

UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE CUENCA

CARRERA DE INGENIERÍA ELECTRÓNICA.

Tesis previa a la obtención del Título de:

Ingeniero Electrónico

TÍTULO:

“IMPLEMENTACIÓN DE UN AMBIENTE DE SIMULACIÓN BASADO EN SOFTWARE LIBRE PARA EL ESTUDIO DE LA PROVISIÓN DE SERVICIOS DE COMUNICACIONES EN REDES VEHICULARES AD-HOC MEDIANTE EL USO DE NODOS MÓVILES VIRTUALES”

AUTORES:

Luis Alberto Caldas Calle

Juan Carlos Zaruma Villamarín

DIRECTOR:

Ing. Jack Bravo

Cuenca, Mayo del 2013

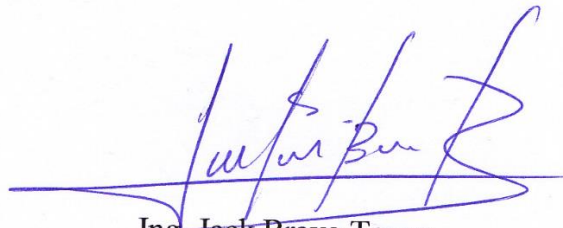
Ing. Jack Bravo Torres,

Director de Tesis.

CERTIFICA

Que el trabajo intitulado *“Implementación de un ambiente de simulación basado en software libre para el estudio de la provisión de servicios de comunicaciones en redes vehiculares ad-hoc mediante el uso de nodos móviles virtuales.”*, realizado por Luis Alberto Caldas Calle y Juan Carlos Zaruma Villamarín, cumple con todos los objetivos trazados.

Cuenca, Mayo del 2013



Ing. Jack Bravo Torres


DIRECTOR DE TESIS

DECLARATORIA DE RESPONSABILIDAD


Nosotros, Luis Alberto Caldas Calle y Juan Carlos Zaruma Villamarín, Autores del presente Trabajo de Tesis intitulado, “IMPLEMENTACIÓN DE UN AMBIENTE DE SIMULACIÓN BASADO EN SOFTWARE LIBRE PARA EL ESTUDIO DE LA PROVISIÓN DE SERVICIOS DE COMUNICACIONES EN REDES VEHICULARES AD-HOC MEDIANTE EL USO DE NODOS MÓVILES VIRTUALES” declaramos que:

Los conceptos desarrollados, análisis realizados y las conclusiones del presente trabajo, son de exclusiva responsabilidad de los Autores. Autorizamos a la Universidad Politécnica Salesiana el uso de la misma con fines académicos.

Cuenca, Mayo de 2013



Luis Alberto Caldas C.



Juan Carlos Zaruma V.

AGRADECIMIENTO

A mis padres, hermanos y hermanas por el apoyo entregados durante toda mi vida, en especial al director de mi tesis Jack Bravo por darme el apoyo y la dirección para la culminación de este trabajo investigativo.

Luis Alberto

AGRADECIMIENTO

Quiero agradecer de manera muy especial a todas las personas que me brindaron su apoyo de forma incondicional a lo largo de mi formación profesional, a mis padres, hermanos, familiares y amigos. También quería agradecerle, a nuestro director de tesis, por su paciencia y constancia, ya que sin su apoyo no hubiera sido posible culminar este proyecto de tesis de manera exitosa.

Juan Carlos

DEDICATORIA

A mis padres, hermanos y hermanas por darme el apoyo, para poder alcanzar este logro.

Luis Alberto

A mis padres, hermanos, a mi sobrino y a mis amigos, pero de manera especial a mi padre Jorge Antonio, que aunque ya no está con nosotros, sin su apoyo no hubiera sido posible cumplir esta meta.

Juan Carlos

RESUMEN

Las redes VANETs son una clase de red ad-hoc, que se destacan por aspectos como la autonomía y movilidad de sus nodos. La movilidad de sus nodos hace que la topología de red sea altamente variable, dado que los nodos pueden entrar o salir de una región. Por este motivo, el protocolo de comunicación establecido entre los miembros de la red debe cubrir eficientemente estas consideraciones con el fin de garantizar una óptima calidad del servicio ofrecido. Los protocolos que se utilizan para controlar el encaminamiento de los paquetes a intercambiar dentro de una red VANET mantienen ciertas características propias de los protocolos desarrollados para MANET. La diferencia se da en el hecho de que deben soportar una topología escalable y variable. Así, la alta movilidad de los nodos ha hecho que estos protocolos muestren deficiencias en su implementación en redes vehiculares, por lo que se requiere el estudio de nuevos protocolos que consideren las características propias de las VANETs.

Nuestro simulador dará el soporte para la implementación de nuevos constructos que se proponen desde el Grupo de Investigaciones de la Universidad Politécnica Salesiana con respecto a los nodos virtuales en redes vehiculares. Así, nuestro ambiente de simulación contiene tres agentes: Master, Members y Constancy, en ns-2. El agente Master realiza un rastreo del movimiento del nodo y elige entre los nodos de la región un líder. El rastreo del nodo determina su ubicación, velocidad y dirección. El Agente Constancy inicia su función al momento de recibir un paquete desde el Master. El Constancy coordina las acciones de los tres agentes. El Agente Members inicia su función al momento en que el Constancy notifique de elección del líder, y se comprobó su funcionamiento en ambientes vehiculares.

