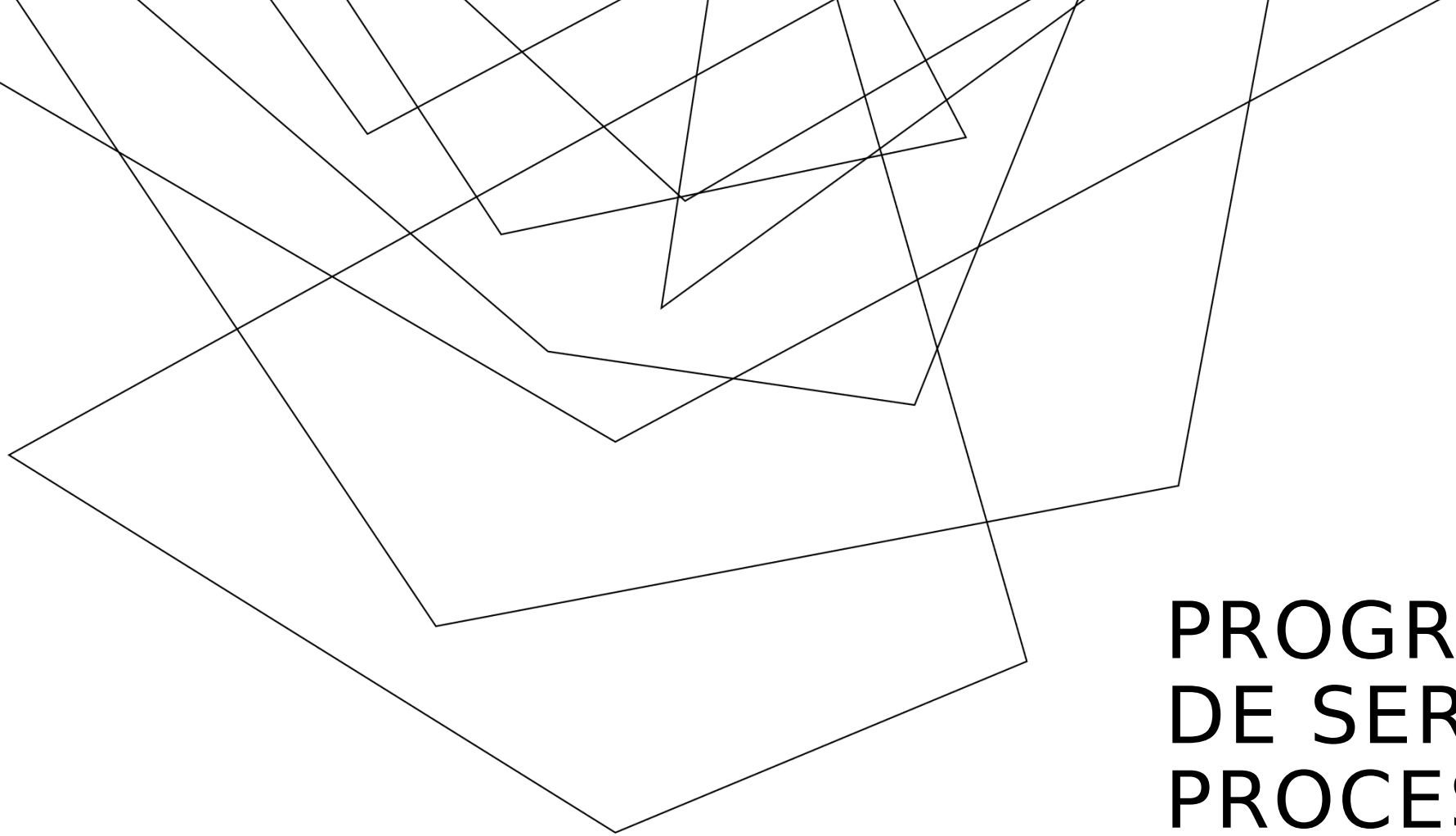


PROGRAMACIÓN DE SERVICIOS Y PROCESOS 2º DAM

Andrés Chillón Sesma

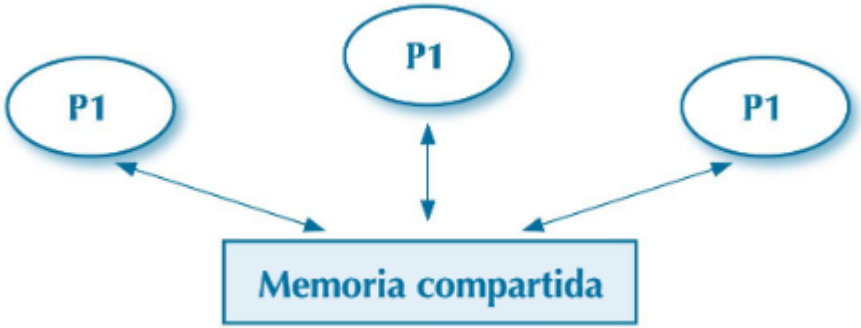
andres.chillon@educa.madrid.org

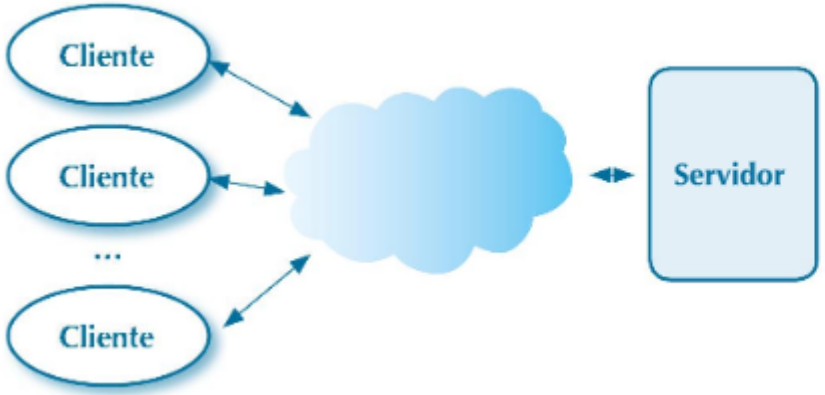
Programación de comunicaciones en red

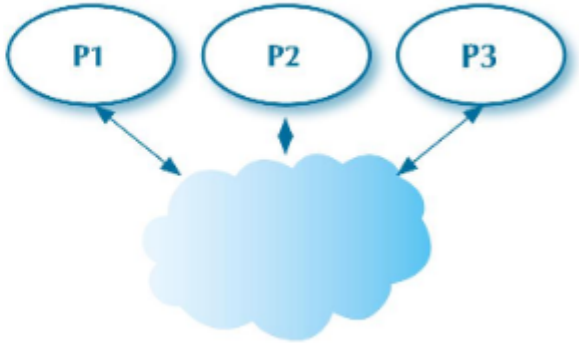
CONTENIDOS

- Protocolos de comunicaciones (IP-Internet Protocol, TCP-Transmission Control Protocol, UDP-User Datagram Protocol).
- Comunicación entre aplicaciones. Modelos cliente/servidor, p2p (peer-to-peer) e híbridos.
- Sockets: Conceptos, tipos, características, utilización.
- Elementos de programación de aplicaciones en red. Librerías y clases. APIs (Application Programming Interface) de sockets.
- Programación de aplicaciones cliente y servidor

