DAM - Programación de servicios y procesos

Tema 5 - Generación de servicios en red

Roberto Sanz Requena
rsanz@florida-uni.es

Florida
Grup Educatiu

Índice

1. Introducción

Usuarios en red

↓

Necesidades comunes:

- Identificarse
- Compartir información
- Comunicarse
- Obtener información
- Imprimir documentos
- …

↓

Servicios “estándar” en red

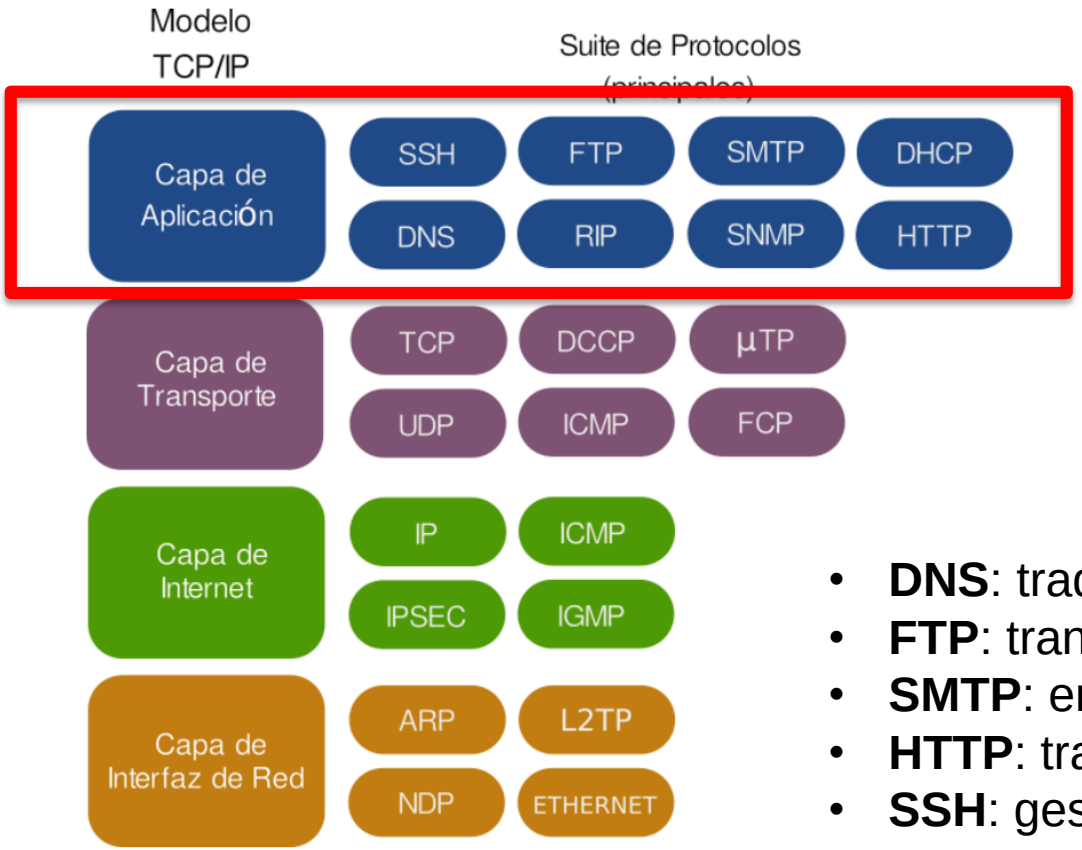
↓

¿Cómo acceder?

↓

Protocolos

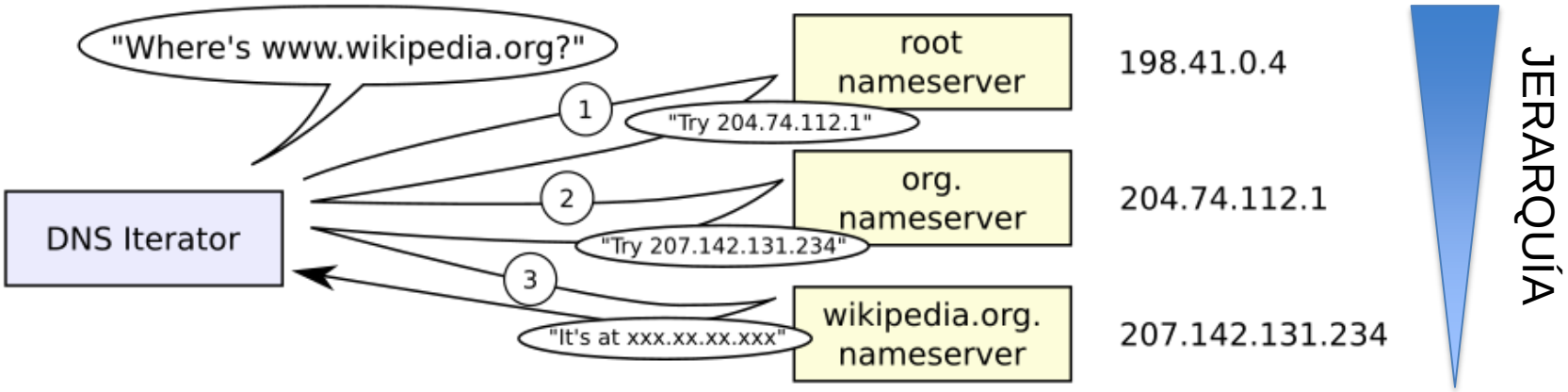
2. Protocolos estándar de comunicación en red



- **DNS**: traducir direcciones de red.
- **FTP**: transferencia de ficheros.
- **SMTP**: envío/recepción correo electrónico.
- **HTTP**: transferencia de hipertexto.
- **SSH**: gestión remota segura.

