UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

APPLET-SIMULADOR DIDACTICO DE PROTOCOLOS DE VENTANA (PROTOCOLOS ARQ)

Autor: José Mª González Heredia
Directores: Francisco Miguel Monzó Sánchez
Francesc Burrull i Mestre
Cartagena, Julio 2007


UNIVERSIDAD POLITECNICA DE CARTAGENA
Fechos Allend Mar
anno MCMXCVIII

Índice General

APPLET-SIMULADOR DIDACTICO DE PROTOCOLOS DE VENTANA (PROTOCOLOS ARQ)

Capítulo 1

Introducción y objetivos

Los Protocolos ARQ son protocolos de control de flujo que esencialmente se caracterizan porque solicitan retransmisiones de una o varias tramas de modo automático cuando se producen errores o pérdidas de información, sin intervención de agentes externos al nivel de enlace.

Por lo tanto, los protocolos de solicitud de repetición automática o ARQ (Automatic Repeat Request) retransmiten aquellos paquetes de datos que han llegado con errores, utilizando para ello códigos de detección de errores. Estos protocolos de repetición automática nacieron como una alternativa a la "verificación de eco", es decir, un método manual de errores que se utiliza aún en los terminales de mainframes, consistente en una técnica muy sencilla: cuando el usuario teclea un carácter, éste es transmitido hasta la computadora y reenvía de nuevo hacia el terminal que lo presenta en pantalla, si el usuario se da cuenta de que lo que se muestra no coincide con lo que él esperaba, es el usuario el encargado de realizar la corrección de errores, usando los caracteres de control de los que dispone el terminal.

Existe diferentes protocolos ARQ, el protocolo de parada y espera (**stop & wait**), el cual se trata del protocolo ARQ original. Consiste en que el emisor transmite un paquete cada vez y espera una confirmación, el receptor por su parte cuando recibe un paquete de forma correcta transmitirá un ACK, de modo que si en el emisor se recibe un ACK entenderá que el paquete ha llegado correctamente y continuará con la transmisión del siguiente paquete en el caso de que existiera. Si se diera el caso en el que se produce un error en algún momento del proceso, para que éste no se bloquee, el emisor dispone de un temporizador o timer que arrancará cada vez que se transmite un paquete y que si expira pasará el emisor automáticamente a la retransmisión del paquete en cuestión. El problema de este protocolo es que solo permite un frame en el link, por lo que generalmente el emisor se encuentra desocupado a la espera de un ACK y como consecuencia obtenemos una baja utilización el ancho de banda, siendo además un protocolo ineficaz si el retardo de propagación es mayor que el tiempo retransmisión de los paquetes.

Este problema de la mala utilización del ancho de banda se puede solucionar pudiendo transmitir un número finito de paquetes antes de esperar los ACK correspondientes. Esto se logra con protocolos de ventana deslizante tales como ARQ con retroceso N (**go-back n**), en donde n indica el número de paquetes que se podrá transmitir sin recibir una confirmación, es decir, n es el tamaño de la ventana o intervalo, dicha ventana irá avanzando a medida que se confirmen las recepciones de los paquetes anteriores, por otra parte el receptor sólo irá aceptando los paquetes de

forma ordenada. El proceso de este protocolo se desglosará más específicamente en el capítulo 3, no obstante podemos adelantar que el mayor problema del protocolo go-back n es que en caso de error en alguno de los paquetes, el protocolo reenviará toda la ventana, es decir, retransmite todos los paquetes que componen en ese momento la ventana independientemente de que el error solo se haya producido en un paquete en particular.

Este problema de tener que retransmitir la ventana completa, lo trata de solucionar el protocolo de repetición selectiva (**selective ARQ**), el cual será capaz de detectar el paquete dentro de la ventana en el que se ha producido el error y retransmitir únicamente dicho paquete.

El objetivo principal de este Proyecto Fin de Carrera es diseñar un simulador de los protocolo stop & wait, go-back n y selective ARQ en un applet Java, para que cualquier interesado en dicho protocolo sea capaz de comprender como funciona el protocolo de manera rápida, intuitiva y visual.

El objetivo principal se puede descomponer así mismo en objetivos menores:

- Crear una interfaz gráfica atractiva e intuitiva que facilite el uso y comprensión del simulador.
- Separar la simulación en dos partes: una más visual, que ayude a comprender el funcionamiento del protocolo paso a paso, y otra simulación que se ejecuta de modo automático y da una idea global de funcionamiento.
- Tener una aplicación gratuita y de código abierto que ayude a comprender mejor algunos de los aspectos de los protocolos ARQ. Al ser una aplicación de código abierto se puede añadir o modificar en función de futuras necesidades.
- El aprendizaje de la programación de Applets Java, ampliamente utilizados en WWW.
- La compresión de un protocolo de comunicaciones distribuido y su correcta programación mediante threads.

.

Capítulo 2

Nivel de enlace

La capa de enlace, que se sitúa inmediatamente encima de la capa física, se ocupa de suministrar un transporte de bits, normalmente fiable, a la capa de red. La capa de enlace solo se ocupa de equipos físicos y directamente conectados, sin tener conocimiento o 'conciencia' de la red en su conjunto.

Una característica importante de la capa de enlace es que los bits han de llegar a su destino en el mismo orden en que han salido; en algunos casos puede haber errores o pérdida de bits, pero nunca debe producirse una reordenación en el camino.

Las principales funciones que desarrolla la capa de enlace son las siguientes:

- Agrupar los bits en grupos discretos denominados tramas. Esto permite desarrollar de forma más eficiente el resto de funciones.

- Efectuar control de flujo, es decir pedir al emisor que baje el ritmo o deje momentáneamente de transmitir porque el receptor no es capaz de asimilarla información enviada.

- Realizar la comprobación de errores mediante el código elegido, que puede ser corrector o simplemente detector. En el caso de código corrector se procede a corregir los errores, en el de un código detector la trama errónea se descarta y opcionalmente se pide retransmisión al emisor.

No todas las funciones se implementan en todos los protocolos de enlace. La retransmisión de tramas erróneas y el control de flujo a menudo se implementan en las superiores (capa de red, de transporte, o incluso en la de aplicación).

La mayoría de las funciones del nivel de enlace se implementan en el hardware de los equipos. Esto hace que los protocolos de nivel de enlace se modifiquen poco con el tiempo.

Los tipos de servicio que la capa de enlace puede suministrar a la capa de red son normalmente los siguientes:

- Servicio no orientado a conexión y sin acuse de recibo
- Servicio no orientado a conexión con acuse de recibo
- Servicio orientado a conexión con acuse de recibo

En el primer caso el envío se hace 'a la buena de dios' sin esperar ninguna indicación del receptor sobre el éxito o fracaso de la operación. Este tipo de servicio es apropiado cuando la tasa de error es muy baja (redes locales o fibra óptica) y se deja la misión de comprobar la corrección de los datos transmitidos a las capas superiores (normalmente el nivel de transporte); se considera en estos casos que la probabilidad de error es tan baja que se pierde más tiempo haciendo comprobaciones inútiles que dejando esta tarea a las capas superiores. También se usa este tipo servicio cuando se quiere transmitir información en tiempo real (por ejemplo en una videoconferencia) y no se quiere sufrir el retraso que impondría un servicio más sofisticado en la capa de enlace (se supone que en este caso preferimos la pequeña tasa de error del medio físico a cambio de minimizar el retardo, o dicho de otro modo si se hiciera reenvío en caso de error sería peor el remedio que la enfermedad).

En el segundo tipo de servicio se produce un acuse de recibo para cada trama enviada. De esta manera el emisor puede estar seguro de que ha llegado. Suele utilizarse en redes con más tasa de error, por ejemplo redes inalámbricas.
El tercer servicio es el más seguro y sofisticado. El emisor y el receptor establecen una conexión explícita de antemano, las tramas a enviar se numeran y se aseguran ambos de que son recibidas todas correctamente en su destino y transmitidas a la capa de red una vez y sólo una.

En el servicio orientado a conexión se pueden distinguir tres fases: establecimiento de la conexión, envío de los datos, y terminación de la conexión. En la primera se establecen los contadores y buffers necesarios para la transmisión, en la segunda se envían los datos con las retransmisiones que sea preciso, y en la tercera se liberan los buffers y variables utilizadas.

2.1. Las Tramas.

La capa de enlace agrupa los bits en paquetes discretos denominados tramas (frames) que son los que envía por la línea. Según el tipo de red la trama puede oscilar entre unos pocos y unos miles de bytes. La utilización de tramas simplifica el proceso de detección y eventual corrección de errores. Una buena parte de las tareas de la capa de enlace tiene que ver con la construcción e identificación de las tramas. Para identificar el principio y final de una trama la capa de enlace puede usar varias técnicas; las más normales son:

- Contador de caracteres
- Caracteres indicadores de inicio y final con caracteres de relleno o "inserción de carácter"
- Bits indicadores de inicio y final, con bits de relleno o "inserción de bit"
- Violaciones de código a nivel físico

En el primer método se utiliza un campo en la cabecera de la trama para indicar el número de caracteres de ésta. Parece lo más sencillo e intuitivo, pero tiene un serio problema: si un error afecta precisamente a la parte de la trama que indica la longitud, o si por un error en la línea se envían bits de más o de menos, todas las tramas posteriores serán mal interpretadas.

El segundo método utiliza una secuencia especial de caracteres para marcar el inicio y final de cada trama, normalmente los caracteres ASCIIDLE STX para el inicio y DLE ETX para el final (DLE es Data Link Escape, STX es Start of Text y ETX End of Text). De esta forma si ocurre un error o incidente grave el receptor sólo tiene que esperar a la siguiente secuencia DLE STX o DLE ETX para saber en que punto se encuentra.

Cuando se usa este sistema para transmitir ficheros binarios es posible que por puro azar aparezcan en el fichero secuencias DLE STX o DLE ETX, lo cual provocaría la interpretación incorrecta de un principio o final de trama por parte del receptor. Para evitar esto se utiliza una técnica conocida como relleno de caracteres ('character stuffing' en inglés): el emisor cuando ve que ha de transmitir un carácter DLE que proviene de la capa de red intercala en la trama otro carácter DLE; el receptor, cuando recibe dos DLE seguidos, ya sabe que ha de quitar un DLE y pasar el otro a la capa de red.

El principal problema que tiene el uso de DLE STX y DLE ETX es su dependencia del código de caracteres ASCII. Este método no resulta adecuado para transmitir otros códigos, especialmente cuando la longitud de carácter no es de 8 bits. Para evitar estos problemas se ha diseñado una técnica que podríamos considerar una generalización de la anterior, consistente en utilizar una determinada secuencia de bits para indicar el inicio de una trama. Generalmente se utiliza para este fin la secuencia de bits 01111110, que se conoce como byte indicador ('flag byte' o 'flag pattern'). El receptor

está permanentemente analizando la trama que recibe buscando en ella la presencia de un flag byte, y en cuanto lo detecta sabe que ha ocurrido un inicio (o final) de trama. Aunque el flag byte tiene ocho bits el receptor no realiza el análisis byte a byte sino bit a bit, es decir la secuencia 01111110 podría suceder 'a caballo' entre dos bytes y el receptor la interpretaría como flag byte; esto permite el envío de tramas de longitud arbitraria.

Queda por resolver el problema de que los datos a transmitir contengan en sí mismos la secuencia 01111110; en este caso se utiliza la técnica conocida como relleno de bits o inserción de bit cero ('bit stuffing' o 'zero bit insertion'). Consiste en que el emisor, en cuanto detecta que el flujo de bits contiene cinco bits contiguos con valor 1, inserta automáticamente un bit con valor 0. El receptor por su parte realiza la función inversa: analiza el flujo de bits entrante y en cuanto detecta un 0 después de cinco unos contiguos lo suprime en la reconstrucción de la trama recibida. De esta forma la secuencia 01111110 no puede nunca aparecer como parte de los datos transmitidos más que como delimitador de tramas. Si las cosas van mal y el receptor pierde noción de donde se encuentra bastará con que se ponga a la escucha de la secuencia 01111110 que le indicará el inicio o final de una trama.

La trama a transmitir incluye casi siempre, además de los datos, alguna información de control de errores, por ejemplo un código CRC como veremos más adelante. Es importante notar que el relleno de bits (o de bytes) debe aplicarse a la trama inmediatamente antes de transmitirla y después de haber calculado el CRC, ya que de lo contrario la trama no sería interpretada correctamente si el CRC contuviera por azar el carácter o secuencia delimitadora de trama.

El cuarto método de identificación de tramas, violaciones de código a nivel físico, se utiliza en determinados tipos de red local aprovechando el hecho de que determinadas secuencias de símbolos no están permitidas y por tanto no pueden ocurrir en los datos a transmitir. Este método está muy relacionado con la codificación utilizada en el nivel físico. Por ejemplo, en el código Manchester se utilizan las combinaciones bajo-alto y alto-bajo para expresar al 1 y al 0, por lo que las combinaciones alto-alto y bajo-bajo están prohibidas. En este caso, podría utilizarse bajo-bajo como delimitador de inicio de trama, y alto-alto para indicar el fin de la misma.

2.2. Control de Flujo.

Cuando dos ordenadores se comunican generalmente han de adoptarse medidas para asegurar que el emisor no satura al receptor. Si la línea entre ellos es de baja capacidad probablemente el factor limitante será la conexión, pero si es una canal rápido (por ejemplo una red local) es posible que el emisor, si es un ordenador más rápido o está menos cargado que el receptor, envíe datos a un ritmo superior al que es capaz de similar éste. En este caso el nivel de enlace en el receptor utilizará los buffers que tenga disponibles para intentar no perder datos, pero si el ritmo acelerado sigue durante el tiempo suficiente se producirá antes o después una pérdida de tramas por desbordamiento. En estos casos es preciso habilitar mecanismos que permitan al receptor frenar al emisor, es decir ejercer control de flujo sobre él.

El control de flujo puede implementarse en el nivel de enlace o niveles superiores (por ejemplo en el nivel de transporte). Es importante que el control reflujo se ejerza de forma que no se produzcan ineficiencias en la comunicación; por ejemplo en enlaces de área extensa, donde la capacidad es un bien muy costoso, es importante mantener el nivel de ocupación del enlace tan alto como sea posible sin incurrir por ello en pérdida de tramas. Es decir, el control de flujo debe cumplir dos funciones:

- No perder tramas por el desbordamiento del receptor, y
- Optimizar el uso del canal, para lo cual necesitamos transmitir muchas tramas lo más rápido posible.

El control de flujo se implementa mediante dos tipos de protocolos:

- Los protocolos de parada y espera.
- Los protocolos de envío continúo con ventana deslizante.

2.3. Protocolos de enlace elementales

La principal característica que diferencia los protocolos de nivel de enlace es su comportamiento frente a errores. Cuando el receptor detecta una trama errónea puede hacer una de las dos cosas siguientes:

1. Descartar silenciosamente la trama errónea sin notificarlo a nadie

2. Solicitar del emisor la retransmisión de la trama errónea.

En el primer caso, es decir cuando no se realiza retransmisión de las tramas erróneas el protocolo de enlace es trivial, por lo que hay poco que decir. En el segundo caso, existen diferentes variantes de protocolos de enlace con retransmisión. Esto provoca lógicamente que al hablar de protocolos de nivel de enlace casi siempre se piense exclusivamente en los que realizan retransmisión de tramas erróneas. Paradójicamente este tipo de protocolos de enlace es hoy en día la excepción y no la regla. Dada la elevada fiabilidad de la mayoría de los medios físicos actuales normalmente no es rentable solicitar comprobación y retransmisión de las tramas, ya que, si la tasa de errores es baja, supondría realizar un proceso casi siempre inútil en cada nodo del trayecto. De esta forma, se deja que protocolos de capas superiores (normalmente el protocolo de transporte) soliciten la retransmisión en caso de error.

Por el contrario, en los casos en que la tasa de errores del medio físico es excesiva se prefiere incorporar en el nivel físico un mecanismo corrector de errores, lo cual se traduce en la práctica en un canal prácticamente libre de errores al nivel de enlace; esto es lo que ocurre por ejemplo en las comunicaciones por red conmutada vía módem gracias al estándar V.42, en las comunicaciones a través de redes GSM o en las transmisiones de televisión digital con el uso de códigos RS.

2.4. Posibles fuentes de error.

Las redes de comunicaciones de datos deben garantizar la fiabilidad de los mismos. Es aceptable, aunque no deseable, la posibilidad de que en una transmisión se produzcan errores. Lo que es del todo aceptable es que se produzca una conexión errónea y que ni el emisor ni el receptor lo detecten.

Los errores en las transmisiones son inevitables, pero existen técnicas que ayudan a paliar los problemas que se generan e incluso disminuir la tasa de error en la comunicación. Las posibles fuentes de errores son:

- El ruido térmico → la temperatura de un cuerpo, en nuestro caso un conductor o semiconductor, es la medida de agitación de los átomos. Cuando un electrón se mueve a través de la materia encuentra una cierta oposición o resistencia a su paso, que depende de varios factores. El primer factor es la naturaleza del material, de modo que un conductor opone poca resistencia y un aislante opone mucha resistencia al paso de la corriente electrónica. El segundo factor es que el movimiento de los átomos produce variaciones en los movimientos de los electrones debido a los choques que se producen entre unos y otros y generan corrientes eléctricas no controladas y aleatorias que llamamos ruido térmico.

- Ruido producido por componentes electromecánicos → algunos componentes electrónicos producen conmutación mecánica entre sus circuitos, estas conmutaciones no son instantáneas y generan picos de corrientes autoinducidas en los circuitos.

- Faltas de linealidad en los medios de transmisión → frecuentemente suele atenuarse la amplitud de una señal al transmitirse en altas frecuencias, lo que genera alteraciones no deseadas en la señal que, aunque son más fácilmente controladas, hay que tratara de evitar y corregir.

- Cruces entre líneas → cualquier corriente genera corrientes autoinducidas en los conductores próximos, cuando dos líneas de comunicaciones circulan paralelas, la información de una de ellas se puede autoinducir en la otra produciendo un cruce de líneas. Para evitarlo se pueden utilizar cables de pares trenzados que lleven cruzados la transmisión y la recepción, de modo que se anule el efecto de inducción de uno en otro. Son apropiados para evitar el efecto de autoinducción los cables UTP y STP.

- Falta de sincronismo → si el emisor y el receptor no están convenientemente sincronizados se pueden producir situaciones de error en la interpretación de los datos, aunque las transferencia haya sido correcta, por ejemplo, la trama se empieza a reconocer en un punto diferente del comienzo de la misma, Una trama es entregada a un destinatario que no le corresponde, etc.

- Eco → cuando no hay una adaptación perfecta en el interface entre dos líneas de transmisión se producen reflexiones de señal no deseadas que en ocasionan una interferencia.

- Atenuación → es la pérdida de señal debido a la resistencia eléctrica del medio de transmisión o da les maquinas de red.

2.5. Tratamiento de los errores.

Hay dos estrategias fundamentales para tratar los errores:

1. Incluir una cantidad de información redundante, junto con cada bloque de datos enviado, para permitirle al receptor deducir cuál fue el carácter que se transmitió.

2. Incluir suficiente redundancia para permitirle al receptor deducir que ocurrió un error, pero no que, tipo de error, y que tiene que solicitar una retransmisión.

La primera estrategia utiliza códigos correctores de errores, mientras que la otra emplea códigos detectores de errores.

El siguiente paso después de detectar un error ha de ser su corrección. La corrección de los errores se puede hacer por retransmisión, o bien en el destinatario siempre que junto a los datos se haya transmitido información redundante con esta finalidad, lo que se conoce en general como dígitos correctores de errores.

En nuestro caso, tratamos únicamente los errores corregidos por retransmisión, que es la estrategia que utilizan los protocolos ARQ.

Los mecanismos básicos utilizados para la recuperación ante fallos son tres:

1. Establecimientos de plazos de espera.

2. Solicitud de una nueva respuesta si vence dicho plazo.

3. Limitación del número de intentos, tras el cual el fallo se da por irrecuperable.

Capítulo 3

Protocolos ARQ

3.1. Protocolo Stop & Wait

Como caso más sencillo de protocolo de retransmisión es el denominado de parada y espera, consiste en que el emisor espera confirmación o acuse de recibo después de cada envío y antes de efectuar el siguiente. El acuse de recibo, también llamado ACK (del ingles acknowledgement) sirve tanto para indicar que la trama ha llegado correctamente como para indicar que se está en condiciones de recibir la siguiente, es decir el protocolo incorpora también la función de control de flujo. Este tipo de protocolos donde el emisor espera una confirmación o acuse de recibo para cada dato enviado se denominan protocolos PAR (Positive Acknowledgement with Retransmission) o también ARQ (Automatic Repeat reQuest).

El receptor verificará cada trama mediante alguno de los métodos de detección de errores (generalmente CRC) y en caso de que la trama recibida sea errónea no se producirá ACK. Lo mismo ocurre cuando la trama enviada se pierde por completo. En este caso, el emisor pasado un tiempo máximo de espera, reenvía la trama. Una optimización que se puede incorporar en el protocolo es el tiempo máximo de espera, reenvía la trama. Una optimización que se puede incorporar en el protocolo es el uso de acuse de recibo negativo o NAK (Negative Acknowledgement) cuando se recibe una trama errónea; de esta forma el emisor puede reenviar la trama sin esperar a agotar el tiempo de espera, con lo que se consigue una mayor utilización de la línea.

En el caso de que se pierda el mensaje del ACK, pasado el tiempo de espera el emisor concluirá erróneamente que la trama se ha perdido y la reenviará, llegando ésta duplicada al receptor. Como el receptor no tiene ningún mecanismo para detectar que la trama es un duplicado, pasará el duplicado al nivel de red, lo cual no esta permitido en un protocolo de enlace. Una forma de que el receptor distinga los duplicados es numerar las tramas, por ejemplo con un campo de un bit podemos numerar tramas en base 2 que es suficiente para detectar duplicados.

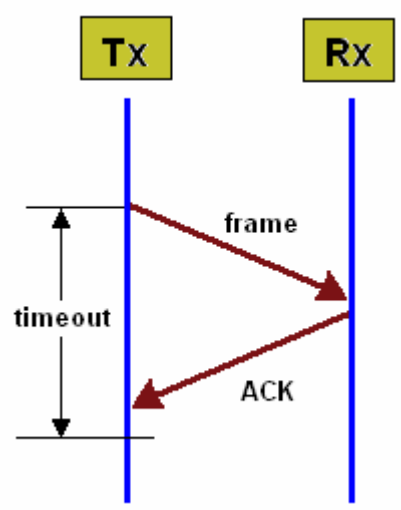
Aunque la transmisión de datos ocurra únicamente en un sentido, este protocolo requiere un canal dúplex o semi-dúplex para funcionar.

Ahora bien, este protocolo transmite datos en una sola dirección, el canal de retorno es utilizado únicamente para enviar los mensajes de acuse de recibo (ACK). Si tuviéramos que transmitir datos en ambas direcciones podríamos utilizar dos canales semi-dúplex con el protocolo anterior, pero nos encontraríamos enviando en cada sentido tramas de datos mezcladas con tramas ACK.

