.105.120], pleased to meet you
MAIL FROM: pepe
250 Sender "pepe” OK...
RCPT TO: juan
250 Recipient "juan" OK...
DATA
354 Enter mail, end with "." on a line by itself
Esto es una prueba de un correo
De varias lineas
enviado desde Telnet.

Fin.
.
250 Message accepted for delivery. <igk3nk5pbe14j3b.291220151022@ulhi-PC>
QUIT
221 Aba he
Connection closed by foreign host.
```

En segundo lugar, se va a acceder al contenido del mensaje recibido por el usuario juan del servidor Argosoft. En la línea de comandos de MS-DOS se introduciría:

```
telnet 172.20.105.120 110
```

El servidor responde con la siguiente línea:

```
220 172.20.105.120 ArgGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)
```

La secuencia de comandos para leer el mensaje recibido por juan sería:

```
USER juan
+OK Password required for juan
PASS juan
+OK Mailbox locked and ready
LIST
+OK
1 322
.
RETR 1
+OK 322 octets
Received: from [172.20.105.120] by DM2-PROF
(ArGoSoft Mail Server Freeware, Version 1.8 (1.8.9.1)); Tue, 29 Dec 2015 10:22:07 +0100
Esto es una prueba de un correo
de varias lineas
enviado desde Telnet.
Message-ID: <igk3nk5pbe14j3b.291220151022@ulhi-PC>
Date: Wed, 16 Nov 2016 10:22:07 +0100
```

```
Fin.
.
QUIT
+OK Aba he
Connection closed by foreign host.
```

4.4.- LIBRERÍA PARA MAILKIT.

Para comunicarnos con un servidor de correo electrónico vamos a utilizar el paquete Mailkit. De igual forma a como se planteó con la librería FluentFTP, no es recomendable utilizarla directamente en el proyecto. Lo ideal sería crear una clase que implemente una interfaz con las funciones que necesitemos (conectar, enviar correo, listar, borrar). De esta manera, si en un futuro hay cambios en el paquete Mailkit o si hay que utilizar otro paquete, únicamente habrá que cambiar la implementación en la nueva clase.

La clase **SmtpClient** encapsula toda la funcionalidad necesaria para el envío de correos electrónicos. La clase cuenta con dos constructores.

SmtpClient()	Inicializa una instancia de la clase.
SmtpClient(IProtocolLogger)	Inicializa una instancia de la clase y establece un logger.

Los métodos más importantes son los siguientes:

Authenticate(String, String, CancellationToken)	Método para realizar la autenticación utilizando nombre de usuario y contraseña.
Connect(String, Int32, SecureSocketOptions, CancellationToken)	Método para establecer la conexión con el servidor utilizando la url, el puerto y las opciones de seguridad.
ConnectAsync(String, Int32, SecureSocketOptions, CancellationToken)	Método para establecer la conexión con el servidor de forma asíncrona.
Disconnect	Método para realizar la desconexión.
DisconnectAsync	Método para realizar la desconexión de forma asíncrona.
Dispose()	Método para liberar los recursos.
Send(MimeMessage, CancellationToken, ITransferProgress)	Método para realizar el envío del mensaje.
SendAsync(MimeMessage, CancellationToken, ITransferProgress)	Método para realizar el envío del mensaje de forma asíncrona.
SendCommand	Método para realizar el envío de comandos al servidor.
SendCommandAsync	Método para realizar el envío de comandos al servidor de forma asíncrona.

La clase Pop3Client encapsula toda la funcionalidad necesaria para realizar la comunicación con un servidor POP3. La clase cuenta con dos constructores:

Pop3Client()	Inicializa una instancia de la clase.
Pop3Client(IProtocolLogger)	Inicializa una instancia de la clase y establece un logger.

Los métodos más importantes son los siguientes:

Authenticate(String, String, CancellationToken)	Método para realizar la autenticación utilizando nombre de usuario y contraseña.
Connect(String, Int32, SecureSocketOptions, CancellationToken)	Método para establecer la conexión con el servidor utilizando la url, el puerto y las opciones de seguridad.
ConnectAsync(String, Int32, SecureSocketOptions, CancellationToken)	Método para establecer la conexión con el servidor de forma asíncrona.
Disconnect	Método para realizar la desconexión.
DisconnectAsync	Método para realizar la desconexión de forma asíncrona.
Dispose()	Método para liberar los recursos.
DeleteAllMessages()	Método para marcar todos mensajes para ser borrados.
DeleteAllMessagesAsync()	Método para marcar todos mensajes para ser borrados de forma asíncrona.
DeleteMessage(int)	Método para marcar un mensaje para ser borrado.
DeleteMessageAsync(int)	Metodo para marcar un mensaje para ser borrado de forma asíncrona.
DeleteMessages(IList<Int32>)	Método para marcar varios mensajes para ser borrados.
DeleteMessages(Int32,Int32)	Método para marcar varios mensajes para ser borrados.
DeleteMessagesAsync(IList<int32>)	Método para marcar varios mensajes para ser borrados de forma asíncrona.
DeleteMessagesAsync(Int32,Int32)	Método para marcar varios mensajes para ser borrados de forma asíncrona.
GetMessage(Int32)	Método para obtener un mensaje.
GetMessageAsync(Int32)	Método para obtener un mensaje de forma asíncrona.
GetMessages(IList<int32>)	Método para obtener varios mensajes.
GetMessages(Int32,Int32)	Método para obtener varios mensajes de forma asíncrona.
Reset	Método para resetear el estado de los mensajes marcados para ser borrados.
ResetAsync	Método para resetear el estado de los mensajes marcados para ser borrados de forma asíncrona.

4.5.- EJMPLO USO DE C# PARA COMUNICAR CON UN SERVIDOR SMTP.

