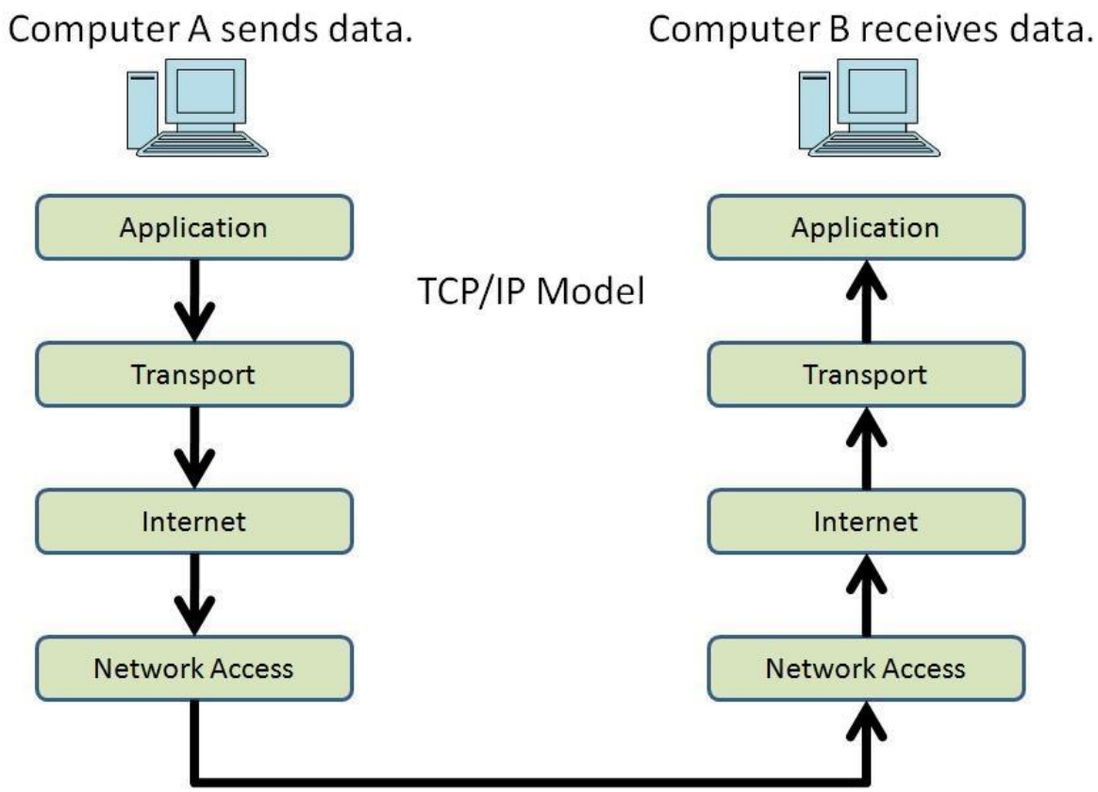
Comunicación UDP

Desarrollo de Sistemas en Red

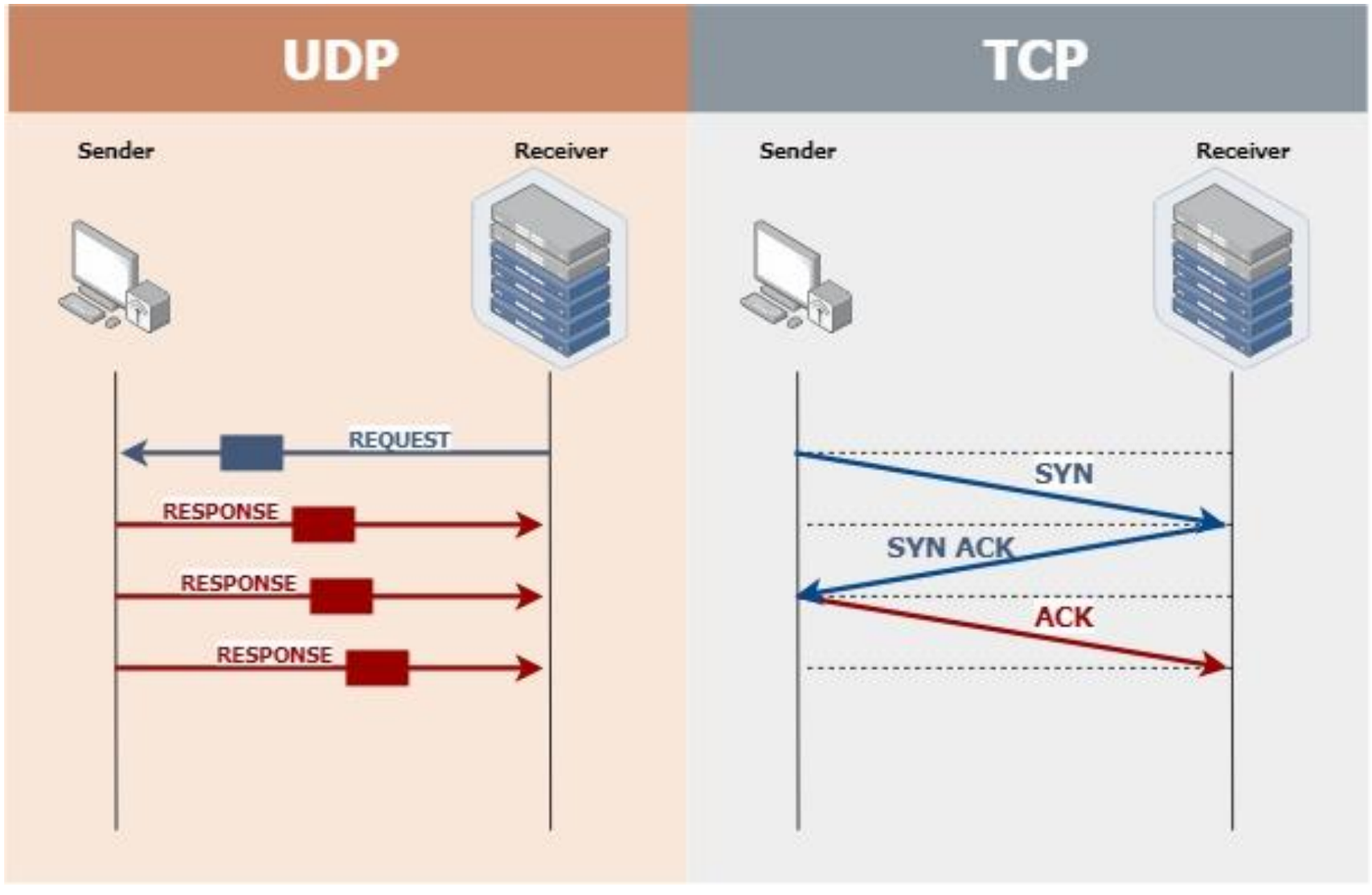
Introducción

- Cuando se requiere la transmisión de un flujo de datos entre dos equipos, **el protocolo TCP establece un conducto exclusivo** entre dichos equipos por