Soporte para la simulación de protocolos de acceso al medio en un canal broadcast bidireccional de acceso múltiple

Guillermo Rigotti
UNICEN – Fac. de Ciencias Exactas-ISISTAN
Pje. Arroyo Seco, (7000) Tandil, Bs. As. Argentina
TE: +54-2293-440363 FAX: +54-2293-440362
Email: grigotti@exa.unicen.edu.ar

Resumen

En este trabajo se presenta un soporte desarrollado con el objetivo de permitir la programación y prueba de diferentes protocolos de acceso al medio (nivel MAC del modelo IEEE 802) operando sobre vínculos broadcast bidireccionales con acceso múltiple. El trabajo presentado aquí forma parte de una serie de desarrollos similares que abarcan diferentes aspectos de los protocolos de comunicación, y cuyo objeto es proveer herramientas que faciliten la enseñanza de las materias del área de comunicación de datos a través de la experimentación con los protocolos correspondientes en cada caso. En particular, este soporte permite el desarrollo, la modificación y prueba a través de simulación de protocolos de nivel MAC existentes, como CSMA/CD, de otros definidos pero no implementados, como Adaptive Tree Walk, o el desarrollo de nuevos protocolos.
Palabras clave: Acceso al medio, protocolos.
Workshop: Redes y Comunicaciones

1. Introducción

El presente trabajo constituye una parte de un proyecto más amplio, destinado a ser utilizado por las cátedras del área de comunicación de datos. El objetivo general perseguido es el desarrollo de un ambiente homogéneo de rápida asimilación por parte de los alumnos, que posibilite la comprensión y el manejo de los conceptos de mayor importancia relacionados con las materias correspondientes, a través de la programación, simulación y/o visualización de interacciones entre los procesos que componen los protocolos. Una descripción de este proyecto puede encontrarse en [Rigotti, 2003].

El trabajo presentado aquí consiste en la implementación de un medioambiente que permite la programación y prueba a través de simulación, de protocolos de acceso al medio. Este medioambiente presenta una interfaz de uso muy simple, evitando que los alumnos deban interiorizarse de las características de alguna plataforma en particular. De esta forma, es posible focalizar la atención en las interacciones entre los procesos, aspecto de interés para las cátedras.

El soporte desarrollado consiste de un conjunto de módulos que proveen una interfaz simple de utilizar a los protocolos de control de acceso al medio (nivel MAC de la arquitectura IEEE 802) a desarrollar por los alumnos. En particular, se ha implementado un canal broadcast bidireccional de acceso múltiple, similar al utilizado en redes Ethernet en sus versiones 10Mbps y modo half duplex de mayores velocidades, ya que permite la experimentación con un amplio conjunto de protocolos, desde los no controlados con sus variantes (detección de actividad, de colisión, estrategias de envio, etc), hasta los controlados como Basic Bit Map y los híbridos como Adaptive Tree Walk.

. Debido a la arquitectura modular de la implementación, es posible reemplazar dicho canal sin afectar sensiblemente la interfaz ofrecida a los protocolos de acceso al medio[1]. La incorporación

[1] Deberían incorporrse o eliminarse funciones propias del tipo de canal, por ejemplo, detección de colisión (CD), testeo de actividad (CS), etc.

de otro tipo de medios se considera como un futuro desarrollo, y es de interés debido al creciente uso de otros tipos de comunicación, como redes inalámbricas.

El trabajo realizado hasta el momento, se compone de varios módulos: el de nivel inferior representa el canal de comunicaciones con las características ya descriptas; en el siguiente nivel se encuentra el módulo que representa al driver para dicho canal y que permite utilizar funciones para controlar la transmisión; por último, en el nivel superior se sitúan los módulos que representan la estrategia de acceso al medio, que son desarrollados por los usuarios para cada protocolo en particular.

El sistema permite el desarrollo de nuevos protocolos, la modificación de los ya existentes y la evaluación de su funcionamiento en base a simulaciones. Asimismo es posible la elaboración de ejemplos representativos de situaciones particulares en los protocolos. En el presente trabajo se describe, como ejemplo, la implementación de un protocolo Aloha puro y de un protocolo Carrier sense.

Tanto el soporte de programación como los módulos a desarrollar por los usuarios deben implementarse en lenguaje Tcl [Osterhout,1994], con el agregado de las facilidades provistas por Ns [Fall, 2002]. De éstas se utiliza sólo un subconjunto, siendo la de mayor relevancia el scheduler de eventos. La elección de dicha plataforma responde a su portabilidad, sencillez de desarrollo y breve tiempo de asimilación por parte de los alumnos.

Este documento está organizado de la siguiente manera: en la sección 2 se describen los componentes del medioambiente desarrollado, en la sección 3 se explica la interacción entre dichos componentes. La sección 4 hace referencia a implementaciones de protocolos en particular, que fueron implementadas para comprobar el funcionamiento del soporte. Finalmente, en la sección 5 se resumen las conclusiones del trabajo realizado hasta el presente y se plantean futuras extensiones y mejoras a realizar.

2.Componentes

El soporte desarrollado se compone de cuatro entidades que representan al canal de comunicación y a los equipos conectados a él. Estos se muestran en la figura 1:

1- Canal: representa el medio que conecta los equipos, cuya función es la de transportar los frames o señales especiales que los equipos envíen e informar de ellas a su debido tiempo (considerando la demora de propagación) a los demás equipos.
2- Drivers: es la parte del control de acceso al medio que interactúa con el canal. Es responsable de la emisión de señales (datos o señales especiales como refuerzo de colisión), de mantener el estado del canal (recibiendo datos, ocioso, colisión, etc) e informar al módulo de acceso al medio de cualquier cambio de estado para que éste pueda tomar las acciones correspondientes. Debe mantener además su propio estado (ocioso, enviando, etc).
3- El protocolo de acceso al medio: es el encargado de implementar la estrategia de acceso al medio, basado en las demandas de transmisión de la aplicación y en la información que le ofrece el driver. Por ejemplo, decidirá si debe o no comenzar una transmisión, cuánto tiempo esperar para chequear el canal, implementar estrategias de retransmisión como exponential backoff, etc.
4- Los equipos que representan las estaciones conectadas al canal. Tienen por función contener y servir de nexo a los dos elementos que componen el protocolo MAC: el driver y el acceso al medio. A través de los equipos se establece también la ubicación física de los mismos en el cable.

