Control de Congestión en Redes Inalámbicas de Sensores

Camilo Alejandro Medina Mondragón

Maestría Ingeniería Electrónica
Trabajo de Investigación

Pontificia Universidad Javeriana
Facultad de Ingeniería
Departamento de Electrónica
2017

AGRADECIMIENTOS

Extiendo un agradecimiento sincero al Ingeniero Manuel Ricardo Pérez Cerquera Ph.D, director del presente proyecto de investigación, por sus conocimientos compartidos, sus sugerencias y comentarios oportunos. Sus indicaciones me han permitido adquirir fuertes conocimientos en telecomunicaciones y redes inalámbricas.

Mi agradecimiento muy especial al asesor Ingeniero Daniel Jaramillo Ramírez Ph.D por brindarme herramientas académicas e investigativas que fueron sumamente útiles para el desarrollo del presente proyecto, por su tiempo compartido y su constante apoyo. Además su enseñanzas en las aulas de clase me han hecho adquirir un espectro más amplio de las telecomunicaciones que estoy seguro aplicaré durante mi desarrollo profesional.

Al Centro de Excelencia y Apropiación en IoT, en especial a los Ingenieros Luis Carlos Trujillo Arboleda M.Sc. y Diego Méndez Chaves Ph.D por confiar en mi para formar parte de este excelente equipo de ingenieros y por todo su apoyo en mi desarrollo académico y profesional durante el curso de mi maestría.

A mis compañeros y amigos de la maestría y del CEA-IoT que siempre estuvieron presentes con sus palabras de apoyo y ánimo durante el desarrollo del presente proyecto.

A mi familia y Carolina por brindarme su cariño, confianza y apoyo constante. Estaré eternamente agradecido con ustedes.

Resumen

La congestión en una red inalámbrica de sensores causa diferentes comportamientos no deseados como una baja eficiencia energética, uso de no eficiente de los recursos de red, latencias en la comunicación, alta probabilidad de degradación de servicio, alto riesgo de perdida y bloqueo de paquetes, entre otros. Con las proyecciones de aumento de tráfico y la adopción del IoT en la vida diaria de las personas, es necesario investigar y desarrollar métodos de control de congestión que detecten, notifiquen y mitiguen la congestión en una WSN. Para cumplir este reto, se propone un método de control de congestión multi-capa y distribuido llamado MLDC el cual considera las capas 1) Acceso prioritario al canal, 2) Administración del buffer del nodo, 3) Algoritmo de enrutamiento para evitar y mitigar congestión y 4) Aplicación tipo MLDC. Se realiza una novedosa evaluación comparativa de diferentes métodos de control de congestión en diferentes topologías de red encontrando que el método propuesto MLDC funciona adecuadamente en las topologías evaluadas mejorando el comportamiento de la red y métricas propuestas de evaluación en una WSN.

Índice general

Índice de cuadros

Capítulo 1

Introducción

1.1. Redes inalámbricas de sensores

Una Redes Inalámbica de Sensores está compuesta de dispositivos de baja potencia los cuales son distribuidos en un área geográfica acotada para realizar algún tipo de monitoreo [1], el canal de comunicación entre los dispositivos es inalámbrico y la información recolectada por cada nodo es enviada por este medio a una estación base (Nodo *Sink*) para que este a su vez lo envíe a una red o servidor específico. Este tipo de redes son consideradas auto-configurables en cuanto a su topología, enrutamiento, generación de agrupaciones (*Clusters*), control de fallas, entre otros. [2]

La Figura 1.1 representa el flujo típico de información en una WSN. El nodo *Sink* puede tener las mismas especificaciones de los otros sensores, puede ser un dispositivo personalizado como un *Smart Phone*, computador o PDA, o tipo estación base que conecta la red a Internet a un centro de control y/o monitoreo, la cual es una arquitectura típica en Internet de las Cosas (IoT).

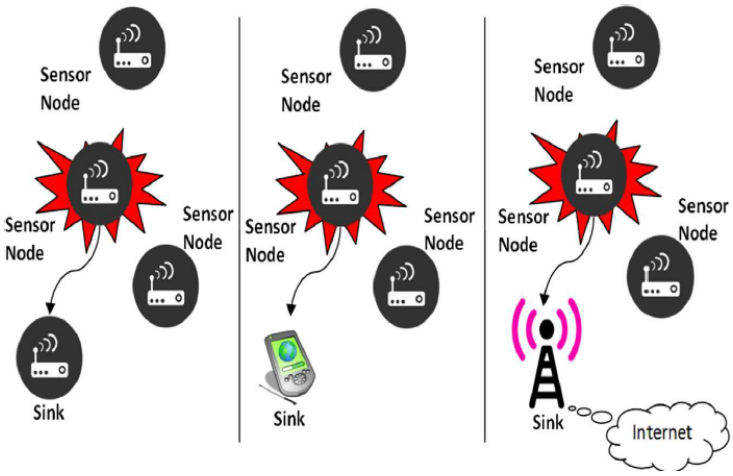


Figura 1.1: Arquitectura de una WSN. Tomado de [3]

Actualmente las redes inalámbricas de sensores están en constante desarrollo e investigación por la comunidad cientifica y la industria [4], lo que ha llevado a que se implementen en diferentes aplicaciones militares, ambientales, logistica, HAR (Human Activity Recognition), urbanas entre otros. En [3] se

presenta un estudio detallado de aplicaciones de WSN en áreas urbanas donde se evidencia que estas redes pueden variar significativamente dependiendo de la aplicación, área de cobertura, tipos de nodos, tolerancia a fallas, tiempo de vida, etc. Por este motivo el diseño de una WSN implica un trabajo de diseño exhaustivo en el cual se tenga en cuenta protocolos de enrutamiento confiables, algoritmos de control de congestión, algoritmos de auto-curación entre otros.

1.2. Control de congestión en WSN

Debido a las características propias de la red y los nodos, se puede presentar el fenómeno de congestión el cual está directamente relacionado con la capacidad de los nodos y la cantidad de tráfico generado y transportado. Sean: λ la tasa de llegadas de paquetes a un punto de la red (enlace o nodo), μ la tasa de servicio de paquetes en el punto y ρ la utilización del mismo, la congestión en la WSN se presenta cuando la tasa de llegadas supera a la tasa de servicio como se muestra en la ecuación 1.1.

$$\rho = \frac{\lambda}{\mu} > 1 \tag{1.1}$$

La congestión puede generar:

- Un alto riesgo de paquetes perdidos o bloqueados.
- Mayores probabilidades de degradación de servicio.
- Uso extra de recursos en caso de re-transmisiones.
- Mayores latencias en la comunicación.
- Discontinuación de la conexión del canal.
- Baja calidad de servicio.
- Pérdidas significativas de energía.
- Información obsoleta en la red debido a los altos retardos en los procesos de transmisión

Por este motivo es necesario implementar métodos de control de congestión que disminuyan la probabilidad de presencia de congestión en la red, o que actúen de forma adecuada cuando esta ya se sucedió.

Específicamente en WSN, diferentes autores [5] [6] [7] clasifican la congestión según la ubicación donde se presentó:

- Congestión a nivel de nodo: Es directamente descrito cuando la utilización del nodo en cuanto a su tráfico es mayor a 1 (ecuación 1.1). Este tipo de congestión ocurre comúnmente en los nodos cercanos al sumidero (*Sink node*).
- Congestión a nivel de enlace: Es producido cuando es necesario competir con otros nodos para acceder al medio inalámbrico compartido, además cuando suceden colisiones y se presentan errores en la transmisión debido a interferencia.

Los métodos de control de congestión en WSN constan de tres fases:

- Detección: Se refiere al proceso de detección de la congestión en algún punto específico de la WSN. Algunos parámetros usados en este proceso son: tamaño de la cola, carga en el canal, tiempo de servicio de paquetes y la combinación ocupación buffer y carga del canal.

- Notificación: Una vez la congestión es detectada, se debe informar a nodos vecinos la presencia de la congestión en un punto específico de la red. Esta notificación se puede realizar de las siguientes formas: - Implícita. Se utilizan paquetes de enrutamiento en cuyas cabeceras se informa la situación de la congestión. - Explicita se introducen nuevos paquetes a la red informando el evento de congestión. Este método de notificación no es deseado ya que utiliza paquetes adicionales de control que pueden generar aún más carga en la red.
- Control: Se refiere a las acciones reactivas cuando se presenta congestión. Los métodos pueden ser clasificados en: Control de tráfico y control de recursos.

Los métodos que ejercen control de tráfico buscan reducir la tasa de transmisión en los nodos congestionados (mediante la reducción del la tasa de llegada de paquetes λ) informando a los nodos transmisores sobre el evento que sucedió. Existen diferentes formas de disminución de tráfico por unidad de tiempo, por ejemplo AIMD (*Additive increase/multiplicative decrease*) que es usado en TCP mediante la fórmula matemática presentada en la Ecuación 1.2, la cual representa un aumento lineal en la tasa de transmisión cuando no existe congestión y una reducción exponencial cuando se presenta.

$$f(t+1) = \begin{cases} f(t) + a & \text{si } c = 0 \\ f(t) * b & \text{si } c = 1 \end{cases} \quad a > 0,\ 0 < b < 1 \tag{1.2}$$

Por otra parte los métodos que ejercen control de recursos buscan que cuando se presente congestión se usen nodos redundantes en la red o se creen nuevas rutas mediante la modificación de la topología original de la red.

1.3. Motivaciones y contribuciones

Con los últimos avances en tecnología, la miniaturización de MEMS (*Micro-Electro-Mechanical Systems*) y la adopción del IoT en la industria y en la vida cotidiana, el tráfico en Internet está creciendo significativamente. Cisco ha catalogado este crecimiento como la era Zettabyte ya que afirman que el tráfico IP alcanzará 3.3 ZB por año en 2021 [8]. En la Figura 1.2 se presenta la proyección de cantidad de dispositivos que se conectarán a Internet en los próximos años. El tráfico M2M (Machine-to-Machine) será aproximadamente el 50 % de dispositivos conectados los cuales son el tipo de dispositivos más comunes en IoT y WSN.

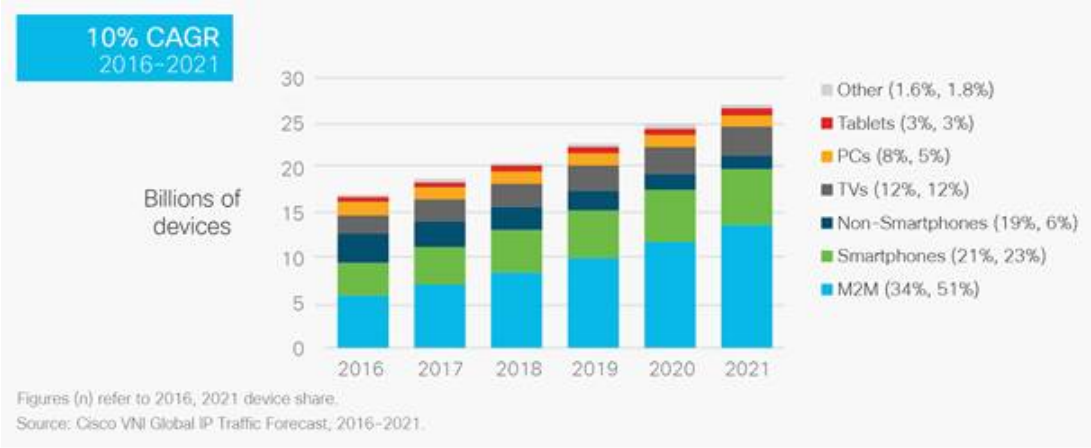


Figura 1.2: Dispositivos y crecimiento de conexiones. Tomado de [8].

Esta proyección en el aumento del tráfico hace importante que se realicen investigaciones en el campo del control de congestión, ya que se hace cada vez más necesario detectar, prevenir y mitigar la congestión en dispositivos limitados en capacidad y energía como en los nodos de una WSN.

El presente trabajo de investigación tiene como aportes a la comunidad científica:

- Definición de métricas relacionadas a congestión en WSN para la evaluación de algoritmos desarrollados. Con estas métricas se pretende tener medidas para evaluación de parámetros de QoS relacionados al control de congestión en WSN.
- Evaluación comparativa de tres algoritmos de control de congestión publicados bajo las métricas propuestas.
- Formulación, simulación y evaluación de un nuevo algoritmo de control de congestión.

1.4. Objetivos

OBJETIVO GENERAL

Desarrollar e implementar un algoritmo de control de congestión que optimice parámetros de calidad de servicio en una WSN.

OBJETIVOS ESPECÍFICOS

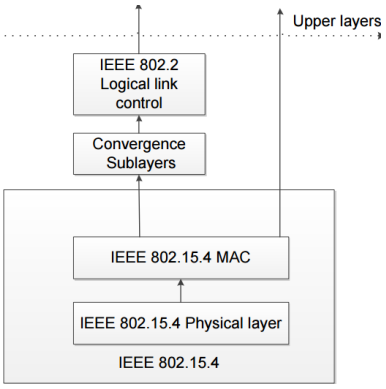
- Identificar y seleccionar métodos de control de congestión en WSN.
- Selección y definición de métricas de QoS relacionados a congestión en WSN.
- Implementar y evaluar tres métodos de control de congestión en diferentes topologías de red mediante simulaciones.
- Desarrollar una técnica de control de congestión que optimice parámetros de QoS.
- Evaluar comparativamente los resultados de las técnicas de control de congestión analizadas y la técnica desarrollada.

Capítulo 2

Estado del Arte

2.1. Estándar IEEE 802.15.4

IEEE 802.15.4 es un estándar que especifica la capa física y de control de acceso al medio para redes tipo LR-WPANs (Low Rate Wireless Personal Area Networks) las cuales son de bajo costo y baja tasa de transmisión. Las primeras versiones de este estándar especificaban comunicaciones de hasta 10 metros con tasa de transferencia de hasta 250 Kbits/seg. En la Figura 2.1a se presenta el stack del estándar.



(a) Capas Estándar 802.15.4

OPTIONS FOR FREQUENCY ASSIGNMENTS			
Geographical regions	Europe	Americas	Worldwide
Frequency assignment	868 to 868.6 MHz	902 to 928 MHz	2.4 to 2.4835 GHz
Number of channels	1	10	16
Channel bandwidth	600 kHz	2 MHz	5 MHz
Symbol rate	20 ksymbols/s	40 ksymbols/s	62.5 ksymbols/s
Data rate	20 kbits/s	40 kbits/s	250 kbits/s
Modulation	BPSK	BPSK	Q-QPSK

(b) Asignación de Frecuencias

Figura 2.1: Estándar 802.15.4. Tomado de [9].

La capa física controla el transceptor de frecuencia y la selección de canales en diferentes bandas las cuales son las presentadas en la Figura 2.1b, en la cual se especifica el esquema de modulación utilizado, la tasa de datos, y la cantidad de canales disponibles en cada una de las bandas. Es necesario tener en cuenta la banda y el canal de transmisión ya que por ejemplo en 2.4 GHz también se encuentran los protocolos Bluetooth y WiFi (802.11b). La separación entre los canales es de 5 MHz. Es recomendable realizar la transmisión en los canales 25 y 26 ya que la interferencia es menor.

La capa en enlace se divide en dos subcapas: control de acceso al medio (MAC) y control de enlaces lógicos (LLC). La subcapa MAC (dependiente del Hardware) utiliza acceso tipo CSMA/CA. La

subcapa LLC es común a todos los estándares tipo IEEE 802, y especifica control de errores y direccionamiento de la subcapa MAC.

Este estándar puede funcionar de dos formas:

- Beacon Mode: Los nodos siempre están sincronizados con un coordinador de la red. Es usado cuando se requiere que todos los dispositivos estén sincronizados con un PAN Coordinator el cual envía periódicamente un paquete Broadcast especial, el cual también puede servir para actualizar las tablas de enrutamiento.
- Non-Beacon Mode: En este tipo, cada dispositivo es autónomo y puede transmitir información a su coordinador en culquier momento, por lo tanto el PAN Coordinator siempre tiene que estar activo y esperando nuevas conexiones.

