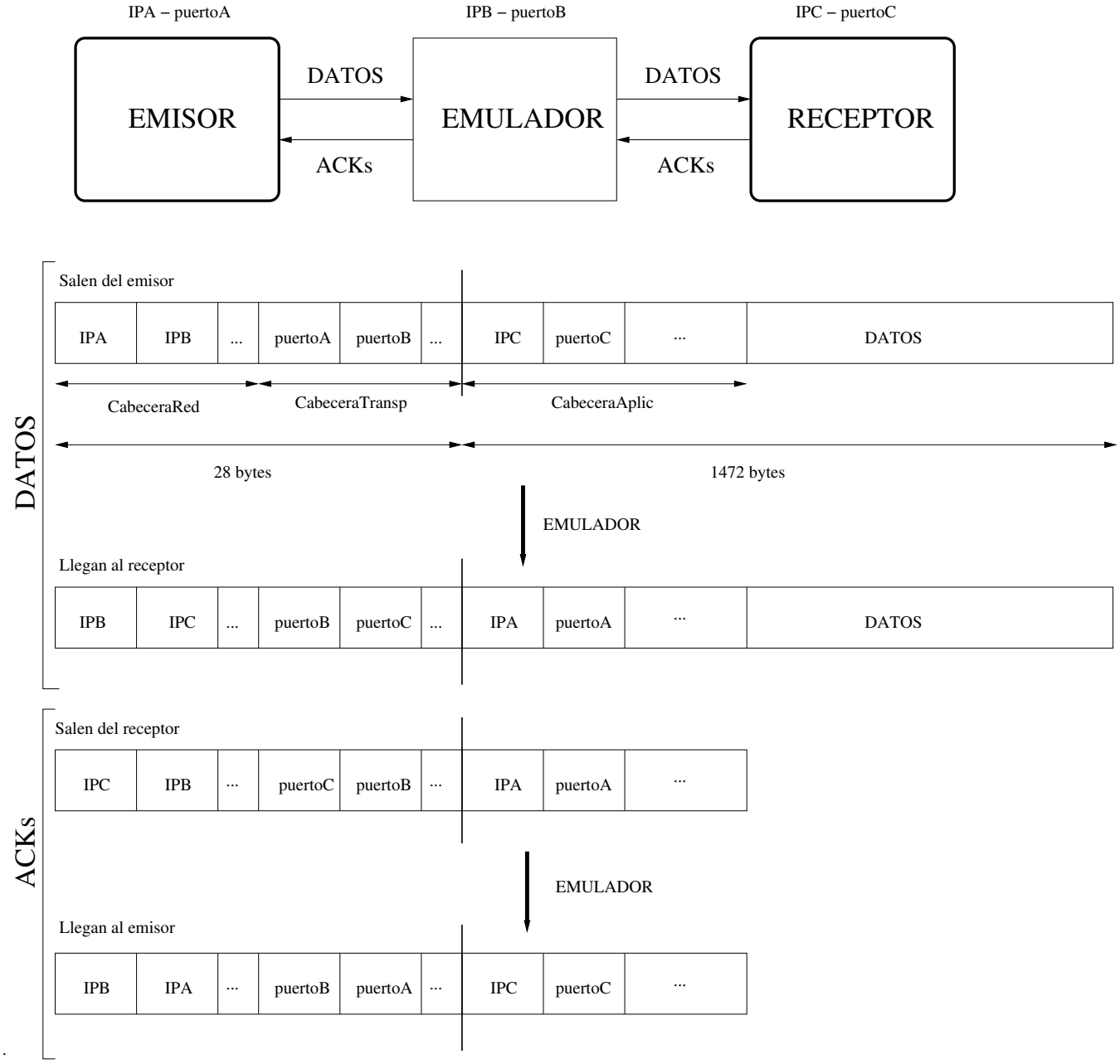
TRANSMISIÓN LIBRE DE ERRORES DE UN FICHERO


IPA – puertoA
IPB – puertoB
IPC – puertoC
EMISOR
DATOS
ACKs
EMULADOR
DATOS
ACKs
RECEPTOR
DATOS
Salen del emisor
IPA
IPB
...
puertoA
puertoB
...
IPC
puertoC
···
DATOS
CabeceraRed
CabeceraTransp
CabeceraAplic
28 bytes
1472 bytes
EMULADOR
Llegan al receptor
IPB
IPC
...
puertoB
puertoC
...
IPA
puertoA
···
DATOS
ACKs
Salen del receptor
IPC
IPB
···
puertoC
puertoB
···
IPA
puertoA
···
EMULADOR
Llegan al emisor
IPB
IPA
···
puertoB
puertoA
···
IPC
puertoC
···

Los tres procesos

- **La aplicación emisora**

  `ro2021send fichEntrada IPC puertoC IPB puertoB`

  - Envía al emulador bloques de bytes del fichero, indicándole en la cabecera de aplicación la IP y el puerto del receptor.
    `|IPC|puertoC|...|datos|`
  - Puede aprovechar la cabecera de aplicación para enviar números de secuencia, sellos temporales, ...
  - Recibe del emulador asentimientos
    `|IPC|puertoC|...|`
  - Se puede hacer un último envío sin datos para indicar al receptor el fin de la transmisión del fichero.