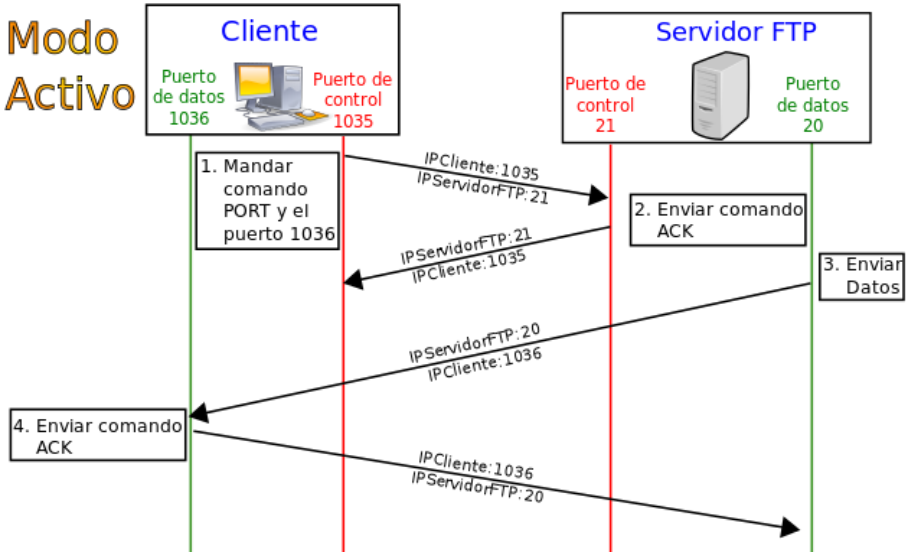
2. Protocolos estándar de comunicación en red

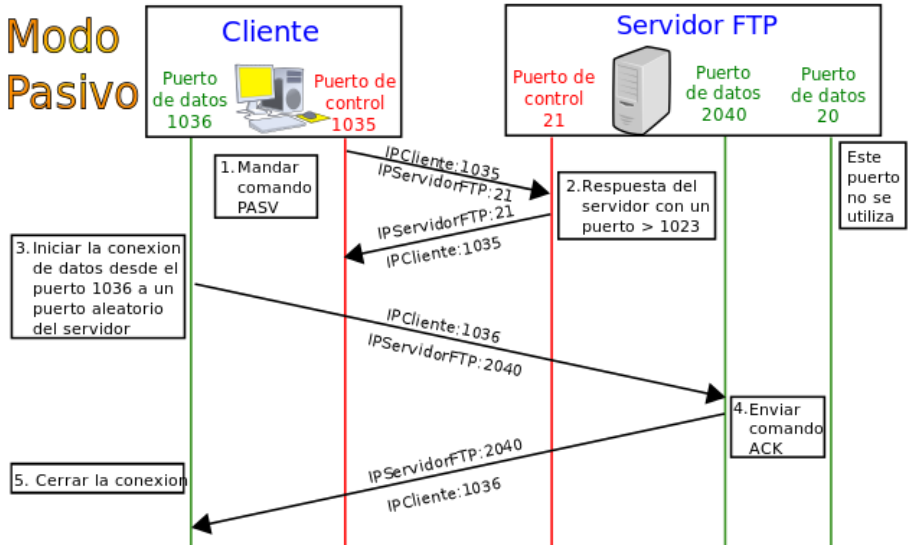
DNS: Domain Name System (sistema de nombres de dominio)



2. Protocolos estándar de comunicación en red

FTP: File Transfer Protocol (protocolo de transferencia de archivos)



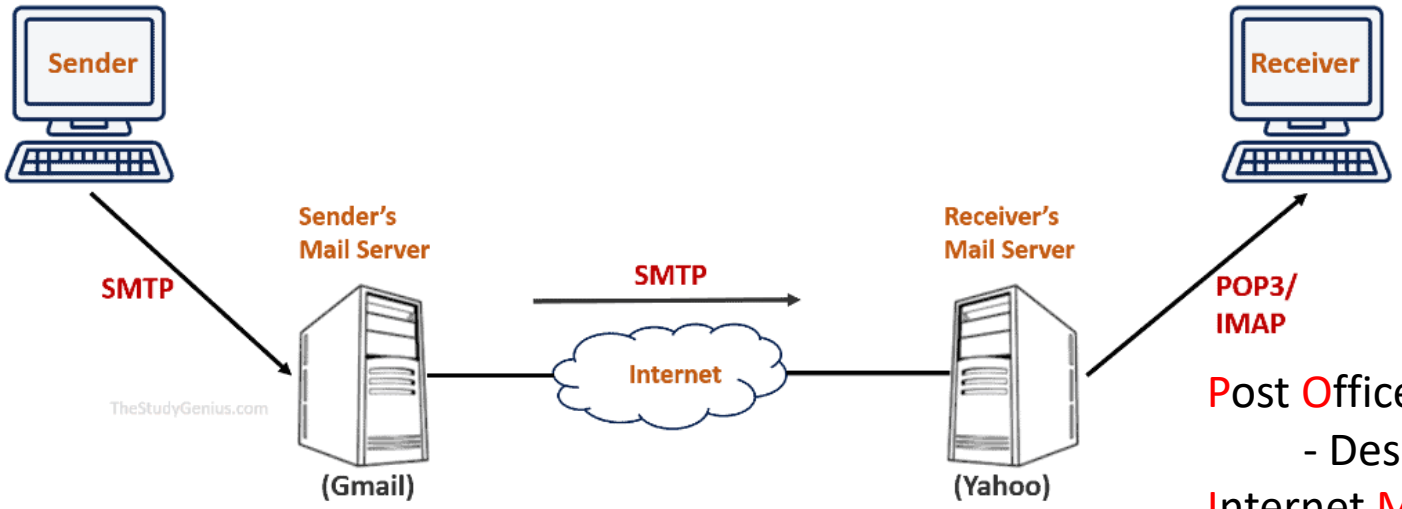


Control de permisos en el servidor (acceso/lectura/escritura por directorios).