ÍNDICE

1. INTRODUCCIÓN	1
1.1. Redes Móviles Ad-hoc	2
1.2. Infraestructura Virtual	3
1.3. Organización.....	4
2. ESTADO DEL ARTE.....	6
2.1. Introducción.....	6
2.2. Redes Móviles Ad-hoc.	8
2.2.1. Tipos de redes Inalámbricas: Infraestructura y Ad-hoc	8
2.2.2. Red de Infraestructura Inalámbrica.....	8
2.2.3. Red Inalámbrica Ad-hoc.....	9
2.2.4. Red Ad-hoc: MANET.....	10
2.2.5. Protocolos de Encaminamiento	11
2.3. Redes Vehiculares Ad-hoc: VANET	12
2.3.1. Descripción General de la Estructura VANET	13
2.3.2. Características y Consideraciones Importantes de una VANET.....	14
2.3.3. Frecuencias de Operación de VANET.....	15
2.3.4. Principales Protocolos de Enrutamiento	15
2.3.5. Aplicaciones Principales en VANET.....	16
2.3.6. Futuras Investigaciones	18
2.4. Virtualización en Redes Vehiculares	19
2.5. Simuladores de Redes de Telecomunicaciones	22
2.5.1. Network Simulator 2.....	22
2.5.2. Objective Modular Network Testbed in C++ (OMNeT++).....	23
2.5.3. Simuladores Escritos en Java.....	24
2.6. Simuladores de Tránsito Vehicular.....	25
2.6.1. Simulation of Urban MObility (SUMO).....	26
2.6.2. VanetMobisim	27
2.6.3. STreet RAndow Waypoint (STRAW).....	28
3. ESTRUCTURA DEL SIMULADOR.....	31
3.1. Introducción.....	31
3.2. Bloques Funcionales.....	32
3.3. Identificación y Diseño de las Clases, Atributos y Métodos del Ambiente de Simulación. 36	
3.3.1. Paquetes de la Capa Virtual	39
3.3.2. Algoritmo para elección de un Nodo Líder	43
3.3.3. Agentes	45

3.3.3.1. Agente Master.....	46
3.3.3.1.1. Clases del Agente Master.....	46
3.3.3.2. Agente Constancy.....	57
3.3.3.2.1. Clases del Agente Members.....	57
3.3.3.3. Agente Members.....	61
3.3.3.3.1. Clases del Agente Members.....	61
3.3.4. Archivo de Resultados de la Capa Virtual.....	63
3.4. Enlace del Simulador de Red y El Simulador de Tránsito Vehicular.....	64
3.4.1. TraceExporter.....	64
4. IMPLEMENTACIÓN Y PRUEBAS.....	66
4.1. Implementación sobre ns-2 y SUMO.....	66
4.1.1. Implementación sobre Sumo.....	66
4.1.1.1. Características de Sumo.....	66
4.1.1.2. Escenarios de Simulación.....	68
4.1.1.2.1. Creación de la Red Vial.....	68
4.1.1.2.2. Creación del Tránsito Vehicular.....	80
4.1.2. Implementación del Ambiente de Simulación Diseñado en ns-2.34.....	84
4.1.3. Elaboración de un script para Simulación de Enlaces Inalámbricos en ns-2.....	94
4.2. Diseño de Experimentos sobre un Nodo Virtual.....	102
4.3. Análisis de los resultados.....	129
5. CONCLUSIONES Y RECOMENDACIONES.....	131
5.1. Conclusiones.....	131
5.2. Contribuciones.....	132
5.3. Futuras Investigaciones.....	132
ANEXOS.....	140