2.1.1. Protocolo ZigBee

Zigbee es un protocolo muy usado en redes inalámbricas tipo Low Data Rate (baja tasa de datos) que fue concebida en el año 2005 por ZigBee Alliance. Este protocolo ha sido muy utilizado para aplicaciones que requieran bajo consumo de energía y alta tolerancia a bajas tasas de datos. Zigbee toma todas especificaciones de radio y capa MAC del estándar IEEE 802.15.4, y añade la capa lógica de red, seguridad y software de aplicación como se presenta en la Figura 2.2

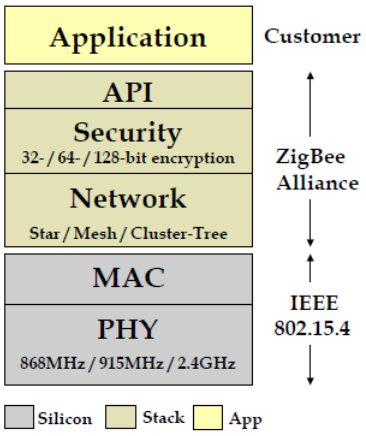


Figura 2.2: Stack de Zigbee

El algoritmo de enrutamiento que usa Zigbee es AODV (Ad-hoc On-demand Distance Vector), el cual se considera de tipo proactivo ya que descubre una ruta a un destino cuando se requiere, por lo que no mantiene una tabla de enrutamiento actualizada constantemente, únicamente por demanda. La métrica que utiliza es "costo del enlace".

La capa de seguridad de Zigbee se encarga de la correcta administración de llaves simétricas, la implementación de políticas de seguridad mediante el uso de llaves de 128-bits para sus mecanismos de seguridad como son el establecimiento de una comunicación inalámbrica segura, cifrado de tramas y control de dispositivos.

2.2. Métodos de control de congestión

El presente trabajo de grado tiene como objetivo proponer un método de control de congestión en WSN y evaluar el comportamiento de algunos algoritmos que actualmente se encuentren en el estado del arte y hayan sido implementados. Para realizar la selección de los mismos, la metodología realizada se basó en la lectura de diferentes artículos tipo *survey* [5] [6] [7] los cuales realizan la comparación de los métodos y algoritmos disponibles, enuncian sus fortalezas y debilidades y brindan ciertas recomendaciones sobre la investigación en ese campo.

Con base en la lectura realizada, se preseleccionaron 8 métodos de control de congestión [10], [11], [12], [13], [14], [15], [16],[17], y basado en los siguientes criterios se escogieron 3 de ellos:

- Año de publicación. La selección partió del análisis del año de publicación descartando los artículos con más de 10 años.
- Cantidad de citas. Este es un parámetro importante ya que indica indirectamente la calidad del articulo publicado y la utilización, puesta en producción y/o análisis del mismo por otros autores.
- Pruebas realizadas. Se analizaron los experimentos y los resultados de los mismos que se presentaban en cada articulo, con su respectivo análisis de las métricas de QoS (calidad de servicio) evaluadas.
- Diversidad. Es necesario analizar métodos que difieran en la forma de detección y en su forma de controlar la congestión (control de recursos o control de tráfico).

Con base en los resultados se seleccionaron los métodos DALPAS, ECODA y FUSION:

2.2.1. DALPAS

Dynamic Alternative Path Selection (DALPaS) [11] es un algoritmo de control de congestión que utiliza control de recursos mediante caminos alternativos para mitigar la congestión. La capa de acceso al medio MAC está basada en CSMA.

En general DALPaS consta de 4 etapas:

- Setup: Sólo se ejecuta una vez en el momento de la inicialización de la red. Los nodos se se descubren unos a otros y actualizan sus tablas de enrutamiento. Esta fase comienza desde el nodo *Sink* enviando un Broadcast *HELLO* con su nivel el cual es cero (El nivel indica la distancia en saltos al *Sink*). Cada nodo que recibe este mensaje, responde con un *ACK*, cuando el *Sink* recibe el mensaje *ACK*, este le responde con un mensaje *CONECT*. Este nodo puede comunicarse directamente con el *Sink*, por lo que lo convierte en un nodo nivel 1 y actualiza esta información en la tabla de enrutamiento. Este proceso continua así sucesivamente en toda la red. Cada nodo almacena una tabla de vecinos, en la Figura 2.3 se presenta un ejemplo de un nodo de nivel 2. Esta tabla indica el ID de los nodos vecinos, la ocupación del *buffer* de cada uno, la energía restante, el nivel y un Flag que indica la disponibilidad del nodo.

Node ID	Buffer occupancy	Remaining power	Level	Flag
2	0.84	0.91 J	1	True
3	0.90	0.95 J	1	True
7	0.92	0.96 J	2	True

Figura 2.3: Ejemplo tabla de vecinos en DALPAS. Tomado de [11]

- Topology Control Scheme: Emplea una forma de control de topología dinámico sin añadir carga extra a la red. Utiliza la información de la tabla de vecinos creada en la fase anterior para transmitir los datos, cada nodo revisa esta tabla y envía la información al nodo vecino con menor nivel (más cerca del nodo *Sink*). Cuando un nodo tiene mas de un nodo con el mismo nivel en la tabla de vecinos, escoge el siguiente salto basado en Round-Robin. Usando esta técnica el tráfico es separado entre los nodos por lo que se el consumo de energía es balanceado antes que la congestión ocurra.

- Soft Stage Scheme: Esta fase se presenta cuando un nodo recibe mas de un flujo, en este caso, el nodo congestionado informa a uno de los nodos que prefiere dejar de recibir flujos de ese nodo. El nodo que recibe este mensaje está programado para verificar caminos alternativos. Por temas de rendimiento, cada nodo escoge servir un flujo con la mas alta tasa. Esta fase solo aconseja a los nodos encontrar un camino alternativo.

- Hard Stage Scheme: Si en el paso anterior el nodo no cambió su enrutamiento, y es necesario actual reactivamente ante la congestión, es necesario pasar a esta fase. El propósito es cambiar el valor de la bandera de la tabla de vecinos de True a False o viceversa. Cuando un nodo altera el valor de este campo, este cambio se informa a los vecinos mediante paquetes ACK modificados.

  Un nodo puede declararse como *No Disponible* cuando la ocupación de *buffer* ha llegado a un limite específico, se tiene muy poca potencia y cuando no tiene nodos de diferentes niveles disponibles en la tabla.

Para actualizar la tabla de vecinos, el nodo organiza la tabla con respecto a el nivel en la red (saltos para el nodo *Sink*), potencia restante (comparado con puntos específicos) y basado en la ocupación de *buffer*. Si se presenta el caso en el que existe más de un nodo con los mismos valores de estos parámetros se escoge el que tenga el menor ID para enviar el paquete.

2.2.2. ECODA

Enhanced Congestion Detection and Avoidance [10], es un algoritmo catalogado como control de tráfico, que utiliza una técnica de doble *buffer* para detectar congestión en los nodos. Se definen estados de *buffer accept state*, *filter state* y *reject state*. El método hace uso de dos umbrales en el *buffer* adoptando diferentes estrategias para aceptar o rechazar paquetes. En la Figura 2.4 se presenta un ejemplo con los dos umbrales y los estados de un *buffer* específico, para cada estado existe una técnica especial para aceptar o rechazar los paquetes.

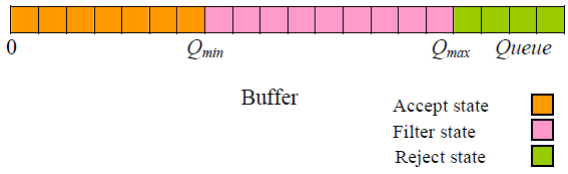


Figura 2.4: Estados del buffer en ECODA

Ecoda define dos tipos de prioridad a los paquetes:

- Prioridad Estática: es un numero entero que se le asigna a los paquetes en el nodo origen para identificar diferentes servicios. Por ejemplo es posible asignar prioridades a paquetes de datos, de enrutamiento o tipo beacon.

- Prioridad Dinámica: este numero cambia constantemente con cada salto que da el paquete en la red. Su valor depende de la prioridad estática, el delay que tiene el paquete desde el momento que fue generado, el numero de saltos y dos parámetros de ajuste:

$$DP = \frac{\alpha \cdot hop + SP}{1 + \beta \cdot delay} \tag{2.1}$$

En cada nodo se crean dos sub-colas como se presenta en la Figura 2.5; en la cola de tráfico enrutado, los paquetes son organizados dependiendo su prioridad dinámica. Se utiliza el método round robin entre las dos sub-colas para seleccionar el siguiente paquete a enviar.

Una vez se se reciben paquetes en *buffer* del nodo, se pueden presentar los siguientes escenarios:

- Si 0 ≤ N ≤ Qmin. Todos los paquetes entrantes son puestos en el *buffer* debido a que la utilización de la cola es baja.
- Si Qmin ≤ N ≤ Qmax. Algunos paquetes con baja prioridad son rechazados o sobre-escritos con paquetes de mayor prioridad.
- Si Qmax ≤ N ≤ Q. Algunos paquetes con alta prioridad son rechazados o sobre-escritos .

Cuando se presenta congestión los parámetros δ_1 y δ_2 son utilizados para limitar el crecimiento del *buffer* y filtrar paquetes. Por ejemplo cuando se ha alcanzado el nivel Qmin, algunos paquetes con baja prioridad son rechazados para limitar R≤ δ_1, esto mismo sucede cuando Qmax es alcanzado pero con el parámetro δ_2. Estos parámetros pueden ser configurables dependiendo de la aplicación.

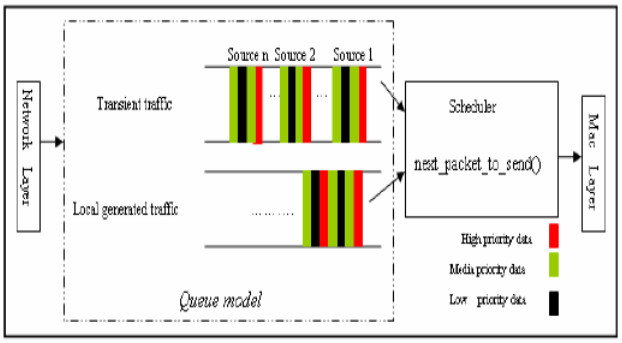


Figura 2.5: Modelo de colas en ECODA

Cuando la congestión es persistente, se propone un paso adicional que incluye la determinación de un camino de enrutamiento al *Sink* basado en el delay de los paquetes para detección de nodos cuello de botella y control de tasa de envío de datos desde el origen tipo AIMD.

2.2.3. FUSION

Este método de control de congestión utiliza tres diferentes capas: Control de flujo salto a salto, disminución de la tasa de tráfico de origen cuando existe tráfico de enrutamiento y acceso al medio de

forma prioritaria [16]. El primero es utilizado para prevenir que los nodos transmitan si los paquetes están destinados a ser rechazados o perdidos debido a espacio insuficiente en las colas de los nodos downstream, la disminución de la tasa de tráfico se usa para prevenir *unfairness* alrededor del nodo *Sink*, y la MAC priorizada asegura que los nodos reciban acceso prioritario al canal. Fusion está catalogado como un método de control de tráfico.

- Hop-by-hop flow control: Cada nodo configura un bit en la cabecera de cada paquete saliente del nodo y debido a que el canal inalambrico es completamente broadcast, todos los nodos vecinos se pueden enterar del estado de congestión en un nodo específico. Para detectar la congestión se revisa el tamaño de la cola, y la carga del canal mediante muestras del canal en intervalos de tiempo periódicos. Cuando un nodo sensor recibe un paquete que un nodo está congestionado, este detiene la transmisión permitiendo que el nodo congestionado drene su *buffer*.
- Rate Limiting: En este cada nodo-sensor monitorea el tráfico en transito para determinar la tasa de envío de estas hacia el nodo padre, con base en esta información se calcula N, el numero total de rutas por el padre. Específicamente se usa el método de *Token Bucket* para regular la tasa de envío de cada nodo. El nodo transmite cuando la cuenta de token es superior a cero.
- MAC Layer: Las formas de control descritas previamente, no pueden reaccionar rápido ni evitar que el *buffer* se llene sin la ayuda de la capa de control de acceso al medio. En este caso CSMA (Carrier Sense Multiple Access) puede ayudar a controlar la congestión. En este método de control de acceso, todos los sensores compiten para transmitir, por este motivo cuando la congestión está presente la probabilidad de pérdida de paquetes aumenta ya que los nodos no pueden transmitir y sus *buffer* se están llenando cada vez más. Por este motivo es necesario tener acceso al medio con prioridad. La técnica adoptada consiste en utilizar el *backoff window* para realizar contención de transmisión cuando el nodo no esta congestionado, es decir, si un nodo tiene mas información para enviar, su *backoff window* es menor.

Estos algoritmos evaluados difieren en su método de detección, notificación y mitigación de la congestión. En la Tabla 2.1 se presenta un resumen de las características principales de cada uno de estos.

Método	**Detección de Congestión**	**Notificación de Congestión**	**Control de Congestión**	**Capas de control**	**Ventajas**	**Desventajas**
ECODA	Tamaño del buffer	Implicita	Control de Trafico	Buffer Red	- Muy buena Administración del Buffer. - Establecimiento de prioridades de paquetes	Únicamente se centra en manejo de buffer y no considera método adicional
DALPAS	- Tamaño del buffer	Explicita	Control de recursos	Red	- Modifica la topología para convertirla en un árbol - Tiene en cuenta la energía y disponibilidad del nodo para decidir enrutamiento	Únicamente se centra en el manejo de la capa de red y no considera método adicional
FUSION	- Tamaño del buffer - Carga del canal	Implicita	Control de Trafico	MAC Red Aplicación	- Implementa acceso prioritario al canal cuando hay congestión. - Involucra a la aplicación generadora de trafico	- El método Token Bucket no es eficiente y puede llevar a llenar mucho el buffer. - Cuando hay congestión en un padre se detiene la transmisión de paquetes, esto lleva a que se aumente el tamaño del buffer.

Cuadro 2.1: Comparación métodos de control de congestión

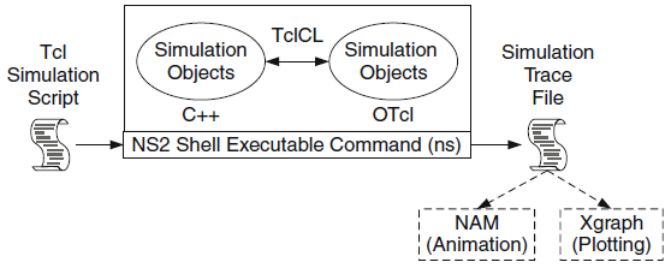


Figura 2.6: Arquitectura Basica de NS-2. Tomado de [19]

2.3. Software de simulación NS-2

Actualmente se tienen disponibles diferentes herramientas de simulación de WSN como NS-2, NS-3, OMNET ++, GloMoSiM, las cuales han sido ampliamente usadas para simular fenómenos relacionados a las redes inalámbricas. En la Tabla 2.2 se presenta la herramienta de simulación utilizada por cada método de control de congestión pre-seleccionado.

Plataforma Simulación	**Método**
NS-2	CADA, CONSISE, ECODA
Prowler Simulator	HTAP, DALPAS
OPNET	HOCA
MICA2 Mote (TinyOS)	FUSION

Cuadro 2.2: Plataformas de simulación usadas

Para el presente trabajo de investigación se hace uso de NS2 (Network Simulator Version 2) el cual es un simulador basado en eventos de código abierto que ha ganado bastante popularidad en la comunidad científica desde 1989 debido a su flexibilidad y naturaleza modular.

NS2 tiene diferentes módulos para simular redes cableadas, satelitales, inalámbricas, celulares entre otras. Este simulador permite incluir nuevos módulos de diferentes tipos para realizar evaluaciones de algoritmos y protocolos por lo que es utilizado antes de la implementación real o puesta en producción para conocer su comportamiento. En [18] se presenta una comparación entre diferentes simuladores de código abierto para redes inalámbricas y aunque NS-3 es la que mejor presenta resultados, NS-2 tiene mejores desarrollos para redes de sensores y el estándar IEEE 802.15.4.

