Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

"MODELADO Y ANÁLISIS DEL MECANISMO DE RETRANSMISIÓN DE BLOQUES ARQ EN LA CAPA MAC DEL IEEE 802.16 UTILIZANDO REDES DE PETRI COLOREADAS (CPNs)"

Trabajo Especial de Grado presentado por el bachiller:
Luisana Margarita Contreras Pernía
Ante la ilustre Universidad Central de Venezuela

Para optar al título de:
Licenciada en Computación

Tutores:
Profa. María Elena Villapol
Profa. Ana Morales

Caracas, 10 de Abril de 2014

ACTA

Quienes suscriben, miembros del Jurado designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por la Bachiller Luisana Margarita Contreras Pernía C.I. V – 17.926.989, con el título: "Modelado y análisis del mecanismo de retransmisión de bloques ARQ en la capa MAC del IEEE 802.16 utilizando Redes de Petri Coloreadas (CPNs)", a los fines de optar al título de Licenciada en Computación, dejan constancia de lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del jurado, se fijó el día a las, para que el autor lo defendiera en forma pública, lo que se hizo en el en la Facultad de Ciencias, mediante una presentación oral de su contenido, luego de lo cual respondió las preguntas formuladas. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con la nota de 20 puntos.

En Fe de lo cual se levanta la presente Acta, en Caracas a los 10 días del mes de Abril de 2014.

_____	_____
Profa. María Elena Villapol (Tutor)	Profa. Ana Morales (Tutor)
_____	_____
Prof. Robinson Rivas (Jurado)	Prof. Walter Hernández (Jurado)

DEDICATORIA

A Dios por darme la fortaleza y confianza, para ser perseverante y constante en los momentos difíciles.

A mi madre Alis Pernía, por ser mi ejemplo de lucha y de trabajo.

AGRADECIMIENTOS

Agradezco a Dios por darme la fortaleza para perseverar en mis objetivos a pesar de las circunstancias. A mis padres Alis Pernía y Luis Contreras García, mi hermano Luis Contreras Pernía por su apoyo en los años de carrera. Agradezco a Jorge Herrero por su constante apoyo, ayuda cuando estudiábamos juntos y el amor que me ha brindado durante estos años. A mis tutoras Profa. María Elena Villapol y Profa. Ana Morales por todos los conocimientos que me brindaron, por el interés hacia este trabajo y la amabilidad y apoyo que recibí durante el desarrollo del mismo. Al personal del laboratorio de ICARO por permitirme utilizar sus instalaciones para realizar mi trabajo. A todos los que de alguna forma contribuyeron para el logro de esta meta. Gracias a Todos.

-Luisana Contreras-

CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

Durante las última década las expectativas y realidades en torno al acceso de última milla de banda ancha por medio inalámbrico han creado una ventana de oportunidades para fabricantes, integradores y prestadores de servicios. Hoy por hoy uno de los términos asociados con esta tecnología es IEEE 802.16. Los lineamientos de este estándar están dirigidos a la prestación de gran ancho de banda de manera inalámbrica para voz y datos para el ámbito residencial como empresarial. Sin embargo, la especificación del estándar IEEE 802.16 proporciona una descripción narrativa del protocolo sin ningún uso de técnicas formales. Así, algunas partes del documento pueden ser ambiguas, difíciles de entender, e imprecisas, especialmente en cuanto al mecanismo de retransmisión de bloques ARQ, que forma parte de la funcionalidad de la capa MAC. La implementación de este mecanismo es la única referencia para una validación del mismo.

Este trabajo modela el mecanismo de retransmisión de bloques ARQ usando una técnica formal conocida como *Redes de Petri Coloreadas (CPNs).* Los pasos siguientes resumen el proceso de verificación de la operación ARQ. Primero, la especificación de la operación ARQ derivada de la descripción del estándar. En segundo lugar, se modela la operación ARQ usando CPNs. Por último, se realiza el análisis basado en el método del gráfico de estado, cuyos resultados son comparados contra las propiedades de las CPNs. El resultado de este trabajo muestra que no hay ningún problema funcional significativo y la operación ARQ funciona según lo esperado, bajo las asunciones de modelado y de análisis.

Palabras Claves: Redes de Petri, CPNs, operación ARQ, gráfico de estado, capa MAC

CAPÍTULO 1: INTRODUCCIÓN

1.1 Contexto de la Investigación

Actualmente el campo de las redes inalámbricas ha evolucionado a pasos agigantados. Uno de los principales problemas de las redes inalámbricas es el crecimiento o expansión del área de servicio para las estaciones que hacen uso de la red. La necesidad de cubrir mayores distancias dio origen a un conjunto de tecnologías y estándares que han permitido que el acceso a la información a través del uso de medios inalámbricos sea cada vez mejor, con altas velocidades, seguridad y amplio alcance. Entre éstas tecnologías están las *Red Inalámbrica de Área Amplia* (*WWAN*), representadas principalmente por la Red de Telefonía Celular, y las *Red Inalámbrica de Área Metropolitana* (*WMAN*), dentro de las que se encuentran las tecnologías desarrolladas bajo el estándar IEEE 802.16 [11], la cual permite mayor velocidad de transmisión, cobertura y calidad de la señal respecto a otras tecnologías como IEEE 802.11. La primera versión del estándar fue completada en octubre de 2000 y publicada el 8 de abril de 2002 [12]. Este define la interfaz Física y la capa de Control de Acceso al Medio (MAC) para redes WMAN, con la intención de proveer banda ancha inalámbrica para servicios de voz y datos con usos residenciales y empresariales. La primera versión solo fue considerada para usuarios fijos [12]. En 2009 se define la interfaz aérea para acceso a un punto fijo de banda ancha (paquete acumulativo de 802.16-2004 ,802.16e, 802.16f, 802.16g [12]). En este estándar se enfoca el presente trabajo, siendo esta tecnología una potente solución a las necesidades de redes de acceso inalámbricas de banda ancha, de amplia cobertura y elevado rendimiento.

La cualidad más importante de un sistema es que tenga un funcionamiento correcto, es decir, que éste exhiba cierto comportamiento o propiedades cualitativas, las cuales determinan el éxito de un sistema. Muchos de los estándares y tecnologías desarrollados hacen muy poco uso de métodos formales para especificar sus propiedades y describir los protocolos de comunicación, siendo estos documentos, muchas veces ambiguos, e imprecisos [5].

En algunos campos de la ciencia los fenómenos no son estudiados directamente sino a través de un modelo o simulación. Un modelo es una representación generalmente en términos matemáticos. Una Red de Petri es una representación matemática de un sistema distribuido discreto y representa una generalización de la teoría de autómatas que permite expresar eventos concurrentes.

El formalismo de modelado de las Redes de Petri proporcionan técnicas para soportar el diseño y mantenimiento de los protocolos de comunicación [5], igualmente permiten la utilización de una conveniente visualización gráfica de los modelos de sistemas, y ayuda a los diseñadores de sistemas y desarrolladores de estándares, y protocolos a asegurar y analizar las propiedades de corrección y rendimiento [5] de los mismos. Así mismo, las Redes de Petri Coloreadas (CPNs) son un lenguaje gráfico que modela eventos discretos y sistemas concurrentes para analizar sus propiedades, combinándose las capacidades de las Redes de Petri, con las bondades de los lenguajes de programación de alto nivel. Las CPNs incorporan algunas definiciones, tales como tipos de datos y el procesamiento de valores de datos encontrados en los lenguajes de programación, así como primitivas básicas para el modelado de la concurrencia, comunicación y sincronización.

Se utilizó esta técnica para modelar el proceso de operación del mecanismo de retrasmisión de bloques ARQ, del estándar IEEE 802.16.

1.2 Planteamiento del Problema

El IEEE 802.16 es un estándar que propone una solución tecnológica para conexiones inalámbrica con gran ancho de banda. Algunos aspectos de la tecnología, tal como lo es el mecanismo de retransmisión de bloques ARQ, encargado de retransmitir los paquetes que no llegaron correctamente a su destino, es más complicado que la transmisión basada en tramas que utilizan la mayoría de los protocolos de la capa MAC de otras tecnologías, debido a que se utiliza una unidad de datos independiente denominada bloque ARQ, y los procedimientos realizados por la entidad emisora y receptora involucran una serie de temporizadores cuyo funcionamiento es complejo. En el estándar IEEE 802.16, se presenta una descripción narrativa del procedimiento soportada por un diagrama de estado. Usualmente las descripciones narrativas son pocos claras y ambiguas. Lo cual conlleva a que las implementaciones basadas en este estándar puedan contener errores o no funcionar correctamente en ciertas situaciones. En este trabajo se pretende dar respuesta a la siguiente interrogante.

¿El procedimiento de operación ARQ de la capa MAC IEEE 802.16 funciona de acuerdo a lo establecido en la especificación?

1.3 Objetivos de la investigación

En la siguiente sección se presentan los objetivos que se plantearon alcanzar con el trabajo de investigación presentado, así como los objetivos específicos requeridos para cumplir con tal fin.

1.3.1 Objetivo General

El objetivo de este trabajo se define a continuación:

Modelar y analizar el procedimiento de operación ARQ de la capa MAC del IEEE 802.16 usando las Redes de Petri Coloreadas.

1.3.2 Objetivos Específicos

Los objetivos específicos de este trabajo son:

1. Determinar las características del procedimiento de operación ARQ que se van a modelar.
2. Modelar las características del procedimiento de operación ARQ usando Redes de Petri Coloreadas.
3. Definir parámetros iniciales para la realización de simulaciones.
4. Validar el modelo en función de las propiedades básicas de las CPNs.
5. Analizar los resultados obtenidos.

1.4 Justificación

IEEE 802.16 es una tecnología en avance que ya se ha implementado como una solución de última milla en varios lugares alrededor del mundo. Sus múltiples características y flexibilidad de configuración permiten una gran diversidad de opciones a los administradores de estas redes. Esta, similarmente a otras tecnologías, no ha sido verificada utilizando métodos formales antes de ser implementadas. Siendo muchas veces el único mecanismo de validación la interoperabilidad entre implementaciones de distintos fabricantes. Lo dicho anteriormente justifica el hecho que ciertos mecanismos de

la tecnología IEEE 802.16, como la operación ARQ, sean modelados y verificados, usando técnicas como las Redes de Petri Coloreadas.

1.5 Estructura del Trabajo de Investigación

A fin de abarcar todos los aspectos relacionados a la investigación, el presente trabajo se encuentra estructurado como sigue:

Capítulo 2: Este capítulo proporciona una descripción detallada del estándar IEEE 802.16, haciendo un mayor énfasis a los procesos relacionados con el mecanismo de retransmisión de bloques ARQ a nivel de la subcapa CPS de la capa MAC.

Capítulo 3: Se describen a las Redes de Petri y las Redes de Petri Coloreadas (CPNs) como método formal para el modelado de protocolo

Capítulo 4: Se presenta la metodología para el modelado del mecanismo de retransmisión de bloques ARQ en la capa MAC IEEE 802.16.

Capítulo 5: Se presenta el modelo, especificación y descripción del mecanismo de retransmisión de bloques ARQ en la capa MAC IEEE 802.16.

Capítulo 6: Se presenta el análisis del modelo del mecanismo de retransmisión de bloques ARQ en la capa MAC IEEE 802.16.

Capítulo 7: Concluye este trabajo con las contribuciones del mismo, limitaciones encontradas e ideas para trabajos futuros.

CAPÍTULO 2: IEEE 802.16 (WIMAX)

WiMAX cuyas siglas significan interoperabilidad mundial para acceso por microondas, es la marca que certifica que un producto implementa los estándares de acceso inalámbrico IEEE 802.16 [3][7]. Es una tecnología inalámbrica capaz de proporcionar conectividad de banda ancha a hogares, empresas y usuarios móviles. WiMAX es similar a WiFi, pero puede ofrecer un ancho de banda mucho más grande, encriptación más fuerte, mayor capacidad de usuarios y mejora el rendimiento en largas distancias. Estos estándares permiten conexiones de velocidades similares al ADSL o al cable módem, sin cables, y hasta una distancia de 50-60 km.

A pesar de que el proyecto para la creación de un nuevo estándar se gestó años atrás en el IEEE, no fue hasta abril de 2002 que la primera versión del mismo 802.16a[11] se publicó, y se refería a enlaces fijos de radio con visión directa entre emisor y receptor, pensada para cubrir grandes áreas de servicio utilizando eficientemente varias frecuencias dentro de la banda de 10 a 66 GHz

Un año más tarde, en marzo de 2003, se ratificó una nueva versión, el 802.16a [11], y fue entonces cuando WiMAX, como una tecnología de banda ancha inalámbrica, empezó a cobrar relevancia. También se pensó para enlaces fijos, pero llega a extender el rango alcanzado desde 40 a 70 kilómetros, operando en la banda de 2 a 11 GHz, parte del cual es de uso común y no requiere licencia para su operación. Es válido para topologías punto a multipunto y, opcionalmente, para redes en malla, y no requiere línea de visión directa. Emplea las bandas de 3,5 GHz y 10,5 GHZ, válidas internacionalmente, que requieren licencia (2,5-2,7 GHz en Estados Unidos), y las de 2,4 GHz y 5,725-5,825 GHz que son de uso común y no requieren disponer de licencia alguna.

Un aspecto importante del estándar 802.16x es que define un nivel MAC que soporta múltiples enlaces físicos.

Actualmente dentro del estándar 802.16, existen dos variantes. Una variante es el acceso fijo (802.16d), en el que se establece un enlace radio entre la estación base y un equipo de usuario situado en el domicilio del usuario. Para el entorno fijo, las velocidades teóricas máximas que se pueden obtener son de 70 Mbps con un ancho de banda de 20 MHz. Sin embargo, en entornos reales se han conseguido velocidades de 20 Mbps con radios de celda de hasta 6 Km, ancho de banda que es compartido por todos los usuarios de la celda. Fue publicado en julio de 2004.

La otra variante es la movilidad completa (802.16e), que permite el desplazamiento del usuario de un modo similar al que se puede dar en GSM/UMTS. Fue publicado en diciembre de 2005.

2.1 Características WiMAX

A continuación se muestran las características de WiMAX:

- El estándar 802.16 puede alcanzar una velocidad de comunicación de más de 100 Mbit/s en un canal con un ancho de banda de 28 MHz (en la banda de 10 a 66 GHz), mientras que el 802.16a puede llegar a los 70 Mbit/s, operando en un rango de frecuencias más bajo (11 GHz).
- Flexibilidad y extensibilidad con una MAC común.
- Distancias de hasta 80 kilómetros, con antenas multi-direccionales y de alta ganancia.
- Modularidad
- Facilidades para añadir más canales, dependiendo de la regulación de cada país.

- Anchos de banda configurables y no cerrados, sujeto a la relación de espectro.
- Permite dividir el canal de comunicación en pequeñas subportadoras.
- Topología de múltiples redes
- El radio del área de cobertura estimada es de 15 km en condiciones sin línea de vista (NLOS) y de 50 km en condiciones con línea de vista (LOS).
- Soporte para diferentes subcapas de convergencia a nivel MAC
- Políticas de retransmisión flexibles
- Privacidad
- QoS Integrada
- Soporte para Duplexación por División de Tiempo (TDD) y Duplexación por División de Frecuencia (FDD)

2.2 Fundamentos Básicos de la capa MAC de IEEE 802.16

El estándar IEEE 802.16 especifica la interfaz aire de los Sistemas de Acceso Inalámbricos de Banda Ancha punto-multipunto fijo para proveer múltiples servicios, incluyendo las capas MAC y Física.

La estación base tiene el control del ancho de banda de cada subscriptor. La capa MAC tiene paquetes de la capa superior, los cuales se denominan Unidades de Datos de Servicio de la capa MAC (MSDUs).

2.2.1 Componentes Básicos de la Arquitectura IEEE 802.16

IEEE 802.16 está compuesto de varias estaciones "cliente" o "suscriptor", conectada a una estación base; switches de interconexión y bases de datos. Los suscriptores reciben y convierten la señal de radio en información para el usuario, mientras que las estaciones base representan las estaciones o torres de radio transmisión. Los switches para interconexión y líneas de transmisión transfieren las señales entre las estaciones base y

otros sistemas (tales como la red pública de telefonía o bien Internet).Las bases de datos son colecciones de información que están interrelacionadas y almacenadas en memoria (discos rígidos, computadores y otros medios de almacenamiento de datos). El IEEE 802.16 contiene diversas bases de datos que guardan información de los suscriptores, configuración de equipos, listas de especificación y códigos de seguridad. A continuación se definen los componentes.

2.2.1.1 Estación Base (BS)

La estación base provee el servicio de control, gestión y conectividad a la estación suscriptora y posee las instancias de las capas MAC y PHY (Física).

2.2.1.2 Estación Suscriptora (SS)

La estación subscriptora provee conectividad entre un equipo suscriptor y una estación base y posee las instancias de las capas MAC y PHY. En la Figura 2.1 se muestran los distintos tipos de dispositivos de acceso IEEE 802.16.

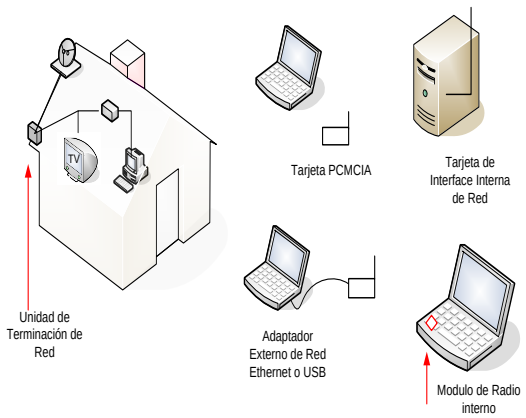

Figura 2.1: Componentes de la Estación Subscriptora.

2.2.1.3 Estación Móvil (MS)

La estación móvil realiza lo siguiente:

- Gestión de movilidad
- Handoff (transferencia del servicio de una estación base a otra cuando la calidad del enlace es insuficiente).
- Conservación de las baterías

2.3 Topologías

El estándar IEEE 802.16 permite la conformación de los siguientes tipos de topologías:

2.3.1 Punto a Punto (PTP)

Punto a Punto es el proceso de transferir información de un dispositivo (o punto) a otro dispositivo (solo punto receptor). El sistema IEEE 802.16 puede usar la comunicación PTP para enlaces de alta velocidad para aplicaciones de backhaul (interconexión de sistemas)

2.3.2 Punto a Multipunto (PMP)

La comunicación Punto a Multipunto transfiere la información de un dispositivo a múltiples dispositivos. El sistema IEEE 802.16 puede usar PMP a estaciones base para proveer acceso de banda ancha a múltiples usuarios por estación base en la banda de 10 a 66 GHz. A estas frecuencias la transmisión debe realizarse con visión directa. La capa de control de acceso al medio (MAC), soporta muchas especificaciones diferentes de capa Física (PHY), para bandas de frecuencia reguladas y sin regular.

La capa MAC de 802.16 cada estación base distribuye dinámicamente el ancho de banda disponible en los enlaces ascendente y descendente entre las estaciones de abonado mediante Acceso Múltiple por División en el Tiempo (TDMA).

2.3.3 Malla

Una red de malla es un sistema de comunicación donde cada dispositivo es interconectado a múltiples nodos (puntos de conexión) en la red donde los paquetes de datos pueden viajar a través de caminos alternos para alcanzar su destino.

Algunos o todos de los recursos de los sistemas IEEE 802.16 pueden ser configurados para proveer servicios de red malla para poder dar acceso en situaciones donde no hay visión directa entre el transmisor y el receptor a más baja frecuencia. Opera en las frecuencias de 11 y 2 GHz, utilizando la técnica de Multiplicación Ortogonal en Frecuencia (OFDM).

2.4 Modelo de Referencia

IEEE 802.16 define dos componentes principales de los sistemas móviles inalámbricos: El plano de control/datos y el plano de gestión. Con el fin de garantizar interoperabilidad entre los distintos estándares, el estándar IEEE 802.16 define una capa MAC capaz de soportar distintas interfaces de nivel PHY, estos se comunican a través de Punto de acceso al servicio (PHY-SAP) con el nivel MAC.

El estándar IEEE 802.16 el nivel MAC comprende tres subcapas: La Subcapa de Convergencia de Servicio Específico (CS), la Subcapa de Parte Común de Control de Acceso al Medio (CPS), y la Subcapa de Privacidad. En la Figura 2.2 se puede apreciar el modelo de referencia.

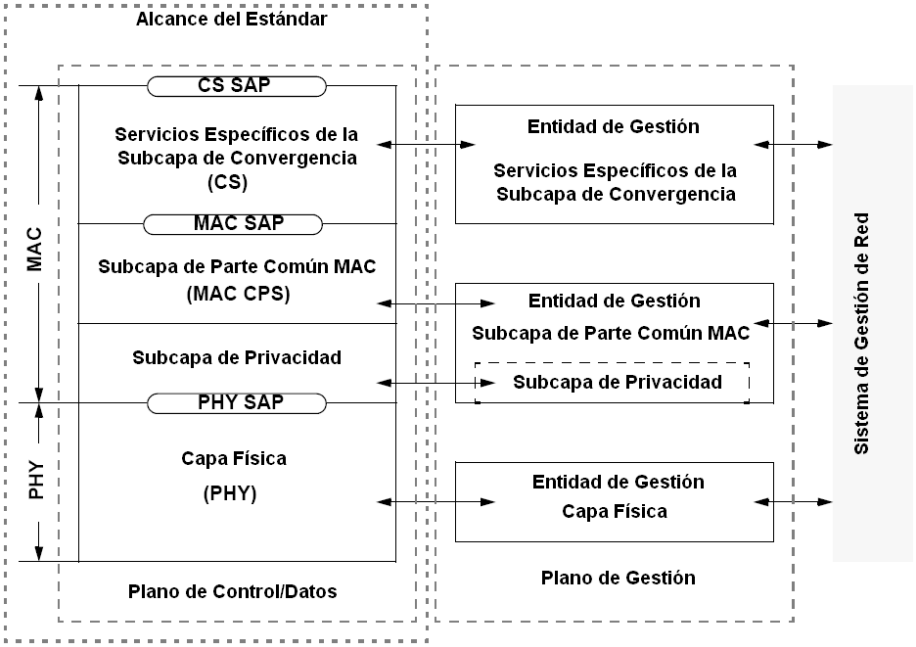

Figura 2.2: Modelo de referencia IEEE 802.16.

A nivel de la capa MAC se llevan a cabo tareas tales como: dar formato a los paquetes de usuario (PDUs y SDUs), demultiplexar cabeceras de paquetes MAC, fragmentación de MPDUs, empaquetamiento de MSDUs, concatenación de múltiples MPDUs, retransmisiones de paquetes, encriptación, planificación de los enlaces y gestión del intercambio de mensajes de información y control. Todas estas tareas son realizadas a través de las tres subcapas en las cuales está conformada la capa MAC, ellas se describen en las siguientes secciones:

2.4.1 Subcapa de Convergencia (CS)

La subcapa de convergencia realiza toda la transformación y el mapeo de los datos recibidos de niveles superiores a través del punto de acceso al servicio (SAP) para formar las unidades de datos del protocolo (MAC PDU) enviadas a la MAC CPS, esto incluye

clasificación de las Unidades de Datos de Servicio (SDUs) externas y asociar éstas con el apropiado flujo de servicios (SFID) e Identificador de conexión (CID). La subcapa CS puede incluir también algunas funciones como la supresión de cabeceras. Además, se prevén múltiples especificaciones de la CS como interfaces con los distintos protocolos. La configuración interna de la CS *payload* concierne solo a la subcapa CS, y no se le pide a la MAC que comprenda su configuración o analice su información.

2.4.2 Subcapa de Parte Común (MAC CPS)

Debido a que la MAC está orientada a la conexión. La MAC CPS provee las funcionalidades bases del sistema de acceso, asignación de ancho de banda, establecimiento y mantenimiento de las conexiones. También clasifica los datos en una conexión MAC particular y recibe los datos de las distintas CS a través del MAC SAP. Además están previstos mecanismos para diferenciar la Calidad de Servicio (QoS) en función de las distintas necesidades ligadas a las diferentes aplicaciones.

Los enlaces descendentes (DownLink: BS->SS) operan en forma de punto a multipunto, es decir, todas las estaciones reciben la misma transmisión. La BS es la única que genera tráfico en ésta dirección y transmite sin tener que coordinarse con las estaciones.

La BS realiza difusión a todas las estaciones dentro de su área de cobertura y frecuencia, y las estaciones comprueban el campo de dirección destino de los mensajes recibidos y retienen sólo los dirigidos a ellas.

En los enlaces ascendentes (UpLink: SS->BS), las SS comparten los enlaces de subida hacia las BS según la demanda. Dependiendo de la clase de servicio utilizado, la SS puede asignarse el derecho a transmitir o la BS puede conceder el derecho a transmitir luego de recibir un mensaje de solicitud de parte del usuario.

Los mensajes pueden ser enviados de tres formas: *unicast*, *multicast* (sólo mensajes de control y distribución de video) y *broadcast* (mensajes de gestión).

Dentro de cada sector los usuarios (SS) se adhieren a un protocolo de transmisión que controla la contención entre ellos y habilita los servicios para tolerar los requerimientos de retardo y ancho de banda de cada aplicación de usuario.

2.4.3 Subcapa de Privacidad

La subcapa de privacidad se encarga de proveer autenticación, intercambio de claves seguras y encriptación.

2.4.4 Capa Física (PHY)

A través de la capa física la MAC CPS transmite los datos, el control de la capa física, y las estadísticas al nivel físico**.** El estándar ha sido pensado para desarrollarse como un conjunto de interfaces radio basadas sobre un único protocolo MAC, pero que incluye múltiples especificaciones de PHY, cada una adaptada a un rango de frecuencias particular y a una aplicación en concreto. IEEE 802.16 soporta bandas licenciadas y no licenciadas entre 2 y 66 GHz. La capa física utiliza multiplexación por división de frecuencias ortogonales (OFDM) y consiste en una técnica de comunicación que divide un canal de frecuencia, en un número determinado de bandas de frecuencias espaciadas de forma equivalente, en cada banda se transmite una subportadora que transporta una porción de la información del usuario. Cada subportadora es ortogonal al resto, dándole el nombre a esta técnica de multiplexación por división de frecuencia.

Se definen distintos tipos de canal para operar en distintas bases (ej. 3.5, 5, 10 MHz). IEEE 802.16 soporta una modulación adaptativa, equilibrando eficazmente las diversas tasas de datos con la calidad de los enlaces. El método de la modulación puede

ser ajustado casi instantáneamente para hacer óptima la transferencia de datos. El estándar permite tanto la multiplicación por división de frecuencias (FDD) como la multiplicación por división de tiempo (TDD). Soporta requerimiento de repetición automática (ARQ) y calidad de servicio (QoS).

2.5 Cabeceras MAC

2.5.1 Cabecera MAC Stand-Alone

La cabecera MAC Stand-Alone se utiliza para transportar información de control compacta o de señalización entre una SS y una BS. Estas cabeceras no se pueden utilizar para encapsular carga útil ya que son autónomas y se utilizan para un propósito específico. Estas cabeceras solo tienen relevancia en los enlaces UL. Por lo tanto, solo la BS es requerida para procesar estas cabeceras. Dentro de las cabeceras se utiliza un bit para definir el tipo de cabecera.

Adicionalmente, IEEE 802.16 define dos cabeceras MAC principales para los enlaces DL, la primera es la cabecera MAC genérica que comienza con cada MAC PDU conteniendo aquellos mensajes de gestión MAC o datos de la CS. La segunda es la cabecera de solicitud de BW utilizada para solicitar ancho de banda adicional. El campo que distingue una cabecera de otra es (HT) tipo de cabecera. Si tiene el valor de cero es genérica y si es uno es de solicitud. La cabecera Stand-Alone se posiciona dentro de la categoría HT=1. En la Figura 2.3 se puede visualizar el formato de la cabecera MAC genérica.

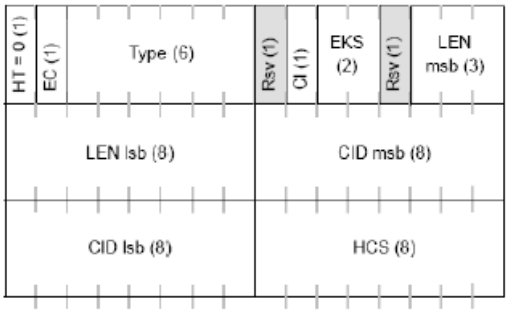

Figura 2.3: Cabecera MAC Genérica.

La carga útil de una MPDU puede estar conformada por otra sub-cabeceras, mensajes de gestión, cargas útiles adicionales, MSDUs recibidas desde una CS o información de relleno. Las cabeceras inician con un campo tipo, seguido de un campo (EC) para indicar si el control de la encriptación está presente o no.

El tipo de carga útil llevado dentro de una MPDU está indicado por el CID y el campo tipo. Si el CID es un CID de gestión, la carga útil puede consistir de uno o más mensajes de gestión MAC. Si el CID es un CID de transporte, la carga útil puede consistir de una o más MSDUs o fragmentos de MSDU.

Si se incluyen sub-cabeceras opcionales, serán consideradas como parte de la útil y no estarían protegidas por la Secuencia de Chequeo de Cabecera (HCS). El campo tipo tiene 6 bits y cada bit identifica a un tipo de sub-cabecera opcional o a un tipo de carga útil opcional.

2.5.2 Subcabeceras MAC IEEE 802.16

Las subcabeceras MAC son utilizadas para extender las funcionalidades básicas soportadas por la cabecera MAC genérica. Entre ellas tenemos: subcabeceras de

fragmentación (FSH), subcabecera de empaquetamiento (PSH). Las subcabeceras están definidas por un PDU, a excepción de la PSH la cual está definida por un SDU. La subcabecera PSH es utilizada para empaquetar más de una PDU o SDU fragmentada en una sola MPDU, siendo un súper conjunto de la subcabecera de fragmentación (FSH). Las subcabeceras PSH y el FSH son mutuamente excluyentes (ambas no pueden estar presentes dentro de la misma MAC PDU). La subcabecera FSH se puede utilizar solo cuando el MPDU consiste de solo un fragmento SDU o un SDU.

2.5.2.1 Subcabeceras de Fragmentación (FSH)

Las subcabeceras de fragmentación son utilizadas para transportar fragmentos MSDU o SDU encapsulados en una conexión IEEE 802.16. Estas transportan suficiente información para identificar apropiadamente al receptor y re ensamblar los fragmentos. Una vez que se ha establecido una conexión con la fragmentación habilitada, todos los MPDUs transportados en una conexión pueden tener un FSH, con independencia de si una MPDU específica lleva fragmentación o SDU completa.

2.5.2.2 Subcabecera de Empaquetamiento (PSH)

La subcabecera de empaquetamiento permite la transmisión de múltiples fragmentos MSDUs o un MSDU en un solo MPDU en conexiones IEEE 802.16. El mecanismo ARQ requiere que los fragmentos SDUs o SDU se transmitan fuera de secuencia cuando se han perdido los bloques retransmitidos durante transmisiones previas. El mecanismo de empaquetamiento está designado para soportar tanto transmisiones de bloques ARQ no continuos como fragmentos.

2.5.3 Formato de las MAC PDU

Las unidades de datos de protocolo MAC (MAC PDUs) IEEE 802.16, se pueden formar como se observa en la Figura 2.4. Cada PDU puede comenzar con una cabecera MAC genérica de longitud fija. La cabecera puede estar seguida por la carga útil del MAC PDU. Si está presente la carga útil podría consistir de cero o más subcabeceras, y de cero o más MAC SDUs y/o fragmentos de estos. Cuando están presentes las subcabeceras y carga útil especial todas las subcabeceras por PDU se colocan justo después de la cabecera genérica MAC. Si están presentes las subcabeceras opcionales, son consideradas parte de la carga útil y por lo tanto están protegidas por el mismo CRC que protege al resto de la carga útil. La información de la carga útil puede variar en longitud por lo que una MAC PDU puede representar un número variable de bytes. Esto permite a la MAC agrupar varios tipos de tráfico de capas superiores más allá del conocimiento de los formatos o patrones de bit de estos mensajes. Adicionalmente, una MAC PDU puede tener un código de redundancia cíclica (CRC). La MAC IEEE 802.16 recibe MSDUs desde la CS. En la figura 2.4 se puede observar el formato del MAC PDU.

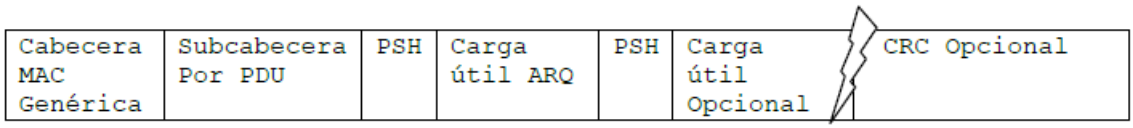

Figura 2.4: Formato del MAC PDU.

Hay dos clases de información MAC IEEE 802.16 transportada entre pares de entidades: cabeceras MAC Stand-Alone y MPDUs.