ÍNDICE DE FIGURAS

Fig. 1. La red Inalámbrica Infraestructura	9
Fig. 2. Estructura de la red inalámbrica Ad-hoc. (A) El nodo A no puede establecer un enlace de comunicación con el nodo C, (B) El nodo B realiza la tarea de enrutador en el enlace de comunicación del nodo A y nodo C.....	10
Fig. 3. Organigrama de protocolos de enrutamiento ad-hoc [10].	11
Fig. 4. Representación general de los elementos de una VANET [39].....	14
Fig. 5. Representación de los tipos de comunicación en una VANET [23].....	17
Fig. 6. La capa virtual. La figura de la izquierda muestra la rejilla para la implementación de una capa virtual, en donde los rectángulos representan los nodos virtuales, y los pequeños círculos los nodos reales. La figura derecha nos muestra los nodos físicos reales [36].....	20
Fig. 7. La figura nos muestra la implementación de los VVC sobre los nodos móviles virtuales [7].	21
Fig. 8. Estructura del simulador KivaNS [13].	25
Fig. 9. Entorno gráfico del simulador de tránsito SUMO [3].	27
Fig. 10. Entorno gráfico del simulador de tránsito VanetMobisim [18].	28
Fig. 11. Entorno gráfico del simulador de tránsito STRAW [1].	28
Fig. 12. Diagrama de bloques funcionales del ambiente de simulación propuesto [8].....	34
Fig. 13. Diagrama de bloques funcionales para describir la capa de virtualización [43].....	35
Fig. 14. Bloques funcionales para describir el funcionamiento de ns-2 [30].....	36
Fig. 15. Arquitectura del Ambiente de Simulación en la plataforma ns-2 [2].	37
Fig. 16. Formato de un paquete de ns-2 [21].	39
Fig. 17. Arquitectura interna de un paquete en ns-2 [19].....	40
Fig. 18. Principales elementos de las cabeceras de los paquetes implementadas en el ambiente de simulación.....	42
Fig. 19. Algoritmo para elección de Líder [42].	45
Fig. 20. Enlace del Agente Master con OTcl.	47
Fig. 21. Diagrama de Colaboración entre Master y los contadores de tiempo para el control de la elección del Líder.	48
Fig. 22. Ubicación del Agente Master en ns-2.....	49
Fig. 23. Ubicación de las cabeceras de los paquetes MasterHeaderClass y EstandarHeaderClass en ns-2.....	57
Fig. 24. Enlace del Agente Constancy con OTcl	58
Fig. 25. Ubicación del Agente Constancy en ns-2.....	58
Fig. 26. Enlace del Agente Master con OTcl	61
Fig. 27. Ubicación del Agente Members en ns-2.....	62
Fig. 28. Captura de un fichero de simulación para el enlace de SUMO y ns-2	65
Fig. 29. Aplicaciones del paquete de instalación de SUMO 0.16.0.....	67
Fig. 30. Red vial tipo Grid	70
Fig. 31. Red vial tipo Spider	71
Fig. 32. Red vial tipo Aleatoria.....	73
Fig. 33. Esquema para la generación manual de una red vial	74
Fig. 34. Archivo de configuración de una red vial generada manualmente.....	77
Fig. 35. Red vial generada manualmente: a) Zoom 100m, b) Zoom 10m	78
Fig. 36. Zona de la Universidad Politécnica Salesiana sede Cuenca con OpenStreetMap ...	79
Fig. 37. Vista del mapa de la Universidad Politécnica Salesiana en JOSM	79
Fig. 38. Esquema para la generación del archivo *.rou.xml	80
Fig. 39. Definiciones de variables por defecto para los agentes Master, Constancy y Members	86

Fig. 40. Definición de PT_MASTER y PT_ESTANDAR en la clase p_info()	86
Fig. 41. Definiciones de identificadores para PT_ESTANDAR y PT_MASTER	87
Fig. 42. Definiciones de PT_ESTANDAR y PT_MASTER dentro de clase p_info()	87
Fig. 43. Modificaciones dentro de la función initName()	88
Fig. 44. Especificación de PT_ESTANDAR y PT_MASTER como paquete de datos	88
Fig. 45. Modificaciones sobre el fichero ns-packet.tcl	89
Fig. 46. Modificaciones sobre el fichero cmu-trace.h	90
Fig. 47. Incluimos el fichero vncaza_pkt.h que contiene las cabeceras en cmu-trace.cc	90
Fig. 48. Llamamos a las funciones agregadas en cmu-trace.h	91
Fig. 49. Modificaciones en fichero cmu-trace.cc para visualización de trazas	92
Fig. 50. Modificaciones en el fichero Makefile	93
Fig. 51. Compilación y Detección de errores	94
Fig. 52. Descripción interna de un script para simulación en ns-2 [29]	95
Fig. 53. Descripción de los principales elementos que forman parte de la cabecera de un script para un enlace inalámbrico	95
Fig. 54. Descripción de las características del medio de transmisión	96
Fig. 55. Descripción de las características del nodo	97
Fig. 56. Definimos el espacio de trabajo para el ambiente de simulación	97
Fig. 57. Creación de un objeto GOD	97
Fig. 58. Configuración y definición de todos los nodos definidos para la topología	98
Fig. 59. Configuración y definición de todos los nodos definidos para la topología	99
Fig. 60. Configuración de patrones de movimiento de todos los nodos definidos para la topología	100
Fig. 61. Configuración de agentes Master. Constancy Members	101
Fig. 62. Definición de procesos para finalización de simulación en el script	101
Fig. 63. Escenario de simulación de nodos estáticos	103
Fig. 64. Umbrales de detección de transmisión y recepción de paquetes para un nodo [19]	104
Fig. 65. Diagrama de estados del nodo en el escenario de simulación con nodo estáticos en la Región [0; 3]	105
Fig. 66. Estado de los nodos 7, 8 y 9 en el tiempo de simulación [0s; 120s] en el escenario de simulación de nodos Estáticos	106
Fig. 67. Escenario de simulación de MANET	108
Fig. 68. Estado del nodo 0 en el tiempo de simulación [0s; 50s] en el escenario de simulación MANET	111
Fig. 69. Diagrama de estados del nodo en el escenario MANET: Región [3; 1]	112
Fig. 70. Estado de los nodos 0, 7, 9, 11, 12 y 13 en el tiempo de simulación [0s; 150s] en el escenario de simulación VANET	114
Fig. 71. Vista del mapa de la Av. Huayna-Cápac en JOSM	116
Fig. 72. Generación de la demanda de tránsito vehicular sobre la Av. Huayna-Cápac; a) La Red Vial de la Av. Huayna-Cápac, b) Captura del movimiento de los vehículos sobre la red vial	118
Fig. 73. Escenario de simulación VANET	120
Fig. 74. Diagrama de estados del nodo en el escenario VANET: Región [0; 3]	121
Fig. 75. Estado de los nodos 1, 2, 9, 15, 17 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET	123
Fig. 76. Estado del nodo 15 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET	124
Fig. 77. Escenario de simulación de los tres agentes Master, Constancy y Members	126
Fig. 78. Comunicación de los Agentes Master-Constancy	128
Fig. 79. Comunicación de los Agentes Constancy-Members	128