```
using MailKit.Net.Smtp;
using MimeKit;

namespace Correo
{
    class Program
    {
        public static void Main(string[] args)
        {
            var message = new MimeMessage();
            message.From.Add(new MailboxAddress("Unai"));
            message.To.Add(new MailboxAddress("Pepe"));
            message.Subject = "Paintball sabado";

            message.Body = new TextPart("plain")
```

```
            {
                Text = @"Hola Pepe,

Solo queria decirte que este sabado vamos a ir a jugar a paintball. Si vas a venir, me dices.

-- Unai"
            };

            using (var client = new SmtpClient())
            {
                client.Connect("169.254.192.27", 25);

                client.Send(message);

                client.Disconnect(true);

            }
        }
    }
}
```

4.6.- USO DE C# PARA COMUNICAR CON UN SERVIDOR SMTP SEGURO.

```
using MailKit.Net.Smtp;
using MailKit.Security;
using MimeKit;

namespace Correo
{
    class Program
    {
        public static void Main(string[] args)
        {
            var message = new MimeMessage();
            message.From.Add(new MailboxAddress("Unai"));
            message.To.Add(new MailboxAddress("Pepe"));
            message.Subject = "Paintball sabado";

            message.Body = new TextPart("plain")
            {
                Text = @"Hola Pepe,

Solo queria decirte que este sabado vamos a ir a jugar a paintball. Si vas a venir, me dices.

-- Unai"
            };

            using (var client = new SmtpClient())
            {
                client.Connect("169.254.192.27", 25, SecureSocketOptions.Auto);

                client.Authenticate("unai", "unai");

                client.Send(message);

                client.Disconnect(true);

            }
        }
    }
}
```

4.7.- USO DE C# PARA COMUNICAR CON UN SERVIDOR POP3.

En el correo electrónico se usan otros protocolos, además de SMTP, para funciones adicionales. Entre los más utilizados están los siguientes:

- **MIME** (Multipurpose Internet Mail Extensions): define una serie de especificaciones para expandir las capacidades limitadas del correo electrónico y, en particular, para permitir la inserción de archivos (imágenes, sonido y texto) en un mensaje.

- **IMAP** (Internet Message Access Protocol): permite acceder a los mensajes de correo electrónico almacenados en los servidores SMTP. Permite que los usuarios accedan a su correo desde cualquier equipo que tenga una conexión a Internet. Tiene alguna ventaja con respecto al protocolo POP; por ejemplo: los mensajes continúan siempre almacenados en su servidor (cosa que no ocurre con POP) y los usuarios pueden organizar los mensajes en carpetas.
- **POP** (Post Office Protocol): proporciona acceso a los mensajes de los servidores SMTP. En general, cuando se hace referencia al término POP, se hace referencia a POP3, que es la última versión.

El servidor POP3 es el **servidor de correo entrante** y en general usa el puerto 110. Al igual que SMTP funciona con comandos de texto.

4.8.- EJMPLO USO DE C# PARA LA COMUNICACIÓN CON UN SERVIDOR POP3.

```
using MailKit;
using MailKit.Net.Pop3;
using MailKit.Security;

namespace Correo
{
    class Program
    {
        public static void Main(string[] args)
        {
            using (var client = new Pop3Client(new ProtocolLogger("pop3.log")))
            {
                client.Connect("169.254.192.27", 110, SecureSocketOptions.Auto);

                client.Authenticate("pepe", "pepe");

                for (int i = 0; i < client.Count; i++)
                {
                    var message = client.GetMessage(i);

                    // escribir el mensaje en el fichero de log
                    message.WriteTo(string.Format("{0}.msg", i));