A continuación se describe cada uno de los componentes con mayor detalle.

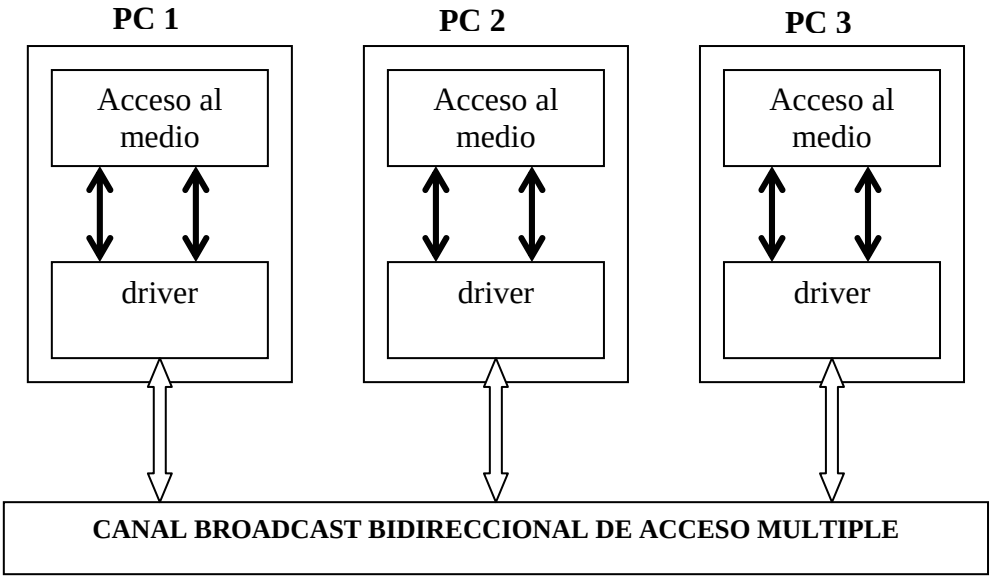

Fig. 1. Componentes del soporte de programación para protocolos de acceso al medio

2.1 Medio de comunicación

Representa el medio de transmisión. Las características relevantes para nuestro propósito son su longitud, su demora de propagación, y la velocidad de transmisión.

Estos parámetros deben especificarse de la siguiente manera: velocidad de transmisión en bits/segundo, demora de propagación en microsegundos por km y longitud del link y ubicación de los equipos en metros.

A continuación se muestra un ejemplo de especificación de las características del canal. Es el caso de un vinculo Ethernet, de 10Mbps con una longitud de 500 metros y una demora de propagación de 5 microsegundos por kilómetro. Hay tres equipos conectados, a los 100, 200 y 400 metros a partir de un extremo del canal.

Vt = 10.000.000 (bps), dp = 5 (us/Km) long_canal = 500 (m), ubicación equipos: 100, 200, y 400 (m).

2.1.1 Funcionamiento del canal de comunicación (Link)

Se describe a continuación el funcionamiento interno del canal de comunicación. Este funcionamiento trata de simular el desplazamiento de las señales en un canal del tipo broadcast múltiple acceso soportado por un cable. Para otros tipos de vínculo, por ejemplo satélite o radio, el funcionamiento del canal deberá modificarse, siendo importante mantener similar la interacción canal-driver.

Para simular el recorrido de las señales en el canal, éste se considera dividido en sectores de longitud tal que la emisión de un bit o señal especial a emitir insuma una o más unidades de tiempo. Esto implica que la señal se desplazará al siguiente sector en cada unidad de tiempo de simulación (se elige como unidad un microsegundo).

En la figura 2 se muestra un canal dividido en 10 sectores, con dos PCs conectadas. Se asume que un bloque emitido por la PC A tiene una duración igual a dos unidades de tiempo, es decir, ocupará dos sectores de dicho canal. En la figura se observan los diferentes estados del canal desde que este frame es emitido hasta 3 unidades de tiempo después.

Para cada sector del canal se almacena, para un determinado slot de tiempo, la siguiente

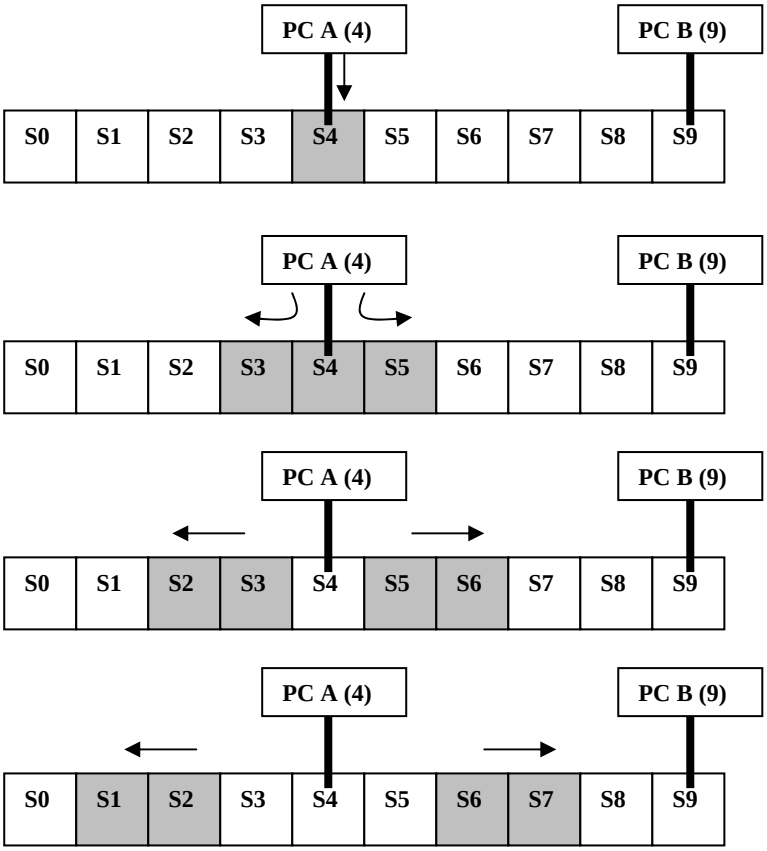

Fig 2. Emisión de un frame con una duración igual a dos unidades de tiempo y su desplazamiento en el canal

información:

1-El estado de la señal en dicho sector (indicado por el valor almacenado en el arreglo señal). Un valor 0 significa ausencia de señal (en la figura se muestra en blanco), es decir, canal ocioso. Un valor 1, significa que hay una señal que se corresponde con envío de datos (gris en la figura). Un valor 2, significa que en ese momento se registra o bien la superposición de dos señales de datos, o bien una señal de colisión emitida por un equipo. Un valor 3 implica la superposición de tres señales de datos, o de una de colisión con una de datos. De la misma manera deberán interpretarse los valores mayores. Lo que debe tenerse en cuenta desde el punto de vista de los receptores, es que un nivel mayor o igual a dos significa colisión. El hecho de incrementar el valor por cada señal que se superpone, responde a que en un momento determinado cada una de las señales que está

circulando por el sector en particular finalizará, produciendo una disminución de dicho valor, y de esta manera se determinará en qué momento el canal vuelve a estado ocioso (valor cero).

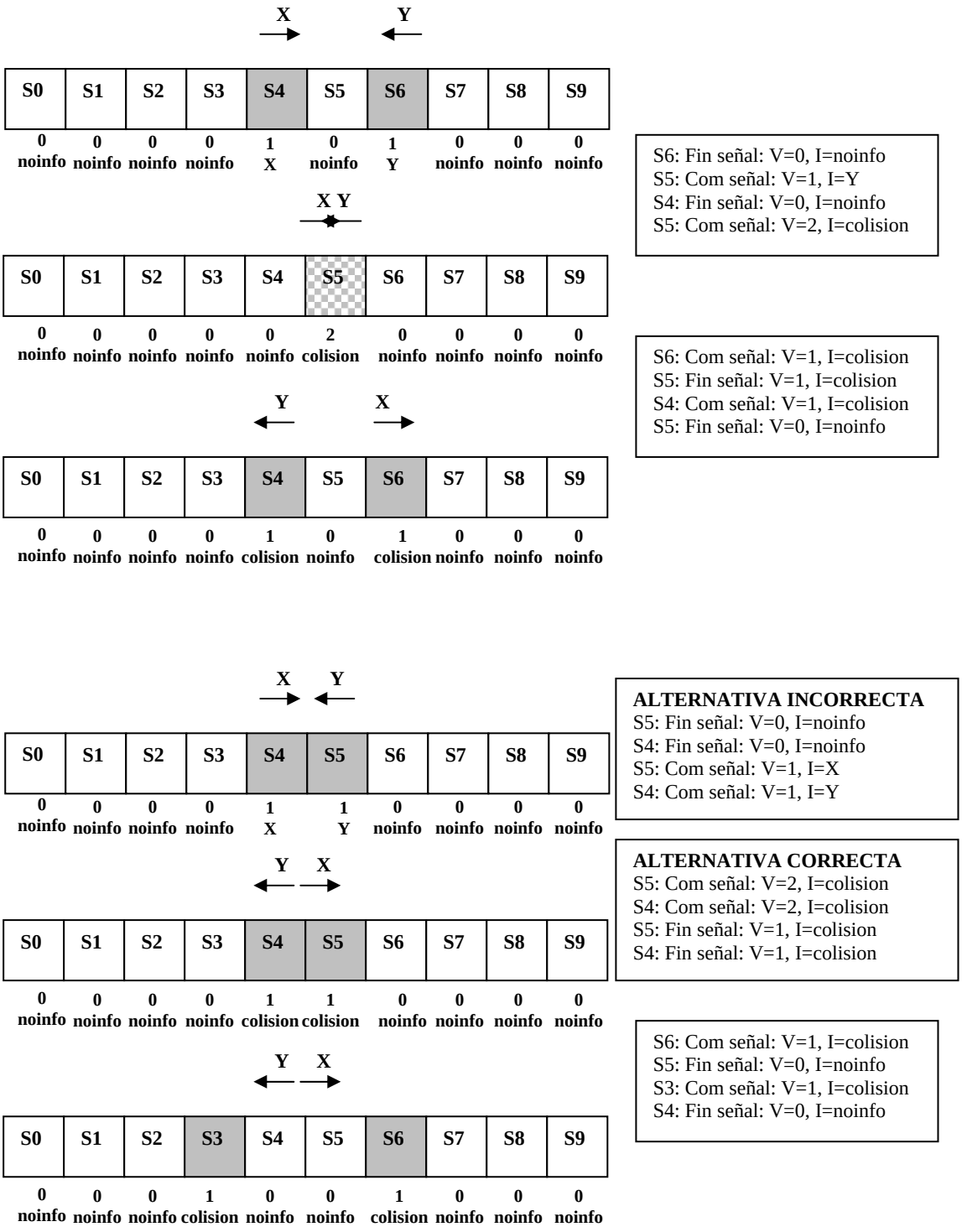

Fig 3. Cruce de dos señales y alternativa de programación adoptada. S: spot, V: estado (volumen) de la señal, I: información

2-Información acerca del contenido del frame o señal que actualmente está en parte ocupando un sector del canal. Esta información es necesaria debido a que será el medio por el cual un driver conocerá la información que transporta un frame de datos cuando es recibido. Por supuesto que el driver no considerará esta información como recibida hasta que la totalidad del frame no haya llegado (si un frame ocupa por ejemplo 2 slots del canal, la información estará repetida en ambos). Esta información es un string con campos separados por el carácter /, que contienen la identificación del equipo de origen, la del equipo de destino, una identificación unívoca del frame y el tiempo que éste dura –esto se utiliza para saber si un frame ha llegado completo a destino o si ha sido truncado-.