2.6 Solicitud de Repetición Automática (ARQ)

ARQ es el protocolo utilizado para el control de errores, que retransmite aquellos paquetes que no llegaron correctamente a su destino. Es decir, ARQ permite conexiones que requieren que cada paquete transmitido sea reconocido por el receptor, los paquetes desconocidos son asumidos como paquetes perdidos y son automáticamente retransmitidos.

El mecanismo de retransmisión ARQ es una parte de la funcionalidad de la capa MAC específicamente la subcapa de parte común (CPS), que es opcional para su implementación. Cuando se implementa, ARQ puede ser habilitado por conexión, el cual deberá especificarse y negociarse durante creación de la conexión. Una conexión no puede tener una mezcla de tráfico ARQ y no ARQ. Al igual que otras propiedades del protocolo MAC, el ámbito de una instancia específica de ARQ se limita a una conexión unidireccional. Para las conexiones ARQ habilitadas permitir la fragmentación es opcional. Cuando se habilita la fragmentación, el transmisor puede particionar cada SDU en fragmentos, para la transmisión por separado en función del valor del parámetro *ARQ_BLOCK_SIZE.* Cuando la fragmentación no está activada, la conexión deberá ser manejada como si la fragmentación se hubiese habilitado. En este caso, independientemente del tamaño del bloque de negociación, cada fragmento formado para la trasmisión deberá contener todos los bloques de datos asociados a la SDU padres.

Un MAC SDU está particionado lógicamente en bloques cuya longitud se especifica en el parámetro de conexión *ARQ_BLOCK_SIZE*. Cuando la longitud del SDU no es un múltiplo entero del tamaño del bloque de la conexión, el bloque final del SDU se forma con los bytes restantes del SDU después de que el bloque completo final ha sido determinado. Una vez que el SDU se divide en un conjunto de bloques, este

particionado permanece hasta que todos los bloques de la SDU son entregados con éxito al receptor, o el SDU es descartado por el emisor.

Grupos de bloques seleccionados para la transmisión o retransmisión se encapsulan en un PDU. El PDU puede contener bloques que se transmiten por primera vez, así como aquellos que están siendo retransmitidos.

La fragmentación se produce sólo en los límites de bloque ARQ. Si una PDU no está lleno, todos los bloques en el PDU tendrán números de bloque contiguos. Cuando un PDU está lleno la secuencia de bloques que estén inmediatamente entre las subcabeceras MAC y la secuencia de bloques después de la PSH, tendrán números de bloque continuos.

Si el mecanismo ARQ está habilitado en la conexión, la FSH y la PSH contienen un BSN, que es el número de secuencia del primer bloque ARQ en la secuencia de bloques siguientes a la subcabecera .Se trata de una cuestión de política, si un conjunto de bloques fueron una vez transmitido en un solo PDU, entonces debe ser retransmitido también como un solo PDU. En la Figura 2.5 se ilustra el uso de bloques en las transmisiones y retransmisiones ARQ, se tienen dos opciones para la retransmisión ya que se puede presentar con y sin reordenamientos de los bloques.

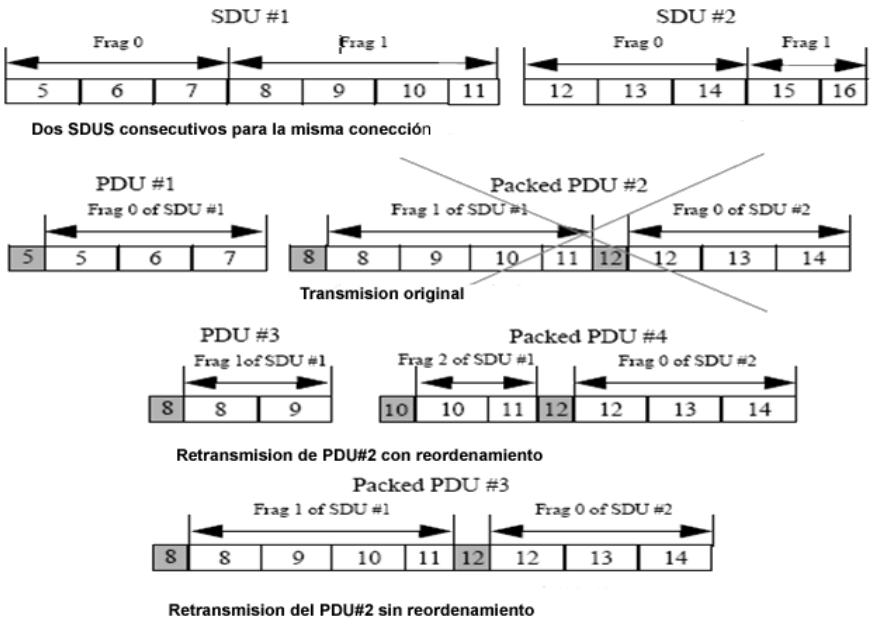

Figura 2.5: Ejemplo de bloques ARQ con y sin ordenamiento [7].

2.6.1 Bloques ARQ

Un bloque ARQ IEEE 802.16 es la unidad básica de transmisión o retransmisión para una conexión ARQ habilitada. Una MSDU está lógicamente particionada en bloques de tamaños fijo, negociados durante la configuración de la conexión. Cada bloque ARQ de un SDU tiene asignado un número de secuencia denominado BSN. La fragmentación puede ocurrir solo en los bordes de los bloques ARQ. El valor llevado por el BSN en el FSH es el BSN del primer bloque ARQ que aparezca en el segmento. En la Figura 2.6 se observa el concepto de bloques ARQ con dos SDUs consecutivos de la misma conexión.

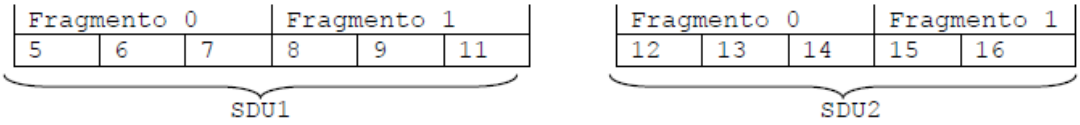

Figura 2.6: Bloques ARQ.

2.7 Retransmisiones ARQ basadas en Bloques

La retransmisión usa la detección de errores, retroalimentación, procesos de transmisión y la retransmisión de bloques de datos en paquetes. El estándar IEEE802.16 utiliza dos formas de ARQ: repetición selectiva automática y petición de repetición automática híbrida (HARQ), en este trabajo se estudia únicamente la repetición selectiva automática.

2.7.1 Repetición Selectiva Automática

La Repetición Selectiva Automática es un control de transmisión de datos que permite al receptor pedir la retransmisión de bloques selectivos de datos, cuando se desea implementar puede habilitarse por conexión, y será especificado y negociado durante la creación de la conexión. Para conexiones con ARQ habilitado, la fragmentación es opcional.

Los datos a ser transmitidos son agrupados en bloques y se les da un número de secuencia de bloque (BSN), el máximo tamaño de un bloque de datos en el sistema IEEE 802.16 es de 2040 bytes.

Cuando los bloques son transmitidos usando ARQ cada bloque recibe un identificador de sub-paquete (SPID). Un SPID es un índice que puede ser usado para identificar paquetes que están aguardando la conformación en un proceso ARQ.

Como los bloques son transferidos entre los dispositivos de envió y de recepción, los mensajes de acuse de recibido son enviados. Los tipos de acuses IEEE 802.16 pueden ser selectivos, acumulativos y acumulativos con selectivo. Los mensajes ARQ pueden ser enviados en mensajes separados o pueden ser combinados en otros mensajes.

2.7.2 Operación ARQ

A continuación se muestra detalladamente el proceso de retransmisión ARQ basada en bloques, tanto desde el punto de vista del emisor como del receptor.

2.7.2.1 Variables Emisor

- ***ARQ_TX_WINDOW_START*:** Todos los bloques cuyo BSN este en el rango de ARQ_TX_WINDOW_START -1 han sido reconocidos.

- ***ARQ_TX_NEXT_BSN*:** BSN del próximo bloque a enviar. Este valor reside en el intervalo *ARQ_TX_WINDOW_START* a (*ARQ_TX_WINDOW_START* + *ARQ_WINDOW_SIZE*), inclusive.

2.7.2.2 Variables del Receptor

- ***ARQ_RX_WINDOW_START*:** Todos los bloques cuyo BSN este en el rango de ARQ_RX_WINDOW_START -1 han sido reconocidos correctamente.

- ***ARQ_RX_HIGHEST_BSN*:** BSN del más alto bloque recibido más uno. Este valor reside en el intervalo *ARQ_RX_WINDOW_START* a (*ARQ_RX_WINDOW_START* + *ARQ_WINDOW_SIZE*), inclusive.

2.7.2.3 Parámetros ARQ

A continuación se muestran los parámetros utilizados en la operación ARQ

- **ARQ_BSN_MODULUS:** Es igual al número de valores únicos BSN.

- **ARQ_WINDOW_SIZE:** Es el número máximo de bloques ARQ con BSN consecutivos en la ventana deslizante, que es administrado por el receptor y el transmisor. Deberá ser inferior o igual a la mitad de los *ARQ_BSN_MODULUS*.

- **ARQ_BLOCK_LIFETIME:** Es el máximo intervalo de tiempo en que un bloque ARQ es gestionado por el emisor, una vez que la transmisión inicial del bloque se ha producido. Si la transmisión (o posterior retransmisión) del bloque no es reconocida por el receptor antes del tiempo límite, el bloque se descarta. El inicio de la vida del bloque comienza una vez transmitido.

- **ARQ_RETRY_TIMEOUT:** Es el intervalo de tiempo mínimo que el emisor deberá esperar antes de la retransmisión de un bloque no reconocido. El intervalo de tiempo comienza una vez que el bloque ha sido transmitido.

- **ARQ_SYNC_LOSS_TIMEOUT:** Es el intervalo de tiempo máximo en que se le permitirá a los parámetros *ARQ_TX_WINDOW_START* y *ARQ_RX_WINDOW_START* permanecer en el mismo valor antes de declarar una pérdida de sincronización de las máquinas de estado del emisor y el receptor cuando la transferencia de datos esta activa. Las máquinas de estado del emisor y el receptor gestionan temporizadores independientes.

- **ARQ_RX_PURGE_TIMEOUT:** Es el intervalo de tiempo que el receptor debe esperar después de la recepción exitosa de un bloque que no resulta en el progreso de *ARQ_RX_WINDOW_START*, antes de avanzar**.**

- **ARQ_BLOCK_SIZE:** Es el tamaño usado para la partición de un SDU en una secuencia de bloques para ser transmitidos**.**

2.7.2.4 Máquina de estado del emisor

Un bloque de ARQ puede estar en uno de estos cuatro estados: *Nosent (*no enviado), *Outstanding* (pendiente), *Discarded* (descartado), y *Waiting for retransmission* (en espera de la retransmisión). Cualquier bloque ARQ comienza en el estado *Nonsent*.

Posterior al envío de un bloque ARQ, el bloque cambia su estado de *Nosent* a *Outstanding* por un período de tiempo denominado *ARQ_RETRY_TIMEOUT*. Mientras que un bloque se encuentra en estado *Outstanding*, puede ser reconocido o descartado. El bloque puede cambiar también al estado *Waiting for retransmission* si se ha expirado el tiempo de *ARQ_RETRY_TIMEOUT* o se ha recibido *NACK* para ese bloque.

Un bloque de ARQ también puede cambiar del estado *Waiting for retransmission* al estado *Discarded*, cuando un mensaje de reconocimiento de recepción (ACK) es recibido o después de que haya expirado el temporizador *ARQ_BLOCK_LIFETIME*.

Para una conexión dada el emisor primero tratará de manejar los bloques en estado *Waiting for retransmission* y solamente luego los bloques en estado de *Nosent*. Los bloques en estado *Outstanding* o D*iscarded* no serán transmitidos. Cuando los bloques se retransmiten, el bloque con el menor BSN deberá ser transmitido primero. La secuencia de estados de los bloques ARQ se muestra en la Figura 2.7.

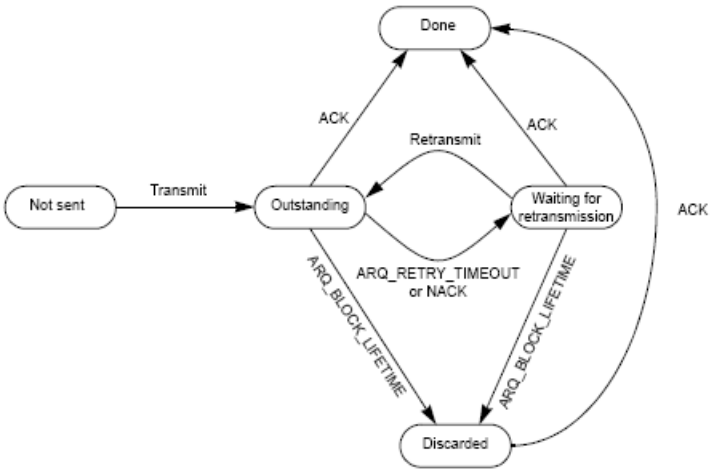

Figura 2.7: Estados Bloques ARQ.

La formación de la MAC PDU continúa con los SDUS no enviados de la conexión. El emisor construye cada MAC PDU utilizando las reglas de la fragmentación y el ensamblado, siempre y cuando el número de bloques que se enviarán más el número de bloque ya transmitidos y en estado *Waiting for retransmission* no exceda el límite impuesto por *ARQ_WINDOW_SIZE*. A medida que cada bloque *Nonsent* se forma es incluido en un PDU MAC, se le asigna el valor actual de *ARQ_TX_NEXT_BSN*, que luego se incrementa.

Cuando un acuse de recibo ACK acumulativo es recibido, el emisor deberá verificar la validez de la BSN. Un BSN es válido si está en el intervalo *ARQ_TX_WINDOW_START* a *ARQ_TX_NEXT_BSN* – 1 (inclusive). Si el BSN no es válido el emisor no toma en cuenta el reconocimiento (ACK).

Cuando un ACK selectivo, acumulativo o acumulativo y selectivo con una secuencia de bloques reconocidos es recibido, el emisor chequeará la validez de cada bloque

descrito en el mensaje. El reconocimiento (ACK) de un bloque es válido si el número de bloque correspondiente se encuentra en el intervalo *ARQ_TX_WINDOW_START* a *ARQ_TX_NEXT_BSN-1* (inclusive). Si el número de bloque se encuentra fuera de este intervalo, el emisor ignorará el reconocimiento de ese bloque.

Cuando un reconocimiento acumulativo con un BSN válido es recibido, el emisor tendrá en cuenta todos los bloques en el intervalo entre *ARQ_TX_WINDOW_START* y el BSN (inclusive) como reconocidos y establece *ARQ_TX_WINDOW_START* a BSN + 1.

Cuando un reconocimiento selectivo es recibido, el emisor considerará los reconocimientos de todos los bloques indicado por las entradas en el mapa de bits para los valores de BSN válida.

Como las entradas de mapa de bits se procesan en un orden creciente de BSN, *ARQ_TX_WINDOW_START* se incrementa cada vez que el BSN de un bloque reconocido es igual al valor de *ARQ_TX_WINDOW_START*.

Cuando *ARQ_TX_WINDOW_START* ha sido avanzado o incrementado por alguno de los métodos anteriores y el reconocimiento de recepción (ACK) ya ha sido recibido para el bloque con el valor de BSN actualmente asignado a *ARQ_TX_WINDOW_START*, el valor de *ARQ_TX_WINDOW_START* se incrementa hasta el valor BSN que se alcanza para los bloques de los cuales no se ha recibido reconocimiento.

Una entrada de mapa de bits que no indica un reconocimiento deberá ser considerado como un NACK para los bloques correspondientes.

Un ACK selectivo es un mapa de bits que hace referencia a un BSN específico, lo cual indica el número absoluto de bloques referenciados por el primer bit en el mapa de

bits. Todos los bits del mapa de bits, se definen como un ACK ó un NACK para un bloque especifico.

Cuando un ACK acumulativo con reconocimiento selectivo y un BSN válido son recibidos, el emisor realiza las acciones descritas anteriormente para el reconocimiento acumulativo, seguido por los de reconocimiento selectivo.

Todos los temporizadores asociados con los bloques reconocidos, quedan anulados.

Un mensaje de descarte será enviado siguiendo el incumplimiento del *ARQ_BLOCK_LIFETIME*. El mensaje puede ser enviado inmediatamente o ser postergado hasta *ARQ_RX_PURGE_TIMEOUT* + *ARQ_RETRY_TIMEOUT*. Si ARQ_RX_PURGE_TIMEOUT es infinito (es decir, tiene un valor cero), entonces el mensaje se puede postergar hasta *ARQ_RETRY_TIMEOUT.*

Después de la primera transmisión, el orden sucesivo de descarte será enviado al receptor en intervalos de *ARQ_RETRY_TIMEOUT* hasta que un reconocimiento para descartar BSN ha sido recibido. El orden de descarte para los valores adyacentes de BSN puede ser acumulado en un solo mensaje de descarte.

Las acciones a ser tomadas por la máquina de estado del emisor y el receptor cuando se quiere iniciar un restablecimiento de la conexión son provistas en la Figura 2.8 y Figura 2.9.

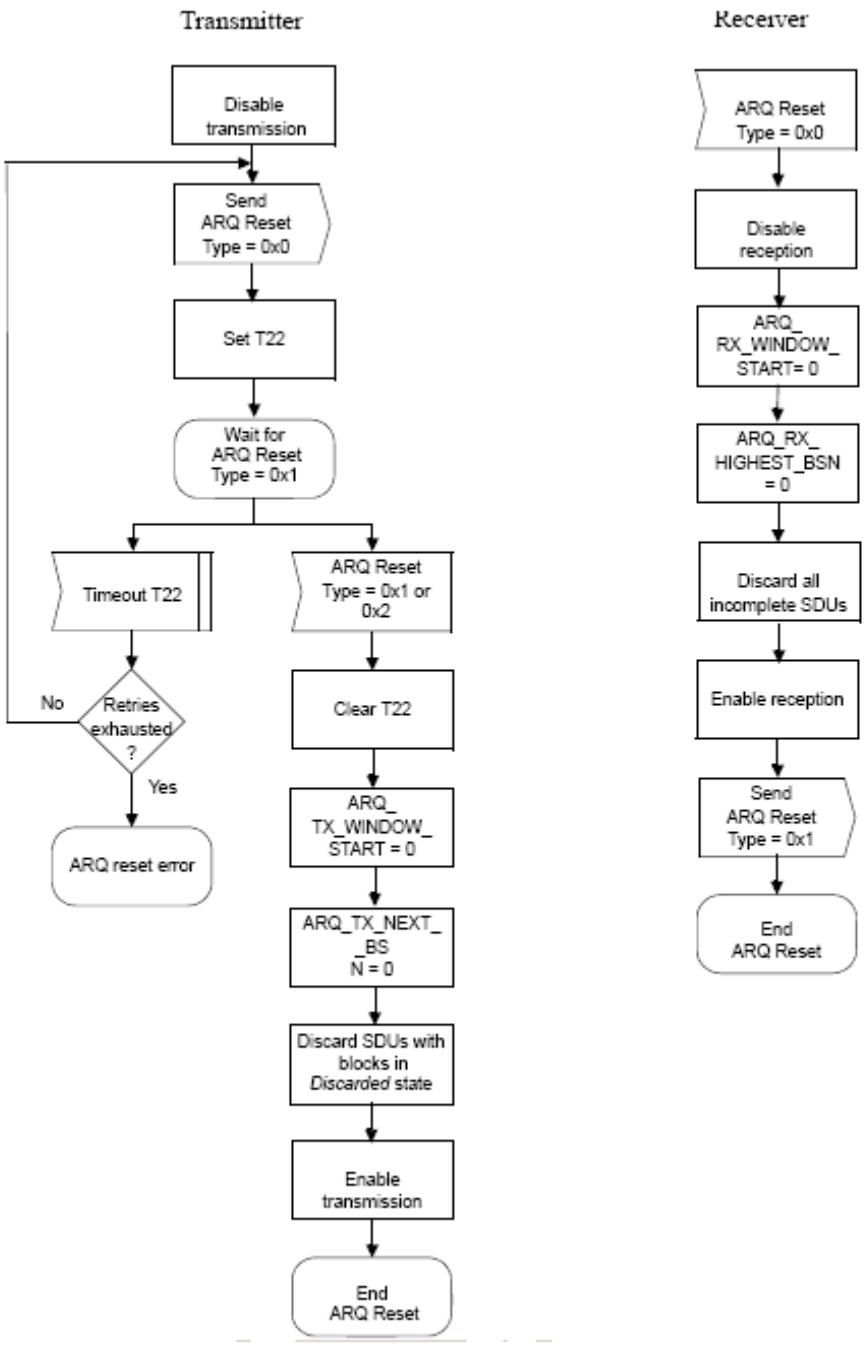

Figura 2.8: Diálogo de mensaje ARQ Restablecimiento del Emisor.

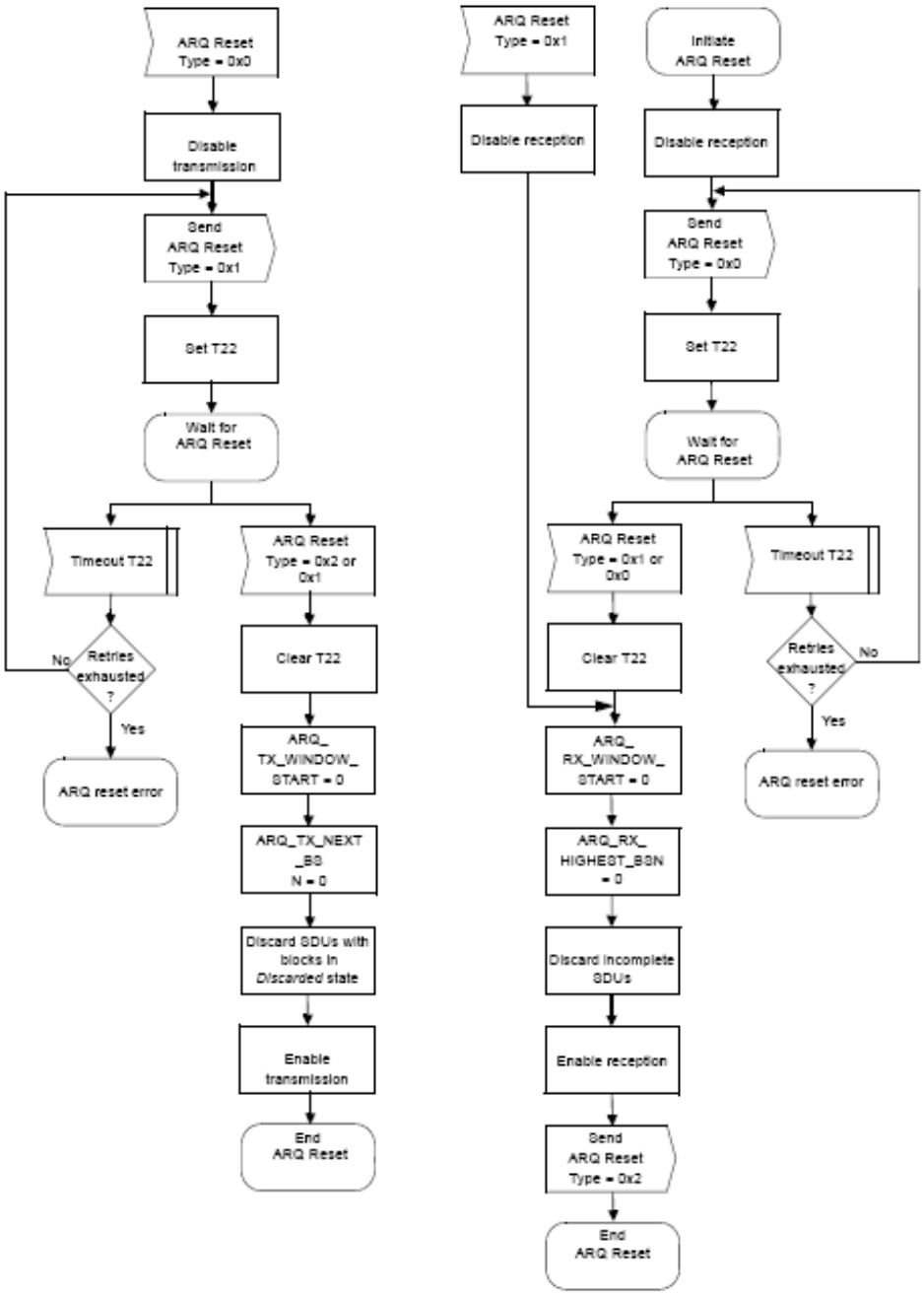

Figura 2.9: Diálogo de mensaje de ARQ Restablecimiento del Receptor.

La sincronización ARQ de las máquinas de estado se rige por un temporizador administrado por la máquina de estado del emisor y del receptor. Cada vez que se actualiza la variable *ARQ_TX_WINDOW_START* en el emisor, el contador se pone a cero. Cuando el tiempo supera el valor de *ARQ_SYNC_LOSS_TIMEOUT,* la máquina del estado emisor deberá iniciar un restablecimiento de la conexión como se describe en la Figura 2.8.

Cuando se está en el estado de error de restablecimiento ARQ en la Figura 2.8 y Figura 2.9, la SS reinicializa su MAC, y el comportamiento de BS depende de la implementación.

Un mensaje de descarte puede ser enviado al receptor cuando el emisor quiere saltarse los bloques ARQ hasta el valor BSN especificado en el mensaje de descarte. Una vez recibido el mensaje, el receptor actualiza su información de estado para indicar los bloques específicos que fueron recibidos.

2.7.2.4 Receptor

Cuando se recibe un PDU, su integridad es determinada con base en la suma de comprobación CRC-32.Un PDU pasa la comprobación checksum, se desempaqueta y desfragmenta, si es necesario. El receptor mantiene una ventana deslizante definida por la variable de estado *ARQ_RX_WINDOW_START* y el parámetro *ARQ_WINDOW_SIZE*.

Cuando un bloque ARQ con un número que está dentro del rango definido por la ventana deslizante es recibido, el receptor lo aceptará. Los números de bloque ARQ fuera de la ventana deslizante serán rechazados. El receptor debe descartar bloques ARQ duplicados. Ver la Figura 2.10.

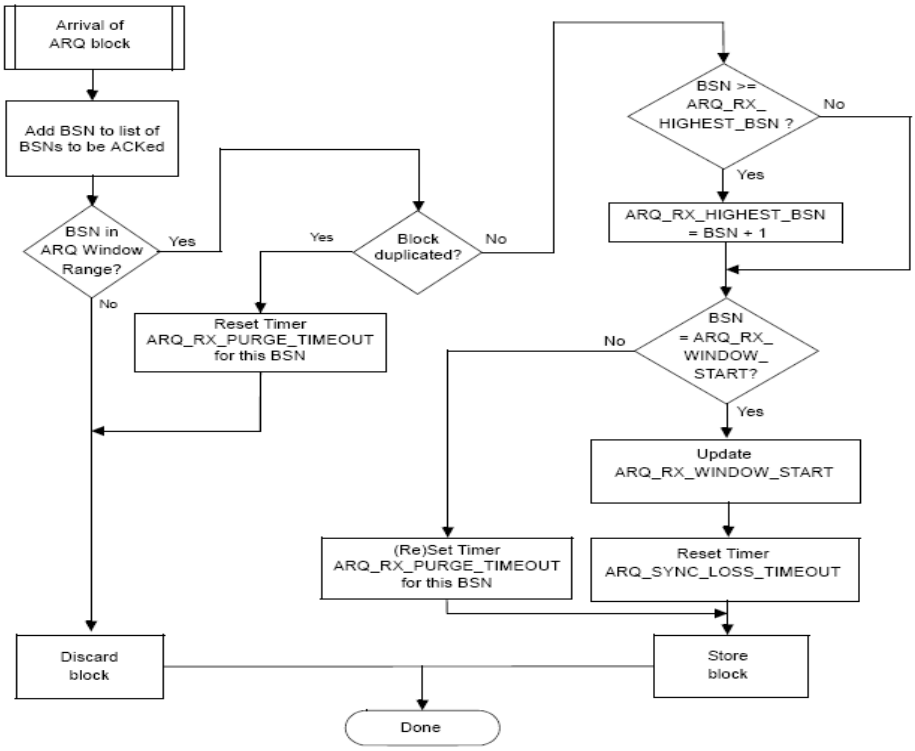

Figura 2.10: Bloque ARQ del receptor.

La ventana deslizante se mantiene ya que la variable *ARQ_RX_WINDOW_START* siempre apunta al bloque ARQ con el número más bajo que no se ha recibido o ha sido recibido con errores. Cuando un bloque ARQ con un número correspondiente a la variable *ARQ_RX_WINDOW_START* se recibe, la ventana avanza(es decir, se incrementa *ARQ_RX_WINDOW_START* a *ARQ_BSN_MODULUS)* para que la variable *ARQ_RX_WINDOW_START* apunte al siguiente bloque ARQ con número más bajo que no se ha recibido o ha sido recibido con errores. El temporizador asociado con *ARQ_SYNC_LOSS_TIMEOUT* se restablece.

Cuando un bloque no resulta en un avance de la *ARQ_RX_WINDOW_START*, el *ARQ_RX_PURGE_TIMEOUT* para ese bloque se pondrá en marcha. Cuando el valor del

temporizador para un bloque es excedido *ARQ_RX_PURGE_TIMEOUT*, la condición de tiempo de espera es marcada. Cuando la condición de tiempo de espera es marcada, *ARQ_RX_WINDOW_START* avanza hasta el BSN del siguiente bloque que aún no ha sido recibido después del bloque marcado. Los temporizadores para los bloques entregados siguen activos y son monitoreados por tiempo de espera hasta que los valores BSN estén fuera de la ventana de recepción.

Cuando *ARQ_RX_WINDOW_START* está casi completo, cualquier valor BSN correspondientes a los bloques que aún no han sido recibidos residen en el intervalo entre el valor anterior y el valor actual del parámetro *ARQ_RX_WINDOW_START*, se marcará como recibido. Todos los bloques que pertenecen a la SDU completa serán entregados. Los bloques parcialmente exitosos deben ser desechados.

Cuando un mensaje de descarte es recibido desde el emisor, el receptor descartará los bloques especificados, avanzando el *ARQ_RX_WINDOW_START* hacia al BSN del primer bloque que aún no ha sido recibido después del BSN provisto en el mensaje de descarte.

Para cada bloque ARQ recibido, un reconocimiento será enviado al emisor. Los reconocimientos para los bloques fuera de tamaño de la ventana deslizante deben ser acumulados. Los reconocimientos para los bloques dentro de la ventana deslizante pueden ser específicos para cada bloque ARQ, o acumulativos que contiene el número más alto de bloques ARQ permitiendo que todos los bloques ARQ tengan que ser recibidos correctamente, o la combinación de ambos, reconocimientos acumulativos y selectivos. Los reconocimientos serán enviados en el orden de los números de bloque ARQ reconocidos. La frecuencia de generación de reconocimiento depende de la implementación.

Un MAC SDU está listo para ser entregada a las capas superiores, cuando todos los bloques ARQ de la MSC SDU han sido recibidos correctamente dentro de los valores de *time-out* definidos.

Cuando *ARQ_DELIVER_IN_ORDER* no está habilitado, la MAC SDU es transmitida a las capas superiores tan pronto como todos los bloques de la MAC SDU han sido recibidos exitosamente dentro de los valores de *time-out* definidos.

Las acciones a ser tomadas por el receptor cuando un mensaje ARQ de Restablecimiento es recibido son mostradas en la Figura 2.9.

CAPÍTULO 3: REDES DE PETRI

3.1 Definición

Las Redes de Petri fueron introducidas por Carl Adam Petri a principios de 1960 [1] [2]. En 1962 para su tesis doctoral "Comunicación con autómatas". Petri formuló la teoría básica de la comunicación entre componentes asíncronos de sistemas de cómputo. Estuvo enfocado particularmente en la descripción de las relaciones causales entre eventos. Uno de sus principales atractivos como formalismo de modelado es cómo los aspectos básicos de sistemas concurrentes son identificados tanto conceptual como matemáticamente. Las Redes de Petri permiten modelar sistemas de evolución en paralelo compuesto por procesos que requieren sincronía para realizar un objetivo en común y pueden ser analizadas de manera formal para obtener información del comportamiento dinámico del sistema**.**

Estructuralmente, una Red de Petri (PN) es un grafo tripartito dirigido compuesto por plazas, transiciones y arcos dirigidos. $R = \{P,T,Pre,Post\}$, dónde P es un conjunto de plazas de cardinal n, T un conjunto de transiciones de cardinal m, Pre la aplicación de incidencia previa que viene definida como Pre: P x T → Naturales y Post la aplicación de incidencia posterior que viene definida como Post: P x T → Naturales.