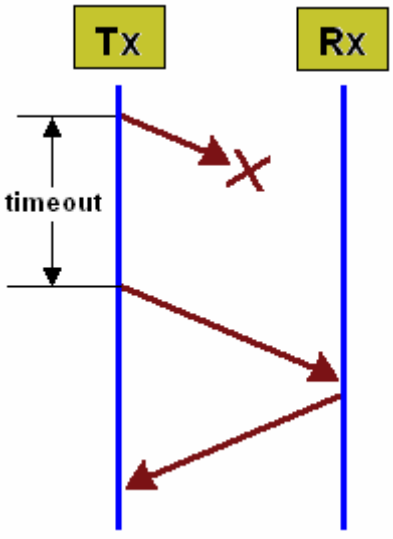
La trama ACK contiene una cantidad mínima de información útil, pero ha de contener una serie de campos de control imprescindibles que ocupan más bits que la propia información de ACK. Si se están transmitiendo datos en ambas direcciones resulta más eficiente, en vez de enviar el ACK solo en una trama, enviarlo dentro de una trama de datos; de esta forma el ACK viajará “casi gratis” y se ahorrara el envío de una trama. Esta técnica se conoce con el nombre de piggybacking o piggyback acknowledgement; (en inglés piggyback significa llevar a alguien a hombros o a cuestas).

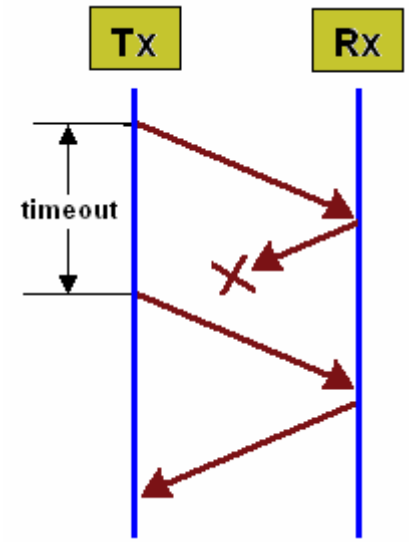
Ahora bien, para montar el ACK en una trama de datos es preciso que esta se envíe en un tiempo razonablemente corto con respecto a cuando debería enviarse el ACK; de lo contrario el emisor, al ver que el ACK esperado no llega reenviará la trama. Como no es posible saber de antemano cuando se va a enviar la siguiente trama de datos, la solución generalmente adoptada es esperar cierto tiempo (inferior a si tiempo máximo de espera) y si el nivel de red no genera ningún paquete en ese tiempo se genera una trama ACK. Ahora pasamos a ver sus características de forma más detallada:

En esta primera gráfica podemos ver el funcionamiento correcto del protocolo Stop & Wait.
El emisor transmite el frame, el cual llega al receptor que transmite el ACK correspondiente que llega al emisor antes de que expire el timeout.
La transmisión del frame se considera que ha llegado de forma correcta.

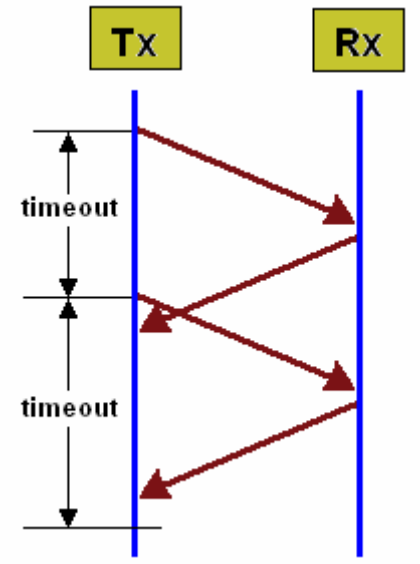


En este segundo caso se supone que se pierde el frame original, como al receptor no le llega ningún paquete no hace nada.
Pasado un tiempo el timeout del emisor expira y como no ha recibido ninguna confirmación del último paquete lo retransmite.





En este tercer caso, es el paquete del ACK el que se pierde. Como a causa de la pérdida el emisor no recibe ninguna confirmación, cuando expira el timeout lo retransmitirá.
Al receptor le llegará un paquete repetido por lo que uno de los dos será descartado en el propio receptor o en capas superiores para evitar duplicados.



En el cuarto caso, presento una situación en el que un timeout está mal programado.
Cuando se reciban dos ACK iguales se descartará uno de ellos, en el emisor ante dos paquetes iguales actuará igual que en el caso anterior.
El peligro de esta situación es que el timeout siempre esté mal configurado lo cual provocará que no se pueda realizar la comunicación y que el proceso se encuentre atrapado en un bucle.

Este protocolo hace necesario que los paquetes estén numerados ya que de esta forma es fácil determinar cuando un paquete está duplicado. Los números de secuencia deben estar en módulo 2. Esto funcionará correctamente siempre que:

- Las tramas viajen en orden (FCFS) por los enlaces
- El CRC detecte los errores siempre
- El sistema se inicia adecuadamente

Suponiendo que tenemos una condición inicial de SN=0, el algoritmo que ha de seguir el emisor será:

1. Acepta el paquete de la capa superior, si está disponible; le asigna un número SN

2. Transmite el paquete en una trama con número de secuencia SN

3. Espera una trama sin errores del receptor:

    i. si lo recibe y contiene RN>SN en el campo del número de petición, establece SN en RN y va a 1

    ii. si no lo recibe en un tiempo dado, va a 2

En este caso, el algoritmo del receptor será el siguiente:

1. Siempre que se recibe del emisor una trama sin errores con un número de secuencia igual a RN, se entrega el paquete recibido a una capa superior y se incrementa RN.

2. En tiempos arbitrarios, dentro del retardo delimitado tras recibir del emisor cualquier trama sin errores, se le transmite una trama con el RN en el campo del número de petición

Supongamos ahora que en la transmisión desde A (emisor) hasta B (receptor):

- Todos los errores se detectan como tales.

- Inicialmente no hay tramas en el enlace: SN=0 y RN=0

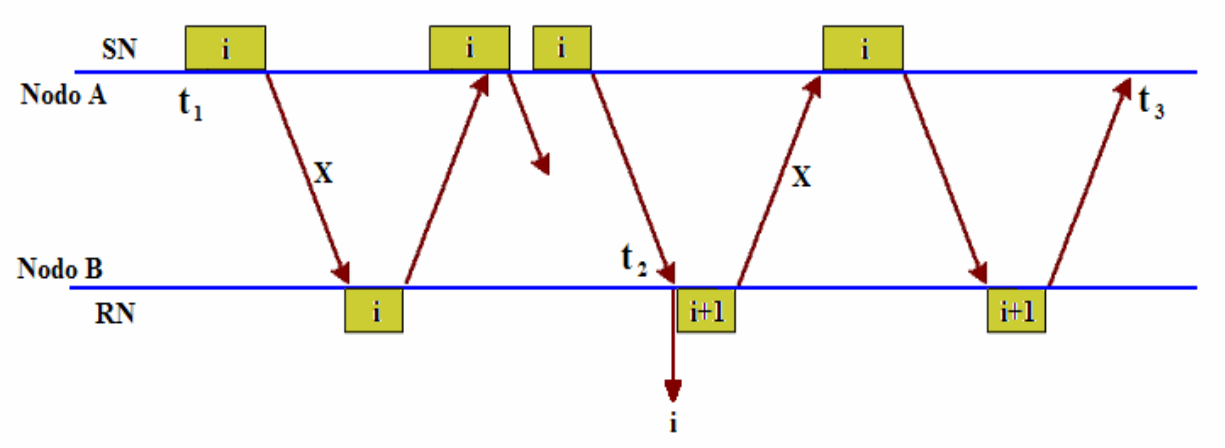
- Las tramas se pueden perder o retrasar arbitrariamente

- Cada trama se recibe correctamente con, al menos, cierta probabilidad q>0

Y ahora dividimos el proceso en dos partes, por un lado la seguridad y por otro el progreso:

- Respecto a la seguridad, ¿cómo podríamos asegurar que ningún paquete se ha entregado desordenado en más de una ocasión? Pues bien , inicialmente no hay tramas en el enlace, el paquete 0 es el primer paquete aceptado en A, el único al que se le asigna SN=0 y deberá ser el que libere B, si B llega a liberar un paquete Posteriormente (mediante inducción), si B ha liberado paquetes hasta n-1 (incluido), RN se actualiza a n cuando se entrega n-1 y, a continuación, solo se podrá liberar n

- Respecto al progreso,¿cómo sabemos que los paquetes se entregan tarde o temprano ? para ello atendemos a la siguiente gráfica:



t_1 = tiempo en que A comienza a transmitir el paquete i
t_2 = tiempo en que B recibe y libera i correctamente e incrementa RN a i+1
t_3 = tiempo en que SN se incrementa a i+1

Demostraremos que $t_1 < t_2 < t_3 < \infty$. => Progreso

Sean SN(t) y RN(t) valores de SN y RN en un tiempo t

Según el algoritmo:

1) SN(t) y RN(t) se incrementan en t y SN(t) ≤ RN(t) para todo t

2) Según la seguridad (dado que i no se envía antes que t_1) $RN(t_1) \leq i$ y $SN(t_1) = i$

De modo que:

- De (1) y (2), deducimos que $RN(t_1) = SN(t_1) = i$
- RN se incrementa en t_2 y SN en t_3, por lo que $t_2 < t_3$
- A transmite i repetidamente hasta t_3 y hasta t_2 cuando se recibe correctamente. Como q>0, t_2 es finito
- B transmite RN=i+1 una y otra vez hasta que se recibe correctamente en t_3; q>0 implica que t_3 es finito.

Supongamos que las tramas viajan ordenadas por el enlace. Obsérvese que con SN y RN enteros:

SN=RN	(de t_1 a t_2)	ó	(3)
SN=RN-1	(de t_2 a t_3)		(4)

Como las tramas viajan ordenadas, los números de secuencia que llegan a B y los números de petición que llegan a A aumentan, por lo que un solo bit puede resolver la ambigüedad entre (3) y (4)

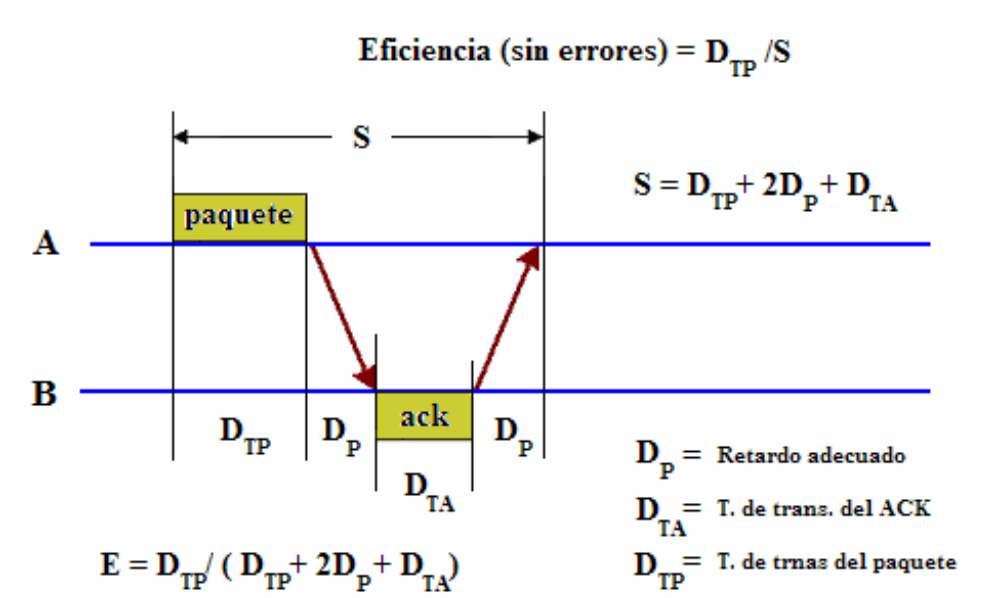
- RN = 0 y SN = 1 ó RN =1 y SN = 0 => el paquete recibido es un paquete antiguo

- RN = 0 y SN = 0 ó RN = 1 y SN = 1 => el paquete recibido es nuevo

Ahora estudiaremos la eficiencia del protocolo parada y espera:

Sea S = el tiempo total entre la transmisión de un paquete y la recepción de su confirmación (ACK)

D_{TP} = tiempo de transmisión del paquete



Sea P = la probabilidad de que se produzca un error en la transmisión de un paquete o en su confirmación

$$S = D_{TP} + 2D_P + D_{TA}$$

TO = el intervalo de tiempo de espera (timeout)
X = la cantidad de tiempo que lleva transmitir un paquete y recibir su confirmación. Este tiempo se tiene en cuenta en las retransmisiones debidas a errores

$$E[X] = S + TO*P/(1-P)$$
$$\text{Eficiencia} = DTP/E[X]$$

Donde:

TO = D_{TP} en un sistema Full duplex (bidireccional)
TO = S en un sistema Half duplex (de un solo sentido)

3.2. Protocolos de ventana deslizante

Los protocolos de parada y espera son sencillos de implementar pero tienden a ser poco eficientes, ya que producen tiempos muertos en la línea de transmisión, perdiendo su capacidad. La situación se agrava cuanto mayor sea el tiempo total de ida y vuelta de los mensajes (por ejemplo cuando se utilizan enlaces vía satélite)

La forma de aprovechar mejor los enlaces con elevados valores del tiempo de ida y vuelta es utilizar protocolos que permitan tener varias tramas en ruta por el canal de transmisión.

Al tener varias tramas pendientes de confirmación necesitamos un mecanismo que nos permita referirnos a cada una de ellas de manera no ambigua, ya que al recibir los ACK debemos saber a que trama se refieren. Para ello utilizamos un número de secuencia, que debe ser lo más pequeño posible, ya que va a aparecer en todas las tramas y los mensaje ACK.

Suponiendo un retardo nulo en el envío de los bits y en el proceso de las tramas en los respectivos sistemas, así como una longitud nula de las tramas ACK, el tamaño mínimo necesario W para poder llenar un canal de comunicación puede calcularse:

$$EW = 2t*v/n + 1$$

Debiendo rodearse el valor al entero siguiente por encima. En esta fórmula 2t es el tiempo (en segundos) que tarda una trama en hacer el viaje de ida y vuelta (round-trip time), v es la velocidad del canal de transmisión y n el tamaño de la trama a transmitir.

La suposición de que los tiempos de proceso y la longitud de las tramas ACK son despreciables no es correcta, por lo que en la práctica se consigue una mejora en el rendimiento incluso para valores de W bastante elevados.

Independientemente del tamaño de trama, velocidad de la línea y tiempo de ida y vuelta, un protocolo de parada y espera nunca puede conseguir un 100% de ocupación de una línea. Cuando se utiliza un protocolo de ventana deslizante con ventana mayor que uno el emisor no actúa de forma sincronizada con el receptor; cuando el receptor detecta una trama defectuosa puede haber varias posteriores ya en camino, que llegarán irremediablemente a él, aún cuando reporte el problema inmediatamente. Existen dos posibles estrategias en este caso:

- El receptor ignora las tramas recibidas a partir de la errónea (inclusive) y solicita al emisor retransmisión de todas las tramas a partir de la errónea. Esta técnica se llama retroceso n

- El receptor descarta la trama errónea y pide retransmisión de ésta, pero acepta las tramas posteriores que hayan llegado correctamente. Esto se conoce como repetición selectiva.

Todos los protocolos que forman parte de la familia llamados de “ventana deslizante” o sliding window tienen las siguientes características:

- Cada una de las tramas de salida está numerada secuencialmente, de modo que quedan unívocamente identificadas. Si el número de bits de la secuencia es mayor, será posible un mayor rendimiento del protocolo en ausencia de errores, si elegimos adecuadamente los parámetros de la transición.

- En cualquier instante de la transmisión el emisor posee una lista de los números de trama que ha enviado al receptor. Todas estas tramas constituyen lo que se denomina ventana emisora.

- Del mismo modo, el receptor posee una lista con las tramas que está dispuesto a aceptar del emisor, es decir, una ventana receptora. La ventana emisora de un emisor no tiene porque ser del mismo tamaño que la ventana receptora del receptor.

- El receptor aceptará la trama procedente del emisor si cae dentro de su ventana receptora; sin embargo, no importa el orden en que las tramas son enviadas, lo que proporciona más libertad a la capa de enlace para realizar la gestión de las tramas.

- Una vez iniciada la transmisión, la ventana emisora contiene los números de trama que se han enviado y de las que todavía no se ha recibido información. Cuando la capa de red suministra un paquete a la capa de enlace, ésta lo encapsula en una o varias tramas a las que le asigna los números de secuencia siguientes y son puestos en la ventana emisora, se envían al receptor y se espera en ese estado que lleguen las confirmaciones.

- Cada trama enviada es mantenida en un buffer de memoria en espera de la confirmación. Si ésta es positiva se libera el buffer, si es negativa o no se recibe ninguna confirmación, se efectúa la retransmisión. La ventana emisora contiene el número máximo de tramas que pueden ser enviadas al receptor sin necesidad de confirmación. Una vez que se ha superado ese número, el proceso de envío se detiene en espera de alguna confirmación que libere alguna trama enviada y pueda ser repuesta por un nuevo envío.

- De modo análogo, el receptor mantiene una ventana receptora con los números de secuencia de las tramas que es capaz de recibir. Si recibe una trama cuyo número de secuencia no está en su ventana receptora, es descartada sin más. Como el emisor nunca recibirá una confirmación de esta trama descartada, se tendrá que ocupar más delante de retransmitirla, una vez que hayan vencido sus temporizadores. Sólo las tramas recibidas dentro de su ventana receptora son pasadas a la capa de red, generando una confirmación que liberará una o más tramas de la ventana emisora en el emisor, lo que le dará a éste permiso para nuevos envíos.

- La ventana del receptor siempre tiene un tamaño constante a diferencia de la del emisor, que puede ir creciendo paulatinamente hasta llegar a un máximo fijado por el protocolo.

El protocolo de envío y espera es un protocolo de ventana deslizante, en el que tanto el tamaño de la ventana receptora como emisora es uno. De modo, que cuando el emisor envía la única trama que puede enviar, debe pararse hasta que el receptor le envíe la confirmación de que le llegó, liberando la trama enviada y dando paso a la siguiente.

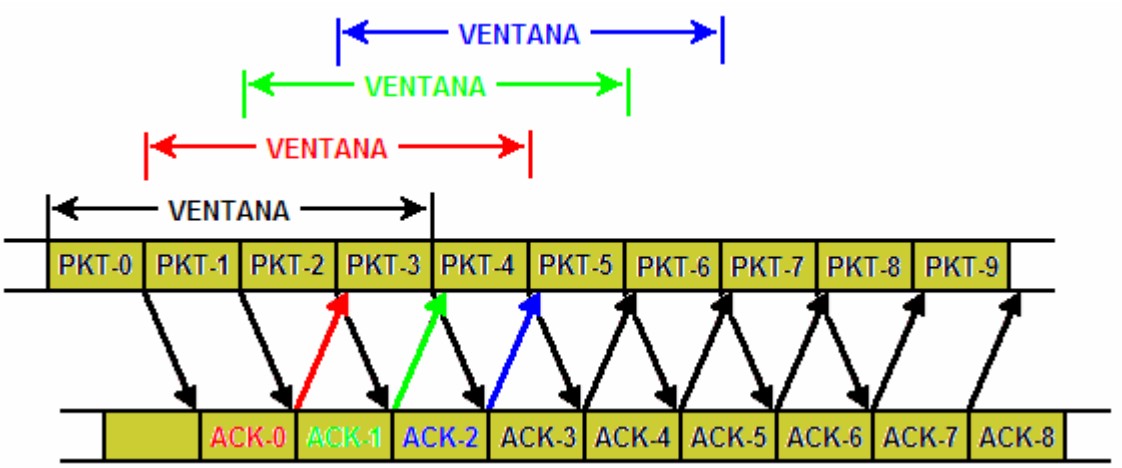
Pos su parte, el receptor puede recibir sólo una trama, concretamente la siguiente a la anterior en su número de secuencia. Si no recibe ésta entiende que ha habido alguna trama que se ha perdido y no confirma la trama afirmativamente, con lo que el emisor retransmitirá. Si la trama llegó correctamente genera una confirmación afirmativa para liberar de su espera al emisor y activar la transmisión de la siguiente trama.

Como en el resto de protocolos de envío y espera, cuando las ventanas son mayores que uno, las confirmaciones se pueden enviar una a una por cada trama recibida o por conjuntos de tramas. Del mismo modo, las retransmisiones pueden ser selectiva o no.

3.2.1. ARQ con retroceso n (go-back n)

El protocolo de parada y espera es ineficaz si el retardo de propagación es mayor que el tiempo de transmisión de los paquetes, el protocolo de retroceso N permite la transmisión de nuevos paquetes antes de que se confirmen los anteriores.

El protocolo retroceso N utiliza un mecanismo de ventana en el que se pueden enviar aquellos paquetes que se encuentren dentro de la ventana sin recibir confirmación, la ventana irá avanzando a medida que se vallan confirmando los paquetes anteriores.



N indica en tamaño de la ventana, de modo que el emisor no puede enviar el paquete i+N hasta que haya recibido confirmación del paquete i.

El receptor opera de igual forma que el protocolo parada y espera:

- Recibe los paquetes ordenados
- El receptor no puede aceptar paquetes fuera de la secuencia
- Se envía RN=i+1 □ confirma todos los paquetes hasta i (inclusive)

El emisor por su parte, tiene una "ventana" de N paquetes que puede enviar sin confirmación. Esta ventana abarca desde el último valor RN obtenido del receptor (llamado $SN_{mín}$) a $SN_{mín}$+N-1. Cuando el emisor llega al final de su ventana o finaliza el tiempo de espera, retrocede y retransmite el $SN_{mín}$.