2.3.1. Arquitectura

En la Figura 2.6 se presenta la arquitectura básica de NS-2, la cual consiste en dos lenguajes de programacion: C++ y OTcL (Object-oriented Tool Command Language)[19]. C++ define los mecanismos internos de la simulación (Backend), mientras que OTcL configura la simulación mediante la creación configuración de objetos y el manejo de los eventos (Frontend). C++ y OTcL están unidos por TcLCL mediante el cual se permite acceder a variables, funciones y datos desde los dos lenguajes de programación hasta el otro.

Después de una simulación, NS-2 presenta los resultados en archivos de texto, para interpretarlos gráficamente, la herramienta NAM (Network AniMator) y XGraph pueden ser utilizados. Además para analizar un comportamiento particular de la red (métricas como Delay, Troughput, Paquetes perdidos,

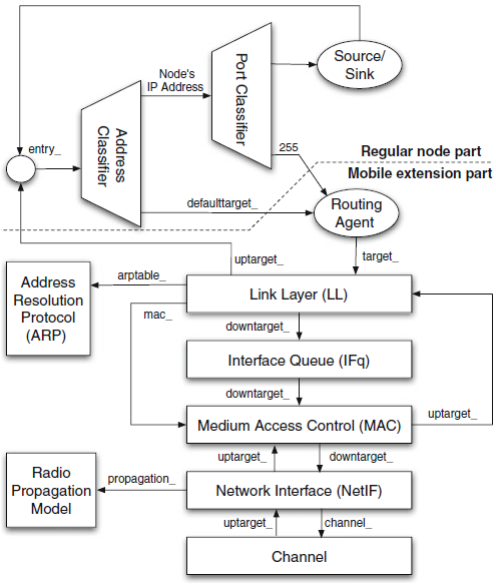


Figura 2.7: Arquitecura interna de nodos inalámbricos NS2. Tomado de [19]

etc), se usan scripts tipo AWK que permiten procesar datos a partir de los archivos de salida de la simulación.

2.3.2. WSN en NS2

En NS-2 se han realizado diferentes implementaciones y evaluaciones de nuevos algoritmos y protocolos como [20], [10], [21], en este ultimo se presenta como se diseña e implementa una simulación de una WSN en NS-2 en la cual se resalta la importancia de tener una configuración correcta de los parámetros del canal inalámbrico, la capa de control de acceso al medio y el modelo de energía adecuado para los nodos, en el cual se especifica la energía inicial, potencia de transmisión, potencia de recepción y potencia de sensado. En el Capitulo 4 se presenta el detalle de la definición de los parámetros de la simulación.

En la Figura 2.7 se presenta la arquitectura de nodos inalámbricos en NS-2. Cada bloque representa una clase en C++, por lo que tener módulos funcionales en los nodos, hace que la implementación y evaluación de algoritmos nuevos sea integrables con relativa facilidad a un modelo de nodo específico. Los cambios y actualizaciones a los módulos del simulador para implementar los algoritmos de control de congestión se realizan en el *Routing Agent*, *Interface Queue* y/o *Medium Access Control* los cuales permiten definir el algoritmo de enrutamiento de que usan los nodos, manejar los paquetes en el *buffer* del nodo y el método de control de acceso al medio inalambrico compartido respectivamente.

En la Figura 2.8 se presentan la topologías que se pueden generar en una red IEEE 802.15.4 con base en los tipos de dispositivos y funcionalidades de los mismos. NS permite configurar tres tipos de dispositivos en una red inalambrica basada en este estándar:

- Full Function Device: El nodo puede servir como coordinador de red (permitiendo enrutar paquetes y sincronizar otros nodos) y como un nodo común (realizar sensado y generación de tráfico).

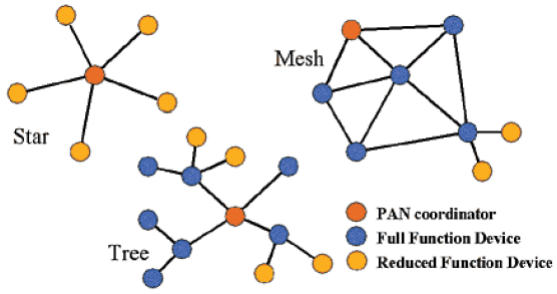


Figura 2.8: Topologias IEEE 802.15.4 . Tomado de [22]

- PAN Coordinator: Es un tipo especial de FFD, el cual actúa como coordinador de toda una red. Es asociado a un nodo *Sink* en una WSN. Comúnmente existe solo un nodo de estas características en la red, aunque se pueden tener redes con multiples *Sink* sin embargo no se consideran en el presente.
- Reduced Function Devices: Estos son nodos extremadamente simples los cuales únicamente se pueden comunicar con dispositivos tipo FFD. No cumplen funciones de enrutamiento ni de sincronización con otros nodos.

Con base en estas funcionalidades de los nodos se generan las diferentes topologías de red: estrella, arbol y malla sobre las cuales se realizarán las respectivas evaluaciones de los métodos de control de congestión.

Otra característica importante de una red inalambrica de sensores es que esta debe ser limitada en energía. Para esto NS2 tiene disponible un modelo que permite simular las características energéticas del nodo mediante la especificación de la energía inicial del nodo, potencia de transmisión, potencia de recepción, y potencia utilizada en el sensado y procesamiento. Cuando un nodo en la red Ad-hoc consume toda la energía, el dispositivo no es capaz de realizar ningún tipo de procesamiento interno, no puede recibir paquetes ni enviar por lo que este queda por fuera de la red.

Capítulo 3

Multi-Layer and Distributed Congestion Control

Para mitigar la congestión en una red inalambrica de sensores, existen diferentes métodos que actúan en diferentes capas de comunicación en los nodos. En [23] se presenta una clasificación de los mecanismos de control de congestión basada en la capa que recae el control, en donde se analiza que las técnicas que se basan en más de una capa le dan mayor confiabilidad a la red ya que hay más puntos de control y variables a controlar. La mayoría de los métodos de control de congestión investigados actúan en máximo dos niveles, sin embargo se enfocaban principalmente en uno de ellos. En el presente trabajo de investigación se propone el método MLDC (*Multi-Layer and Distributed Congestion Control*) el cual implementa acciones en la capa MAC, red, manejo del *buffer* y capa de aplicación para mitigar los efectos producidos en una red cuando se presenta congestión en algún punto de la red.

Además de las características *Multi-Layer*, se considera un control distribuido sobre todos los nodos de la red (origen, *Sink* e intermediarios). Este tipo de métodos tienen mayores ventajas con respecto a los centralizados en una WSN porque se pueden tomar acciones preventivas y correctivas salto a salto [24] y son más tolerantes a fallas a costa de más procesamiento en el nodo y latencia [25].

MLDC se considera un algoritmo que resuelve la congestión a nivel de nodo (con el manejo de buffer, algoritmo de enrutamiento y aplicación) y a nivel de enlace con el acceso al medio prioritario.

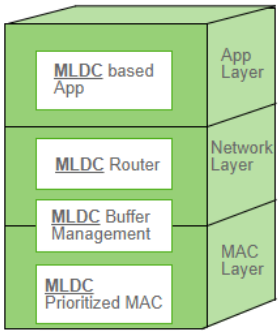


Figura 3.1: Módulos MLDC

En la Figura 3.1 se presentan los módulos diseñados en MLDC y su respectiva capa donde realizan sus acciones para controlar la congestión. A continuación se presenta el detalle de cada uno de estos:

3.1. Acceso prioritario al canal

Los métodos tradicionales de control de congestión en WSN no siempre pueden reaccionar a cambios repentinos de tráfico y del estado de la congestión debido a que son basados en la capa de red o superiores [16]. *M. Zawodniok et al* en [26] justifica que el uso de capas inferiores para el manejo de la congestión presenta mejores resultados debido a que los esquemas tradicionales dan resultados con un porcentaje alto de paquetes perdidos, escenarios injustos, bajo throughput y un significante desperdicio de energía debido a re-transmisiones. En MLDC se propone el uso de un acceso prioritario al canal propuesto en [16] cuando el nodo se encuentra en estado *congestionado* (estado descrito en la Sección 3).

En redes basadas en IEEE 802.15.4, se utiliza CSMA/CA para acceder al medio inalambrico. Este mecanismo utiliza el algoritmo BEB (Binary Exponential Backoff) para controlar el acceso de los nodos al canal mediante un intervalo aleatorio. Cuando un nodo se encuentra congestionado, se notifica a la MAC con el fin de que modifique el tamaño de la ventana de backoff para que sea menor con respecto a los nodos no congestionados. Especificamente se sugiere configurar el tamaño al 25 % del original ya que en promedio un nodo-sensor ganará acceso al canal solo después de que la mitad de sus nodos vecinos hallan transmitido [16].

Esta modificación al *backoff* se puede aplicar en redes que utilicen CSMA/CA para acceso al medio como IEEE 802.11 e IEEE 802.15.4 en sus modo super frame (Beacon Mode) y Non-Beacon-Mode. Para el primer caso funciona cuando el acceso al medio se encuentra en el momento de CAP (*Contention Access Period*) como se presenta en la Figura 3.2

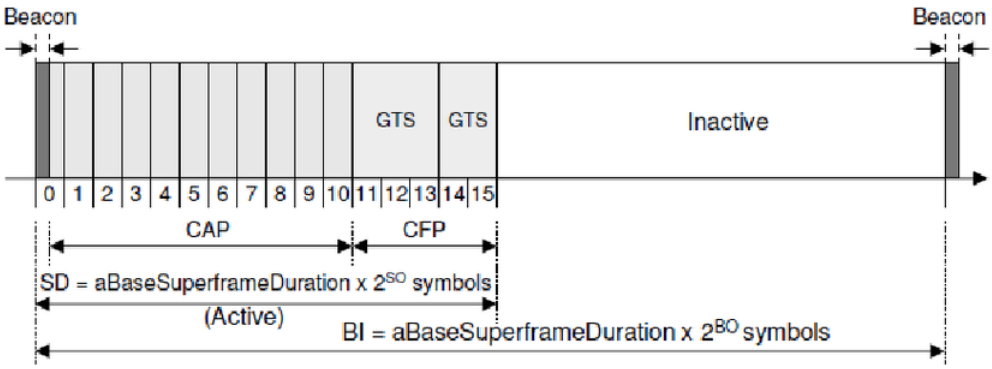


Figura 3.2: Estructura Super Frame en IEEE 802.15.4 Beacon Mode. Tomado de [27]

Este acceso prioritario al canal, le permite al nodo congestionado drenar con mayor rapidez los paquetes que se encuentren encolados en su *buffer*. Además que la latencia de los paquetes disminuye cuando se presenta congestión.

En el Algoritmo 1 se presenta el pseudo-código de las acciones realizadas por este módulo. En este se realiza el calculo de una variable llamada Congestión que tiene en cuenta la disponibilidad del nodo, el tamaño del buffer y la energía, si alguno de estos parámetros es 1 (o supera un umbral establecido por el altoritmo, Ver Sección 3.3) se considera que el nodo está congestionado.

Algorithm 1 Prioritized Channel Access

```
1: procedure WindowSize(D,B, E)
2:     Congestion = D or B or E. If some parameter is one, Congestion is one.
3:     while Congestion = 1 do
4:         Notify the MAC layer about congestion event.
5:         Decrease the Back-Off window to 25 % of the original size
6:     end while
7: end procedure
```

3.2. Administración del Buffer

Debido a los recursos limitados de los nodos en una red inalambrica de sensores, es necesario realizar uso de estos lo más eficientemente posible. El *buffer* del nodo es uno de los recursos que debe ser aprovechado de una forma optima ya que cuando alcanza su máxima utilización, inevitablemente el nodo entra en estado de congestión y es necesario descartar paquetes.

Hacer uso eficiente de este recurso en el nodo ha sido ampliamente investigado por la comunidad cientifica. Diferentes trabajos como [28], [29] y [30] han propuesto métodos muy interesantes de administración del *buffer* para WSN, sin embargo el propuesto en [10] además de presentar un metodo de doble *buffer* para tráfico generado por el nodo y tráfico enrutado, sugiere organizar los paquetes dentro de la cola mediante prioridades el cual es considerado para el método propuesto MLDC.

Como se especifica en la Sección 2.2.2, esta forma de administrar el *buffer* tiene en cuenta el gasto de recursos de los nodos para transportar paquetes en una red multi-saltos, dando mayor prioridad a estos paquetes. Este método permite dar mayor confiabilidad a los nodos alejados del *Sink* para que puedan transmitir su información al nodo central y que sus paquetes no se perderán en los múltiples saltos que se tengan que presentar.

Adicionalmente, MLDC sugiere que las prioridades estáticas de los paquetes generados sean como se presenta en la Tabla 3.1 y que la máxima prioridad dinámica en la red sea 5, es decir, cualquier paquete que alcance este umbral siempre se ubicará en la cabeza de la cola y será el primer paquete en ser servido por el nodo.

Tipo Paquete	**Prioridad**
Enrutamiento	3
Información	2
Otro	1

Cuadro 3.1: Prioridad paquetes MLDC

Estas prioridades estáticas permiten que el algoritmo de enrutamiento presentado en la Sección 3 pueda propagar sus actualizaciones de forma rápida por la red, tener convergencia de la información de enrutamiento con velocidad e informar el estado de la congestión a nodos vecinos.

Además para el manejo de los estados del *buffer*, se sugiere que se dividan así:

- Si la ocupación del *buffer* es inferior al 60 %, la cola se encuentra en estado Accept y ningún paquete se descarta
- Si la ocupación del *buffer* es inferior al 80 % y superior al 60 %, la cola se encuentra en estado Filter y solo paquetes de baja prioridad son descartados.

- Si la ocupación del *buffer* es superior al 80 %, la cola se encuentra en estado Reject y algunos paquetes de alta prioridad son descartados

Esta recomendación en la distribución de los estados difiere al utilizado en Ecoda ya que así se garantiza que se descartan menos paquetes conservando la característica preventiva de llenado del *buffer* debido a tráfico alto en la red.

En el Algoritmo 2 se presenta el pseudo-código con el detalle de las acciones que se realizan en este módulo. Se definen dos procedimientos, uno actúa a la llegada de cualquier paquete al buffer, el otro consiste en la programación del envío del siguiente paquete basado en las prioridades y en el turno correspondiente a cada sub-cola.

Algorithm 2 Buffer Administration

```
1: procedure Packet Arrival(Packet).
2:     Calculate packet dynamic priority.
3:     if The packet is generated by the node then
4:         Enqueue the packet in Local Packets Buffer.
5:     else
6:         Enqueue the packet in Transient Packets Buffer.
7:     end if
8: end procedure
9: procedure Packet Departure(Packet).
10:    Sort Local and Transient Buffers by dynamic priority.
11:    Perform Round-Robin mechanism to choose wich packet to send.
12: end procedure
```

3.3. Enrutamiento

En [31] se presenta un estudio del estado del arte de protocolos de enrutamiento diseñados para WSN, muchos de estos están diseñados para ser tolerantes a fallas, auto-configurables, conservar y disminuir el consumo de energía, creación de clusters entre otros; sin embargo al igual que protocolos bien conocidos como AODV (ZigBee) y DSDV, no ofrecen cambios en el enrutamiento cuando hay congestión en algún punto de la red.

En [32] y [33], los autores presentan protocolos de enrutamiento que fueron diseñados para actualizarse dependiendo de la congestión presente en los nodos. Estos buscan mitigar la congestión mediante la limitación de las tasas de transmisión. Sin embargo MLDC considera un método de enrutamiento basado en el presentado [11] el cual realiza la selección de rutas para enrutamiento basado en información de los vecinos como ocupación del *buffer*, energia y disponibilidad. Este método es descrito en la Sección 2.2.1.

MLDC no hace uso de los paquetes *Connect* y *ACK* ya que estos aumentan la carga en la red en el momento de la creación inicial de la topología. En la etapa de Setup el nodo *Sink* empieza la transmisión de un paquete tipo broadcast (Paquete Hello de DALPAS) para que se propague por la red con el fin de que cada nodo construya su tabla de vecinos como se presenta en la Figura 2.3.