ÍNDICE DE TABLAS

Tabla 1. Análisis de los Simuladores de Tránsito.....	29
Tabla 2. Contadores de tiempo del Agente Master.....	48
Tabla 3. Comandos de la herramienta TraceExporter.....	64
Tabla 4. Características de las aplicaciones del paquete de instalación de SUMO 0.16.0.....	67
Tabla 5. Comandos de generación de la red tipo Grid.....	69
Tabla 6. Comandos de generación de cruces de la red tipo Grid.....	69
Tabla 7. Comandos de generación de cruces de la red tipo Spider.....	71
Tabla 8. Comandos de generación de cruces de la red tipo Aleatoria.....	72
Tabla 9. Comandos de generación de un archivo node.....	75
Tabla 10. Comandos de generación de un archivo node.....	76
Tabla 11. Comandos de generación de un archivo type.....	76
Tabla 12. Comandos de generación de un archivo connection.....	77
Tabla 13. Comandos de generación de un archivo de rutas usando trips.....	82
Tabla 14. Comandos de generación de rutas aleatorias.....	83
Tabla 15. Comandos de generación de rutas manualmente.....	83
Tabla 16. Valores de la simulación del canal inalámbrico IEEE 802.11 para el escenario de simulación de nodos Estáticos.....	102
Tabla 17. Valores de la Simulación del Agente Master para el escenario de simulación de nodos Estáticos.....	103
Tabla 18. Estado de los nodos 7, 8 y 9 en el tiempo de simulación [0s; 120s] en el escenario de simulación de nodos Estáticos.....	106
Tabla 19. Valores de la simulación del canal inalámbrico IEEE 802.11 para el escenario de simulación MANET.....	108
Tabla 20. Valores del Nodo Virtual en el escenario de simulación MANET.....	109
Tabla 21. Valores de la Simulación del Agente Master para el escenario de simulación MANET.....	109
Tabla 22. Estado de los nodos 0, 7, 9, 11, 12 y 13 en el tiempo de simulación [0s; 150s] en el escenario de simulación MANET.....	113
Tabla 23. Valores de la Simulación del canal Inalámbrico IEEE 802.11 en el escenario de simulación VANET.....	117
Tabla 24. Valores del Nodo Virtual en el escenario de simulación VANET.....	119
Tabla 25. Valores de la Simulación del Agente Master en el escenario de simulación VANET.....	120
Tabla 26. Estado de los nodos 1, 2, 9, 15, 17 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET.....	123
Tabla 27. Valores de la Simulación del canal Inalámbrico IEEE 802.11 en el escenario de simulación de los tres agentes Master, Constancy y Members.....	125
Tabla 28. Valores de la Simulación del Agente Master en el escenario de simulación de los tres Agentes: Master, Constancy y Members.....	126
Tabla 29. Valores del Nodo Virtual en el escenario de simulación de los tres Agentes: Master, Constancy y Members.....	127

1. INTRODUCCIÓN

Los sistemas de comunicación son parte fundamental del desarrollo de las sociedades, transformando el estilo de vida de las personas. Así, las redes de comunicaciones han evolucionado para permitirnos estar conectados en cualquier parte e instante. La flexibilidad en los dispositivos de comunicación —redes heterogéneas— permite hablar de ubicuidad en los servicios prestados al usuario y sobre todo analizar la posibilidad de nuevas redes basadas en dispositivos no pensados para transmisión de datos. Un ejemplo de esta evolución son las redes de comunicaciones vehiculares, sistemas de transporte transformados en redes de comunicaciones, su implementación permitirá mejorar la calidad del tiempo que las personas gastan en sus traslados, brindándoles seguridad y confort.

El estudio de las redes de comunicaciones vehiculares es un campo de amplia expectativa entre la comunidad científica. Así, nuevos servicios de comunicaciones, no observados hasta el momento, son posibles: llamadas e intercambio de mensajes entre usuarios presentes en la red, avisos de seguridad, acceso a Internet, servicios multimedia interactivos, entre otros. Sin embargo, la alta variabilidad de la topología en este tipo de redes ha hecho que los protocolos de comunicaciones, propuestos hasta la fecha para VANETs, incurran en un excesivo overhead, retardos de extremo a extremo, entre otros, lo que hace inviable el despliegue de servicios más demandantes [5].

Con la finalidad de hacer frente a la variabilidad en la topologías de las redes móviles ad hoc (MANETs), el grupo de investigación de la Prof. Nancy Lynch del Computer Science y el Artificial Intelligence Laboratory del MIT [9], propone el uso de nodos móviles virtuales (VMNs) —nodos abstractos que se mueven de una manera predeterminada— para generar una capa virtual sobre la cual desplegar servicios de comunicaciones. Así, valiéndose de esta noción, Bravo, et al. [5] proponen el uso de

los VMNs en ambientes vehiculares tomando ventaja del hecho de que este tipo de redes tienen mayores limitaciones en movilidad. Dentro de estas investigaciones, en [5] se propone un ambiente de simulación de nodos virtuales en ambientes vehiculares basados en los trabajos desarrollados en [36] y [42] el uso de software libre.