Modelos de comunicación entre procesos

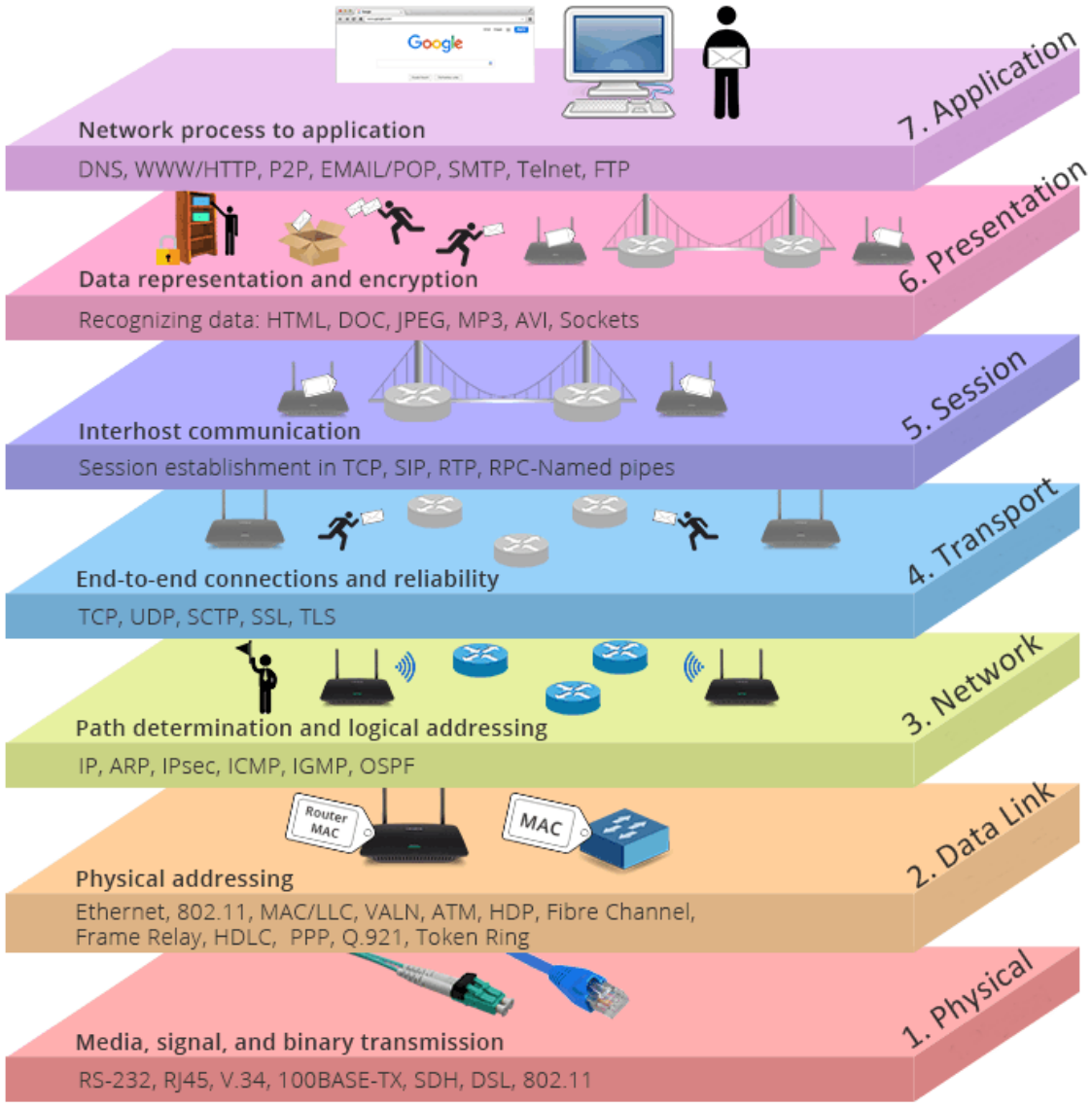




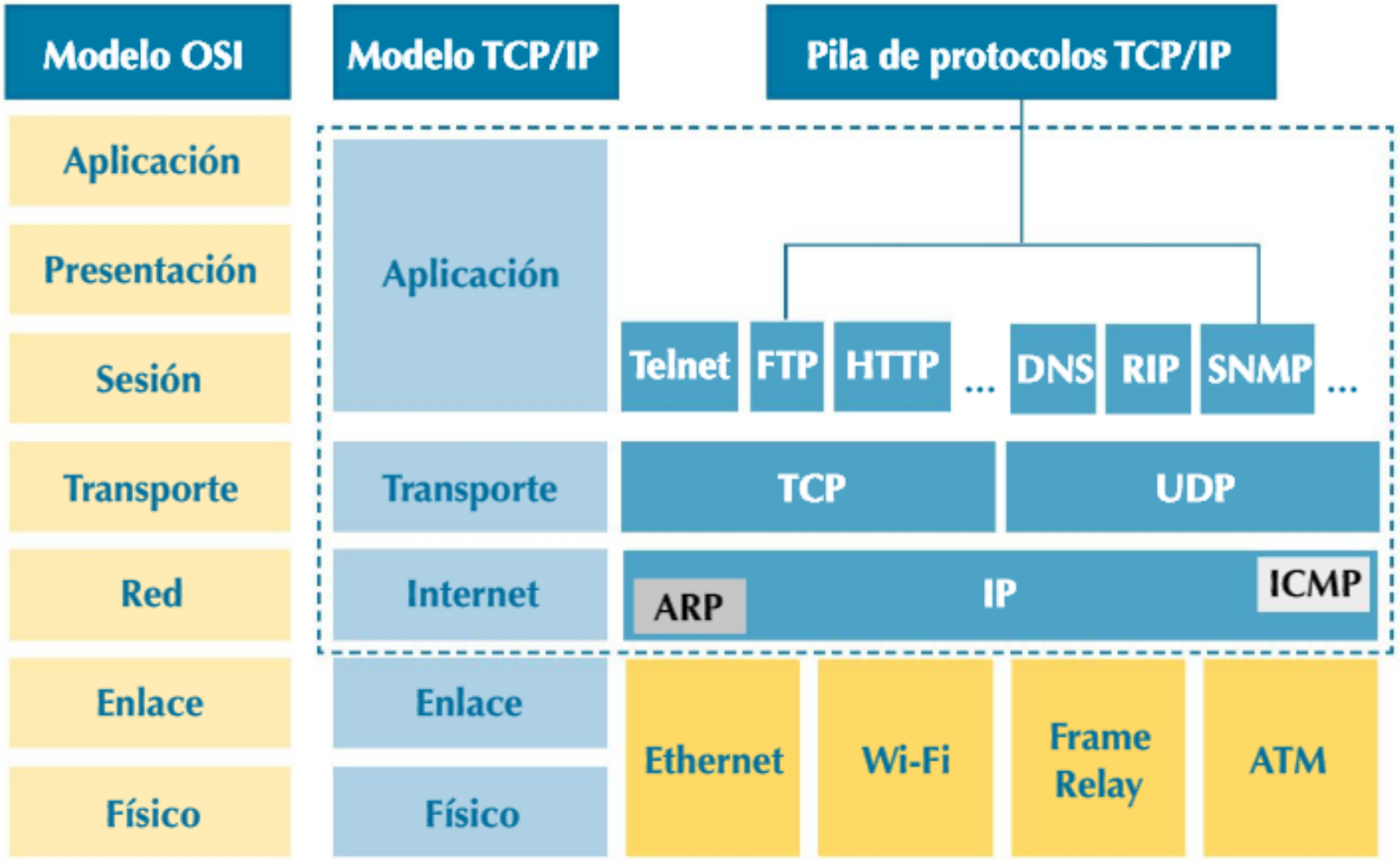


Modelo OSI

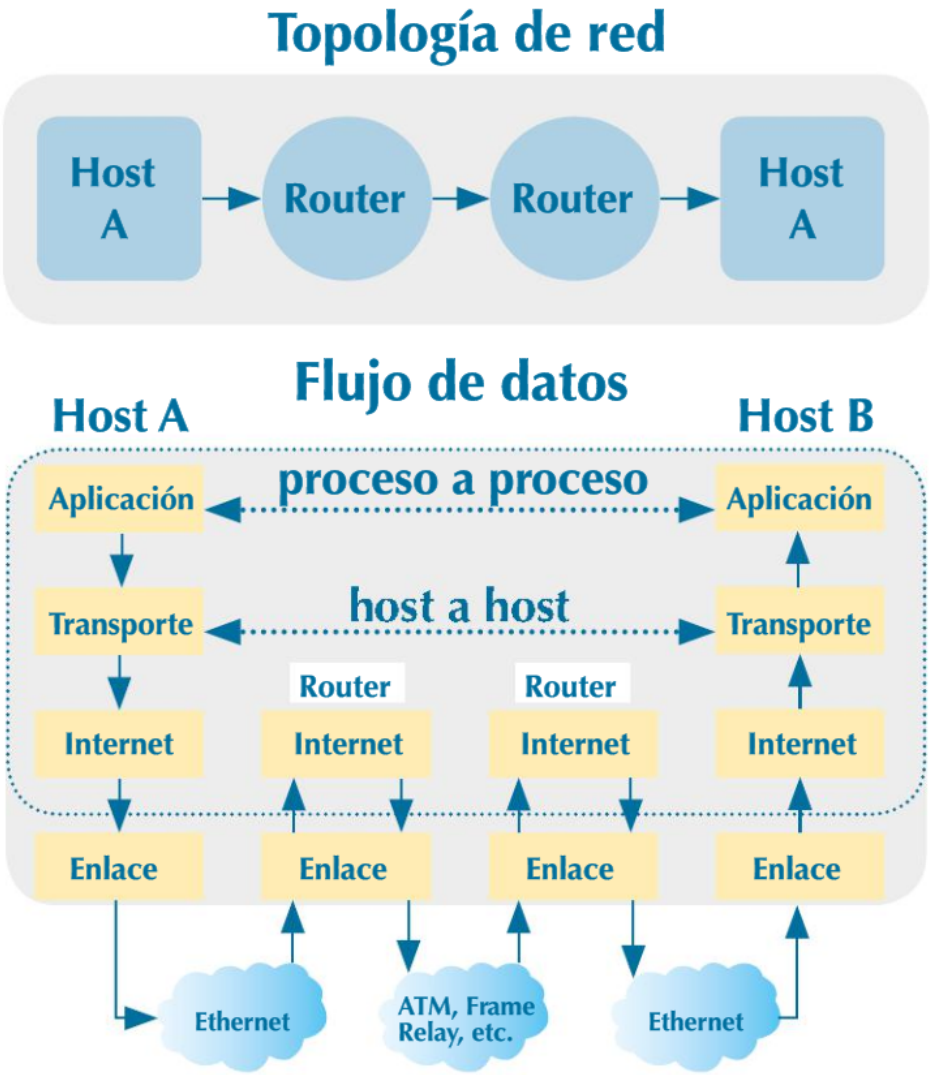
Nº de capa	Nombre de capa	Descripción
7	Aplicación	Se compone de los servicios y aplicaciones de comunicación estándar que puede utilizar todo el mundo.
6	Presentación	Se asegura de que la información se transfiera al sistema receptor de un modo comprensible para el sistema.
5	Sesión	Administra las conexiones y terminaciones entre los sistemas que cooperan.
4	Transporte	Administra la transferencia de datos. Asimismo, garantiza que los datos recibidos sean idénticos a los transmitidos.
3	Red	Administra las direcciones de datos y la transferencia entre redes.
2	Vínculo de datos	Administra la transferencia de datos en el medio de red.
1	Física	Define las características del hardware de red.