Máxima velocidad de transferencia → Compromete la seguridad → SFTP (SSH FTP).

2. Protocolos estándar de comunicación en red

SMTP: Simple Mail Transfer Protocol (protocolo de transferencia simple de correo)



Post Office Protocol
- Descarga y gestión local

Internet Message Access Protocol
- Gestión online

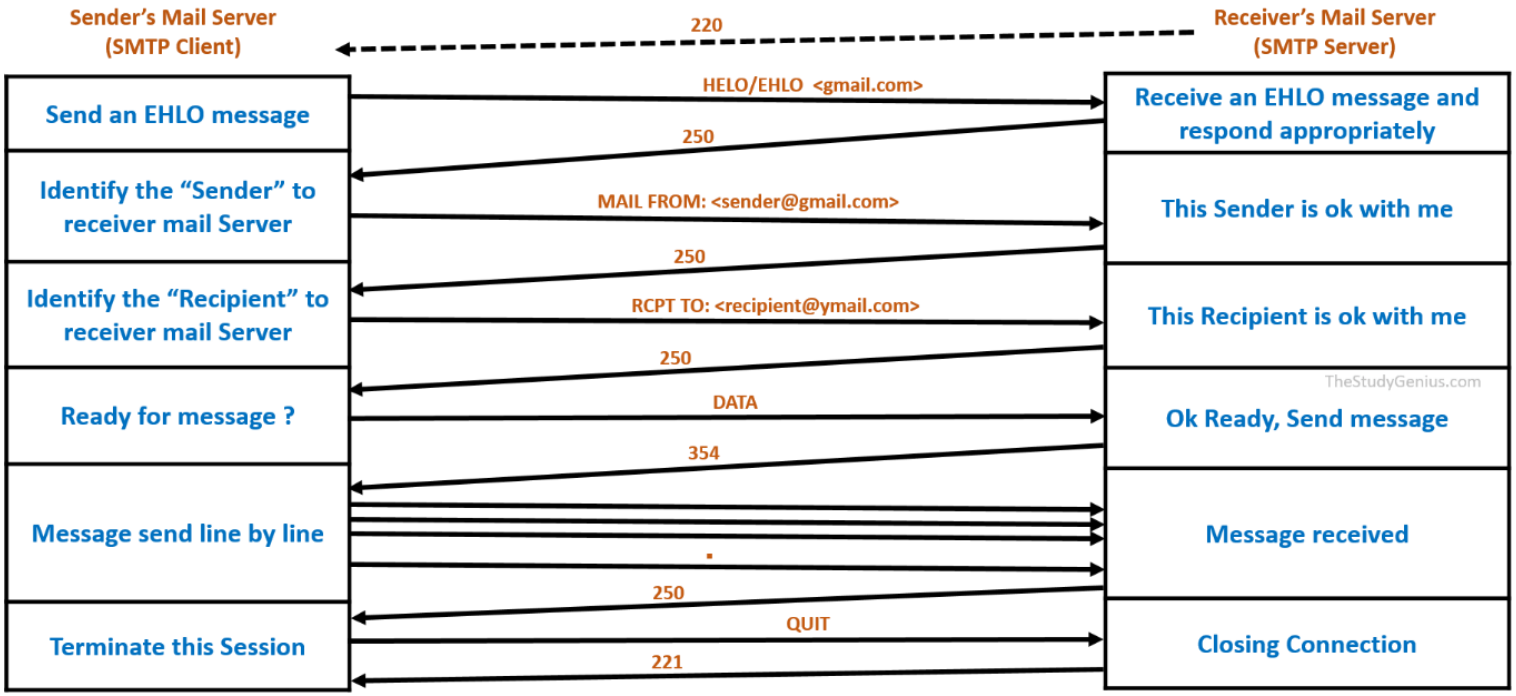
- Utiliza el puerto TCP 25 (465 o 587 para cifrado SSL o TLS)
- Difícil verificar si un remitente es legítimo o no (spam, phishing, etc.)

https://www.thestudygenius.com/simple-mail-transfer-protocol-smtp/

2. Protocolos estándar de comunicación en red

SMTP: Simple Mail Transfer Protocol (protocolo de transferencia simple de correo)

Ejemplos de envío de mensajes (sockets).