En la actualidad, el Grupo de Investigaciones en Telecomunicaciones de la Universidad Politécnica Salesiana (GITEL) —del cual somos parte— y profesores de la Universidad de Vigo están desarrollando nuevas investigaciones para proporcionar servicios de comunicaciones más demandantes en VANETs a través del uso de VMNs. Para ello, la implementación de un simulador basado en software libre que integre los conceptos de los nodos móviles virtuales en ambientes vehiculares es necesaria. Si bien existe una primera aproximación en cuanto al simulador propuesto por nuestro grupo de investigaciones, es necesario potencializar en implementar varios de los bloques estructurales del mismo.

Por otra parte, el llevar a cabo esta investigación sobre un tema que no es ampliamente conocido a nivel de pregrado (VANETs, simuladores para las redes de comunicaciones y redes de tránsito vehicular) permitirá incrementar y afianzar el conocimiento relacionado a este campo de estudio. La falta de investigaciones similares e información distribuida de manera libre acerca de estos temas, implica que el presente trabajo puede ser considerado como pionero en este campo a nivel local.

1.1. Redes Móviles Ad-hoc

Las VANETs son una subclase de las redes MANETs. Este tipo de redes no tiene una infraestructura fija sino que por el contrario confían en los vehículos para proveer las funcionalidades de la red. El despliegue de este tipo de redes ha causado

una gran expectativa no sólo en la comunidad científica sino también en las entidades gubernamentales, debido a las posibles aplicaciones que se pueden desprender en bien de los usuarios de las vías públicas: sistemas de información de tránsito, localización de lugares de interés, mensajes de emergencia, alarmas sobre las condiciones de la vía, entre otras [44]. Sin embargo, la alta variabilidad en la topología de la red —debido a las fluctuaciones en las condiciones del tránsito— ha hecho que los protocolos de comunicaciones propuestos hasta el momento para VANETs no presenten las condiciones suficientes para brindar servicios de comunicaciones más allá de la disseminación de mensajes de alarma [8]. En este trabajo de investigación, se analizarán las principales concepciones sobre los nodos móviles virtuales y los simuladores disponibles para la implementación de un ambiente de simulación de estos nodos en redes vehiculares.

1.2. Infraestructura Virtual

Como se estableció anteriormente, el objetivo es implementar un ambiente de simulación de nodos virtuales en escenarios de redes vehiculares. Para este propósito, la base serán las investigaciones preliminares desarrolladas por el grupo de investigación y el diseño propuesto en [5]. Nuestro trabajo básicamente se enfocará a implementar esta propuesta mediante el uso de software libre tanto de simuladores de redes de comunicaciones como de tránsito vehicular. Los módulos propuestos se describen a continuación:

- a. **Generador de Tránsito Vehicular:** es el encargado de adecuar los patrones de movimiento generados por el simulador de tránsito vehicular al formato apropiado para el simulador de red. En este sentido se maneja como un enlace entre los dos simuladores.
- b. **Analizador de Tránsito:** este módulo analiza las principales características y estadísticas del tránsito vehicular generado con la finalidad de proporcionar información suficiente para los diferentes constructos de la capa de virtualización.

- c. La capa de virtualización:** está implementada sobre el simulador de red y es la encargada de gestionar las diferentes funcionalidades de los nodos móviles virtuales. Está compuesta por tres agentes: Master, Members y Constancy:
- El agente Master realiza un rastreo del movimiento del nodo físico y elige entre los nodos físicos de la región un *Líder*. El seguimiento del nodo determina su ubicación, velocidad y dirección. La elección del *Líder* es manejado por el algoritmo diseñado en [42].
 - El agente Constancy coordina los agentes de la capa virtual. El agente inicia su función al momento de recibir un paquete desde el Master. Una vez que un *Líder* es seleccionado en la región¹, éste coordina las acciones del agente Members [42].
 - El agente Members inicia su función al momento en que recibe un paquete del Constancy. En [42] especifica que el agente Members permite la coordinación de la capa de aplicación con la capa virtual.
- d. La capa de aplicación:** basada en el simulador de red, es la encargada de generar las diferentes aplicaciones a desplegarse.

1.3. Organización

En el capítulo II, nosotros estudiaremos el concepto de nodo móvil virtual y los principales trabajos desarrollados en este campo. Además, analizaremos los diferentes tipos de software de simulación de sistemas de tránsito vehicular y simulación de redes de comunicaciones que brinden soporte a nuestra investigación.

¹ En la capa virtual, se denomina región al área física de un nodo virtual [42].

En el capítulo III, nosotros ilustraremos la estructura del simulador mediante bloques funcionales con el diseño de las clases del simulador de red. Además, describimos el enlace entre el simulador de red con el simulador de tránsito vehicular.

El capítulo IV, contiene la implementación del ambiente de simulación en el simulador de redes. Nosotros presentamos experimentos, resultados y análisis del ambiente de simulación en redes ad hoc. Por último, en el capítulo V nosotros concluimos y recomendamos.