- **La aplicación receptora**

  `ro2021recv fichSalida puertoC`

  - Recibe del emulador bloques de bytes del fichero
    `|IPA|puertoA|...|datos|`
  - Obtiene la IP y el puerto del emisor de la cabecera del nivel de aplicación.
  - Obtiene la IP y el puerto del emulador de los metadatos asociados a los datagramas recibidos.
  - Envía al emulador asentimientos, indicándole en la cabecera de aplicación la IP y el puerto del emisor.
    `|IPA|puertoA|...|`
  - Puede aprovechar la cabecera de aplicación para enviar números de secuencia, sellos temporales, ...

Los tres procesos

- **El emulador de red**
  - Recibe bloques de bytes del emisor y los envía al receptor, cuyos datos de direccionamiento le llegan en la cabecera de aplicación. Cambia dichos datos por los datos de direccionamiento del emisor.
    `|IPC|puertoC|...|datos|  ->  |IPA|puertoA|...|datos|`
  - Recibe asentimientos del receptor y los envía al emisor, cuyos datos de direccionamiento le llegan en la cabecera de aplicación. Cambia dichos datos por los datos de direccionamiento del receptor.
    `|IPA|puertoA|...|  ->  |IPC|puertoC|...|`
  - Emula distintos escenarios de red, perdiendo o retardando los paquetes de forma determinista o aleatoria, lo que puede provocar que lleguen desordenados al receptor.

    Ejemplo:

    ```
    shufflerouter -d 0.01 -m 5 -r 10 -p 4000

      -d probabilidad de pérdidas (por defecto 0.0)
      -m y -r retardo aleatorio uniforme en el intervalo [m,m+r] (por defecto 0 y 0)
      -p puertoB (por defecto 2021)
    ```

    En el ejemplo, se pierden un 1% de los paquetes y los demás se retardan de forma aleatoria, siguiendo una distribución uniforme en el intervalo [5,15]

El temporizador de retransmisión

- El temporizador de retransmisión, RTO, debe adaptarse dinámicamente a los cambios en el medio.
- Para ello, el emisor debe medir continuamente el RTT, tiempo de ida y vuelta en la red.
- Para medir el RTT:
  - Se puede introducir un sello temporal que indique el tiempo en el que se transmitió cada paquete.
  - El receptor no varía el sello temporal al enviar el asentimiento del paquete.
  - Cuando un asentimiento llega al emisor, se puede calcula la diferencia entre el tiempo actual y el tiempo que trae el asentimiento en el sello temporal, obteniéndose una medición del RTT.
- A partir de esa medición del RTT, para estimar el RTO:
  - Inicialmente:
    $\widehat{rtt[1]} = rtt[1]$
    $\widehat{\sigma_{rtt}[1]} = \frac{rtt[1]}{2}$
  - Para cada medición del RTT:
    $\widehat{rtt[k]} = (1-\alpha)\widehat{rtt[k-1]} + \alpha rtt[k]$
    $\widehat{\sigma_{rtt}[k]} = (1-\beta)\widehat{\sigma_{rtt}[k-1]} + \beta|\widehat{rtt[k]} - rtt[k]|$
  - Y con esos valores, una posible estimación del RTO:
    $RTO = \widehat{rtt[k]} + 4\widehat{\sigma_{rtt}[k]}$

Parada y Espera

- **El emisor**

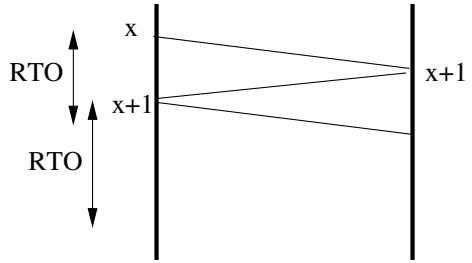
  Mantiene el número de secuencia del último paquete enviado.

  Envía el primer paquete, anotando el instante de envío y poniendo el temporizador de espera por el asentimiento igual al RTO inicial.