Las plazas se representan gráficamente a través de círculos y permiten modelar condiciones o objetos. Dentro de las plazas hay marcados representados por puntos negros, los cuales representan el valor específico de una condición u objeto. La presencia de marcados en las plazas se conoce como marcado o estado (es decir, marcado o estado de la red). El sistema inicia con algunas marcas en las plazas, conocido como marcado inicial. Un marcado de la plaza indica el número de marcas en la plaza particular. La

presencia de marcas en una plaza se interpreta habitualmente como presencia de recursos. Las transiciones, se representan gráficamente con rectángulos, las cuales se utilizan para describir eventos que pueden modificar el estado del sistema. Los arcos dirigidos representan la relación entre las plazas y los eventos de dos formas: indican las condiciones sobre las cuales el evento puede ocurrir y la transformación de la plaza introducida por el evento. Los arcos se dirigen de una plaza a una transición o de una transición a una plaza y pueden tener un peso (número entero positivo) asociado a ellos (el peso por defecto de un arco es uno y no se muestra) como se muestra en la Figura 3.1.

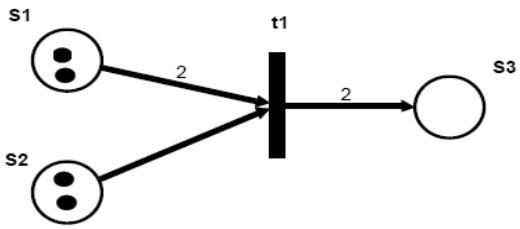

Figura 3.1: Ejemplo de una Red de Petri [5].

Dependiendo del sistema que se modelará, las plazas y las transiciones pueden tener diversas interpretaciones. Por ejemplo, una plaza puede representar una condición y una transición un evento. También, una plaza puede representar recursos y una transición una tarea o un trabajo, que requieren esos recursos. Finalmente, una plaza puede representar el estado del sistema y una transición, una acción que pueda ser tomada, basada en ese estado.

El comportamiento dinámico de una Red de Petri se puede considerar como el estado o el marcado de la red cambiando según las ocurrencias de las transiciones. Una transición puede tener plazas de entrada conectadas por los arcos entrantes y plazas de salida conectadas por los arcos salientes. Las entradas de las transiciones son las precondiciones y la salida las post condiciones. La ocurrencia de un evento corresponde con el disparo de una transición. Una transición está habilitada (es decir puede ocurrir) si el marcado de cada plaza de entrada contiene tantas marcas como lo indicado por el peso del arco de entrada, que conecta a las plazas con la transición. La ocurrencia o disparo de una transición habilitada toma marcas de las plazas de entrada (representando las precondiciones) y agrega marcas a las plazas de salida (representando las post condiciones). El número de marcas removidas de cada plaza de entrada corresponde al número de marcas indicadas por el peso del arco (de entrada), que conecta las plazas y la transición. El número de marcas agregadas a cada plaza de salida corresponde al número de marcas indicadas por el peso del arco (de salida), que conecta la transición con las plazas. En la Figura 3.2 se puede observar un ejemplo de la ocurrencia de la transición t1 mostrada en la Figura 3.1 Después que la transición t1 ocurre, dos marcas se remueven de la plaza S1 y uno de la plaza S2, y dos marcas se agregan a la plaza S3.

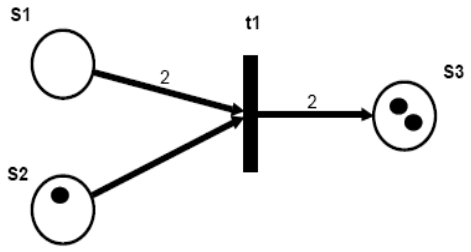

Figura 3.2: Ocurrencia de la Transición t1.

Así mismo, existen los arcos inhibidores (arcos formando ciclos desde las plazas hacia las transiciones). Se permiten múltiples arcos entre las plazas y las transiciones (de entrada, salida e inhibidores) y anotando con un número especificando sus multiplicidades.

Partiendo del marcado inicial, la ocurrencia de las transiciones puede generar nuevos marcados, que a su vez dan lugar a la ocurrencia de más transiciones.

3.2 Modelado de un sistema

La ejecución de una Red de Petri y el comportamiento del sistema en el modelo es una secuencia de eventos discretos. El orden de ocurrencia de los eventos posiblemente pueda ser determinado en la estructura básica. Sin embargo las Redes de Petri reflejan los factores que en situaciones de tiempo real puedan ocasionar la ocurrencia de eventos concurrentes, el orden aparente de estos eventos no es único, sino que cualquier secuencia de eventos puede ocurrir.

El enfoque convencional descrito para el uso de Redes Petri en el diseño de sistemas, requiere una constante conversión o equivalencia entre el diseño del sistema y el modelo de la Red Petri. En términos prácticos el diseño completo y la especificación de los procesos debe ser llevado en términos de Redes Petri como se puede ver en la Figura 3.3

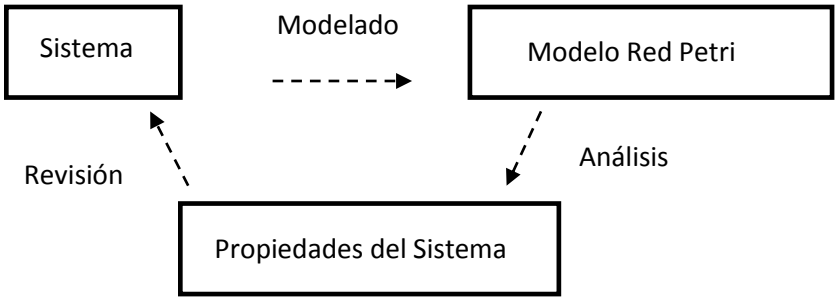

Figura 3.3: El uso de las Redes de Petri en el modelado y análisis de sistemas.

El sistema es primeramente modelado como una Red Petri y luego éste modelo es analizado para corroborar la correspondencia con el diseño. Este estudio aporta una amplia perspectiva de las características del sistema.

3.3 Redes de Petri Coloreadas (CPNs)

Mientras que los sistemas se tornan más complejos, los modelos basados en las Redes de Petri pueden llegar a ser muy grandes, complicados y probablemente ilegibles. Este problema ha sido superado introduciendo las Redes de Petri de Alto Nivel o Redes de Petri Coloreadas (CPNs)[6][2], las cuales son un lenguaje gráfico que modela eventos discretos y sistemas concurrentes para analizar sus propiedades, para ello se combinan las capacidades de las Redes de Petri, con las bondades de los lenguajes de programación de alto nivel. Las CPNs incorporan algunas definiciones, tales como tipos de datos y el procesamiento de valores de datos encontrados en los lenguajes de programación, así como primitivas básicas para el modelado de la concurrencia, comunicación y sincronización.

Las CPNs se utilizan para condensar la descripción y el análisis de sistemas en los que se identifican diferentes subsistemas independientes pero que operan en paralelo o de forma síncrona.

El lenguaje utilizado para modelar CPNs provee las primitivas para la definición de tipos de datos, descripción de la manipulación de los datos, y la creación de un modelo parametrizable.

Es un leguaje de modelado con propósito general no está enfocado a sistemas específicos, pero está orientado hacia sistemas que puedan ser caracterizados como sistemas concurrentes, generalmente protocolos de comunicación, redes de datos, algoritmos distribuidos y sistemas embebidos. Las CPNs son utilizadas generalmente para el modelado de sistemas donde la concurrencia y comunicación son características claves. La diferencia entre la definición anterior de Redes Petri y la de Redes de Petri Coloreadas (CPNs) viene marcada por las consideraciones en CPN de colores y de funciones lineales asociadas a sus aristas. Las marcas de color pueden representar un atributo o distintivo, si es necesario definir dos atributos entonces surge la idea de colores compuestos.

El tiempo juega un rol importante en los sistemas concurrentes. El correcto funcionamiento de algunos sistemas depende esencialmente del tiempo que toman ciertas actividades añadido a la diferentes decisiones que pueden ser tomadas en un momento dado, las cuales tienen un impacto significativo en el rendimiento del sistema. Las CPNs incluyen el concepto de tiempo lo cual hace posible medir el tiempo empleado en los eventos del sistema. Esta definición significa que las CPNs pueden ser aplicadas para analizar la simulación de un comportamiento donde el rendimiento muestre retardos, así como realizar una observación de la velocidad de transferencia, tamaño de las colas en el sistema, entre otros. Esto es especialmente importante para el modelado y

la validación de sistemas en tiempo real. A continuación se muestra un ejemplo del uso de las CPNs.

3.3.1 Ejemplo: Módulo Emisor del modelo del mecanismo de retransmisión ARQ

A continuación se muestra una abstracción del modelo de la máquina de estados del emisor del mecanismo de retransmisión de bloques ARQ, en cuando al proceso de transmisión de bloques. El funcionamiento principal de la operación ARQ es enviar bloques de datos desde una entidad emisora a una receptora; para ello se mantienen ventanas deslizantes para llevar a cabo control de flujo, y se tiene un mecanismo de parada y espera mediante la recepción de reconocimientos ACK por parte del receptor. En la Figura 3.4 podemos visualizar las funciones principales de la máquina de estados del emisor para la operación ARQ, en cuanto al proceso de transmisión. A continuación se describen los componentes de las CPNs.

Plazas: Representan las entidades. En la Figura 3.4 hay seis plazas dibujadas como elipses: *WindowsSender, List of blocks to Retransmit, To the Network, Blocks to Sent, Next BSN, Limit Next BSN*. La plaza *WindowsSender* representa la ventana deslizante que permite el control de flujo. Las plaza *To the Network* representan las salidas de bloques y paquetes hacia la red. La plaza *List of blocks to Retransmit* representa la lista de bloques en cola para ser retrasmitidos. La plaza *Blocks to Sent* contiene lo bloques a ser enviados al receptor. La plaza *Next BSN* almacena el BSN del próximo bloque a ser enviado. Y por último, la plaza *Limit Next BSN* tiene como función limitar o restringir la cantidad de marcas que puede contener la plaza *Next BSN*.

Tipos: cada plaza tiene un tipo asociado o un conjunto de colores (*colour set*), el cuál determina el tipo de datos que la plaza puede tener. Ellos se escriben en el modelo en el lado izquierdo o derecho de la plaza. Las definiciones de tipo se muestran en la Figura 3.4

y son similares a los tipos en los lenguajes de programación. Por ejemplo, el *colour set ListBSN* cuya definición es *colset ListBSN = list BLOCK*, que es una lista compuesta del *colour set BLOCK* definido como *colset BLOCK=product BSN * DATA* que representa los pares de bloques *(n,d)* siendo n un número entero que representa el BSN o identificador del bloque y d los datos a transmitir. *El colour set ListBSN* representa la lista de bloques en cola para ser retransmitidos.

Marcados: las marcas se asocian a cada plaza. Una marca es un valor (*colour*), que pertenece al tipo de la plaza. El marcado de una plaza es el multi-conjunto (*multi-set*) de marcas presentes en la plaza. Es un multi-conjunto, puesto que puede contener varias marcas con el mismo valor. Las CPNs incluyen el estado inicial del sistema, que se denomina marcado inicial y la cual se escribe en la parte izquierda o derecha superior de la plaza. Por ejemplo, la plaza *Windows Sender* tiene los marcados iniciales: *1`SRPacket(1,Nosent,0,"A")++,1`SRPacket(2,Nosent,0,"B")++,1`SRPacket(3,Nosent,0,"C")++ , 1`SRPacket(4,Nosent,0,"D").* Luego de la ejecución del modelo la plaza *Windows Sender* tendrá otras marcas dependiendo del comportamiento del modelo para esa simulación. Un ejemplo de marcas para la plaza *Blocks to Send* luego de un ejecución de la simulación: *1`ProcessedPacket((1,done))++,1`ProcessedPacket((2,done))++,1`ProcessedPacket((3,done) )++ ,1`ProcessedPacket((4,done)).*

Transiciones: las transiciones representan las acciones del sistema. Se representan como rectángulos en la Figura 3.4. Se pueden observar dos transiciones en la Figura 3.4. Por ejemplo, la transición *Transmit* modela el envío de bloques ARQ hacia el receptor. La transición *Next Block to Send* modela la selección del próximo bloque a ser enviado.

Arcos: los arcos conectan transiciones y plazas. Una transición puede tener plazas de entrada conectadas por los arcos entrantes y plazas de salida conectadas por los arcos salientes. Los arcos tienen expresiones asociadas con ellos, las cuales están situadas al

lado de los arcos y determinan cuales marcas son removidas o agregadas a las plazas. Como ejemplo tenemos los arcos que conectan la plaza *To the Network* con la transición *Transmit*, estos son *PACKET(n,d)* siendo este la marca que representa el paquete que pasa de la plaza *Blocks to Send* a la plaza *To the Network*, y *SLOT Free* que simboliza el consumo de un slot o cupo disponible, para poder colocar en el bloque representado por *PACKET(n,d)* en la plaza.

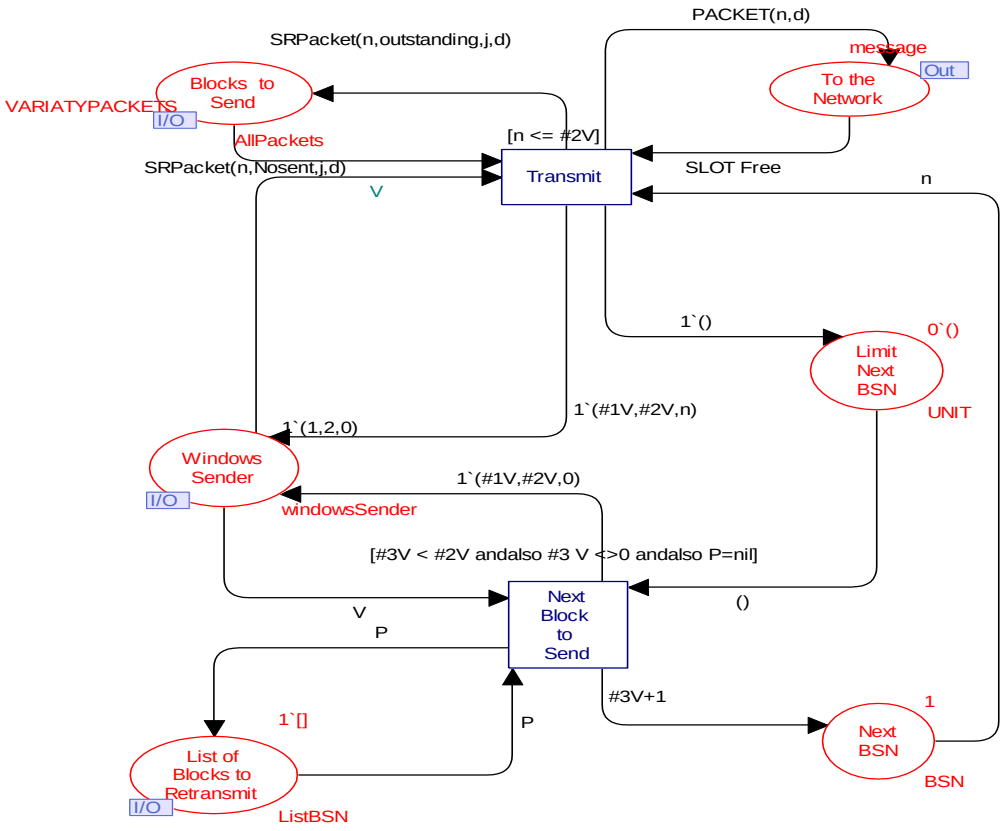

Figura 3.4: Transmisión de bloques ARQ.

```
colset windowsSender=product BSN * BSN * NextBsn ; color PR=product;

colset message= union PACKET:BLOCK+

colset ListBSN = list BLOCK;

colset BSN=int with 1..blocksnumber;

colset UNIT = unit;
```

Figura 3.5: Especificación de los conjuntos de colores de la Figura 3.4.

3.3.2 Asociaciones y Ocurrencia de las Transiciones

Una transición en CPN está en estado habilitado si todos sus nodos de entrada contienen un número de colores igual o mayor que los definidos por *fi<c>* donde *fi* es una función lineal asociada al nodo *pi* con la transición *tj.* Debe ser posible asociar valores de los datos a las variables que aparecen en las expresiones circundantes al arco y en la guarda y las siguientes condiciones deben ser satisfechas. En otras palabras, cuando se aplica un marcado (estado) μ y una transición *tj*, esta función produce el nuevo marcado (estado) como resultado del disparo (ejecución) de la transición *tj* en el marcado μ. La ocurrencia de una transición remueve marcas de las plazas entrantes y agrega marcas a las plazas salientes. Las marcas removidas son el resultado de evaluar las expresiones en los arcos entrantes correspondientes, mientras que los valores de las marcas agregadas son el resultado de evaluar las expresiones del arco en los arcos salientes correspondientes. La transición *tj* puede ocurrir o ser disparada solamente si está habilitada *fi(μ,tj),* es indefinida si *tj* no está habilitada en el marcado μ. Si *tj* está habilitada entonces, *fi(μ,tj)= μ'*, donde μ' es el marcado resultante de remover las marcas de la entrada de *tj* y añadir marcas a la salida de *tj*.

Una guarda es una expresión booleana que debe ser evaluada como true, para que una transición pueda estar habilitada, es decir, pueda ocurrir. En caso contrario, la transición estará deshabilitada y no podrá ocurrir. Por lo tanto, una guarda pone una restricción adicional en la ocurrencia de una transición. En la Figura 3.6 se puede visualizar el uso de una guarda para la transición *Transmit* del ejemplo de la Figura 3.4, para la cual se verifica que el valor de *n* que representa el *BSN* del próximo bloque ARQ a enviar, sea menor o igual que el valor de *#2V* que representa un apuntador al final de la ventana deslizante, es decir, que los bloques ARQ a ser enviados se encuentren dentro del rango de la ventana deslizante.

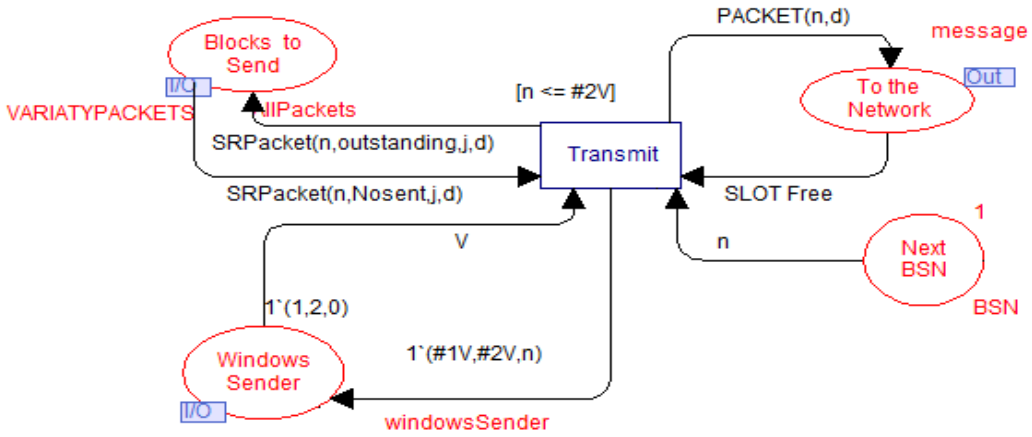

Figura 3.6: Guarda transición Transmit de la Figura 3.4.

3.3.3 CPNs jerárquicas

Las CPNs jerárquicas permiten que los modelos sean construidos de forma modular y estructurada. Los diversos niveles jerárquicos muestran los diferentes niveles de detalle

del modelo. Los modelos CPNs pueden ser estructurados dentro de módulos, esto es particularmente importante cuando se está tratando con modelos CPNs de sistemas muy grandes. Los módulos interactúan con cada uno de los otros de forma similar a como trabajan los lenguajes. El concepto de módulos en las redes CPNs está basada en mecanismos estructurados de forma jerárquica, que permite que cada módulo tenga sub módulos.

Un modelo CPN puede estar conformado por un número de módulos. Las CPNs jerárquicas proporcionan dos mecanismos para interconectar estos módulos: transiciones de substitución y plazas fusionadas. Las transiciones de substitución permiten que los modelos sean construidos en una manera *top-down*. Las jerarquías se construyen usando las transiciones de sustitución, las cuales pueden ser consideradas como una macro expansión del modelo. El modelo se inicia con un diagrama CPN en el nivel superior, el cual proporciona una visión general del sistema que está siendo modelado y su ambiente. En las CPNs jerárquicas, este diagrama en el nivel superior contendrá un número de transiciones de sustitución, las cuales se caracterizan por tener un doble cuadrado de borde, como se muestra en la Figura 3.7 para las transiciones de sustitución *Sender* y *Receiver*. Cada una de estas transiciones de sustitución es refinada en otro diagrama CPN, el cual puede a su vez contener transiciones de sustitución. El diagrama en el nivel superior y cada una de las transiciones de sustitución es definido por un módulo. Si una transición es una transición de substitución, tiene una sub modulo relacionado con ella, que incluye una descripción más detallada del modelo. El nombre del sub módulo de esta transición se muestra en un recuadro en la parte superior de la transición y se llama de la misma forma que las transiciones de substitución. Los sub módulos y módulos de CPN se interconectan a través de plazas puertos y zócalos. Los sub módulos tienen plazas puertos, que les permiten recibir y/o entregar marcas de los módulos de un nivel más alto.

En la Figura 3.7 se muestra la jerarquía superior del modelo de retransmisión de bloques ARQ.

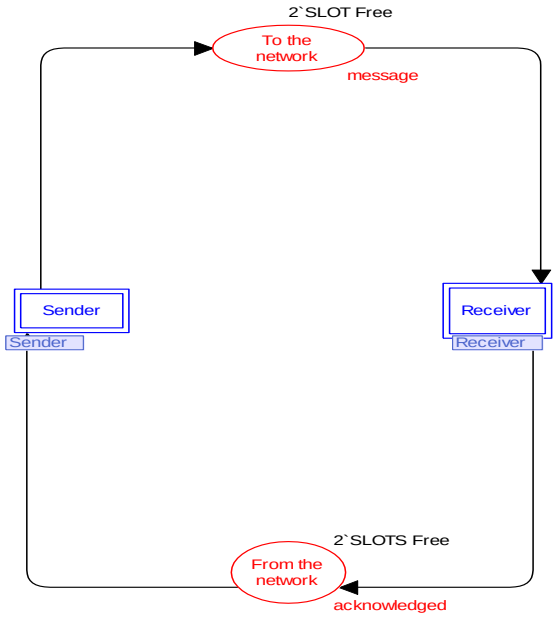

Figura 3.7: Jerarquía Superior (Ejemplo Figura 3.4).

3.3.4 Propiedades de las CPNs

Las propiedades que describen el comportamiento previsto del modelo pueden ser definidas. En esta sección se describen, las principales propiedades de comportamiento o dinámicas de las CPNs.

3.3.4.1 Accesibilidad

Por convención, Mn denota el marcado del nodo número *n*. Mn es *alcanzable* desde M1 si hay una secuencia de ocurrencia desde de marcado M1 al Mn.

3.3.4.2 Acotamiento

Una red *(N, M0)* se dice que está *k-acotada* o simplemente *acotada* si el número de marcas en cada plaza no excede un número finito *k* para ningún marcado alcanzable desde *M0*. Las cotas enteras superiores e inferiores indican el número máximo y mínimo de marcas que se pueden colocar en cada plaza en los marcados alcanzables. El otro concepto relacionado con las propiedades de acotamiento es el de *cotas de los multi-conjuntos,* los cuales proporcionan información sobre los valores de las marcas que las plazas pueden contener. La cota de multi-conjunto superior de una plaza se define como el más pequeño multi-conjunto que es más grande o igual que todos los marcados alcanzables de la plaza [6].

3.3.4.3 Marcados Locales

Un marcado local es un marcado que puede ser siempre alcanzado por el resto de los marcados alcanzables. Un *espacio local* es un conjunto de marcados tales que desde cada marcado alcanzable, es posible alcanzar por lo menos uno de estos marcados.

3.3.4.4 No Abrazos Mortales

Un marcado muerto es un marcado sin elementos de asociación habilitados. Un sistema se dice estar libre de abrazos mortales, si ningún marcado muerto se puede alcanzar desde el marcado inicial.

3.3.4.5 Transiciones Muertas

Una transición muerta no está habilitada para ningún marcado alcanzable.

3.3.5 Análisis de las CPNs usando espacio de estado

El estado de una CPN está definido por su marcado. La ocurrencia de una transición representa un cambio en un estado mediante el cambio en las marcas de la red. El espacio de estado de una CPN con n plazas es la forma de cambiar todas las marcas, esto es N^n .

Las simulaciones permiten que el usuario entienda y elimine los errores del modelo. Sin embargo, no es posible comprobar las propiedades de las CPNs en el modelado por medio de la simulación a menos que el sistema sea trivial. Así, se han creado varios métodos de análisis. Uno de ellos es el método del grafo de estado.

El grafo de estado incluye todos los marcados posibles (espacios de estado) que se puedan alcanzar desde el marcado inicial y se representa como un grafo dirigido donde los nodos representan los marcados y los arcos los elementos de asociación que ocurren. Los grafos de estado pueden ser construidos automáticamente, lo que permite el análisis y verificación automatizada del comportamiento del sistema modelado. Asimismo, el grafo de estado incluye información sobre el comportamiento del sistema, que son importantes para realizar el análisis y verificación. Adicionalmente, el método del grafo de estado se puede utilizar para eliminar errores y probar el sistema. En algunos casos, el modelo del sistema puede generar un número muy grande o inclusive infinito de marcados. En tales casos, el grafo de estado no puede ser generado, no obstante algunas propiedades de las CPNs se pueden probar o refutar basado en grafos de estado parciales incluyendo sub-grafos finitos del grafo de estado completo de un modelo específico. En la Figura 3.8 se puede ver un ejemplo de un grafo de estado correspondiente al caso de estudio de la Figura 3.4.

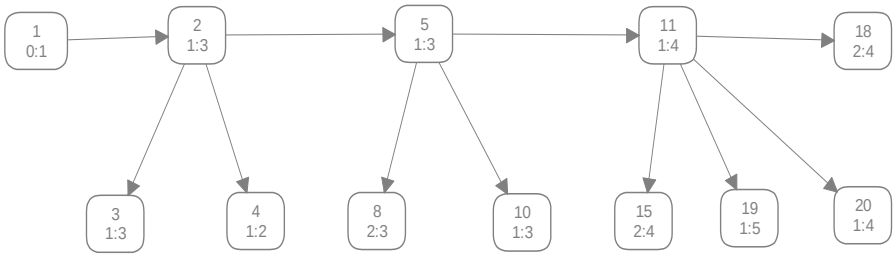

Figura 3.8: Grafo de Estado (Ejemplo Figura 3.4)

3.4 CPN Tools

Es una herramienta para editar, simular y analizar Redes de Petri Coloreadas (Ver Figura 3.9). Cuenta con un simulador y una herramienta de análisis de espacio de estado. Estados totales y parciales pueden ser generados y analizados. CPN Tools soporta modelos jerárquicos síncronos o asíncronos. Fue originalmente desarrollado en la Universidad de Aarhus 2000 a 2010[21]. Los arquitectos principales de la herramienta son Kurt Jensen, Søren Christensen, Lars M. Kristensen, y Michael Westergaard. Desde el otoño de 2010, CPN Tools es transferido al grupo de AIS, Universidad Tecnológica de Eindhoven , Holanda. [21]

CPN Tools es usado por más de 8000 usuarios en 140 países del mundo y puede ser utilizado en Windows y Linux. Incluye dos componentes principales, un editor gráfico y un componente de simulador de back-end. El editor gráfico que está escrito en el lenguaje académico, Beta , y el backend simulador está escrito en el estándar ML variante SML / NJ.

Los usuarios de CPN Tools trabajan directamente con la representación gráfica del modelo CPNs. La interfaz gráfica tiene menús desplegables, ya que está basado en técnicas de interacción como barra de herramientas.

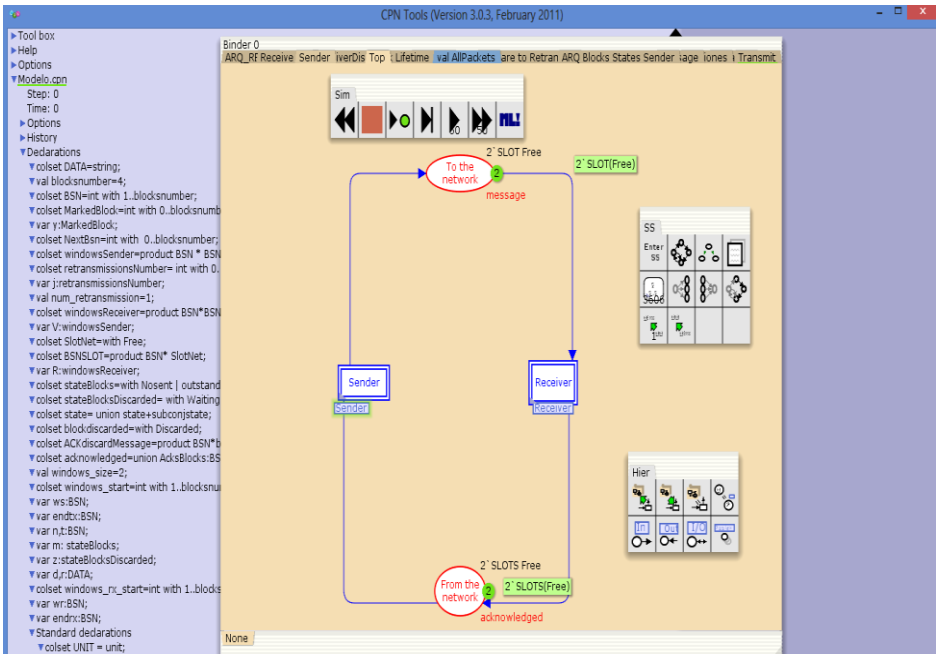

Figura 3.9: Herramienta CPN Tools.

El área rectangular en la izquierda es un índice. Este incluye el Tool box el cual está disponible para que los usuarios puedan manipular las declaraciones y módulos que constituyen el modulo CPNs. El Tool box incluye las herramientas para creación y copiando de los elementos básicos de la red CPNs, también contiene unas herramientas para selección del tamaño del diseño gráfico y la apariencia de los objetos en el modelo CPNs. El segundo grupo de herramientas es muy importante en el orden en que se pueden crear gráficos legibles en los modelos CPNs. La parte restante de la pantalla es el área de trabajo, que en este caso contiene cuatro carpetas (las ventanas rectangulares) y una circular del menú emergente.

Cada carpeta contiene una serie de elementos que se puede acceder haciendo click en las etiquetas en la parte superior de la carpeta (sólo un elemento es visible a la vez). Una clase contiene los elementos del modelo de CPNs, es decir, los módulos y

declaraciones, la otra clase contiene las herramientas que el usuario utiliza para la construcción y manipulación de los modelos de CPNs.

Los artículos pueden ser arrastrados desde el índice de las carpetas, y de una carpeta para otra carpeta del mismo tipo. Un menú circular marcado ha aparecido en la parte superior de la carpeta abajo a la izquierda. Los menús marcados son los menús contextuales que permiten seleccionar entre las posibles operaciones en un objeto determinado.

CPN Tools realiza la comprobación de sintaxis, tipo y mensajes de error siempre separa al usuario de una manera contextual junto al objeto que causa el error. La sintaxis de verificación y generación de código son elementales y se realizan en paralelo con la edición.

Esto significa que es posible ejecución de las piezas de un modelo, aunque el modelo no esté completo, y que cuando las partes de un modelo de CPN se modifican, una comprobación de sintaxis y generación de código se realizan sólo en los elementos que dependen de las partes que se modificaron. El principal resultado de la etapa de generación de código es la simulación código. El código de simulación contiene las funciones para inferir el conjunto de los eventos habilitados en un estado dado del modelo CPNs, y para calcular el estado resultante de la ocurrencia (ejecución) de un evento activado en un estado dado.

CPN Tools admite dos tipos de simulación: interactivo y automático. En un sistema de simulación interactivo, el usuario tiene todo el control de la simulación por lo que determina los pasos de la misma, mediante la selección entre los eventos habilitados en el estado actual.

CPN Tools muestra el efecto de la ejecución de un paso seleccionado en la representación gráfica del modelo de CPN. En una simulación automática el usuario especifica el número de pasos que deben ser ejecutadas y / o establece una serie de criterios de parada y puntos de interrupción. El simulador ejecuta automáticamente el modelo sin interacción del usuario para tomar decisiones al azar entre los eventos habilitados en los estados encontrados.

El simulador aprovecha herramientas basadas en un número de estructuras de datos avanzadas, para lograr una simulación eficiente de grandes modelos jerárquicos CPNs.

CPN Tools ofrece varios medios para analizar las propiedades del sistema tomando en cuenta los espacios de estados. El primer paso es por lo general crear un informe de espacio de estado que contiene las respuestas a un conjunto de propiedades estándar de comportamiento de los modelos de CPN, tales como la ausencia o presencia de bloqueos y el número mínimo y máximo de marcas sobre los lugares individuales. El gráfico de espacio de estado se puede generar de forma totalmente automática. También es posible para el usuario dibujar de forma interactiva su grafo de espacio de estado, seleccionando partes de un espacio de estado e inspeccionando los estados individuales y eventos.