2. ESTADO DEL ARTE

2.1. Introducción

Los sistemas de comunicación son parte fundamental del desarrollo de las sociedades, transformando el estilo de vida de las personas. Las redes de comunicaciones han evolucionado hasta el punto de permitirnos estar conectados en cualquier parte e instante. La flexibilidad en los dispositivos de comunicación — redes heterogéneas— permite hablar de ubicuidad en los servicios prestados al usuario y sobre todo analizar la posibilidad de nuevas redes basadas en dispositivos no pensados para transmisión de datos. Un ejemplo de esta evolución son las redes de comunicaciones vehiculares, sistemas de transporte transformados en redes de comunicaciones, su implementación permitirá mejorar la calidad del tiempo que las personas gastan en sus traslados, brindándoles seguridad y confort.

El estudio de las redes de comunicaciones vehiculares es un campo de amplia expectativa entre la comunidad científica. Así, nuevos servicios de comunicaciones, no observados hasta el momento, son posibles: llamadas e intercambio de mensajes entre usuarios presentes en la red, avisos de seguridad, acceso a Internet, servicios multimedia interactivos, entre otros. Las redes VANETs son una subclase de las redes MANETs de allí la importancia de profundizar acerca de este tipo de redes. El interés en realizar investigaciones sobre este tipo de redes ad-hoc surge como producto de que este tipo de redes no tiene una infraestructura fija sino que por el contrario confían en los vehículos para proveer las funcionalidades de la red. El despliegue de este tipo de redes ha causado una gran expectativa no sólo en la comunidad científica sino también en las entidades gubernamentales, debido a las posibles aplicaciones que se pueden desprender en bien de los usuarios de las vías públicas: sistemas de información de tránsito, localización de lugares de interés, mensajes de emergencia, alarmas sobre las condiciones de la vía, entre otras [44].

Un gran inconveniente para la implementación de las redes vehiculares VANETs es establecer un protocolo de comunicación funcional frente a la movilidad de los nodos y la variabilidad provocados por estos en la topología existente. Con la finalidad de hacer frente a esta variabilidad en la topologías de las redes móviles ad-hoc (MANETs), el grupo de investigación de la Prof. Nancy Lynch del Computer Science y el Artificial Intelligence Laboratory del MIT [9], propone el uso de nodos móviles virtuales (VMNs) —nodos abstractos que se mueven de una manera predeterminada— para generar una capa virtual sobre la cual desplegar servicios de comunicaciones. Dentro de estas investigaciones, en [8] se propone un ambiente de simulación de nodos virtuales en ambientes vehiculares basados en los trabajos desarrollados en [36] y [42] el uso de software libre.

Por esta razón, la implementación de un simulador basado en software libre que integre los conceptos de los nodos móviles virtuales en ambientes vehiculares es necesaria. Por otra parte, el llevar a cabo esta investigación sobre un tema que no es ampliamente conocido a nivel de pregrado (VANETs, simuladores para las redes de comunicaciones y redes de tránsito vehicular) permitirá incrementar y afianzar el conocimiento relacionado a este campo de estudio. La falta de investigaciones similares e información distribuida de manera libre acerca de estos temas, implica que el presente trabajo puede ser considerado como pionero en este campo a nivel local.

Este capítulo está estructurado de la siguiente manera: en las secciones 2.1 y 2.2 se analizan las redes móviles ad-hoc y las redes vehiculares ad-hoc, respectivamente. En la sección 4 se describen procesos de virtualización en VANETs y finalmente en las secciones 5 y 6 se muestran los principales simuladores de red y transporte.

2.2. Redes Móviles Ad-hoc.

En la actualidad, el estilo de vida de las personas genera un aumento en la necesidad de transmitir datos a través de una red inalámbrica. Esto es consecuencia de la multiplicación y flexibilidad de dispositivos de comunicación portátiles e inalámbricos. Así, las redes de comunicaciones inalámbricas han evolucionado para permitirnos estar conectados en cualquier parte e instante. Estas redes presentan variaciones en su topología y protocolos a ser utilizados. En esta sección se analiza los tipos de redes existentes y se da una visión de los protocolos de encaminamiento existentes.

2.2.1. Tipos de redes Inalámbricas: Infraestructura y Ad-hoc

Las redes de infraestructura se caracterizan por el empleo de un punto de acceso central que se encarga de direccionar la información entre los dispositivos terminales. Por otra parte, en las redes ad-hoc, los dispositivos terminales transmiten información entre sí en forma directa, sin la necesidad de un punto de acceso central.

2.2.2. Red de Infraestructura Inalámbrica

Las redes de infraestructura utilizan un dispositivo central que realiza el trabajo de administrador de la red. Véase la *Fig. 1*. Éste dispositivo se encarga de encaminar la información de los terminales que componen la red de comunicación: terminales móviles o fijos. Los dispositivos fijos pueden tener enlaces cableados o inalámbricos al dispositivo central. Los terminales móviles usan enlaces inalámbricos. Ejemplos de este tipo de redes son las redes inalámbricas LAN, las redes de telefonía celular, entre otras [38].