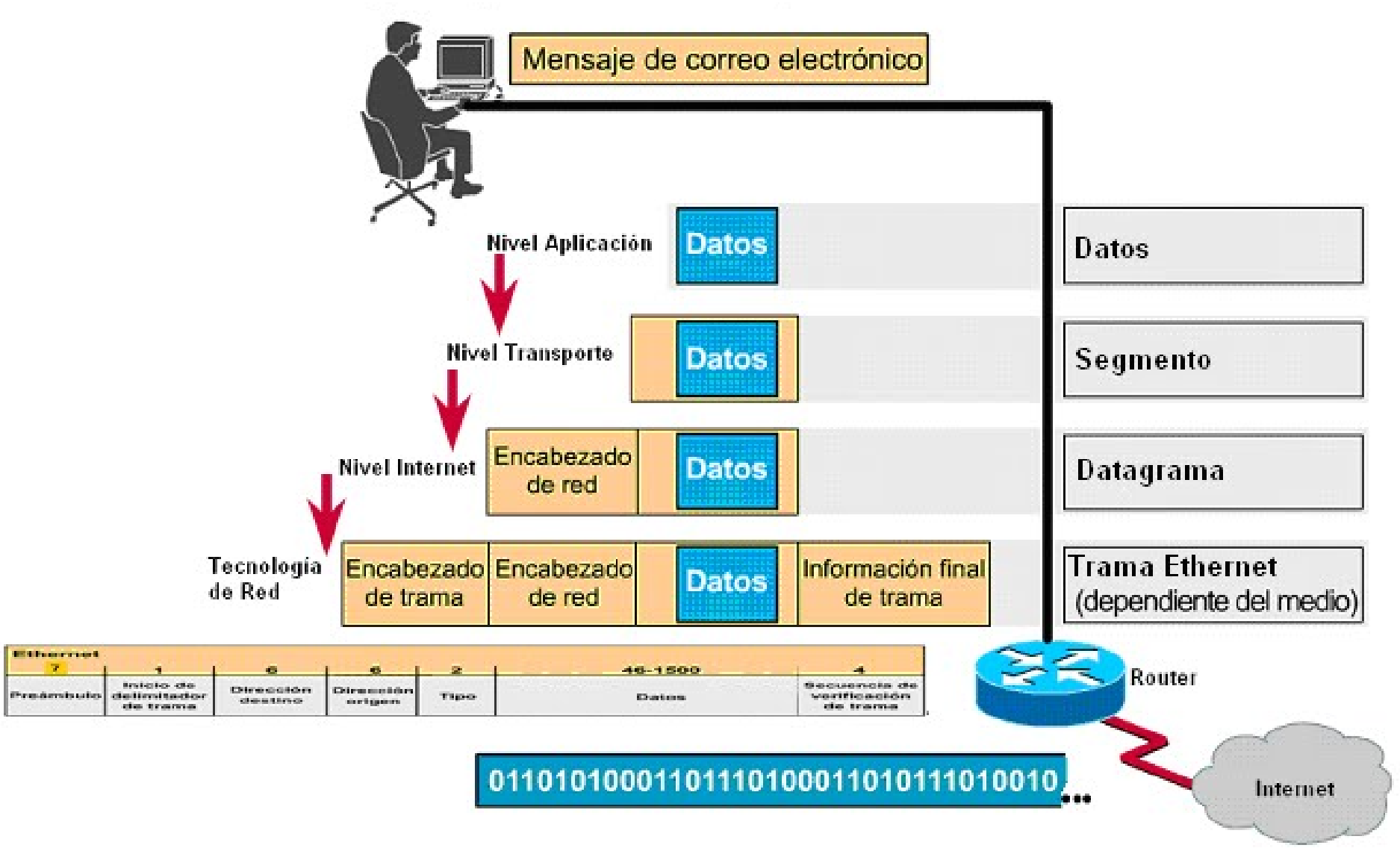
Modelo de niveles TCP/IP



Modelo de niveles TCP/IP



Ejemplo de encapsulamiento de datos



Clases comunicaciones en red

Interfaces de red	`NetworkInterface:` interfaz de red.
Direcciones IP	`InetAddress:` dirección IP. `Inet4Address:` dirección IPv4. `Inet6Address:` dirección IPv6. `InterfaceAddress:` dirección IP más información relativa a la red a la que pertenece la interfaz.
Sockets	`Socket:` *socket* TCP de cliente, para conectar con un proceso servidor mediante TCP. `ServerSocket:` *socket* TCP de servidor, que acepta conexiones desde un *socket* de cliente mediante TCP. `DatagramSocket:` *socket* de UDP, que se usa para enviar y recibir datagramas de UDP.

Clases comunicaciones en red

Métodos de la clase NetworkInterface

Método	Funcionalidad
`static Enumeration<NetworkInterface> getNetworkInterfaces()`	Devuelve todas las interfaces de red existentes en la máquina. A partir del `Enumeration` se puede obtener un `Iterator` para iterar sobre ellas.
`String getDisplayName()` `String getName()`	Devuelven el nombre de la interfaz.
`byte[] getHardwareAddress()`	Devuelve la dirección MAC en forma de array de 6 bytes (48 bits).
`Enumeration<InetAddress> getInetAddresses()` `List<InterfaceAddress> getInterfaceAddresses()`	Devuelven las direcciones IP asociadas a la interfaz. En general, será al menos una dirección IPv4 y una IPv6. Un objeto `InterfaceAddress` proporciona más información que `InetAddress`. Por ejemplo, la longitud de la máscara de red y, para IPv4, la dirección de *broadcast* parara la red a la que pertenece la interfaz.
`boolean isLoopback()`	Determina si una interfaz es de tipo *loopback*.

Clases comunicaciones en red

Métodos de la clase InterfaceAddress

Método	Funcionalidad
`InetAddress getAddress()`	Devuelve la dirección IP.
`short getNetworkPrefixLength()`	Devuelve la longitud de la máscara de red, es decir, el número de bits dentro de la dirección IP que especifican la red.
`InetAddress getBroadcast()`	Devuelve la dirección de *broadcast* para la red a la que pertenece la dirección IP, pero solo en caso de que se trate de una dirección de IPv4. Si se trata de una dirección de IPv6, devuelve `null`.