Sea $SN_{mín}$ el paquete con menor número aún no confirmado y $SN_{máx}$ el número del siguiente paquete que se aceptará desde la capa superior (es decir, el siguiente paquete nuevo que se transmitirá)

Las reglas que ha de seguir el emisor son las siguientes:

Condición inicial : $SN_{mín} = 0$; $SN_{máx} = 0$

Repetir:

- Si $SN_{máx} < SN_{mín} + N$ (toda la ventana aún sin enviar)
  Enviar paquete $SN_{máx}$ y poner $SN_{máx} = SN_{máx} + 1$;

- Si el paquete llega del receptor con $RN > SN_{mín}$
  $SN_{mín} = RN$;

- Si $SN_{min} < SN_{max}$ (sigue habiendo paquetes sin confirmar) y el emisor no puede enviar paquetes nuevos
  Elegir algún paquete entre $SN_{mín}$ y $SN_{máx}$ y reenviarlo

La última regla dice que, cuando no es posible enviar paquetes nuevos, se debe reenviar un paquete antiguo (aún sin confirmar):

- Puede haber dos razones para no poder enviar un paquete nuevo:
  1. No se recibe ninguno nuevo de la capa superior
  2. La ventana ha caducado (SN máx. = SNmín + N)

- No hay ninguna regla para saber qué paquete se debe reenviar
  Enviar el menos reciente

Por otro lado las reglas que ha de seguir el receptor son las siguientes:

Condición inicial: RN = 0;

Repetir:

- Cuando llega un paquete correcto, si SN = RN:
  1. Aceptar el paquete
  2. Incrementar RN = RN +1

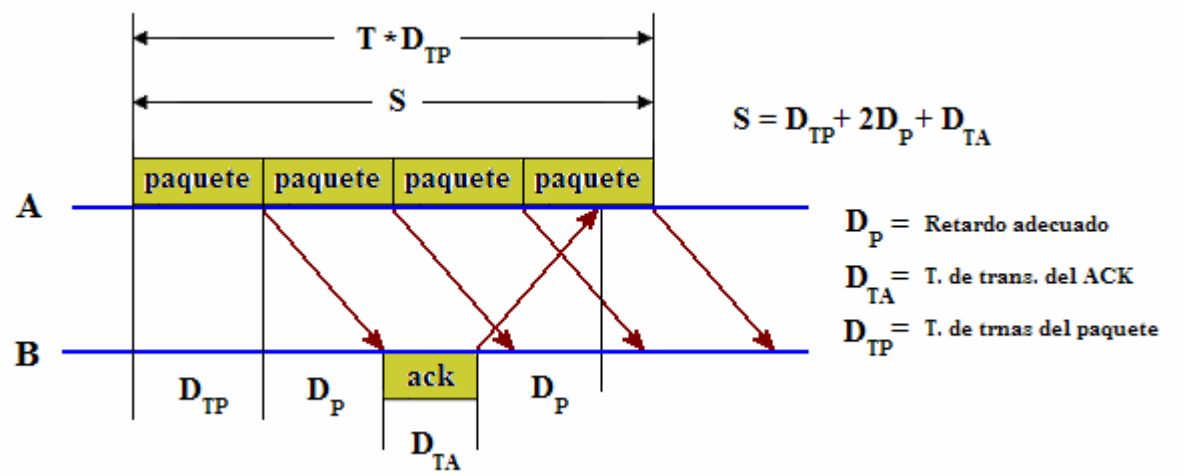
En intervalos regulares, enviar un paquete de confirmación con RN:

- La mayoría de los DLC envían una confirmación cada vez que reciben un paquete desde la otra dirección:
  Confirmación retardada para ser insertada en el siguiente mensaje de vuelta (piggybacking)

El receptor rechaza todos los paquetes con SN distinto de RN:

- No obstante, estos paquetes pueden seguir conteniendo números RN útiles

La eficiencia de este protocolo es:



Queremos elegir un N lo suficientemente grande como para permitir la transmisión continua mientras espera una confirmación para el primer paquete de la ventana:

$$N > S/ DTP$$

Sin errores, la eficacia del Retroceso N es:

$$E = \min.\{1, N*DTP/S\}$$

Suponiendo que:

$$N = \left\lceil \frac{S}{D_{TP}} \right\rceil \qquad TO = N*D_{TP}$$

Cuando se produce un error, hay que retransmitir toda la ventana de N paquetes. Sea X = el número de paquetes enviados por cada transmisión correcta

$$E[X] = 1*(1-P) + (X+N)*P = 1 + N*P/(1-P)$$

$$Eficacia = 1/E[X]$$

El funcionamiento del Retroceso N está garantizado, independientemente de qué paquetes se seleccionen para repetir, si:

1. El sistema se inicializa adecuadamente.
2. No se producen fallos en la detección de errores
3. Los paquetes viajan en orden FCFS
4. Existe una probabilidad positiva de recepción correcta
5. El emisor reenvía ocasionalmente $Sn_{mín}$ (p.ej., en el t. de espera)

6. El receptor envía el RN de vez en cuando

Por último para terminar este punto quiero resaltar las siguientes características sobre el protocolo go-back n:

- No requiere almacenamiento de paquetes en el buffer del receptor
- El emisor debe almacenar en el buffer hasta N paquetes mientras espera su confirmación
- El emisor debe reenviar toda la ventana en caso de error
- Los paquetes se pueden numerar en módulo M, donde M > N:
  - Porque se pueden enviar simultáneamente N paquetes como máximo
- El receptor sólo puede aceptar los paquetes en orden:
  - El receptor debe enviar los paquetes ordenados a la capa superior
  - No puede aceptar el paquete i+1 antes que el paquete i
  - Con esto se elimina la necesidad de almacenar en el buffer
  - Y se introduce la necesidad de reenviar toda la ventana en caso de error
- El mayor problema del Retroceso N es esta necesidad de reenviar toda la ventana en caso de error, lo cual se debe al hecho de que el receptor sólo pueda aceptar los paquetes ordenados

3.2.2. Protocolo de repetición selectiva (Selective ARQ)

La repetición selectiva intenta retransmitir únicamente los paquetes que se han perdido, debido a errores. Además El receptor debe poder aceptar paquetes desordenados, además debe poder almacenar en buffer algunos paquetes ya que los tiene que entregar a la capa superior los paquetes de forma ordenada.

Las peticiones de retransmisión pueden ser:

- Implícitas → El receptor confirma todos los paquetes correctos; los paquetes que no se han confirmado antes del tiempo de espera se dan por perdidos o con errores Obsérvese que se debe utilizar este enfoque para asegurarse de que finalmente se reciben todos los paquetes

- Explícitas → Un NAK explícito (rechazo selectivo) puede solicitar la retransmisión de un solo paquete. Este enfoque puede acelerar la retransmisión, pero no es estrictamente necesario

- En la práctica, se emplean uno o ambos enfoques

Las características de este protocolo son las siguientes:

- Protocolo de ventanas similar al del Retroceso N (Tamaño W de ventana).

- Los paquetes se numeran en módulo M, donde M >= 2W

- El emisor puede transmitir paquetes nuevos siempre que su número sea W en todos los paquetes sin confirmar

- El emisor retransmite los paquetes no confirmados tras un tiempo de espera o después de un NAK (en caso de utilizar NAK)

- El receptor confirma todos los paquetes correctos

- El receptor almacena los paquetes correctos hasta que puedan enviarse ordenados a la capa superior

- El emisor debe almacenar en el buffer todos los paquetes hasta que sean confirmados (es posible almacenar hasta W paquetes sin confirmar)

- El receptor debe almacenar los paquetes hasta que pueda enviarlos en orden, es decir, hasta que se hayan recibido todos los paquetes de menor número ya que debe enviar los paquetes ordenados a la capa superior
- Implicación del tamaño del buffer = W (mirar punto anterior)

    - Número de paquetes no confirmados en el emisor =< W en donde el buffer estará limitado en el emisor.

- El número de paquetes no confirmados en el emisor no puede diferir en más de W donde el Buffer estará limitado en el receptor (necesidad de enviar paquetes ordenados)

- Los paquetes deben numerarse en módulo M >= 2W (utilizando $\log_2(M)$ bits)

Ahora tratemos la eficacia de este protocolo:

En un protocolo de repetición selectiva ideal, sólo se retransmiten los paquetes con errores, lo cual, no es realista, ya que, en ocasiones, los paquetes se deben retransmitir porque la ventana ha caducado. No obstante, si el tamaño de la ventana es mucho mayor que el valor del tiempo de espera, que es poco probable.

Suponiendo un protocolo de repetición selectiva ideal y teniendo en cuenta que P es la probabilidad de un error en los paquetes:

$$\text{eficacia} = 1 - P$$

Obsérvese la diferencia con el Retroceso N, donde:

$$\text{eficacia (Retroceso N)} = 1/(1 + N*P/(1-P))$$

Cuando el tamaño de la ventana es pequeño el rendimiento es parecido, pero con una ventana grande, el protocolo de repetición selectiva es mucho mejor, ya que a medida que aumentan las tasas de transmisión, se necesitan ventanas más grandes y, en consecuencia, un mayor uso del protocolo de repetición selectiva.

Capítulo 4

Elección del entorno de desarrollo

Un IDE (Integrated Development Environment) es una aplicación que reúne varios programas necesarios para el desarrollador: editor, compilador, depurador, etc. Para la plataforma Java existen diversos entornos de desarrollo, entre los que cabe destacar:

RealJ

Programa: RealJ 3.7
Plataformas: Windows.
Versión: Beta January 2003

Entorno de Desarrollo Integrado en castellano, que integra la edición, compilación y ejecución de programas Java.

Es un programa sencillo y no tan potente como otros editores, no obstante decidí utilizar este IDE ya que este fue el primer IDE que aprendí a utilizar en la carrera, más específicamente en la asignatura Fundamentos de Programación de 2º curso de Ingeniería Técnica de Telecomunicaciones Especialidad en Telemática.

No obstante aunque el grueso de la aplicación lo he realizado a través de este programa, también he utilizado otros programas para aspectos más concretos que paso a comentar a continuación.

HTML Gate

Programa: HTML Gate FREE 12.2.1B.
Plataformas: Linux, Solaris, y Windows.
Versión: FreeWare.

Potente editor de HTML que te sorprenderá por la incorporación en el mismo programa una gran multitud de opciones añadidas que lo diferencian del resto.

Estos añadidos de los que te hablamos son javascripts, scripts de DHTML, VBScripts, XML, soporte para CSS, cliparts, compresor HTML, asistentes para ASP, filtros de todo tipo para Internet Explorer, GIFs animados, y mucho más...

Podrás llegar a hacer con este programa tus propias cookies, tus propios buscadores dentro de tu web. Incluye corrección de sintaxis a partir del cambio del color del texto. Incluye también herramientas básicas de tratamiento de imágenes.

Hay que destacar otras opciones que no son nada típicas de un editor web como el que nos encontramos, ya que incluye dentro de su abanico de funciones un cliente de correo electrónico, FTP, y hasta un editor para canales de Internet Explorer.

Además, incluye soporte para arrastrar y soltar ficheros en el cliente de FTP.

Este programa, lo he utilizado para crear la página web en la que se ha incluido el applet y otros enlaces de interés.

JBuilder

Borland®

Programa: JBuilder 2006.
Plataformas: Linux, Solaris, y Windows.
Versión: Evaluación, Personal (gratuita), Profesional y Enterprise (ambas de pago).

JBuilder Foundation está diseñado para desarrolladores Java que quieran una alta productividad IDE (Entorno de Desarrollo Integrado) para crear más fácilmente aplicaciones multiplataforma para Linux, Solaris, y Windows.

JBuilder Foundation permite desarrollar rápidamente, compilar, ejecutar, y encontrar errores, usando las aplicaciones visuales de JBuilder o con métodos tradicionales de código.

Adicionalmente, JBuilder permite que los usuarios retoquen a su gusto y extiendan el entorno según sus necesidades de desarrollo usando Open Tools API, el cual facilita la integración de otros componentes adicionales.

Tiene la gran utilidad de visualizar los diagramas UML, además de poder desarrollar aplicaciones web con ISP y servlets.

Programa ampliamente más potente que el RealJ y que lo he utilizado para aspectos como la búsqueda de métodos de clases concretas, realización de los diagramas uml, etc.

NetBeans

Programa: NetBeans 4.1.
Plataformas: Windows, Linux, Solaris SPARC, Solaris Intel, MAC OS X.
Versión: FreeWare.

NetBeans es una aplicación open source de desarrollo escrita en Java.
Obviamente, actualmente soporta el desarrollo en Java, pero su arquitectura le permite soportar otros lenguajes. Requiere J2SE JDK 1.4.2 o mayor.

Algunas de sus funciones y características son:

- Completado de código
- Soporte para escritura de servlets
- Ayudas con el código
- Ayuda on-line

Presenta soporte para el J2SE 5.0, además de que el usuario puede crear sus propios plug-ins utilizando las APIS de NetBeans.

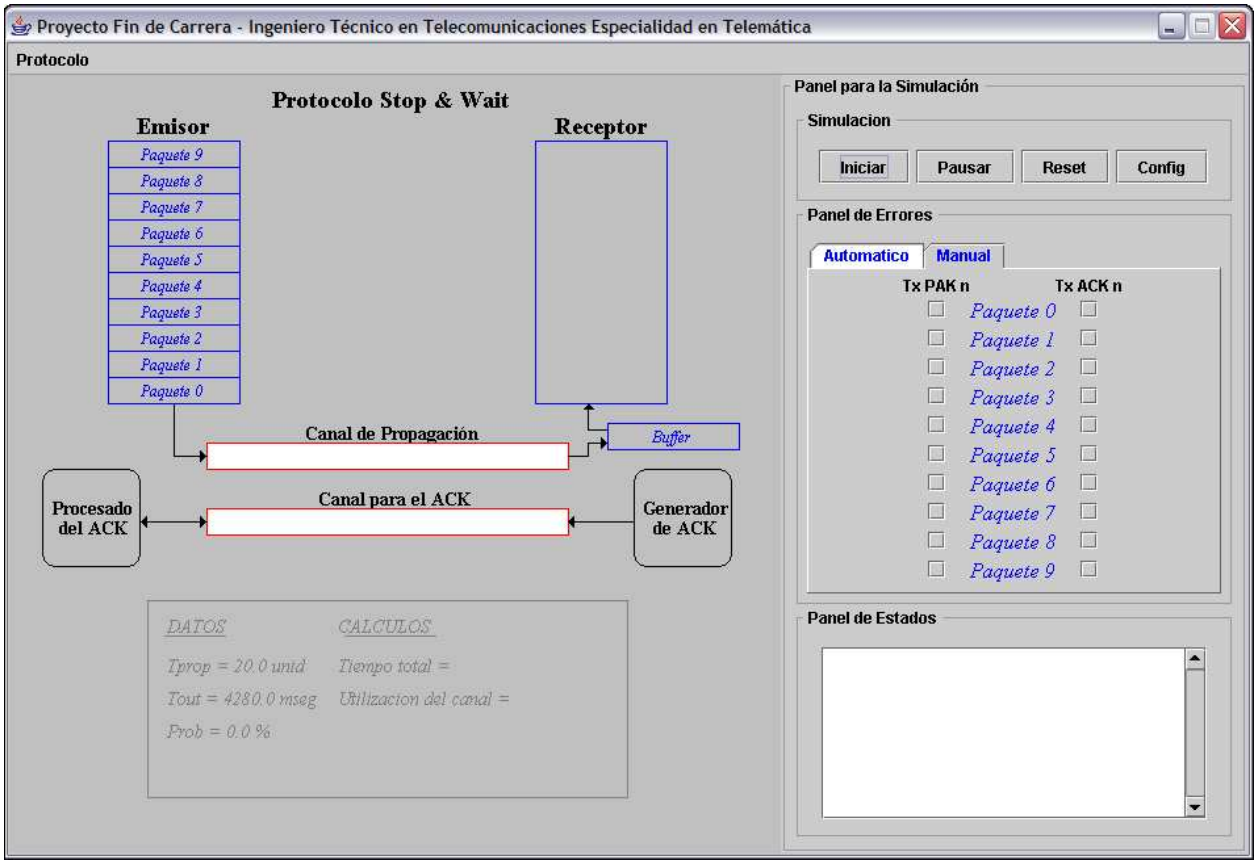
Programa que se utilizó en un principio para desarrollar el entorno gráfico, ya que ahorra al usuario escribir código tan sólo con arrastrar los componentes gráficos que soporta java y que están representados mediante iconos. Al añadir estos componentes, en la aplicación se genera el código correspondiente a dicha inclusión del elemento gráfico.

Capítulo 5

Descripción de la aplicación

El objetivo con el que se comenzó y se propuso este proyecto, era el de ayudar a la comprensión de aquellas personas y fundamentalmente a ingenieros en telecomunicaciones de los protocolos ARQ. De modo, que no sólo se trata de un simulador sino que se pretende que sea una herramienta didáctica que permita al usuario simular situaciones concretas y observar como actuaría el protocolo en cuestión.

El aspecto de la aplicación es el siguiente:



La aplicación está dividida en tres partes bien diferenciadas, un menú que es característico en cualquier aplicación, una parte gráfica y otra parte de control, y su vez la parte de control de control se divide en tres partes más que serán descritas en los siguientes puntos.

Antes de explicar cada una de las partes de la que forma la aplicación, hay que tener en cuenta las siguientes especificaciones que se tuvieron en cuenta:

- **El número de paquetes utilizado es 10**. Emisor y receptor se deben intercambiar un número finito de paquetes, se llegó a la conclusión de utilizar 10 paquetes que aunque no se corresponde generalmente con casos reales, si ajusta al objetivo fundamental del proyecto, es decir, obtener una simulación visual y eficiente del funcionamiento del protocolo.

- **Se utiliza un canal para la propagación del paquete y otro para el ack**. Esto lógicamente tampoco se correspondería con casos reales, pero al igual que el punto anterior lo que se pretende es que la simulación sea lo menos confusa posible, por ello los paquetes utilizarán un canal y una única dirección y el ack otro distinto.

- **Se relacionan los tiempos de propagación de transmisión con espacios dentro de la simulación**. En los protocolos ARQ hay dos parámetros que resultan muy importantes ya que a partir de ellos se obtendría la eficacia o utilización del protocolo.

  Estos parámetros son el tiempo de transmisión y el tiempo de propagación, entendiendo por tiempo de transmisión al tiempo que tarda el paquete en transmitirse al medio e incluso teniendo que diferenciar según sea el caso entre tiempo de transmisión en el emisor y tiempo de transmisión en el receptor, ya que no es lo mismo transmitir un paquete de un determinado tamaño que transmitir un ack cuyo tamaño en ínfimo ya que apenas contiene información.

  Por otro lado, el tiempo de propagación es el tiempo que tarda el paquete una vez que ha sido transmitido al medio en alcanzar al receptor, este tiempo generalmente se suele considerar el mismo tanto para el paquete como para el ack.

  Como he comentado anteriormente, el tiempo de transmisión y de propagación son dos valores a tener en cuenta, ya que gracias a ellos se puede calcular la utilización del protocolo en cuestión.

  Se entiende por utilización o eficacia del protocolo al porcentaje de tiempo que se encuentra ocupado el emisor durante la comunicación respecto del tiempo total que tarda todo el proceso.

  Hay que tener en cuenta que el emisor no se encuentra transmitiendo durante toda la comunicación, por ejemplo, supongamos el caso más sencillo utilizando el protocolo Stop&Wait. El emisor transmite un paquete, éste se propaga, el receptor lo recibe correctamente y transmite el ack correspondiente, el cual se propaga hasta el emisor que lo recibe, lo procesa reconociéndolo como el ack correcto y pasa a transmitir el siguiente paquete en el caso de que lo hubiera. Pues bien, en este proceso descrito el emisor únicamente ha intervenido en dos instantes del tiempo total, el primero cuando transmite el paquete al medio y el segundo cuando recibe e ack y lo procesa. El resto del

tiempo el emisor se encuentra bloqueado, en este caso, a la espera de recibir la confirmación. Por este motivo el factor de utilización o eficacia se obtiene respecto del emisor ya que es éste el que determina en función del tiempo si la transmisión esta siendo eficiente.

Lo más sencillo era tratar los tiempos como tales, es decir, si el usuario determina que el tiempo de transmisión debe ser α y el tiempo de propagación debe ser β, respetar dichos tiempos y que la simulación tarde exactamente lo que el usuario ha determinado.

Pero esto provocaría sobre todo dos inconvenientes debido fundamentalmente al gráfico usurado en la simulación.

El primer inconveniente, es que dar excesiva libertad al usuario podría provocar situaciones incoherentes, como por ejemplo, el introducir un valor en el tiempo de transmisión muy superior al de propagación. Este caso se podría dar bajo ciertas circunstancias en un caso real aunque es muy poco probable. No obstante esta situación, en el gráfico que se decidió para utilizar en el applet no se podría dar, la explicación sería la siguiente: el proceso en la simulación funciona de modo que cuando el paquete se está transmitiendo, éste va apareciendo en el canal de propagación, cuando este paquete se ha transmitido completamente se va desplazando por el canal de propagación hacia el receptor, al mismo tiempo que ocurre esto si un segundo paquete se empieza a transmitir, comenzará también a aparecer en el canal de propagación. De modo que en el caso en el que el tiempo de transmisión sea muy superior al de propagación se podrían producir situaciones en las que visualmente dos paquetes se solapen ya que si el primer paquete que ya se ha transmitido y que se comienza a desplazar por el canal muy lentamente mientras que el segundo paquete comienza a aparecer rápidamente, es evidente que como ambos están utilizando el mismo espacio, que el segundo paquete alcance al primero. Este efecto en la simulación provocaría visualmente circunstancias poco lógicas y en consecuencia no ayudaría en el entendimiento del protocolo.

Un segundo problema a tener en cuenta, es que si partimos de casos reales, el tiempo de propagación suele medirse en milésimas de segundo, segundos, etc., dependiendo de la distancia en la que se encuentran el emisor y el receptor y de la tecnología que se esté utilizando, en cambio el tiempo de transmisión depende del tamaño del paquete y de la tecnología, no obstante el tiempo de transmisión suele ser muy pequeño utilizando para ello milésimas de segundo. Por este motivo no es coherente en una aplicación que fundamentalmente es visual que se produzcan movimientos de milésimas de segundos.

Teniendo en cuenta estos dos problemas expuestos y alguno más pero no tan importantes como los anteriores, se llegó a la solución de en vez de utilizar periodos de tiempos utilizar espacios, es decir, la velocidad con la que se transmitirá y se propagará un paquete será siempre la misma, una velocidad constante y acorde para obtener comprensión eficiente de lo que está ocurriendo en cada momento.

El tiempo de propagación lo entenderemos de la siguiente forma: un tiempo de propagación implicaría que un paquete tardaría mucho en alcanzar su destino, mientras que un tiempo de propagación pequeño implicaría que el paquete alcanzaría el destino rápidamente. Bajo estas premisas y al utilizar una velocidad constante, el tiempo de propagación se visualizará de modo que para un valor pequeño el tamaño del paquete aumentará por lo que el espacio que el paquete debe recorrer una vez que ha sido transmitido completamente será menor que el de un paquete cuyo tamaño sea menor.

Pongamos como ejemplo el caso en el que el tiempo de propagación sea cero, esto en un caso real implica que tardaría cero segundos en propagarse desde el emisor hasta el receptor, en cambio en la aplicación como no se puede variar el tamaño del canal de propagación ni la velocidad de desplazamiento, este caso se representaría con un paquete cuyo tamaño sería el tamaño completo del canal de propagación, de modo que cuando el paquete se haya transmitido ya ocupara todo el canal y consecuentemente comenzará a recibirse en el receptor, es decir, debido al tamaño del paquete, éste no llega a desplazarse por el canal de propagación y por lo tanto se respeta lo que implica un tiempo de propagación de cero segundos.

Como último detalle referente a este punto y para cuadrar la simulación con respecto al tamaño en píxeles del grafico, se restringe los valores posibles del tiempo de propagación a un rango de 1 a 25 ambos inclusives.

Una vez explicada estas modificaciones, una cuestión a tener en cuenta es que si tomamos el tiempo de propagación como espacio y no como tiempo, cómo hallaríamos el factor de eficacia si como he comentado anteriormente el factor de eficacia se encuentra directamente ligado a los tiempos de transmisión y de propagación.

La respuesta sería la siguiente: se tiene en cuenta la propia definición de factor de utilización, el cual consistía en el porcentaje de tiempo que el emisor se encuentra ocupado transmitiendo respecto del tiempo total del proceso. Para ello utilizo dos contadores, ambos se incrementan a una misma velocidad pero, uno de ellos aumenta desde que comienza hasta que termina la simulación, mientras que el otro contador únicamente aumenta cuando el emisor se encuentre ocupado transmitiendo. De este modo al realizar el cociente entre ambos contadores, obtengo un factor de utilización muy aproximado del que se daría en cada caso y que servirá al usuario de forma intuitiva de cómo funciona cada protocolo en las circunstancias simuladas.