Al igual que DALPAS, MLDC implementa el esquema de control de topología mediante intercambio de paquetes pequeños que indican cuando hay un cambio en alguna de las variables de interes para la congestión. Cuando un nodo tiene dos o más padres con iguales características, balancea la carga entre estos con el fin de evitar congestionar una ruta.

En cuanto al esquema *Soft Stage*, MLDC no lo implementa ya que el *buffer* propuesto de los nodos puede mantener diferentes flujos de manera justa, además que esta técnica aumenta considerablemente

la carga en la red sin realizar un control de congestión efectivo.

Este algoritmo de enrutamiento está diseñado para WSN donde la comunicación es de tipo *Many-to-One* y nodos de las mismas características no intercambian información de aplicación, esto hace que se cree una topología tipo árbol y que las tablas de vecinos sean más pequeñas en comparación a las sugeridas en ZigBee o DSDV.

En el Algoritmo 3 se presenta el pseudo-código con el detalle de las actividades realizadas por el protocolo de enrutamiento diseñado. Este consta de 3 procedimientos que se ejecutan en la capa de red del nodo. El primer procedimiento se ejecuta una sola vez en el momento de la inicialización de la red, los restantes son ejecutados para mantener la topología y seleccionar la ruta por la cual se enviará el paquete.

Algorithm 3 Routing Algorithm

```
1: procedure Topology Creation.
2:     Sink Node send a brodcast packet.
3:     while Node receive a broadcast packet do Each node will send one broadcast packet
4:         Update the neighbor table
5:         Send one brodcast packet with the own congestion information.
6:     end while
7: end procedure
8: procedure Topology Control.
9:     if Any change in Bufffer, Energy or Availability then
10:        Send a packet to the neighbors about the congestion update.
11:    end if
12: end procedure
13: procedure Route Desision.
14:    Find the best route to the Sink node in the neighbor table.
15:    if There are more than one route then
16:        Perform Round-Robin Algorithm for the routes.
17:    else
18:        Send a packet via the route.
19:    end if
20: end procedure
```

3.4. Aplicación MLDC

Las capas de acceso al medio, manejo de *buffer* y red pueden controlar y mitigar la congestión de manera eficiente, sin embargo para tener una red donde la congestión sea un fenómeno esporádico, la capa de aplicación debe ser consciente del estado de los nodos y la red con el fin de no cargar de información y paquetes que posiblemente sean descartados. Para esto MLDC sugiere que la aplicación pueda revisar el estado del nodo con el fin de poder disminuir o aumentar su tasa de generación de paquetes basado en la información que le suministren sus capas inferiores. La aplicación debe generar paquetes a una tasa igual o inferior a la tasa de servicio con el fin que la utilización nunca supere uno como se presenta en la Ecuación 1.1.

Para lograr esto, MLDC sugiere que la capa de aplicación implemente alguno de los siguientes métodos:

- Generación de información cuando existan recursos para la transmisión. La aplicación debe revi-

sar si no hay congestión en el nodo, tomar los datos de sensores y luego transmitirlo. No hacerlo de esta forma puede ocasionar que se genere información que va a ser rechazada por las capas inferiores.

- Implementar técnicas de limitación de la tasa de transmisión y generación de datos basadas en el estado de la congestión.
- Control tipo AIMD como se presenta en la Ecuación 1.2

La capa de red retroalimenta a la capa de aplicación mediante una variable de estado, la cual puede ser consultada por la capa superior cuando se requiera realizar una transmisión, generar información, o tomar alguna decisión con respecto a comunicación.

Capítulo 4

Simulación y pruebas

4.1. Parámetros de simulación

Para realizar la evaluación en NS2 de los algoritmos de control de congestión se configuraron los siguientes parámetros generales:

Canal: *WirelessChannel* (definida en ns/mac/channel.h). Modela el medio físico compartido.

Propagación: *Propagation/Shadowing* (definida en ns/mobile/shadowing.h). Describe el modelo de propagación sobre el canal inalámbrico, el cual consta de un modelo de Path Loss que predice la potencia media recibida a una distancia especifica, y el modelo de shadowing que refleja la variación de la potencia mediante una variable aleatoria log-normal. Este modelo también tiene en cuenta una variable aleatoria debido a fenómenos de Fading.

Interfaz de red: *phy/wireless/phy/802 15 4* (definida en ns/wpan/p802.15.4phy.h). Se utiliza la capa física IEEE 802.15.4. Detalles en Sección 2.1.

MAC: *mac/802 15 4* (definida en ns/wpan/p802.15.4mac.h). Se utiliza el método de control de acceso al medio IEEE 802.15.4. Detalles en Sección 2.1.

Tipo de Buffer: La definición y configuración depende del algoritmo de control de congestión a utilizar. En la simulación sin método de control de congestión se utiliza una cola estándar tipo *Queue/DropTail* que implementa un *buffer* FIFO con admisión drop-on-overflow.

Link Layer: *LL*(definida en ns/mac/LL.h). Este objeto especifica características como retardo de propagación, fragmentación de paquetes, y ancho de banda del enlace.

Antena: *Antenna/omniAntenna*(definida en ns/mobile/omniAntenna.h). Antena con patrón de radiación horizontal de 360 grados.

Tamaño del Buffer: Cada nodo tiene capacidad para encolar 20 paquetes.

Enrutamiento: La definición y configuración depende del algoritmo de control de congestión a utilizar. En la simulación sin método de control de congestión se utiliza AODV el cual es utilizado en redes ZigBee.

Tráfico: *Application/Traffic/Exponential*. Este objeto modela tráfico tipo On/Off. Durante el tiempo de On, el nodo genera tráfico tipo CBR (Constant Bit Rate) y durante el tiempo de Off no se

genera tráfico. Los dos tiempos siguen una distribución exponencial con una media que puede ser especificada en el simulador. Para la realización de la evaluación generan trafico el 25 % de la totalidad de los nodos, el restante puede ser coordinador de la red.

Modelo de energía: Se hace uso del modelo definido en ns/mobile/energyModel.h. A este modelo se le especifica la potencia de transmisión, sensado, recepción entre otros.

Cantidad de nodos: Se configuran 101 nodos: 1 nodo tipo "IEEE 802.15.4 PAN Coordinator" el cual es el nodo *Sink* de la red, 100 nodos de tipo "Coordinator" lo cuales funcionan como dispositivos finales y además enrutan paquetes.

Distribución de nodos: Los nodos son uniformemente distribuidos de forma aleatoria en el terreno acotado. El nodo *Sink* siempre está ubicado en el centro del terreno.

Topología de red: Se consideran tres topologías de red: Estrella, Árbol y Malla. Para la generación de estas se tuvo en cuenta la configuración de los nodos en NS2 acorde a lo descrito en la sección 2.3.2.

4.2. Simulación ECODA

ECODA define mecanismos para manejo del buffer de los nodos, limitación de la tasa de envío de paquetes y verificación del estado de las rutas. Para esto en NS2 se implementaron los módulos presentados en la Figura 4.1

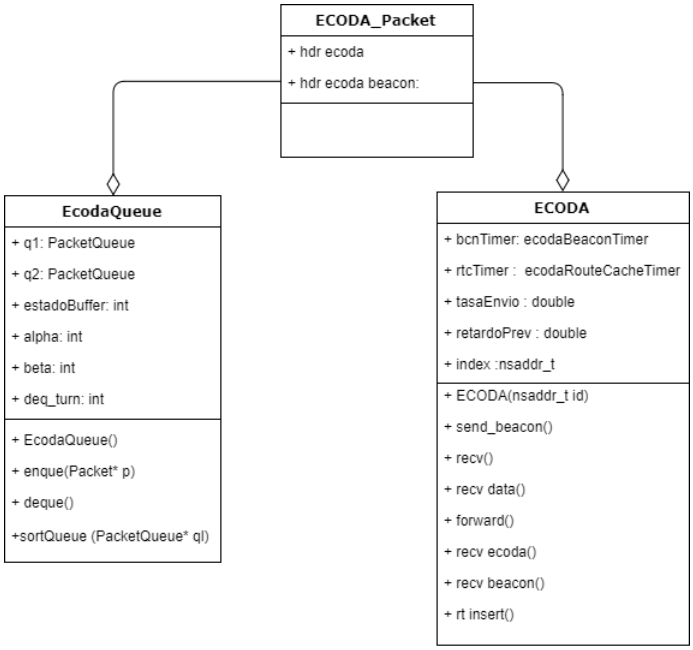


Figura 4.1: Modelo de clases Ecoda

El detalle de los módulos implementados se presentan a continuación:

1. ECODA_Packet: Debido a que es necesario tener campos específicos para transmitir información del protocolo, se crearon estructuras de datos que representan las cabeceras de ECODA y auxiliares:

a) hdr_ecoda: Se define un entero sin signo de 8 bits que representa el tipo de paquete que se está enviando. Además de la variable offset_ la cual NS2 exige en la implementación de un nuevo modelo de paquete con el fin de especificar donde esta cabecera reside dentro de la cabecera principal. Además se especifican los métodos de acceso a la información de la cabecera personalizada.

b) hdr_ecoda_beacon: Es un tipo de paquete especial el cual es generado por el nodo *Sink* con el fin de propagarse por toda la red y así descubrirse todos los dispositivos y actualizar sus tablas de enrutamiento. Posee un timestamp el cual es utilizado para calcular el *delay* y el estado de los caminos al nodo principal conforme a lo especificado por ECODA. Cada paquete beacon posee los atributos: tipo de paquete, cantidad de saltos que ha dado el paquete, id del beacon y dirección origen del generador del beacon.

2. ECODA_Queue: Este modulo implementa todas las acciones que realiza ECODA a nivel de manejo de *buffer*, politica de admisión de paquetes, clasificación de los paquetes y definición del siguiente paquete a enviar.

   a) Atributos:

   - Dos colas tipo *PacketQueue* que representan el *buffer* del nodo segun lo especificado por el algoritmo. Una cola almacena los paquetes generados por el nodo y la otra paquetes que fueron generados por otros nodos peor están siendo enrutados por el analizado.
   - Un entero representa el estado del *buffer* dependiendo que dependiendo de la congestión puede ser: Accept State, Filter State o Reject State .
   - Los parámetros α y β los cuales son utilizados para calcular la prioridad dinámica según la Ecuación 2.2.2.
   - Un entero que permite implementar el algoritmo round robin para escoger de que sub-cola se enviará el siguiente paquete.

   b) Métodos:

   - Constructor: Recibe como parámetro el ID del nodo. Inicializa las dos sub-colas, el primer turno de envío en round robin, el estado del *buffer* y los parámetros α y β.
   - enqueue(): Accede a las cabeceras de los paquetes para identificar el nodo origen del paquete y así poder ubicarlo en la sub-cola respectiva. Verifica el estado actual del *buffer* y configura la variable entera que representa el estado del mismo, realiza el calculo de las prioridades estáticas (si el origen es el nodo) o actualiza la prioridad dinámica para paquetes enrutados. Además es el encargado de implementar la política de admisión de tráfico dependiendo del estado del *buffer* en la cual algunos paquetes pueden ser removidos.
   - sortQueue(): Encargado de ordenar la cola que se pase como parámetro basado en sus prioridades dinámicas según lo especificado por el algoritmo.
   - dequeue(): Esta función es llamada cuando se requiere transmitir un paquete que está en el buffer y las condiciones de la red son las adecuadas para la transmisión. Primero invoca al método sortQueue() para cada una de las dos sub-colas y luego implementa el algoritmo round-robin para seleccionar el siguiente paquete a enviar.

3. ECODA_router: Este modulo define todas las operaciones que se realizan en la capa de red especificadas por ECODA. Se implementan dos Timers para envío periódico de beacon y otro que funciona para definir tiempos de expiración de rutas. Dos clases son implementadas: RouteCache la cual contiene atributos relacionados a la información relevante que debe tener un nodo con respecto a sus rutas como el tiempo de expiración, el numero de saltos, estado, siguiente salto e ID de la ruta, y ECODA la cual tiene las siguientes características:

a) Atributos:

- Instancia de timers ecodaBeaconTimer y ecodaRouteCacheTime.
- Variable de tipo ecoda_rtcache el cual contiene la tabla de enrutamiento.
- Tasa de envío: controla el momento en el cual se libera el paquete para sus capas inferiores basado en el estado actual de la congestión.
- Retardo previo: Es una variable auxiliar que almacena el retardo en una ruta especifica para identificar cuellos de botella.
- Dirección del nodo.

b) Métodos:

- Constructor: Recibe como parámetro el ID del nodo. Crea e inicia los dos Timers de la clase, se crea la tabla de enrutamiento y la tasa de envío inicial.
- send_beacon(): Se crea un paquete de tipo HDR_ECODA_BEACON(), se llenan todas sus cabeceras incluidas las direcciones origen y Broadcast, se agrega un timestamp y se hace la programación del envio del paquete.
- recv(): Se invoca cuando la capa de red recibe un paquete, se analizan las cabeceras para identificar el tipo de paquete y hace un llamado al método recv_ecoda() cuando el paquete es de tipo ECODA o recv_data() cuando el paquete es de información de tráfico de aplicación.
- recv_data(): Revisa si el destino del paquete es el nodo, si esto sucede lo envía a las capas superiores, si no, lo enruta mediante el método forward().
- forward(): Se verifica el TTL del paquete para evitar loops en la red y programa la transmision del paquete a su siguiente salto con base en la tabla de enrutamiento.
- recv_ecoda(): Invoca al método recv_beacon() cuando el paquete llegó al modulo correctamente.
- recv_beacon(): Con base en el timestamp del paquete se calcula el retardo del mismo al llegar al nodo, con base en esto se especifica la tasa de envío con base en la especificación del algoritmo que es de tipo AIMD. Revisa y actualiza la tabla de enrutamiento si es necesario invocando al método rt_insert().
- rt_insert(): Actualiza la tabla de enrutamiento con base a la información nueva del beacon.

El detalle del código implementado se encuentra en el Anexo 1.

4.3. Simulación DAlPaS

DALPAS define mecanismos para manejo la topología de red para evitar y controlar congestión. Para esto en NS2 se implementaron los siguientes módulos presentados en la diagrama UML de la Figura 4.2. El detalle del código implementado se encuentra en el Anexo 2.

Cada uno de estos módulos fueron implementados en NS2 con las siguientes funciones:

1. DALPAS_Packet: Este módulo especifica estructuras de datos que son utilizadas como cabeceras de los paquetes propios a transmitir en DALPAS:

   a) hdr_daipas: Se define un entero sin signo de 8 bits que representa el tipo de paquete que se está enviando. Además de la variable offset_ la cual NS2 exige en la implementación de un nuevo modelo de paquete con el fin de especificar donde esta cabecera reside dentro de la cabecera principal. Además se especifican los métodos de acceso a la información de la cabecera personalizada.