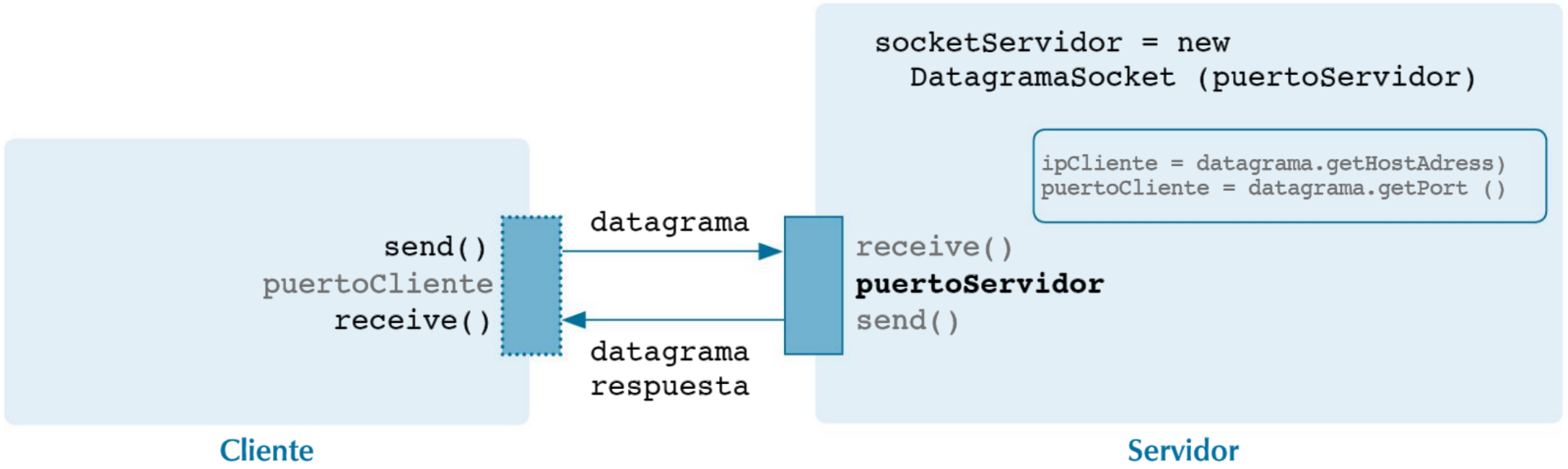
Clases comunicaciones en red

Métodos de la clase InetAddress

Método	Funcionalidad
`static InetAddress getByName(String host)`	Obtiene una dirección IP a partir de un nombre de host, utilizando servicios de resolución de nombres, o bien a partir de una representación textual de una dirección IP.
`static InetAddress[] getAllByName(String host)`	Similar al método anterior, pero más completo, porque devuelve un array con todas las direcciones IP que se pueden obtener utilizando servicios de resolución de nombres. También se puede especificar el host mediante la representación textual de una dirección IPv4 o IPv6.
`public String getHostName()`	Este método permite realizar una resolución inversa de nombre, es decir, obtener un nombre de host a partir de su dirección IP. Si la `InetAddress` se creó sin especificar un nombre para el host, entonces se realiza una resolución inversa de nombres para obtener el nombre del host a partir de su dirección IP. También se puede crear una `InetAddress` indicando una dirección IP y un nombre de host, con `static InetAddress getByAddress(String host, byte[] addr)`. En ese caso, este método devolvería el nombre proporcionado al crear la `InetAddress`, sin realizar resolución inversa de nombres.
`String getCanonicalHostName()`	Devuelve el nombre de host canónico o FQDN (*fully qualified domain name*), utilizando el protocolo DNS para obtener esta información.
`static InetAddress getLocalHost()`	Devuelve la dirección IP del propio host. En Linux, normalmente se obtiene su nombre de `/etc/hostname`, y se resuelve el nombre para obtener una dirección IP. Dependiendo del sistema, se podría obtener la dirección de la interfaz de *loopback*, una dirección IP privada perteneciente a la red local, o incluso una IP externa o pública. Si no se puede resolver este nombre, se lanza una excepción de tipo `UnknownHostException`.

Método	Funcionalidad
`static InetAddress getByName(String host)`	Obtiene una `InetAddress` para el host. En host se puede proporcionar una dirección IP o un nombre de host. En el primer caso, se utiliza el mecanismo de resolución de nombres para obtener una dirección IP. En el segundo, se analiza y valida la dirección IP y se obtiene un objeto de clase `Inet4Address` o `Inet6Address`.
`String getHostAddress()`	Devuelve la representación textual de la dirección IP.
`byte[] getAddress()`	Devuelve un array con los bytes que componen la dirección IP: 4 en el caso de una dirección IPv4, o 16 en el caso de una dirección IPv6.
`static InetAddress getByAddress(byte[] addr)`	Devuelve una `InetAddress` para la dirección IP dada, especificada como un array de bytes.
`boolean isReachable( int timeout)`	Determina si la dirección es alcanzable. Se especifica un tiempo máximo de espera en milisegundos `timeout`.
`boolean isSiteLocalAddress()`	Determina si la dirección es local o privada. Los rangos de direcciones para redes privadas se especifican en la RFC 1918 (https://tools.ietf.org/html/rfc1918). Como ya se ha explicado, las redes privadas de clase C para IPv4, por otra parte las más frecuentes, son 192.168.X.0/24, con 0 <= X <= 255.
`boolean isLoopbackAddress()`	Determina si la dirección IP es de una interfaz de tipo *loopback*.
`static InetAddress getLoopbackAddress()`	Devuelve la dirección de *loopback*. Esta es, como ya se ha visto: – 127.0.0.1 para IPv4. En realidad, según los estándares de internet, es cualquiera de la forma 127/8, es decir, cuyos ocho primeros bits formen el valor 127, es decir, de la forma 127.*.*.*. – ::1 para IPv6.

Socket UDP


socketServidor = new
DatagramaSocket (puertoServidor)
ipCliente = datagrama.getHostAdress)
puertoCliente = datagrama.getPort ()
send()
puertoCliente
receive()
datagrama
datagrama
respuesta
receive()
puertoServidor
send()
Cliente
Servidor

Clases comunicaciones en red

Métodos de la clase DatagramPacket

Método	Funcionalidad

`DatagramSocket()`
`DatagramSocket(int port)`
`DatagramSocket(int port, InetAddress laddr)`
	Diversos constructores para un `DatagramSocket` para envío y recepción de datagramas de UDP.
	El puerto y la dirección IP especificados son locales.
	La dirección IP y el puerto de destino para datagramas enviados van especificados en el propio datagrama.
	Si se especifica la dirección IP de una interfaz activa, el *socket* se liga a ella. Si no, se liga a todas las interfaces activas (wildcard address). Se puede especificar un puerto para enviar y recibir datagramas por él. Un proceso servidor normalmente hará esto. Un programa cliente normalmente no especificará un puerto, y entonces el sistema elegirá un puerto efímero.