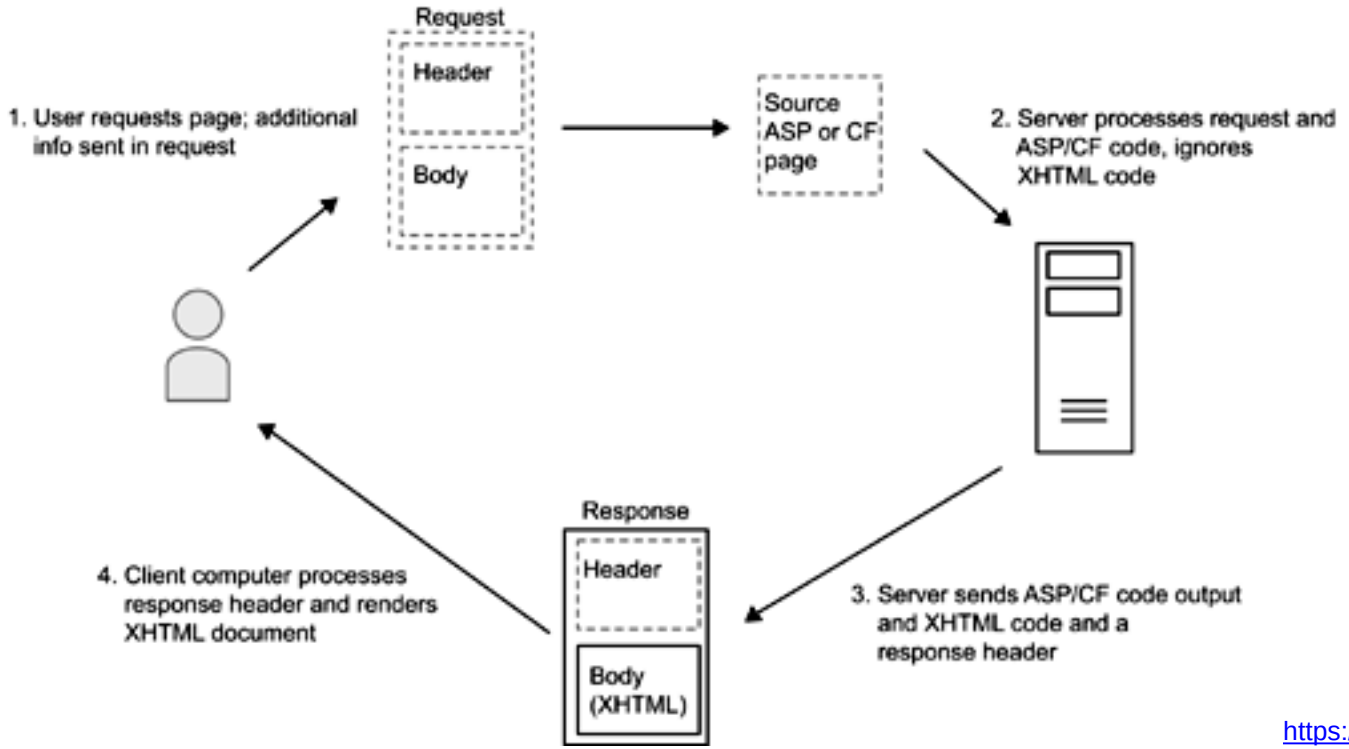


https://www.thestudygenius.com/simple-mail-transfer-protocol-smtp/

2. Protocolos estándar de comunicación en red

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

Transferencia de información a través de archivos (HTML) en Internet.



https://tinyurl.com/8atezk5n

2. Protocolos estándar de comunicación en red

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

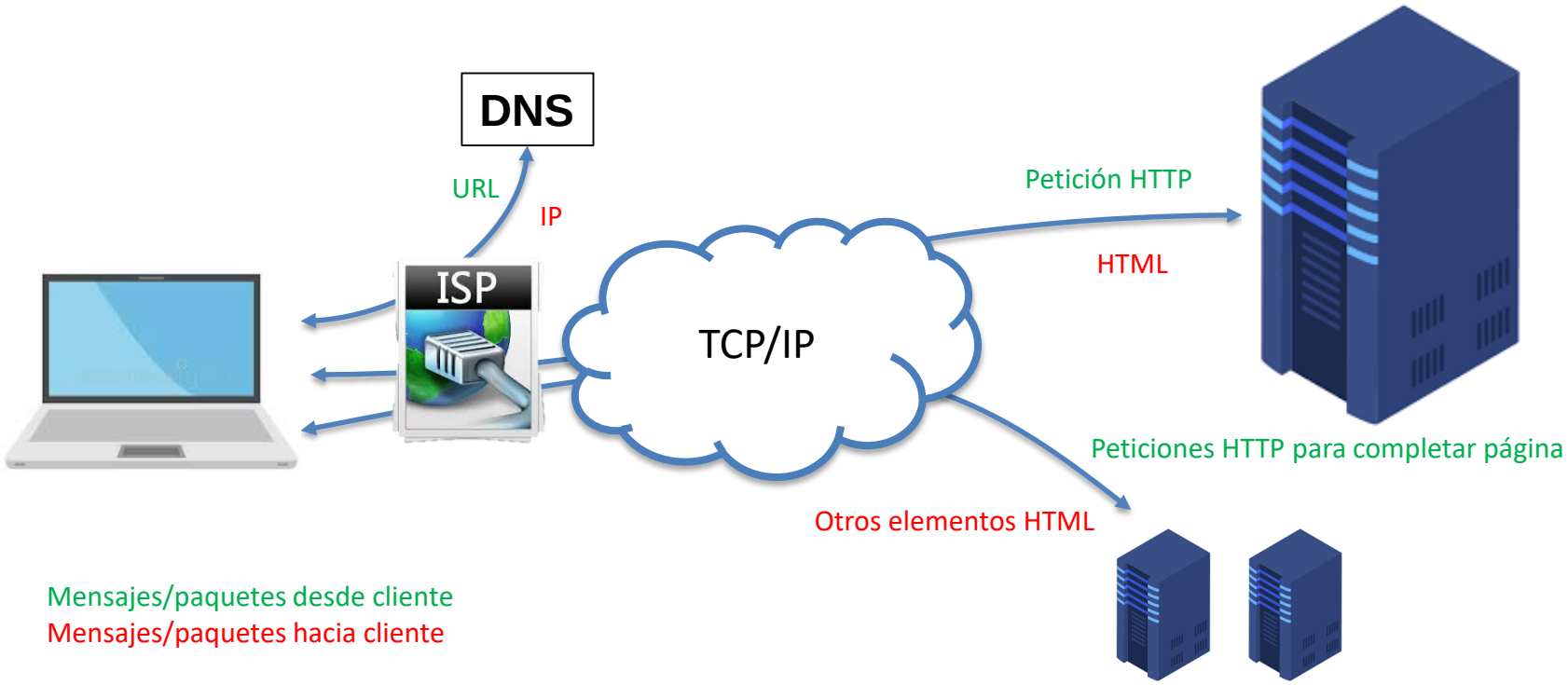
Cuando se introduce una dirección en el navegador…