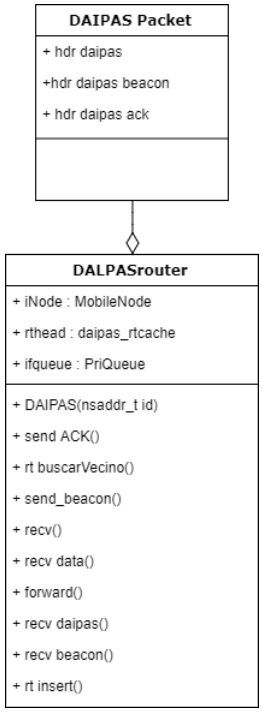


Figura 4.2: Modelo de clases DAlPas

b) hdr_daipas_beacon: Esta cabecera representa el paquete HELLO especificado en el algoritmo DAIPAS, es generado en la fase de setup por el nodo *Sink*. En esta estructura se define un entero sin signo de 8 bits que representa el tipo de paquete, una variable que representa el timestamp del paquete, la dirección de nodo origen, los niveles de utilización del *buffer* y energía disponible, el nivel del nodo en la red y una bandera tipo bool que representa la disponibilidad del nodo.

c) hdr_daipas_ack: Esta cabecera contiene los mismos atributos de la estructura anterior además de la variable tipo bool nextPacket que es utilizado para indicar si el nodo que genera el paquete se encuentra en estado Soft Stage.

d) hdr_daipas_connect: Esta cabecera contiene los mismos atributos de hdr_daipas_beacon

2. DAIPASrouter: Este modulo define todas las operaciones que se realizan en la capa de red especificadas por DAIPAS. Dos clases son implementadas: RouteCache la cual contiene atributos relacionados a la información relevante que debe tener un nodo con respecto a sus vecinos como ocupación del *buffer*, energía disponible, nivel, disponible y prioridad, y DAIPAS la cual tiene las siguientes características:

   a) Atributos:

      - Instancia de MobileNode: Se crea una instancia al nodo en el cual se puede tener características del nodo además de extraer el nivel de energía del mismo.
      - Variable de tipo daipas_rtcache el cual contiene la tabla de vecinos del nodo.
      - Instancia de PriQueue la cual es utilizada para extraer información del tamaño del *buffer*.

   b) Métodos:

- Constructor: Recibe como parámetro el ID del nodo, se inicia la matriz de estadísticas de vecinos la cual contiene el histórico de la cantidad de paquetes que han llegado de cada vecino, la tabla de vecinos que contiene toda la información de enrutamiento, variable que controla el algoritmo Round Robin y el nivel inicial del nodo.
- send_beacon(): Cumple las funciones del paquete Hello especificado en DAIPAS. Inicialmente mide el nivel de energía actual y la utilización del buffer del nodo para crear un paquete tipo hdr_daipas. Este paquete es tipo Broadcast e inicialmente es enviado por el nodo *Sink* para construir la topología en árbol. especificada en el algoritmo.
- recv(): Se invoca cuando la capa de red recibe un paquete, se analizan las cabeceras para identificar el tipo de paquete y hace un llamado al método recv_daipas() cuando el paquete es de tipo DALPAS o recv_data() cuando el paquete es de información de tráfico de aplicación.
- recv_daipas(): Identifica el tipo de paquete DAIPAS que llegó a la capa de red. Invoca al método recv_beacon() cuando el paquete DALPAS HELLO llegó al modulo correctamente, o recv_ack() si el paquete es DALPAS_ACK-
- recv_beacon(): Se extraen las cabeceras del paquete, se revisa si el origen del paquete recibido se encuentra dentro de la tabla de vecinos, si no se invoca el método rt_insert() el cual agrega una nueva entrada en la tabla. Una vez se agrega satisfactoriamente el vecino, de envía un nuevo beacon (paquete Hello en DALPAS) para continuar con la creación de la topología según la fase setup del algoritmo.
- send_ACK(): Se crea un paquete de tipo hdr_daipas_ack con sus cabeceras común e IP. Se llenan todos los campos de la cabeceras además del envío de la información relacionada al estado del soft stage, utilización del *buffer*, nivel, disponibilidad del nodo y energía restante. Este método es invocado cuando es necesario informar a los vecinos sobre algún evento relacionado a la congestión del nodo.
- recv_data(): Este método es invocado cuando llega un paquete de datos al nodo, se extraen las cabeceras, se realiza la búsqueda de la mejor ruta para enviar el paquete mediante el método rt_buscarVecino() y se realiza el forward del paquete.
- rt_buscarVecino(): Este método es muy importante debido a que controla la topología de acuerdo al algoritmo DALPAS. Al inicio se cuentan los nodos vecinos con menor nivel recorriendo la tabla de enrutamiento, esto sirve para seleccionar que vecinos pueden recibir el paquete a transmitir. Se actualizan las estadísticas de cantidad de paquetes que están llegando al nodo lo cual sirve para decidir si hay más de dos flujos activos y entrar en modo soft stage. Si el nodo se encuentra en este estado, se decide a que nodo hay que seguir sirviendo y al resto de los nodos hijo enviarles un paquete ACK sugiriendo buscar un nuevo nodo. Después de este proceso se analiza si el nodo esta en estado hard stage, si alguna condición se cumple se genera un ACK para todos los vecinos indicando que el nodo ya no se encuentra disponible, por lo que obliga a los vecinos a modificar su tabla de enrutamiento y buscar rutas alternativas. Si el nodo no se encuentra en estado hard o soft stage, se utiliza método de balanceo Round Robin entre sus nodos menores para transmitir los paquetes. Este método implementa la mayor parte del diagrama de flujo presentado en la Figura 4.3
- rt_insert(): Se recibe como parámetros la dirección del vecino, el tamaño del *buffer*, el porcentaje de energía disponible, el nivel y una bandera de disponibilidad del nodo vecino. Con esta información se crea una nueva ruta al vecino en la tabla de enrutamiento con la información suministrada.
- rt_lookup(): Se recibe como parametro una dirección destino, se recorre toda la tabla de enrutamiento, si se encuentra se retorna esta ruta, si no, se retorna un NULL indicando que falló la búsqueda.

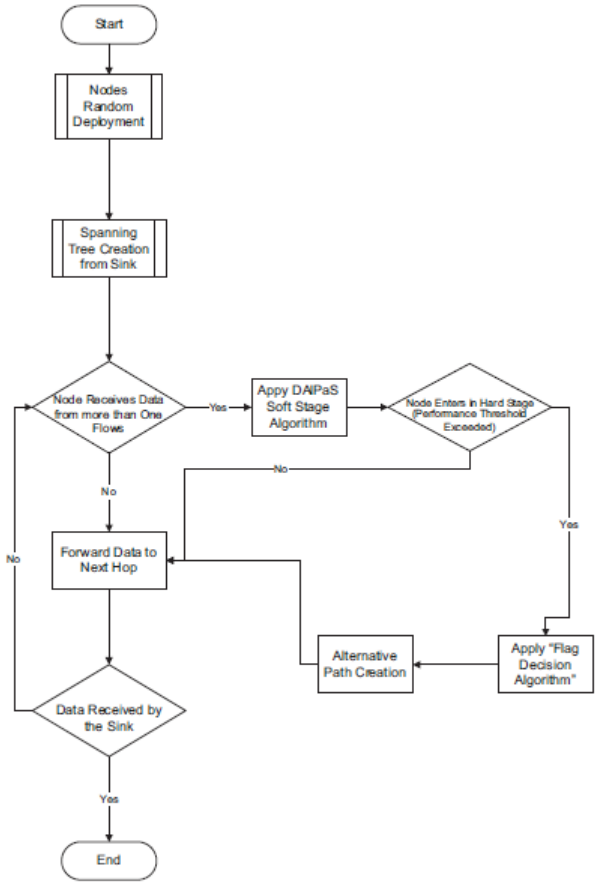


Figura 4.3: Diagrama de Flujo DALPAS. Tomado de [11]

4.4. Simulación FUSION

El algoritmo FUSION combina 4 diferentes técnicas para mitigar la congestión que se presenta en una WSN. La implementación de este algoritmo en NS2 tiene en cuenta módulos presentados en la Figura 4.4.

1. FUSION: Este es el modulo principal del algoritmo en donde se definen todas las funcionalidades de la capa de red del algoritmo.

   Se implementa el protocolo de enrutamiento DSDV (Destination-Sequenced Distance Vector Routing) el cual es el implementado por los autores del algoritmo. Para el desarrollo de este protocolo se crearon las siguientes funciones:

   - Clase RouteCache: En esta clase se definen todos los datos e información relacionada a las rutas. El conjunto de estas clases forman la tabla de enrutamiento del nodo. DSDV en su tabla de rutas tiene: Destino, siguiente salto, numero de saltos, numero de secuencia y el tiempo en que fue instalada la ruta.
   - rt_insert: Este método permite añadir una nueva entrada a la tabla de enrutamiento del nodo.

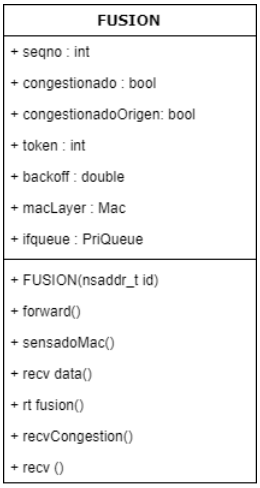


Figura 4.4: Modelo de clases FUSION

- rt_lookup: Dada una dirección destino, este método busca la mejor ruta al destino.
- fusionRouteCacheTimer: Timer que tiene la funcionalidad de indicar al nodo cada periodo en que una entrada de la tabla de enrutamiento ha caducado.

Además de las funcionalidades de DSDV, se implementaron los siguientes atributos y métodos en el modulo:

a) Atributos:
   - seqno: Esta variable entera siempre tiene valor par según lo especificado en DSDV. Es utilizada en el momento de enviar información sobre la tabla de enrutamiento.
   - congestionado: Esta variable booleana indica al control tipo Hop-by-hop que el nodo se encuentra congestionado.
   - congestionadoOrigen: Variable booleana que indica ala la aplicación instalada en el nodo que genera la información que el nodo se encuentra congestionado y es necesario ajustar su tasa de envio de datos.
   - fusionSensadoMACTimer: Timer que especifica los tiempos en los que el nodo realiza el sensado del canal con el fin de calcular la carga del mismo y detectar si hay congestión basado en estas medidas.
   - token: Variable entera que indica la cantidad de tokens disponibles para implementar el método de limitación de la tasa de envio mediante *token bucket scheme*.
   - backoff: Esta variable indica a la capa MAC el tiempo promedio de backoff que debe ser implementado cuando el nodo esta congestionado.
   - macLayer: Instancia de la clase MAC, en la cual se realizan las mediciones del estado del canal para determinar la utilización del mismo.
   - ifqueue: Instancia al *buffer* del nodo.

b) Métodos:
   - Constructor: Es el encargado de crear la instancia del objeto Fusion en el nodo. Inicializa las variables: seqno, tabla de enrutamiento, buffer, backoff, token, dirección del nodo padre, entre otros.

- recv: Se encarga de extraer las cabeceras del paquete que llega a la capa de red. Cuando recibe un paquete del nodo padre, se aumenta la variable token de acuerdo al esquema planteado en el algoritmo. Además, se revisa el estado actual del *buffer* y se invoca el método sensadoMac() para que se calcule la utilización del canal con el fin de determinar si el nodo está congestionado y que tipo de control se va a ejercer. Si el nodo está congestionado se modifican los tiempos de backoff para tener acceso prioritario al canal, se configura el bit congestión en las cabeceras para informar a los vecinos y se le informa a la aplicación que el nodo está congestionado.
  Este método también se encarga de analizar que tipo de paquete llegó. Si es paquete de datos se invoca al método recv_data, si es paquete de enrutamiento recv_fusion y si se recibió el paquete con un bit de congestión del vecino recv_congestion.
- sensadoMac(): Tiene como función realizar el muestreo del canal 4 veces cada 100 milisegundos, promediar los datos y utilizar EWMA con parámetro α=0.85 para retornar la utilización del canal.
- recv_data: Se extraen las cabeceras del paquete, se revisa que el destino sea el nodo actual o que se encuentre en la tabla de enrutamiento. Además se aplica el esquema *Token bucket* para regular la tasa de transmisión. Si existe la tabla de enrutamiento y existen tokens para transmisión del paquete se invoca el método forward(), en caso contrario se descarta el paquete.
- recv_fusion: Este método identifica el tipo de paquete Fusion para invocar el método apropiado para manejar el paquete. Si se recibe un paquete con el bit de congestión activo, recv_congestion() se encarga de manejarlo, por otro lado si se recibe un Beacon, se hace el llamado a recv_beacon() para que se actualice la tabla de enrutamiento del nodo.
- recv_congestion(): Es el encargado de actualizar la tabla de enrutamiento cuando los nodos padre entran o salen de congestión.

2. MacLayer: El algoritmo Fusion [16] sugiere cambios en la capa de acceso al medio en los cuales se cambian los tiempos de backoff cuando un nodo se encuentra congestionado con el fin de tener acceso prioritario al medio compartido. Para esto se modificaron atributos de esta clase para permitir que la capa de red pudiera indicarle a la MAC cuando se requiere tener acceso prioritario. Además el algoritmo Fusion propone no utilizar paquetes de control tipo RTS y CTS, sin embargo estos paquetes son generados en redes Wi-Fi IEEE 802.11 y en el presente se consideran redes IEEE 802.15.4 que no utiliza estos paquetes, por este motivo esos cambios sugeridos no fueron necesarios incluirlos.

El detalle del código implementado en NS2 se encuentra en el Anexo 3.

4.5. Simulación MLDC

MLDC es un método de control de congestión que actúa distribuidamente en la red y en múltiples capas de comunicación de los nodos. El código implementado en NS2 se encuentra en el Anexo 4.

Los módulos desarrollos son los descritos a continuación y presentados en la Figura 4.5 :

1. MLDCPacket: Este módulo especifica estructuras de datos que son utilizadas como cabeceras de los paquetes propios a transmitir en MLDC:

   a) hdr_mldc: Se define un entero sin signo de 8 bits que representa el tipo de paquete que se está enviando. Además de la variable offset_ la cual NS2 exige en la implementación de un nuevo modelo de paquete con el fin de especificar donde esta cabecera reside dentro de la cabecera principal. Además se especifican los métodos de acceso a la información de la cabecera personalizada.

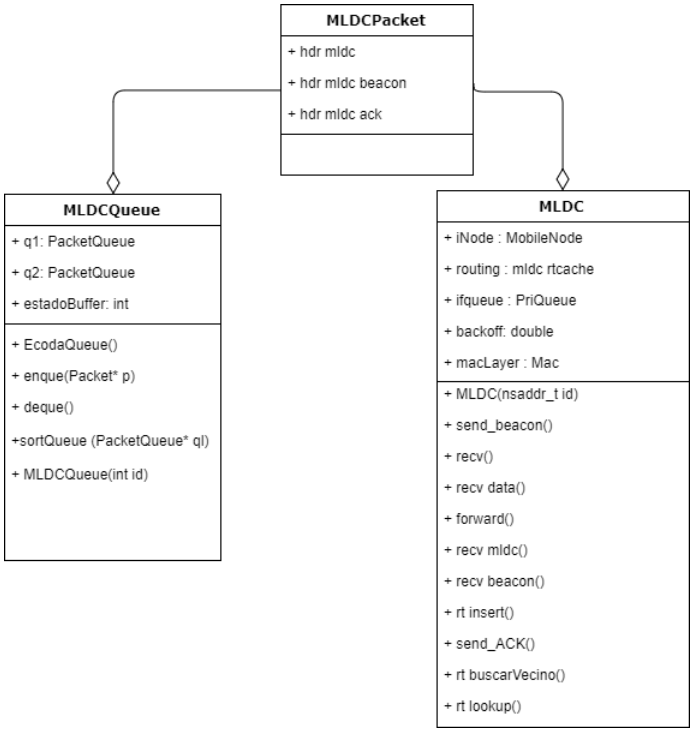


Figura 4.5: Modelo de clases MLDC

b) hdr_mldc_beacon: Esta cabecera representa el paquete inicial intercambiado en la etapa de configuración de la red. En esta estructura se define un entero sin signo de 8 bits que representa el tipo de paquete, una variable que representa el timestamp del paquete, la dirección de nodo origen, los niveles de utilización del *buffer* y energía disponible, el nivel del nodo en la red y una bandera tipo bool que representa la disponibilidad del nodo.

c) hdr_mldc_ack: Esta cabecera contiene los mismos atributos de la estructura anterior. Es utilizado cuando se desea dar retroalimentación a los vecinos del nodo cuando se presenta congestión.