el cual los datos serán transmitidos y este perdurará hasta que la transmisión haya finalizado.



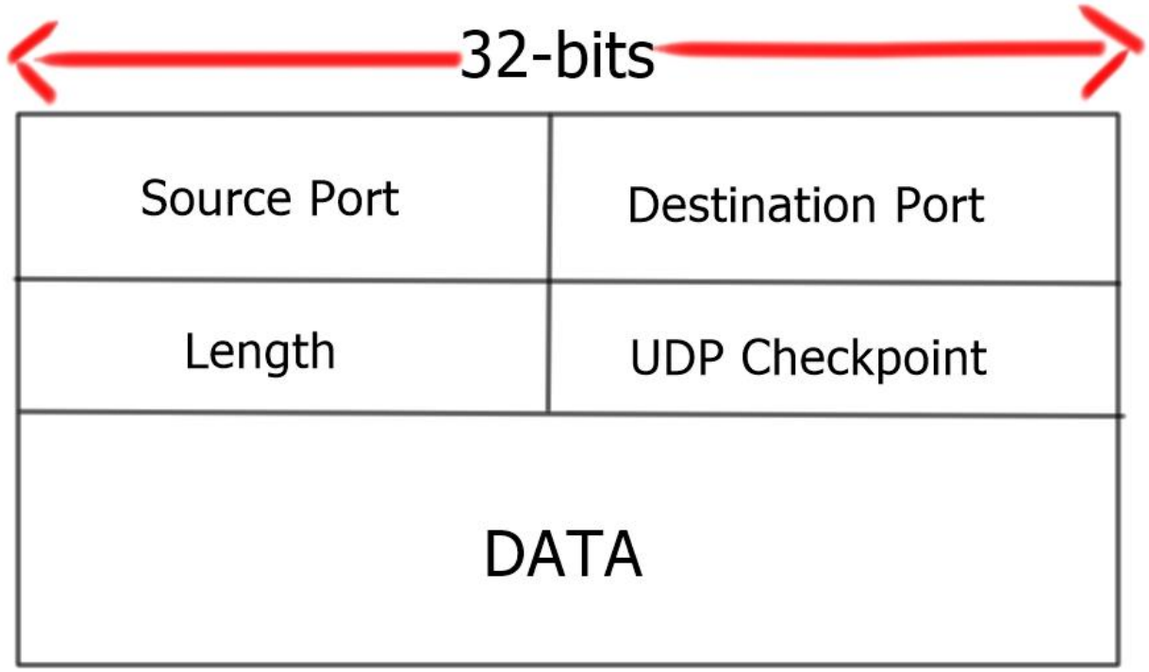
Sockets UDP

- Por su parte el protocolo UDP ofrece un tipo de conexión para transmisiones sujetas a limitaciones temporales, como la reproducción de vídeo.