El mecanismo utilizado para simular el desplazamiento de las señales en el canal es el siguiente: cuando un equipo desea enviar un frame o señal al canal, invoca a la función evento del canal, indicando que se trata de la emisión del comienzo de un frame o señal, su tipo y su contenido. Cuando llega el tiempo de finalización del frame o señal, el equipo generará una invocación especificando el fin de su emisión.

Por su parte, al recibir el evento de emisión, el canal actualizará el estado del slot al cual está conectado el emisor, y producirá dos eventos para la siguiente unidad de tiempo, uno que indicará la llegada de esta señal al slot contiguo a la derecha y otro que la indicará al slot contiguo a la izquierda. De esta manera, el estado de los slots del canal se irá modificando de acuerdo al tiempo de simulación transcurrido. Debe notarse que una vez introducido el estado en un slot, este se mantiene en él (además de propagarse a los contiguos); por lo tanto, es necesario el evento de fin de envío, que se propaga de manera similar al de comienzo, pero eliminando el estado en los slots correspondientes.

Cuando dos señales se superponen en el canal, la información original se pierde, produciéndose una colisión. El estado de cada slot correspondiente se determina por el valor asignado, que es la suma de los valores introducidos. La información acerca del contenido de la señal corriente en el slot, no será ninguna de las que originalmente llevaban las señales, sino que estas se pierden y se la reemplaza por “colisión”. De esta manera es como el driver identificará que se ha producido una colisión en el caso en que dos señales de muy corta duración choquen en una zona del canal y luego, sin superponerse entre sí, sigan su camino. En este caso, lo que se recibe será un valor de señal igual a uno (debido a que permanece una única señal – distorsionada por el choque-, pero con un campo de información propagado igual a “colisión”. Este caso se muestra en la figura 3.

En ella se muestran dos casos posibles de choque de dos señales que ocupan un slot cada una. Se detallan los 4 eventos involucrados en el movimiento de las mismas, resaltando en el segundo caso, que el orden en que se tratan los mismos puede conducir a la no detección de la colisión.

Para que el proceso sea correcto en un caso como el que se muestra en segundo termino, los eventos deben ser procesados en un orden determinado: primero los de comienzo de señal y luego los de finalización. Por esta razón, el driver, cuando genera el evento de fin de señal, lo hace con una milésima parte de unidad de tiempo de atraso respecto al tiempo correcto, asegurando que el scheduler trate primero a los eventos de comienzo.

2.1.2 Funciones del canal de comunicación

init (nom, long, vt, dp): Crea una instancia del canal de acceso múltiple que conecta los equipos. Los parámetros que recibe son el nombre del canal, la longitud , la demora de propagación y la velocidad de transmisión. El canal se divide en unidades llamadas slots; una señal se desplaza en el

canal slot por slot (no es posible que dos puntos dentro de un mismo slot tengan un estado diferente).
emitir(clase, tipo, ubicación, info): Es invocada por el driver para enviar una señal al canal. Esta función se encarga de actualizar el nivel de señal en el slot correspondiente, invocando luego a **propagar1**, que se encarga de procesar todos los eventos del slot y actualizarlo de acuerdo a ellos (tener en cuenta que un slot puede ser modificado por más de una señal en el mismo intervalo de tiempo).
Clase de señal se refiere al tipo de señal que se inyecta, puede ser datos o una señal especial .
Tipo indica si se trata del comienzo de la emisión o del fin de la emisión. Cuando el driver comienza a emitir, en un tiempo determinado, el estado en el slot al que está conectado se mantiene (además de propagarse a los adyacentes) hasta que el driver indica el fin de la emisión.
Ubicación indica el punto en el canal al que está conectado el equipo.
Info: es la información que contiene el frame. En caso de que haya colisión en el canal, esta información original se pierde, reemplazándosela por "colisión"..
propagar(clase, tipo,ubicación, direccion, info): esta función se utiliza de manera similar a emitir. La diferencia es que es invocada por el mismo canal, para propagar una señal al slot adyacente. El parámetro adicional, dirección, indica si la propagación se debe hacer hacia uno u otro lado del canal. Una invocación a emitir, hará que un slot del canal (el conectado al driver) actualice su estado y luego realice (en el próximo slot de tiempo) una invocación a **propagar** para el slot de la izquierda y otra para el de la derecha, y así sucesivamente hasta llegar a los extremos del canal.
propagar1(clase, origen, tipo, lugar, direccion, info): es invocada por emitir y propagar para resolver el estado del slot una vez que se conocen los eventos que lo afectan. Se encarga también de producir los eventos de propagación a los slots adyacentes.
conectar(pc, ubi): conecta al vínculo de transmisión el equipo a la distancia indicados
get_vel_trans(): devuelve la velocidad de transmisión
get_longitud(): devuelve la longitud del vinculo de transmisión
get_equipo(indice): obtiene el equipo conectado al canal en el orden dado por índice
get_cant_equipos() : devuelve la cantidad de equipos conectados al vínculo de transmisión

2.2 Driver

Es el módulo que se encarga de interactuar con el canal. Es quien invoca, a pedido del módulo de acceso al medio, a la función **emitir** del canal. En la parte de emisión, mantiene el estado del driver, que puede ser ocioso o transmitiendo. Cualquier cambio de estado, es informado al módulo de acceso al medio.

En la parte de recepción, registra el estado del canal en cada momento, y en base a las señales que está recibiendo de cada uno de los equipos, resuelve el estado del canal. Cualquier cambio de estado que se produzca, es informado al módulo de acceso al medio.
En la recepción, el driver debe resolver el estado del canal y el estado de un bloque enviado por un emisor (por ejemplo, un bloque de datos).

Dependiendo de las señales que puedan emitir los equipos, podrá variar parte de la programación de este módulo.

El driver será el responsable de determinar, en base a las señales recibidas por el canal, posibles condiciones de error, señales especiales o bloques con información.