2. MLDCQueue: Este modulo implementa todas las acciones que realiza MLDC a nivel de buffer, política de admisión de paquetes, clasificación de los paquetes y definición del siguiente paquete a enviar.

a) Atributos:

- Dos colas tipo *PacketQueue* que representan el *buffer* del nodo según lo especificado por el algoritmo. Una cola almacena los paquetes generados por el nodo y la otra paquetes que fueron generados por otros nodos peor están siendo enrutados por el analizado.
- Un entero representa el estado del *buffer* dependiendo que dependiendo de la congestión puede ser: Accept State, Filter State o Reject State.
- Un entero que permite implementar el algoritmo Round Robin para escoger de que sub-cola se enviará el siguiente paquete.

b) Métodos:

- Constructor: Recibe como parámetro el ID del nodo. Inicializa las dos sub-colas, el primer turno de envío en round robin, el estado del *buffer* y los parámetros α y β.
- enqueue(): Accede a las cabeceras de los paquetes para identificar el nodo origen del paquete y así poder ubicarlo en la sub-cola respectiva. Verifica el estado actual del *buffer* y configura la variable entera que representa el estado del mismo, realiza el calculo de las prioridades estáticas (si el origen es el nodo) o actualiza la prioridad dinámica para paquetes enrutados. Además es el encargado de implementar la política de admisión de tráfico dependiendo del estado del *buffer* en la cual algunos paquetes pueden ser removidos.
- sortQueue(): Encargado de ordenar la cola que se pase como parámetro basado en sus prioridades dinámicas según lo especificado por el algoritmo.
- dequeue(): Esta función es llamada cuando se requiere transmitir un paquete que está en el *buffer* y las condiciones de la red son las adecuadas para la transmisión. Primero invoca al método sortQueue() para cada una de las dos sub-colas y luego implementa el algoritmo round-robin para seleccionar el siguiente paquete a enviar.

3. MLDCRouter: Este modulo define todas las operaciones que se realizan en la capa de red especificadas en MLDC. Dos clases son implementadas: RouteCache la cual contiene atributos relacionados a la información relevante que debe tener un nodo con respecto a sus vecinos como ocupación del buffer, energía disponible, nivel, disponible y prioridad, y MLDC la cual tiene las siguientes características:

   a) Atributos:
   - Instancia de MobileNode: Se crea una instancia al nodo en el cual se puede tener características del nodo además de extraer el nivel de energía del mismo.
   - Variable de tipo mldc_rtcache el cual contiene la tabla de vecinos del nodo.
   - Instancia de PriQueue la cual es utilizada para extraer información del tamaño del *buffer*.
   - backoff: Esta variable indica a la capa MAC el tiempo promedio de backoff que debe ser implementado cuando el nodo esta congestionado.
   - macLayer: Instancia de la clase MAC, en la cual se realizan las mediciones del estado del canal para determinar la utilización del mismo

   b) Métodos:
   - Constructor: Recibe como parámetro el ID del nodo, se inicia la tabla de vecinos que contiene toda la información de enrutamiento, una variable que controla el algoritmo Round Robin y el nivel inicial del nodo.
   - send_beacon(): Inicialmente mide el nivel de energía actual y la utilización del *buffer* del nodo para crear un paquete tipo hdr_mldc. Este paquete es tipo Broadcast e inicialmente es enviado por el nodo *Sink* para construir la topología en árbol. especificada en el algoritmo.
   - recv(): Se invoca cuando la capa de red recibe un paquete, se analizan las cabeceras para identificar el tipo de paquete y hace un llamado al método recv_mldc() cuando el paquete es de tipo MLDC o recv_data() cuando el paquete es de información de tráfico de aplicación. Además en este método se realiza la acción de notificación del estado del stack de congestión a la capa de aplicación superior.
   - recv_mldc(): Identifica el tipo de paquete DALPAS que llegó a la capa de red. Invoca al método recv_beacon() cuando el paquete de construcción de topología llegó al modulo correctamente, o recv_ack() si el paquete es DALPAS_ACK
   - recv_beacon(): Se extraen las cabeceras del paquete, se revisa si el origen del paquete recibido se encuentra dentro de la tabla de vecinos, si no se invoca el método rt_insert() el cual agrega

una nueva entrada en la tabla. Una vez se agrega satisfactoriamente el vecino, de envía un nuevo beacon (paquete Hello en DALPAS) para continuar con la creación de la topología según la fase setup del algoritmo.

- send_ACK(): Se crea un paquete de tipo hdr_daipas_ack con sus cabeceras común e IP. Se llenan todos los campos de la cabeceras además del envío de la información relacionada al estado del soft stage, utilización del *buffer*, nivel, disponibilidad del nodo y energía restante. Este método es invocado cuando es necesario informar a los vecinos sobre algún evento relacionado a la congestión del nodo.
- recv_data(): Este método es invocado cuando llega un paquete de datos al nodo, se extraen las cabeceras, se realiza la búsqueda de la mejor ruta para enviar el paquete mediante el método rt_buscarVecino() y se realiza el forward del paquete.
- rt_buscarVecino(): Este método es muy importante debido a que controla la topología de acuerdo al algoritmo MLDC. Al inicio se cuentan los nodos vecinos con menor nivel recorriendo la tabla de enrutamiento, esto sirve para seleccionar que vecinos pueden recibir el paquete a transmitir. En este proceso se analiza si el nodo esta en estado congestionado (hard stage de DALPAS), si alguna condición se cumple se genera un ACK para todos los vecinos indicando que el nodo ya no se encuentra disponible, por lo que obliga a los vecinos a modificar su tabla de enrutamiento y buscar rutas alternativas. Si el nodo no se encuentra en estado hard stage, se utiliza método de balanceo Round Robin entre sus nodos menores para transmitir los paquetes; si por el contrario el nodo se encuentra congestionado, la capa de red le informa a la capa MAC que modifique el tiempo de backoff para tener prioridad en el acceso al canal.
- rt_insert(): Se recibe como parámetros la dirección del vecino, el tamaño del buffer, el porcentaje de energía disponible, el nivel y una bandera de disponibilidad del nodo vecino. Con esta información se crea una nueva ruta al vecino en la tabla de enrutamiento con la información suministrada.
- rt_lookup(): Se recibe como parámetro una dirección destino, se recorre toda la tabla de enrutamiento, si se encuentra se retorna esta ruta, si no, se retorna un NULL indicando que falló la búsqueda.

Capítulo 5

Evaluación de resultados

Para realizar la evaluación cuantitativa del comportamiento de los métodos de control de congestión con respecto al tráfico generado y la topología usada, es necesario definir las métricas de calidad de servicio y definir un procedimiento claro para su medición. Cada autor de los algoritmos analizados ha propuesto sus propias métricas que se presentan en la tabla 5.1. Para realizar un análisis comparativo en el presente trabajo se proponen las siguientes métricas de QoS:

- Throughput promedio: Indica la tasa de transmisión promedio de los paquetes en la red. Para la evaluación de este parámetro se realizó la medida las tasa es de transporte de datos de todos los paquetes que lleguen al nodo *Sink* sin errores entre el tiempo de la simulación. Sean X el Throughput medido en el nodo Sink, tmax y tmin el tiempo de llegada del ultimo paquete recibido y el tiempo de llegada del primer paquete respectivamente y Pz el tamaño del paquete recibido, esta metrica se mide como se presenta en la Ecuación 5.1.

$$X = \frac{\sum Pz}{tmax - tmin} \tag{5.1}$$

- Fairness: Esta métrica es usada para determinar si múltiples usuarios (en WSN: nodos), están recibiendo recursos equitativos entre ellos. El índice *Jain's Fairness* califica la justicia en una red con base en el Throughput de los paquetes que llegan al nodo *Sink*. [34]. Para realizar el calculo de esta métrica se utilizó la Ecuación 5.2.

$$J(X) = \frac{(\sum_{i=1}^{N} Xi)^2}{N \cdot \sum_{i=1}^{N} Xi^2} \tag{5.2}$$

- Retardo promedio: Esta métrica indica como los algoritmos pueden manejar interferencias en la red, tiempos de procesamiento y de transmisión desde el momento que se generó el paquete hasta que llega al nodo Coordinador de la red. Para realizar la medida, se calcula el retardo de cada paquete que llegue al nodo *Sink* sin errores, luego se calcula el promedio de estos datos. En la Ecuación 5.3 se presenta la forma de calcular el retardo el cual es un promedio de la diferencia de tiempos entre la llegada de los paquetes al nodo Sink y la generación de los mismos.

$$D = \frac{\sum(tb - ta)}{n} \tag{5.3}$$

MÉTRICAS DE CONTROL DE CONGESTIÓN		
Ítem	Método	Métricas evaluadas
1	DALPAS	• Porcentaje de paquetes recibidos. • Throughput promedio en Sink • Retardo promedio salto a salto • Porcentaje de energía restante en la red.
2	ECODA	• Throughput • Retardo • Fairness
3	FUSION	• Eficiencia • Fairness • In-balance

Cuadro 5.1: Métricas de QoS usadas en métodos seleccionados

- Tiempo de vida de la red: El tiempo de vida será medido como la energía total restante de la red en diferentes momentos. En este caso se realizan mediciones a los 60, 120, 180, 240, 300 y 360 segundos de iniciada la simulación.
- Porcentaje pérdida de paquetes: Esta métrica indica la relación existente entre la cantidad de paquetes perdidos con respecto a todos los paquetes generados por la red. En la Ecuación 5.4 se presenta la relación descrita.

$$PPP = \frac{\sum Pl}{\sum P} \tag{5.4}$$

  El detalle de la implementación de estas métricas en código AWK se encuentran en el Anexo 5.

La implementación de las medidas de las métricas propuestas se realizó mediante AWK el cual es un lenguaje de programación diseñado para procesar datos basados en texto, que en este caso son las trazas de salida que se han obtenido después de la ejecución de la simulación.

En cuanto al detalle de las pruebas realizadas para evaluar estas metricas, el simulador NS-2 se configuró de acuerdo a los parametros descritos en la Sección 4.1. En cuanto al trafico generado por los nodos en una WSN, existen diferentes modelos que describen el trafico en la red (en especifico el generado por el nodo). En muchas aplicaciones se requiere generación periódica de datos, por lo que se configura en los nodos transmisión tipo CBR (*Constant Bit Rate*, sin embargo en muchas aplicaciones de IoT se prefiere utilizar modelos de tasa variable debido a las características propias de las soluciones. Para esto muchos autores proponen modelar el trafico con un proceso de Poisson [35] [36], no obstante se afirma que no existen solidas bases para modelar las características de una WSN con este tipo de procesos [37]. En el presente trabajo de investigación se realizaron las simulaciones con un modelo tipo ON/OFF similar al descrito en [38], el cual modela el trafico origen en los nodos como ráfagas en momentos específicos. Este modelo también aplica para aplicaciones que son basadas por eventos.

Específicamente para las simulaciones el tiempo de OFF 992ms y el tiempo de ON se configuró en 8ms en los cuales se transmitían paquetes de 70 Bytes a una tasa de 250 Kbps. Por lo tanto la tasa

efectiva de transmisión de los nodos es de aproximadamente 2 Kbps. Durante el tiempo restante el nodo no envía información al *Sink*.

Los nodos fueron distribuidos aleatoriamente en cada simulación en un terreno de 150m x 150m (aproximadamente cuatro canchas de fútbol). La configuración y características principales de los nodos se realizaron basados en dispositivos embebidos muy utilizados en IoT como la Sky Mote y la Crossbow Mica2. En este caso se calculó el nivel de energía inicial como el entregado por dos pilas de 1.5 V, potencia de transmisión de 0 dBm y sensitividad en el receptor de -64 dBm.

En total se realizaron 1500 simulaciones repartidas así: 500 simulaciones por cada topología implementaban (estrella, malla y árbol) en la cual se evaluaban los casos sin control de congestión, ECODA, DAlPaS, Fusion y MLDC (100 simulaciones por cada método). Cada simulación se ejecuta por 410 segundos (6.8 minutos), tiempo durante el cual se configura la red ad-hoc, se intercambian mensajes de enrutamiento y *Beacons*, además de la generación de trafico como se describió previamente.

A continuación se presentan los resultados obtenidos por cada topología:

5.1. Resultados Experimentales Topología Estrella

Para la implementación de la topología estrella en el simulador NS-2, todos los nodos se configuraron de tipo RFD (*Reduced Function Device*) los cuales no cumplen funciones de enrutamiento y su única función es realizar el sensado y transmitirlo directamente al nodo *Sink* el cual se encuentra en el centro del terreno.

Se realizaron en total 500 simulaciones (100 por cada caso método de control de congestión evaluado). Los resultados promedio son presentados en la Figura 5.1 donde se puede evidenciar comparativamente los resultados obtenidos en cada caso en cada una de las métricas evaluadas.

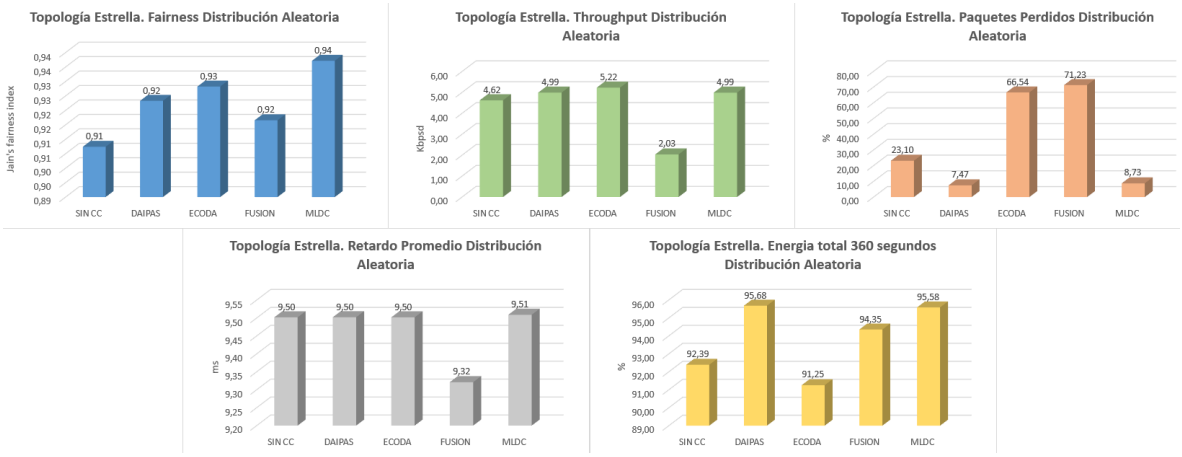


Figura 5.1: Resultados metricas topologia estrella

La topología estrella se caracteriza por tener un Fairness alto y bajo Throughput, el retardo varia muy poco entre los métodos de control de congestión y el consumo de energía es bajo. Este comportamiento se presenta debido a que el alcance, la potencia de transmisión y el nivel mínimo de energía en el receptor son críticos para esta topología. Al no tener multi-saltos, un nodo que no tenga conexión directa y alcance al nodo *Sink*, no podrá transportar su información generada.

El Fairness es alto debido a que los pocos nodos que logran comunicarse con el nodo central *Sink* lo

hacen con tasas muy similares y las más altas posibles. Sin embargo el Fairness tiene que ser analizado en conjunto con el Throughput el cual es bajo debido a que no todos los nodos que generan trafico se pueden comunicar.

El retardo no cambia significativamente entre los métodos debido a que esta métrica depende del tiempo de la propagación de la señal por el medio inalámbrico y el tiempo de procesamiento en el nodo origen y destino.

Esta topología es muy conservadora en cuanto a consumo de energía, ya que los nodos no cumplen con funciones de enrutamiento y únicamente transmiten paquetes de información y sincronización.

En la Tabla 5.2 se presenta el consolidado de las ganancias de los métodos de control de congestión con respecto al caso que no se utilizó ningún método para cada una de las métricas. En este, se evidencia que los dos métodos que mejores resultados obtuvieron fueron DAlPas (mejora en un 45 % el comportamiento de la red) y MLDC (mejora en 36 %). Este comportamiento se presenta debido a que ambos métodos tienden a convertir la red en una topología tipo árbol, además que agregan información en sus tablas de vecinos que hace que se no se pierdan muchos paquetes, el cual es la métrica que más impacta en la ganancia promedio de la topología analizada.

Metrica	**DAlPaS**	**ECODA**	**Fusion**	**MLDC**
Fairness	1.02	1.02	1.01	1.03
Throughput	1.08	1.13	0.44	1.08
Paquetes Perdidos	3.09	0.35	0.32	2.65
Retardo Promedio	1.0	1.0	1.02	1.0
Energía 360 Seg	1.04	0.99	1.02	1.03
Ganancia Promedio	1.45	0.90	0.76	1.36

Cuadro 5.2: Ganancia promedio métodos de control de congestión en topología estrella

En resumen, una WSN en topología estrella es de bajo alcance por lo que se debe analizar muy bien la potencia de transmisión de los nodos y umbrales de recepción en las interfaces de red de los nodos. Se espera que el tiempo de vida de la red sea mayor a otras debido a que no existe enrutamiento entre los nodos, por lo que si se requiere que la red inalámbrica de sensores con bajo consumo de energía, se recomienda implementarla en topología estrella a costa de bajo Throughput.