CPN Tools implementa un conjunto de funciones de consulta que hace posible que el usuario pueda recorrer el espacio de estados de diferentes formas y por lo tanto pueda investigar las propiedades que dependen del sistema.

La idea básica es llevar a cabo una serie de simulaciones a lo largo del modelo, de manera de obtener datos acerca del rendimiento del sistema y hacer análisis de los mismos para mejorar el diseño de lo que se quiere modelar.

CAPÍTULO 4: METODOLOGIA

En esta sección, se presenta una descripción detallada de la metodología a seguir en este trabajo de investigación, siendo ésta la metodología que plantea Bellington J. [28] la cual describe cómo las especificaciones del estándar pueden ser modeladas, analizadas y comparadas.

Las actividades implicadas en el modelado y análisis de las especificaciones del servicio y del protocolo son similares y se muestran en la Figura 4.1 y se describen en las secciones siguientes. Una caja sombreada denota una actividad, que es particular a la especificación del estándar. No todas las actividades que son parte de la metodología y que son descritas en este capítulo son implementadas en este trabajo, debido a que están fueran del alcance del mismo. Las actividades que serán implementadas en este trabajo, se explican a continuación.

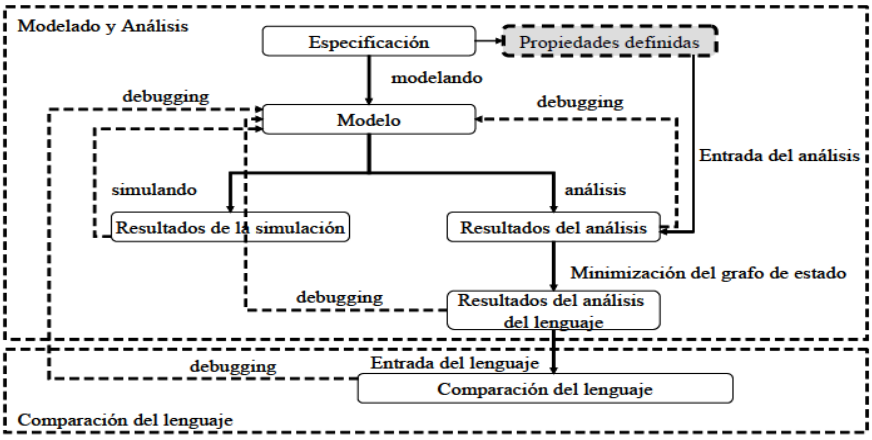

Figura 4.1: Diagrama que muestra las actividades de modelado, análisis y actividades de generación para las especificaciones del protocolo.

4.1. Especificación

La especificación describe el estándar usando, por ejemplo, narrativa, diagramas de bloque o tablas de estado. Ejemplos de los documentos que incluyen tales especificaciones son los documentos de los estándares del IEEE. En este caso trataremos con el estándar del IEEE 802.16 [7].

4.2 Modelo

Se desarrolla un modelo basado en la especificación del estándar a modelar, usando una técnica formal. La técnica formal usada en este trabajo para el modelado es las Redes de Petri Coloreadas, con la ayuda de la herramienta de software CPN Tools.

4.3 Propiedades Definidas

Una propiedad se refiere a una característica particular, que debe estar presente en un protocolo. Por ejemplo, Holzmann [9] define un conjunto de las características generales, que pueden aplicarse a cualquier protocolo, tales como: abrazos mortales inesperados.

4.4 Resultados de la Simulación

Las simulaciones se pueden utilizar para eliminar errores iniciales del modelo. Por ejemplo, la simulación automática o interactiva proporcionada por CPN Tools puede ayudar a encontrar errores tales como inscripciones erróneas en arcos y secuencias de evento del protocolo erróneas .Mientras se encuentren errores en la simulación, se modifica el modelo y las actividades desde simulación se repiten según lo mostrado en la Figura 4.1.

4.5 Resultados del Análisis

El análisis del modelo del proceso de retransmisión ARQ del estándar IEEE 802.16 se puede dividir en: verificación de las propiedades. La verificación de las propiedades requiere la prueba de las propiedades deseadas de IEEE 802.16.

Varios métodos para el análisis formal de los protocolos de comunicación se han definido (por ejemplo, análisis del grafo de estado, invariantes del sistema, lógica temporal, chequeo de modelos). Holzmann [9] describe algunos de estos métodos. Además, los múltiples ejemplos de las técnicas formales de análisis y de sus aplicaciones se pueden encontrar en los artículos presentados en los *Proceedings* de la *Joint International Conference on Formal Description Techniques for Distributed Systems and Communications Protocols (FORTE)* y el *Protocol Specification, Testing & Verification (PSTV)* [10].

CAPÍTULO 5: MODELADO DEL MECANISMO DE RETRANSMISION ARQ

5.1 Introducción

Se modela el mecanismo de retransmisión de bloques ARQ usando CPNs con la ayuda de CPN Tools. El modelo se basa en las especificaciones del estándar IEEE 802.16 respecto a este mecanismo, el cual es modelado y analizado basado en la metodología de verificación descrita en el Capítulo 4. En el Capítulos 2, se describe el funcionamiento de las máquinas de estados de las entidades emisora y receptora y de la operación ARQ. El documento del estándar IEEE 802.16 especifica el procedimiento de retransmisión de bloques ARQ para ser incluidos en los SDUs, lo cual es parte de la funcionalidad de la capa MAC específicamente la subcapa de parte común (CPS).

En este trabajo se modela la operación ARQ según se especifica en el estándar IEEE 802.16 [9]. La Figura 5.1 muestra el funcionamiento básico de la operación ARQ.

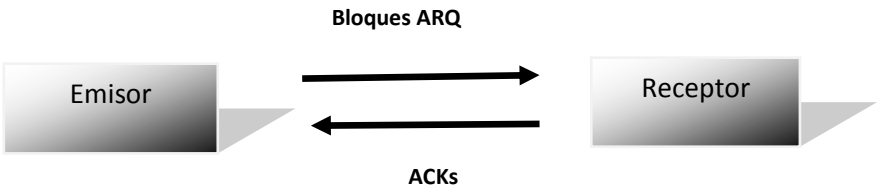

Figura 5.1: Operación ARQ

Las acciones o eventos que intervienen en el proceso son las siguientes:

1. Transmisión de los bloques ARQ
2. Retransmisión de los bloques ARQ
3. Descarte de los bloques ARQ
4. Recepción de reconocimiento(ACK)
5. Recepción de descarte de bloque ARQ(ACK de descarte)

A continuación se muestra una vista detallada del modelo de CPN, alcance, asunciones del modelo y decisiones de modelado.

5.2 Alcance

En este trabajo, son consideradas las características principales del mecanismo de retransmisión de bloques ARQ. Sin embargo, puesto que es un mecanismo complejo, el alcance del modelo se limita para simplificar la tarea de modelado y para hacer el modelo manejable para el análisis. La Tabla 1 resume estas características e indica si están incluidas en el modelado de la siguiente forma (SI) o no (NO).

Características mecanismo ARQ	Descripción	Presente en el modelo
Máquina de estados del Emisor	Establecimiento Conexión	NO
	Restablecimiento de la Conexión	NO
	Transmisión bloques	SI
	Retransmisión de bloques	SI
	Descarte de bloques por timeout (tiempo)	SI
	Recepción de reconocimientos(ACK selectivo)	SI
	Recepción de reconocimientos(ACK acumulativo)	NO
	Control de flujo	SI
	ARQ Feedback	NO
Receptor	Establecimiento Conexión	NO
	Restablecimiento Conexión	NO
	Recepción de bloques	SI
	Descarte de bloques duplicados y fuera de orden	SI
	Envío de ACK selectivo	SI
	Envío de ACK acumulativo	NO
	Control de flujo	SI
	ARQ Feedback	NO

Tabla 1: Alcance del modelo de la operación ARQ

5.3 Asunciones

En el modelo del mecanismo de retransmisión de bloques ARQ se asume que la conexión ha sido establecida y la operación ARQ está habilitada. Dentro de una conexión

no puede existir una mezcla de tráfico ARQ con tráfico no-ARQ. El ámbito de una instancia específica ARQ se limita a una conexión unidireccional. El enlace o red, que conecta dos nodos adyacentes, se asume perfecto, así que los bloques ARQ no pueden tener errores de bits. Por lo tanto, el procedimiento de checksum, no es requerido. No se toma en cuenta la pérdida de paquetes. Los mensajes enviados desde un nodo pueden no llegar a sus destinos en el mismo orden que fueron entregados.

5.4 Descripción del modelo de retransmisión de bloques ARQ

A continuación se explica detalladamente el funcionamiento de las máquinas de estados de la entidad emisora y receptora en relación al modelo realizado.

5.4.1 Máquina de estados del emisor

El funcionamiento del modelo de la máquina de estados del emisor (Ver Figura 4.2) es el siguiente:

a) Un bloque ARQ puede estar en uno de los siguientes estados*: Nosent, Outstanding, Waiting, WaitingSendMD, SendedMD, Discarded, Done*. El estado *Nosent* indica que el bloque ARQ no ha sido enviado. El estado *Outstanding* indica que el bloque ARQ ha sido enviado y se espera un reconocimiento (ACK). El estado *Waiting* indica que el bloque ha sido enviado, se ha vencido el temporizador *ARQ_RETRY_TIMEOUT* y aun no se ha recibido ACK por lo que el bloque espera para ser retransmitido. El estado *WaitingSendMD* indica que se ha vencido el temporizador *ARQ_BLOCK_LIFETIME* por lo cual el bloque se encuentra en espera para enviar un mensaje de descarte. El estado *SendedMD* indica que le mensaje de descarte ha sido enviado y se espera recibir un ACK de descarte por el receptor. El bloque ARQ cambiara de los estados *WaitingSendMD* o *SendedMD* al estado *Discarded* cuando se haya recibido un reconocimiento (ACK de descarte) para el

bloque del cual se envió el mensaje de descarte. El estado *Done* indica que el bloque fue recibido correctamente por el receptor.

b) Luego de que un bloque en estado *No Sent* es enviado, este pasa al estado *Outstanding* por un periodo de tiempo terminado en *ARQ_RETRY_TIMEOUT*. Una vez que el bloque está en el estado O*utstanding* puede ser reconocido mediante un ACK y cambiar al estado *Done*, ó modificar su estado a *Waiting* después de expirado el temporizador *ARQ_RETRY_TIMEOUT* o recibir un *NACK*, ó cambiar al estado *WaitingSendMD* en caso de que *ARQ_BLOCK_LIFETIME* haya expirado, y un mensaje de descarte será enviado al receptor.

c) Un bloque puede cambiar del estado *Waiting* a *Done* si se recibe un ACK. Asimismo el bloque puede cambiar de *Waiting* a WaitingSendMD luego de expirado el temporizador *ARQ_BLOCK_LIFETIME*.

d) El emisor envía primero los bloques en el estado *Waiting* y luego en el estado *No Sent.*

e) Cuando los bloques son retransmitidos, el bloque con el más bajo BSN es retransmitido primero.

f) Cuando un ACK es recibido, el emisor chequea la validez del BSN contenido en el mismo. Un BSN es válido si se encuentra en el intervalo *ARQ_TX_WINDOWS-START* a *ARQ_TX_NEXT_BSN* -1 inclusive. Si el BSN de un bloque está fuera del intervalo, el emisor ignorara el reconocimiento.

g) Cuando se recibe un ACK para el valor de BSN que está asignado a *ARQ_TX_WINDOWS_START* dicha variable debe ser incrementada.

h) Un mensaje de descarte es enviado al receptor siguiendo el fin del temporizador *ARQ_BLOCK_LIFETIME*. Se reenviará el mensaje de descarte cada vez que expire el temporizador *ARQ_RETRY_TIMEOUT* hasta que un reconocimiento haya sido recibido para el mensaje de descarte. Se asume para el modelado el envío de un número limitado de mensajes de descarte para un bloque.

5.4.2 Receptor

El funcionamiento del modelado de la entidad receptora (Ver Figura 2.10) es el siguiente:

a) El receptor mantiene una ventana deslizante definida como *ARQ_RX_WINDOWS_START*. Si se recibe un bloque cuyo BSN se encuentra dentro del rango de la ventana deslizante este será aceptado, de lo contrario rechazado.

b) Se deberá descartar los bloques duplicados.

c) El valor de la ventana deslizante se mantiene tal que *ARQ_RX_WINDOWS_START* apunte al BSN más bajo que no ha sido recibido.

d) Cuando un bloque con un BSN correspondiente al valor de *ARQ_RX_WINDOWS_START* es recibido, la ventana es actualizada tal que *ARQ_RX_WINDOWS_START* apunte al BSN del próximo bloque que se espera recibir.

e) Cuando la recepción de un bloque no conlleva al avance de *ARQ_RX_WINDOWS_START* debido a que aún no se ha sido recibido el bloque cuyo valor de BSN corresponde a esta variable, el temporizador *ARQ_RX_PURGE_TIMEOUT* para el bloque recibido será iniciado. Una vez concluido

este tiempo, *ARQ_RX_WINDOWS_START* es avanzada al BSN del próximo bloque que aún no ha sido recibido después del bloque marcado.

f) Cuando un mensaje de descarte es recibido por parte del emisor, se verificara que el bloque no haya sido recibido, si el bloque fue recibido correctamente se envía un ACK de reconocimiento para ese bloque, en caso contrario, se mueve la ventana deslizante al BSN del próximo bloque que se espera recibir, y se envía un ACK de descarte al emisor. Si el mensaje de descarte recibido es invalido, porque el BSN del bloque a descartar se encuentra fuera del rango de la ventana deslizante, este descarte será eliminado.

g) Para cada bloque recibido un reconocimiento a ACK será enviando al emisor. Para la realización del modelo únicamente fue considerado el envío de ACK selectivo, reconociendo solo un bloque por envío de ACK.

5.5 Jerarquía del Modelo

El modelo detallado del mecanismo de retransmisión de bloques ARQ consiste en nueve (9) módulos. La visión jerárquica se ha diseñado basada en la topología de árbol y la funcionalidad de las entidades del mecanismo de retransmisión de bloques ARQ en cada nodo está representada por los módulos *Top, Sender, Receiver* y sus correspondientes sub-módulos. En la Figura 5.2 se muestra en módulo superior de la jerarquía que incluye las transiciones de sustitución *Sender* y *Receiver* que modelan todo el mecanismo de retransmisión de bloques ARQ. Estas transiciones contienen además transiciones de substitución que modelan los procesos de transmisión, retransmisión y descarte. Estas transiciones de sustitución son: *Transmit, ARQ_RETRY, Prepare to Retransmit, Receive ACK, ReceiverDiscard, Block lifetime.*

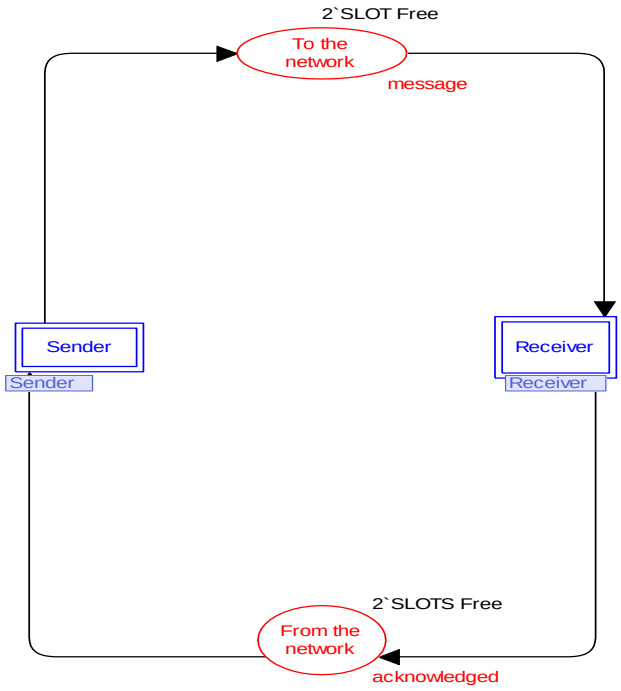

Figura 5.2: Top modelo de retransmisión de bloques ARQ

Cada módulo en el nivel inferior de la jerarquía puede incluir uno o más de los siguientes tipos de transiciones: transiciones de envío, retransmisión y descarte de bloques ARQ, siguiendo el diagrama de estados acorde al estándar IEEE 802.16.

5.6 Declaración Global

De la Figura 5.3 a la Figura 5.10 se muestran los conjuntos de colores (*colour sets*), las variables y las funciones del modelo. Se divide en las secciones siguientes: estados del emisor, mensajes del emisor, mensajes del receptor, variables y funciones.

5.6.1 Estados del Emisor

Esta sección define los estados de la entidad Emisor, los cuales se muestran en la Figura 5.3. El conjunto de colores *stateBlocks* es un tipo enumerado, que representa los estados en los que pueden estar los bloques ARQ durante el proceso de envío hacia el receptor. Una vez que los bloques ARQ son enviados, pueden ser retransmitidos o descartados, dependiendo de las circunstancias de la red. El conjunto de colores *stateBlocksDiscarded* define el subconjunto de estados *WaitingSendMD* y *SendedMD* relacionados con el proceso de envió de mensaje de descarte para un bloque ARQ.

1. *No Sent*: El bloque está listo para ser enviado, pero aún no ha sido enviado.

2. *Outstanding*: El bloque fue enviado y se espera por una confirmación de recepción por parte del receptor.

3. *Waiting*: Ha expirado el temporizador ARQ_RETRY_TIMEOUT para el bloque, por lo que este será retransmitido.

4. *WaitingSendMD*: Ha expirado el temporizador ARQ_BLOCK_LIFETIME, por lo cual un mensaje con órdenes de descarte para el bloque será enviado al receptor.

5. *SendedMD*: El mensaje con instrucciones de descarte del bloque ha sido enviado al receptor.

6. *Discarded*: El receptor ha recibido el mensaje con órdenes de descarte para el bloque, ha descartado el bloque y enviado una confirmación del descarte el emisor.

7. *Done*: El bloque ha sido recibido correctamente por parte de la entidad receptora.

```
colset stateBlocks=with nosent | outstanding | waiting |discarded|done;

colset stateBlocksDiscarded= with WaitingSendMD|SendedMD;
```

Figura 5.3: Estados del emisor

5.6.2 Mensajes del Emisor

El conjunto de colores que representan los mensajes del emisor se muestran en la Figura 5.4. El conjunto de colores *VARIATYPACKETS* representan la unión de los distintos mensajes que pueden ser transportados en el proceso de transmisión y retransmisión de los bloques ARQ.

Los colores *PacketsCD, PacketsCR* y *PacketsProseced* son subconjuntos del color *VARIATYPACKETS*. El color *PacketsCD* representa los bloques que están en proceso de descarte, contiene el BSN que es el identificador del bloque, seguido del estado perteneciente al subconjunto de estados relacionados con el proceso de envío de mensaje de descarte, representado por el color *stateBlocksDiscarded*, seguidamente del número de mensajes de descarte que han sido enviados, modelado por el color *numDiscardMessageSended* y por último los datos a enviar.

El color *PacketsCR* representa los bloques en proceso de retransmisión, contiene el BSN que es el identificador del bloque, seguido del estado del bloque representado por el

color *stateBlocks*, posteriormente el número de retransmisiones realizadas representado por el color *retransmissionsNumber* y por último los datos.

El color *PacketsProseced* representa los bloques que ya han sido procesados, es decir, enviados y dependiendo de las circunstancias pueden haber sido recibidos correctamente y finalizar en el estado *Done*, o descartados terminando en el estado *Discarded*.

El color *BSN* representa el BSN o identificador de un bloque ARQ, el cual inicia en uno y finaliza en el valor total de bloques ARQ a enviar.

El color *stateBlocksDiscarded* contiene los estados *WaitingSendMD* y *SendedMD*, utilizados para la gestión del envío de mensaje de descarte. Un bloque ARQ se encuentra en el estado *WaitingSendMD* cuando espera por el envío de un mensaje de descarte. Asimismo, un bloque se encuentra en el estado *SendedMD* luego de que el mensaje de descarte ha sido enviado.

El color *numDiscardMessageSended* representa el número de mensajes de descarte que han sido enviados para un bloque.

El color *stateBlocks* contiene los estados utilizados por la entidad emisora en el proceso de transmisión. Estos estados son: *nosent* indica que el bloque aún no ha sido enviado; *outstanding*: indica que el bloque fue enviado y se espera recibir una confirmación o ACK; *waiting*: indica que el bloque espera para ser retransmitido; *discarded*: indica que el bloque fue descartado, es decir, el tiempo de vida para este bloque ha expirado y no volverá a ser transmitido; *done:* indica que el bloque fue recibido correctamente.

El color *retransmissionsNumber* indica la cantidad máxima de retransmisiones que puede realizar la entidad emisora.

El color *DATA* representa los datos que se envían desde el emisor al receptor.

El color *idDM* representa un número entero que identifica un mensaje como mensaje de descarte.

El color *discardmessage* representa la estructura del mensaje de descarte que es enviado del emisor al receptor. Está conformado por el identificador de mensaje de descarte dado por la variable *idDM*, seguido del BSN del bloque a descartar.

```
colset VARIATYPACKETS=union SDPacket:PacketsCD+
SRPacket:PacketsCR+ProcessedPacket:PacketsProseced;

colset PacketsCD=product BSN * stateBlocksDiscarded* numDiscardMessageSended ;

colset PacketsCR=product BSN * stateBlocks *retransmissionsNumber* DATA ;

colset PacketsProseced=product BSN * stateBlocks ;

colset BSN=int with 1..blocksnumber;

colset stateBlocksDiscarded= with WaitingSendMD|SendedMD;

colset numDiscardMessageSended=int with 0..3;

colset stateBlocks=with nosent | outstanding | waiting |discarded|done;

colset retransmissionsNumber= int with 0..2;

colset DATA=string;

colset idDM=int with 34..34;

colset discardmessage=product  idDM * BSN;
```

Figura 5.4: Mensajes del emisor

5.6.3 Mensajes del Receptor

El conjunto de colores que representan los mensajes del receptor se muestran en la Figura 5.5. El color *acknowledged* representa los distintos mensajes que se pueden enviar al emisor, es decir, los ACK de reconocimiento de bloque y ACK de reconocimiento de mensaje de descarte.

El color *ACKdiscardMessage* representa el ACK para el mensaje de descarte. Quiere decir que el mensaje de descarte fue recibido, y el bloque fue descartado.

El color *blockdiscarded* contiene el estado *Discarded* que se utiliza para diferenciar los ACKs para mensaje de descarte, de los ACKs para los bloques recibidos.

El color SlotNet representa la disponibilidad de la red de transportar datos. Está compuesto de un enumerado con el valor *Free,* para indicar que el espacio en la red está libre y se puede enviar un bloque de datos.

```
colset acknowledged=union AcksBlocks:BSN+
ACKMessageDiscard:ACKdiscardMessage+SLOTS:SlotNet;

colset ACKdiscardMessage=product BSN*blockdiscarded;

colset blockdiscarded=with Discarded;

colset SlotNet=with Free;
```

Figura 5.5: Mensajes del receptor

5.6.4 Conjunto de Colores

La Figura 5.6 y Figura 5.7 muestra los conjuntos de colores definidos para la realización del modelado tanto para la entidad emisora como receptora.

El color *MarkedBlock* representa el BSN de un bloque ARQ que ha sido marcado, por haber llegado fuera de orden al receptor. El rango para el BSN del bloque marcado varía entre cero y la variable *blocksnumber*, que representa el total de los bloque a enviar.

El color *NextBsn* representa el BSN del próximo bloque a enviar.

El color *windowsSender* representa la ventana deslizante del emisor, está conformado por un producto de colores, donde el primer elemento representa el inicio de la ventana, el segundo elemento el fin de la ventana y el tercer elemento el próximo bloque a enviar.

El color *windowsReceiver* representa la ventana deslizante del receptor y está conformada por un producto de colores, donde el primer elemento representa el inicio de la ventana deslizante, el segundo elemento corresponde al fin de la ventana deslizante, el tercer elemento indica el BSN del último bloque descartado, el cuarto elemento indica el BSN del último bloque marcado.

Los conjuntos de colores *windows_start* y *windows_rx_start* representa los rangos numéricos de las ventanas deslizantes del emisor y receptor.

El color *BLOCK* consiste en el producto de los colores *BSN* y *DATA* y representa el contenido de un bloque ARQ.

El conjunto de colores *message* consiste en la unión de los colores *BLOCK*, *discardmessage* (Ver sección 5.6.2), *ACKdiscardMessage* y *SlotNet* (Ver sección 5.6.3).

El color *ListDiscardMessage* consiste en una lista de los mensajes de descarte enviados al receptor.

El color *Listbsndiscard* consiste en una lista de BSNs que representan ACKs para bloques y mensajes de descarte enviados y recibidos.

El color *ListBSN* representa una lista de bloques ARQ. Contiene el BSN o identificador y los datos.

```
colset MarkedBlock=int with 0..blocksnumber;

colset NextBsn=int with  0..blocksnumber;

colset windowsSender=product BSN * BSN * NextBsn ;

colset windowsReceiver=product BSN*BSN*NextBsn*NextBsn;

colset windows_start=int with 1..blocksnumber;

colset windows_rx_start=int with 1..blocksnumber;
```

Figura 5.6: Conjunto de colores del emisor y receptor

```
colset BLOCK=product BSN * DATA;

colset message= union
PACKET:BLOCK+PACKETDISCARD:discardmessage+ACK:BSNSLOT+ACKMenjDiscard:ACKdiscar
dMessage+SLOT:SlotNet;

colset ListDiscardMessage=list  discardmessage;

colset Listbsndiscard= list BSN;

colset ListBSN = list BLOCK;
```

Figura 5.7: Continuación Figura 5.6 Conjunto de Colores del Emisor y Receptor

5.6.5 Variables

En la Figura 5.8 y Figura 5.9 se pueden visualizar la definición de las variables del emisor.

La variable *AllPackets* pertenece al conjunto de colores VARIATYPACKETS y almacena los paquetes a ser enviados desde el emisor al receptor.

La variable *a* pertenece al conjunto de colores *numDiscardMessageSended* y contiene el número de mensajes de descarte que han sido enviados para un bloque.

La constante *MaxNumDiscardM* contiene el máximo número de mensajes de descarte que se puede enviar para un bloque.

Las variables *J,F,W,X,M,U* pertenecen al conjunto de colores *Listbsndiscard* y contienen listas de BSNs que representan ACKs recibidos.

La variable *type_message* pertenece al conjunto de colores *idDM* y contiene un identificador de mensaje de descarte, el número 34.

La variable *m* pertenece al conjunto de colores *stateBlocks* y almacena el estado en el que se encuentra un bloque luego de ser transmitido.

La variable *z* pertenece al conjunto de colores *stateBlocksDiscarded* y contiene el estado del bloque luego de haberse enviado un mensaje de descarte para este, es decir, indica si se acaba de enviar un mensaje de descarte para ese bloque o se espera para la realización del envío del mismo.

La variable *d* pertenece al conjunto de colores *DATA* y representan los datos a ser enviados en los bloques ARQ.

La constante *windows_size* representa el tamaño de la ventana deslizante.

La variable *V* pertenece al conjunto de colores *windowsSender* y representa la ventana deslizante del emisor.

La variable *j* pertenece al conjunto de colores *retransmissionsNumber* y contiene el número de retransmisiones realizadas.

La constante *num_retransmission* contiene la cantidad máxima de retransmisiones permitidas.

La variable *P* pertenece al conjunto de colores *ListBSN* y representa la lista de bloques en cola para ser retransmitidos.

La variable *ws* pertenece al conjunto de colores *BSN* y representa la ventana del emisor.

Las variables *n,t* pertenecen al conjunto de colores *BSN y* se utiliza para representar el BSN del bloque a enviar y el BSN de los ACKs enviados y recibidos.

La constante *blocksnumber* almacena el número de bloques a enviar.

La variable *L* pertenece al conjunto de colores *ListBSN y* representa la lista de bloques ARQ recibidos por el receptor.

La variable *i* pertenece al conjunto de colores *MarkedBlock* y se utiliza para almacenar el BSN de un bloque marcado.

La variable R pertenece al conjunto de colores *windowsReceiver* y representa la ventana deslizante del receptor.

```
val AllPackets = 1`SRPacket(1,Nosent,0,"A")++ 1`SRPacket(2,Nosent,0,"B")++
 1`SRPacket(2,Nosent,0,"B");
 var a:numDiscardMessageSended;
val MaxNumDiscardM=1;
var J,F,W,X,M,U: Listbsndiscard;
var type_message:idDM
```

Figura 5.8: Variables del emisor

```
var m: stateBlocks;
var z:stateBlocksDiscarded;
var d:DATA;
val windows_size=2;
var V:windowsSender;
var j:retransmissionsNumber;
val num_retransmission=1;
var P:ListBSN;
var ws:BSN;
var n,t:BSN;
val blocksnumber=3;
var L :ListBSN;
var R:windowsReceiver;
```

Figura 5.9: Variables emisor y receptor

5.6.6 Funciones

Las funciones se utilizan para simplificar las expresiones de los arcos y se visualizan en el apéndice 4. Estas funciones realizan tareas como: validar que los ACK recibidos en el emisor sean correctos; validar que los bloques recibidos en el receptor no estén duplicados; calcular el movimiento de los apuntadores de inicio y fin de las ventanas

deslizantes que mantienen la entidad emisora y receptora; seleccionar el menor BSN que se encuentre en cola de retransmisión; eliminar ACKs o mensajes de descarte duplicados.

5.7 Módulo Sender

El módulo *Sender* (Ver la Figura 5.10) está situada en el segundo nivel de abstracción e incluye las funciones principales que son realizadas por la entidad emisora, como lo son trasmitir, retransmitir los bloques ARQ, y recibir reconocimientos (ACK), añadido al manejo de los temporizadores *ARQ_RETRY_TIMEOUT* y *ARQ_BLOCK_LFETIME*. Contiene cinco plazas y cinco transiciones de sustitución.

La plaza *Windows Sender* representa la ventana deslizante, mediante la cual se realiza control de flujo para el envío de los bloques ARQ, y se lleva un control del último bloque enviado. La plaza *Blocks to Send* representan los bloques a ser enviados a través de la red y los diferentes estados por los que pasan durante el proceso de transmisión. La plaza *List of Blocks to Retransmit* representa la lista de BSN de los bloques a ser retransmitidos. La plaza *To The Network* mantiene los diferentes paquetes que pueden viajar a través de la red (bloques ARQ y mensajes de descarte). La plaza *From The Network* modela el tráfico de datos proveniente de la red (ACKs para bloques y ACKs para mensajes de descarte) enviados por el receptor. La transición de sustitución *Transmit* realiza el proceso de transmisión de los bloques ARQ. La transición de sustitución *ARQ_RETRY* modela el funcionamiento del temporizador ARQ_RETRY_TIMEOUT que toma la decisión de retransmitir bloques y mensaje de descarte en el emisor. La transición de sustitución *Prepare to Retransmit* modela el procedimiento de retransmisión de los bloques que se encuentren en estado *Waiting for retransmission*. La transición de sustitución *Block Lifetime* modela el vencimiento del temporizador ARQ_BLOCK_LIFETIME, por lo que está encargada del envío de mensajes de descarte al receptor para un bloque ARQ. La

transición de sustitución *Receive ACK* es la encargada de procesar los ACKs recibidos por parte del receptor y modificar el estado de los bloques a Done.

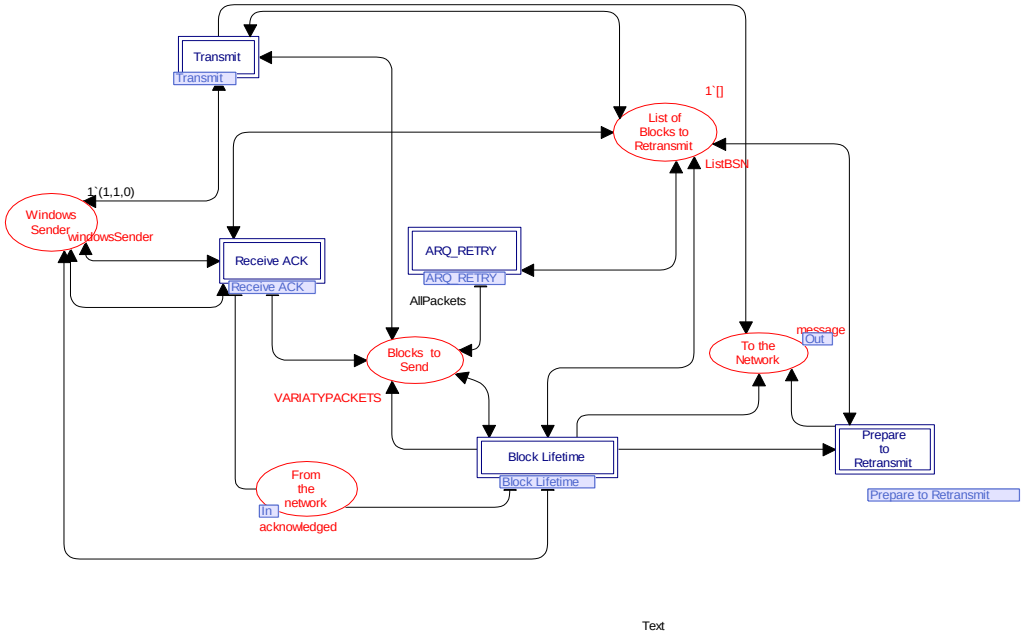

Figura 5.10: Módulo Sender

5.8 Módulo Receiver

El módulo *Receiver* (Ver Figura 5.11) modela los procedimientos que lleva a cabo el receptor para la recepción de bloques y envíos de ACK, así como también la eliminación de los bloques con BSN inválido o duplicado. Contiene seis plazas, tres transiciones de las cuales una es transición de sustitución. La transición *Receive Packets* tiene la función de recibir los bloques ARQ y comprobar que el BSN del bloque esté en el rango de la ventana deslízate, no esté duplicado y no haya sido descartado con anterioridad. Esta transición recibe los bloques provenientes del emisor a través de la plaza *From the Network* y una

vez que estos son reconocidos como válidos, un ACK será enviado al emisor, por medio de la plaza *To the Network*.