2.2.1 Funciones del driver

init(pc) : crea una instancia del driver asociándola a un equipo que se pasa como parámetro. Configura el estado inicial del driver y del canal como idle (ocioso), y se registra con el equipo.

evento(señal, info): invocada por el canal cada vez que un cambio en la señal se propaga al slot donde se encuentra conectado el equipo. Almacena el nuevo estado y la información recibida e invoca a **resolver_estado_link**.
resolver_estado_link() :Invocada por evento cuando detecta un cambio en la señal que recibe del link. Es la encargada de determinar el estado del canal (data, idle, colisión) y eventualmente informar de condiciones normales (frame recibido) o anormales al módulo que implementa el método de acceso.
enviar_frame(numbits, frame): produce el envío de un frame de la longitud especificada por numbits. El contenido del frame está dado por el string frame pasado como parámetro. El resultado de esta función es la invocación a la función **emitir** del canal, para un tipo de señal datos, durante el tiempo calculado de transmisión del frame. El estado del driver es cambiado a ENVIANDO y es actualizado en el módulo de acceso al medio. Almacena temporariamente los eventos que producirán el fin de envío local y el cese de la emisión al canal, para el caso en que reciba la orden de abortar.
enviar_señal(tipo, tiempo): es similar a **enviar_frame**. La diferencia esta en el tipo de dato (en este caso el de la señal especial).
abortar(): el driver aborta la transmisión corriente, anulando el evento de fin de señal previo y reemplazándolo por uno mas próximo. Igual con el evento de fin de envío. El estado del driver cambia a idle.
consultar_estado _link (): devuelve el estado del link.
consultar_estado_driver(): devuelve el estado del driver.
fin_envio(): es invocada al finalizar el envío de un frame (o señal especial). Cambia el estado del driver a idle e invoca a la función **fin_envio** del control de acceso al medio.

2.3 Módulo de control de acceso al medio

Es el módulo que implementa la política de acceso al medio, basándose para tomar las decisiones en la información entregada por el driver.

Es un módulo altamente dependiente del protocolo de acceso al medio utilizado, pero pueden definirse funciones genéricas que le permiten interactuar con el driver.

El módulo de acceso al medio será el encargado de implementar el protocolo utilizado, interactuando con el driver para solicitar el envío de señales o de información, y para recibir los bloques entrantes.

2.3.1 Funciones del módulo de control del acceso al medio

En el módulo de acceso al medio, se proveen funciones básicas que deben ser completadas y/o reemplazadas para cada protocolo.
init(eq) crea una instancia del módulo de acceso al medio, la registra en el equipo que se pasa como parámetro y obtiene una referencia al canal al que está conectado el equipo.
consultar_estado_driver y **consultar_estado_link:** Las consultas al driver por el estado del canal (ocioso, colisión, etc) y el estado del driver (transmitiendo, ocioso) se realizan utilizando las funciones anteriores, provistas por el driver. Estos estados los necesitará para su operación. Estas funciones no serán normalmente utilizadas durante el resto de la operación del protocolo, ya que una vez registrado el módulo de acceso al medio, el driver se encarga de actualizar automáticamente las variables correspondientes en el módulo de acceso al medio (**estado_drvr** y **estado_link**).
fin_envio(): El driver informa al módulo de acceso al medio que ha finalizado la transmisión del frame corriente. La funcionalidad de este método dependerá de las características del protocolo de acceso al medio.

actualizar_estado_driver(nuevo_estado): El driver invoca esta función cada vez que se produce un cambio de estado en el driver. De esta manera se mantiene actualizado dicho estado en el módulo de acceso al medio.
actualizar_estado_link(nuevo_estado): El driver invoca esta función cada vez que se produce un cambio de estado en el canal. De esta manera se mantiene actualizado dicho estado en el módulo de acceso al medio.
actualizar_frame(nuevo_frame): El driver invoca a esta función cada vez que recibe un frane de datos sin error para el equipo. Su objetivo es almacenar en variables del módulo de acceso al medio el dato recibido.
actualizar_señal(nueva_señal): Es invocada por el driver cuando recibe una señal especial reconocible a través del canal. Almacena el tipo de señal en variables del módulo de acceso al medio.
interrupcion(codigo):.es una función invocada por el driver para anunciar que se ha producido una situación excepcional. Están previstas las interrupciones por recepción de una señal especial, datos, cambio de estado en el link o en el driver.
start() : Es una función invocada para activar el módulo de acceso al medio. La funcionalidad asignada depende del protocolo que se esté desarrollando.

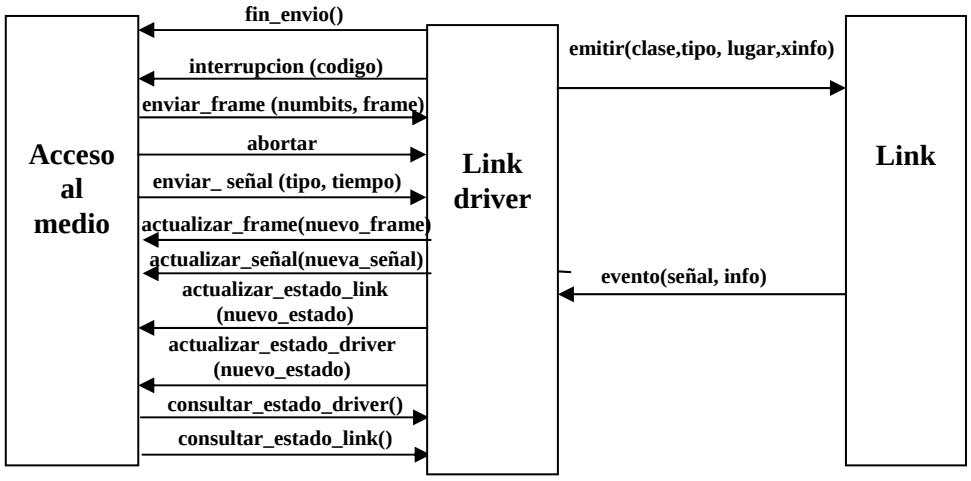

Fig 4. Interacción entre los módulos descriptos

2.4 Representación de los equipos (PC)

Como fue comentado, los equipos conectados al canal están representados por objetos de la clase PC. Su funcionalidad es mínima y consiste en conectarse al canal en un determinado punto, y en servir de nexo a los componentes que residen en cada equipo: el driver y el módulo de acceso al medio. Ambos pueden interactuar luego de registrarse con la PC.