5.2. Resultados Experimentales Topología Malla

Para realizar la simulación de una WSN en topología malla, todos los nodos se configuran de tipo FFD (*Full Function Device*) el cual cumple con todas las funciones del nodo RFD y además puede ser coordinador de la red lo cual permite enrutar paquetes y sincronizar dispositivos ya que están habilitados para utilizar *Beacons*.

En este tipo de red, todos los dispositivos se pueden conectar con sus vecinos creando enlaces y enrutando paquetes, por lo que a nivel global se ve como una malla, esto crea la posibilidad de enrutar los paquetes por diferentes caminos y modificar el siguiente salto de la tabla de enrutamiento basado en información de congestión o disponibilidad.

En la Figura 5.2 se presentan las gráficas de barras con los resultados promedio de las 500 simulaciones realizadas en esta topología para cada una de las métricas evaluadas. En estas se evidencia

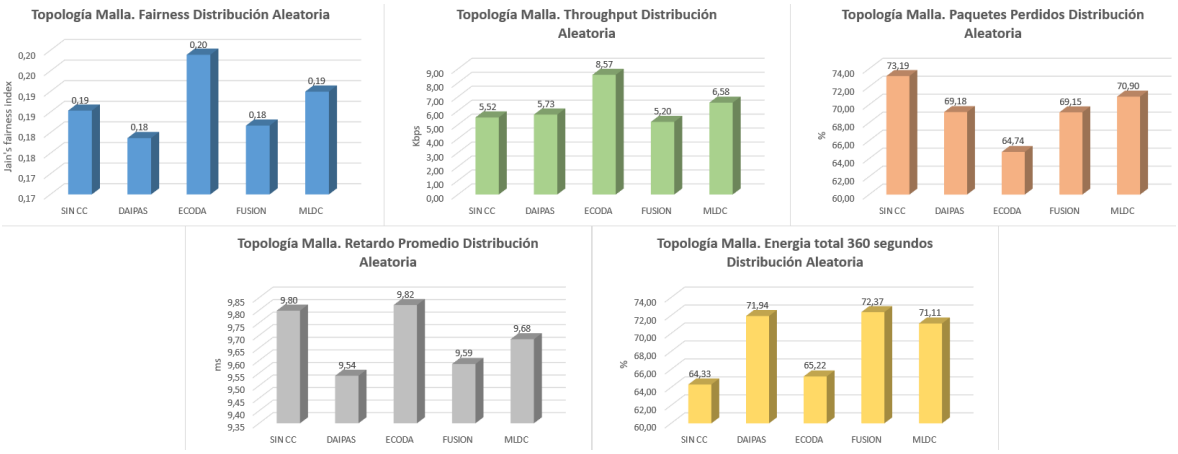


Figura 5.2: Resultados promedio métricas topología malla

el Fairness es más bajo en comparación a la WSN en estrella y que el Throughput es mayor. Esto se debe a que nodos generadores de información que se encuentran muy lejanos al nodo *Sink* y no tienen conectividad directa con este, puedan enrutar su trafico por medio de sus nodos vecinos. Sin embargo esto hace que el porcentaje de perdidas de paquetes aumente ya que es posible llenar los buffer de nodos intermedios y la congestión es mayor.

En cuanto al retardo promedio de los paquetes, este es mayor en comparación a la topología estrella debido a que en cada salto se aumenta este parámetro ya que los nodos intermedios tienen que procesar los paquetes, agregarlos en su buffer y cambiar las direcciones MAC de origen y destino. Esto hace que el consumo de energia de la red sea mayor y el tiempo de vida de la red se vea disminuido.

En general todos los métodos de control de congestión evaluados mejoran el comportamiento de la red con respecto al caso en que no se hace uso de control de congestión como se presenta en la Tabla 5.3. Los métodos de control de congestión que mejores resultados presentan en este tipo de red es ECODA y MLDC los cuales mejoran las métricas de control de congestión propuestas en un 15 % y 7 % respectivamente. Estos dos métodos mejoran significativamente el Troughput total de la red sin dañar el Fairness como lo hacen los otros dos métodos. Con estos resultados se evidencia que en nodos intermedios de la comunicación es necesario implementar formas eficientes de manejo de paquetes en los nodos.

En resumen, esta topología se caracteriza por aumentar el alcance de la red en un terreno a costa de usar múltiples saltos para garantizar la comunicación de los nodos al *Sink*. Esta topología aumenta el porcentaje de paquetes perdidos en la red, el consumo de energía y el retardo promedio de los paquetes que viajar por la red. Este tipo de redes tienen un nivel de justicia (Fairness) bajo ya que hay mucha varianza entre las tasas de transmisión de los nodos servidos, sin embargo se puede transportar más información de más nodos y de mayor área geográfica que la topología estrella.

Metrica	DAlPaS	ECODA	Fusion	MLDC
Fairness	0.96	1.07	0.98	1.02
Throughput	1.04	1.55	0.94	1.19
Paquetes Perdidos	1.06	1.13	1.06	1.03
Retardo Promedio	1.03	1.0	1.02	1.01
Energía 360 Seg	1.12	1.01	1.13	1.11
Ganancia Promedio	1.04	1.15	1.03	1.07

Cuadro 5.3: Ganancia promedio métodos de control de congestión en topología malla

5.3. Resultados Experimentales Topología Árbol

La implementación de esta topología en el simulador NS-2 se realizó configurando el 35 % de los nodos de toda la red como coordinadores de forma aleatoria. Estos nodos tienen funciones de coordinación de la red pudiendo enrutar paquetes por este. En la topología árbol comúnmente se conocen como nodos padre, los cuales sirven a sus nodos hijos como gateway para enviar los paquetes al destino, en este caso el nodo central *Sink*.

Se realizaron 100 simulaciones por cada método de control de congestión y 100 sin hacer uso de alguno. Los resultados son presentados en la Figura 5.3, en la cual se evidencia que todos los métodos de control de congestión evaluados disminuyen el Fairness a costa de un aumento del Throughput, sin embargo la justicia entre las tasas evaluadas en promedio es mayor que el de la malla y el Throughput total de la red medido en el nodo *Sink* es el mayor de todas las topologías evaluadas.

En cuanto al porcentaje total de paquetes perdidos, todos los métodos de control de congestión lo disminuyen, lo que da una mayor confiabilidad en la comunicación de los nodos. El retardo promedio de los paquetes que llegan satisfactoriamente al nodo central es disminuido por todos los métodos y no es superior a los generados en la topología malla.

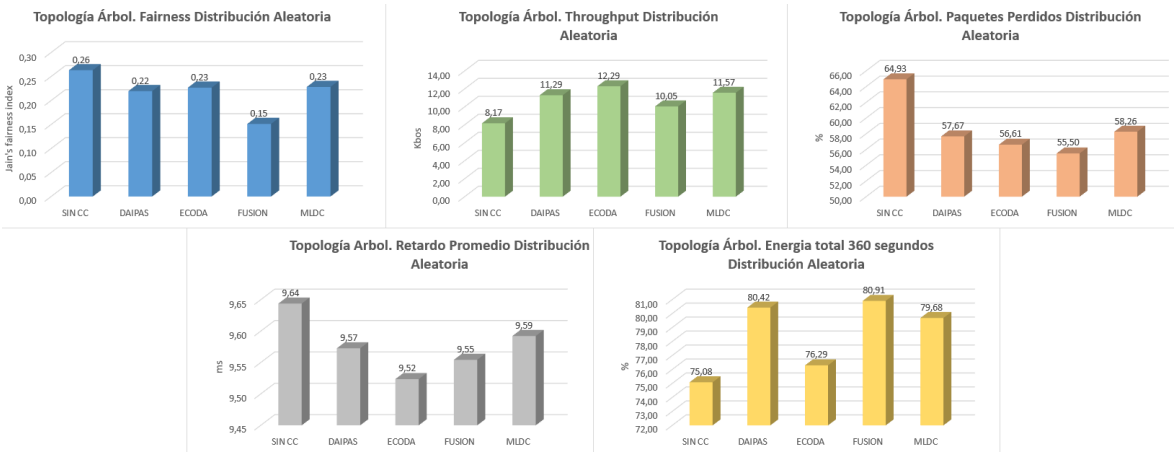


Figura 5.3: Resultados promedio métricas topología árbol

El consumo de energía también es mejorado por todos los algoritmos evaluados, y aunque el consumo es mayor que la estrella, se transportan más paquetes por la red (más del doble).

Esta topología de red presenta muy buenos resultados para las redes en que se desea que el Throughput sea alto (mayor información transportada) sin comprometer significativamente el consumo de energía y el retardo promedio de los paquetes.

En la Tabla 5.4 se presenta el detalle de la ganancia promedio total y por métrica evaluada para las 500 simulaciones realizadas en la topología árbol. En esta se evidencia que todos los métodos de control de congestión evaluados mejoran las métricas propuestas excepto el Fairness el cual es disminuido en aproximadamente 20 % en promedio. Además, todos los métodos mejoran en promedio todas mas métricas, sin embargo, ECODA y MLDC son los dos algoritmos que mejoran más significativamente estas, mejorando el comportamiento de la red en un 11 % y 9 % respectivamente.

Metrica	**DAlPaS**	**ECODA**	**Fusion**	**MLDC**
Fairness	0.83	0.86	0.57	0.87
Throughput	1.38	1.50	1.23	1.42
Paquetes Perdidos	1.13	1.15	1.17	1.11
Retardo Promedio	1.01	1.01	1.01	1.01
Energía 360 Seg	1.07	1.02	1.08	1.06
Ganancia Promedio	1.08	1.11	1.01	1.09

Cuadro 5.4: Ganancia promedio métodos de control de congestión en topología árbol

En resumen, la topología en árbol presenta muy buenos resultados en cuanto a Troughput recibido en el nodo Sink, su consumo de energía es bueno con relación a la cantidad de información transportada, la comunicación es confiable ya que el porcentaje de perdida de paquetes en la red es aceptable y además es mejorado por todos los algoritmos de control de congestión. Esta topología es en la que los métodos de control de congestión evaluados obtuvieron mejores resultados y pudieron ejercer mayor control sobre el tráfico y la topología.

5.4. Análisis comparativo de topologías

Con base en los resultados obtenidos en las 1500 simulaciones realizadas, en la presente sección se realiza un análisis comparativo de los resultados en de cada uno de los métodos de control de congestión evaluados en las diferentes topologías implementadas con base en cada una de las métricas propuestas.

5.4.1. Fairness

Esta métrica mide que tan justa fue la repartición del Throughput entre los nodos que generan información en la red. Esta fue medida con base en el índice de Jain como se presenta al inicio de este capitulo.

La topología que mayor justicia le da a la red es la estrella, seguida en promedio por la topología árbol y por ultimo la malla. Este comportamiento indica que los nodos se comunicaron con tasas similares al nodo *Sink*. Sin embargo esta métrica debe ser analizada en conjunto con el Throughput como se presenta en la Sección 5.4.6.

5.4.2. Porcentaje paquetes perdidos

Esta métrica mide la cantidad total de paquetes perdidos por problemas de comunicación y descartados por nodos debido a buffer lleno o problemas en la capa de enlace. Tener un porcentaje de paquetes perdidos bajo le da a la red mayor confiabilidad en que los paquetes generados serán transmitidos y recibidos adecuadamente en el nodo *Sink*.

La topología con mejor comportamiento en esta métrica es la estrella, seguida del árbol y por ultimo la malla. Estos resultados se presentan debido a que la estrella no considera múltiples saltos en los paquetes, por lo tanto los puntos de falla en la comunicación en la red disminuye con respecto a las otras dos topologías evaluadas.

5.4.3. Retardo promedio

El retardo promedio de los paquetes es medido como el tiempo desde que el paquete fue transmitido por el nodo origen hasta el momento en que el nodo Sink lo recibió correctamente. Un paquete no debe permanecer por mucho tiempo en transito por la red debido a que si se trata de información de eventos o alarmas, los algoritmos de control de congestión deben actuar rápido para su transmisión.

Por las características intrínsecas de la topología estrella en la cual no existen multi-saltos, esta es la que mejor comportamiento tiene seguida por el árbol y la estrella.

5.4.4. Throughput

Esta métrica mide en promedio la cantidad de información que llega correctamente al nodo *Sink* por unidad de tiempo (medido en Kbps). En una WSN se desea que este parámetro sea el más alto posible ya que indica que la red esta recolectando la mayor información posible de los sensores y de las variables leídas.

Sin embargo en redes que tienen muchos nodos generadores de tráfico, es recomendable analizar esta métrica en conjunto con la justicia en la repartición de recursos en la red (Fairness) ya que se desea que la WSN tenga alto Throughput con alto Fairness lo que indicaría que muchos nodos tuvieron altas tasas de transmisión.

La topología que mejores resultados obtuvo es el árbol, seguido por la malla y por ultimo la estrella. En la sección 5.4.6 se presenta un análisis conjunto entre el Fairness y el Throughput de todas las simulaciones realizadas.

5.4.5. Tiempo de vida

Este parámetro es muy importante en una WSN debido a que esta comúnmente debe funcionar para que el mantenimiento de la red sea bajo, específicamente en cuanto a carga de baterías o cambio de las mismas.

El consumo de energía en procesamiento, transmisión y recepción de paquetes afecta directamente el tiempo de vida de la red. Por este motivo este métrica es medida analizando la cantidad de energía de todos los nodos en diferentes momentos de la red.

Debido a que en la topología estrella no se presentan multi-saltos, el consumo de energía en procesamiento en los nodos es menor, se consume menos en transmisión y el único nodo que recibe una cantidad significativa de paquetes es el *Sink*. El segundo método de control de congestión en ahorro energético es el árbol seguido de la malla.

5.4.6. Relación Fairness - Throughput

Es necesario analizar estas dos métricas en conjunto debido a que en redes con varios usuarios se pueden llegar a mal-interpretar los resultados ya que por ejemplo al tener un Fairness alto pero un Throughput bajo significa que muy pocos nodos fueron atendidos y accedieron a los recursos de la red dejando a otros sin servicio.

El escenario ideal es lograr tener un Fairness alto con Throughput alto también, lo cual implica que muchos nodos obtuvieron de forma equitativa sus tasas de transmisión de información al nodo central de la red.

En la Figura 5.4 se presenta una gráfica de puntos de los valores promedios obtenidos en las simulaciones realizadas en las tres topologías evaluadas y en cada método de control de congestión evaluado. En este se identifica que las topologías con mejor Throughput son el árbol seguido por la malla. En cuanto a la métrica Fairness la que mejor resultado presenta es la estrella seguida del árbol y por ultimo la malla. Sin embargo este resultado en "justicia en la red"no es deseado en una WSN porque significa que se atendieron muy pocos nodos de forma equitativa y el resto transmitieron 0 Kbps como se presentará en la Sección 5.4.7.

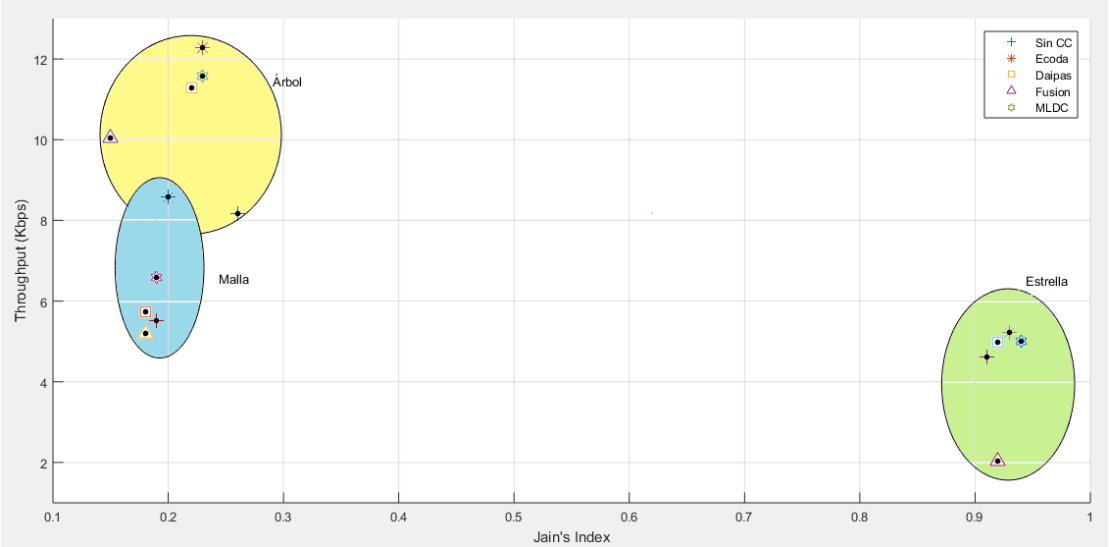


Figura 5.4: Relación Fairness - Throughput métodos de control de congestión en topologías.