Una vez que el bloque es reconocido como válido, se procede a avanzar la ventana deslizante al próximo bloque que aún no ha sido recibido. Para que la ventana deslizante pueda ser avanzada es necesario que se haya recibido el bloque cuyo BSN corresponde con el inicio de la ventana deslizante, en caso contrario, no podrá avanzarse la misma y el bloque recibido será calificado como un bloque marcado, por lo que es almacenado en la plaza *Block Marked.* La transición *ARQ_PURGE TIMEOUT* representa a un temporizador con el mismo nombre (Ver sección 2.6.2.6), toma el bloque contenido en plaza *Block Marked* y realiza el avance de la ventana deslizante al BSN del próximo bloque que se espera recibir después del bloque marcado. La plaza *Windows Receiver* es la encargada de mantener los apuntadores al inicio y fin de la ventana deslizante, con el fin de verificar la validez de los bloques recibidos, así como llevar un control del último bloque descartado y/o marcado. La plaza *List of blocks received* almacena los bloques ARQ recibidos para poder realizar comprobaciones de bloques duplicados. La transición de sustitución *Discard block* es la encargada de eliminar bloques, ACKs para bloques y mensajes de descarte duplicados, así como realizar el descarte de los bloques ARQ para los cuales se haya recibido un mensaje de descarte.

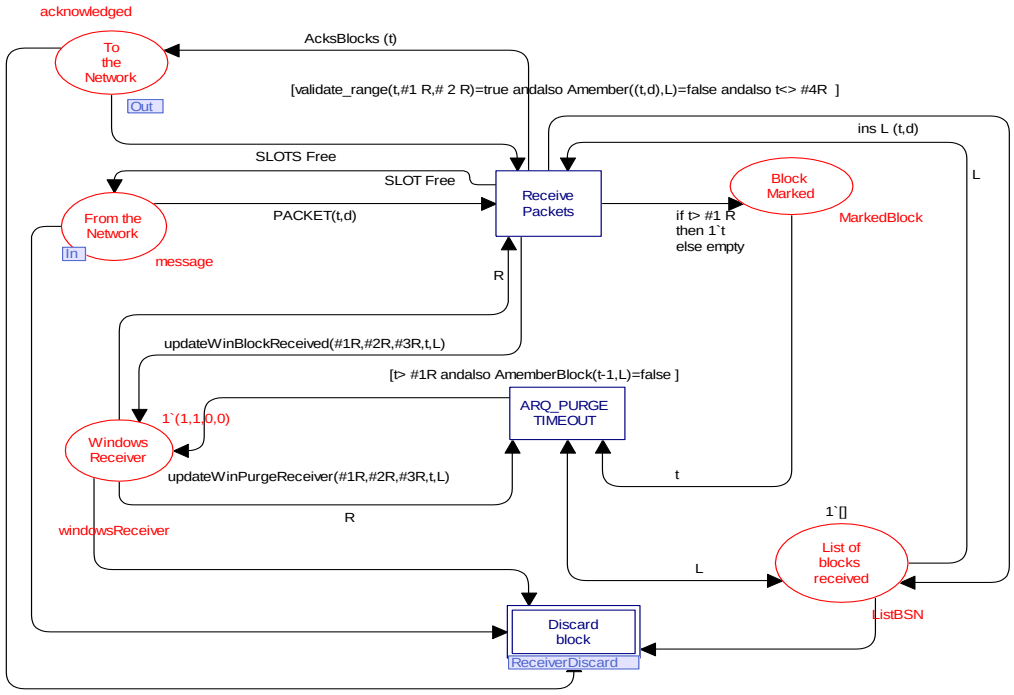

Figura 5.11: Módulo Receiver

5.9 Módulo Transmit

El módulo *Transmit* (Ver Figura 5.12) modela el envío de bloques ARQ a través de la red. Contiene dos transiciones y seis plazas. La plaza *Blocks to Send* representa los bloques a ser enviados y los distintos estados por los que pasa a lo largo del proceso. La plaza *List of Blocks to Retransmit* modela los BSN de los bloques en cola para ser retransmitidos. La plaza *Next BSN* representa el BSN del próximo bloque a ser enviado. La transición *Transmit* toma un bloque de la plaza *Blocks To Send* y lo envía al receptor a través de la plaza *To The Network*. La transición *Next Block to Send* selecciona el BSN del próximo bloque a ser enviado, valor que se almacena en la plaza *Next BSN*. La plaza *Limit Next BSN* limita la cantidad de marcas en la plaza *Next BSN*. La plaza *Windows Sender*

modela el inicio y fin de la ventana deslizante, así como también almacena el último bloque enviado, valor que es consumido por la transición *Next Block to Send*.

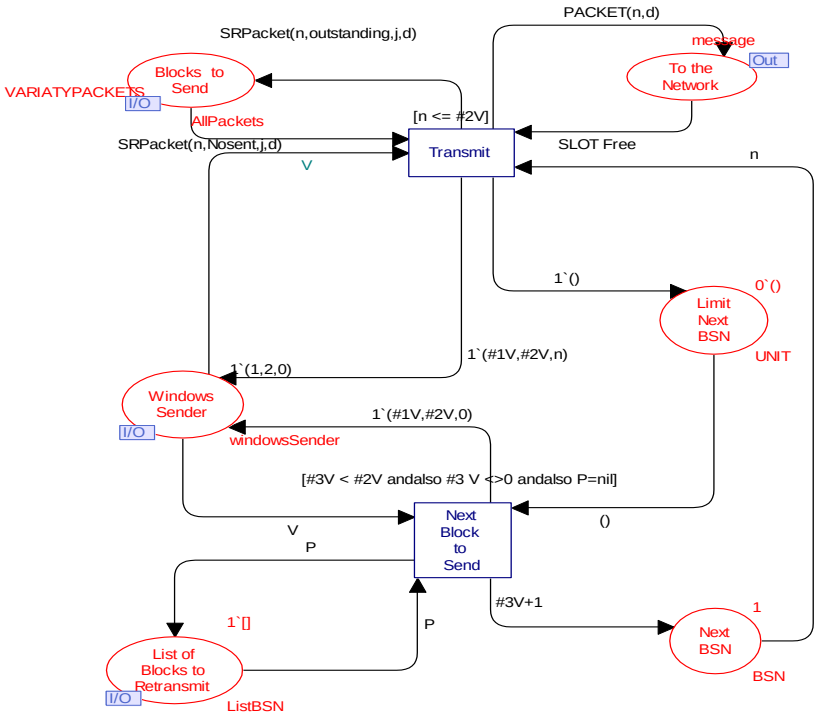

Figura 5.12: Módulo Transmit

5.10 Módulo ARQ_RETRY

El módulo *ARQ_RETRY* (Ver Figura 5.13) modela la retransmisión de bloques una vez expirado el temporizador *ARQ_RETRY_TIMEOUT*. Está compuesto de dos plazas y dos transiciones. La transición *ARQ_RETRY timeout o NACK* representa el temporizador con el mismo nombre. La ocurrencia de esta transición implica que el bloque ARQ será retransmitido, y el estado del bloque debe cambiar de *Outstanding* a *Waiting for*

retransmission. Esta información de estado es almacenada en la plaza *Blocks to Send* y el bloque a retransmitir es añadido a la plaza *List of Blocks to Retransmit*, que es una lista de los bloques a ser retransmitidos.

La transición *ARQ_RETRY_TIMEOUT for retransmit discard message*, mantiene un temporizador para el envío de los mensajes de descarte, ya que una vez enviado un mensaje de descarte se espera recibir un reconocimiento del mismo de parte del receptor, si transcurrido este tiempo el reconocimiento no es recibido, entonces el mensaje de descarte será retransmitido. Por consiguiente, la ocurrencia de la transición *ARQ_RETRY_TIMEOUT for retransmit discard message* implica la retransmisión del mensaje de descarte para un bloque ARQ, y dependerá de que la guarda de esta transición sea evaluada a verdad, es decir, que el número máximo de retransmisiones permitidas por bloque no se haya alcanzado.

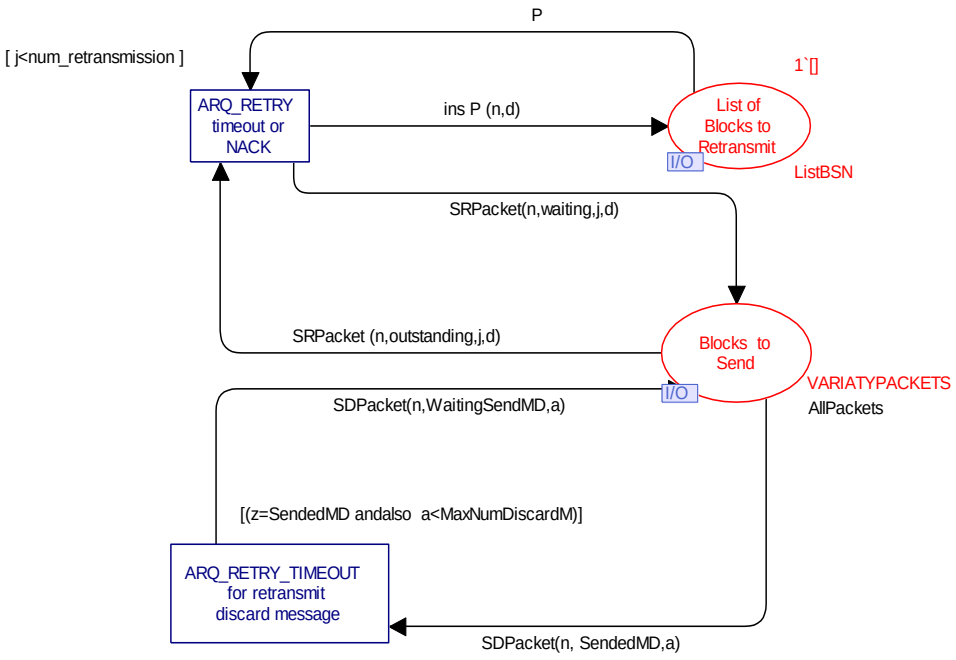

Figura 5.13: Módulo ARQ_RETRY

5.11 Módulo Prepare to Retransmit

El módulo *Prepare to Retransmit* (Ver Figura 5.14) modela los procedimientos de selección del próximo bloque a ser retransmitido, así como la realización de la retransmisión del mismo. Contiene dos transiciones y cinco plazas. Las transición *Select block to retransmit* selecciona de la lista de bloques a retransmitir representada por la plaza *List of Blocks to Retransmit, el* valor del BSN más bajo, y lo almacena en la plaza *block to retransmit*; ésta selección se realiza si la guarda presente en la transición *Select block to retransmit* es evaluada a verdad, es decir, se valida que esté permitido realizar la retransmisión de un bloque, esto se debe a que existe un número máximo de retransmisiones permitidas. La transición *Retransmit* envía el bloque almacenado en la plaza *block to retransmit* ARQ al receptor a través de la plaza *To The Network*. La plaza *Blocks to Send* mantiene los estados de los bloques ARQ durante el proceso de retransmisión. Por último, la plaza *Limit Next Block to Retransmit* limita el número de bloques seleccionados para ser retransmitidos.

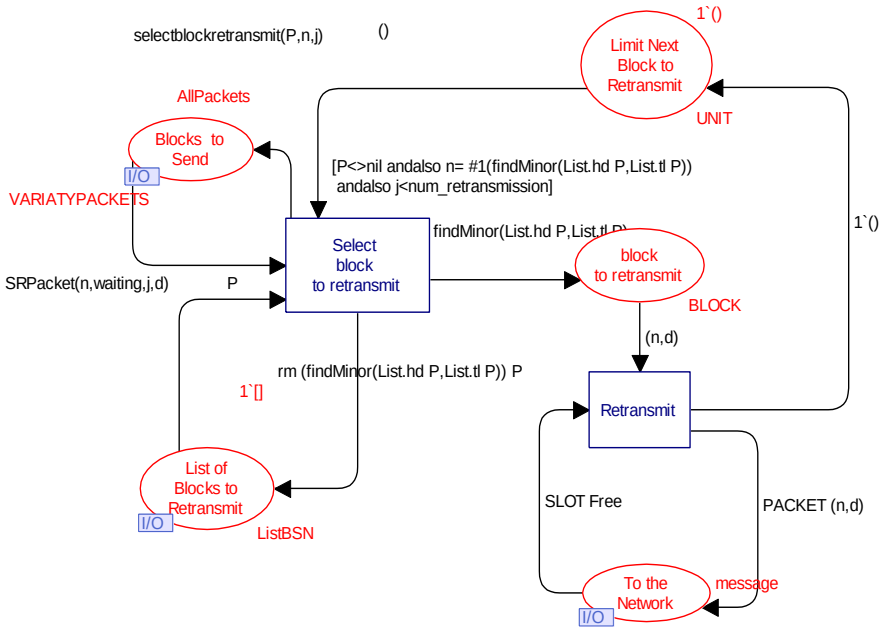

Figura 5.14: Módulo Prepare to Retransmit

5.12 Módulo Block Lifetime

El módulo *Block Lifetime* (Ver Figura 5.15) modela la secuencia de acciones relacionadas con el envío de mensaje de descarte. Contiene cuatro transiciones y cinco plazas. La transición *Block Lifetime/ Send discard message* modela el vencimiento del temporizador *ARQ_BLOCK_LIFETIME* para un bloque, lo que implica que ese bloque será descartado. Para ello se envía un mensaje de descarte al receptor por medio de la plaza *To the Network*, y se espera recibir un ACK de reconocimiento para el mismo que es manejado por el Módulo Receive ACK (Ver sección 5.13) La transición *Retransmit Discard Message* retransmitirá el mensaje de descarte tantas veces como se indique en la variable *MaxNumDiscardM* una vez vencido el temporizador ARQ_RETRY_TIMEOUT modelado en el Módulo ARQ_RETRY (Ver sección 5.10). La transición *Discard Invalide ACK* elimina los ACKs recibidos por medio de la plaza *From The Network* para bloques cuyos BSNs seas

inválidos o duplicados. La plaza *Blocks to Send* mantiene el estado del bloque ARQ en *outstanding* una vez retransmitido. La plaza *Windows Sender* maneja los apuntadores a inicio y fin de la ventana deslizante, permite realizar la validación de rango para los ACKs recibidos, y así realizar la eliminación de los ACKs con rango inválido.

Una vez que se dispara la transición *Block Lifetime/ Send discard message* para un bloque, si ese bloque se encuentra en la plaza *List of Blocks to Retransmit* esperando para ser retransmitido, será eliminado de esta plaza y ya no será retransmitido, pues el tiempo de vida para este bloque se ha agotado y un mensaje de descarte debe ser enviado al receptor.

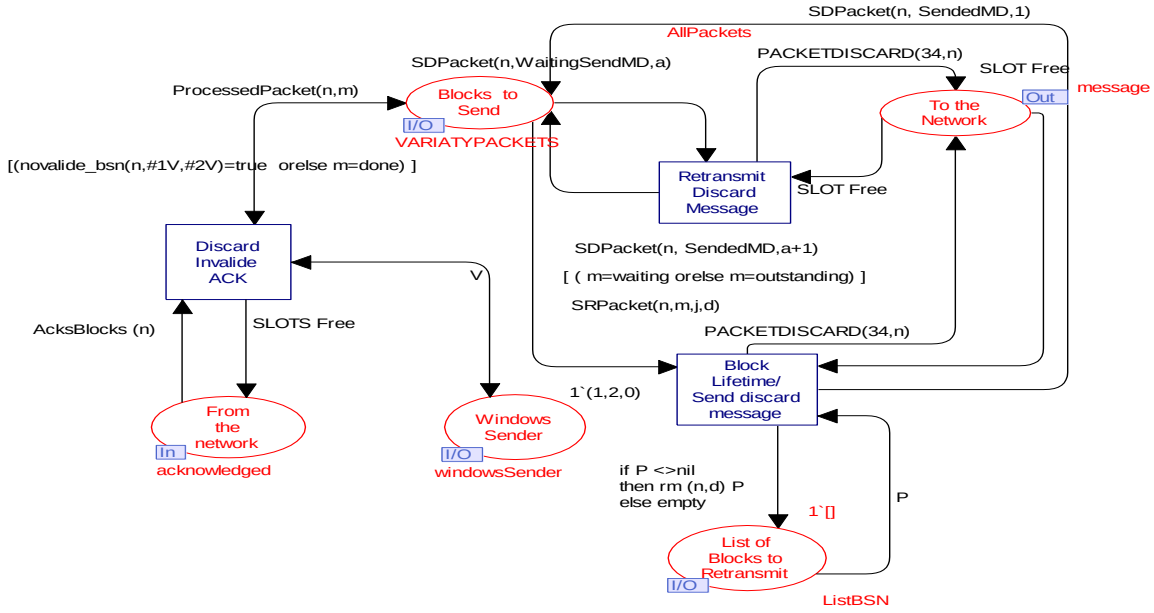

Figura 5.15: Módulo Block Lifetime

5.13 Módulo Receive ACK

El módulo Receive ACK (Ver Figura 5.16) modela el mecanismo de recepción de ACKs en el emisor. Contiene cinco plazas y cuatro transiciones. La transición *Receive ACK for Blocks Sended Retransmitted* valida que el BSN indicado en el ACK recibido para el bloque ARQ se encuentre en el rango de la ventana deslizante, para cuyo caso deberá avanzar la ventana deslizante, al siguiente bloque que aún no ha sido recibido. Esta transición se disparará para recibir los ACKs de los bloques que se encuentren en los estados *Outstanding* o *Waiting for retransmission* almacenados en la plaza *Blocks To Send*. La transición *Receive ACK for Blocks in process to discard* representa la recepción de los ACKs de reconocimiento de bloque, para algún bloque para el cual fue enviado un mensaje de descarte por vencimiento de temporizador *ARQ_BLOCK_LIFETIME*, pero este descarte no fue confirmado por el receptor, ya que el receptor ha recibido el bloque correctamente; por esta razón, el proceso de retransmisión de mensaje de descarte es interrumpido y el bloque cambiará de los estados *SendedMD* o *WaitingSendMD* a *Done*, y se realiza el movimiento de la ventana deslizante al próximo bloque cuyo reconocimiento no haya sido recibido.

La transición *Receive discarded message Ack* recibirá los ACKs correspondiente a uno o varios mensajes de descarte enviados, cambiará el estado del bloque *de SendedMD, WaitingSendMD a Discarded* y realizará el movimiento de la ventana deslizante para que apunte al próximo bloque para el cual aún no se ha recibido ningún ACK como confirmación de llegada en el receptor.

La plaza *List Ack Received* representa la lista de ACKs recibidos. La plaza *List ACK discard Message* almacena los ACKs para mensaje de descarte que han sido recibidos. La plaza *Blocks to Send* mantiene el estado de los bloques una vez recibido los ACKs para mensaje de descarte. La plaza *Windows Sender* permite validar que los ACKs recibidos se

encuentren dentro del rango de la ventana deslizante. La plaza *From the Network* permite la recepción de los ACKs provenientes del receptor.

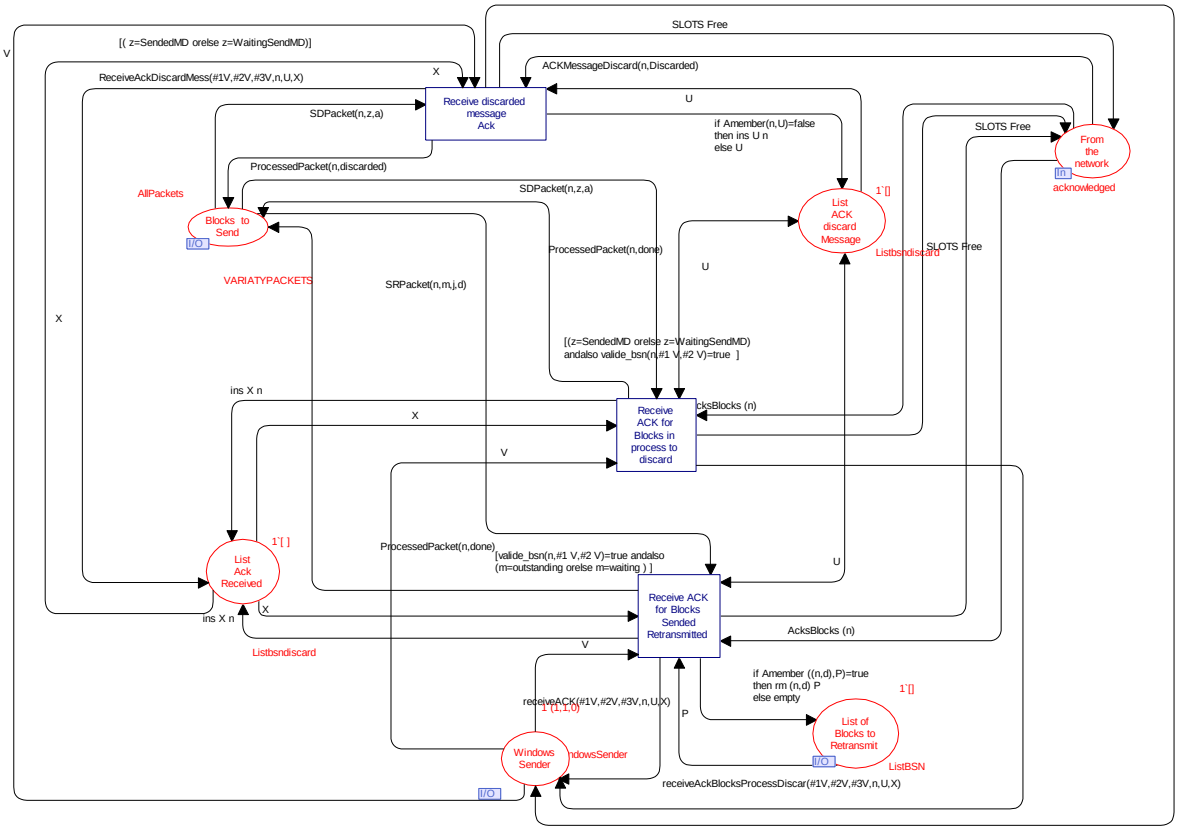

Figura 5.16: Módulo Receive ACK

5.14 Módulo ReceiverDiscard

El módulo *ReceiverDiscard* (Ver Figura 5.17) modela el procedimiento para recibir un mensaje de descarte de parte del emisor, así como eliminar bloques inválidos o duplicados. Contiene y cuatro plazas y cinco transiciones. La transición *Move Windows by discard message* recibe un mensaje de descarte de parte del emisor por medio de la plaza

From the Network, comprueba que el bloque no haya sido recibido, es decir, que no se encuentre en la plaza *List of blocks received* y avanza la ventana deslizante al siguiente bloque que se espera recibir, valor que es almacenado en la plaza *Windows Receiver*. La transición *Discard No valide blocks y discard invalide discarded blocks* eliminan los bloques y mensajes de descarte respectivamente, cuyo BSN no se encuentra en el rango de la ventana deslizante. La transición *discard blocks by message* recibe un mensaje de descarte para un bloque a través de la plaza *From the Network* y comprueba que el BSN del bloque especificado en el mensaje de descarte haya sido recibido, es decir, se encuentre en la plaza *List of blocks received*; en este caso, se elimina el mensaje de descarte y se envía un ACK para ese bloque al emisor. La transición *Discard duplicate blocks* elimina los bloques duplicados, es decir que ya se encuentren en la plaza *List of blocks received***.** La plaza *Windows Receiver* permite realizar el movimiento de la ventana deslizante una vez recibido y validado un mensaje de descarte para un bloque. La plaza *From the Network* permite la recepción del mensaje de descarte para los bloques ARQ. La plaza *To the Network* permite el envío del ACK para mensaje de descarte para un bloque.

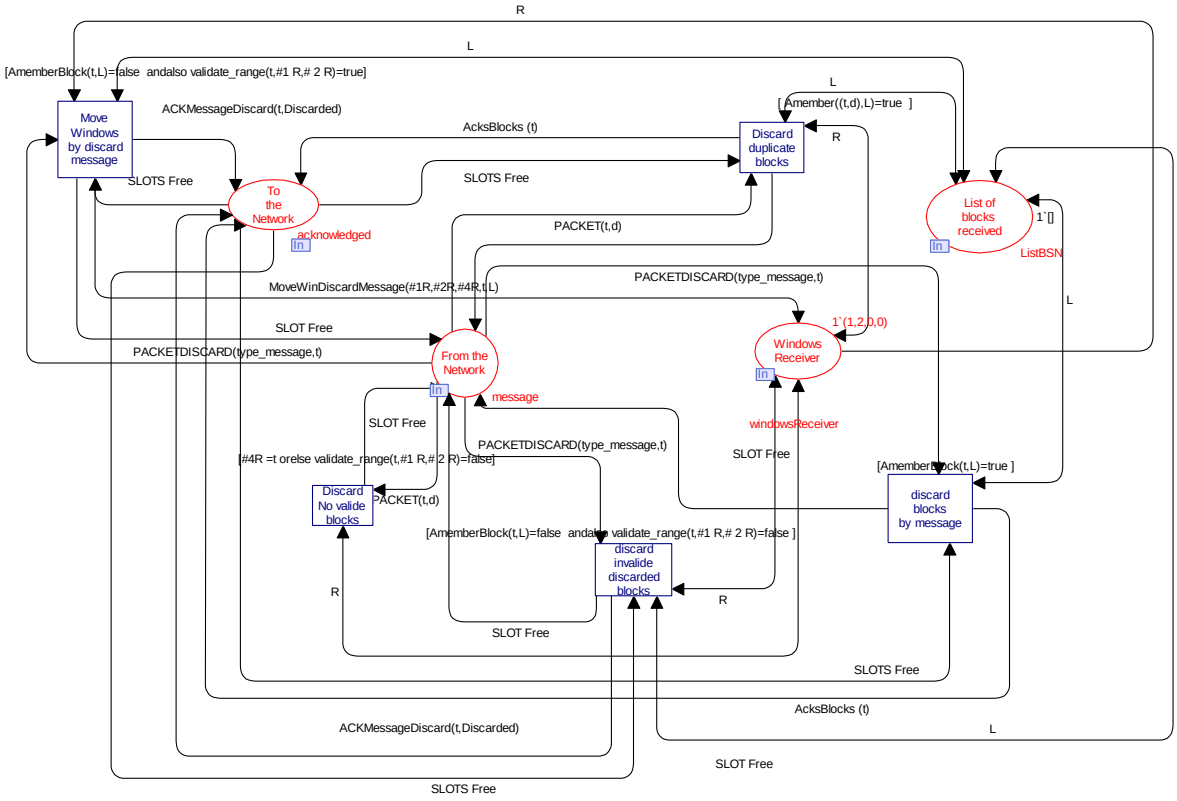

Figura 5.17: Módulo ReceiverDiscard

CAPÍTULO 6: ANÁLISIS DEL MODELO DEL MECANISMO DE RETRANSMISIÓN ARQ

6.1 Introducción

En este capítulo, se analiza el modelo del mecanismo de retransmisión de bloques ARQ usando el método del grafo de estado. En este trabajo, las principales propiedades de comportamiento o dinámicas de las CPNs descritas en el Capítulo 3 y las propiedades estándares se analizan para comprobar si la operación ARQ se comporta según lo esperado. El objetivo de este capítulo es detallar las aproximaciones para el análisis y presentar los resultados del análisis.

Todos los grafos de estado presentados en este capítulo son generados por una máquina Windows 7 con 2 Gb de memoria y un procesador Intel Atom 1.66Ghz.

6.2 Modificación del Modelo CPN para el Análisis

El modelo CPN del mecanismo de retransmisión de bloques ARQ puede generar un grafo de ocurrencia infinito cuando los mecanismos de vaciado de listas son incluidos. Adicionalmente un número arbitrario de marcas representando mensajes de *paquetes y ACKs* pueden estar en las plazas de comunicación (*From The Network, To the Network*), debido a que el número de marcas que pueden estar en las plazas de comunicación no se puede determinar. Para hacer más realista y para obtener un grafo de estado finito, se limitaron las capacidades de las plazas de comunicación.

Los conjuntos de colores *message* y *acknowleged* se modifican para indicar el número de *slots* vacíos en los *buffers* de comunicación intermedios. Los *slots* indican cuántos mensajes o bloques ARQ pueden estar en la red antes de ser procesados por la

entidad de emisora o receptora correspondiente. Figura 6.1 muestra los conjuntos de colores modificados: *message* y *acknowleged*, en la declaración global.

El conjunto de colores SLOT Free indica un *slot* vacío. La Figura 6.1 ilustra las modificaciones hechas para limitar las capacidades. El remitente puede enviar un mensaje a la red, si la plaza *To the Network* contiene al menos un *slot* vacío (es decir *1`SLOT Free*).

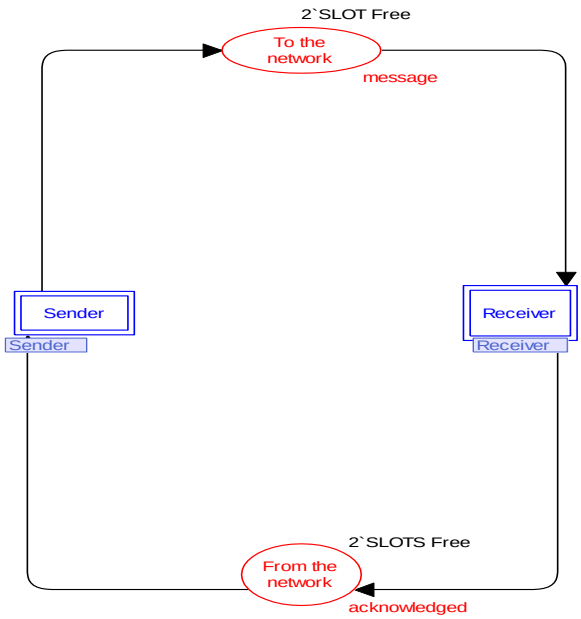

Figura 6.1: Marcas de las plazas To the Network y From the Network

De igual forma para limitar el número de marcas que están en las plazas *Transmit* y *Prepare to Retransmit* de la entidad emisora en un momento dado, se agregaron las plazas *Limit Next BSN* y *Limit Next Block to Retransmit* respectivamente, las cuales controlan la ocurrencia de las transiciones que agregan marcas a estas plazas, como se muestra en la Figura 6.2 y Figura 6.3.