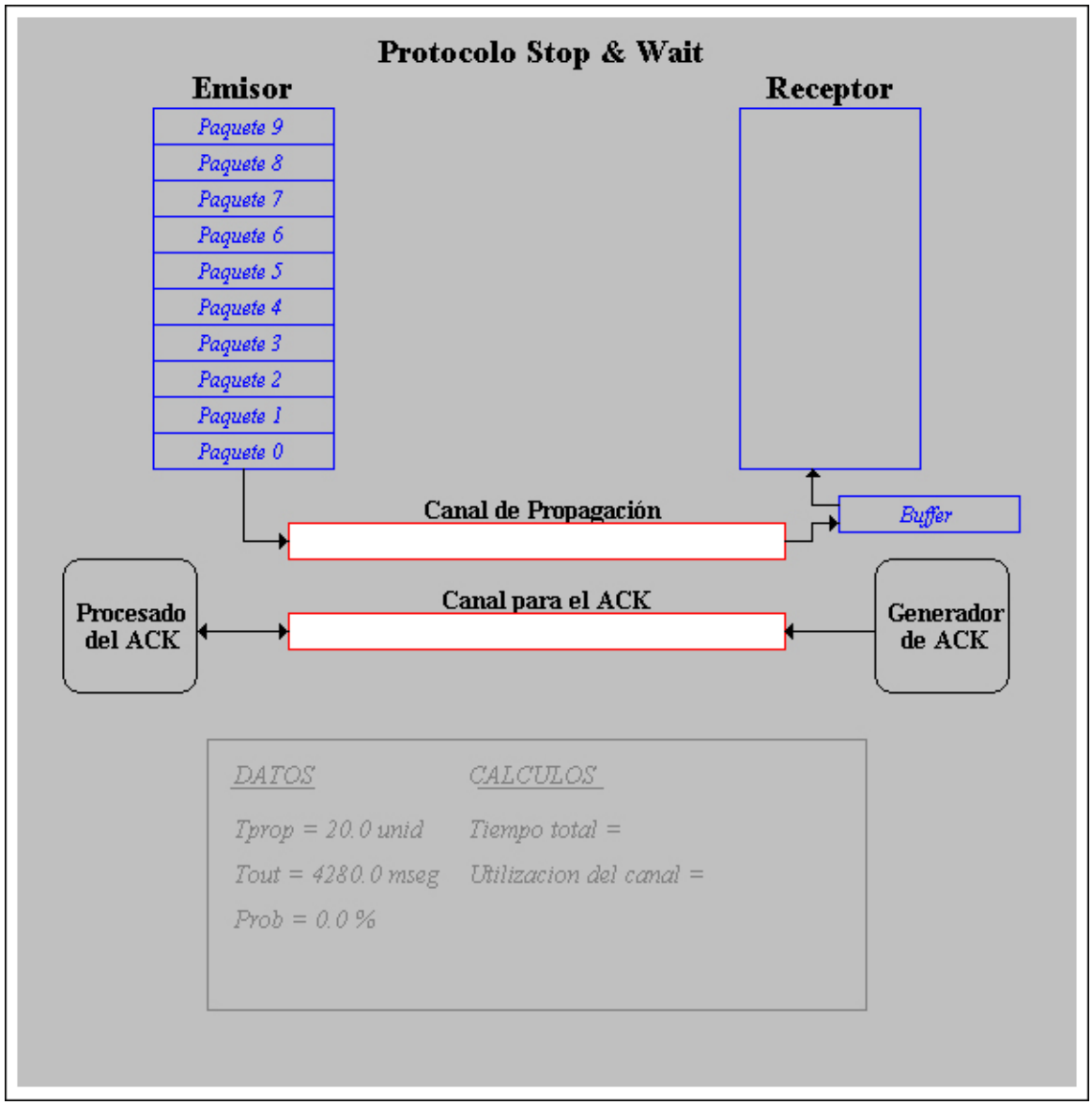
Una vez comentado los distintos aspectos a tener en cuenta, paso a comentar más detenidamente aquellas partes de las que forman parte la simulación.

5.1. Panel del Menú

Este panel nos permite una forma de ahorrar espacio y permitir al usuario elegir uno de los tres protocolos que se encuentran implementados.

En particular este menú utiliza tres JRadioButtonMenuItem y un JMenuItem. Los JRadioButtonMenuItem los utilizo para determinar el protocolo que se esta ejecutando, ya que estos tipos de botones para el menú son más intuitivos a la hora de visualizar, el JMenuItem lo utilizo para implementar la función de salir del programa y de cerrar la ventana sin necesidad de utilizar el botón de close de la ventana.

5.2. Panel Gráfico



El panel gráfico es una clase independiente de la clase principal y que extiende de JPanel. Esta clase utiliza para pintar el método público paint (Graphics g), y en función de g se pintan todos los elementos.

Para obtener el gráfico que se puede observar en la imagen, utilizo diferentes métodos, por ejemplo:

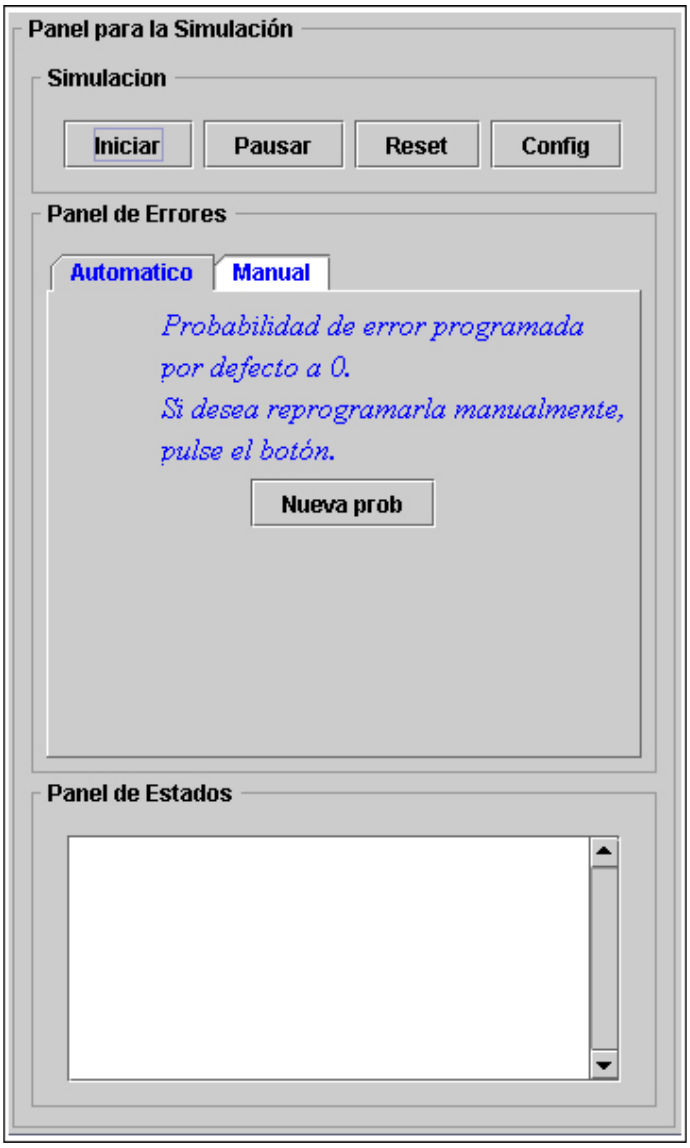
- Para el color → setColor("color a utilizar");
- Para pintar un línea → drawLine(x,y,posx,posy);
- Para utilizar una fuente determinada → setFont("la fuente a utilizar");

- Para escribir texto → drawString(“texto”, posx, posy);
- Para pintar un polígono relleno → fillPolygon(punto1,punto2,….,nº de puntos);
- Para pintar un rectángulo hueco → drawRect(x, y, anchura, altura);
- Para pintar un rectángulo no hueco → fillRect(x, y, anchura, altura);
- Etc.

Gracias a estos métodos y algunos más obtengo el gráfico deseado, el cual se diseñó de tal forma que resultase atractivo a la vista y sobre todo intuitivo y de fácil comprensión.

Además, en su parte inferior se podrá observar en cada momento los valores de los parámetros a tener en cuenta en la simulación, así como el tiempo total que ha tardado la simulación y el facto de utilización del canal, valores los cuales lógicamente se calcularán cuando finalice la simulación.

5.3. Panel de Control



El panel de control permite al usuario automatizar el simulador y lo transforma en una herramienta didáctica para su uso. Está programado como un BoxLayout y determino que los paneles que se le vallan introduciendo se ordenen en función de el eje y.

Este panel esta formado a su vez por otros tres paneles, los cuales y según la imagen de arriba a abajo son una botonera o panel para el control de la simulación, un panel para el control de la probabilidad y un tercer panel para registrar todos los eventos y estados por los que pasa la simulación.

A continuación paso a comentar cada uno de los paneles que forman este panel.

5.3.1. Panel para la simulación

Este panel contiene cuatro JButton, los cuales gracias a sus etiquetas resultan muy intuitivos y fáciles de usar.

El primer botón “Iniciar” se utiliza para dar comienzo la simulación y sólo estará activo cuando la simulación no haya sido iniciada anteriormente y esta se encuentre todavía en funcionamiento. De modo que una vez que se hayan establecido a priori los eventos y las acciones que se deseen, se pulsará el botón “Iniciar” y se visualizará la simulación en función a las normas del protocolo que se encuentre en ese momento activo.

El segundo botón “Pausar” sirve para detener la simulación una vez que ésta se encuentra activa. Esta utilidad es necesaria para poder entender ciertas situaciones que se estén produciendo sin que ocurran acciones nuevas y que solo aumentaría la confusión en el usuario. Además, para una mayor facilidad una vez que la simulación se ha pausado, la etiqueta de este botón pasa de “Pausar” a “Seguir”.

Una vez que la simulación arranca por primera vez, si el usuario no ha determinado errores ni ha variado los parámetros, la simulación arrancará en función de unos parámetros por defecto. Estos parámetros han sido elegidos de tal forma que se produzca una simulación lo más equilibrada en función a tamaños, velocidades de desplazamiento, etc. Una vez mencionado este aspecto, pasamos al tercer botón, el cual posee la etiqueta “reset” y que devuelve todos los parámetros a sus valores por defecto, es decir, resetea la simulación. Este botón podrá ser pulsado en cualquier momento y aplicación volverá a un estado igual al que había al iniciar el programa pero manteniendo activo el protocolo que hubiera en ese momento.

El cuarto y último botón, con la etiqueta “config” permite al usuario cambiar el valor de el tiempo de propagación. Recordar que este valor afectará al tamaño del paquete en la simulación y se utilizará a la hora de calcular el factor de utilización del protocolo. Una vez que se ha pulsado este botón, el programa informa que la simulación se reseteará automáticamente, de este modo se podrá nuevamente programar la simulación, la cual actuará utilizando el nuevo valor para el tiempo de propagación.

Para la reprogramación de este parámetro utilizo el componente javax.swing.JOptionPane que se puede emplear para obtener datos de entrada y mostrar mensajes de salida y más concretamente utilizo un JOptionPane.showMessageDialog para informar al usuario de que simulación se reseteará y un JOptionPane.showInputDialog para preguntar al usuario por el nuevo valor. En este punto también capturo posibles excepciones que se pueden dar al ofrecer a un usuario la libertad de escribir, de modo que si escribe un valor numérico

que no se encuentre en el rango [1,25] ambos inclusive, se entenderá que el usuario desconocía este rango por lo que se le informará a través de un JOptionPane.showMessageDialog de dicha acción y se le permitirá volver a introducir un nuevo parámetro para el tiempo de propagación. Si en cambio se diera el caso que el usuario introdujese otro tipo de caracteres no numéricos y le diera a aceptar la aplicación a través de un JOptionPane.showMessageDialog le informará que es acción no es aplicable y que se tomará para el tiempo de propagación su valor por defecto.

También debería mencionar que el rango elegido para los posibles valores del tiempo de propagación está elegido para favorecer la reconfiguración del nuevo tamaño del paquete en función de los píxeles que tiene el canal de propagación. Además una vez reprogramado este parámetro se podrá confirmar su nuevo valor en la parte inferior del panel gráfico.

5.3.2. Panel de Probabilidad

Este panel se utiliza para controlar la probabilidad de error en la simulación.

Este punto es fundamental a la hora de entender el objetivo del proyecto, es decir, además de ver como funcionan los diferentes protocolos es situaciones ideales que ayuda a tener una visión global de cada protocolo, se debe también ofrecer la posibilidad de que en la transmisiones hayan errores, y por tanto situaciones que son las que se acoplan a los casos reales. De modo que también se debe ver como actúa cada protocolo ante situaciones desfavorables para la transmisión, que nos enseñarán realmente qué protocolos utilizar según las condiciones para la transmisión.

Ahora bien, para que el usuario pueda configurar los errores, he determinado dos posibles maneras, cada una de éstas con un panel independiente en función de la manera que el usuario desee utilizar.

Esto lo consigo utilizando un JTabbedPane, es decir, un panel que está configurado a base de pestañas, de modo que en función de la pestaña que esté activa, el panel tendrá un aspecto y unas funciones específicas.

La primera posibilidad para determinar los errores es utilizar la forma automática, utilizando para ello el siguiente panel:

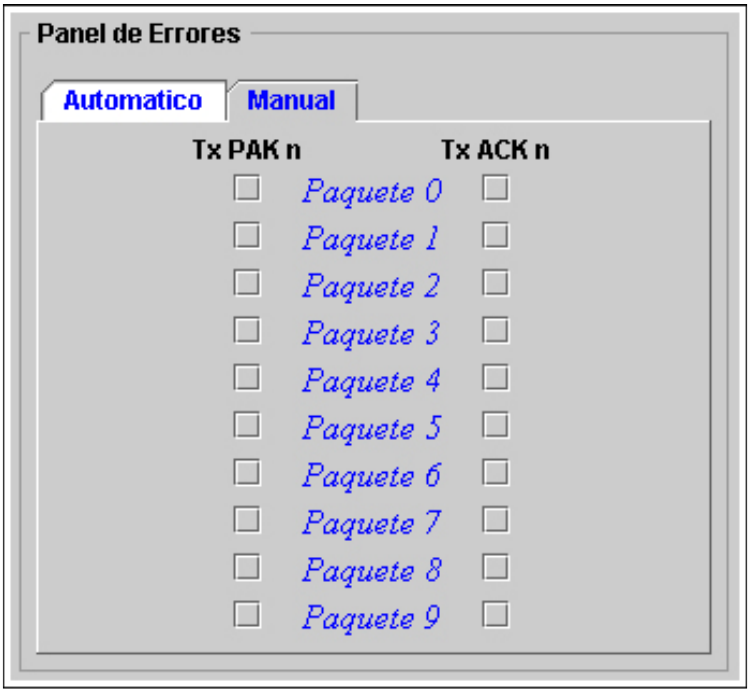
Este panel denominado panel automático, provoca errores en función del parámetro “probabilidad” que determinará la probabilidad de error que actuará sobre cada paquete.

Este panel esta configurado como BoxLayout e introduce los diferentes elementos en función de el eje y. gráficamente contiene cuatro líneas de texto que nos informa de que la probabilidad por defecto será cero (caso ideal) y un botón, el cual lo utilizaremos para introducir un nuevo valor para la probabilidad de error.

Al pulsar el botón se nos abre automáticamente un JOptionPane.showInputDialog preguntándonos por el nuevo valor de la probabilidad e informándonos que solo se admiten los valores comprendidos en el rango [0,1] ambos inclusives. De modo que una vez que se ha introducido un valor optimo, éste se multiplicará por 100 y se utilizará como porcentaje de error. Mencionar en este punto que se deben capturar las posibilidades en las que el usuario introduzca valores no posibles como números fuera del rango óptimo u otros caracteres no numéricos. Ante una situación así el simulador lanzará un showMessageDialog informando del error en el parámetro introducido e informando que se tomará el valor por defecto para este parámetro.

Supongamos que se ha pulsado el botón y se ha introducido un valor válido para la probabilidad, la aplicación actuaría de la siguiente manera: para una mayor comprensión utilizaré un ejemplo, supongamos que se introduce por teclado el valor 0.2 ó .2, la aplicación lo multiplicara por 100 y visualizará en el panel gráfico una probabilidad del 20%, ahora bien, internamente para cada paquete y para cada ack se calcula un número aleatorio entre 0 y 1, de modo que si un paquete n obtiene un valor aleatorio inferior a 0.2 se considerará que en ese paquete se producirá un error en la transmisión, por lo que repetiremos la probabilidad que se ha introducido. Este proceso se aplicará como he mencionado para cada paquete y para cada ack en particular, en el caso que en un paquete se haya producido error, en la retransmisión se volverá a calcular un valor aleatorio para ese paquete y se volverá a actuar de la misma manera.

La segunda forma de configurar los errores sería utilizar una forma manual, la cual se correspondería con el siguiente panel:



Este panel, también denominado panel manual esta formado principalmente por dos columnas de jCheckBox. Estos componentes son similares a los botones y muy visuales, de modo que se activan cuando son pulsados y visualmente quedan señalados.

Lo que se visualiza en este panel es lo siguiente: hay una primera columna formada por 10 jCheckBox encabezados por un texto “Tx PAK n”, una segunda columna formada únicamente por textos que nos indica el numero de paquete y una tercera columna formada también por 10 jCheckBox encabezados por el texto “tx ACK n”.

La idea es que el usuario pueda provocar un error en el paquete y en el momento que desee, de modo que para causar un error en el ack del paquete 4, el usuario tendrá que dirigirse al paquete 4 de la columna central y marcar el jCheckBox de la derecha que es el que se corresponde con los acks.

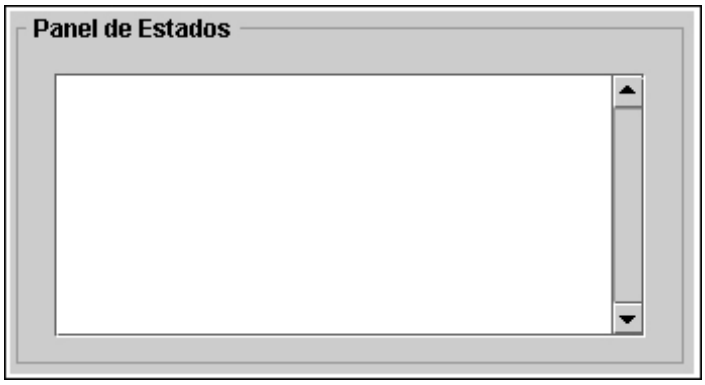
De este modo obtenemos un panel que a través del cual se podrán forzar situaciones concretas y que por lo tanto sabrá a priori que paquetes y en que momento se producirán errores. Esto es útil ya que ofrece al usuario gran libertad para provocar las situaciones que desee.

Cuando se ha decidido utilizar este panel y se han marcado aquellos jCheckBox que se han deseado, al pulsar botón para iniciar la simulación, todos los jCheckBox se ponen en forma no editable, esto ocurre para evitar posibles inconcluencias como la de activar nuevos JCheckBox durante una simulación ya iniciada y que podrían provocar errores de ejecución. Una vez que se ha finalizado la simulación o se ha pulsado el botón de reset, los jCheckBox volverán a estar activos.

Una vez que el jCheckBox de un paquete se ha activado y se ha producido el error, en la retransmisión se desactivará automáticamente el jCheckBox que provocó el error para poder continuar con la simulación.

El panel manual ha sido programado como GridBagLayout, ya que este tipo de panel de distribución nos permite actuar sobre él como si fuera una cuadrícula, en la que colocamos en cada lugar determinado el elemento que deseemos. Para los elementos he utilizado tres arrays, dos para contener los 10 jCheckBox cada uno, uno para la transmisión de los paquetes y el otro para los acks y un tercer array que contendrá únicamente JLabels, es decir, para los textos de paquete 1, paquete 2, etc. Insertando todos los elementos de los arrays en el GridBagLayout de la forma adecuada, obtenemos el aspecto de la imagen.

5.3.3. Panel de estados



Es el último panel que nos queda por comentar, es el panel más sencillo de programar pero que cuya importancia resulta evidente.

Este panel esta formado por un JTextArea configurado a un tamaño determinado, con un color blanco y sobre el cual no se podrá escribir directamente.

El panel de estados tiene una función informativa y aclarativa de lo que está o ha ocurrido en la simulación. Aunque en el simulador se ha utilizado una velocidad constante, colores variados, un gráfico determinado, etc., todo para facilitar el entendimiento visual, resulta evidente tener un campo en donde queden registrados todas las situaciones por las que pase el simulador. De modo que en este panel aparecerán todos los estados y eventos que vallan ocurriendo, ordenados temporalmente y utilizando una funcionalidad de los JTextArea, la cual consiste en que siempre se muestre la ultima línea que se ha escrito, dando a dicha línea más importancia y lógicamente facilitando la labor al usuario ya que no tendrá que estar pendiente de la barra de desplazamiento de este panel y poder así dedicar la mayor parte de atención al panel gráfico.

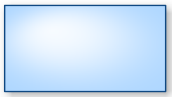
5.4. Flujogramas

Los diagramas de flujo son una manera de representar visualmente el flujo de datos a través de sistemas de tratamiento de información. Los diagramas de flujo describen que operaciones y en que secuencia se requieren para solucionar un problema dado.

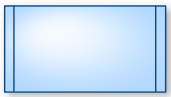
Un diagrama de flujo u organigrama es una representación diagramática que ilustra la secuencia de las operaciones que se realizarán para conseguir la solución de un problema, no obstante, en la programación estructurada las combinaciones de figuras permitidas son limitadas, pero la idea de estos diagramas es la de ofrecer una visón funcional del proceso. Por este motivo, no tendría sentido realizar el diagrama de flujo a partir de la ejecución del código, pero si resultaría útil hacerlo a partir de lo que ocurre cuando un usuario pulsa el botón “Iniciar”, ya que a partir de este punto la simulación se ejecuta según el protocolo.

Los Diagramas de flujo se dibujan generalmente usando algunos símbolos estándares, se muestran a continuación:

Inicio o fin del programa.



Pasos, procesos o líneas de instrucción de programa de cómputo.



Utilizado para representar métodos, es decir, que este símbolo lleva consigo un significado más abstracto que el caso anterior.

Toma de decisiones y Ramificación.

Conector para unir el flujo a otra parte del diagrama.

Como he mencionado anteriormente, mostraré un diagrama de bloques de cada uno de los protocolos para que se pueda tener una visión de cómo arrancan y se comunican los threads dentro de la simulación.