Protocolo UDP

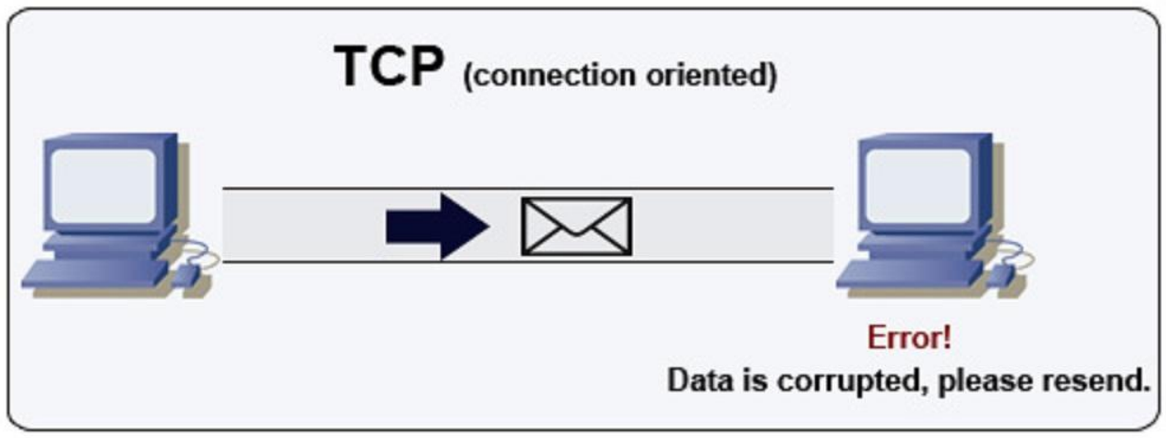
El protocolo **UDP no es orientado a la conexión** debido a que la forma de trasmitir los datos no garantiza en primera instancia su llegada al destino, e incluso si este llegara al destino final, **tampoco garantiza la integridad de los datos**

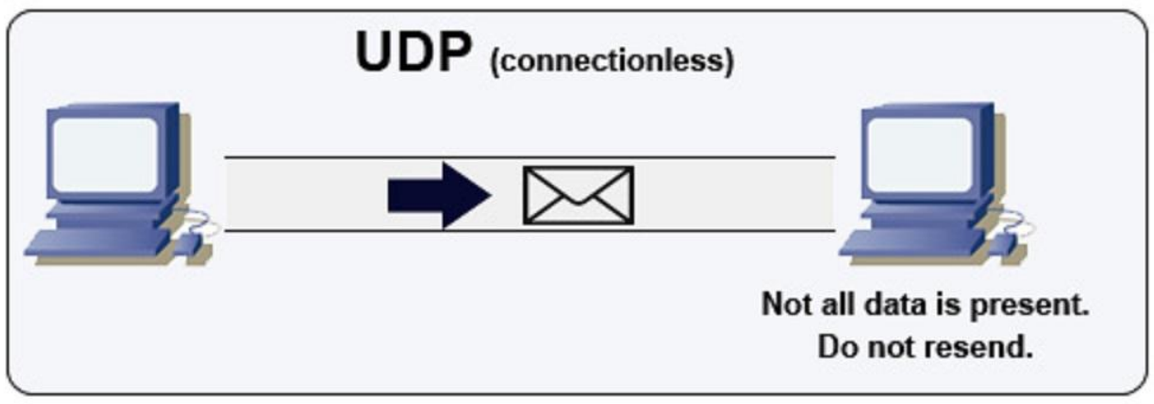


UDP segment structure

Protocolo UDP

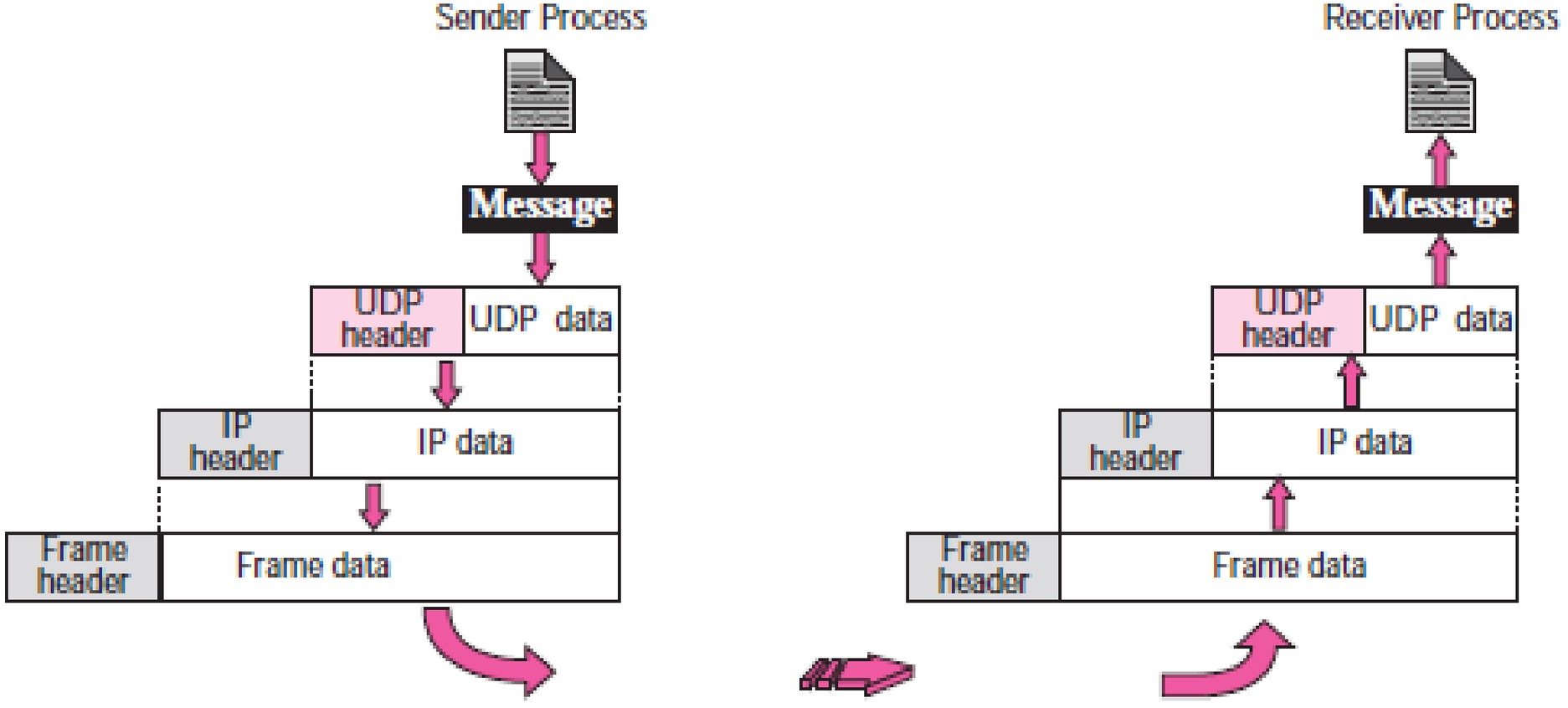
UPD **utiliza datagramas**, los cuales contienen una porción de la información y que son enviados a la red en espera de ser capturados por el equipo destino