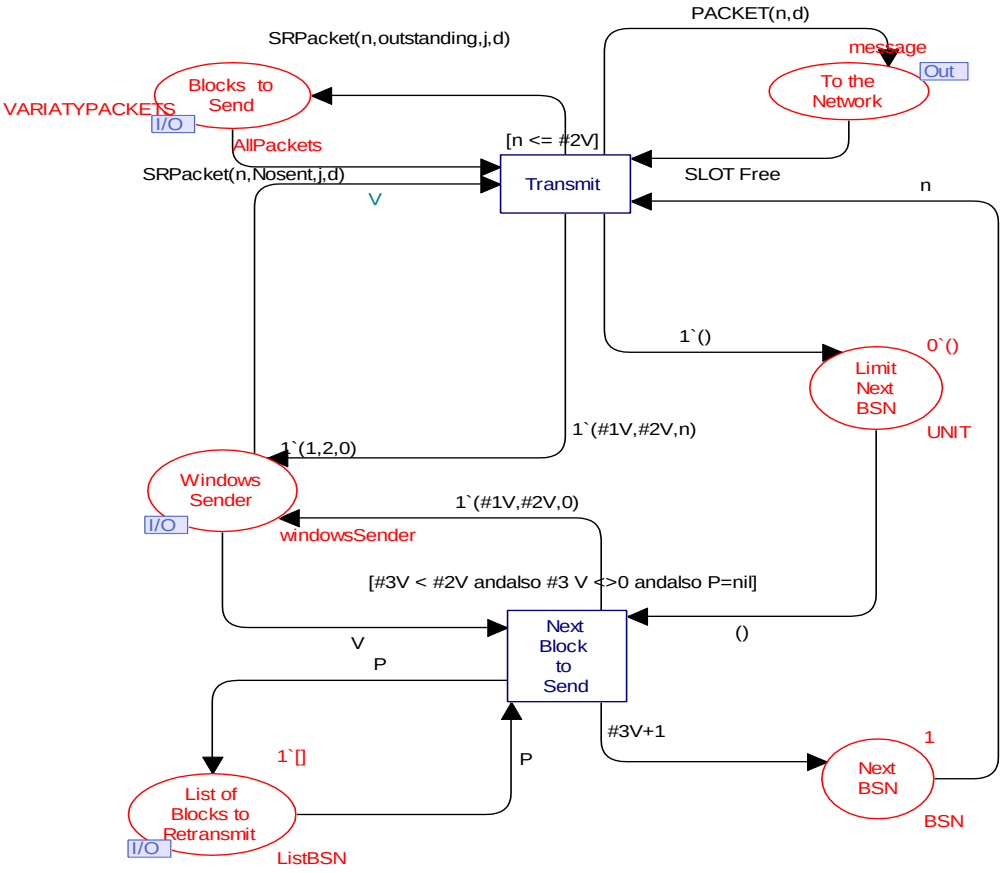

Figura 6.2: Plaza Limit Next BSN

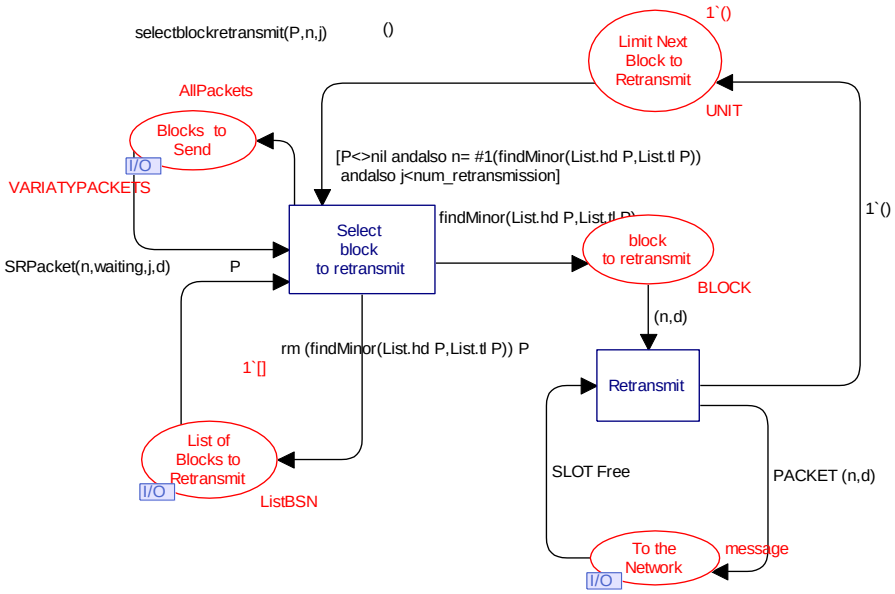

Figura 6.3: Plaza Limit Next Block to retransmit

6.3 Valores Iniciales de la Simulación

Los valores iniciales para la simulación han sido determinados en función de parámetros tales como: número de bloques a enviar, número de retransmisiones permitidas, número de mensajes de descarte permitidos, tamaño de la ventana deslizante y capacidad de la red. En la Tabla 2 se visualizan los diferentes valores iniciales que son representados como variables en el modelado del mecanismo de retransmisión de bloques ARQ. El valor inicial de simulación *Nro. Bloques* a enviar está relacionado con el marcado inicial de la plaza *Blocks to Send*. El valor inicial de simulación *Nro. Slots* (capacidad de la red) está relacionado con el mercado inicial de las plazas *To the Network, From the Network.* Asimismo, el valor inicial de simulación *Tam. Ventana* está relacionado con el marcado inicial de las plazas, *Windows Sender, Windows Receiver*.

Nro. Valor Inicial	Nro. Bloques a Enviar	Nro. Retransmisiones Permitidas	Nro. Descartes Permitidos	Nro. Slots (Capacidad de la red)	Tam. Ventana
1	2	1	1	2	1
2	2	1	2	3	1
3	2	1	1	3	1
4	2	1	1	2	2
5	2	1	2	2	2
6	2	2	1	2	2
7	3	1	1	2	1
8	3	2	1	2	1
9	3	2	1	3	1
10	3	1	2	3	1
11	3	2	2	3	1
12	3	1	1	2	2
13	3	2	1	2	2
14	3	1	2	2	2
15	4	1	1	2	2
16	4	1	1	2	1
17	4	1	2	2	1
18	4	2	1	2	1

Tabla 2: Valores Iniciales

6.4 Marcado Inicial

El marcado inicial del modelo incluye la distribución inicial de las marcas en cada una de las plazas del modelo, en la Tabla 3 se muestra el marcado inicial para las plazas del emisor y receptor, con respecto al valor inicial número quince (15), por ser éste el marcado inicial más completo en cuanto al número de bloques a enviar, permitiendo visualizar el funcionamiento de la ventana deslizante con un tamaño de dos bloques.

Plaza	Marcado Inicial
To the Network	2`SLOT(Free)
From the Network	2`SLOT(Free)
Blocks to Send	1`SRPacket(1,Nosent,0,"A")++ 1`SRPacket(2,Nosent,0,"B")++ 1`SRPacket(3,Nosent,0,"C")++ 1`SRPacket(4,Nosent,0,"D")
Windows Sender	1`(1,2,0)
List of Blocks to Retransmit	1`[]
List ACK Discard Message	1`[]
List ACK Received	1`[]
Next BSN	1`1
Limit Next Block to Retransmit	1`()
Windows Receiver	1`(1,2,0,0)
List of blocks received	1`[]

Tabla 3: Marcado inicial según valor inicial Nro. 15

Las plazas *To the Network y From the Network* que son comunes a la entidad emisora y receptora, permiten el envío de datos entre emisor y receptor, y son inicializadas con la marca *2`SLOT(Free)*.

6.4.1 Marcado Inicial del Emisor

Para la plaza *Blocks to Send* las marcas iniciales son: *1`SRPacket(1,Nosent,0,"A"*), *1`SRPacket(2,Nosent,0,"B"*),*1`SRPacket(3,Nosent,0,"C"*), *1`SRPacket(4,Nosent,0,"D"*).

La plaza *Windows Sender* representa la ventana deslizante, es inicializada en la marca *1`(1,2,0)* .

La plaza *List of Blocks to Retransmit, List ACK discard Message, List Ack Received,* se inicializan en la marca 1`[] , lo cual indica que las listas están vacías.

La plaza *Next BSN* se inicializa en la marca 1`1, indicando que el próximo bloque a enviar es el primero, es decir, el bloque con BSN igual a 1.

La plaza *Limit of Block to Retransmit* se inicializa en 1`(), ya que al principio de la simulación no se tiene ningún bloque para retransmitir.

6.4.2. Receptor

En la plaza *Windows Receiver* que representa la ventana deslizante se tiene la marca inicial *1`(1,2,0,0)* .

La plaza *List of blocks received* se inicializa en 1`[] indicando que está vacía, porque ningún bloque ha sido recibido todavía.

6.5 Estadística del Grafo de Ocurrencia

Para el análisis del modelo se generaron los reportes de estado completos. La información estadística que incluye el tamaño del OG y el grafo SCC se muestran en la Tabla 4.

Nro. Condición Inicial	Nro. Nodos	Nro. Arcos	Nro. Marcados Muertos	Tiempo (segundos)
1	373	944	4	1
2	2084	6820	9	1
3	541	1633	4	1
4	2155	6069	13	1
5	5004	14665	24	3
6	4149	12565	13	2
7	2084	6085	8	1
8	3085	9436	8	1
9	10864	43767	8	11
10	22967	85260	27	55
11	34158	130030	27	116
12	18135	56573	41	32
13	31590	103789	41	93
14	41217	125826	120	154
15	134545	450147	158	3196
16	8743	27484	16	6
17	23080	66330	119	35
18	12080	39200	16	39200

Tabla 4: Estadísticas del grafo de ocurrencia

En la Tabla 4 se puede observar los resultados de la simulación en relación a las condiciones iniciales de la Tabla 2, en la cual se toman en cuenta el envío de dos, tres y cuatro bloques ARQ, así como tamaños de ventana deslizante de uno y dos bloques, la realización de una o dos retransmisiones y envío de mensaje de descarte, y una capacidad de dos ó tres slots en la red.

6.6 Chequeo de las Propiedades Generales

Las propiedades generales de las CPNs descritas a continuación han sido investigadas para comprobar la validez del modelo y depurar el mismo. Tales propiedades son las de acotamiento y vivacidad.

6.6.1 Acotamiento

Las cotas enteras y las cotas de los multi-conjuntos son analizadas para las plazas del modelo. Esta información es resumida en las tablas de la 5 a la 11. Para este análisis de cotas se toma el resultado de la simulación con cuatro bloques (es decir, para el valor inicial Nro. 15), envío de un mensaje de descarte y realización de una retransmisión por bloque, por ser la simulación más completa, en cuanto a número de bloques a enviar y tamaño de ventana deslizante; para el resto de las simulaciones con dos y tres bloques los resultados son similares. Estos resultados son esperados y proporcionan la visión, y el nivel de confianza sobre el correcto funcionamiento del modelo.

Las cotas enteras superiores e inferiores describen el máximo y el mínimo número de marcas que pueden tener una plaza en un momento dado de la simulación, mientras que el valor de las cotas de los multi-conjuntos indica todas las marcas posibles en una plaza presente en el modelado de la máquina de estados del emisor y el receptor.

Plaza	block_to_retransmit	List_ACK_discard_Message			
Cota Entera Superior	1	1			
Cota Entera Inferior	0	1			
Cota de los Multiconjuntos	1`(1,"A")++ 1`(2,"B")++ 1`(3,"C")++ 1`(4,"D")	1`[]++ 1`[1]++ 1`[1,2]++ 1`[1,2,3]++ 1`[1,2,3,4]++ 1`[1,2,4]++ 1`[1,2,4,3]++ 1`[1,3]++ 1`[1,3,2]++	1`[1,3,2,4]++ 1`[1,3,4]++ 1`[1,4]++ 1`[1,4,3]++ 1`[2]++ 1`[2,1]++ 1`[2,1,3]++	1`[2,1,3,4]++ 1`[2,1,4]++ 1`[2,1,4,3]++ 1`[2,3]++ 1`[2,3,4]++ 1`[2,4]++ 1`[2,4,3]++	1`[3]++ 1`[3,2]++ 1`[3,2,4]++ 1`[3,4]++ 1`[4]++ 1`[4,3]

Tabla 5: Acotamiento de las plazas block_to_retransmit y List_ACK_discard_Message del modelo de operación ARQ

Plaza	Block_Marked	List_Ack_Received			
Cota Entera Superior	2	1			
Cota Entera Inferior	0	1			
Cota de los Multiconjuntos	1`2++ 1`3++ 1`4	1`[]++ 1`[1]++ 1`[1,2]++ 1`[1,2,3]++ 1`[1,2,3,4]++ 1`[1,2,4]++ 1`[1,2,4,3]++	1`[1,3]++ 1`[1,3,2]++ 1`[1,3,2,4]++ 1`[1,3,4]++ 1`[1,4]++ 1`[1,4,3]++ 1`[2]++ 1`[2,1]++ 1`[2,1,3]++	1`[2,1,3,4]++ 1`[2,1,4]++ 1`[2,1,4,3]++ 1`[2,3]++ 1`[2,3,4]++ 1`[2,4]++ 1`[2,4,3]++	1`[3]++ 1`[3,2]++ 1`[3,2,4]++ 1`[3,4]++ 1`[4]++ 1`[4,3]

Tabla 6: Acotamiento de las plazas Block_Marked y List_Ack_Received del modelo de operación ARQ

Plaza	List_of_blocks_received	
Cota Entera Sup	1	
Cota Entera Inferior	1	
Cotas de los Multiconjuntos	1`[]++ 1`[(1,"A")]++ 1`[(1,"A"),(2,"B")]++ 1`[(1,"A"),(2,"B"),(3,"C")]++ 1`[(1,"A"),(2,"B"),(3,"C"),(4,"D")]++ 1`[(1,"A"),(2,"B"),(4,"D")]++ 1`[(1,"A"),(2,"B"),(4,"D"),(3,"C")]++ 1`[(1,"A"),(3,"C")]++ 1`[(1,"A"),(3,"C"),(2,"B")]++ 1`[(1,"A"),(3,"C"),(2,"B"),(4,"D")]++ 1`[(1,"A"),(3,"C"),(4,"D")]++ 1`[(1,"A"),(4,"D")]++	1`[(1,"A"),(4,"D"),(3,"C")]++ 1`[(2,"B")]++ 1`[(2,"B"),(1,"A")]++ 1`[(2,"B"),(1,"A"),(3,"C")]++ 1`[(2,"B"),(1,"A"),(3,"C"),(4,"D")]++ 1`[(2,"B"),(1,"A"),(4,"D")]++ 1`[(2,"B"),(1,"A"),(4,"D"),(3,"C")]++ 1`[(2,"B"),(3,"C")]++ 1`[(2,"B"),(3,"C"),(4,"D")]++ 1`[(2,"B"),(4,"D")]++ 1`[(2,"B"),(4,"D"),(3,"C")]++ 1`[(3,"C")]++ 1`[(3,"C"),(2,"B")]++ 1`[(3,"C"),(2,"B"),(4,"D")]++ 1`[(3,"C"),(4,"D")]++ 1`[(4,"D")]++ 1`[(4,"D"),(3,"C")]

Tabla 7: Acotamiento de las plazas Data_Received y List_of_blocks_received del modelo de operación ARQ

Plaza	Windows_Receiver		From_the_network
Cota Entera Superior	1		2
Cota Entera Inferior	1		2
Cota de los Multiconjuntos	1`(1,2,0,0)++ 1`(1,2,0,2)++ 1`(2,3,0,0)++ 1`(2,3,0,3)++ 1`(2,3,1,0)++ 1`(2,3,1,3)++ 1`(3,4,0,0)++ 1`(3,4,0,3)++ 1`(3,4,0,4)++ 1`(3,4,1,0)++ 1`(3,4,1,2)++ 1`(3,4,1,3)++ 1`(3,4,1,4)++ 1`(3,4,2,0)++ 1`(3,4,2,3)++	1`(3,4,2,4)++ 1`(3,4,3,0)++ 1`(3,4,3,4)++ 1`(3,4,4,0)++ 1`(3,4,4,3)++ 1`(3,4,4,4)++ 1`(4,4,0,0)++ 1`(4,4,1,0)++ 1`(4,4,2,0)++ 1`(4,4,3,0)++ 1`(4,4,3,2)++ 1`(4,4,4,0)++ 1`(4,4,4,2)	2`AcksBlocks(1)++ 2`AcksBlocks(2)++ 2`AcksBlocks(3)++ 2`AcksBlocks(4)++ 1`ACKMessageDiscard((1,Discarded))++ 1`ACKMessageDiscard((2,Discarded))++ 1`ACKMessageDiscard((3,Discarded))++ 1`ACKMessageDiscard((4,Discarded))++ 2`SLOTS(Free)

Tabla 8: Acotamiento de las plazas Windows_Receiver y From_the_network del modelo de operación ARQ

Plaza	Blocks_to_Send		To_the_network
Cota Entera Superior	4		2
Cota Entera Inferior	4		2
Cota de los Multi Conjuntos	1`COMPLETEPACKET((1,SendedMD,1,"A"))++ 1`COMPLETEPACKET((2,SendedMD,1,"B"))++ 1`COMPLETEPACKET((3,SendedMD,1,"C"))++ 1`COMPLETEPACKET((4,SendedMD,1,"D"))++ 1`NOCOMPLETE((1,Nosent,0,"A"))++ 1`NOCOMPLETE((1,outstanding,0,"A"))++ 1`NOCOMPLETE((1,outstanding,1,"A"))++ 1`NOCOMPLETE((1,waiting,0,"A"))++ 1`NOCOMPLETE((2,Nosent,0,"B"))++ 1`NOCOMPLETE((2,outstanding,0,"B"))++ 1`NOCOMPLETE((2,outstanding,1,"B"))++ 1`NOCOMPLETE((2,waiting,0,"B"))++ 1`NOCOMPLETE((3,Nosent,0,"C"))++ 1`NOCOMPLETE((3,outstanding,0"C"))++	1`NOCOMPLETE((3,outstanding,1,"C"))++ 1`NOCOMPLETE((3,waiting,0,"C"))++ 1`NOCOMPLETE((4,Nosent,0,"D"))++ 1`NOCOMPLETE((4,outstanding,0,"D"))++ 1`NOCOMPLETE((4,outstanding,1,"D"))++ 1`NOCOMPLETE((4,waiting,0,"D"))++ 1`FINALPACKET((1,discardedA))++ 1`FINALPACKET((1,done))++ 1`FINALPACKET((2,discardedA))++ 1`FINALPACKET((2,done))++ 1`FINALPACKET((3,discardedA))++ 1`FINALPACKET((3,done))++ 1`FINALPACKET((4,discardedA))++ 1`FINALPACKET((4,done))	2`PACKET((1,"A"))++ 2`PACKET((2,"B"))++ 2`PACKET((3,"C"))++ 2`PACKET((4,"D"))++ 1`PACKETDISCARD((34,1))++ 1`PACKETDISCARD((34,2))++ 1`PACKETDISCARD((34,3))++ 1`PACKETDISCARD((34,4))++ 2`SLOT(Free)

Tabla 9: Acotamiento de las plazas Blocks_to_Send y To_the_network del modelo de operación ARQ

Plaza	Windows_Sender	List_of_Blocks_to_Retransmit
Cota Entera Superior	1	1
Cota Entera Inferior	1	1
Cota de los Multiconjuntos	1`(1,2,0)++ 1`(1,2,1)++ 1`(1,2,2)++ 1`(2,3,0)++ 1`(2,3,1)++ 1`(2,3,2)++ 1`(2,3,3)++ 1`(3,4,0)++ 1`(3,4,2)++ 1`(3,4,3)++ 1`(3,4,4)++ 1`(4,4,0)++ 1`(4,4,3)++ 1`(4,4,4)	1`[]++ 1`[(1,"A")]++ 1`[(1,"A"),(2,"B")]++ 1`[(2,"B")]++ 1`[(2,"B"),(1,"A")]++ 1`[(2,"B"),(3,"C")]++ 1`[(3,"C")]++ 1`[(3,"C"),(2,"B")]++ 1`[(3,"C"),(4,"D")]++ 1`[(4,"D")]++ 1`[(4,"D"),(3,"C")]

Tabla 10: Acotamiento de las plazas Windows_Sender y List_of_Blocks_to_Retransmit del modelo de operación ARQ

Plaza	Limit_Next_BSN	Next_BSN
Cota Entera Superior	1	1
Cota Entera Inferior	0	0
Cota de los Multiconjuntos	1`()	1`1++ 1`2++ 1`3++ 1`4

Tabla 11: Acotamiento de las plazas Limit_Next_BSN y Next_BSN del modelo de operación ARQ

La plaza *Blocks_to_Send* tiene cota superior e inferior de cuatro marcas, es decir, los cuatro bloques involucrados en el proceso de transmisión, los cuales pasaran por distintos estados, generando variedad de marcas.

Las plazas *List_ACK_discard_Message*, *List_Ack_Received*, *List_of_blocks_received*, *List_of_Blocks_to_Retransmit* tienen la cota superior e inferior de uno, ya que las listas inicialmente están vacías, y se le irán añadiendo elementos durante la ejecución de la simulación, pero la plaza siempre mantendrá la misma lista.

Las plazas *Windows_Receiver*, *Windows_sender* tienen una cota superior e inferior de uno, ya que los apuntadores a inicio y fin de la ventana deslizante se irán actualizando con el envío de bloques y recepción de ACK, manteniéndose en estas plazas.

La plaza *block_to_retransmit,* tiene como cota inferior cero y superior uno, esto se debe a que el tipo de marca que puede contener esta plaza representa un bloque ARQ a ser retransmitido, e inicialmente ningún bloque puede ser retransmitido, dado que no ha sido transmitido por primera vez.

La plaza *Limit_Next_BSN* tiene como cota inferior cero y superior uno, no tiene marcado inicial y una vez que haya ocurrido la transición *Transmit* contendrá la marca 1`(), que a su vez habilita transición *Next Block to Send* para seleccionar el BSN del próximo bloque a enviar.

La plaza *Next BSN* tiene cota inferior y superior de uno, ya que almacena el BSN del próximo bloque a enviar. El primer bloque a enviar será el que contenga BSN igual a uno.

6.6.2 Propiedades Locales y de Vivacidad

6.6.2.1 Marcados Muertos

Un Marcado Muerto, es un marcado sin elementos de asociación habilitados, es decir, ninguna transición puede ocurrir a partir de dicho marcado. En el modelo existen marcados muertos cuyo resultado es completamente acorde a lo esperado, puesto que al finalizar la simulación las plazas quedaran con marcas que representan estados terminales.

6.6.2.1.1 Condiciones Generales Marcados Muertos

Como resultado del análisis del modelo de retransmisión de bloques ARQ se obtuvieron marcados muertos para las distintas condiciones o valores iniciales. La Tabla 12 muestra las características que deben tener los marcados muertos, es decir las marcas esperadas en cada una de las plazas al final de la simulación, para que se pueda afirmar que estos marcados muertos son correctos de acuerdo al diseño del modelo.

Plaza	Marcado Muerto
To the Network	Cantidad Slots (Capacidad Red): 2,3 Ej: 2`SLOT(Free)
From the Network	Cantidad Slots (Capacidad Red): 2,3 Ej: 2`SLOT(Free)
Blocks to Send	BSN del bloque enviado, seguido de su estado terminal. Ej: 1`ProcessedPacket((1,discarded))++  1`ProcessedPacket((2,done))++ 1`ProcessedPacket((3,discarded))++  1`ProcessedPacket((4,done))++
Windows Sender	Ventana deslizante del emisor. Contiene: inicio ventana, fin ventana, próximo bloque a enviar. Ej: 1`(4,4,4)
List of Blocks to Retransmit	Lista de bloques para retransmitir. Debe estar vacía, lo que indica que todos los bloques en espera para retrasmitirse fueron retransmitidos. Ej:1`[]
List ACK Discard Message	Lista de ACK para mensaje de descarte. Debe contener los BSNs de los bloques que fueron descartados, los cuales deben coincidir con los BSN y el estado *discarded* en la plaza *Blocks to Send* Ej: 1`[1,3]

Tabla 12: Condiciones Generales Marcados Muertos

Plaza	Marcado Muerto
List ACK Received	Lista de ACKs recibidos. Debe contener los BSNs de los bloques recibidos por el receptor, los cuales deben coincidir con los BSN y el estado *done* en la plaza *Blocks to Send* Ej: 1`[2,4]
Next BSN	Contiene el próximo bloque a enviar, que al final de la simulación debe ser el último bloque enviado Ej: 1`4
Limit Next Block to Retransmit	Limita la cantidad de marcas que puede tener la plaza Block to Retransmit Ej: 1`()
Windows Receiver	Ventana deslizante del receptor. Contiene: inicio ventana, fin ventana, último bloque descartado, último bloque marcado. Ej: 1`(4,4,3,0)
List of blocks received	Lista de los bloques recibidos por el receptor. Debe coincidir con los BSN y el estado *done* en la plaza *Blocks to Send* Ej: [(2,"B"),(4,"D")]

Tabla 13: Continuación Condiciones Generales Marcados Muertos

6.6.2.1.1 Marcados Muertos para Simulación con Valor Inicial Nro.15

Para la simulación del envío de cuatro bloques correspondiente al valor inicial Nro. 15 de la Tabla 2, en el cual se realiza una retransmisión y se envía un mensaje de descarte. Para obtener los números de todos los marcados muertos se utilizó la función

ListDeadMarkings() la cual es una primitiva de la herramienta CPN Tools en lenguaje ML que retorna una lista con los números de los nodos que contienen marcados muertos. Posteriormente, se dibujó cada nodo de la lista resultante, en la herramienta CPN Tools utilizando los menús relativos al cálculo y dibujo del gráfico de estado, para obtener la información de las marcas o estados terminales de las plazas en cada nodo. Se obtuvieron 158 marcados muertos los cuales se enumeran a continuación:

{9988,9963,96930,96861,96367,96298,95630,8849,8842,8799,8774,81707,81699,81484,81252,81197,81189,80974,80515,8008,7984,7971,79424,79411,7745,7603,7596,75773,75760,7553,7528,68334,68290,68277,66111,66090,65684,65663,65432,65141,64162,64092,60385,60315,52689,52619,51951,50373,50366,50323,50298,50044,50037,49994,49969,49933,49638,49625,49609,49187,48929,48921,48706,48469,46030,46017,45669,45661,45446,45373,45355,45137,45124,41306,41293,39106,39051,39043,38828,38369,37278,37265,35797,35776,35091,33359,33290,33026,33005,32853,32847,32813,32790,32608,32539,29410,29341,27540,27519,27297,26997,26018,25948,25356,25332,25319,25092,24950,24943,24900,24875,24726,23429,23171,23163,22948,22868,22861,22818,22793,22595,22587,22372,20186,20178,19963,18775,18768,18725,18700,18667,18369,18356,18340,17918,17660,17652,17437,15564,15543,15180,15159,13475,13454,11693,11672,109410,109397,108882,108838,108825,10423,10281,10274,10231,10206,10038,10031}.

En la Figura 6.4 se puede visualizar la lista de marcados muertos resultantes de ejecutar la función *ListDeadMarkings()*.Sin embargo, el dibujo no muestra todos los marcados muertos solamente los primeros cien, por lo cual se utilizó la función *OGtoGraphviz.ExportNodes* para exportar a un archivo de blog de notas la lista completa de los marcados muertos, de la siguiente manera *OGtoGraphviz.ExportNodes ("Marcados_Muertos.txt" ,ListDeadMarkings())* donde el primer parámetro de la función indica el nombre del archivo en el que se exportará los datos, y el segundo parámetro representa los datos a ser exportados en este caso, la lista de marcados muertos que es

obtenida mediante la función *ListDeadMarkings().*El contenido del archivo *Marcados_Muertos.txt se puede visualizar en el apéndice A.6.*

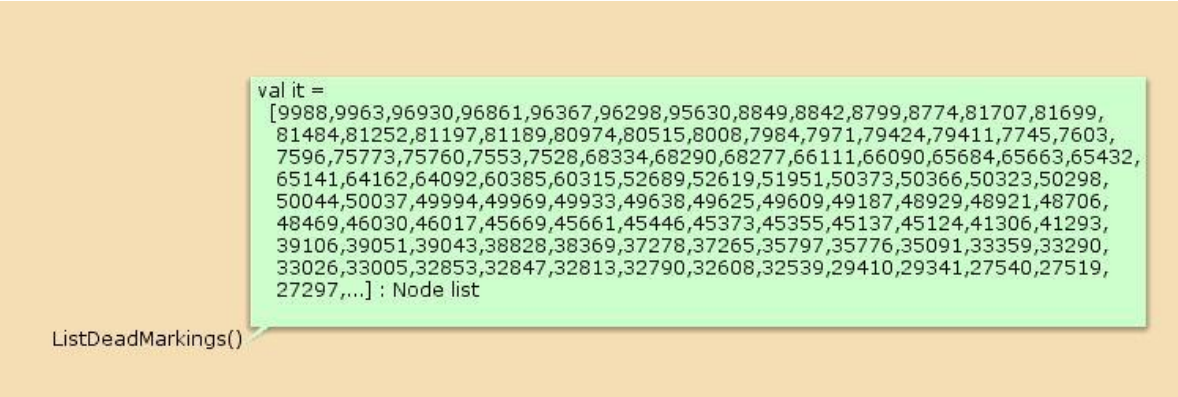

Figura 6.4: Lista de Marcados Muertos

Del conjunto de marcados muertos se describen los marcados 9988, 81484,66111, 45373, ya que representan distintos escenarios resultantes de la simulación para el valor inicial nro. 15, el resto de los marcados muertos contienen resultados muy similares y se pueden visualizar en el apéndice A.5.

El marcado muerto 9988 se muestra en la Tabla 14 y en la Figura 6.5 se puede visualizar el dibujo del nodo referente a este marcado en la herramienta CPN Tools. Para este marcado muerto la plaza *Blocks_to_Send* finaliza con las marcas *1'ProcessedPacket(1,done), 1'ProcessedPacket(2,done),1'ProcessedPacket(3,done), 1'ProcessedPacket(4,done)* lo cual indica que los bloques uno, dos, tres y cuatro fueron recibidos por el receptor. La ventana deslizante del emisor representada por la plaza *Windows Sender* finaliza con el valor de 1'(4,4,4),lo cual indica que todos los bloques fueron enviados. La plaza *List_of_blocks_received* finaliza con la marca 1'[(1,"A")],

[(2,”B”)], [(3,”C”)], [(4,”D”)]. La plaza *Windows_Receiver* finaliza con la marca 1’(4,4,0,0), la plaza *List_ACk_received* finaliza con la marca 1’[2,1,4,3] y la plaza *List_ACK_discard_Message* finaliza con la marca 1’[] lo indica que ningún bloque ARQ fue descartado .

Plaza	Marcado
Blocks_to_Send	1’ProcessedPacket (1,done) 1’ProcessedPacket (2,done) 1’ProcessedPacket (3,done) 1’ProcessedPacket (4,done)
Windows_Sender	1’(4,4,4)
List_of_blocks_received	1’[(1,”A”), (2,”B”), (3,”C”), (4,”D”)]
Windows_Receiver	1’(4,4,0,0)
List_ACk_received	1’[2,1,4,3]
List_ACK_discard_Message	1’[]

Tabla 14: Marcados Muertos 9988 para la simulación con 4 bloques ARQ

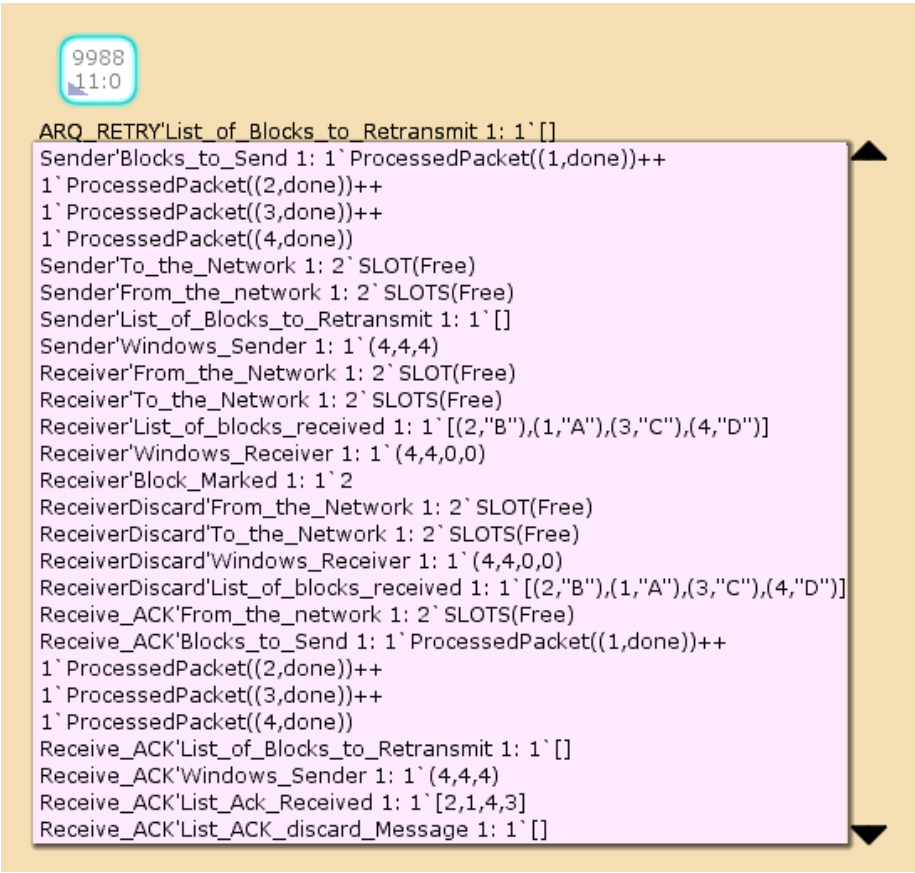

Figura 6.5: Marcados muertos Nodo 9988

El marcado muerto 81484 se muestra en la Tabla 15 y en la Figura 6.6 se puede visualizar el dibujo del nodo referente a este marcado en la herramienta CPN Tools. Para este marcado muerto la plaza *Blocks_to_Send* finaliza con las marcas 1'ProcessedPacket (1,discarded), 1'ProcessedPacket(2,discarded), 1'ProcessedPacket(3,done), 1'Processed Packet (4,discarded), lo cual indica que los bloques uno, dos y cuatro fueron descartados y el bloque tres fue recibido por el receptor. La ventana deslizante del emisor representada por la plaza *Windows Sender* finaliza con el valor de 1'(4,4,4), lo cual indica que todos los bloques fueron enviados. La plaza *List_of_blocks_received* finaliza con la marca 1' [(3,"C")]. La plaza *Windows_Receiver* finaliza con la marca 1'(4,4,4,0), la plaza

List_ACk_received finaliza con la marca 1'[3] ya que el emisor recibió el ACK correspondiente al bloque tres, y la plaza *List_ACK_discard_Message* finaliza con la marca 1'[2,1,4] lo indica que estos bloques ARQ fueron descartados.