1. Ordenador conectado por módem a un ISP (Telefónica, Vodafone,…).
2. Escribimos una URL (dirección) en el navegador.
3. El ISP recibe la URL y la traduce a una dirección IP (protocolo DNS).
4. El navegador solicita una petición HTTP al servidor que “escucha” en la dirección IP para que le envíe el recurso (por ejemplo, una página web).
5. El servidor acepta (mensaje “200 OK”) la petición y envía el recurso como paquetes de datos (protocolo TCP). Puede dar otros mensajes.
6. El navegador recibe la página y la recorre en busca de elementos que necesite para completarla (por ejemplo, imágenes).
7. El navegador realiza nuevas peticiones al servidor (al mismo o a otros) para obtener cada elemento de la página.
8. El navegador muestra la página completa.

2. Protocolos estándar de comunicación en red

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

Cuando se introduce una dirección en el navegador…



2. Protocolos estándar de comunicación en red

HTTP: Hypertext Transfer Protocol (protocolo de transferencia de hipertexto)

Protocolo orientado a transacciones con esquema petición-respuesta entre cliente (*user agent*, navegadores web) y servidor. El cliente solicita un recurso identificado unívocamente como URL (*uniform resource locator*).

Tipos de peticiones más utilizadas:

- GET: solicita una representación del recurso especificado.
- POST: envía datos en la URI de la petición para que sean procesados por el recurso identificado del servidor (creación de un nuevo recurso).
- PUT: actualización de un recurso.
- DELETE: borrador de un recurso.

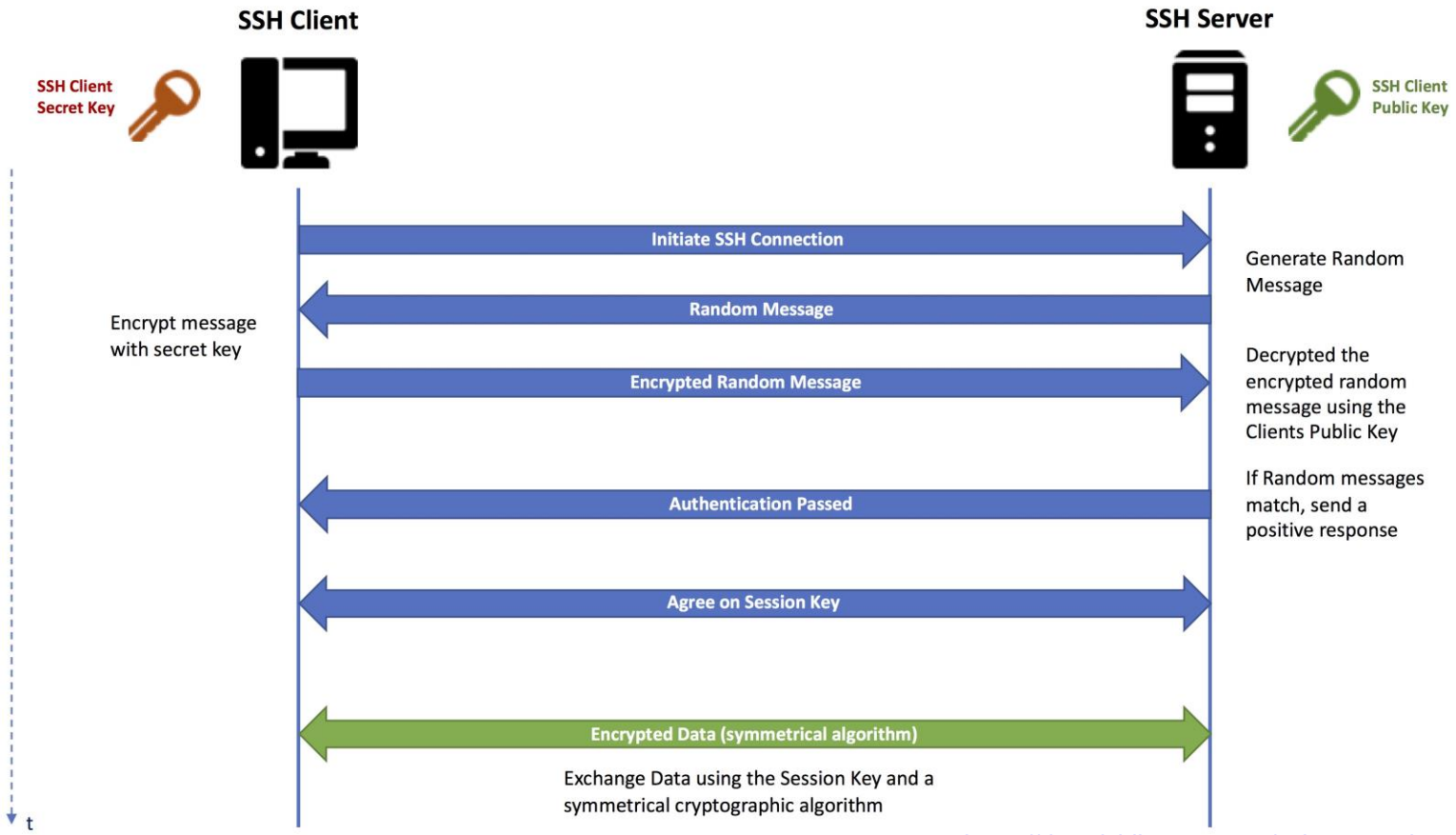
2. Protocolos estándar de comunicación en red

SSH: Secure SHell (intérprete de órdenes seguro)

- Acceso seguro (encriptado) a máquinas remotas para manejarlas por completo a través de línea de comandos o interfaz gráfica (servidor X).
- Copia de datos de forma segura.
- Evolución del protocolo Telnet (texto plano).
- Se basa en el uso de claves
  - Criptografía asimétrica para autenticación y establecimiento de la conexión (claves pública y privada).
  - Criptografía simétrica (clave acordada) para encriptar/desencriptar los datos que se transfieren a través de la conexión (túnel SSH).