`void receive(DatagramPacket p)`	Recibe un datagrama desde la interfaz (especificada por una dirección IP) y el puerto al que está ligado el socket. Si no está ligado a ninguna interfaz específica, puede recibir el datagrama por cualquier interfaz. La dirección IP y el puerto de origen vendrán indicados en el propio datagrama recibido.
`void send(DatagramPacket p)`	Envía un datagrama. La dirección IP y el puerto deben estar especificados en el propio datagrama.
`setSoTimeout(int timeout)`	Establece un tiempo máximo de espera cuando se recibe un datagrama con el método *receive*. Si, pasado ese tiempo, no se ha recibido un datagrama, se generará una excepción de tipo `SocketTimeoutException`. Un valor 0 para `timeout` significa espera indefinida, es decir, la funcionalidad por defecto de `receive()`.
`void close()`	Cierra el *socket*.

`void connect(InetAddress address, int puerto)`
`void connect(SocketAddress addr)`
 	
`void disconnect{}`
	Conecta el *socket* a una dirección y puerto remotos, de manera que solo se pueden enviar datagramas a esa dirección y puerto, o recibirlos de esa dirección y puerto.
	El nombre de este método no debe inducir a confusión. Que exista un método `connect` no es contradictorio con el hecho de que UDP sea un protocolo no orientado a conexión. Más adelante se verá TCP, que sí es un protocolo de transporte orientado a conexión.
	Desconecta el *socket*.

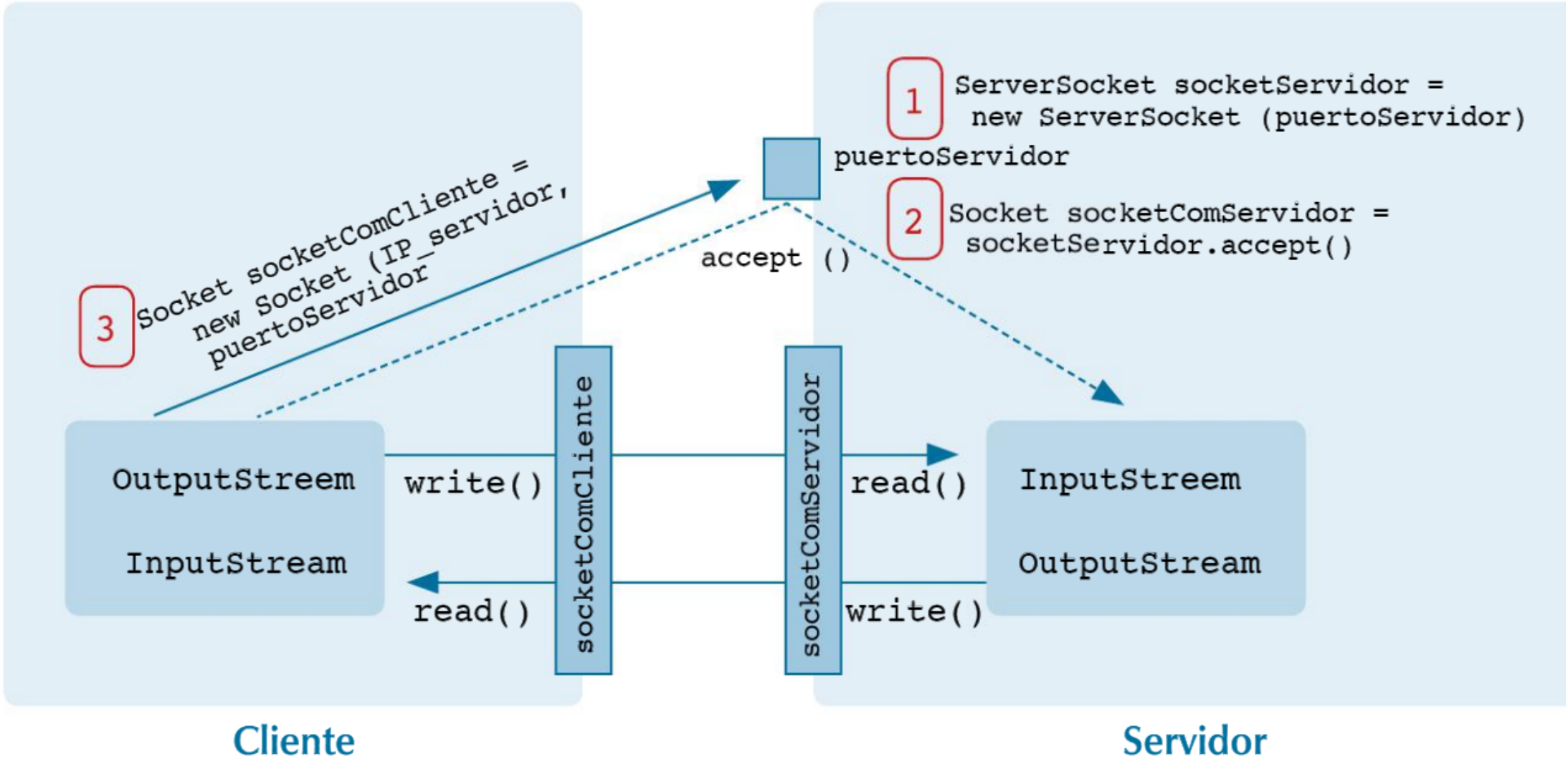
Método	Funcionalidad
`byte[] getData()` `int getLength()`	Obtienen los datos que contiene el datagrama como un array de bytes, y la longitud del array.
`InetAddress GetAddress()` `int getPort()`	Obtienen dirección IP y puerto desde donde se envió el datagrama. Podrían no estar informados. Un datagrama en el que no están informados sería análogo a una carta anónima, sin remitente.

Método	Funcionalidad

`DatagramPacket(byte[] buf, int length)`
`DatagramPacket(byte[] buf, int offset, int length)`
`DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)`
`DatagramPacket(byte[] buf, int length, InetAddress address, int port)`
	Diversos constructores. Los parámetros corresponden a:
	• Secuencia de bytes que enviar en el datagrama. Esta está en el array de bytes `buff`, de cuyos contenidos hay que enviar un número de bytes dado por `length`.
	• Dirección IP y puerto de destino, dados por `address` y `port`. Se pueden asignar también con `setAddress(InetAddress iaddr)` y `setSocketAddress(SocketAddress address)`

{get, set} { Address, Data, Length, Offset, Port, SocketAddress }	Diversos métodos *getter* y *setter*.