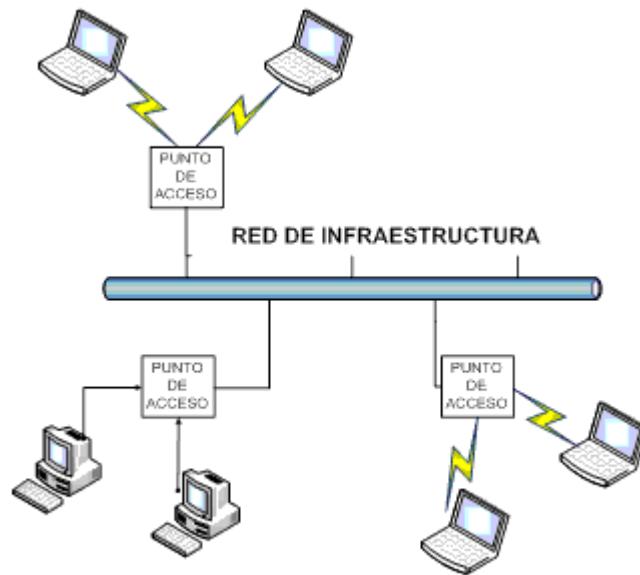


Fig. 1. La red Inalámbrica Infraestructura

2.2.3. Red Inalámbrica Ad-hoc

En una red inalámbrica ad-hoc los dispositivos terminales no dependen de un dispositivo central, éstos se comunican directamente entre sí; de esta manera, los terminales se convierten en transmisores, receptores y enrutadores de datos. En éstas redes, a los dispositivos se les denominan nodos [38].

Un nodo puede establecer un enlace inalámbrico con otro nodo en dos formas: si el nodo receptor está dentro de su área de cobertura, el enlace es directo; en caso contrario, la transmisión de datos se desarrollará a través de los nodos que se encuentre entre los dos nodos comunicantes. Por ejemplo, en la *Fig. 2*. Se presentan tres nodos: A, B y C. El nodo A y nodo C requieren establecer comunicación; sin embargo, debido a sus radios de cobertura esto es imposible. Por tanto, para establecer el enlace de comunicación entre los nodos A y C se utiliza al nodo B como enlace.

Los nodos que integran una red inalámbrica ad-hoc se encuentran en un estado pasivo o activo. Al momento de que un nodo realiza la tarea de encaminamiento

sufre un mayor consumo de energía disminuyendo el rendimiento del dispositivo. Así, participaciones inequitativas de los nodos resulta en una desventaja de las redes ad-hoc [38].

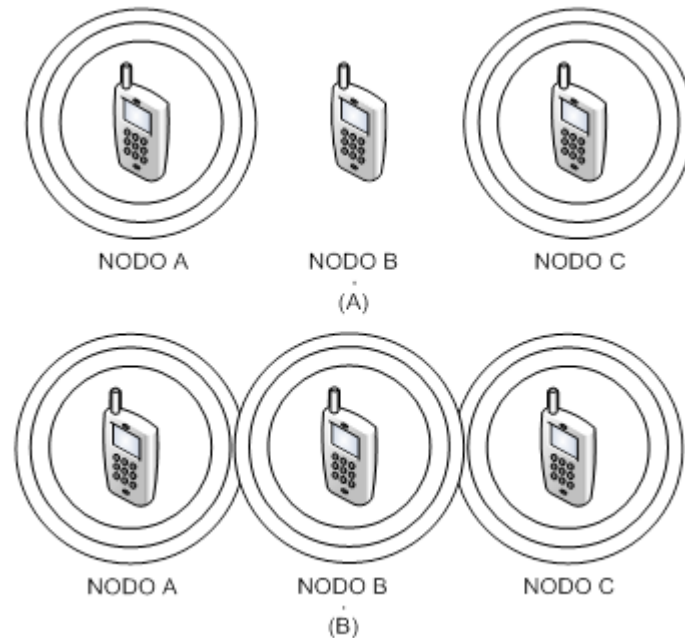


Fig. 2. Estructura de la red inalámbrica Ad-hoc. (A) El nodo A no puede establecer un enlace de comunicación con el nodo C, (B) El nodo B realiza la tarea de enrutador en el enlace de comunicación del nodo A y nodo C.

Las redes MANET, VANET, y la de sensores son un ejemplo de los diferentes tipos de redes inalámbricas ad-hoc en proceso de investigación. En lo que sigue de esta capítulo analizaremos las dos primeras por su importancia para nuestro trabajo.

2.2.4. Red Ad-hoc: MANET

Las redes MANETs son una subclase de red ad-hoc, cuyos nodos o dispositivos terminales se encuentran en constante movimiento. Para enlazar dos puntos comunicantes, los nodos intermedios toman la función de enrutadores. Para ello, estos nodos pasan de un estado pasivo a uno activo [22]. Entre las características de estas redes podemos mencionar:

- **Variabilidad en los enlaces de comunicación:** El radio de cobertura de los nodos hace que dos dispositivos distantes se comuniquen a través de terminales intermedios. Para ello, se debe considerar, para cada enlace, aspectos como pérdidas de propagación, tasas de error de bit, entre otros. Estos factores influyen en la calidad de servicio esperada [22].
- **Mayor consumo de energía:** Los dispositivos electrónicos cuentan con una batería de consumo limitado. Cuando estos equipos forman parte de una red MANET, el consumo de energía aumentará y por consiguiente disminuirá el tiempo de trabajo del dispositivo por parte del usuario.
- **Topología Dinámica:** Debido al movimiento de los nodos, la topología de red varía en forma constante y aleatoria. Así, protocolos de enrutamiento de redes fijas o WLAN (OSPF, RIP, EIGRP o BGP) no son de utilidad en una red MANET. Se requiere, por tanto, protocolos más flexibles [22].