2. Protocolos estándar de comunicación en red

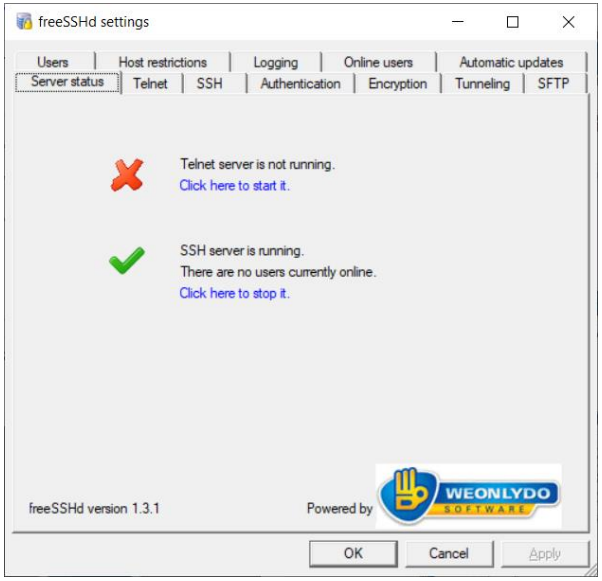
SSH: Secure SHell (intérprete de órdenes seguro)



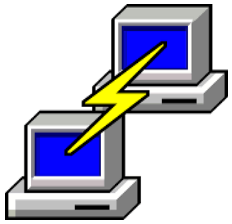
https://dev.vividbreeze.com/ssh-protocol-and-key-generation/

2. Protocolos estándar de comunicación en red

SSH: Secure SHell (intérprete de órdenes seguro)



http://www.freesshd.com/



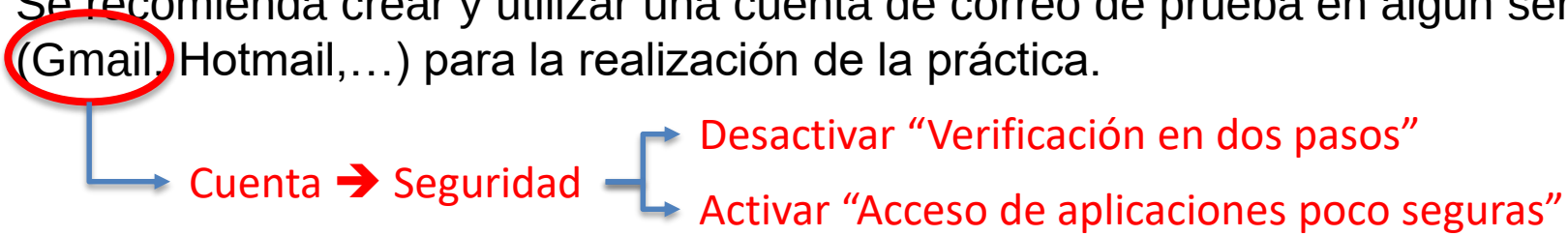
https://www.putty.org/

3. Programación de clientes y servidores

Cliente de correo

- Se desea crear un programa Java que permita enviar un correo con un fichero adjunto (nos proporcionan la ruta).
- El cuerpo del mensaje debe incluir un texto de saludo y el nombre del archivo.
- El programa debe ser capaz de enviar el correo a varios destinatarios e incluir varios adjuntos a la vez.

Se recomienda crear y utilizar una cuenta de correo de prueba en algún servidor (Gmail, Hotmail,…) para la realización de la práctica.



3. Programación de clientes y servidores

Cliente de correo

Biblioteca: `JavaMail`.

Convertir proyecto Java a proyecto Maven:

- Clic derecho ➔ Configure ➔ Convert to Maven Project ➔ Finish
- Añadir en fichero pom.xml, después de </build>:

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/javax.mail/javax.mail-api -->
    <dependency>
        <groupId>com.sun.mail</groupId>
        <artifactId>javax.mail</artifactId>
        <version>1.6.0</version>
    </dependency>
</dependencies>
```

3. Programación de clientes y servidores

Cliente de correo - Operativa

```
Properties props = System.getProperties();
```
➔ Añadir las propiedades necesarias para la conexión al servidor: servidor de correo, puerto, e-mail remitente, password remitente, tipo de autenticación y tipo de cifrado.

```
Session session = Session.getDefaultInstance(props);
```
➔ Crear una sesión.

```
MimeMessage message = new MimeMessage(session);
```
➔ Crear mensaje y completarlo con los campos mínimos (remitente, destinatario/s y asunto).
➔ Añadir las partes que se quieran (`BodyPart`) para el texto del mensaje y anexos.
➔ Agrupar las `BodyPart` en un objeto `Multipart`.
➔ Añadir el objeto `Multipart` al mensaje.

```
Transport transport = session.getTransport("smtp");
```
➔ Envío del mensaje mediante un objeto de tipo `Transport` sobre la sesión creada.