2.4.1 Funciones del módulo que implementa el equipo (PC)

init(nom): crea una instancia de un equipo conectado al canal. Se inicializa su variable nombre, dada como parámetro. Las demás variables se inicializan en valores nulos, ya que tomarán valor al registrarse los elementos correspondientes en el objeto PC. Estos son: el canal, la ubicación del equipo en el medio físico, y los componentes de la función de acceso al medio, driver, y el módulo de acceso al medio.
registrar_link_driver(ld): registra el driver. Invocada por este módulo.
get_link_driver(): devuelve la referencia al driver
registrar_am(accm): registra el módulo de control de acceso al medio invocado por este módulo.

get_am(): devuelve el la referencia al módulo de control de acceso al medio
get_nombre(): devuelve el nombre del equipo.
get_ubicación(): devuelve la ubicación del equipo en el vínculo de comunicación.
set_ubicación(ubi): determina la ubicación del equipo en el link. Invocado por el método conectar de link.
get_link(): obtiene una referencia al canal al que se encuentra conectado el equipo.
set_link(l): inicializa la variable que contiene la referencia al canal al que se encuentra conectado el equipo. Invocado por el método conectar del canal.
distancia(pc): devuelve la distancia en metros entre el equipo y el que le es pasado como parámetro.

3. Interacción entre los componentes

A continuación se describe brevemente cómo interactúan los diferentes componentes, canal con driver y driver con módulo de acceso al medio a efectos de facilitar la comprensión del funcionamiento del medioambiente implementado. Estas interacciones son generales pero puede ser necesario efectuar cambios en función de las características del protocolo de acceso al medio que se implemente.

3.1 Interacción vínculo de comunicación/driver

La interacción entre estas dos entidades se produce en dos sentidos: desde el driver al canal, para emitir señales especiales o datos, y desde el canal al driver para anunciar el estado de la señal y la recepción de información.

La interacción driver/canal se realiza a través de la invocación por parte del driver de la función **emitir** del canal, de la manera descripta anteriormente. Es una función genérica que solo deberá modificarse en el caso de incorporar nuevas señales a ser enviadas por el canal y reconocidas por el driver.

La interacción canal/driver tiene por objeto informar al driver acerca de la llegada de información (datos) o señales especiales que pueda reconocer el driver. Esta función es genérica y se adapta a gran parte de los posibles casos, debiendo modificarse sólo en el caso de agregar señales especiales a ser enviadas, diferentes a las que se han tenido en cuenta: datos, canal ocioso y colisión.

La interacción se basa en lo siguiente. el link informa al driver, cuando se produce un cambio, el estado de señal que existe en el punto del canal al cual el equipo correspondiente se halla conectado; esto se produce a través de la invocación a la función **evento** del driver por parte del link.

La función **evento**, que es general para cualquier protocolo de acceso al medio (sólo habrá que modificarla si se definen niveles de señal adicionales a datos, colisión y ocioso), invoca a su vez a **resolver_estado_link**, que se encarga de determinar el estado del canal, para informarlo en caso de cambio al módulo de acceso al medio, y de determinar la recepción completa de una señal especial o frame de datos para informarlo también al módulo de acceso al medio.

Por un lado, **evento** informa acerca del nivel de señal que se está recibiendo. Por simplicidad se puede suponer que el hecho de recibir durante un slot unidad de tiempo (el periodo al que hace referencia la función **evento**) el estado de señal es suficiente para informar al módulo de acceso al medio el cambio de estado en el canal, de esta manera, la función **resolver_estado_link** informará del cambio de estado al detectar un cambio en el valor que entregue **evento**.

Por otra parte, es necesario considerar que en ciertos casos no bastará con recibir un determinado nivel de señal durante un slot para informar al módulo de acceso al medio acerca de la recepción de un tipo de señal determinado o de un frame de datos. Un ejemplo claro de este caso lo constituye la recepción de datos: un frame de datos tiene una longitud tal que su duración excede el

tiempo de un slot, por lo tanto, no bastará con que el driver reciba una invocación a **evento** para que anuncie al módulo de control de acceso al medio acerca de la recepción de un frame, sino que el método **resolver_estado_link** deberá chequear que el estado datos se mantenga por el tiempo prefijado y que no sea interrumpido por niveles de señal diferentes, por ejemplo colisión en el caso en que en una parte del frame colisione con otra señal.

Resumiendo, el link invocará ante cambios a la función **evento** del driver para mantenerlo informado acerca del nivel de señal que existe en el canal. El driver, al ser informado, invocará a su función **resolver_estado_link**, que se encargará de determinar si efectivamente se ha recibido una señal definida para el módulo de acceso al medio, y en este caso, anunciarla.

3.2 Interacción driver-módulo de acceso al medio

Esta interacción, de manera similar a la descripta anteriormente, se produce en dos sentidos.

Desde el módulo de acceso al medio, para producir el envío de datos o señales especiales definidas por el protocolo de acceso al medio, y para eventualmente abortar la emisión de los mismos. Esta interacciones se dan a través de las funciones **emitir_señal**, **emitir_frame** y **abortar** del driver, y de la función **fin_envio** del módulo de acceso al medio.

Desde el driver al modulo de acceso al medio, para informar acerca de las condiciones del canal, del driver, de la llegada de señales significativas para el protocolo, o de datos. Esta interacción se da a través de un mecanismo simple de actualización de variables e interrupciones por parte del driver al módulo de acceso al medio.

Dicho módulo almacena el estado del link, el estado del driver, el último frame recibido y la última señal especial recibida. Estos elementos son genéricos, pero podrían ser cambiados según el protocolo y características del canal.