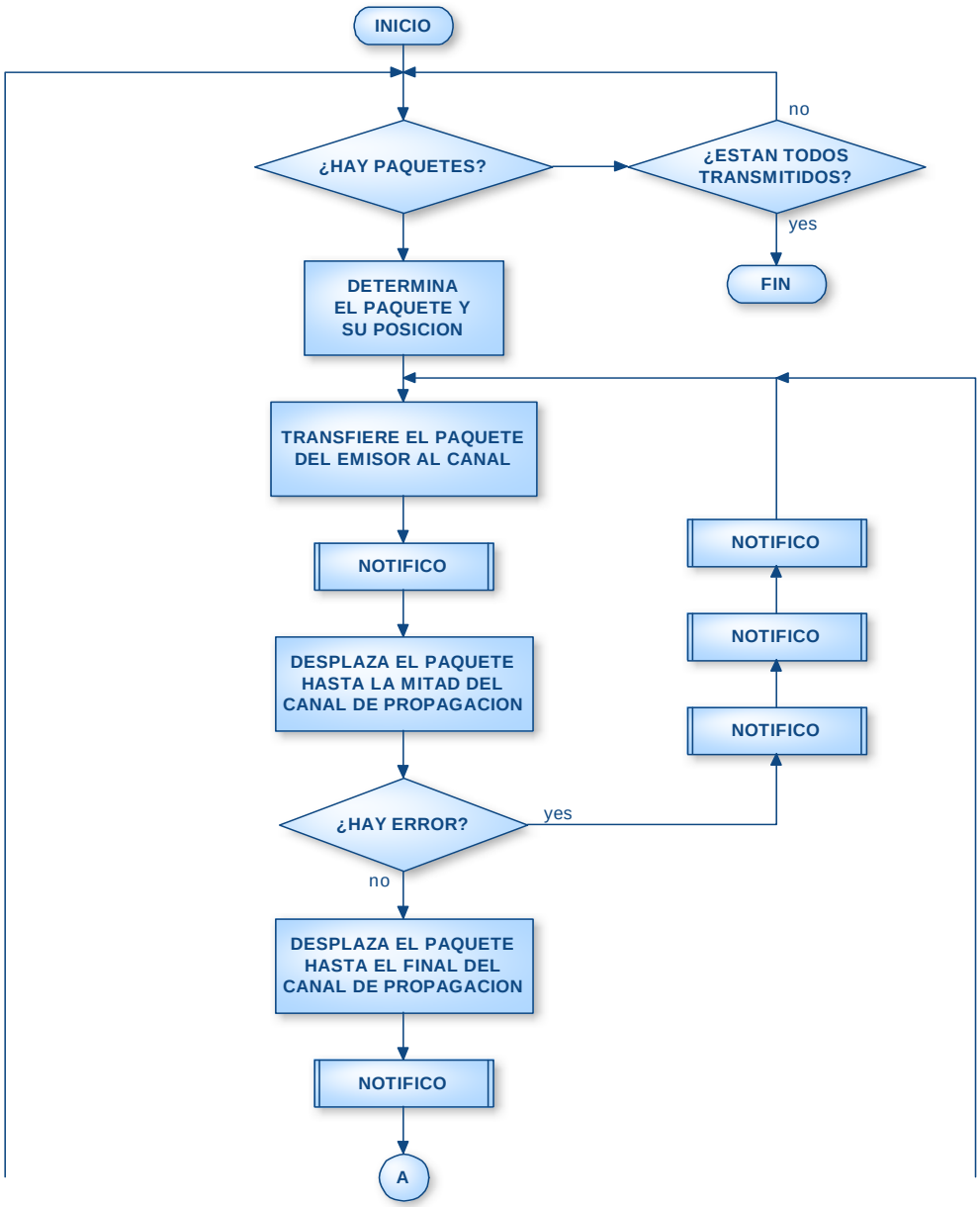
5.4.1. Flujograma del protocolo Stop&Wait

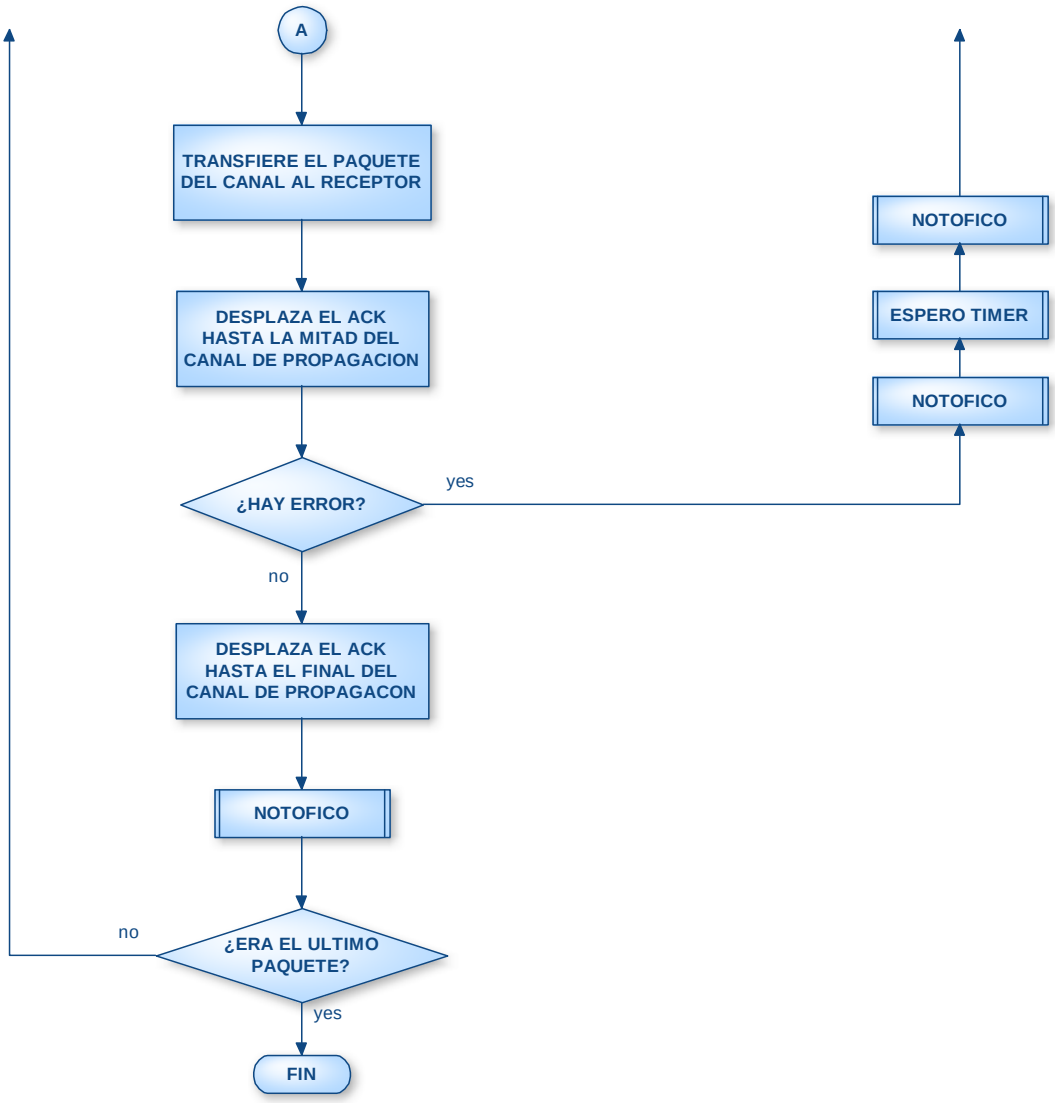
En este diagrama únicamente actúa un hilo el cual se detiene y se vuelve a arrancar en función de que queden o no paquetes por transmitir.
En este diagrama habría que explicar dos métodos que actúan:

1. Método “Notifico” → Esto implica un mensaje escrito en el panel de estados informando de algún evento que acaba de ocurrir.

2. Método “Espero timer” → este método bloquea la ejecución del hilo un determinado tiempo tras el cual se reanuda.

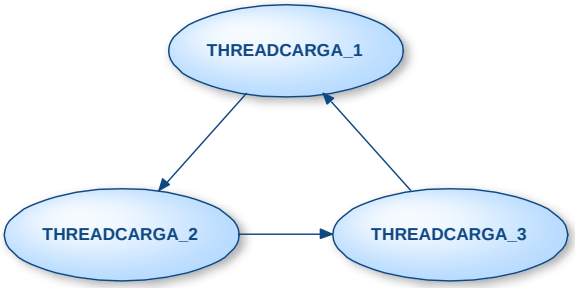
El diagrama de bloques correspondiente al protocolo Stop&Wait y que utiliza el hilo ThreadCarga es el siguiente:





5.4.2. Flujograma del protocolo Go-Back n

En este diagrama actúa sobre tres hilos, los cuales arrancan y se detienen dependiendo del hilo siguiente y el hilo anterior, es decir, una vez que ha arranca el primer hilo, este arrancará al segundo hilo siempre que se encuentra disponible y si hay más paquetes, el segundo hilo arrancará al tercer hilo si se encuentra disponible y si hay más paquetes y el tercer hilo volverá a arrancar el primer hilo si se encuentra disponible y si hay más paquetes. Este proceso se repetirá hasta que no queden paquetes.

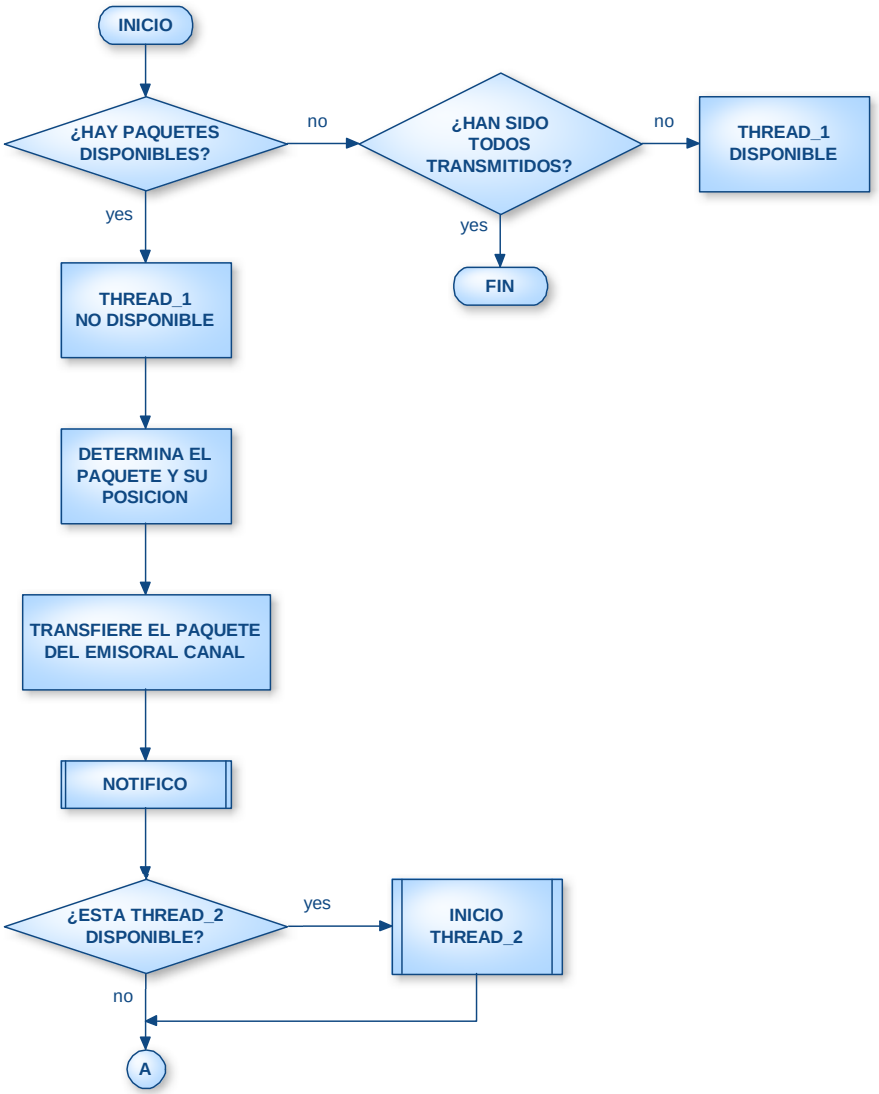


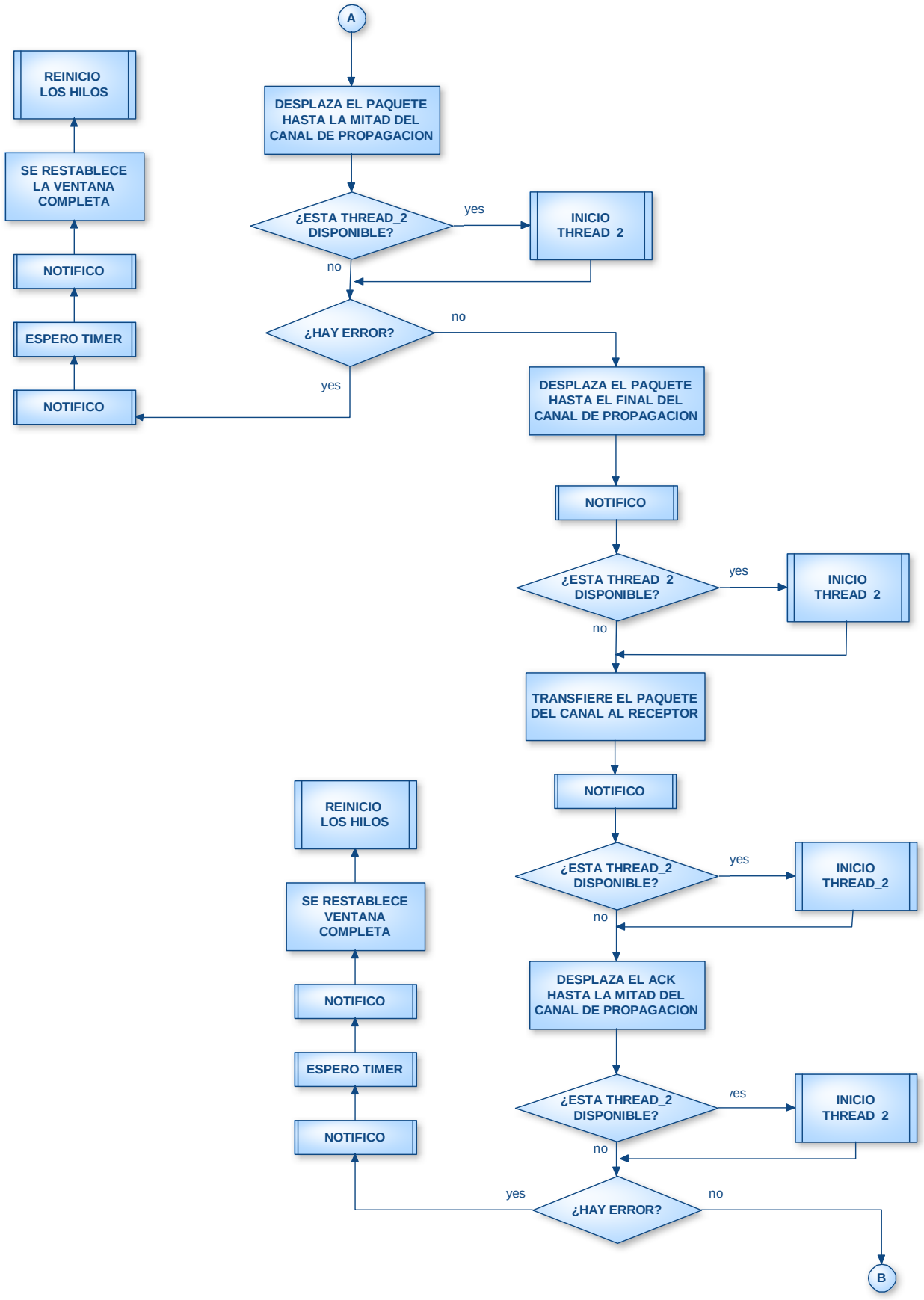
En este diagrama vuelven a aparecer los métodos “Notifico” y “Espero Timer” que funcionan igual que en el protocolo anterior y aparecen dos nuevos métodos:

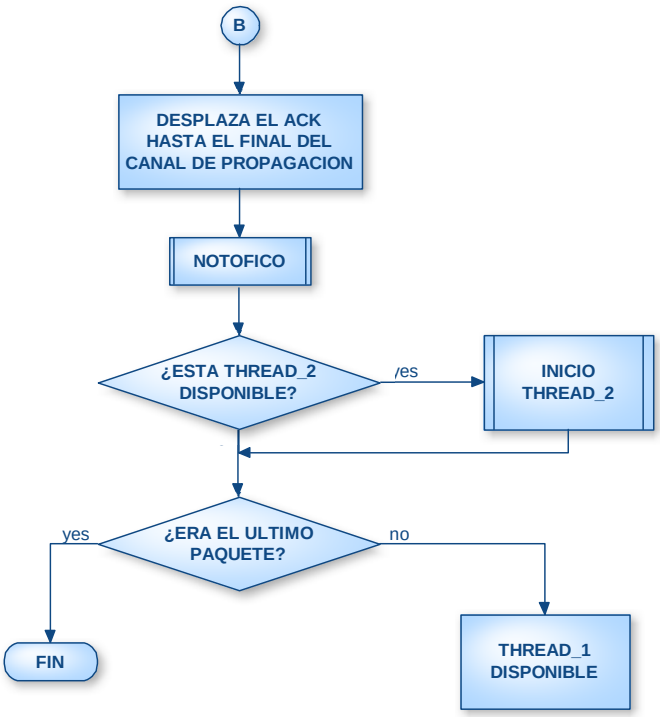
1. Método “Inicio Thread_2” → este método lo que nos indica es que en ese momento se inicia la ejecución de otro hilo, en este caso el thread_2.

2. Método “Reinicio Hilos” → este método sólo se utiliza cuando ha habido algún error, de modo que la simulación se encuentra detenida a consecuencia del timer, por lo que no habrá hilos ejecutándose hasta que transcurra dicho tiempo y entonces se tendrá que arrancar el primer hilo en el paquete que determine la ventana y el protocolo en ese momento.

El diagrama de bloques correspondiente al protocolo GoBack-n y que utilizan los hilos ThreadCarga1, ThreadCarga2 y ThreadCarga3, teniendo en cuenta que aunque sólo se representa para el ThreadCarga1, los correspondientes para el ThreadCarga2 y el ThreadCarga3 serían iguales cambiando la numeración de los threads en cada caso.

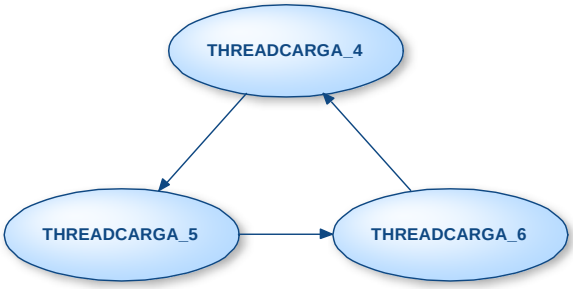



A
DESPLAZA EL PAQUETE HASTA LA MITAD DEL CANAL DE PROPAGACION
¿ESTA THREAD_2 DISPONIBLE?
yes
INICIO THREAD_2
no
¿HAY ERROR?
no
yes
REINICIO LOS HILOS
SE RESTABLECE LA VENTANA COMPLETA
NOTIFICO
ESPERO TIMER
NOTIFICO
DESPLAZA EL PAQUETE HASTA EL FINAL DEL CANAL DE PROPAGACION
NOTIFICO
¿ESTA THREAD_2 DISPONIBLE?
yes
INICIO THREAD_2
no
TRANSFIERE EL PAQUETE DEL CANAL AL RECEPTOR
NOTIFICO
¿ESTA THREAD_2 DISPONIBLE?
yes
INICIO THREAD_2
no
DESPLAZA EL ACK HASTA LA MITAD DEL CANAL DE PROPAGACION
¿ESTA THREAD_2 DISPONIBLE?
yes
INICIO THREAD_2
no
¿HAY ERROR?
yes
no
REINICIO LOS HILOS
SE RESTABLECE VENTANA COMPLETA
NOTIFICO
ESPERO TIMER
NOTIFICO
B

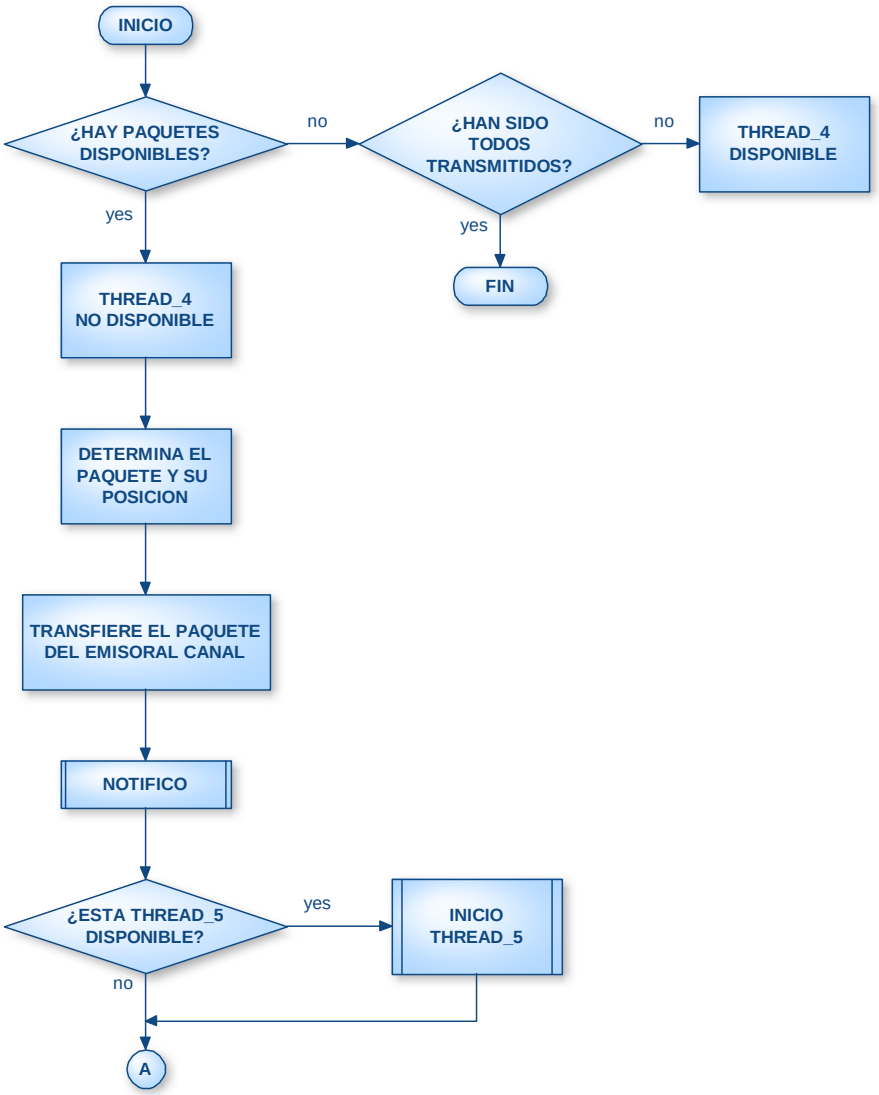


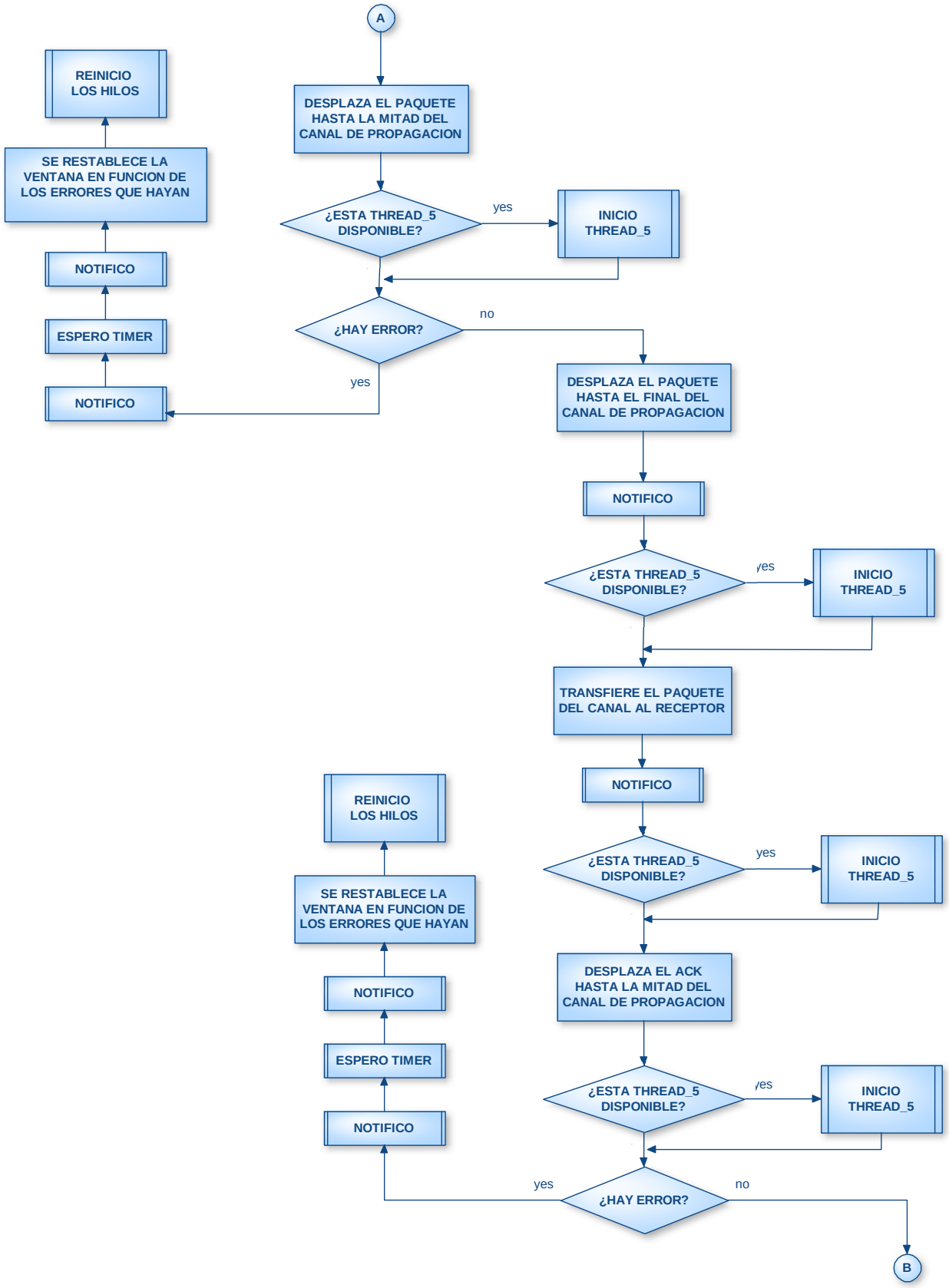
5.4.3. Flujograma del protocolo Selective ARQ