```java
public static void envioMail(String mensaje, String asunto, String email_remitente, String email_remitente_pass,
String host_email, String port_email, String[] email_destino, String[] anexo) throws
UnsupportedEncodingException, MessagingException {

    Properties props = System.getProperties();
    props.put("mail.smtp.host", host_email);
    props.put("mail.smtp.user", email_remitente);
    props.put("mail.smtp.clave", email_remitente_pass);
    props.put("mail.smtp.auth", "true");
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.port", port_email);

    Session session = Session.getDefaultInstance(props);

    MimeMessage message = new MimeMessage(session);
    message.setFrom(new InternetAddress(email_remitente));
    message.addRecipients(Message.RecipientType.TO, email_destino[0]);
    message.setSubject(asunto);

    BodyPart messageBodyPart1 = new MimeBodyPart();
    messageBodyPart1.setText(mensaje);

    BodyPart messageBodyPart2 = new MimeBodyPart();
    DataSource src= new FileDataSource(anexo[0]);
    messageBodyPart2.setDataHandler(new DataHandler(src));
    messageBodyPart2.setFileName(anexo[0]);

    Multipart multipart = new MimeMultipart();
    multipart.addBodyPart(messageBodyPart1);
    multipart.addBodyPart(messageBodyPart2);

    message.setContent(multipart);

    Transport transport = session.getTransport("smtp");
    transport.connect(host_email, email_remitente, email_remitente_pass);
    transport.sendMessage(message, message.getAllRecipients());
    transport.close();

}
```

TLS ➔ puerto 587

➔ CREAR CLASE Y AÑADIR MAIN

3. Programación de clientes y servidores

Servidor

- Programa que ofrece servicios a otros programas (clientes).
- Programación cliente/servidor: establecer un **protocolo** de comunicaciones.
- El cliente y el servidor pueden estar en sitios distintos y comunicarse a través de protocolos de red.
- El cliente y el servidor pueden estar programados en **distintos lenguajes**.
- Ejemplo: servidor HTTP simple para procesar operaciones GET y POST.

3. Programación de clientes y servidores

Servidor HTTP

- Ejemplo de operación GET:
  - Introducir la URL en el navegador: http://localhost:5000/test?name=amigo
  - Devuelve una página con “Hola amigo”

- Ejemplo de operación POST:
  - Utilizar un cliente tipo Postman, seleccionar el método POST e introducir la URL: http://localhost:5000/test
  - En el cuerpo (Body) introducir cualquier texto (se podría introducir un objeto en formato JSON, por ejemplo, para almacenarlo en MongoDB)

https://www.postman.com/downloads/

3. Programación de clientes y servidores

Servidor HTTP - Clase HTTPServer

Objeto InetSocketAddress

Conexiones en cola

```
HttpServer servidor = HttpServer.create(direccionTCPIP, backlog);
```

```
servidor.createContext(rutaRespuesta, gestorHTTP);
```

Ruta en la URL a partir de la cual el servidor dará respuesta (p.ej. /test)

Clase que gestionará las peticiones GET, POST, etc.

Ejecutor multihilo

```
ThreadPoolExecutor threadPoolExecutor =
    (ThreadPoolExecutor) Executors.newFixedThreadPool(10);
servidor.setExecutor(threadPoolExecutor);
servidor.start();
```

3. Programación de clientes y servidores

```
public class ServidorHTTP {

    public static void main(String[] args) throws Exception {

    String host = "localhost"; //127.0.0.1
    int puerto = 5000;
    InetSocketAddress direccionTCPIP = new InetSocketAddress(host,puerto);
    int backlog = 0;
    HttpServer servidor = HttpServer.create(direccionTCPIP, backlog);

    GestorHTTP gestorHTTP = new GestorHTTP();
    String rutaRespuesta = "/test";
    servidor.createContext(rutaRespuesta, gestorHTTP);

    //Opcion 1 de ejecucion: no multihilo
//    servidor.setExecutor(null);

    //Opcion 2 de ejecucion: multihilo con ThreadPoolExecutor
    ThreadPoolExecutor threadPoolExecutor = (ThreadPoolExecutor)Executors.newFixedThreadPool(10);
    servidor.setExecutor(threadPoolExecutor);

    servidor.start();
    System.out.println("Servidor HTTP arranca en el puerto " + puerto);

    }
}
```

3. Programación de clientes y servidores

Servidor HTTP - Clase GestorHTTP

```
public class GestorHTTP implements HttpHandler {

     @Override
     public void handle(HttpExchange httpExchange) throws IOException {

          String requestParamValue=null;
          if("GET".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handleGetRequest(httpExchange);
               handleGETResponse(httpExchange,requestParamValue);
          } else if ("POST".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handlePostRequest(httpExchange);
               handlePOSTResponse(httpExchange,requestParamValue);
          }

     }
}
```

3. Programación de clientes y servidores

Servidor HTTP - Clase GestorHTTP

```
private String handleGetRequest(HttpExchange httpExchange) {

    return httpExchange.getRequestURI().toString().split("\\?")[1].split("=")[1];

}
```

```
            String requestParamValue=null;
            if("GET".equals(httpExchange.getRequestMethod())) {
                requestParamValue = handleGetRequest(httpExchange);
                handleGETResponse(httpExchange,requestParamValue);
            } else if ("POST".equals(httpExchange.getRequestMethod())) {
                requestParamValue = handlePostRequest(httpExchange);
                handlePOSTResponse(httpExchange,requestParamValue);
            }

        }
    }
```

3. Programación de clientes y servidores

```
private void handleGETResponse(HttpExchange httpExchange, String requestParamValue) {
        OutputStream outputStream = httpExchange.getResponseBody();
        String htmlResponse = "<html><body>Hola"+requestParamValue+</body></html>";
        httpExchange.sendResponseHeaders(200, htmlResponse.length());
        outputStream.write(htmlResponse.getBytes());
        outputStream.flush();
        outputStream.close();
}
```

```
            String requestParamValue=null;
            if("GET".equals(httpExchange.getRequestMethod())) {
                requestParamValue = handleGetRequest(httpExchange);
                handleGETResponse(httpExchange,requestParamValue);
            } else if ("POST".equals(httpExchange.getRequestMethod())) {
                requestParamValue = handlePostRequest(httpExchange);
                handlePOSTResponse(httpExchange,requestParamValue);
            }