El mecanismo de anuncio consiste en lo siguiente: cuando el driver detecta la producción de algún evento, a través de su función **resolver_estado_link**, modifica la variable correspondiente en el módulo de control de acceso al medio (por ejemplo frame recibido, señal recibida, estado link o estado driver) e inmediatamente después (en el mismo tiempo de simulación), invoca a la función genérica **interrupcion** del método de acceso. Esta función recibe un parámetro identificando la causa de la interrupción (una de las cuatro citadas) en base a la cual el módulo de acceso al medio deberá tomar la decisión apropiada.

4. Ejemplos

A continuación se describen los aspectos más relevantes de dos protocolos de acceso al medio implementados y probados utilizando el soporte descripto en las secciones anteriores. Los objetivos de estas implementaciones fueron dos: 1-comprobar el funcionamiento y la adaptación del medioambiente implementado a diferentes casos reales, y la complejidad que implica este tipo de implementaciones y 2-evaluar el uso de estas implementaciones de protocolos específicos para la comprensión de los mismos y evaluación de sus performances a través de simulaciones.

Con respecto al primer objetivo, si bien la implementación de protocolos de acceso al medio resultó relativamente sencilla, se determinó que el tiempo que insume interiorizarse de la plataforma y realizar el desarrollo es mayor al que se asigna a los alumnos para la realización de trabajos prácticos comunes[2], por lo cual este aspecto (desarrollo de protocolos) queda limitado a ser propuesto en trabajos especiales ofrecidos por la cátedra.

El segundo objetivo será comprobado durante el próximo año con alumnos de la cátedra, siendo satisfactorios los resultados parciales obtenidos hasta el momento.

Los protocolos desarrollados fueron Aloha Puro y Carrier Sense con estrategia 1-persistente. En ambos casos se modifican algunas funciones del acceso al medio:

[2] El curso es de un cuatrimestre de duración, en cuarto año de la carrera, e incluye los niveles de acceso al medio, de red y de transporte.

1-**start**; para inicialización de las variables propias del protocolo, 2-**interrupcion**, para informar al resto del código los eventos correspondientes anunciados por el driver y en base a ellos ejecutar el procedimiento correspondiente y 3-**fin_envio**, para tomar acciones cuando se termina de enviar un bloque.

De manera similar, se modifica la función del driver **resolver_estado_link**, que dependerá de la capacidad del medio de transmisión para enviar y recibir determinados estados de señal.

En las secciones siguientes se describen las modificaciones necesarias en el módulo de acceso al medio. La información más detallada y el código de los ejemplos pueden solicitarse por e-mail al autor.

4.1 Protocolo Aloha puro

El protocolo Aloha puro fue desarrollado teniendo como base el soporte descripto, aunque por sus características no lo utiliza en su totalidad.

Del canal de comunicaciones, sólo utiliza la capacidad de transmisión de frames de datos, y no de señales de colisión o especiales, mientras que del driver sólo utiliza la capacidad de envío de frames de datos y el anuncio de recepción de frame.

De acuerdo al protocolo Aloha, un equipo envía un frame sin realizar un testeo previo del canal. Luego espera un tiempo para recibir el asentimiento para el frame y en caso de no recibirlo lo retransmite utilizando un mecanismo exponential backoff.

El método **enviar** de Aloha tiene como consecuencia que el bloque a enviar, entregado por la aplicación, sea puesto en la cola de envío (FIFO), y se active a la función de transmisión mediante el método transmitir. Esta función solicitará al driver, si éste está ocioso, el envío del primer frame de la cola. La función enviar puede ser también invocada por el módulo de acceso al medio, para generar el asentimiento correspondiente a un bloque. Tener en cuenta que este asentimiento podría ser generado directamente por la aplicación; aquí estamos suponiendo por simplicidad que el asentimiento lo genera el módulo de acceso al medio.

Respecto al envío de frames, debe tenerse en cuenta que habrá frames para los cuales se reintenta la transmisión (los frames de datos entregados por la aplicación) y habrá otros frames para los cuales no se hace ese reintento, los frames que llevan el asentimiento. Por esta razón, la información que se almacena en la cola de frames a enviar consiste en la identificación del frame, su longitud y una indicación acerca de si deben ser reenviados o no.

La función **transmitir** se encarga de tomar el primer blque de la cola y entregarlo al canal (la transmisión). Al enviar el frame, si se trata de un asentimiento, sólo lo elimina de la cola; en caso de que se trate de un frame de datos, lo mantiene en la cola y arranca un timer para esperar por el asentimiento; si éste no llega, incrementará el número de reintento y retransmitirá el frame de acuerdo a una estrategia exponential backoff.

Las condiciones para que un frame sea transmitido son que el driver este en estado ocioso y que no se este esperando el asentimiento para el primer frame de la cola (el hecho de que se ordenara la transmisión de un bloque que recién fue enviado, seria un error en el método de control de acceso).

Un aspecto a tener en cuenta en la parte de transmisión es el siguiente: cuando se ha enviado un frame y se esta esperando el asentimiento, puede producirse el timeout. Si se produce, indicará que se genere un tiempo al azar de acuerdo a exponential backoff, y luego se transmita nuevamente el frame. La función timeout generará entonces el tiempo de espera y un evento a producirse en ese tiempo que consistirá en la retransmisión del bloque (esto se hace en el método reintento). Durante el período en que se produjo el timeout y se reintenta la transmisión, no puede enviarse ningún otro frame, ya que esta en cola el fallido (además enviar otro frame seria introducir más carga en el canal). Pero se pueden procesar los frames que eventualmente llegan por el canal. Para simplificar

el protocolo, en este periodo de tiempo se ignoran posibles asentimientos retrasados que puedan llegar para el bloque que espera ser reenviado. Esto es indicado por la variable **ignorar_ok**.

Cuando es invocada la función de recepción de frame, si éste es un frame de datos se chequea que sea para el equipo y en este caso se encola el correspondiente asentimiento. En el caso de que se trate de un asentimiento para este equipo y se esté esperándolo, se cancela el evento de retransmisión y se da por bien recibido el frame. Se invoca a **transmitir** para reactivar la transmisión.