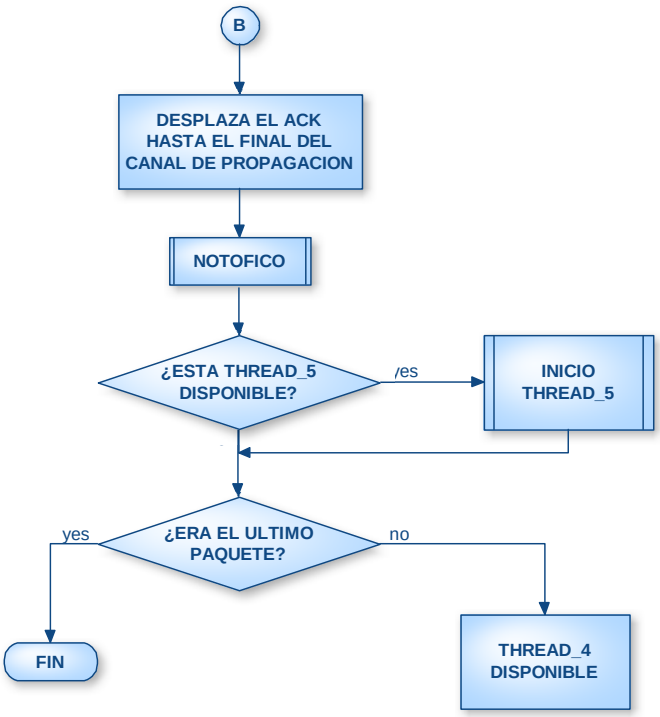
En este diagrama al igual que en el caso anterior actúa sobre tres hilos, los cuales arrancan y se detienen dependiendo del hilo siguiente y de el hilo anterior, es decir, una vez que ha arranca el primer hilo, éste arrancará el segundo hilo siempre y cuando se encuentra disponible y haya más paquetes, el segundo hilo arrancará el tercer hilo si se encuentra disponible y si hay más paquetes y el tercer hilo volverá a arrancar el primer hilo si se encuentra disponible y si hay más paquetes. Este proceso se repetirá hasta que no queden paquetes.



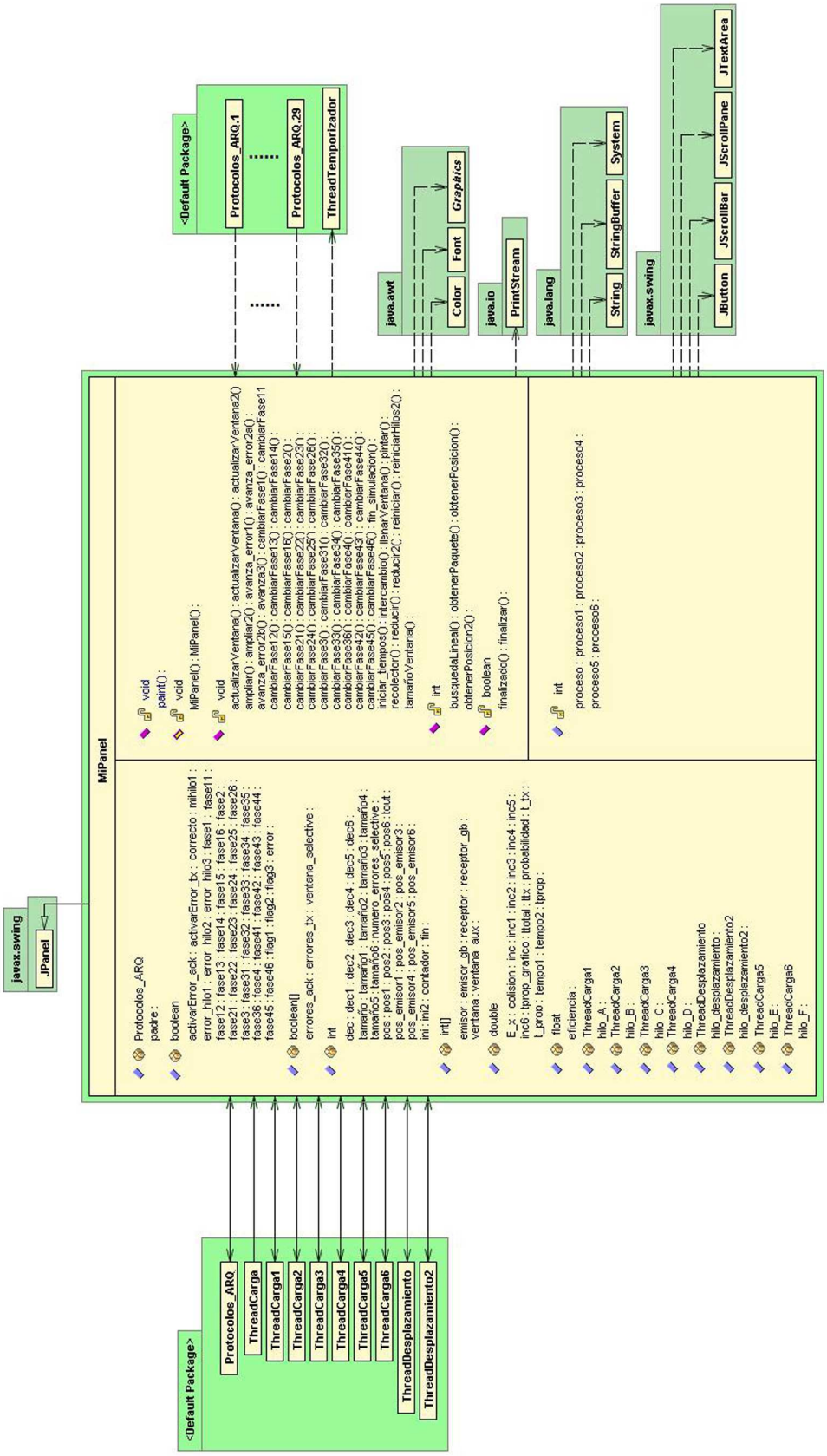
El diagrama de bloques correspondiente al protocolo GoBack-n y que utilizan los hilos ThreadCarga4, ThreadCarga5 y ThreadCarga6, teniendo en cuenta que aunque sólo se representa para el ThreadCarga4, los correspondientes para el ThreadCarga5 y el ThreadCarga6 serían iguales cambiando la numeración de los threads en cada caso.

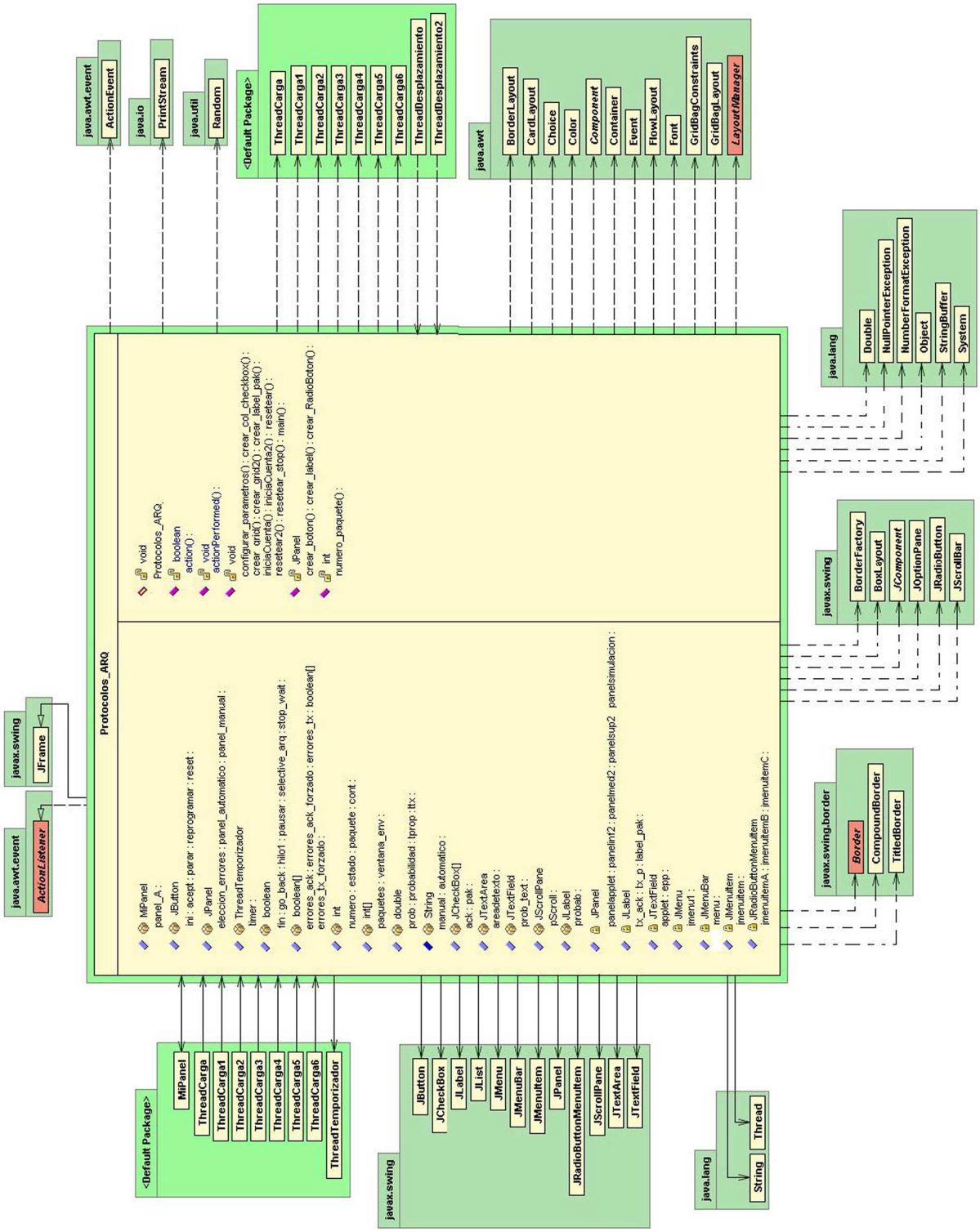


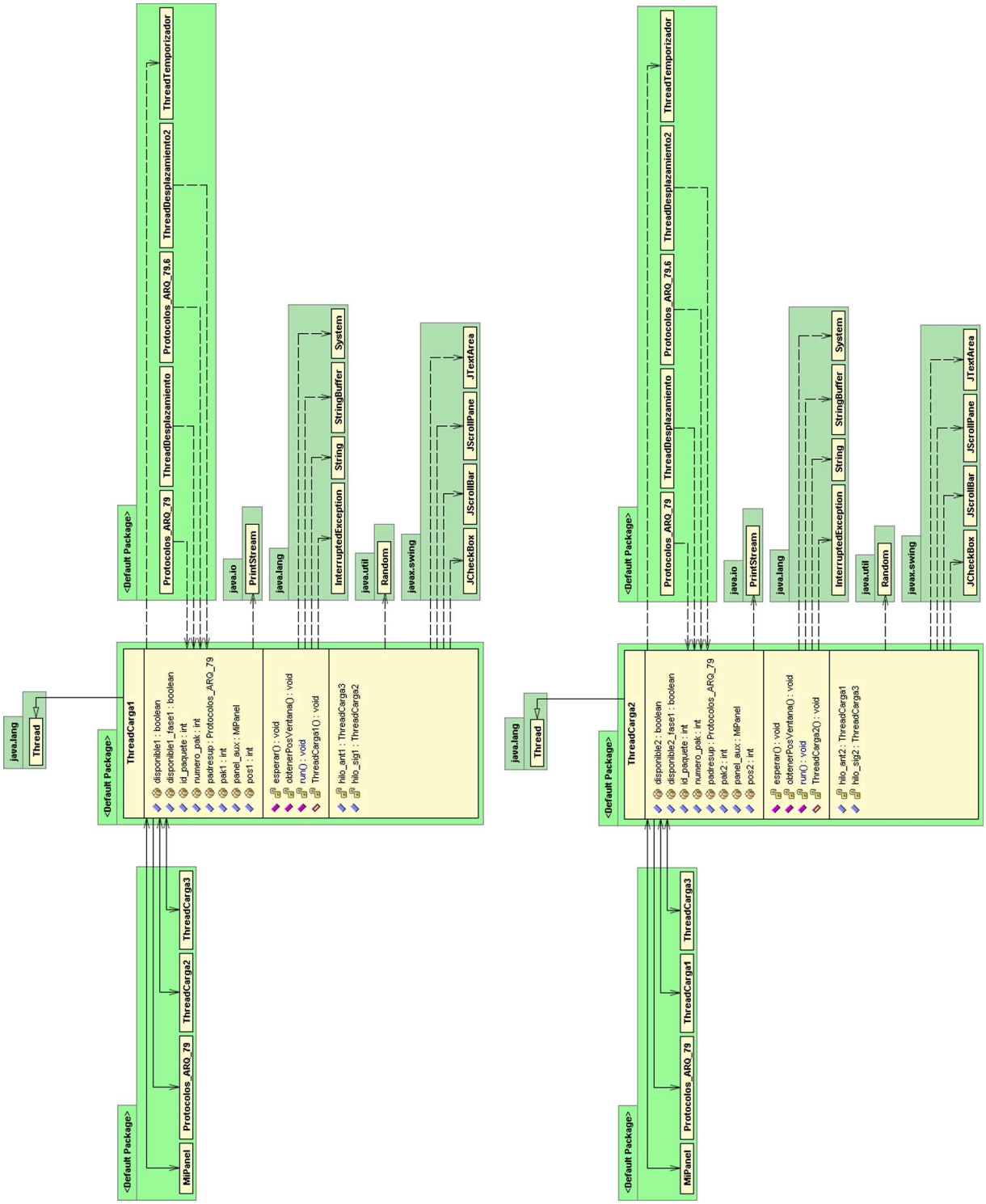

A
DESPLAZA EL PAQUETE HASTA LA MITAD DEL CANAL DE PROPAGACION
¿ESTA THREAD_5 DISPONIBLE?
yes
INICIO THREAD_5
¿HAY ERROR?
no
yes
NOTIFICO
ESPERO TIMER
NOTIFICO
SE RESTABLECE LA VENTANA EN FUNCION DE LOS ERRORES QUE HAYAN
REINICIO LOS HILOS
DESPLAZA EL PAQUETE HASTA EL FINAL DEL CANAL DE PROPAGACION
NOTIFICO
¿ESTA THREAD_5 DISPONIBLE?
yes
INICIO THREAD_5
TRANSFIERE EL PAQUETE DEL CANAL AL RECEPTOR
NOTIFICO
¿ESTA THREAD_5 DISPONIBLE?
yes
INICIO THREAD_5
DESPLAZA EL ACK HASTA LA MITAD DEL CANAL DE PROPAGACION
¿ESTA THREAD_5 DISPONIBLE?
yes
INICIO THREAD_5
¿HAY ERROR?
yes
no
NOTIFICO
ESPERO TIMER
NOTIFICO
SE RESTABLECE LA VENTANA EN FUNCION DE LOS ERRORES QUE HAYAN
REINICIO LOS HILOS
B

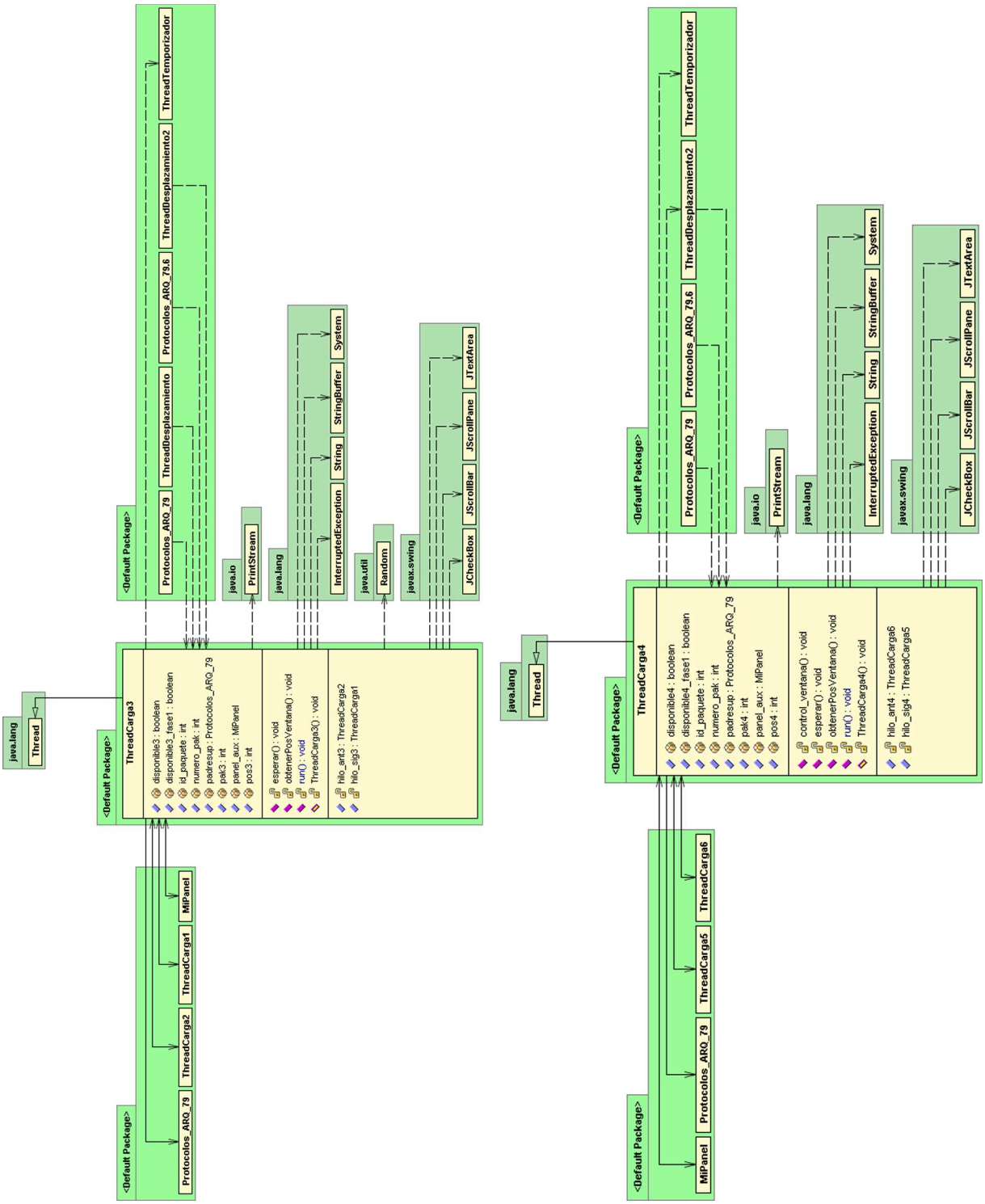

B
DESPLAZA EL ACK HASTA EL FINAL DEL CANAL DE PROPAGACION
NOTOFICO
¿ESTA THREAD_5 DISPONIBLE?
yes
INICIO THREAD_5
yes
¿ERA EL ULTIMO PAQUETE?
no
FIN
THREAD_4 DISPONIBLE