Socket TCP


1
ServerSocket socketServidor =
new ServerSocket (puertoServidor)
puertoServidor
2
Socket socketComServidor =
socketServidor.accept()
accept ()
3
Socket socketComCliente =
new Socket (IP_servidor,
puertoServidor
OutputStreem
InputStream
write()
read()
socketComCliente
socketComServidor
read()
write()
InputStreem
OutputStream
Cliente
Servidor

Clases comunicaciones en red

Métodos de la clase ServerSocket

Método	Funcionalidad

`ServerSocket()`
`ServerSocket(int port)`
`ServerSocket(int port, int backlog)`
`ServerSocket(int port, int backlog, InetAddress bindAddr)`
	Distintos creadores para la clase `ServerSocket`.
	Si no se especifica el puerto, deberá ligarse más tarde a un puerto con el método `bind`.
	Si se especifica la dirección IP de una interfaz activa (`bindAddr`), se ligará a ella. Si no, a las direcciones IP de todas las interfaces activas (*wildcard address*).
	`backlog` es el tamaño máximo de una cola de peticiones de conexión de clientes pendientes de ser atendidas.

`Socket accept()`	Queda a la espera de una petición de conexión de un cliente. Una vez recibida, la acepta. Devuelve el nuevo *socket* creado para la comunicación con el cliente.
`void close()`	Cierra el *socket*.

Clases comunicaciones en red

Métodos de la clase ServerSocket- https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/net/ServerSocket.html

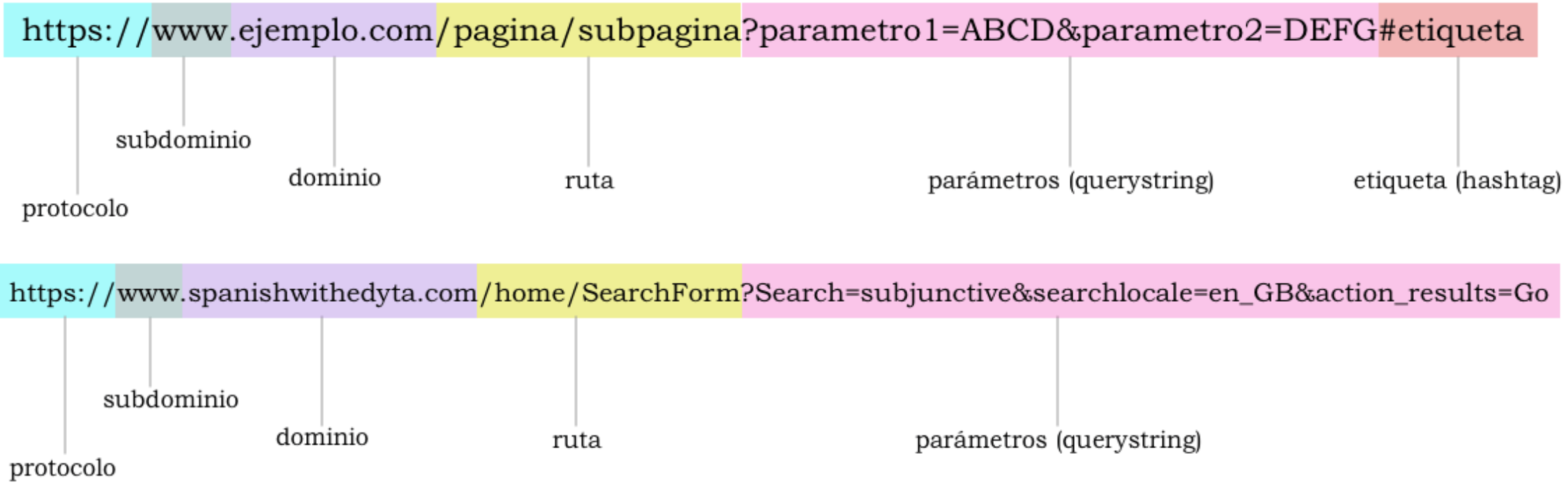
Método	Funcionalidad

`ServerSocket()`
`ServerSocket(int port)`
`ServerSocket(int port, int backlog)`
`ServerSocket(int port, int backlog, InetAddress bindAddr)`
	Distintos creadores para la clase `ServerSocket`.
	Si no se especifica el puerto, deberá ligarse más tarde a un puerto con el método `bind`.
	Si se especifica la dirección IP de una interfaz activa (`bindAddr`), se ligará a ella. Si no, a las direcciones IP de todas las interfaces activas (*wildcard address*).
	`backlog` es el tamaño máximo de una cola de peticiones de conexión de clientes pendientes de ser atendidas.

`Socket accept()`	Queda a la espera de una petición de conexión de un cliente. Una vez recibida, la acepta. Devuelve el nuevo *socket* creado para la comunicación con el cliente.
`void close()`	Cierra el *socket*.

URL

URL Uniform Resource Locator y es la dirección única y específica que se asigna a cada uno de los recursos disponibles World Wide Web (WWW) para que puedan ser localizados



Clases comunicaciones en red

Clase URL

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/net/URL.html

CONSTRUCTOR	DESCRIPTION
URL(String url) throws MalformedURLException	This constructor creates the URL object from the specified **String**
URL(String protocol, String host, int port, String file) throws MalformedURLException	This constructor creates the URL object using the specified **protocol, host, port number,** and **file**
URL(String protocol, String host, int port, String file, URLStreamHandler handler)	This constructor creates the URL object using the specified **protocol, host, port number, file,** and **handler**
URL(String protocol, String host, String file)	This Constructor creates the URL from the specified **protocol, host,** and **file**
URL(URL context, String spec)	This Constructors creates the URL by parsing the given **spec** within the context
URL(URL context, String spec, URLStreamHandler handler)	This Constructors creates a URL by parsing the given **spec** with the specified **handler** within the context

Clases comunicaciones en red

Clase URL https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/net/URL.html

METHODS	DESCRIPTION
public String getQuery()	This Method gets the **query part** of the URL
public String getPath()	This Method gets the **path part** of the URL
public String getUserInfo()	This Method gets the **user information** of the URL
public String getAuthority()	This Method gets the **authority information** of the URL
public int getPort()	This Method gets the **port number** of the URL
public int getDefaultPort()	This Method gets the **default port** number of the protocol which is associated with the URL
public String getProtocol()	This Method gets the **protocol name** of the URL
public String getHost()	This Method gets **host name** of the URL, format conforms to RFC 2732
public String getFile()	This Method gets the **file name** of the URL, it will be same as getPath() when there is no query parameter
public String getRef()	This Method gets the **anchor / reference** of the URL

Métodos de petición HTTP

GET El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

HEAD El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

POST El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

PUT El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

DELETE El método DELETE borra un recurso en específico.