Protocolo UDP

Cuando el destino captura los datagramas debe reconstruir la información, para esto debe ordenar la información que recibe ya que la información transmitida no viene con un orden específico, además se debe tener conciencia de que no toda la información va a llegar

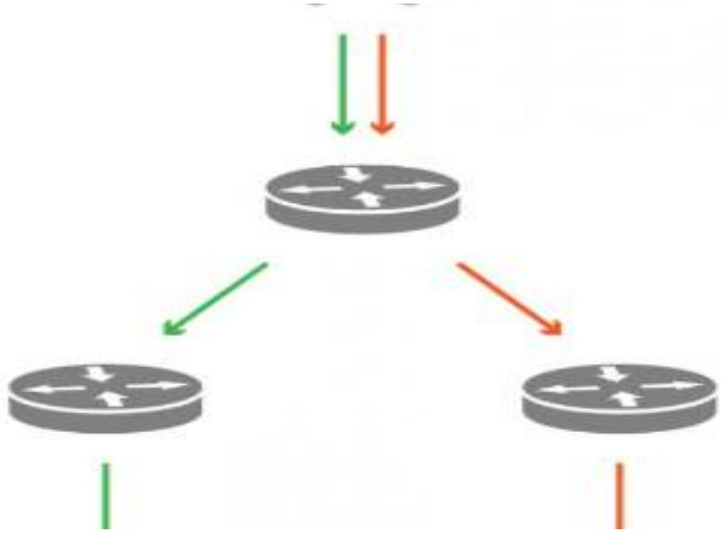


a. Encapsulation

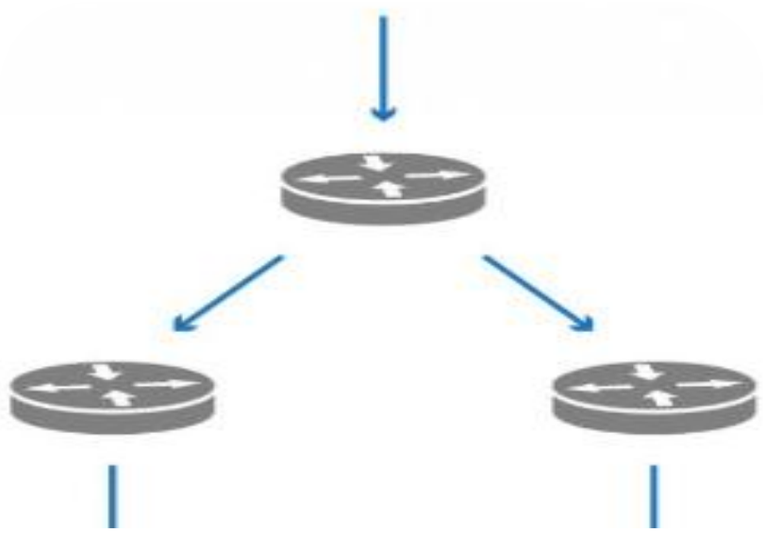
b. Decapsulation

Casting

Casting en redes informáticas significa **transmitir datos** (flujo de paquetes) a través de una red. Los siguientes son los diferentes tipos de fundición utilizados en la creación de redes:



Unicast

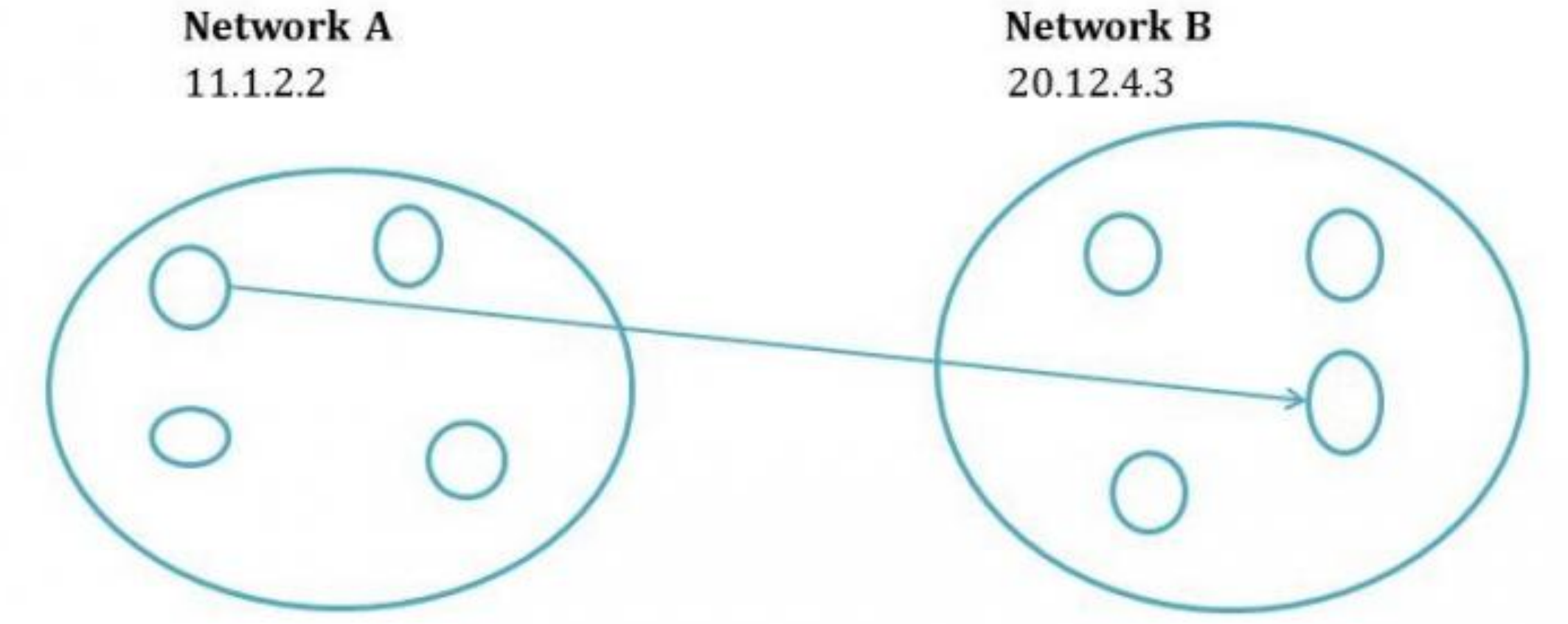


Multicast

Unicast (uno a uno)

- En la transmisión Unicast, los datos se transfieren desde **un solo remitente** (o un solo host de origen) a **un solo receptor** (o un solo host de destino).
- Los conmutadores de red escuchan las direcciones MAC de los dispositivos en las redes a las que están conectados. Luego, pueden reenviar paquetes solo a aquellas redes que contienen dispositivos con las direcciones MAC conectadas.
- Unicast gradualmente se vuelve **menos eficiente** a medida que más receptores necesitan ver datos idénticos.