5.4.7. Análisis distribución geográfica, Fairness y Throughput

El objetivo de esta sección es analizar comparativamente la relación entre la distribución geográfica de los nodos, las tasas de transmisión de algunos nodos, el Throughput total en el nodo *Sink* y el Fairness obtenido con estas tasas. Para esto se realizaron 15 simulaciones en una topología aleatoria especifica. En la Figura 5.5 se presenta el histograma de la distancia de los nodos al *Sink*.

Esta distribución fue generada de forma aleatoria al igual que las en las otras 1500 simulaciones previamente analizadas. En este histograma se observa que la mayor cantidad de nodos están ubicados aproximadamente a 70 metros del nodo *Sink* (el cual se encuentra en el centro del terreno), esto afecta considerablemente a la topología en estrella debido a que esta es limitada en cuanto a alcance de la red. Por este motivo se espera que sean atendidos únicamente los nodos que se encuentren cerca al

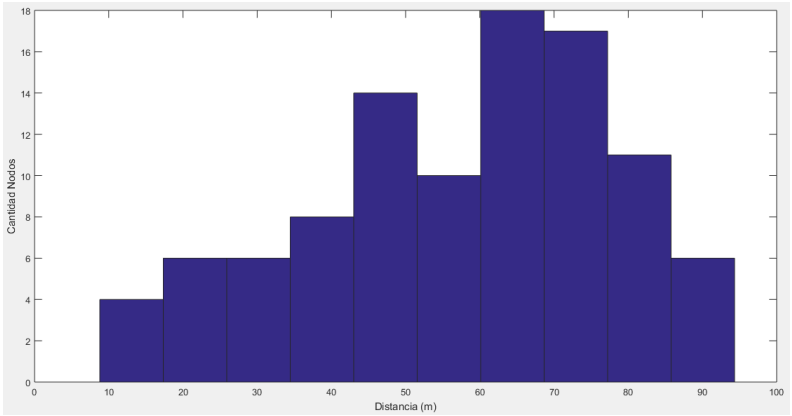


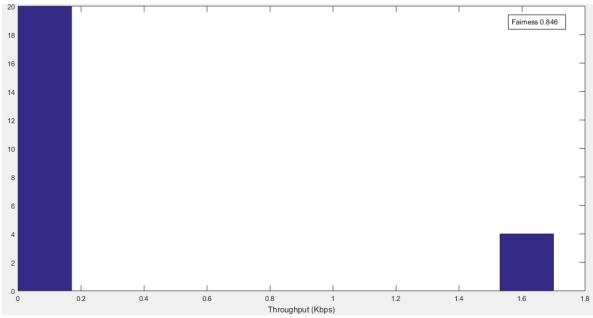
Figura 5.5: Histograma distancia de nodos a Sink.

nodo central y que logren acceder al medio.

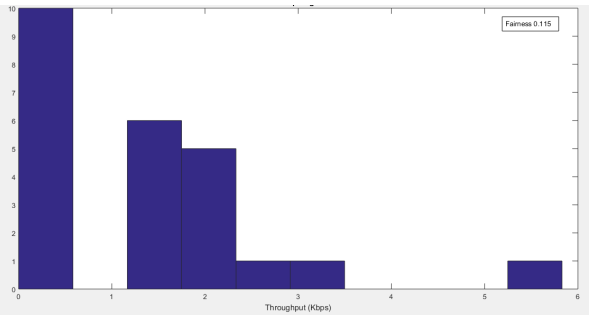
Para este conjunto de simulaciones, el 25 % de los nodos generaban trafico; estos nodos fueron distribuidos aleatoriamente por el terreno previamente descrito. Para hacer el presente análisis se midió el Fairness y las tasas únicamente de estos nodos, el resto de nodos pueden funcionar como coordinadores de la red que ayudan a enrutar el trafico según la topología analizada.

En la Figura 5.6 se presenta los histogramas de las tasas de transmisión de los nodos generadores de información obtenidos en las simulaciones sin utilizar control de congestión bajo la misma distribución aleatoria de los nodos. En este se evidencia que la topología que mayor Fairness tiene es la estrella, sin embargo en este caso únicamente 4 nodos pudieron transmitir. La segunda topología que mejor Fairness obtuvo fue el árbol con 0.115 seguido y por ultimo la malla con 0.097. Los resultados de las otras simulaciones con el comportamiento de las tasas de transmisión de los nodos usando los métodos de control de congestión evaluados son presentados en el Anexo 6.

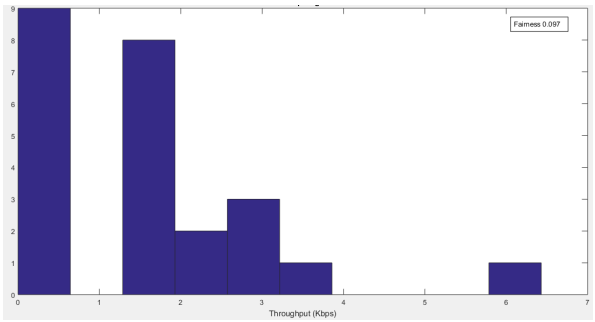
Este resultado permite justificar el análisis que se debe realizar a las métricas de control de congestión específicamente el Fairness y el Throughput, ya que aunque la estrella optimiza algunas métricas, lo hace sacrificando la cantidad de nodos que efectivamente transmitirán su información al nodo central *Sink*.



(a) Topología Estrella



(b) Topología Arbol



(c) Topología Malla

Figura 5.6: Histogramas tasas de transmisión nodos

Capítulo 6

Conclusiones y trabajo futuro

Mediante las pruebas y simulaciones realizadas en el software de simulación NS-2 se justifica la importancia de tener un control de congestión en una WSN debido a que estos mejoran el comportamiento de la red bajo las métricas seleccionadas y mitigan los efectos producidos cuando existe alto tráfico en la red. El proceso de selección del método de control de congestión se debe hacer de forma rigurosa bajo un análisis de las implicaciones que pueda tener la implementación en los nodos.

Otro factor importante en el diseño de una WSN es la topología de red de los nodos. Estas por sus propiedades intrínsecas afectan directamente la comunicación y el estado de la congestión en diferentes puntos de la red. Topologías tipo árbol y malla permiten distribuir los nodos en terrenos mas amplios en comparación a la estrella que su alcance esta limitado a la potencia de transmisión de los nodos y el umbral de recepción de la señal en el nodo destino. Por este motivo, la selección de la topología de red debe analizarse dependiendo de los requerimientos de la aplicación y solución que la WSN brindará.

Cuando se haga uso de topologías multi-salto es necesario implementar una estrategia adecuada de manejo y administración de los recursos de los nodos en especial del buffer ya que se demostró que los métodos de control de congestión que poseen algoritmos de administración de este, presentan mejores resultados en redes que requieren enrutamiento para garantizar la conectividad de los nodos.

Una red inalámbrica de sensores tiene diferentes restricciones y diferencias a las redes cableadas e inalámbricas tradicionales, por este motivo se deben definir métricas especificas que describan el comportamiento de la red bajo esas restricciones. Las métricas de congestión en WSN propuestas describen el comportamiento de la WSN bajo un punto de vista de tráfico en la red; el diseño de algoritmos de control de congestión, enrutamiento, auto-curación, etc, deben buscar optimizar estas métricas.

El Fairness y Throughput deben ser analizados en conjunto debido a que se puede llegar a mal interpretar los resultados obtenidos en el análisis de una WSN.

Aunque la selección de la topología de red debe ser basada en la aplicación, para una WSN típica de monitoreo de variables, la topología adecuada es tipo árbol con el método de control de congestión ECODA. Sin embargo, el método propuesto MLDC presenta mejoras en las métricas evaluadas en todas las tres topologías evaluadas y mejora el comportamiento general de la red por lo que se recomienda su uso e implementación. ECODA no es el método adecuado para la topología estrella.

Como trabajo futuro se sugiere realizar la evaluación de los métodos de control de congestión en ambientes reales, buscar alternativas de programación del tiempo de *backoff* en el acceso al canal compartido, evaluar diferentes algoritmos de administración del buffer en los nodos, evaluar otras topologías de red como tipo *cluster* e implementar un método de selección de *cluster heads* basado en información de congestión.

Bibliografía

[1] Q.-A. Z. Dharma P. Agrawal, *Introduction to Wireless and Mobile Systems*, 3rd ed. CL Engineering, 2010.

[2] M. Z. Alam, M. B. U. RehmanSheikh, and A. Zeshan, "Topology aware auto-configuration in wireless sensor network," 2014.

[3] B. Rashid and M. H. Rehmani, "Applications of wireless sensor networks for urban areas: A survey," *Journal of Network and Computer Applications*, vol. 60, pp. 192 – 219, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804515002702

[4] T. Arampatzis, J. Lygeros, and S. Manesis, "A survey of applications of wireless sensors and wireless sensor networks," pp. 719–724, June 2005.

[5] A. Ghaffari, "Congestion control mechanisms in wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 52, pp. 101–115, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2015.03.002

[6] M. A. Kafi, D. Djenouri, J. Ben-Othman, and N. Badache, "Congestion control protocols in wireless sensor networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1369–1390, Third 2014.

[7] C. Sergiou, P. Antoniou, and V. Vassiliou, "A comprehensive survey of congestion control protocols in wireless sensor networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 1839–1859, Fourthquarter 2014.

[8] C. V. N. Index. (2017, jun) The zettabyte era: Trends and analysis. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

[9] A. L. Colina, A. Vives, A. Bagula, M. Zennaro, and E. Pietrosemoli, *IoT in five Days*. E-Book, june 2016, rev 1.1. [Online]. Available: https://github.com/marcozennaro/IPv6-WSN-book/releases/

[10] L. Q. Tao and F. Q. Yu, "ECODA: Enhanced congestion detection and avoidance for multiple class of traffic in sensor networks," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 3, pp. 1387–1394, 2010.

[11] C. Sergiou, V. Vassiliou, and A. Paphitis, "Congestion control in Wireless Sensor Networks through dynamic alternative path selection," *Computer Networks*, vol. 75, no. PartA, pp. 226–238, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2014.10.007

[12] R. Vedantham, R. Sivakumar, and S.-j. Park, "Sink-to-Sensors Congestion Control," vol. 00, no. C, pp. 3211–3217, 2005.

[13] W.-w. Fang, J.-m. Chen, L. Shu, T.-s. Chu, and D.-p. Qian, "Congestion avoidance, detection and alleviation in wireless sensor networks," *Journal of Zhejiang University SCIENCE C*, vol. 11, no. 1, pp. 63–73, 2010.

[14] C. Sergiou, V. Vassiliou, and A. Paphitis, "Hierarchical Tree Alternative Path (HTAP) algorithm for congestion control in wireless sensor networks," *Ad Hoc Networks*, vol. 11, no. 1, pp. 257–272, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2012.05.010

[15] A. A. Rezaee, M. H. Yaghmaee, A. M. Rahmani, and A. H. Mohajerzadeh, "HOCA: Healthcare aware optimized congestion avoidance and control protocol for wireless sensor networks," *Journal of Network and Computer Applications*, vol. 37, no. 1, pp. 216–228, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jnca.2013.02.014

[16] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," *Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04*, p. 134, 2004. [Online]. Available: http://dl.acm.org/citation.cfm?id=1031512{&}dl=ACM{&}coll=DL{&}CFID=45479109{&}CFTOKEN=41920372{%}5Cnhttp://portal.acm.org/citation.cfm?doid=1031495.1031512

[17] P. Antoniou, A. Pitsillides, T. Blackwell, A. Engelbrecht, and L. Michael, "Congestion control in wireless sensor networks based on bird flocking behavior," *Computer Networks*, vol. 57, no. 5, pp. 1167–1191, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2012.12.008

[18] M. O. Atta ur Rehman Khana, Atta ur Rehman Khana, "A Performance Comparison of Open Source Network Simulators for Wireless Networks A Performance Comparison of Network Simulators for Wireless Networks," no. November, 2012.

[19] E. H. Teerawat Issariyakul, *Introduction to Network Simulator NS2*, 2nd ed. Springer, 2012.

[20] I. T. Downard, "Simulating Sensor Networks in NS-2," 2004.

[21] G. Gautam, "Design and Simulation of Wireless Sensor Network in NS2," vol. 113, no. 16, pp. 14–16, 2015.

[22] S. Maurya and N. C. Barwar, "Study and Analysis of AODV and DSDV Routing Protocols Over Zigbee Network for Different Topologies under FTP Traffic Pattern," vol. 4, no. 07, pp. 745–753, 2015.

[23] J. Sayyad and N. K. Choudhari, "Congestion Control Techniques in WSN and Their Performance Comparisons," vol. 3, pp. 108–113, 2015.

[24] X. Shen, X. Cheng, and R. Zhang, "Distributed Congestion Control Approaches," 2013.

[25] D. Shah and Q. Xie, "CENTRALIZED CONGESTION CONTROL AND SCHEDULING IN A DATACENTER," 2017.

[26] M. Zawodniok and S. Jagannathan, "Predictive congestion control mac protocol for wireless sensor networks," vol. 1, pp. 185–190 Vol. 1, June 2005.

[27] S. C. Ergen, "ZigBee/IEEE 802.15.4 Summary," in *ZigBee/IEEE 802.15.4 Summary*, Berkeley, California, EE. UU., 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.477.6183{&}rep=rep1{&}type=pdf

[28] H. Byun and J. Yu, "Adaptive duty cycle control with queue management in wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 12, no. 6, pp. 1214–1224, June 2013.

[29] K. B. T. S, "Congestion Control using Adaptive Buffer Flow Managements in WSN," vol. 3, no. 8, pp. 1–6, 2012.

[30] A. Melikov and A. Rustamov, "Queuing Management in Wireless Sensor Networks for QoS Measurement," vol. 2012, no. September, pp. 211–218, 2012.

[31] D. Goyal and M. R. Tripathy, "Routing protocols in wireless sensor networks: A survey," pp. 474–480, Jan 2012.

[32] J. Dzisigadze and K. Diawuo, "ADAPTIVE CONGESTION CONTROL PROTOCOL (ACCP) FOR WIRELESS SENSOR NETWORKS," vol. 5, no. 5, pp. 129–144, 2013.

[33] D. Xia and Q. Li, "A routing protocol for congestion control in rfid wireless sensor networks based on stackelberg game with sleep mechanism," pp. 207–211, Sept 2013.

[34] H. Shi, R. V. Prasad, E. Onur, and I. G. M. M. Niemegeers, "Fairness in Wireless Networks - Issues , Measures and Challenges," pp. 1–20.

[35] Y. Ma and J. H. Aylor, "System lifetime optimization for heterogeneous sensor networks with a hub-spoke technology," *IEEE Transactions on Mobile Computing*, vol. 3, no. 3, pp. 286–294, July 2004.

[36] S. Tang, "An analytical traffic flow model for cluster-based wireless sensor networks," pp. 5 pp.–, Jan 2006.

[37] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, Jun 1995.

[38] Q. Wang and T. Zhang, "Source traffic modeling in wireless sensor networks for target tracking," in *Proceedings of the 5th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*, ser. PE-WASUN '08. New York, NY, USA: ACM, 2008, pp. 96–100. [Online]. Available: http://doi.acm.org.ezproxy.javeriana.edu.co:2048/10.1145/1454609.1454629