Plaza	Marcado
Blocks_to_Send	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)
Windows_Sender	1'(4,4,4)
List_of_blocks_received	1'[(3,"C")]
Windows_Receiver	1'(4,4,4,0)
List_ACk_received	1'[3]
List_ACK_discard_Message	1'[2,1,4]

Tabla 15: Marcados Muertos 81484 para la simulación con 4 bloques ARQ

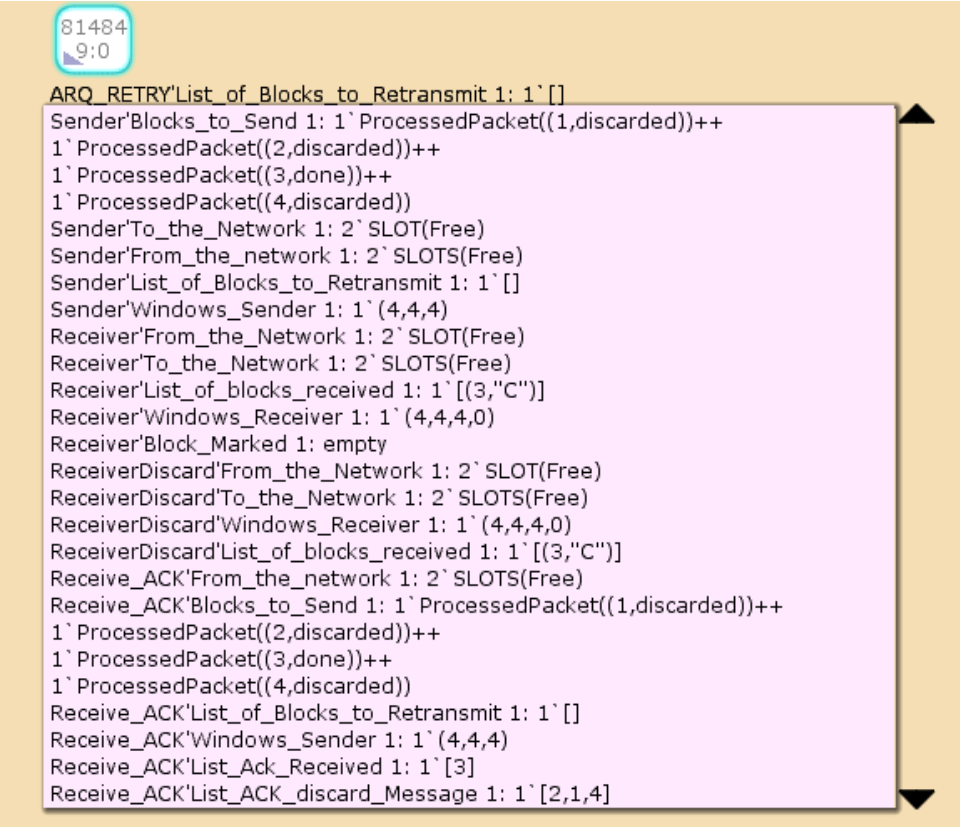

Figura 6.6: Marcados muertos Nodo 81484

El marcado muerto 66111 se muestra en la Tabla 16 y en la Figura 6.7 se puede visualizar el dibujo del nodo referente a este marcado en la herramienta CPN Tools. Para este marcado muerto la plaza *Blocks_to_Send* finaliza con las marcas 1'ProcessedPacket(1,discarded),1'ProcessedPacket(2,discarded),1'ProcessedPacket(3,don e), 1'ProcessedPacket (4,done), lo cual indica que los bloques uno y dos fueron descartados y los bloques tres y cuatro fueron recibidos correctamente por el receptor. La ventana deslizante del emisor representada por la plaza *Windows Sender* finaliza con el valor de 1'(4,4,4), lo cual indica que todos los bloques fueron enviados. La plaza *List_of_blocks_received* finaliza con la marca 1' [(3,"C"), (4,"D")]. La plaza

Windows_Receiver finaliza con la marca 1'(3,4,2,0), la plaza *List_ACk_received* finaliza con la marca 1'[3,4] ya que el emisor recibió el ACK correspondiente a los bloques tres y cuatro, y la plaza *List_ACK_discard_Message* finaliza con la marca 1'[2,1], indicando que los bloques ARQ uno y dos fueron descartados.

Plaza	Marcado
Blocks_to_Send	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)
Windows_Sender	1'(4,4,4)
List_of_blocks_received	1'[(4,"D"), (3,"C")]
Windows_Receiver	1'(3,4,2,0)
List_ACk_received	1'[3,4]
List_ACK_discard_Message	1'[2,1]

Tabla 16: Marcados Muertos 66111 para la simulación con 4 bloques ARQ

Figura 6.7: Marcados muertos Nodo 66111

El marcado muerto 45373 se muestra en la Tabla 17 y en la Figura 6.8 se puede visualizar el dibujo del nodo referente a este marcado en la herramienta CPN Tools. Para este marcado muerto la plaza *Blocks_to_Send* finaliza con las marcas 1'ProcessedPacket(1,discarded), 1'ProcessedPacket(2,done), 1'ProcessedPacket(3,done), 1'ProcessedPacket (4,done), lo cual indica que el bloque uno fue descartado y los bloques dos, tres y cuatro fueron recibidos correctamente por el receptor. La ventana deslizante del emisor representada por la plaza *Windows Sender* finaliza con el valor de 1'(4,4,4), lo cual indica que todos los bloques fueron enviados. La plaza *List_of_blocks_received* finaliza con la marca 1' [(2,"B"), (4,"D"), (3,"C")]. La plaza *Windows_Receiver* finaliza con la

marca 1’(3,4,1,0), la plaza *List_ACk_received* finaliza con la marca 1’[2,4,3] ya que el emisor recibió el ACK correspondiente a los bloques dos, tres y cuatro, y la plaza *List_ACK_discard_Message* finaliza con la marca 1’[1] , indicando que el bloque uno fue descartado.

Plaza	Marcado
Blocks_to_Send	1’ProcessedPacket(1,discarded), 1’ProcessedPacket (2,done) 1’ProcessedPacket (3,done) 1’ProcessedPacket (4,done)
Windows_Sender	1’(4,4,4)
List_of_blocks_received	1’[(2,”B”), (4,”D”), (3,”C”)]
Windows_Receiver	1’(3,4,0,0)
List_ACk_received	1’[2,4,3]
List_ACK_discard_Message	1’[1]

Tabla 17: Marcados Muertos 45373 para la simulación con 4 bloques ARQ

Figura 6.8: Marcados muertos Nodo 45373

Los marcados muertos que resultan del análisis del grafo de estado son esperados, debido a que las plazas anteriormente mencionadas fueron diseñadas para mantener entidades que llevan el control y gestión del proceso de envió y retransmisión de bloques ARQ, por lo cual, la plaza *Blocks_to_Send* al final de la simulación contiene las marcas que representan los bloques enviados en sus estados terminales, es decir, el BSN del bloque seguido de su estado, que puede haber sido recibido o descartado. Las plazas *Windows_Sender* y *Windows_Receiver* al final de la simulación contienen los valores terminales de la ventana deslizante; en el caso de la ventana deslizante del emisor, los apuntadores a inicio y fin, así como un apuntador al siguiente bloque a enviar. En el caso

de la ventana deslizante del receptor añadido a los apuntadores de inicio y fin, se tienen dos indicadores: último bloque descartado y último bloque marcado. Asimismo, las plazas *List_of_blocks_received*, *List_ACk_received*, *List_ACK_discard_Message* al final de la simulación contienen los bloques ARQ y ACKs enviados y recibidos durante la simulación, esto permite validar los resultados de la transmisión de bloques.

6.6.2.2 Marcado Local

Un Marcado Local, es un marcado que puede ser siempre alcanzado por el resto de los marcados alcanzables. En el modelo no se tienen marcados locales, ya que existen marcados que representan estados finales, a partir de los cuales no es posible volver al marcado inicial.

6.6.2.3 Transiciones Muertas

Una transición muerta es aquella que no está habilitada en ningún marcado alcanzable [2]. En el modelo presentado, existen transiciones muertas para algunos valores iniciales lo cual se muestra en la Tabla 18. Cuando en la simulación se lleva a cabo con más de una retransmisión por bloque, adicional al envío de más de un mensaje de descarte por bloque, se genera una explosión del grafo de estado que impide que este pueda ser generado de forma total, por lo cual esto limita el número de retransmisiones y mensajes de descartes que pueden ser enviados desde el emisor, a uno o dos por bloque. Se realizaron simulaciones en las que se envía un mensaje de descarte y se realizan dos retransmisiones y viceversa. Para el escenario en el que se realiza el envío de un mensaje de descarte correspondiente a las condiciones iniciales Nro. 1,3,4,6,7,8,9,12,13,15,18 las transiciones *ARQ_RETRY_TIMEOUT_for retransmit discard message* que representa el temporizador *ARQRETRY_TIMEOUT* que debe vencerse para realizar la retransmisión del mensaje de descarte y la transición *Retransmit_Discard_Message* que es la encargada de

colocar el mensaje de descarte en la red, no pueden ocurrir, por lo cual son transiciones muertas.

Nro. valor inicial	Transiciones Muertas
1	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
2	ARQ_PURGE_TIMEOUT
3	ARQ_RETRY'ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
4	ARQ_RETRY'ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message discard_invalide_discarded_blocks
5	No aplica
6	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message discard_invalide_discarded_blocks
7	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
8	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
9	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
10	ARQ_PURGE_TIMEOUT
11	ARQ_PURGE_TIMEOUT
12	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message
13	RQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message
14	No aplica
15	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message
16	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks
17	ARQ_PURGE_TIMEOUT
18	ARQ_RETRY_TIMEOUT_for_retransmit_discard_message Retransmit_Discard_Message ARQ_PURGE_TIMEOUT discard_invalide_discarded_blocks

Tabla 18: Transiciones Muertas para Valores Iniciales

Para los valores iniciales en las que se realizó la simulación con tamaño de ventana deslizante igual a uno, la transición *ARQ_PURGE_TIMEOUT* no podrá ocurrir dado que esta transición es la encargada de avanzar la ventana una vez que ha llegado un bloque fuera de orden, dentro del rango de la ventana deslizante. Sin embargo, si la ventana deslizante es de tamaño uno, obliga que los bloques tengan que ser recibidos en orden y esta transición no puede ocurrir, resultando como transición muerta para los valores iniciales nro.1,2,3,7,8,9,10,11,16,17,18.

Para los valores iniciales en las que se realizó la simulación con número de envío mensajes de descarte igual a uno, la transición *discard_invalide_discarded_blocks* encargada de eliminar los mensajes de descarte recibidos que estén fuera del rango de la ventana deslizante, cuyo BSN del bloque indicado en el mensaje de descarte no haya sido recibido, no podrá ocurrir. Esta situación obedece a la ocurrencia de los eventos que se explican a continuación (para hacer referencia al identificador del bloque ARQ o BSN se utilizará la letra *x*):

1) Se ha vencido el tiempo de vida para el bloque *x* en el emisor, por lo que se ha enviado un mensaje de descarte al receptor.

2) El receptor ha recibido el mensaje de descarte para el bloque *x*, como el bloque *x* no ha llegado con anterioridad el receptor avanzará la ventana deslizante cuyo punto de inicio será *x+1*, y envía un ACK para confirmar el mensaje de descarte para el bloque *x*.

3) El emisor no ha recibido el ACK para el mensaje de descarte aún y se vence el temporizador *ARQ_RETRY_TIMEOUT* y adicionalmente la simulación está

habilitada para realizar más de una transmisión del mensaje de descarte, por lo cual se retransmite el mensaje de descarte para el bloque *x*.

4) El receptor recibe el mensaje de descarte comprueba con la ocurrencia de la transición *discard_invalide_discarded_blocks* que el BSN indicado en el mismo no está en el rango de su ventana deslizante, pues ésta fue avanzada como resultado de la recepción del primer mensaje de descarte enviado por el emisor. Comprueba además, que el bloque indicado en el mensaje de descarte no haya sido recibido y en caso afirmativo, elimina el mensaje de descarte.

Las situaciones o eventos explicados en los puntos 1,2,3,4 no pueden suceder si no está habilitado el envío de más de un mensaje de descarte, por lo cual la transición *discard_invalide_discarded_blocks* es transición muerta para los valores iniciales Nro. 1,3,4,6,7,8,9,12,13,15,16,18.

El modelo culminará la ejecución de la simulación una vez que todos los bloques ARQ hayan sido transmitidos desde el emisor al receptor, y hayan sido recibidos de forma correcta, o hayan sido descartados. Los estados terminales o marcados muertos, de un bloque ARQ pueden ser *Done* cuando el bloque fue recibido por el receptor correctamente, *Discarded* cuando el bloque fue eliminado por el emisor, por no haberse recibido el ACK en el tiempo reglamentario. En este caso, se continúa con el proceso de envío del siguiente bloque al receptor, y el bloque anterior queda ignorado, por lo que no se esperará recibir reconocimiento para este.

6.7 Decisiones de Modelado

Se tomaron varias decisiones sobre cómo modelar las características del mecanismo de retransmisión de bloques ARQ y se resumen a continuación.

1) Los posibles estados por los que pueden pasar los bloques, una vez que se inicie el proceso de transmisión por parte de la máquina de estados del emisor son: *Nosent, Outstanding, Waiting, WaitingSendMD, SendedMD, Discarded, Done*. La especificación precisa que el estado *Discarded* se utiliza para indicar que el bloque fue aceptado o descartado indistintamente. Sin embargo, para hacer diferencia entre estas dos situaciones se utilizó un estado distinto para cada una, el estado *Discarded* en el modelado representa únicamente que el bloque fue descartado. Asimismo, se añadió el estado *Done* para representar que el bloque fue recibido correctamente. Los estados *WaitingSendMD* y *SendedMD* fueron añadidos para modelar transmisión y retransmisión de mensajes de descarte. Por su parte, en el receptor el procedimiento es modelado por un Workflow (flujo de trabajo), para el cual no se manejan estados, ya que el proceso termina para un bloque una vez que este es recibido y validado como correcto.

2) En la especificación de la máquina de estados del emisor no se hace diferencia entre el término del temporizador *ARQ_RETRY_TIMEOUT* y la recepción de un *NACK*, es decir, las acciones a tomar son las mismas para estos dos eventos, por lo cual en el modelado estas acciones son representadas por una misma transición.

3) Para simplificar el modelado se utilizó únicamente ACK selectivos, uno por cada bloque.

4) El estándar no precisa cuantos mensajes de descarte deberán ser enviados al receptor como límite, por lo que, por simplicidad del modelo se realiza el envío máximo de dos mensajes de descarte. Si una vez enviado estos dos mensajes

de descarte, no se recibe reconocimiento de los mismos, las ventanas deslizantes no podrán avanzar, y se asume que el temporizador *ARQ_SYNC_LOSS_TIMEOUT* expirará, por lo que se declara una pérdida de la sincronización y deberá establecerse la conexión nuevamente. El procedimiento de reseteo y restablecimiento de la conexión no se encuentra en el alcance de este trabajo.

5) Adicional al procedimiento de descartar bloques duplicados por parte de la entidad receptora descrito en el estándar, se realiza la retransmisión del ACK de reconocimiento del bloque a la entidad emisora, una vez recibido un bloque duplicado. De esta manera, se evita que la entidad emisora continúe retransmitiendo el bloque.

6) En cuanto al envío y recepción de mensajes de descarte, una vez recibido el mensaje de descarte, se comprueba si el bloque indicado en el mensaje de descarte ha sido recibido, en caso afirmativo, se elimina el mensaje de descarte. En caso contrario, se realiza el movimiento de la ventana deslizante al BSN del próximo bloque que se espera recibir y se envía un ACK de descarte.

CAPÍTULO 7: CONCLUSIONES

El popular crecimiento de las aplicaciones multimedia ha hecho del acceso de banda ancha un requerimiento cada vez más necesario. El acceso de banda ancha se ha convertido en un crítico e importante cuello de botella en la revolución de la tecnología de la información actual a nivel mundial. En éste aspecto surge el estándar IEEE 802.16, como una iniciativa de la IEEE a fin de proveer una solución de banda ancha inalámbrica que fuese interoperable con cualquier fabricante, buscando bajar los costos, y ser una alternativa a la red cableada en áreas urbanas, mostrando un alto rendimiento. Sin embargo, El IEEE 802.16 describe el funcionamiento del mecanismo de retransmisión de bloques ARQ de una manera narrativa e imprecisa. Pocos estudios formales se han realizado en cuanto a la operación ARQ, y probablemente las implementaciones y las pruebas son los únicos mecanismos usados hasta ahora para validar su funcionalidad. Por lo tanto, en este trabajo, se analizó el mecanismo de retransmisión de bloques ARQ como se especifica en el estándar IEEE802.16. Se determinaron las características de la operación ARQ para la entidad emisora y receptora y se verificó el correcto funcionamiento de este procedimiento mediante la realización de un modelo, utilizando las Redes de Petri Coloreadas con la herramienta CPN Tools.

La verificación del modelo del mecanismo de retransmisión de bloques ARQ, se basó en una metodología estándar propuesta por Billintong que incluye el análisis de la operación ARQ en función del conjunto de propiedades de las Redes de Petri Coloreadas (CPNs). Asimismo, se analizó la operación ARQ utilizando las técnicas del gráfico de estado para la verificación automática.

Se realizaron diez y ocho simulaciones diferentes para los valores iniciales que se muestran en la Tabla 2 del Capítulo 6, que corresponden al envío de dos, tres ó cuatro bloques, con ventanas deslizantes de tamaño uno ó dos, realizando una ó dos retransmisiones y el envío de uno ó dos mensajes de descarte por bloque. Para ninguna de estas simulaciones de evidenció abrazos mortales. La simulación que se tomó como caso de estudio, corresponde al valor inicial Nro. 15, con cuatro bloques a enviar, un tamaño de ventana deslizante de dos, realizando una retransmisión y el envío de un mensaje de descarte. Para este escenario la realización de la simulación tomó un tiempo de 3196 segundos, es decir, 53.26 minutos.

Se analizaron las propiedades generales de las CPNs en relación al modelo, entre ellas las cotas enteras y las cotas de los multi-conjuntos. Asimismo fueron analizados los marcados muertos resultantes, y determinadas las características que deben tener estos marcados para ser considerados esperados y correctos en relación al diseño del modelo. Para el valor inicial Nro.15 se obtuvieron 158 marcados muertos que son precisados en detalle en la sección 6.6.2.1.1. De igual manera se determinó la existencia de transiciones muertas para ciertos valores iniciales los cuales son precisados en detalle en la sección 6.6.2.3. Para el modelo realizado no existen marcados locales.

Luego de haber modelado y analizado los procesos involucrados en el mecanismo de retransmisión de bloques ARQ de la capa MAC IEEE 802.16, se ha podido corroborar el correcto funcionamiento de la operación ARQ en cuanto al envío y retransmisión de bloques, no se encontraron problemas funcionales significativos, por lo que el mecanismo se comporta según lo esperado, bajo las asunciones del modelo y las limitaciones del análisis.

7.1 Aspectos relevantes del modelo

- En el estudio de la máquina de estados del emisor de la operación ARQ según las especificaciones del IEEE 802.16, se encontró una ambigüedad referente a los estados en los que se encuentran los bloques ARQ, ya que no se hace distinción de estado para la situación en que el bloque haya sido recibido correctamente, o haya sido descartado. Por el contrario, se utiliza el estado *discarded* para ambos casos. Sin embargo en el diagrama de flujo de estados se muestra el estado *done*, el cual no está incluido en la especificación del funcionamiento de las máquinas de estados del emisor y receptor como lo describe el estándar.

- El IEEE 802.16 establece que la máquina de estados del emisor deberá enviar un mensaje de descarte siguiendo la violación del temporizador *ARQ_BLOCK_LIFETIME*. Este mensaje de descarte debe ser enviado al receptor de forma consecutiva cada vez que expire el temporizador ARQ_RETRY_TIMEOUT, hasta que un reconocimiento (ACK de descarte) haya sido recibido. Sin embargo, cuando se hace referencia en el estándar al proceso de descarte de bloques desde el punto de vista del receptor no señala que deba enviarse algún reconocimiento al recibir el mensaje de descarte. Tampoco se especifica ningún temporizador adicional, cuyo vencimiento permita avanzar la ventana deslizante en el emisor para continuar con la transmisión en caso de que el ACK de descarte no sea recibido. Asimismo, tampoco se especifica que acciones debe tomar el receptor si recibe un mensaje de descarte para un bloque que ha recibido correctamente.

7.2 Contribuciones

7.2.1 Descripción detallada de los procedimientos involucrados en el mecanismo de retransmisión de bloques ARQ

En éste trabajo han sido presentados de manera formal y detallada, los procesos realizados por la máquina de estados del emisor y el receptor que constituyen la operación ARQ de la subcapa de parte común (CPS) de la capa MAC IEEE 802.16 utilizando las CPNs.

7.2.2 Desarrollo de la especificación formal de los procesos involucrados en el mecanismo de retransmisión de bloques ARQ

En éste trabajo, los procesos realizados por las máquinas de estados del emisor y el receptor han sido descritos, presentados y estructurados de forma precisa en el capítulo 2. Esta descripción de los procesos, sirvió como base para el desarrollo del modelo del mecanismo de retransmisión de bloques ARQ, permitiendo identificar las funciones realizadas en cada entidad, tanto emisor como receptor y los estados por los cuales pasan los bloques ARQ en el proceso de transmisión. Asimismo, se desarrolló una especificación de los procesos involucrados en el mecanismo de retransmisión de bloques ARQ, en cuanto a vacíos de información existentes en el estándar IEEE 802.16 para la realización del modelado.

7.2.3 Análisis formal del modelo del mecanismo de retransmisión de bloques ARQ

El modelo del mecanismo de retransmisión de bloques ARQ se analiza para determinar si satisface las propiedades de las CPNs. Como resultado de este análisis se determinó que el funcionamiento de la operación ARQ se comporta según lo esperado, bajo las asunciones del modelo y las limitaciones del análisis.

7.3 Limitaciones

Cuando se realiza la simulación para el envío de más de cuatro bloques, se genera un número muy grande de marcados, el cual no se puede determinar, que impide que análisis del gráfico de estado pueda generarse de forma total, esto se conoce como explosión de estados. Por lo tanto, las pruebas realizadas toman como máximo cuatro bloques a enviar, para cuyo caso se realiza una retransmisión y el envío de un mensaje de descarte.

7.4 Trabajos Futuros

El modelo del mecanismo de retransmisión de bloques ARQ, ha sido ampliamente analizado y verificado. Se propone como trabajo futuro el modelado del uso de ACKs acumulativos y la mezcla de ACK acumulativo y selectivos, así como también el mecanismo de establecimiento y reset de la conexión con ARQ habilitado.

REFERENCIAS BIBLIOGRÁFICAS

[1] JAMES L. PETERSON. “Petri Net Theory and the modeling of systems”. 1980.

[2] KURT JENSEN & LARS M. KRISTENSEN. “Colured Petri Nets”. London. 2009.

[3] ERIKA ORDÓÑEZ BRAVO. “Diseño de una red inalámbrica utilizando la tecnología WiMAX para proveer el servicio de internet de banda ancha en la ciudad de Manta.” GUAYAQUIL – ECUADOR.2008.

[4] ANA V. MORALES B. “Fundamentos Básicos de la capa MAC de IEEE 802.16”.Caracas.2009.

[5] ANA V. MORALES B. “Modelado y análisis de los procesos involucrados en la gestión de las conexiones en la capa MAC IEEE 802.16 utilizando Redes de Petri coloreadas CPNs”. Caracas. Abril 2010.

[6] MARIA ELENA VILLAPOL BLANCO. ”Modelado y Análisis del Protocolo de Reservación de Recursos Usando las Redes de Petri Coloreadas” .Caracas. Junio 2003.

[7] IEEE Sta. 802.16-2009. “Local and Metropolitan Area Network, Part 16: Air Interface for Fixed and mobile Broadband Wireless Access Systems”. Mayo 2009.

[8] IEEE Sta. 802.16-2009j. “Local and Metropolitan Area Network, Part 16: Air Interface for Fixed and mobile Broadband Wireless Access Systems”. Amendment 1: Multihop Relay Specification. Mayo 2009.

[9] HOLZMANN G. “Desing and validation of computer protocols”. Prentice Hall. 1991.

[10] Proceedings of Joint International Conference on Formal Description Techniques for distributed Systems and Communications Protocols (FORTE) and Protocol Specification, Testing & Verification (PSTV), 1997-2000.

[11] IEEE Sta. 802.16-2001. “Local and Metropolitan Area Network, Part 16: Air Interface for Fixed Broadband Wireless Access Systems”. Octubre 2002.

[12] IEEE Sta. 802.16-2004. “Local and Metropolitan Area Network, Part 16: Air Interface for Fixed Broadband Wireless Access Systems”. Octubre 2004.

[13] KURT JENSEN, AARHUS UNIVERSITY,DENMARK. “Distributed Data Base”.
Documentación CPN TOOLS. 2010

[14] D.A. ZAITSEV, T.R. SHMELEVA. “Simulating of Telecommunication Systems with CPN Tools”. Odessa 2006.

[15] W.BRAUER G.ROZENBERG A.SALOMAA(Eds).”Nonsequential Processes”.

[16] JAVIER EMILIO SIERRA, ROBERTO HINCAPIÉ, ROBERTO BUSTAMANTE y LEONARDO BETANCUR.”Modelo de simulación de la capa MAC IEEE 802.16-2004 para modo Mesh”. 2006.

[17] CARLOS REY MORENO. “Análisis de la viabilidad de la modificación de la enmienda IEEE 802.16j para su aplicación en la banda no licenciada de 5 GHz”. 2009-2010

[18] JUAN PABLO CABALLERO VILLALOBOS. “Redes de Petri y algoritmos genéticos, una propuesta para la programación de sistemas de manufactura flexible”. Bogotá, Colombia. 2006.

[19] CARL EKLUND. ROGER B. MARKS, KENNETH L. STANWOOD and STANLEY WANG. “IEEE Standard 802.16: A Technical Overview of the WirelessMAN™ Air Interface for Broadband Wireless Access”. 2002.

[20] José Luis Villarroel Salcedo. ”Redes de Petri para tiempo real”. 2011 http://webdiis.unizar.es/~joseluis/TPN&RT.pdf

[21] CPN TOOLS PG official. 2013. http://www.cpntools.org
[22] Departamento de electrónica y computadores. Universidad de Cantabria. “Redes de Petri: Modelado de sistemas concurrentes”. 2012-201.
http: //www.ctr.unican.es/asignaturas/MC_ProCon/Doc/PETRI_3.pdf.

[24] “Providing Complete WiMAX solutions”. 2013.
http://www.freewimaxinfo.com/wimax-services.html

[25] Samuel Garrido Daniel.” Modelado de Workflow con redes de Petri Coloreadas Condicionales”.2013.
http://www.cs.cinvestav.mx/TesisGraduados/2005/tesisSamuelGarrido.pdf

[26] “Wireless BroadBand Network”. 2013.
http://www.infowimax.blogspot.com/2008/04/arquitectura-wimax.html.

[27] “Topics in broadband access”.
http://www.ieee802.org/16/docs/02/C80216-02_05.pdf. Octubre 2011

[28] Billington J, Gallasch G. and Bing H. *A Coloured Petri Net Approach to protocol Verification.* Lectures on Concurrency and Petri Nets: advances in Petri nets 20041973, vol. 3098, pp. 210-290.

Apéndice A

Resultados del Análisis del Modelo de Retransmisión de Bloques ARQ

A.1 Modelo Operación ARQ con dos bloques, una retransmisión, un descarte, tamaño de ventana 2

Statistics

State Space

Nodes: 2155

Arcs: 6069

Secs: 1

Status: Full

Scc Graph

Nodes: 2155

Arcs: 6069

Secs: 0

Boundedness Properties

Best Integer Bounds

Upper Lower

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

1 0

Prepare_to_Retransmit'block_to_retransmit 1

1 0

Receive_ACK'List_ACK_discard_Message 1

1 1

Receive_ACK'List_Ack_Received 1

1 1

Receiver'Block_Marked 1 1 0

Receiver'List_of_blocks_received 1

1 1

Receiver'Windows_Receiver 1

1 1

Sender'Blocks_to_Send 1 2 2

Sender'List_of_Blocks_to_Retransmit 1

1 1

Sender'Windows_Sender 1 1 1

Top'From_the_network 1 2 2

Top'To_the_network 1 2 2

Transmit'Limit_Next_BSN 1

1 0

Transmit'Next_BSN 1 1 0

Best Upper Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

1`()

Prepare_to_Retransmit'block_to_retransmit 1

1`(1,"A")++

1`(2,"B")

Receive_ACK'List_ACK_discard_Message 1

1`[]++

1`[1]++

1`[1,2]++

1`[2]++

1`[2,1]

Receive_ACK'List_Ack_Received 1

1`[]++

1`[1]++

1`[1,2]++

1`[2]++

1`[2,1]

Receiver'Block_Marked 1

1`2

Receiver'List_of_blocks_received 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]

Receiver'Windows_Receiver 1

1`(1,2,0,0)++

1`(1,2,0,2)++

1`(1,2,1,2)++

1`(2,2,0,0)++

1`(2,2,1,0)++

1`(2,2,2,0)

Sender'Blocks_to_Send 1

1`SDPacket((1,SendedMD,1))++

1`SDPacket((2,SendedMD,1))++

1`SRPacket((1,Nosent,0,"A"))++

1`SRPacket((1,outstanding,0,"A"))++

1`SRPacket((1,outstanding,1,"A"))++

1`SRPacket((1,waiting,0,"A"))++

1`SRPacket((2,Nosent,0,"B"))++

1`SRPacket((2,outstanding,0,"B"))++

1`SRPacket((2,outstanding,1,"B"))++

1`SRPacket((2,waiting,0,"B"))++

1`ProcessedPacket((1,discarded))++

1`ProcessedPacket((1,done))++

1`ProcessedPacket((2,discarded))++

1`ProcessedPacket((2,done))

Sender'List_of_Blocks_to_Retransmit 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]

Sender'Windows_Sender 1

1`(1,2,0)++

1`(1,2,1)++

1`(1,2,2)++

1`(2,2,0)++

1`(2,2,1)++

1`(2,2,2)

Top'From_the_network 1

2`AcksBlocks(1)++

2`AcksBlocks(2)++

1`ACKMessageDiscard((1,Discarded))++

1`ACKMessageDiscard((2,Discarded))++

2`SLOTS(Free)

Top'To_the_network 1

2`PACKET((1,"A"))++

2`PACKET((2,"B"))++

1`PACKETDISCARD((34,1))++

1`PACKETDISCARD((34,2))++

2`SLOT(Free)

Transmit'Limit_Next_BSN 1

1`()

Transmit'Next_BSN 1 1`1++

1`2

Best Lower Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

empty

Prepare_to_Retransmit'block_to_retransmit 1

empty

Receive_ACK'List_ACK_discard_Message 1

empty

Receive_ACK'List_Ack_Received 1

empty

Receiver'Block_Marked 1

empty

Receiver'List_of_blocks_received 1

empty

Receiver'Windows_Receiver 1

empty

Sender'Blocks_to_Send 1

empty

Sender'List_of_Blocks_to_Retransmit 1

empty

Sender'Windows_Sender 1

empty

Top'From_the_network 1

empty

Top'To_the_network 1

empty

Transmit'Limit_Next_BSN 1

empty

Transmit'Next_BSN 1 empty

Home Properties

Home Markings

None

Liveness Properties

Dead Markings

13 [888,818,637,629,414,...]

Dead Transition Instances

ARQ_RETRY'ARQ_RETRY_TIMEOUT_for_retransmit_discard_message 1

Block_Lifetime'Retransmit_Discard_Message 1

Live Transition Instances

None

Fairness Properties

No infinite occurrence sequences.