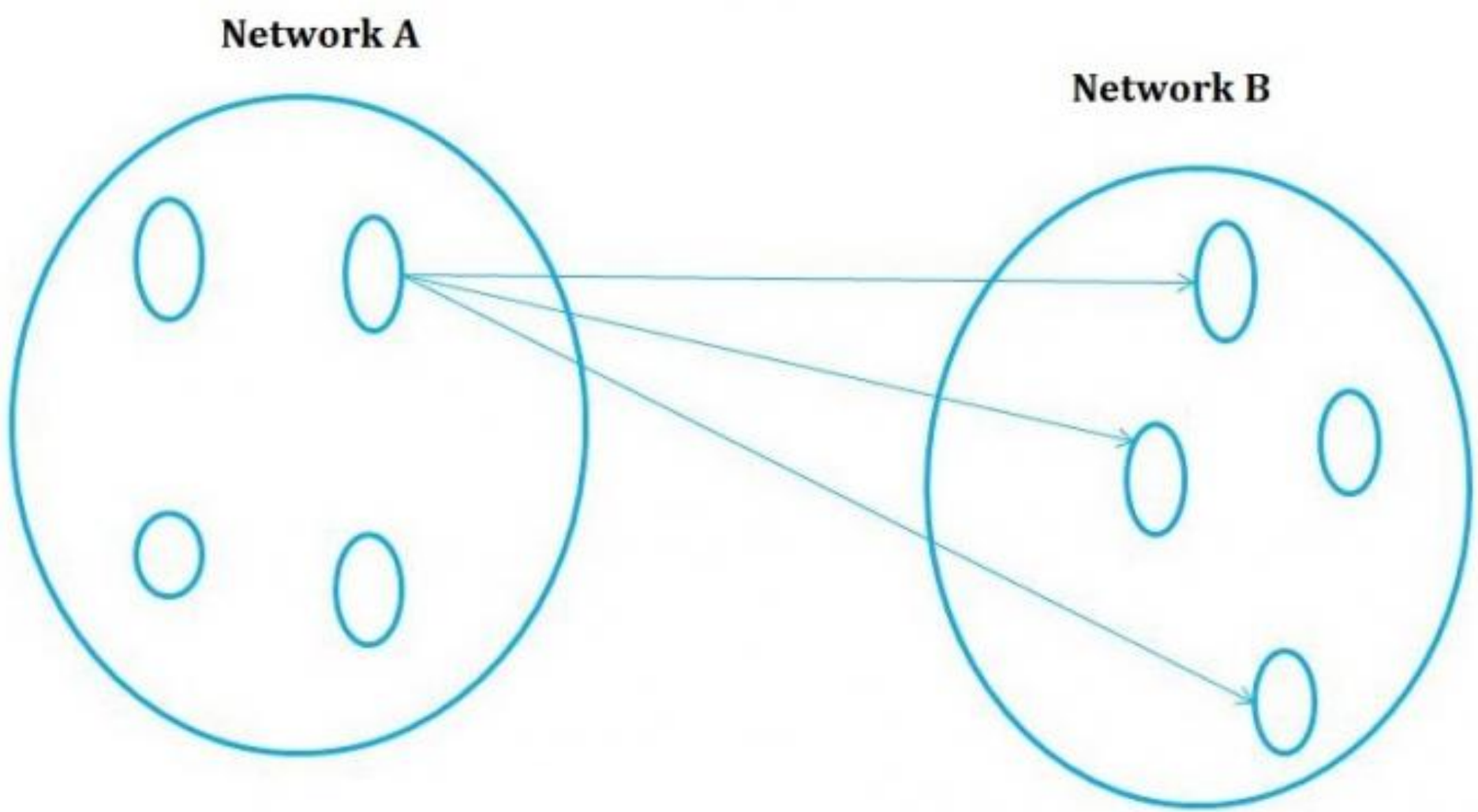
Trasmisión Unicast



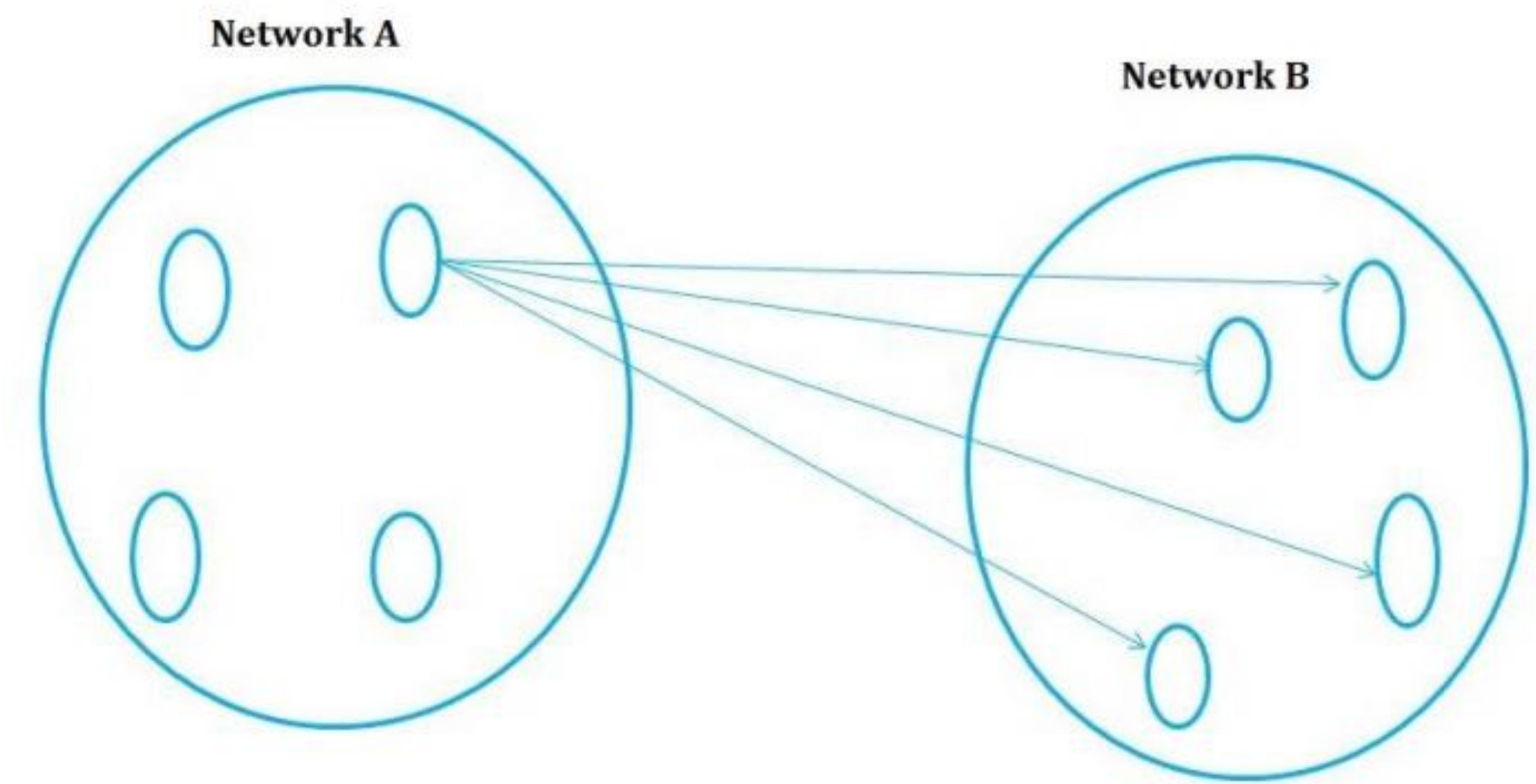
Multicast (uno a muchos)

- Cuando los datos se transmiten desde **un host de origen único** a un grupo específico de hosts que tienen interés en recibir los datos, se conoce como transmisión de multidifusión.
- La multidifusión puede ser más eficiente que la unidifusión cuando diferentes grupos de receptores necesitan ver los mismos datos.