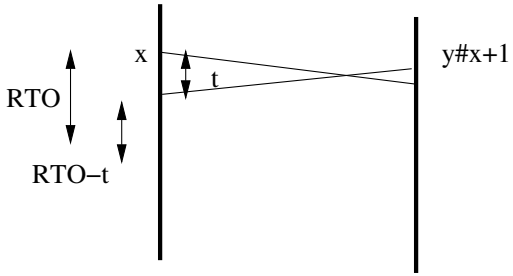
  Continuamente (hasta que llega el asentimiento del último bloque del fichero):

  - Espera el asentimiento.
    * Si llega un asentimiento, si se usan sellos temporales el emisor puede aprovechar para obtener una medición del RTT y con ello una estimación del RTO
      . y si el asentimiento es el esperado, puede pasar a enviar el siguiente paquete, anotando el tiempo de envío y poniendo el temporizador de espera por el asentimiento igual al RTO (el RTO estático o el último RTO estimado).
      . pero si el asentimiento no es el esperado, debe pasar de nuevo a esperar, pero poniendo temporizador de espera al RTO (el RTO estático o el último RTO estimado) menos el tiempo transcurrido desde que se envió el último paquete.
    * Si vence el temporizador debe enviar de nuevo el paquete, anotando el tiempo de envío y poniendo el temporizador de espera al RTO (el RTO estático o el último RTO estimado).

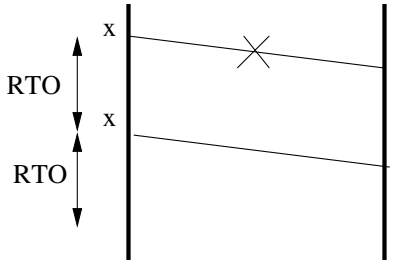
Parada y Espera



ASENTIMIENTO ESPERADO



ASENTIMIENTO NO ESPERADO



FIN TEMPORIZADOR

RTO: RTO estático o el último RTO estimado

Parada y Espera

- **El receptor**

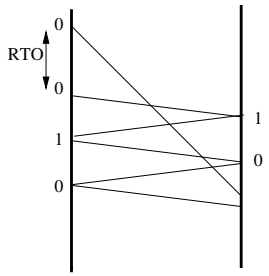
  Mantiene el número de secuencia del siguiente paquete esperado.

  Continuamente (hasta que detecta que ha llegado todo el fichero):
  - Espera un paquete.
  - Si llega el paquete esperado lo acepta e incrementa el número de secuencia del siguiente paquete esperado.
  - Y en cualquier caso envía un asentimiento con el número de secuencia del siguiente paquete esperado.

- **Los números de secuencia**

  Es necesario usar números de secuencia crecientes (no llega con alternar 0 y 1).

  Con retardos aleatorios puede ocurrir, por ejemplo, que una retransmisión adelante a una transmisión que se ha retardado demasiado, siga todo bien y la transmisión retardada sea interpretada más adelante como una nueva transmisión.



Go-Back-N

- **El emisor**
  - Mantiene una ventana de paquetes enviados y pendientes de asentimiento, así como el instante de transmisión de cada paquete de la ventana para gestionar el vencimiento de los temporizadores.
  - Puede aprovechar el sello temporal que traen los asentimientos para hacer mediciones del RTT y estimaciones del RTO.
  - Require intercalar eventos de envío y recepción de forma no bloqueante.
  - Si la ventana está llena sólo se puede esperar por el asentimiento de un paquete dentro de la ventana, actualizando el temporizador de espera al último RTO estimado menos el tiempo transcurrido desde que se envió el primer paquete de la ventana.
    Si vence este temporizador, se debe retransmitir toda la ventana.
  - Pero mientras la ventana no está llena se pueden transmitir más paquetes, mientras se chequea la recepción del asentimiento de alguno de los paquetes de la ventana.
  - Si se recibe el asentimiento de un paquete dentro de la ventana, se desplaza el principio de la ventana al siguiente paquete.
- **El receptor**

  Igual que en Parada y Espera

java.nio

Nuevas clases opcionales en Parada y Espera pero obligatorias en Go-Back-N

- **DatragramChannel:** Similar a DatagramSocket
- **Select:** Permite asociar canales y registrar eventos en un selector, lo que facilita la gestión de eventos de envío o recepción de forma bloqueante o no bloqueante
  - **Método selectNow():** Sale inmediatamente, indicando el número de canales asociados al selector en los que se ha producido al menos un evento de los registrados.
  - **Método select():** Se bloquea hasta que en al menos un canal de los asociados al selector se produce alguno de los eventos registrados.
  - **Método select(timeout):** Se bloquea como máximo timeout milisegundos hasta que en al menos un canal de los asociados al selector se produce alguno de los eventos registrados.
- **ByteBuffer:** Facilita la creación y procesamiento de los paquetes de aplicación (con su cabecera particular).

Ejemplo canal y selector

```
DatagramChannel channel = DatagramChannel.open();
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ|SelectionKey.OP_WRITE);

while(true) {
  int readyChannels = selector.selectNow();
  if(readyChannels == 0) continue;
  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

  while(keyIterator.hasNext()) {
    SelectionKey key = keyIterator.next();
    if (key.isReadable()) {
        // a channel is ready for reading (or receiving)
    } else if (key.isWritable()) {
        // a channel is ready for writing (or sending)
    }
    keyIterator.remove();
  }
}
```