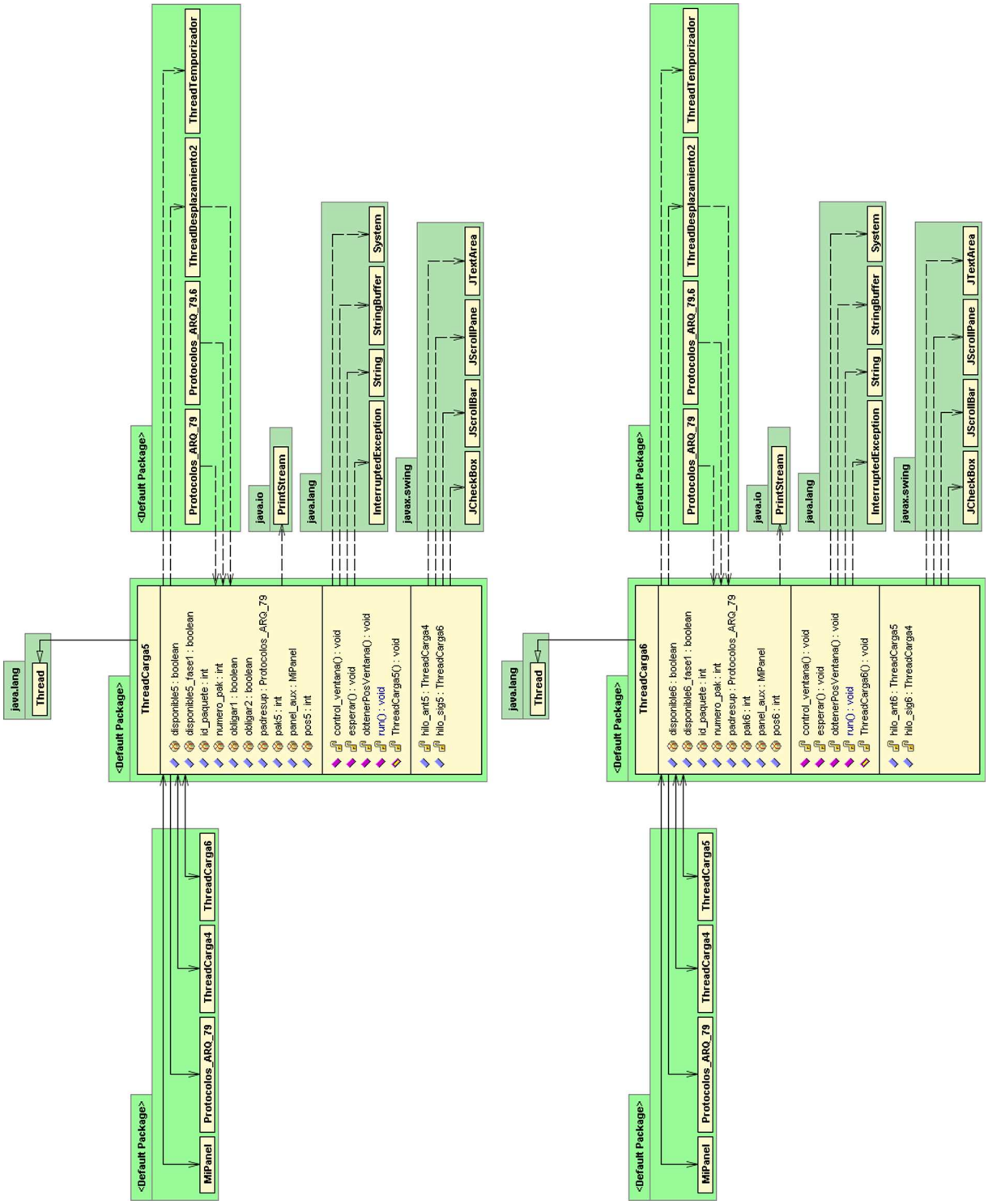
5.5. Diagramas UML

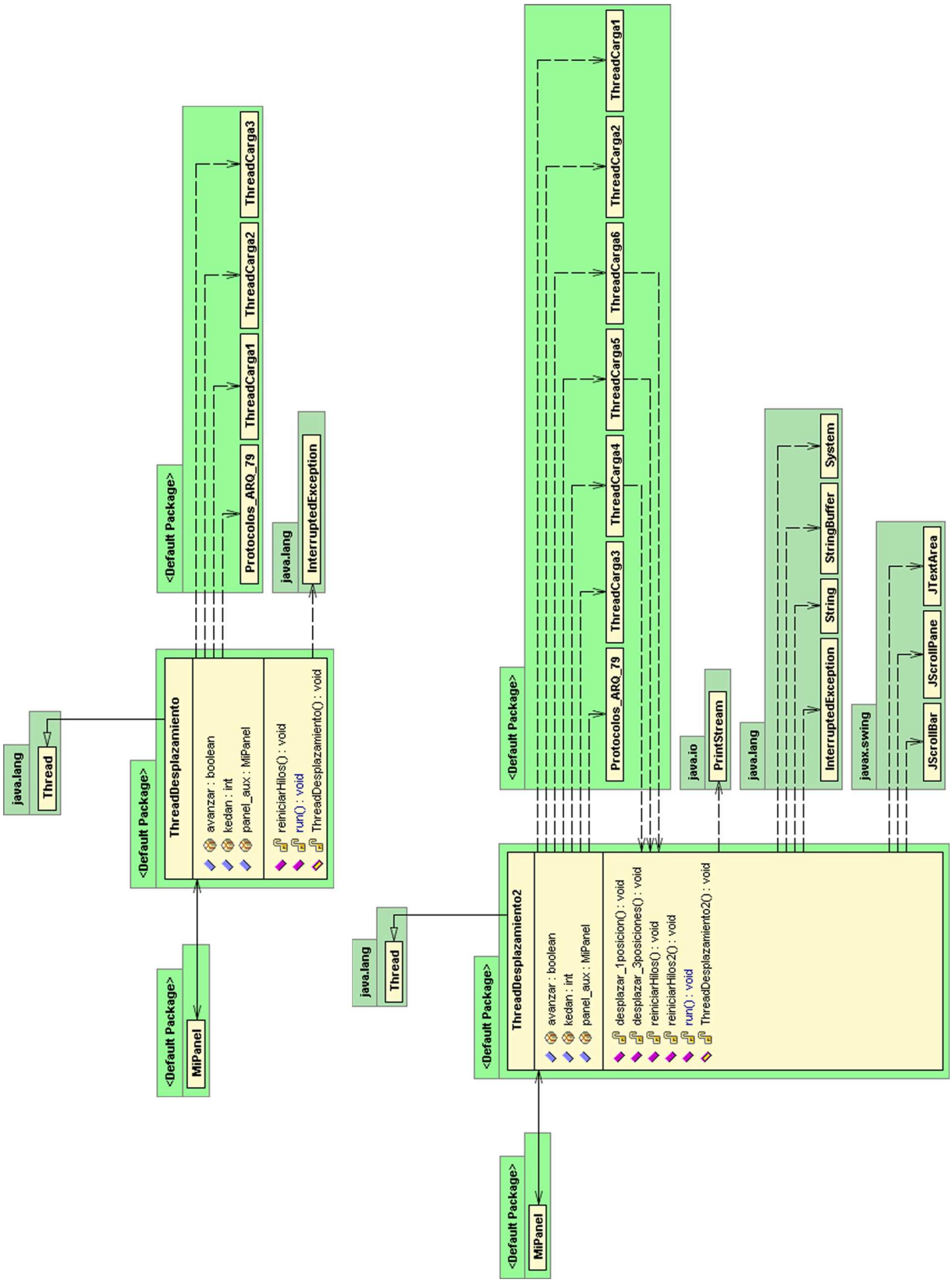


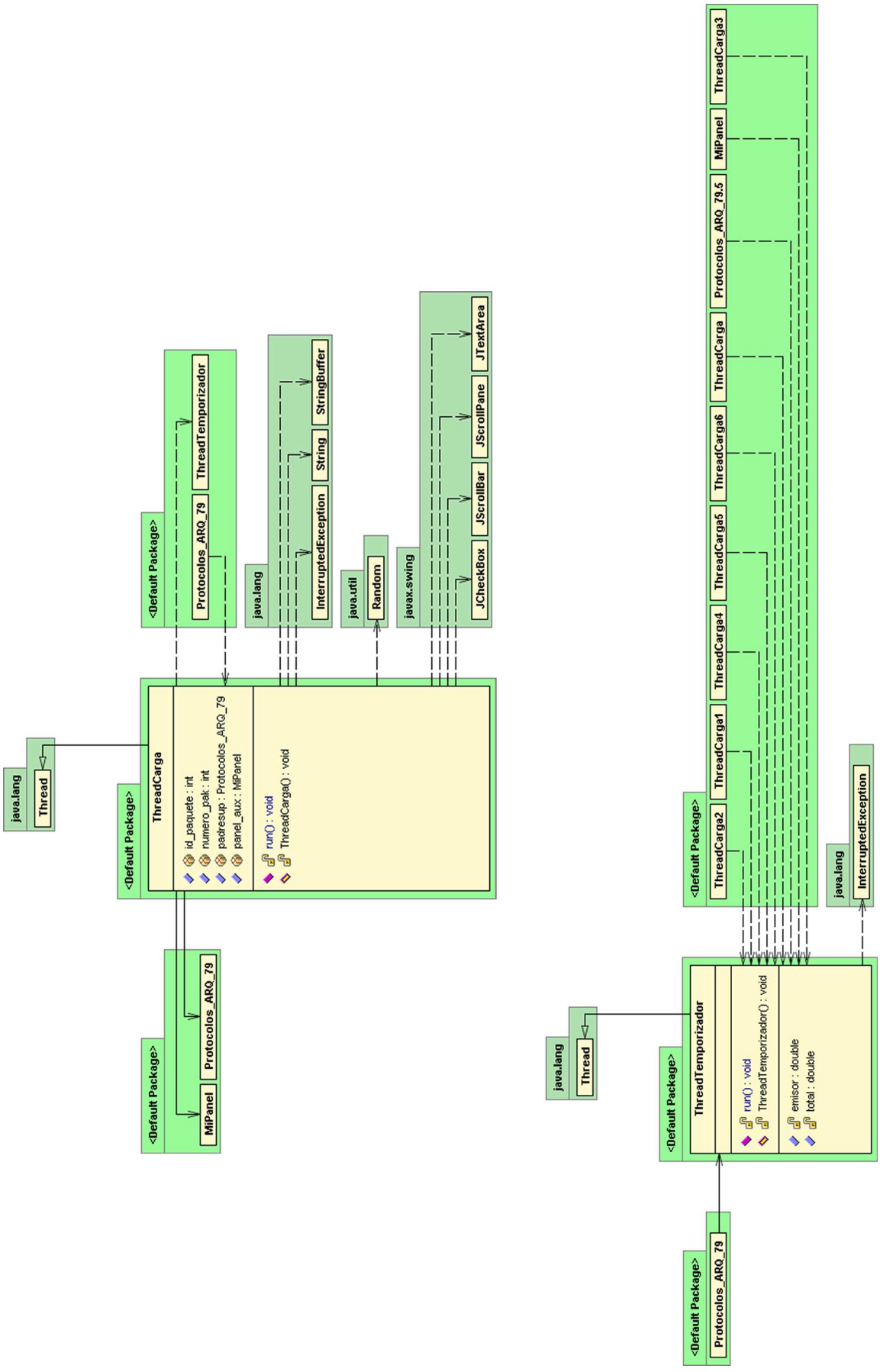

Protocolos_ARQ
MiPanel
panel_A :
JButton
ini : acept : parar : reprogramar : reset :
JPanel
eleccion_errores : panel_automatico : panel_manual :
ThreadTemporizador
timer :
boolean
fin : go_back : hilo1 : pausar : selective_arq : stop_wait :
boolean[]
errores_ack : errores_ack_forzado : errores_tx : boolean[]
errores_tx_forzado :
int
numero : estado : paquete : cont :
int[]
paquetes : ventana_env :
double
prob : probabilidad : tprop : ttx :
String
manual : automatico :
JCheckBox[]
ack : pak :
JTextArea
areadetexto :
JTextField
prob_text :
JScrollPane
pScroll :
JLabel
probab :
JPanel
panelapplet : panelinf2 : panelmed2 : panelsup2 : panelsimulacion :
JLabel
tx_ack : tx_p : label_pak :
JTextField
applet : epp :
JMenu
imenu1 :
JMenuBar
menu :
JMenuItem
imenuItem :
JRadioButtonMenuItem
imenuItemA : imenuItemB : imenuItemC :
void
Protocolos_ARQ
boolean
action() :
void
actionPerformed() :
void
configurar_parametros() : crear_col_checkbox() :
crear_grid() : crear_grid2() : crear_label_pak() :
iniciaCuenta() : iniciaCuenta2() : resetear() :
resetear2() : resetear_stop() : main() :
JPanel
crear_boton() : crear_label() : crear_RadioBoton() :
int
numero_paquete() :
java.awt.event
ActionEvent
ActionListener
java.io
PrintStream
java.util
Random
<Default Package>
ThreadCarga
ThreadCarga1
ThreadCarga2
ThreadCarga3
ThreadCarga4
ThreadCarga5
ThreadCarga6
ThreadDesplazamiento
ThreadDesplazamiento2
java.awt
BorderLayout
CardLayout
Choice
Color
Component
Container
Event
FlowLayout
Font
GridBagConstraints
GridBagLayout
LayoutManager
javax.swing
JFrame
javax.swing.event
javax.swing.border
Border
CompoundBorder
TitledBorder
javax.swing
BorderFactory
BoxLayout
JComponent
JOptionPane
JRadioButton
JScrollBar
java.lang
Double
NullPointerException
NumberFormatException
Object
StringBuffer
System
<Default Package>
MiPanel
ThreadCarga
ThreadCarga1
ThreadCarga2
ThreadCarga3
ThreadCarga4
ThreadCarga5
ThreadCarga6
ThreadTemporizador
javax.swing
JButton
JCheckBox
JLabel
JList
JMenu
JMenuBar
JMenuItem
JPanel
JRadioButtonMenuItem
JScrollPane
JTextArea
JTextField
java.lang
String
Thread


java.lang
Thread
<Default Package>
ThreadCarga1
disponible1 : boolean
disponible1_fase1 : boolean
id_paquete : int
numero_pak : int
padresup : Protocolos_ARQ_79
pak1 : int
panel_aux : MPanel
pos1 : int
esperar() : void
obtenerPosVentana() : void
run() : void
ThreadCarga1() : void
hilo_ant1 : ThreadCarga3
hilo_sig1 : ThreadCarga2
<Default Package>
Protocolos_ARQ_79
ThreadDesplazamiento
Protocolos_ARQ_79.6
ThreadDesplazamiento2
ThreadTemporizador
java.io
PrintStream
java.lang
InterruptedException
String
StringBuffer
System
java.util
Random
javax.swing
JCheckBox
JScrollBar
JScrollPane
JTextArea
<Default Package>
MiPanel
Protocolos_ARQ_79
ThreadCarga1
ThreadCarga2
ThreadCarga3
java.lang
Thread
<Default Package>
ThreadCarga2
disponible2 : boolean
disponible2_fase1 : boolean
id_paquete : int
numero_pak : int
padresup : Protocolos_ARQ_79
pak2 : int
panel_aux : MPanel
pos2 : int
esperar() : void
obtenerPosVentana() : void
run() : void
ThreadCarga2() : void
hilo_ant2 : ThreadCarga1
hilo_sig2 : ThreadCarga3
<Default Package>
Protocolos_ARQ_79
ThreadDesplazamiento
Protocolos_ARQ_79.6
ThreadDesplazamiento2
ThreadTemporizador
java.io
PrintStream
java.lang
InterruptedException
String
StringBuffer
System
java.util
Random
javax.swing
JCheckBox
JScrollBar
JScrollPane
JTextArea
<Default Package>
MiPanel
Protocolos_ARQ_79
ThreadCarga1
ThreadCarga2
ThreadCarga3


<Default Package>
ThreadCarga3
disponible3 : boolean
disponible3_fase1 : boolean
id_paquete : int
numero_pak : int
padresup : Protocolos_ARQ_79
pak3 : int
panel_aux : MiPanel
pos3 : int
esperar() : void
obtenerPosVentana() : void
run() : void
ThreadCarga3() : void
hilo_ant3 : ThreadCarga2
hilo_sig3 : ThreadCarga1
java.lang
Thread
Protocolos_ARQ_79
ThreadDesplazamiento
Protocolos_ARQ_79.6
ThreadDesplazamiento2
ThreadTemporizador
java.io
PrintStream
InterruptedException
String
StringBuffer
System
java.util
Random
JCheckBox
JScrollBar
JScrollPane
JTextArea
javax.swing
ThreadCarga1
ThreadCarga2
MiPanel
ThreadCarga4
disponible4 : boolean
disponible4_fase1 : boolean
id_paquete : int
numero_pak : int
padresup : Protocolos_ARQ_79
pak4 : int
panel_aux : MiPanel
pos4 : int
control_ventana() : void
esperar() : void
obtenerPosVentana() : void
run() : void
ThreadCarga4() : void
hilo_ant4 : ThreadCarga6
hilo_sig4 : ThreadCarga5
ThreadCarga5
ThreadCarga6


<Default Package>
ThreadCarga5
disponible5 : boolean
disponible5_fase1 : boolean
id_paquete : int
numero_pak : int
obligar1 : boolean
obligar2 : boolean
padresup : Protocolos_ARQ_79
pak5 : int
panel_aux : MiPanel
pos5 : int
control_ventana() : void
esperar() : void
obtenerPosVentana() : void
run() : void
ThreadCarga5() : void
hilo_ant5 : ThreadCarga4
hilo_sig5 : ThreadCarga6
java.lang
Thread
Protocolos_ARQ_79
Protocolos_ARQ_79.6
ThreadDesplazamiento2
ThreadTemporizador
java.io
PrintStream
InterruptedException
String
StringBuffer
System
javax.swing
JCheckBox
JScrollBar
JScrollPane
JTextArea
MiPanel
ThreadCarga4
ThreadCarga6
ThreadCarga6
disponible6 : boolean
disponible6_fase1 : boolean
pak6 : int
pos6 : int
hilo_ant6 : ThreadCarga5
hilo_sig6 : ThreadCarga4


<Default Package>
ThreadDesplazamiento
avanzar : boolean
kedan : int
panel_aux : MiPanel
reiniciarHilos() : void
run() : void
ThreadDesplazamiento() : void
Protocolos_ARQ_79
ThreadCarga1
ThreadCarga2
ThreadCarga3
java.lang
Thread
InterruptedException
MiPanel
ThreadDesplazamiento2
avanzar : boolean
kedan : int
panel_aux : MiPanel
desplazar_1posicion() : void
desplazar_3posiciones() : void
reiniciarHilos() : void
reiniciarHilos2() : void
run() : void
ThreadDesplazamiento2() : void
ThreadCarga4
ThreadCarga5
ThreadCarga6
java.io
PrintStream
String
StringBuffer
System
javax.swing
JScrollBar
JScrollPane
JTextArea


<Default Package>
ThreadCarga
id_paquete : int
numero_pak : int
padresup : Protocolos_ARQ_79
panel_aux : MiPanel
run() : void
ThreadCarga() : void
java.lang
Thread
<Default Package>
Protocolos_ARQ_79
ThreadTemporizador
java.lang
InterruptedException
String
StringBuffer
java.util
Random
javax.swing
JCheckBox
JScrollBar
JScrollPane
JTextArea
<Default Package>
MiPanel
Protocolos_ARQ_79
<Default Package>
ThreadCarga1
ThreadCarga2
ThreadCarga3
ThreadCarga4
ThreadCarga5
ThreadCarga6
ThreadCarga
Protocolos_ARQ_79.5
MiPanel
java.lang
InterruptedException
<Default Package>
ThreadTemporizador
run() : void
ThreadTemporizador() : void
emisor : double
total : double
java.lang
Thread
<Default Package>
Protocolos_ARQ_79

Capítulo 6

Applets y html

Java es compatible con todos los sistemas porque basa su funcionamiento en los Byte Codes, que no es más que una precompilación del código fuente de Java.

Estos Byte Codes no son el programa en Java propiamente dicho, sino un archivo que contiene un código intermedio que puede manejar la Máquina Virtual de Java. Cada sistema operativo dispone de una Maquina Virtual de Java que puede interpretar los Byte Codes y transformarlos a sentencias ejecutables en el sistema en cuestión.

Una applet es un fichero de la clase que se escribe específicamente para visualizar gráficos en la red Internet. Las applets se incluyen en páginas web utilizando la etiqueta HTML <APPLET>. Cuando se ejecuta una página Web, las applets de java se descargan automáticamente y el browser las ejecuta, visualizándose en el espacio de la página que se ha reservado para ellas.

Una applet puede hacer de todo, desde trabajar con gráficos, hasta visualizar animaciones, gestionar controles, cuadros de texto, botones, etc. El uso de applets hace que las páginas Web sean activas, no pasivas, lo cual es su principal atracción.

La idea fundamental de visualizar la aplicación a través de un applet, es para que cualquier usuario pueda acceder a ella a través de Internet, además le evitaremos la necesidad de tener que descargarse archivos y la necesidad de utilizar ensambladores, compiladores u otros programas.

El enlace a applet del proyecto se encuentra en la dirección:

http://labit301.upct.es/~jose/

En esta página se encuentra los datos del proyecto como director, autor, título, etc., y el enlace desde donde se arrancará el applet. Además incluyo enlaces hacia otras páginas que considero de interés, tales como un enlace para acceder a la Universidad Politécnica de Cartagena, otro enlace para la Escuela técnica superior de Ingeniería de Telecomunicación y un tercer enlace para acceder al Servidor labitb501.

Cuando pinchamos en el enlace del applet, automáticamente ocurren dos eventos, es decir, se genera una nueva página Web y se visualiza el applet.

La nueva página Web que se genera se hizo con la finalidad de ofrecer información general sobre los protocolos ARQ, además se ha agregado un manual de uso rápido para explicar los aspectos más importantes del manejo de la aplicación de modo que

cualquier usuario pueda entender y utilizar eficientemente el applet sin tener que recurrir a la documentación del proyecto.

El segundo evento que he comentado es que se visualiza el applet, que lógicamente es el objetivo fundamental del proyecto. Hay que mencionar que como cualquier aplicación Java que se visualiza a través de Internet tardará un breve espacio de tiempo, no obstante si pasado un instante no se ha visualizado el applet, podremos saber si todavía se está arrancando observado la parte inferior izquierda de la página Web donde se podrá leer el mensaje “Miniaplicación Applet loaded” que nos indicará que el applet está arrancado, en cualquier otro caso, en esta misma zona se nos mostrará un mensaje de error.

A continuación se presentará el código necesario para obtener las páginas Web que he comentado en los anteriores párrafos, pero antes veo conveniente una introducción a el protocolo html para obtener una mayor comprensión.

El servicio Web o WWW es una nueva forma de representar la información en Internet basada en páginas. Una página WWW puede incluir tres tipos de información: texto, gráficos e hipertexto. Un hipertexto es texto resaltado que el usuario puede activar para cargar otra página WWW. La diferencia entre un documento hipertexto y un documento normal consiste en que el hipertexto contiene, además de la información, una serie de enlaces o conexiones con otros documentos relacionados, de manera que el lector puede pasar de un tema a otro y volver al documento original en el momento en que le interese.

Las principales ventajas del servicio WWW son tres. Primera, que puede combinar texto y gráficos. Segunda, que los hiperenlaces permiten cargar páginas de cualquier otro servidor conectado a Internet, da igual que esté localizado en España o en Australia. Y, tercera, que la creación de páginas WWW es bastante sencilla mediante el lenguaje HTML.

El gran éxito de Web no se debe solamente al empleo del hipertexto. Es normal encontrar que los documentos WWW están compuestos de texto y gráficos, y los enlaces con otros documentos pueden ser palabras clave subrayadas o resaltadas en el texto, pero también la totalidad de una imagen o incluso partes de ella (como un mapa "sensible", que permite acceder a información sobre una ciudad haciendo un "click" del ratón sobre un determinado detalle del plano). Las últimas versiones de los programas navegadores (y la mayor rapidez de las telecomunicaciones) permiten integrar en un mismo documento texto, gráficos, sonidos o, incluso animaciones de vídeo. Estos documentos compuestos son los que reciben el nombre de hipermedia.

HTML, HyperText Markup Language, es un lenguaje simple utilizado para crear documentos de hipertexto para WWW. No es un lenguaje de descripción de página como Postcript; HTML no permite definir de forma estricta la apariencia de una página, aunque una utilización algo desviada hace que se utilice en ocasiones como un lenguaje de presentación. Además, la presentación de la página es muy dependiente del browser (o programa navegador) utilizado: el mismo documento no produce el mismo resultado en la pantalla si se visualiza con un browser en modo línea, Mosaic o

Netscape, o sea, HTML se limita a describir la estructura y el contenido de un documento, y no el formato de la página y su apariencia.

Una de las claves del éxito de WWW, aparte de lo atractivo de su presentación es sin duda, su organización y coherencia. Todos los documentos WWW comparten un mismo aspecto y una única interfaz, lo que facilita enormemente su manejo por parte de cualquier persona. Esto es posible porque el lenguaje HTML, en que están escritos los documentos, no solo permite establecer hiperenlaces entre diferentes documentos, sino que es un "lenguaje de descripción de página" independiente de la plataforma en que se utilice. Es decir un documento HTML contiene toda la información necesaria sobre su aspecto y su interacción con el usuario, y es luego el browser que utilicemos el responsable de asegurar que el documento tenga un aspecto coherente, independientemente del tipo de estación de trabajo desde donde estemos efectuando la consulta.

Su simplicidad es tal que no es necesario utilizar un editor particular. Su gran permisividad exige rigor y atención en la estructura de documentos con el fin de que éstos se visualicen correctamente al margen del contexto y el browser utilizado.