Transmisión Multicast

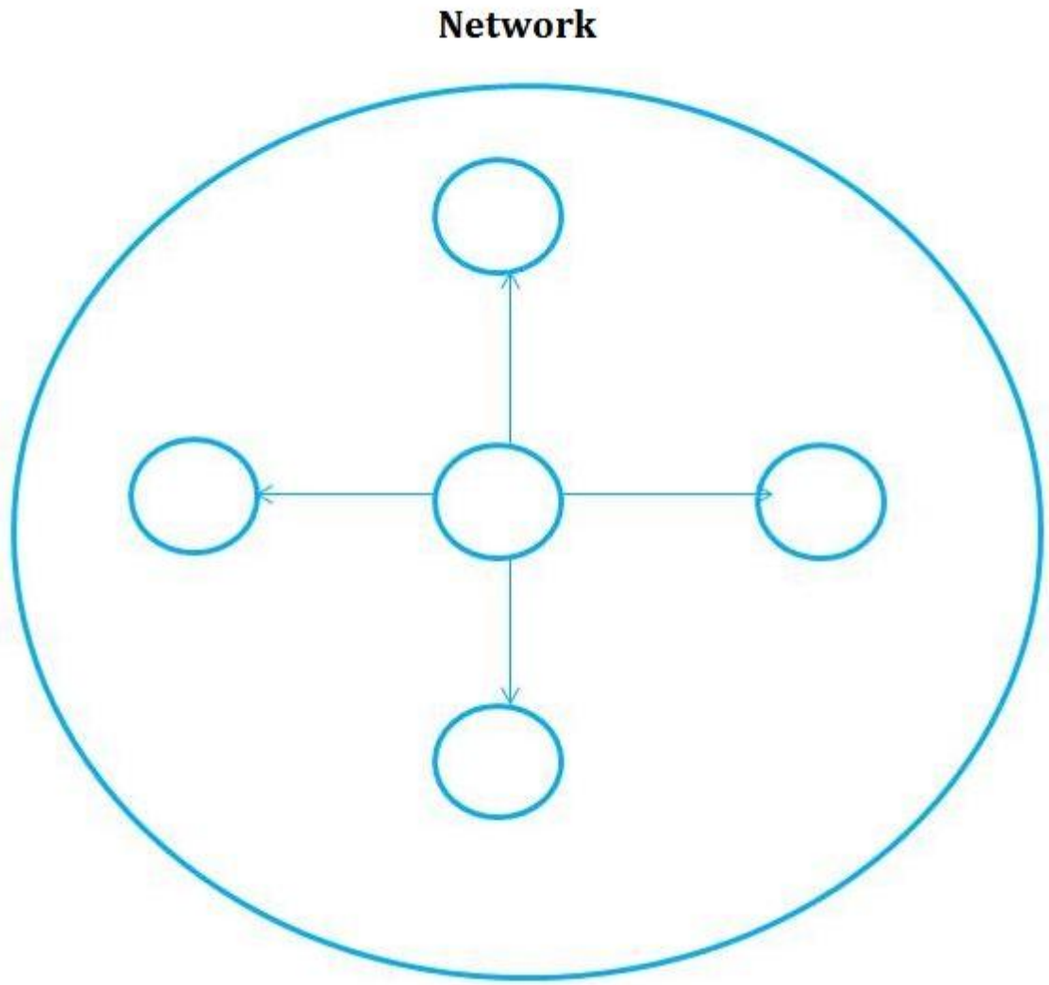


Trasmisión Broadcast Dirigida



Transmisión Broadcast Limitada

- Los datos se transmiten desde **un solo host** de origen a todos los demás hosts que residen en la misma red



Práctica en Java

En esta práctica veremos como da soporte Java a UDP, tanto para envío unicast, como multicast.

Clases principales

Java proporciona tres clases para dar soporte a la comunicación vía datagramas UDP, todas ellas contenidas en el paquete **java.net**.

- DatagramSocket,
- DatagramPacket y
- MulticastSocket

DatagramPacket

- Proporciona constructores para crear **instancias** a partir **de los datagramas recibidos** y para crear instancias de datagramas que van **a ser enviados**.
- Constructores para datagramas que van a ser enviados:

```
DatagramPacket(byte[] buf, int length)

DatagramPacket(byte[] buf, int length,
InetAddress address, int port)
```

Estos constructores crean una instancia de datagrama compuesta por una cadena de bytes que almacena el mensaje, la longitud del mensaje y la dirección de Internet y el número de puerto local del conector destino, tal y como sigue:

- Constructores para datagramas recibido:

```
DatagramPacket(byte[] buf, int offset, int length)

DatagramPacket(byte[] buf, int offset, int length,
InetAddress address, int port).
```

- Estos constructores nos permiten crear instancias de los datagramas recibidos, especificando la **cadena de bytes** en la que alojar el mensaje, la longitud de la misma y el offset dentro de la cadena.