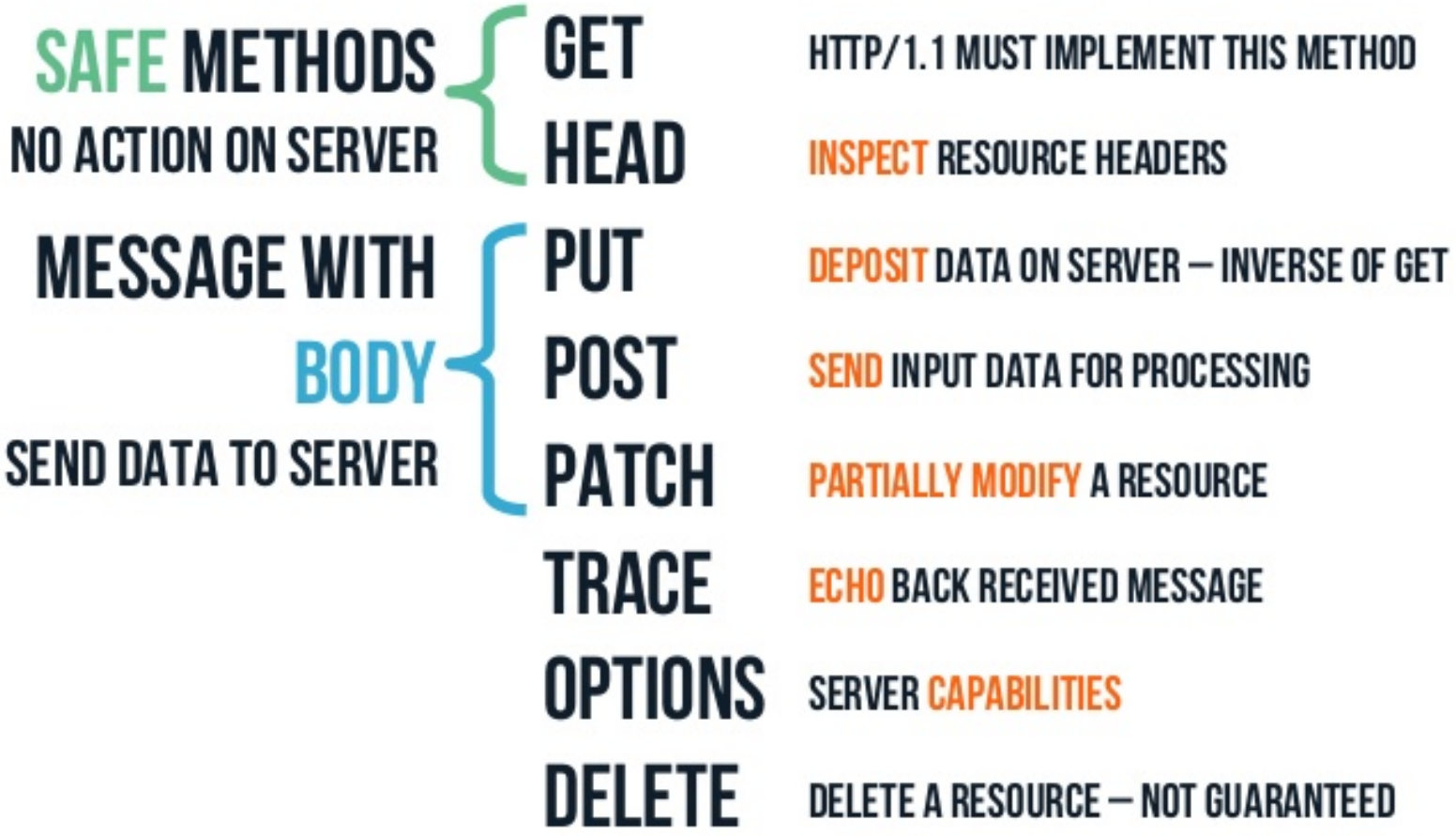
CONNECT El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

OPTIONS El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

TRACE El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

PATCH El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

Métodos de petición HTTP



Material de estudio (Java Networking)

- https://docs.oracle.com/javase/tutorial/networking/index.html
- https://www.javatpoint.com/java-networking
- https://www.tutorialspoint.com/java/java_networking.htm
- http://tutorials.jenkov.com/java-networking/index.html
- https://www.javainterviewpoint.com/java-url-class/

Material de estudio (redes)

- Curso de redes desde cero
- Curso de redes (Avanzado)
- https://www.xataka.com/servicios/internet-explicada-cualquier-persona-pueda-entender-como-funciona-que-a-veces-parte-queda-fuera-combate
- https://www.redeszone.net/tutoriales/internet/protocolos-basicos-redes/
- https://www.profesionalreview.com/redes/
- https://developer.mozilla.org/es/docs/Web/HTTP/Overview
- https://yosoy.dev/peticiones-http-get-post-put-delete-etc/
- http://estilow3b.com/metodos-http-post-get-put-delete/
- http://cv.uoc.edu/UOC/a/moduls/90/90_329/web/main/m2/v4.html
- https://www.ionos.es/digitalguide/servidores/know-how/internet-protocol-definicion-y-fundamentos/
- https://www.ionos.es/digitalguide/servidores/know-how/que-es-tcp-transport-control-protocol/
- https://www.ionos.es/digitalguide/servidores/know-how/udp-user-datagram-protocol/
- https://www.redeszone.net/tutoriales/internet/tcp-udp-caracteristicas-uso-diferencias/
- https://www.interviewbit.com/networking-interview-questions/
- https://www.rfc-editor.org/rfc-index.html
- https://www.rfc-es.org/
- https://blog.nipraas.com/2020/07/basic-tcp-analysis-with-wireshark-part-1.html