        }
    }
```

3. Programación de clientes y servidores

```
private String handlePostRequest(HttpExchange httpExchange) {
     InputStream inputStream = httpExchange.getRequestBody();
     //Procesar lo que hay en inputStream, por ejemplo linea a linea y guardarlo todo
     en un string, que sera el que devuelve el metodo
     String postRequest;
     ...
     return postRequest;
}
```

```
          String requestParamValue=null;
          if("GET".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handleGetRequest(httpExchange);
               handleGETResponse(httpExchange,requestParamValue);
          } else if ("POST".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handlePostRequest(httpExchange);
               handlePOSTResponse(httpExchange,requestParamValue);
          }

     }
}
```

3. Programación de clientes y servidores

```
private void handlePOSTResponse(HttpExchange httpExchange, String requestParamValue) {
     OutputStream outputStream = httpExchange.getResponseBody();
     String htmlResponse = “Respuesta a la petición POST”;
     httpExchange.sendResponseHeaders(200, htmlResponse.length());
     outputStream.write(htmlResponse.getBytes());
     outputStream.flush();
     outputStream.close();
}
```

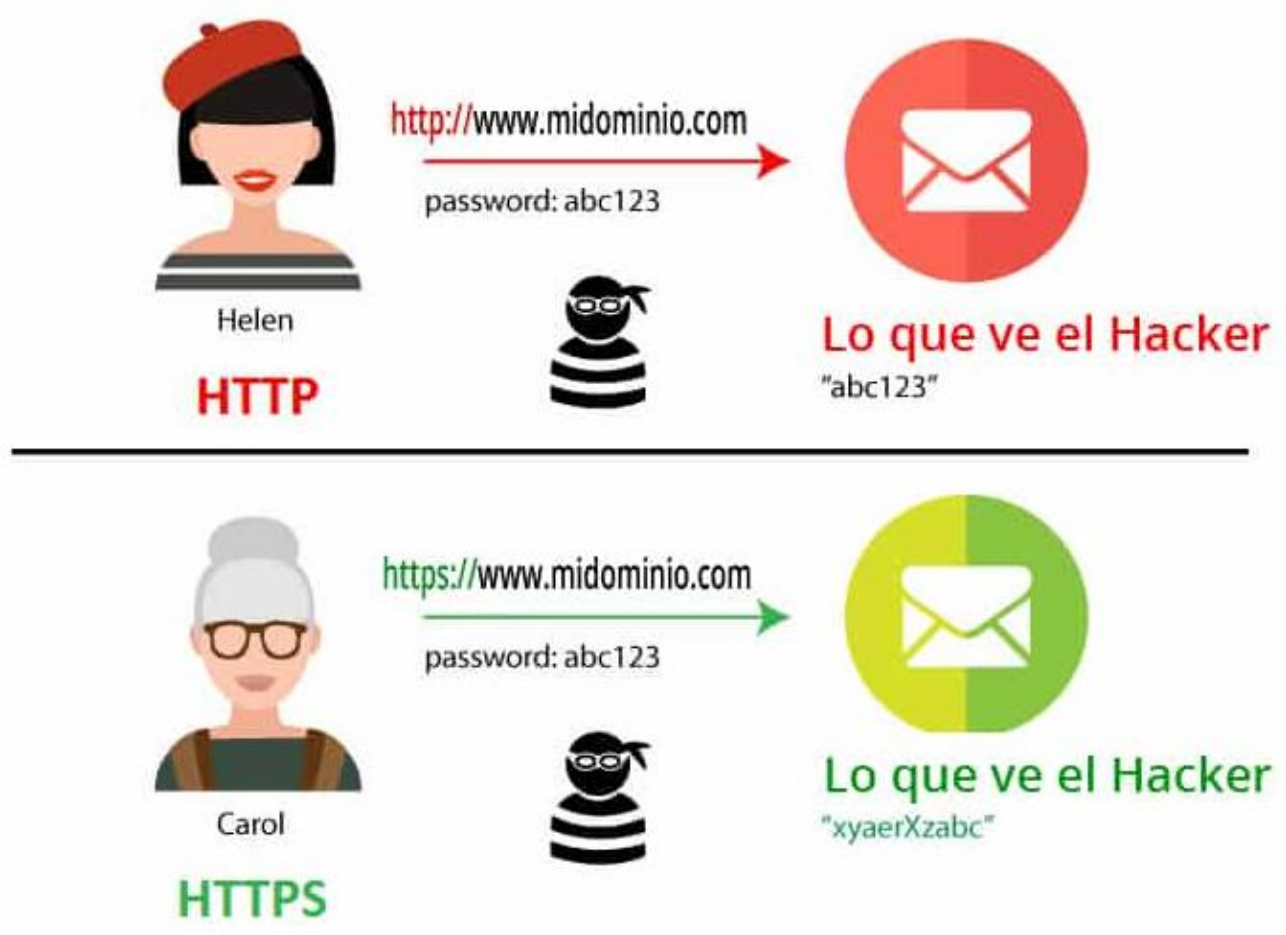
```
          String requestParamValue=null;
          if("GET".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handleGetRequest(httpExchange);
               handleGETResponse(httpExchange,requestParamValue);
          } else if ("POST".equals(httpExchange.getRequestMethod())) {
               requestParamValue = handlePostRequest(httpExchange);
               handlePOSTResponse(httpExchange,requestParamValue);
          }

     }
}
```

3. Programación de clientes y servidores

¿Y HTTPS?

Tema 6



https://www.webebre.net/por-que-pasar-de-http-a-https/

Actividad Entregable 5

Presentación de la Actividad Entregable 5 (AE5_T5_ServiciosRed)