- Dentro de esta clase hay métodos para obtener los diferentes componentes de un datagrama, tanto recibido como enviado:
  - **`getData`**() : para obtener el mensaje contenido en el datagrama.
  - **`getAddress`**() : para obtener la dirección IP.
  - **`getPort`**(): para obtener el puerto.

DatagramSocket

maneja sockets para enviar y recibir datagramas UDP. Proporciona tres constructores:

- **`DatagramSocket()`:** constructor sin argumentos que permite que el sistema elija un puerto entre los que estén libres y selecciona una de las direcciones locales.
- **`DatagramSocket(int port)`:** constructor que toma un número de puerto como argumento, apropiado para los procesos que necesitan un número de puerto (servicios).
- **`DatagramSocket(int port, InetAddress laddr)`:** constructor que toma como argumentos el número de puerto y una determinada dirección local.

La clase DatagramSocket proporciona varios métodos, destacamos los más utilizados:

1. **send**(DatagramPacket p) y **receive**(DatagramPacket p): estos métodos sirven para transmitir datagramas entre un par de conectores.
   - El argumento de send es una instancia de DatagramPacket conteniendo el mensaje y el destino.
   - El argumento de **receive** es un DatagramPacket vacío en el que colocar el mensaje, su longitud y su origen. Ambos métodos pueden lanzar excepciones IOException

2. **`setSoTimeout`**`(int timeout)`: este método permite establecer un tiempo de espera límite. Cuando se fija un límite, el método **`receive`** se bloquea durante el tiempo fijado y después lanza una excepción `InterruptedIOException`

3. **`connect`**`(InetAddress address, int port)`: este método se utiliza para conectarse a un puerto remoto y a una dirección Internet concretos, en cuyo caso el conector sólo podrá enviar y recibir mensajes de esa dirección.

MulticastSocket

- La operación de `multicast` consiste en enviar un único mensaje desde un proceso a cada uno de los miembros de un grupo de procesos, de modo que la pertenencia a un grupo sea transparente al emisor, es decir, el emisor no conoce el número de miembros del grupo ni sus direcciones IP.

- Un grupo `multicast` está especificado por una dirección IP clase D y un puerto.
- **`Las direcciones IP clase D`** están en el rango `224.0.0.0` a `239.255.255.255`, dentro de este rango existen direcciones reservadas, en concreto, la `224.0.0.1` y la `224.0.0.255`.
- El resto de las direcciones del rango pueden ser utilizadas por grupos temporales, los cuales deben ser creados antes de su uso y dejar de existir cuando todos los miembros lo hayan dejado.

La clase MulticastSocket proporciona dos constructores alternativos:

- **`MulticastSocket()`:** que crea el socket en cualquiera de los puertos locales libres.
- **`MulticastSocket(int port)`:** que crea el socket en el puerto local indicado.

- Un proceso puede pertenecer a un grupo multicast invocando el método **`joinGroup(InetAddress mcastaddr)`** de su socket multicast.
  - Así, el socket pertenecerá a un grupo de multidifusión en un puerto dado y recibirá los datagramas enviados por los procesos en otras computadoras a ese grupo en ese puerto.
- Un proceso puede dejar un grupo dado invocando el método **`leaveGroup(InetAddress mcastaddr)`** de su socket multicast.

- Para enviar datos a un grupo multicast se utiliza el método **`send(DatagramPacket p, byte ttl)`**, este método es muy similar al de la clase `DatagramSocket`, la diferencia es que este datagrama será enviado a todos los miembros del grupo multicast.

- El parámetro **`TTL, Time-To-Live`**, lo pondremos siempre a 1, valor por defecto, para que sólo se difunda en la red local.

- Para recibir datos de un grupo multicast se utiliza el método **`receive(DatagramPacket p)`** de la clase `DatagramSocket` superclase de `MulticastSocket`.

- Es necesario pertenecer a un grupo para recibir mensajes multicast enviados a ese grupo, pero no es necesario para enviar mensajes.

Ejercicios

Ejercicio 1

- Crear una aplicación cliente/servidor, en la que el servidor proporciona la hora y el día a los clientes que lo soliciten.
- El programa cliente realizará una petición al servidor y esperará la respuesta un tiempo limitado (5000 milisegundos).
- Si recibe la respuesta, enviará a la salida estándar el día y la hora proporcionados por el servidor.
- Si después de ese tiempo no recibe una respuesta, enviará a la salida estándar un mensaje de error.

Ejercicio 2

- Modificar el cliente del ejercicio 1 para que compare la hora local con la hora remota e indique, con un mensaje por la salida estándar, la diferencia entre ellas.

Ejercicio 3

- Elaborar un chat utilizando multicast.
- El programa tendrá una sencilla interfaz gráfica.
- En la parte superior de la ventana, aparecerán los mensajes enviados por los diferentes participantes en el chat, y la parte inferior, será la zona en la que cada participante escribirá sus propios mensajes, cada vez que se escribe una línea se envía a todos los participantes.

Recursos

- https://docs.oracle.com/javase/tutorial/networking/datagrams/index.html
- https://www.tutorialspoint.com/unicast-broadcast-and-multicast-in-computer-networks