Resumiendo, as funciones que se utilizan del canal y del driver son:

1- Funciones relativas a la transmisión de frames:
  1.1- **enviar_frame**: solicita al link driver que envíe un frame por el canal
  1.2- **fin_envio**: el link driver anuncia que finalizo el envío del frame en curso
2- Funciones relativas a la recepción
  2.1- **frame_recibido**; el link driver entrega un frame recibido correctamente

4.2 Protocolo Carrier Sense

Se asume un protocolo CSMA 1-persistente, es decir, se testea canal continuamente[3] hasta que se lo detecta libre y en ese momento se envía la información al canal.

La operación del protocolo se resume a continuación:

1- Debido a que en carrier sense no hay manera de determinar si un bloque colisiona, ya que no se cuenta con detección de colisión, cada vez que se transmite un bloque entregado por la aplicación (tipo F), se lo retiene en el buffer hasta que se recibe el asentimiento correspondiente. Si pasado un tiempo máximo prefijado no se recibe el asentimiento, se reenvía el bloque. Este tiempo máximo debe ser por lo menos dos tiempos de transmisión más dos demoras de propagación. En la práctica se configura un tiempo mayor debido a las esperas por transmisión a que puede verse sometido el bloque de asentimiento (puede haber bloques en cola previos a él, o bien él mismo puede experimentar demoras de transmisión si el canal está ocupado).
2- La aplicación genera bloques de datos a enviar (tipo F) y los entrega al nivel de acceso al medio invocando a la función enviar
3- El nivel de acceso al medio mantiene una cola de bloques a enviar. En ella habrá bloques de tipo F (datos) y bloques de tipo OK (asentimientos). La cola es FIFO, sin distinciones respecto al tipo de frame.
4- El nivel de acceso al medio, al encargarse de enviar el asentimiento para cada bloque tipo F recibido, genera los bloques tipo OK correspondientes. Lo hace invocando a su propia función enviar, que encolará el bloque con el OK.
5- El tratamiento para los bloques de datos y los de asentimientos es diferente. Un bloque de datos, cuando es enviado al canal para su transmisión, se conserva en el buffer hasta que se obtiene su OK o se lo descarta después de varios intentos de retransmisión[4]. Cuando se logra transmitirlo (se recibe el OK), se lo elimina del buffer. Un bloque de asentimiento se envía una única vez, no esperándose un asentimiento sobre él. Esto implica que cuando se transmite por el canal un bloque de este tipo, inmediatamente se libera el buffer y no se configura un timer para su eventual retransmisión.
6- Puede ocurrir que para un bloque de datos enviado venza el tiempo de espera por el OK, ocurrido esto, se determina un nuevo tiempo de envío utilizando exponential backoff. Podría ocurrir que el OK llegue atrasado, luego de que se programó la retransmisión del bloque. En este caso, se decide

[3] Para simplificar la programación del ejemplo, se testea el canal cada un intervalo de tiempo igual a una demora de propagación punta a punta. Una mejora que puede introducirse es el uso de la interrupción proporcionada por el driver al cambiar el estado del canal. Cuando esta se produce, se debería invocar a una función que envíe el bloque al canal. Esta mejora se propone como trabajo practico.

[4] No implementado en el ejemplo.

que estos OK retrasados se ignorarán. Para esto se utiliza la variable ignorar_ok. Esta variable se pone en SI luego de que se programa una retransmisión, y se pone en NO inmediatamente después que un bloque de datos es transmitido (o retransmitido).

7- Para evitar posibles errores se utiliza la variable esperando_ok. Esta variable se configura con el valor SI cuando se ha enviado un bloque de datos y se esta esperando el OK. Si la función de transmisión fuera invocada en estas circunstancias (por ejemplo cuando la aplicación invoca a enviar), el driver estaría idle, y posiblemente se desencadenara la retransmisión (indeseada) del bloque en cuestión. Para evitar estas situaciones se chequea el valor de la variable esperando_ok.

5. Conclusiones y trabajo relacionado

El soporte desarrollado ha cumplido satisfactoriamente con los objetivos fijados. La interfaz ofrecida a los usuarios mostró ser clara y simple de utilizar, en sus dos casos de aplicación: el más elaborado, que consiste en el desarrollo de un protocolo de acceso al medio por parte del usuario, y el más simple, que consiste en la configuración de los parámetros de funcionamiento, para un protocolo ya dado, con el objeto de comprender su funcionamiento y/o evaluar su comportamiento.

Si bien hasta el momento no se ha implementado un segundo tipo de canal (además del canal broadcast de acceso múltiple), las modificaciones a realizar en el código del driver son mínimas.

En el futuro se trabajara sobre este aspecto, tratando de implementar un soporte de comunicación wireless

Actualmente, se esta desarrollando un conjunto de protocolos de acceso al medio que corren sobre el tipo de canal implementado, con el objeto de incorporar esta herramienta a los trabajos prácticos de protocolos de acceso al medio, en la cátedra correspondiente. En una primera etapa los ejercicios presentados a los alumnos consistirán en evaluar la performance de diferentes protocolos de acceso al medio a través de las simulaciones correspondientes.

Tanto el código correspondiente al soporte desarrollado como los ejemplos provistos a los alumnos pueden solicitarse al autor vía email.

6. Bibliografía

[Fall, 2000] K. Fall (ed), “Ns Notes and Documentation” VINT Project, UC Berkeley, LBL, USC/ISI, Xerox PARC, March 2000.

[Halsall, 1992], “Data Communications, Computer Networks and Open Systems”, Halsall, F., Addison-Wesley, 1992.

[Rigotti, 2003] “Soporte de programación para protocolos del nivel 2 OSI/ISO”, IX Congreso Argentino de Ciencias de la Computacion (CACIC). La Plata, octubre de 2003.

[Osterhout,1994] ”Tcl and the Tk Toolkit”, John K. Osterhout, Addison Wesley, 1994.

[Rose, 1990] “The Open Book. A Practical Perspective on OSI”, Marshall Rose, Prentice Hall, 1990.

[Tanenbaum, 2004] “Computer Networks” 4th edition, Tanenbaum, A., Prentice Hall, 2004.