A.2 Modelo Operación ARQ con tres bloques, una retransmisión, un descarte, tamaño de ventana 2

Statistics

State Space

Nodes: 18135

Arcs: 56573

Secs: 32

Status: Full

Scc Graph

Nodes: 18135

Arcs: 56573

Secs: 1

Boundedness Properties

Best Integer Bounds

Upper Lower

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

1 0

Prepare_to_Retransmit'block_to_retransmit 1

1 0

Receive_ACK'List_ACK_discard_Message 1

1 1

Receive_ACK'List_Ack_Received 1

1 1

Receiver'Block_Marked 1 2 0

Receiver'List_of_blocks_received 1

1 1

Receiver'Windows_Receiver 1

1 1

Sender'Blocks_to_Send 1 3 3

Sender'List_of_Blocks_to_Retransmit 1

1 1

Sender'Windows_Sender 1 1 1

Top'From_the_network 1 2 2

Top'To_the_network 1 2 2

Transmit'Limit_Next_BSN 1

1 0

Transmit'Next_BSN 1 1 0

Best Upper Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

1`()

Prepare_to_Retransmit'block_to_retransmit 1

1`(1,"A")++

1`(2,"B")++

1`(3,"C")

Receive_ACK'List_ACK_discard_Message 1

1`[]++

1`[1]++

1`[1,2]++

1`[1,2,3]++

1`[1,3]++

1`[1,3,2]++

1`[2]++

1`[2,1]++

1`[2,1,3]++

1`[2,3]++

1`[3]++

1`[3,2]

Receive_ACK'List_Ack_Received 1

1`[]++

1`[1]++

1`[1,2]++

1`[1,2,3]++

1`[1,3]++

1`[1,3,2]++

1`[2]++

1`[2,1]++

1`[2,1,3]++

1`[2,3]++

1`[3]++

1`[3,2]

Receiver'Block_Marked 1

1`2++

1`3

Receiver'List_of_blocks_received 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(1,"A"),(2,"B"),(3,"C")]++

1`[(1,"A"),(3,"C")]++

1`[(1,"A"),(3,"C"),(2,"B")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]++

1`[(2,"B"),(1,"A"),(3,"C")]++

1`[(2,"B"),(3,"C")]++

1`[(3,"C")]++

1`[(3,"C"),(2,"B")]

Receiver'Windows_Receiver 1

1`(1,2,0,0)++

1`(1,2,0,2)++

1`(2,3,0,0)++

1`(2,3,0,2)++

1`(2,3,0,3)++

1`(2,3,1,0)++

1`(2,3,1,2)++

1`(2,3,1,3)++

1`(2,3,2,3)++

1`(2,3,3,0)++

1`(2,3,3,2)++

1`(3,3,0,0)++

1`(3,3,1,0)++

1`(3,3,2,0)++

1`(3,3,3,0)

Sender'Blocks_to_Send 1

1`SDPacket((1,SendedMD,1))++

1`SDPacket((2,SendedMD,1))++

1`SDPacket((3,SendedMD,1))++

1`SRPacket((1,Nosent,0,"A"))++

1`SRPacket((1,outstanding,0,"A"))++

1`SRPacket((1,outstanding,1,"A"))++

1`SRPacket((1,waiting,0,"A"))++

1`SRPacket((2,Nosent,0,"B"))++

1`SRPacket((2,outstanding,0,"B"))++

1`SRPacket((2,outstanding,1,"B"))++

1`SRPacket((2,waiting,0,"B"))++

1`SRPacket((3,Nosent,0,"C"))++

1`SRPacket((3,outstanding,0,"C"))++

1`SRPacket((3,outstanding,1,"C"))++

1`SRPacket((3,waiting,0,"C"))++

1`ProcessedPacket((1,discarded))++

1`ProcessedPacket((1,done))++

1`ProcessedPacket((2,discarded))++

1`ProcessedPacket((2,done))++

1`ProcessedPacket((3,discarded))++

1`ProcessedPacket((3,done))

Sender'List_of_Blocks_to_Retransmit 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]++

1`[(2,"B"),(3,"C")]++

1`[(3,"C")]++

1`[(3,"C"),(2,"B")]

Sender'Windows_Sender 1

1`(1,2,0)++

1`(1,2,1)++

1`(1,2,2)++

1`(2,3,0)++

1`(2,3,1)++

1`(2,3,2)++

1`(2,3,3)++

1`(3,3,0)++

1`(3,3,2)++

1`(3,3,3)

Top'From_the_network 1

2`AcksBlocks(1)++

2`AcksBlocks(2)++

2`AcksBlocks(3)++

1`ACKMessageDiscard((1,Discarded))++

1`ACKMessageDiscard((2,Discarded))++

1`ACKMessageDiscard((3,Discarded))++

2`SLOTS(Free)

Top'To_the_network 1

2`PACKET((1,"A"))++

2`PACKET((2,"B"))++

2`PACKET((3,"C"))++

1`PACKETDISCARD((34,1))++

1`PACKETDISCARD((34,2))++

1`PACKETDISCARD((34,3))++

2`SLOT(Free)

Transmit'Limit_Next_BSN 1

1`()

Transmit'Next_BSN 1 1`1++

1`2++

1`3

Best Lower Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

empty

Prepare_to_Retransmit'block_to_retransmit 1

empty

Receive_ACK'List_ACK_discard_Message 1

empty

Receive_ACK'List_Ack_Received 1

empty

Receiver'Block_Marked 1

empty

Receiver'List_of_blocks_received 1

empty

Receiver'Windows_Receiver 1

empty

Sender'Blocks_to_Send 1

empty

Sender'List_of_Blocks_to_Retransmit 1

empty

Sender'Windows_Sender 1

empty

Top'From_the_network 1

empty

Top'To_the_network 1

empty

Transmit'Limit_Next_BSN 1

empty

Transmit'Next_BSN 1 empty

Home Properties

Home Markings

None

Liveness Properties

Dead Markings

41 [9893,8382,8327,8319,8104,...]

Dead Transition Instances

ARQ_RETRY'ARQ_RETRY_TIMEOUT_for_retransmit_discard_message 1

Block_Lifetime'Retransmit_Discard_Message 1

Live Transition Instances

None

Fairness Properties

No infinite occurrence sequences.

A.3 Modelo Operación ARQ con cuatro bloques, una retransmisión, un descarte, tamaño de ventana 2

Statistics

State Space

Nodes: 134545

Arcs: 450147

Secs: 3196

Status: Full

Scc Graph

Nodes: 134545

Arcs: 450147

Secs: 10

Boundedness Properties

Best Integer Bounds

	Upper	Lower
Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1	1	0
Prepare_to_Retransmit'block_to_retransmit 1	1	0

Receive_ACK'List_ACK_discard_Message 1

1 1

Receive_ACK'List_Ack_Received 1

1 1

Receiver'Block_Marked 1 2 0

Receiver'List_of_blocks_received 1

1 1

Receiver'Windows_Receiver 1

1 1

Sender'Blocks_to_Send 1 4 4

Sender'List_of_Blocks_to_Retransmit 1

1 1

Sender'Windows_Sender 1 1 1

Top'From_the_network 1 2 2

Top'To_the_network 1 2 2

Transmit'Limit_Next_BSN 1

1 0

Transmit'Next_BSN 1 1 0

Best Upper Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

1`()

Prepare_to_Retransmit'block_to_retransmit 1

1`(1,"A")++

1`(2,"B")++

1`(3,"C")++

1`(4,"D")

Receive_ACK'List_ACK_discard_Message 1

1`[]++

1`[1]++

1`[1,2]++

1`[1,2,3]++

1`[1,2,3,4]++

1`[1,2,4]++

1`[1,2,4,3]++

1`[1,3]++

1`[1,3,2]++

1`[1,3,2,4]++

1`[1,3,4]++

1`[1,4]++

1`[1,4,3]++

1`[2]++

1`[2,1]++

1`[2,1,3]++

1`[2,1,3,4]++

1`[2,1,4]++

1`[2,1,4,3]++

1`[2,3]++

1`[2,3,4]++

1`[2,4]++

1`[2,4,3]++

1`[3]++

1`[3,2]++

1`[3,2,4]++

1`[3,4]++

1`[4]++

1`[4,3]

Receive_ACK'List_Ack_Received 1

1`[]++

1`[1]++

1`[1,2]++

1`[1,2,3]++

1`[1,2,3,4]++

1`[1,2,4]++

1`[1,2,4,3]++

1`[1,3]++

1`[1,3,2]++

1`[1,3,2,4]++

1`[1,3,4]++

1`[1,4]++

1`[1,4,3]++

1`[2]++

1`[2,1]++

1`[2,1,3]++

1`[2,1,3,4]++

1`[2,1,4]++

1`[2,1,4,3]++

1`[2,3]++

1`[2,3,4]++

1`[2,4]++

1`[2,4,3]++

1`[3]++

1`[3,2]++

1`[3,2,4]++

1`[3,4]++

1`[4]++

1`[4,3]

Receiver'Block_Marked 1

1`2++

1`3++

1`4

Receiver'List_of_blocks_received 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(1,"A"),(2,"B"),(3,"C")]++

1`[(1,"A"),(2,"B"),(3,"C"),(4,"D")]++

1`[(1,"A"),(2,"B"),(4,"D")]++

1`[(1,"A"),(2,"B"),(4,"D"),(3,"C")]++

1`[(1,"A"),(3,"C")]++

1`[(1,"A"),(3,"C"),(2,"B")]++

1`[(1,"A"),(3,"C"),(2,"B"),(4,"D")]++

1`[(1,"A"),(3,"C"),(4,"D")]++

1`[(1,"A"),(4,"D")]++

1`[(1,"A"),(4,"D"),(3,"C")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]++

1`[(2,"B"),(1,"A"),(3,"C")]++

1`[(2,"B"),(1,"A"),(3,"C"),(4,"D")]++

1`[(2,"B"),(1,"A"),(4,"D")]++

1`[(2,"B"),(1,"A"),(4,"D"),(3,"C")]++

1`[(2,"B"),(3,"C")]++

1`[(2,"B"),(3,"C"),(4,"D")]++

1`[(2,"B"),(4,"D")]++

1`[(2,"B"),(4,"D"),(3,"C")]++

1`[(3,"C")]++

1`[(3,"C"),(2,"B")]++

1`[(3,"C"),(2,"B"),(4,"D")]++

1`[(3,"C"),(4,"D")]++

1`[(4,"D")]++

1`[(4,"D"),(3,"C")]

Receiver'Windows_Receiver 1

1`(1,2,0,0)++

1`(1,2,0,2)++

1`(2,3,0,0)++

1`(2,3,0,3)++

1`(2,3,1,0)++

1`(2,3,1,3)++

1`(3,4,0,0)++

1`(3,4,0,3)++

1`(3,4,0,4)++

1`(3,4,1,0)++

1`(3,4,1,2)++

1`(3,4,1,3)++

1`(3,4,1,4)++

1`(3,4,2,0)++

1`(3,4,2,3)++

1`(3,4,2,4)++

1`(3,4,3,4)++

1`(3,4,4,0)++

1`(3,4,4,3)++

1`(4,4,0,0)++

1`(4,4,1,0)++

1`(4,4,2,0)++

1`(4,4,3,0)++

1`(4,4,3,2)++

1`(4,4,4,0)++

1`(4,4,4,2)

Sender'Blocks_to_Send 1

1`SDPacket((1,SendedMD,1))++

1`SDPacket((2,SendedMD,1))++

1`SDPacket((3,SendedMD,1))++

1`SDPacket((4,SendedMD,1))++

1`SRPacket((1,Nosent,0,"A"))++

1`SRPacket((1,outstanding,0,"A"))++

1`SRPacket((1,outstanding,1,"A"))++

1`SRPacket((1,waiting,0,"A"))++

1`SRPacket((2,Nosent,0,"B"))++

1`SRPacket((2,outstanding,0,"B"))++

1`SRPacket((2,outstanding,1,"B"))++

1`SRPacket((2,waiting,0,"B"))++

1`SRPacket((3,Nosent,0,"C"))++

1`SRPacket((3,outstanding,0,"C"))++

1`SRPacket((3,outstanding,1,"C"))++

1`SRPacket((3,waiting,0,"C"))++

1`SRPacket((4,Nosent,0,"D"))++

1`SRPacket((4,outstanding,0,"D"))++

1`SRPacket((4,outstanding,1,"D"))++

1`SRPacket((4,waiting,0,"D"))++

1`ProcessedPacket((1,discarded))++

1`ProcessedPacket((1,done))++

1`ProcessedPacket((2,discarded))++

1`ProcessedPacket((2,done))++

1`ProcessedPacket((3,discarded))++

1`ProcessedPacket((3,done))++

1`ProcessedPacket((4,discarded))++

1`ProcessedPacket((4,done))

Sender'List_of_Blocks_to_Retransmit 1

1`[]++

1`[(1,"A")]++

1`[(1,"A"),(2,"B")]++

1`[(2,"B")]++

1`[(2,"B"),(1,"A")]++

1`[(2,"B"),(3,"C")]++

1`[(3,"C")]++

1`[(3,"C"),(2,"B")]++

1`[(3,"C"),(4,"D")]++

1`[(4,"D")]++

1`[(4,"D"),(3,"C")]

Sender'Windows_Sender 1

1`(1,2,0)++

1`(1,2,1)++

1`(1,2,2)++

1`(2,3,0)++

1`(2,3,1)++

1`(2,3,2)++

1`(2,3,3)++

1`(3,4,0)++

1`(3,4,2)++

1`(3,4,3)++

1`(3,4,4)++

1`(4,4,0)++

1`(4,4,3)++

1`(4,4,4)

Top'From_the_network 1

2`AcksBlocks(1)++

2`AcksBlocks(2)++

2`AcksBlocks(3)++

2`AcksBlocks(4)++

1`ACKMessageDiscard((1,Discarded))++

1`ACKMessageDiscard((2,Discarded))++

1`ACKMessageDiscard((3,Discarded))++

1`ACKMessageDiscard((4,Discarded))++

2`SLOTS(Free)

Top'To_the_network 1

2`PACKET((1,"A"))++

2`PACKET((2,"B"))++

2`PACKET((3,"C"))++

2`PACKET((4,"D"))++

1`PACKETDISCARD((34,1))++

1`PACKETDISCARD((34,2))++

1`PACKETDISCARD((34,3))++

1`PACKETDISCARD((34,4))++

2`SLOT(Free)

Transmit'Limit_Next_BSN 1

1`()

Transmit'Next_BSN 1 1`1++

1`2++

1`3++

1`4

Best Lower Multi-set Bounds

Prepare_to_Retransmit'Limit_Next_Block_to_Retransmit 1

empty

Prepare_to_Retransmit'block_to_retransmit 1

empty

Receive_ACK'List_ACK_discard_Message 1

empty

Receive_ACK'List_Ack_Received 1

empty

Receiver'Block_Marked 1

empty

Receiver'List_of_blocks_received 1

empty

Receiver'Windows_Receiver 1

empty

Sender'Blocks_to_Send 1

empty

Sender'List_of_Blocks_to_Retransmit 1

empty

Sender'Windows_Sender 1

empty

Top'From_the_network 1

empty

Top'To_the_network 1

empty

Transmit'Limit_Next_BSN 1

empty

Transmit'Next_BSN 1 empty

Home Properties

Home Markings

None

Liveness Properties

Dead Markings

158 [9988,9963,96931,96861,96368,...]

Dead Transition Instances

ARQ_RETRY'ARQ_RETRY_TIMEOUT_for_retransmit_discard_message 1

Block_Lifetime'Retransmit_Discard_Message 1

Live Transition Instances

None

Fairness Properties

No infinite occurrence sequences.

A.4 Funciones

```
(* ================= Functions ==================== *)

fun validate_range(t,wr,endrx)=
 if (t>=wr) andalso (t<=endrx)
then true
else false

fun AmemberBlock (x,L) =
if L = []
then false
else
if (compare(hd L, x))
then true
else AmemberBlock(x,List.tl L)

fun findMinor((n,d),L)=
if L = []
then (n,d)
else
findMinor(theLower((n,d),List.hd L),List.tl L)

fun Amember(x,L) =
if L = []
then false
else
if (x = List.hd L)
then true
else Amember(x,List.tl L);

fun FindNW(wr,t,W)=
if Amember(wr+1,W)
then FindNW (wr+1,t,W)
else wr+1

fun  valide_bsn(n,ws,endtx)=
if  n>=ws andalso n<=endtx
then true
else false

fun theLower((n,d),(x,y))=
if x<n
then (x,y)
else (n,d)

fun markedBlock(t,wr)=
if t> wr
then t
else 0

fun  novalide_bsn(n,ws,endtx)=
if  n<ws orelse n>endtx
then true
else false

fun compare((n,d),x)=
if (x=n)
then true
else false

fun FindNWBlock (wr,t,W)=
if AmemberBlock(wr+1,W)
then FindNWBlock (wr+1,t,W)
else wr+1
```

A.5 Marcados Muertos

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
9963	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(4,4,0,0)	1'[2,1,3,4]	1'[]	1'2
96930	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D")]	1'(3,4,3,4)	1'[4]	1'[2,1,3]	empty
96861	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(4,"D")]	1'(3,4,3,4)	1'[4]	1'[2,1,3]	empty
96367	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D")]	1'(3,4,3,4)	1'[4]	1'[1,2,3]	empty
96298	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(4,"D")]	1'(3,4,3,4)	1'[4]	1'[1,2,3]	empty
95630	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(3,"C")]	1'(3,4,4,3)	1'[3]	1'[1,2,4]	empty
8849	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,3,4]	1'[]	1'4

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
8842	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,4,3]	1'[]	1'4
8799	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[2,1,4,3]	1'[]	empty
8774	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[2,1,3,4]	1'[]	empty
81707	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	[(4,"D")]	1'(4,4,3,0)	1'[4]	1'[2,1,3]	empty
81699	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	[(4,"D")]	1'(4,4,3,0)	1'[4]	1'[2,1,3]	empty
81252	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4	[(4,"D")]	1'(4,4,3,0)	1'[4]	1'[1,3,2]	empty
81197	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4	[(4,"D")]	1'(4,4,3,0)	1'[4]	1'[1,2,3]	empty
81189	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4	[(4,"D")]	1'(4,4,3,0)	1'[4]	1'[1,2,3]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
80974	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	[(3,"C")]	1'(4,4,4,0)	1'[3]	1'[1,2,4]	empty
80515	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4	[(3,"C")]	1'(3,4,4,3)	1'[3]	1'[1,2,4]	1'3
8008	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (2,"B"), (4,"D")]	1'(3,4,0,4)	1'[1,2,4,3]	1'[]	1'3 1'4
7984	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (2,"B"), (4,"D")]	1'(3,4,0,4)	1'[1,2,3,4]	1'[]	1'3 1'4
7971	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (2,"B"), (4,"D")]	1'(3,4,0,4)	1'[1,3,2,4]	1'[]	1'3 1'4
79424	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	[(2,"B")]	1'(4,4,4,0)	1'[2]	1'[1,4,3]	empty
79411	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	[(2,"B")]	1'(4,4,4,0)	1'[2]	1'[1,3,4]	empty
7745	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[1,3,2,4]	1'[]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
7603	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,3,4]	1'[]	1'4
7596	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,4,3]	1'[]	1'4
75773	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(2,"B")]	1'(4,4,4,2)	1'[2]	1'[1,4,3]	1'2
75760	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B")]	1'(4,4,4,2)	1'[2]	1'[1,3,4]	1'2
7553	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4"D"), (3,"C")]	1'(4,4,0,0)	1'[1,2,4,3]	1'[]	empty
7528	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4"D"), (3,"C")]	1'(4,4,0,0)	1'[1,2,3,4]	1'[]	empty
68334	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A")]	1'(4,4,0,0)	1'[1]	1'[3,2,4]	empty
68290	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(1,"A")]	1'(4,4,0,0)	1'[1]	1'[2,4,3]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
68277	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A")]	1'(4,4,0,0)	1'[1]	1'[2,3,4]	empty
66090	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4"D"), (3,"C")]	1'(3,4,2,0)	1'[4,3]	1'[2,1]	empty
65684	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4"D"), (3,"C")]	1'(3,4,2,0)	1'[3,4]	1'[1,2]	empty
65663	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4"D"), (3,"C")]	1'(3,4,2,0)	1'[4,3]	1'[1,2]	empty
65432	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3"C"), (4,"D")]	1'(3,4,1,4)	1'[4,3]	1'[1,2]	1'4
65141	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3"C"), (4,"D")]	1'(3,4,1,4)	1'[3,4]	1'[1,2]	1'4
64162	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2"B"), (4,"D")]	1'(3,4,3,4)	1'[2,4]	1'[1,3]	empty
64092	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2"B"), (4,"D")]	1'(3,4,3,4)	1'[2,4]	1'[1,3]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
60385	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2"B"), (4,"D")]	1'(3,4,3,4)	1'[2,4]	1'[1,3]	1'2
60315	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2"B"), (4,"D")]	1'(3,4,3,4)	1'[2,4]	1'[1,3]	1'2
52689	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D")]	1'(3,4,3,4)	1'[1,4]	1'[2,3]	empty
52619	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done	1'(3,4,4)	1'[(1,"A"), (4,"D")]	1'(3,4,3,4)	1'[1,4]	1'[2,3]	empty
51951	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (3,"C")]	1'(3,4,4,3)	1'[1,3]	1'[2,4]	empty
50373	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D"), (3,"C")]	1'(3,4,2,0)	1'[3,4]	1'[2,1]	empty
50366	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D"), (3,"C")]	1'(3,4,2,0)	1'[4,3]	1'[2,1]	1'4
50323	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(4,4,2,0)	1'[4,3]	1'[2,1]	Empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
50298	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(4,4,2,0)	1'[3,4]	1'[2,1]	empty
50044	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D"), (3,"C")]	1'(3,4,2,0)	1'[3,4]	1'[1,2]	1'4
50037	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(4,"D"), (3,"C")]	1'(3,4,2,0)	1'[4,3]	1'[1,2]	1'4
49994	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(4,4,2,0)	1'[4,3]	1'[1,2]	empty
49969	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(4,4,2,0)	1'[3,4]	1'[1,2]	empty
49933	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(3,4,2,4)	1'[4,3]	1'[1,2]	1'3 1'4
49638	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(3,"C"), (2,"B")]	1'(3,4,4,0)	1'[2,3]	1'[1,4]	1'3
49625	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (4,"D")]	1'(3,4,2,4)	1'[3,4]	1'[1,2]	1'3 1'4

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
49609	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(3,"C"), (2,"B")]	1'(3,4,4,0)	1'[3,2]	1'[1,4]	1'3
49187	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (3,"C")]	1'(4,4,4,0)	1'[3,2]	1'[1,4]	empty
48929	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,4]	1'[1,3]	empty
48921	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,4]	1'[1,3]	empty
48706	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (3,"C")]	1'(4,4,4,0)	1'[2,3]	1'[1,4]	empty
48469	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (3,"C")]	1'(4,4,4,0)	1'[3,2]	1'[1,4]	1'2
46030	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(2,"B"), (1,"A")]	1'(4,4,4,0)	1'[1,2]	1'[4,3]	1'2
46017	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (1,"A")]	1'(4,4,4,0)	1'[1,2]	1'[3,4]	1'2

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
45669	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,4]	1'[1,3]	1'2
45661	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,4]	1'[1,3]	1'2
45446	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (3,"C")]	1'(4,4,4,0)	1'[2,3]	1'[1,4]	1'2
45355	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4"D"), (3,"C")]	1'(3,4,4,0)	1'[2,4,3]	1'[1]	empty
45137	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(2,"B"), (1,"A")]	1'(4,4,4,0)	1'[2,1]	1'[4,3]	1'2
45124	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (1,"A")]	1'(4,4,4,0)	1'[2,1]	1'[3,4]	1'2
41306	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(1,"A"), (2,"B")]	1'(4,4,4,0)	1'[2,1]	1'[4,3]	empty
41293	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (2,"B")]	1'(4,4,4,0)	1'[2,1]	1'[3,4]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
39106	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D")]	1'(4,4,3,0)	1'[1,4]	1'[3,2]	empty
39051	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(1,"A"), (4,"D")]	1'(4,4,3,0)	1'[1,4]	1'[2,3]	empty
39043	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D")]	1'(4,4,3,0)	1'[1,4]	1'[2,3]	empty
38828	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (3,"C")]	1'(4,4,4,0)	1'[1,3]	1'[2,4]	empty
38369	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (3,"C")]	1'(3,4,4,3)	1'[1,3]	1'[2,4]	1'3
37278	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[(1,"A"), (2,"B")]	1'(4,4,4,0)	1'[1,2]	1'[4,3]	empty
37265	1'ProcessedPacket(1,done), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (2,"B")]	1'(4,4,4,0)	1'[1,2]	1'[3,4]	empty
35797	1'ProcessedPacket(1,discarded), 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,3,4]	1'[1]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
35776	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,4,3]	1'[1]	empty
35091	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[3,2,4]	1'[1]	empty
33359	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(3,4,3,4)	1'[1,2,4]	1'[3]	1'2
33290	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(3,4,3,4)	1'[1,2,4]	1'[3]	1'2
33026	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,3,4]	1'[1]	1'2
33005	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,4,3]	1'[1]	1'2
32853	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,3,4]	1'[1]	1'4
32847	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,4,3]	1'[1]	1'4
32813	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[2,4,3]	1'[1]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
32790	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[2,3,4]	1'[1]	empty
32609	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(3,4,3,4)	1'[2,1,4]	1'[3]	1'2
32539	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(3,4,3,4)	1'[2,1,4]	1'[3]	1'2
29410	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(3,4,3,4)	1'[2,1,4]	1'[3]	empty
29410	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D"), (3,"C")]	1'(3,4,2,0)	1'[1,3,4]	1'[2]	empty
27540	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(3,4,3,4)	1'[2,1,4]	1'[3]	empty
27519	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D"), (3,"C")]	1'(3,4,2,0)	1'[1,4,3]	1'[2]	empty
27297	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(3,4,0,4)	1'[1,4,3]	1'[2]	1'4
26997	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(3,4,0,4)	1'[1,3,4]	1'[2]	1'4

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
26018	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(3,4,3,4)	1'[1,2,4]	1'[3]	empty
25948	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(3,4,3,4)	1'[1,2,4]	1'[3]	empty
25356	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (2,"B"), (4,"D")]	1'(3,4,1,4)	1'[1,4,3]	1'[1]	1'3 1'4
25332	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (2,"B"), (4,"D")]	1'(3,4,1,4)	1'[2,3,4]	1'[1]	1'3 1'4
25319	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(3,"C"), (2,"B"), (4,"D")]	1'(3,4,1,4)	1'[3,2,4]	1'[1]	1'3 1'4
25092	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[3,2,4]	1'[1]	empty
24950	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,3,4]	1'[1]	1'4
24943	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,4,3]	1'[1]	1'4
24900	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[2,4,3]	1'[1]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
24875	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[2,3,4]	1'[1]	empty
24726	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[3,2,4]	1'[1]	1'2
23429	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C")]	1'(4,4,4,0)	1'[1,3,2]	1'[4]	1'2
23171	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(4,4,3,0)	1'[1,2,4]	1'[3]	1'2
23163	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(4,4,3,0)	1'[1,2,4]	1'[3]	1'2
22948	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C")]	1'(4,4,4,0)	1'[1,2,3]	1'[4]	1'2
22868	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,3,4]	1'[1]	1'2 1'4
22861	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (4,"D"), (3,"C")]	1'(3,4,1,0)	1'[2,4,3]	1'[1]	1'2 1'4
22818	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[2,4,3]	1'[1]	1'2

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
22793	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (3,"C"), (4,"D")]	1'(4,4,1,0)	1'[2,3,4]	1'[1]	1'2
22595	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(4,4,3,0)	1'[2,1,4]	1'[3]	1'2
22587	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D")]	1'(4,4,3,0)	1'[2,1,4]	1'[3]	1'2
22372	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C")]	1'(4,4,4,0)	1'[2,1,3]	1'[4]	1'2
20186	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,1,4]	1'[3]	empty
20178	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(4,4,3,0)	1'[2,1,4]	1'[3]	empty
19963	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C")]	1'(4,4,4,0)	1'[2,1,3]	1'[4]	empty
18775	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D"), (3,"C")]	1'(3,4,2,0)	1'[1,3,4]	1'[2]	1'4
18768	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (4,"D"), (3,"C")]	1'(3,4,2,0)	1'[1,4,3]	1'[2]	1'4

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
18725	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(4,4,2,0)	1'[1,4,3]	1'[2]	empty
18700	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(4,4,2,0)	1'[1,3,4]	1'[2]	empty
18667	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(3,4,2,4)	1'[1,4,3]	1'[2]	1'3 1'4
18369	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (2,"B")]	1'(3,4,4,0)	1'[1,2,3]	1'[4]	1'3
18356	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (4,"D")]	1'(3,4,2,4)	1'[1,3,4]	1'[2]	1'3 1'4
18340	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (3,"C"), (2,"B")]	1'(3,4,4,0)	1'[1,3,2]	1'[4]	1'3
17918	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C")]	1'(4,4,4,0)	1'[1,3,2]	1'[4]	empty
17660	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(3,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(4,4,3,0)	1'[1,2,4]	1'[3]	empty
17652	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D")]	1'(4,4,3,0)	1'[1,2,4]	1'[3]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
17437	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (3,"C")]	1'(4,4,4,0)	1'[1,2,3]	1'[4]	empty
15564	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,3,4]	1'[]	1'2
15543	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,4,3]	1'[]	1'2
15180	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,4,3]	1'[]	1'2
15159	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,3,4]	1'[]	1'2
13475	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,3,4]	1'[]	empty
13454	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,4,3]	1'[]	empty

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
11693	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,3,4]	1'[]	empty
11672	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(1,"A"), (2,"B"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,4,3]	1'[]	empty
10941 0	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[]	1'(4,4,4,0)	1'[]	1'[2,1,4, 3]	empty
10939 7	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[]	1'(4,4,4,0)	1'[]	1'[2,1,3, 4]	empty
10888 2	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[]	1'(4,4,4,0)	1'[]	1'[1,3,2, 4]	empty
10883 8	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(3,4,4)	1'[]	1'(4,4,4,0)	1'[]	1'[1,2,4, 3]	empty
10882 5	1'ProcessedPacket(1,discarded) 1'ProcessedPacket (2,discarded) 1'ProcessedPacket (3,discarded) 1'ProcessedPacket (4,discarded)	1'(4,4,4)	1'[]	1'(4,4,4,0)	1'[]	1'[1,2,3, 4]	empty
10423	1'ProcessedPacket(1,done) 1'ProcessedPacket (2, done) 1'ProcessedPacket (3, done) 1'ProcessedPacket (4, done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[1,3,2,4]	1'[]	1'2
10281	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,3,4]	1'[]	1'2 1'4

Marcado	Blocks To Send	Windows Sender	List of Blocks received	Windows Receiver	List ACk received	List ACK discard Message	Block Marked
10274	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[1,2,4,3]	1'[]	1'2 1'4
10231	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[1,2,4,3]	1'[]	1'2
10206	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (3,"C"), (4,"D")]	1'(4,4,0,0)	1'[1,2,4,3]	1'[]	1'2
10038	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,3,4]	1'[]	1'2 1'4
10031	1'ProcessedPacket(1,done) 1'ProcessedPacket (2,done) 1'ProcessedPacket (3,done) 1'ProcessedPacket (4,done)	1'(4,4,4)	1'[(2,"B"), (1,"A"), (4,"D"), (3,"C")]	1'(3,4,0,0)	1'[2,1,4,3]	1'[]	1'2 1'4

A.6 Archivo “Marcados_muertos.txt”

```
digraph cpn_tools_graph {
  N9988;
  N9963;
  N96930;
  N96861;
  N96367;
  N96298;
  N95630;
  N8849;
  N8842;
  N8799;
  N8774;
  N81707;
  N81699;
  N81484;
  N81252;
  N81197;
  N81189;
  N80974;
  N80515;
  N8008;
  N7984;
  N7971;
  N79424;
  N79411;
```

N7745;

N7603;

N7596;

N75773;

N75760;

N7553;

N7528;

N68334;

N68290;

N68277;

N66111;

N66090;

N65684;

N65663;

N65432;

N65141;

N64162;

N64092;

N60385;

N60315;

N52689;

N52619;

N51951;

N50373;

N50366;

N50323;

N50298;

N50044;

N50037;

N49994;

N49969;

N49933;

N49638;

N49625;

N49609;

N49187;

N48929;

N48921;

N48706;

N48469;

N46030;

N46017;

N45669;

N45661;

N45446;

N45373;

N45355;

N45137;

N45124;

N41306;

N41293;

N39106;

N39051;

N39043;

N38828;

N38369;

N37278;

N37265;

N35797;

N35776;

N35091;

N33359;

N33290;

N33026;

N33005;

N32853;

N32847;

N32813;

N32790;

N32608;

N32539;

N29410;

N29341;

N27540;

N27519;

N27297;

N26997;

N26018;

N25948;

N25356;

N25332;

N25319;

N25092;

N24950;

N24943;

N24900;

N24875;

N24726;

N23429;

N23171;

N23163;

N22948;

N22868;

N22861;

N22818;

N22793;

N22595;

N22587;

N22372;

N20186;

N20178;

N19963;

N18775;

N18768;

N18725;

N18700;

N18667;

N18369;

N18356;

N18340;

N17918;

N17660;

N17652;

N17437;

N15564;

N15543;

N15180;

N15159;

N13475;

N13454;

N11693;

N11672;

N109410;

N109397;

N108882;

N108838;

N108825;

N10423;

N10281;

N10274;

N10231;

N10206;

N10038;

N10031;}