                    // marcar el mensaje para borrado
                    client.DeleteMessage(i);
                }

                client.Disconnect(true);
            }

        }
    }
}
```

5.- DISPONIBILIDAD DEL SERVICIO.

5.1.- REQUISITOS.

Es importante que las organizaciones cuantifiquen los requisitos de disponibilidad de los servicios en red para la correcta elaboración de los Acuerdos de Nivel de Servicio (SLA) con aquellos proveedores (internos o externos) que se los proporcionen.

La disponibilidad propuesta debe encontrase en línea tanto con los necesidades reales de la organización como con las posibilidades que la organización tenga en materia de Tecnologías de la Información.

Aunque en principio todas las organizaciones estarán de acuerdo con unas elevadas cotas de disponibilidad, es importante hacerles ver que una alta disponibilidad puede generar unos costes injustificados dadas sus necesidades reales. Quizás unas pocas horas sin un determinado servicio de red pueden representar poco más allá de una pequeña inconveniencia mientras que la certeza de un servicio prácticamente continuo y sin interrupciones puede requerir la replicación de sistemas u otras medidas igualmente costosas que no van a tener una repercusión real en la rentabilidad de la organización.

Para llevar a cabo eficientemente esta tarea es necesario tener en cuenta lo siguiente:

- Identificar las actividades clave del negocio de la organización.
- Cuantificar los intervalos razonables de interrupción de los diferentes servicios en red dependiendo de sus respectivos impactos.

- Determinar las franjas horarias de disponibilidad de los servicios en red (24/7, 12/5, ...).

5.2.- CALIDAD DE SERVICIO.

La monitorización de la disponibilidad del servicio y la elaboración de los informes correspondientes son dos actividades orientadas a medir la Calidad del servicio.

Desde el momento de la interrupción del servicio en red hasta su restitución o "tiempo de parada", el incidente pasa por distintas fases que deben ser individualizadamente analizadas:

- **Tiempo de detección:** es el tiempo que transcurre desde que ocurre el fallo hasta que la organización TI tiene constancia del mismo.
- **Tiempo de respuesta:** es el tiempo que transcurre desde la detección del problema hasta que se realiza un registro y diagnóstico del incidente.
- **Tiempo de reparación/recuperación:** periodo de tiempo utilizado para reparar el fallo o encontrar un "workaround" o solución temporal al mismo y devolver el sistema a la situación anterior a la interrupción del servicio.

Es importante determinar **métricas** que permitan medir con precisión las diferentes fases del ciclo de vida de la interrupción del servicio. La organización debe conocer estas métricas y dar su conformidad a las mismas para evitar malentendidos. En algunos casos es difícil determinar si el sistema está "caído o en funcionamiento" y la interpretación puede diferir entre proveedores y clientes, por lo tanto, estas métricas deben de poder expresarse en términos que el cliente pueda entender.

Algunos de los parámetros que se suelen utilizar para medir la disponibilidad del servicio son los siguientes:

- **Uptime:** periodo total durante el cual un determinado servicio está operativo dentro de los tiempos de servicio acordados.
- **Downtime:** periodo total durante el cual un determinado servicio no está operativo dentro de los tiempos de servicio acordados.
- **Mean Time Between Service Incidents (MTBSI):** es el cociente entre el tiempo en que el servicio está disponible y el número de caídas del servicio.
- **Mean Time to Restore Services (MTRS):** es el cociente entre el tiempo en que el servicio no está disponible y el número de caídas del servicio.
- **Mean Time Between Failures (MTBF):** es el cociente entre la diferencia entre el tiempo en que el servicio está disponible y el tiempo en que el servicio no está disponible, y el número de caídas del servicio.

En la siguiente figura se muestra la evolución de un servicio entre caída y caída:

Evolución de un servicio entre dos interrupcio

(link: https://www.google.es)

5.3.- DENEGACIÓN DE SERVICIO.

En seguridad informática, un **ataque de denegación de servicio**, también llamado ataque DoS (por sus siglas en inglés), es un ataque a un sistema de ordenadores o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. Normalmente provoca la pérdida de la conectividad con la red por el consumo del ancho de banda de la red de la víctima o sobrecarga de los recursos computacionales del sistema atacado.

Los ataques DoS se generan mediante la saturación de los puertos con múltiples flujos de información, haciendo que el servidor se sobrecargue y no pueda seguir prestando su servicio. Por eso se le denomina denegación, pues hace que el servidor no pueda atender a la cantidad enorme de solicitudes. Esta técnica es usada por los crackers o hackers para dejar fuera de servicio servidores objetivo.

Una ampliación del ataque DoS es el llamado **ataque de denegación de servicio distribuido (DDoS** por sus siglas en inglés) el cual se lleva a cabo generando un gran flujo de información desde varios puntos de conexión. La forma más común de realizar un DDoS es a través de

una red de bots, siendo esta técnica el ciberataque más usual y eficaz por su sencillez tecnológica.

En ocasiones, esta herramienta ha sido utilizada como un buen método para comprobar la capacidad de tráfico que un ordenador puede soportar sin volverse inestable y afectar a los servicios que presta. Un administrador de redes puede así conocer la capacidad real de cada máquina.