Por tanto, HTML es un lenguaje muy sencillo que nos permite preparar documentos Web insertando en el texto de los mismos una serie de marcas (tags) que controlan los diferentes aspectos de la presentación y comportamiento de sus elementos.

Para escribir HTML lo único que se necesita es un editor de texto ASCII, como EDIT del MS-DOS o el Bloc de notas de Windows. Las marcas o tags que controlan el comportamiento del documento son fragmentos de texto encerrados entre los signos "mayor que" y "menor que" (<marca>). Existen diferentes tipos de marcas: algunas controlan simplemente la presentación del texto del documento; otras, la forma en que se incluirán en él imágenes; otras, finalmente, los hiperenlaces con documentos o con diferentes partes del mismo documento. Existen una serie de programas que ayudan en la elaboración de documentos HTML, como HTMLED (shareware) o HTML Assistant, ambos para Windows, pero no son imprescindibles para escribir el código. Lo que si es necesario es un programa cliente WWW, tal como Mosaic, o Netscape, para probar el documento a medida que lo vamos desarrollando.

Las marcas funcionan muchas veces por parejas, una para indicar el inicio de enlace o formato, y otra para señalar el final. La marca de inicio consiste en una letra o una palabra (por ejemplo, estas son marcas de inicio: <B>, <TITLE>). La marca de final es la misma letra o palabra precedida por la barra inclinada o "slash" (es decir, </B>, </TITLE>). Existen, no obstante, algunas marcas que no requieren su pareja de cierre, como
 (que fuerza un salto de línea). Es importante señalar que las marcas, en general pueden estar indistintamente en mayúsculas o en minúsculas.

El lenguaje HTML nace en 1991 de manos de Tim Bernes-Lee del CERN como un sistema hipertexto con el único objetivo de servir como medio de transmisión de información entre físicos de alta energía como parte de la iniciativa WWW. En 1993 Dan Connelly escribe el primer DTD (Document Type Definition) de SGML describiendo el lenguaje. En 1994 el sistema había tenido tal aceptación que la especificación se había quedado ya obsoleta. Por aquel entonces WWW y Mosaic eran casi sinónimos

debido a que el browser Mosaic del NCSA (National Center for Supercomputing Applications) era el más extendido debido a las mejoras que incorporaba. Es entonces cuando nace el HTML 2.0 en un draft realizado también por Dan Connelly. El crecimiento exponencial que comienza a sufrir el sistema lleva a organizar la First International WWW Conference en Mayo de 1994. El principal avance de 2.0 de HTML es la incorporación de los llamados forms, formularios que permiten que el usuario cliente envíe información al servidor y ésta sea recogida y procesada allí. Precisamente con este fin, NCSA presenta la especificación del CGI, Common Gateway Interface, versión 1.0 que define un interfaz entre programas ejecutables y el sistema WWW. Con la incorporación de los forms, aparecen por primera vez campos donde el usuario puede escribir, menús "pull-down" y los denominados "radio-buttons" integrados en páginas WWW.

Desde entonces, el lenguaje ha seguido creciendo como algo dinámico, como una lengua humana, algo vivo, siendo modificado sobre todo por las personas que lo utilizan. Así, una evolución en el lenguaje suele surgir de una propuesta que es adoptada por algunos clientes (browsers). Con el uso se ve si es eficiente y es adoptada y si es así, finalmente se incorpora al estándar. De este modo, a finales de 1993 se comienza a hablar de HTML+ propuesto por Dave Raggett, de HEP Labs, en Bristol que evoluciona a un nuevo draft de Marzo de 1994 para la versión HTML 3.0 incorporando nuevas posibilidades como la realización de tablas complejas, control de proceso de formatos e incorporación de expresiones matemáticas.

El testigo pasa del browser Mosaic al Netscape, que incorpora nuevas mejoras. Aunque el equipo de Netscape anuncia desde el principio que su browser trata HTML 3.0, lo cierto es que no se adapta al estándar. Por el momento, el único browser de HTML 3.0 es experimental y recibe el nombre de Arena. El lenguaje de Netscape, el más utilizado en la actualidad, incorpora etiquetas no definidas en HTML 3.0, y tiene algunas diferencias con algunas de las definidas, por ejemplo en la realización de tablas. Por otra parte, hasta la versión 2.0, recién aparecida, no permitía el empleo de expresiones matemáticas (al escribir este artículo el autor aún no ha analizado la versión 2.0). Y como gran idea propone la incorporación de un tipo MIME experimental que permite la actualización dinámica de documentos, del que se hablará en el apartado dedicado a la programación de CGI. Por ello, en "los ambientes" se ha comenzado a denominar este lenguaje de Netscape como NHTML 1.1 para diferenciarlo de la verdadera propuesta de HTML 3.0.

Para poder utilizar el servicio Web se necesitan dos partes. Por un lado, la empresa o institución que quiere facilitar su información tiene que crear páginas WWW, siguiendo el estándar definido por el lenguaje HTML, y ponerlas a disposición del público en Internet, en lo que se llama un servidor WWW. Por otro lado, el usuario que quiere acceder a dichas páginas tiene que utilizar un programa (cliente WWW) que lea las páginas WWW e interprete su significado (por ejemplo, un hiperenlace). Estos programas navedores o clientes WWW son los que permiten al ordenador del usuario interpretar el lenguaje HTML.

Existen numerosos programas gratuitos, y algunos comerciales, para leer los documentos WWW. El más conocido es probablemente el Mosaic, del Centro Nacional de Aplicaciones de Supercomputación (NCSA) de los Estados Unidos, del que existen

versiones para diferentes plataformas (UNIX, Mac, Windows). Otros programas muy difundidos son Netscape (cuya versión beta es de libre disposición y que resulta más rápido que Mosaic), Cello, WinWeb o MacWeb (para Macintosh). Las capacidades de los diferentes navegadores pueden variar de uno a otro programa: aunque la mayor parte permiten el uso de gráficos como enlaces, quedan algunos como Lynx, para DOS, Unis o VMS) que sólo funcionan en modo texto.

Interconectar documentos por todo el planeta sobreentiende un medio único de identificación en la red Internet. La dirección única de un documento en WWW es llamada URL -Uniform Resource Locator- y se compone de los siguientes elementos:

- el protocolo de intercambio de datos entre el cliente y el servidor. (HTTP)

- la dirección Internet del servidor que difunde los documentos. Esta dirección es única en toda la red, es la dirección TCP/IP de la máquina. Tiene la forma de una serie de números como 134.158.69.113; al ser estos números difíciles de memorizar, un anuario (DNS) resuelve generalmente la relación entre dirección numérica y nombre simbólico de la máquina/nombre del ámbito (ejemplo: 134.158.48.1 es la dirección de la máquina sioux.in2p3.fr en la que sioux representa el nombre de la máquina y .in2p3.fr el nombre del ámbito);

- el árbol de directorios (el camino) que conduce al documento;

- el nombre del documento que tendrá siempre la extensión .html o .htm.

Menos frecuentemente esta dirección podrá completarse con otros elementos:

- el puerto;

- información de autentificación (username y password);

- argumentos que se pasarán a un programa en la llamada de un enlace ejecutable.

La sintaxis mínima utilizada para representar el URL de un documento es la siguiente:

protocolo://nombre_del_servidor/

cuando no se especifica un nombre de fichero se acudirá al fichero predeterminado del servidor, habitualmente la home page.

La sintaxis que se encuentra habitualmente es:

protocolo://nombre_del_servidor/directorio/subdirectorio/nombre_del _documento

La sintaxis completa es:

protocolo://username;password@nombre_del_servidor:puerto/directorio/subdirectorio//nombre_del _documento?argumentos

Se observará también en ciertas direcciones la presencia del signo tilde (~) delante del nombre de un directorio. Se trata de home pages personales, posibilidad ofrecida a los usuarios que tienen una cuenta en la máquina servidor.

Por último comentaré la estructura básica de un documento HTML, para que los códigos expuestos en los dos puntos siguientes, sean comprendidos perfectamente.

Un documento HTML comienza con la etiqueta <html>, y termina con </html>. Dentro del documento (entre las etiquetas de principio y fin de html), hay dos zonas bien diferenciadas: el encabezamiento, delimitado por <head> y </head>, que sirve para definir diversos valores válidos en todo el documento; y el cuerpo, delimitado por <body> y </body>, donde reside la información del documento.

La única utilidad del encabezamiento en la que nos detendremos es la directiva <title>, que permite especificar el título de un documento HTML. El cuerpo de un documento HTML contiene el texto que, con la presentación y los efectos que se decidan, se presentará ante el hiperlector. Dentro del cuerpo son aplicables todos los efectos que se van a mencionar en el resto de esta guía. Dichos efectos se especifican exclusivamente a través de directivas. Esto quiere decir que los espacios, tabulaciones y retornos de carro que se introduzcan en el fichero fuente no tienen ningún efecto a la hora de la presentación final del documento.

En resumen, la estructura básica de un documento HTML queda de la forma siguiente:

```
<html>
    <head>
        <title>Título</title>
    </head>
    <body>
        Texto del documento, menciones a gráficos, enlaces, etc.
    </body>
</html>
```

6.1. Página Web – index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
    <TITLE>PROYECTO FINAL DE CARRERA - INGENIERO TECNICO EN
TELECOMUNICACIONES ESP. TELEMATICA</TITLE>

</HEAD>

<BODY bgcolor="#C0C0C0">

      <DIV align="right"><IMG src="grafico1.gif" alt="escudo2"></img></div>
      <h1><div align=center >Proyecto Final de Carrera</h1></div>
        <UL>
        <LI><P>Departamento: Tecnologías de la Información y las
Comunicaciones</P>
        <LI><P>Curso académico: 2006/07</P>
        <LI><P>Titulación: Ingeniería Técnica de Telecomunicación, especialidad
Telemática</P>
        </ul>
        <BR>
        <BR>
        <H3>Datos del Proyecto:</h3><BR>
        <UL>
        <LI><P><B>Título del Proyecto:</b> Applet-simulador didáctico de protocolos de
ventana</P>
        <LI><P><B>Autor del Proyecto:</b> José María González Heredia</P>
        <LI><P><B>Director del Proyecto:</b> Francesc Burrull i Mestres</P>
        </ul>

        <BR>
        <P>Proyecto fin de carrera:
        <A href="protocolos.html"> Applet didáctico. </a>  </p>
        <BR>

        <P>Enlaces de interés:</P>
        <A href="http://www.upct.es"> Universidad Politécnica de Cartagena. </a>
<BR>
        <A href="http://www.teleco.upct.es"> Escuela Técnica Superior de Ingeniería de
Telecomunicación. </a>      <BR>
      <A href="http://labit501.upct.es"> Servidor labit501 </a>   <BR>

</BODY>
</HTML>
```

6.2. Página Web – protocolos.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>
<HEAD>
   <TITLE>PROYECTO FINAL DE CARRERA - INGENIERO TECNICO EN
TELECOMUNICACIONES ESP. TELEMATICA</TITLE>
   <APPLET  code="Applet.class" width=0 height=0>       </applet>
</HEAD>

<BODY bgcolor="#C0C0C0">

      <DIV align="right"><IMG src="grafico1.gif" alt="escudo2"></img></div>
        <P>Aplicación arrancada.   <BR>
        Se visualizará en unos instantes.  <BR>
          Si no funciona instalar esto <a href="j2sdk-1_4_0_01-windows-i586.exe">
(j2sdk 1.4.0.1)</a>  </p>
          <BR><BR>
        <h1><div align=center >Manual de uso rápido del Applet:</h1></div>
        <BR>

      <DIV align="center"><IMG src="paneles.jpg" alt="escudo"></img></div>

       <h1><div align=center >Protocolos ARQ</h1></div>
        <BR>
        <P>   &nbspLos Protocolos ARQ son protocolos de control de
flujo y que esencialmente se caracterizan porque solicitan retransmisiones de una o
varias tramas de modo automático cuando se producen errores o pérdidas de
información sin intervención de agentes externos al nivel de enlace.      </p>

        <P>   &nbspPor lo tanto, los protocolos de solicitud de
repetición automática o ARQ (Auto-matic Repeat Request) retransmiten aquellos
paquetes de datos que han llegado con errores, utilizando para ello códigos de
detección de errores. Estos protocolos de repetición automática nacieron como una
alternativa a la "verificación de eco", es decir, un método manual de errores que se
utiliza aún en los terminales de mainframes, consistente en una técnica muy sencilla:
cuando el usuario teclea un carácter, éste es transmitido hasta la computadora y
reenvía de nuevo hacia el terminal que lo presenta en pantalla, si el usuario se da
cuenta de que lo que se muestra no coincide con lo que él esperaba, es el usuario el
encargado de realizar la corrección de errores usando los caracteres de control de los
que dispone el terminal.   </p>

        <P>   &nbspExiste diferentes protocolos ARQ, el protocolo de
parada y espera (stop & wait) que se trata del protocolo ARQ original. Consiste en que
el emisor transmite un paquete cada vez y espera una confirmación, el receptor cuando
recibe un paquete de forma correcta transmitirá un ACK, por su parte si en el emisor se
```

recibe un ACK entenderá que el paquete ha llegado correctamente y continuará con la transmisión del siguiente paquete en el caso de que existiera, si se diera el caso en el que se produce un error en algún momento del proceso, para que éste no se bloquee, el emisor dispone de un temporizador o timer que arrancará cada vez que se transmite un paquete y que si expira pasará el emisor automáticamente a la retransmisión del paquete en cuestión. El problema de este protocolo es que solo permite un frame en el link, por lo que generalmente el emisor se encuentra desocupado a la espera de un ACK y como consecuencia la baja utilización el ancho de banda y además es ineficaz si el retardo de propagación es mayor que el tiempo retransmisión de los paquetes. </p>

<P>   Este problema de la mala utilización del ancho de banda se puede solucionar pudiendo transmitir un número finito de paquetes antes de esperar los ACK correspondientes. Esto se logra con protocolos de ventana deslizante tales como ARQ con retroceso N (go-back n), en donde n indica el número de paquetes que se podrá transmitir sin recibir una confirmación, es decir, n es el tamaño de la ventana o intervalo, dicha ventana irá avanzando a medida que se confirmen las recepciones de los paquetes anteriores, por otra parte el receptor sólo irá aceptando los paquetes de forma ordenada. El proceso de este protocolo se desglosará más específicamente en la documentación del proyecto, no obstante podemos adelantar que el mayor problema del protocolo go-back n es que en caso de error en alguno de los paquetes, el protocolo reenviará toda la ventana, es decir, retransmite todos los paquetes que componen en ese momento la ventana independientemente de que el error solo se haya producido en un paquete en particular. </p>

<P>   Este problema de tener que retransmitir la ventana completa, lo trata de solucionar el protocolo de repetición selectiva (selective ARQ), el cual será capaz de detectar el paquete dentro de la ventana en el que se ha producido el error y retransmitir únicamente dicho paquete. </p>

<H3> <I>   <U>Protocolos Stop & Wait </u></i></h3>

<P>   Como caso más sencillo de protocolo de retransmisión es el denominado de parada y espera, consiste en que el emisor espera confirmación o acuse de recibo después de cada envío y antes de efectuar el siguiente. El acuse de recibo, también llamado ACK (del ingles acknowledgement) sirve tanto para indicar que la trama ha llegado correctamente como para indicar que se está en condiciones de recibir la siguiente, es decir el protocolo incorpora también la función de control de flujo. Este tipo de protocolos donde el emisor espera una confirmación o acuse de recibo para cada dato enviado se denominan protocolos PAR (Positive Acknowledgement with Retransmission) o también ARQ (Automatic Repeat reQuest). </p>

<P>   El receptor verificará cada trama mediante alguno de los métodos de detección de errores (generalmente CRC) y en caso de que la trama recibida sea errónea no se producirá ACK. Lo mismo ocurre cuando la trama enviada

se pierde por completo. En este caso, el emisor pasado un tiempo máximo de espera, reenvía la trama. Una optimización que se puede incorporar en el protocolo es el tiempo máximo de espera, reenvía la trama. Una optimización que se puede incorporar en el protocolo es el uso de acuse de recibo negativo o NAK (Negative Acknowledgement) cuando se recibe una trama errónea; de esta forma el emisor puede reenviar la trama sin esperar a agotar el tiempo de espera, con lo que se consigue una mayor utilización de la línea. </p>

<P>   En el caso de que se pierda el mensaje del ACK, pasado el tiempo de espera el emisor concluirá erróneamente que la trama se ha perdido y la reenviará, llegando ésta duplicada al receptor. Como el receptor no tiene ningún mecanismo para detectar que la trama es un duplicado, pasará el duplicado al nivel de red, lo cual no esta permitido en un protocolo de enlace. Una forma de que el receptor distinga los duplicados es numerar las tramas, por ejemplo con un campo de un bit podemos numerar tramas en base 2 que es suficiente para detectar duplicados. </p>

<H3> <I>   <U>Protocolos de Ventana Deslizante (go-Back n y Selective ARQ)</u></i></h3>

<P>   Todos los protocolos que forman parte de la familia llamados de "ventana deslizante" o sliding window tienen las siguientes características: </p>
<UL>

<LI>Cada una de las tramas de salida está numerada secuencialmente, de modo que quedan unívocamente identificadas. Si el número de bits de la secuencia es mayor, será posible un mayor rendimiento del protocolo en ausencia de errores, si elegimos adecuandamente los parámetros de la transición.

<LI>En cualquier instante de la transmisión el emisor posee una lista de los números de trama que ha enviado al receptor. Todas estas tramas constituyen lo que se denomina ventana emisora.

<LI>Del mismo modo, el receptor posee una lista con las tramas que está dispuesto a aceptar del emisor, es decir, una ventana receptora. La ventana emisora de un emisor no tiene porque ser del mismo tamaño que la ventana receptora del receptor.

<LI>El receptor aceptará la trama procedente del emisor si cae dentro de su ventana receptora; sin embargo, no importa el orden en que las tramas son enviadas, lo que proporciona más libertad a la capa de enlace para realizar la gestión de las tramas.

<LI>Una vez iniciada la transmisión, la ventana emisora contiene los números de trama que se han enviado y de las que todavía no se ha recibido información. Cuando la capa de red suministra un paquete a la capa de enlace, ésta lo encapsula en una o varias tramas a las que le asigna los número de secuencia siguientes y son puestos en

la ventana emisora, se envían al receptor y se espera en ese estado que lleguen las confirmaciones.

<LI>Cada trama enviada es mantenida en un buffer de memoria en espera de la confirmación. Si ésta es positiva se libera el buffer, si es negativa o no se recibe ninguna confirmación, se efectúa la retransmisión. La ventana emisora contiene el número máximo de tramas que pueden ser enviadas al receptor sin necesidad de confirmación. Una vez que se ha superado ese número, el proceso de envío se detiene en espera de alguna confirmación que libere alguna trama enviada y pueda ser repuesta por un nuevo envío.

<LI>De modo análogo, el receptor mantiene una ventana receptora con los números de secuencia de las tramas que es capaz de recibir. Si recibe una trama cuyo número de secuencia no está en su ventana receptora, es descartada sin más. Como el emisor nunca recibirá una confirmación de esta trama descartada, se tendrá que ocupar más delante de retransmitirla, una vez que hayan vencido sus temporizadores. Sólo las tramas recibidas dentro de su ventana receptora son pasadas a la capa de red, generando una confirmación que liberará una o más tramas de la ventana emisora en el emisor, lo que le dará a éste permiso para nuevos envíos.

<LI>La ventana del receptor siempre tiene un tamaño constante a diferencia de la del emisor, que puede ir creciendo paulatinamente hasta llegar a un máximo fijado por el protocolo.

<P>   El protocolo de envío y espera es un protocolo de ventana deslizante, en el que tanto el tamaño de la ventana receptora como emisora es uno. De modo, que cuando el emisor envía la única trama que puede enviar, debe pararse hasta que el receptor le envíe la confirmación de que le llegó, liberando la trama enviada y dando paso a la siguiente. </p>

<P>   Pos su parte, el receptor puede recibir sólo una trama, concretamente la siguiente a la anterior en su número de secuencia. Si no recibe ésta entiende que ha habido alguna trama que se ha perdido y no confirma la trama afirmativamente, con lo que el emisor retransmitirá. Si la trama llegó correctamente genera una confirmación afirmativa para liberar de su espera al emisor y activar la transmisión de la siguiente trama. </p>

<P>   Como en el resto de protocolos de envío y espera, cuando las ventanas son mayores que uno, las confirmaciones se pueden enviar una a una por cada trama recibida o por conjuntos de tramas. Del mismo modo, las retransmisiones pueden ser selectiva o no. </p>

</BODY>
</HTML>

Capítulo 7

Conclusiones y líneas futuras

Con este proyecto fin de carrera se ha conseguido todos los objetivos que inicialmente estaban marcados, que a modo resumen son:

- Se ha diseñado y desarrollado un applet para la simulación de los protocolos ARQ. Creando para ello una interfaz gráfica atractiva e intuitiva facilitando el uso y comprensión del simulador.

- Se ha separado la simulación en dos partes: una más visual, que ayude a comprender el funcionamiento del protocolo paso a paso, y otra simulación que se ejecuta de modo automático y da una idea global de funcionamiento.

- Se ha realizado una aplicación gratuita y de código abierto que ayuda a comprender mejor algunos de los aspectos de los protocolos ARQ. El código fuente se ha dejado abierto y disponible, para dejar el programa disponible a posibles mejoras, y a la depuración de posibles errores. Cualquiera puede mejorar, cambiar o añadir funcionalidades al programa, según las futuras necesidades que se tengan.

- El aprendizaje de la programación de Applets Java, ampliamente utilizados en WWW.

- La compresión de un protocolo de comunicaciones distribuido y su programación correcta mediante threads.

- Se ha conseguido el objetivo de la portabilidad, ya que al estar el programa hecho en java, se puede ejecutar en cualquier plataforma (Windows, LINUX, Solaris, etc.)

Respecto a las posibles líneas futuras, es evidente que al haberlo realizado en un lenguaje de programación como es Java, el código fuente queda abierto y por lo tanto disponible para que cualquier usuario pueda mejorar la aplicación, depurar posibles errores e incluso reutilizar partes del código.

Además dada la disponibilidad y la situación de las distintas partes del intefaz gráfico, esta misma aplicación puede ampliarse con otros protocolos similares de transmisión o incluso implementar distintas versiones de los protocolos ARQ.

Bibliografía

Internet:

- http://www.programacion.com/java/
- http://www.ctr.unican.es/fundamentos/
- http://www.programacion.com/tutorial/swing/
- http://java.sun.com

Apuntes:

- Universidad Politécnica de Cartagena:
  - Asignatura: Ingeniería de Protocolos y Servicios.
  - Tema de Control de flujo.

- Universidad del Valle:
  - Asignatura: Redes de Comunicación.
  - Tema: Automatic Retransmission Request.

- Universidad de Almería:
  - Departamento de Arquitectura de Computadores y Electrónica.
  - Libro: Introducción a las Redes de Computadoras.

- Diversidad Técnica Federico Santa María:
  - Asignatura: Redes de computadores.
  - Tema de transmisión de frames.

- Universidad de Barcelona:
  - Asignatura: Xarxes de computadors.
  - Tema de protocolos de comunicación punto a punto.

Libros:

- Como programar en Java: Deitel & Deitel
  ed. Prentice hall
- Java, How to Program. Editorial: Deitel-Deitel, 5º Ed.
  ed. Prentice Hall
- Java2.
  Editorial: Anaya
- Java Thread Programming
  ed. SAMS
- Manual de Java,
  ed. McGraw-Hill


UNIVERSIDAD POLITECNICA DE CARTAGENA
Fechos Allend Mar
anno MCMXCVII