2.2.5. Protocolos de Encaminamiento

La topología dinámica de la red MANET requiere el uso de protocolo específicos de encaminamiento. Es por esto que la Internet Engineering Task Force (IETF) desde 1997 estudia y desarrolla protocolos para MANET. Estos pueden ser clasificados en tres grupos: Protocolos Proactivos, Protocolos Reactivos y Protocolos Híbridos [22]. Véase la Fig. 3.

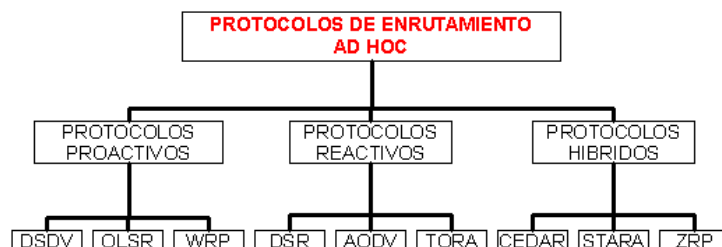


Fig. 3. Organigrama de protocolos de enrutamiento ad-hoc [10].

- a. **Protocolos Proactivos:** En un protocolo proactivo la información de las rutas posibles a través de la red son periódicamente difundidas por los nodos. El objetivo es tener la información necesaria para calcular la ruta antes de que se necesite. Debido a los cambios constantes en la topología, la carga de mensajes de actualización puede ser alta lo que provoca un aumento en el consumo de ancho de banda y energía de los dispositivos.

- b. **Protocolos Reactivos:** El protocolo reactivo determina una ruta solo cuando es necesario. Para ello, un nodo que necesita establecer una ruta hacia otro nodo inicia un proceso de descubrimiento. Una vez finalizada la transmisión, la ruta es mantenida en la tabla de encaminamiento por un periodo de tiempo determinado, luego de ello si no es activada se elimina. Una dificultad con este tipo de protocolos es la falta de actualización de las tablas de encaminamiento, lo que produce un aumento en el tiempo de descubrimiento de ruta.

- c. **Protocolos Híbridos:** Los protocolos híbridos combinan las principales características de los protocolos proactivos y protocolos reactivos. En las redes MANET se los utiliza con una topología jerárquica, en donde se forman celdas de varios nodos. Las rutas de transmisión de datos se realizan de manera interna y externa. Las rutas internas de las celdas utilizan las características de protocolos proactivos. Las rutas externas de las celdas utilizan las características de protocolos reactivos [22].

2.3. Redes Vehiculares Ad-hoc: VANET

En la actualidad, el avance y el continuo desarrollo tecnológico han impulsado el crecimiento de ciudades basadas en sistemas de comunicación inteligentes. Así, los nuevos sistemas de transporte automatizado [26] y los avances en sistemas de información en tiempo real [27] permiten avizorar el surgimiento de sistemas de transporte inteligentes. De igual forma, la versatilidad de los dispositivos de

comunicación y entretenimiento están convirtiendo las redes vehiculares en redes de comunicaciones. Si a ello se suma el interés en las redes ad-hoc —entre otras características por su facilidad de despliegue— muchas investigaciones apuntan a ofrecen nuevos servicios de comunicaciones sin necesidad de usar la infraestructura de un tercero. Sin embargo, la mayoría de servicios ofrecidos hasta el momento se dan aprovechando la conectividad soportada por redes celulares o satelitales [8].

Por otra parte, las VANETs prometen ofrecer nuevos servicios de comunicaciones a un menor costo: difusión de información del estado del tránsito en tiempo real, llamadas y mensajes entre usuarios en la vía, acceso a internet a través de pasarelas públicas, transmisión de información de diagnóstico a bordo (OBD), entre otros [8]. Sin embargo, las características físicas de éste tipo de nodos tales como velocidad, movilidad, entre otras, provocan que la topología sea aún más variable que en las redes MANETs [10].

No obstante lo anterior, el despliegue de servicios de comunicación en las VANETs no ha sido explotado en su totalidad debido entre otros factores a los retos técnicos motivados por la movilidad de los vehículos y la variabilidad en la densidad vehicular [12]. Así, muchas investigaciones buscan desarrollar nuevos protocolos que se ajusten a estas características y permitan establecer una comunicación eficiente.

2.3.1. Descripción General de la Estructura VANET

Las redes vehiculares ad-hoc, en su forma general, constan de dos tipos de nodos: los nodos estáticos y los nodos móviles. Los nodos estáticos mejor conocidos como RSU (Road Side Unit), son elementos fijos, ubicados a lo largo de las carreteras, cuya función es enviar, recibir y retransmitir paquetes para incrementar el rango de cobertura de la red. En cuanto a los nodos móviles, son vehículos equipados con un

dispositivo electrónico conocido como OBU (On Board Unit) que le permite establecer una comunicación con otros vehículos o con las RSU [35]. (Véase la Fig. 4).

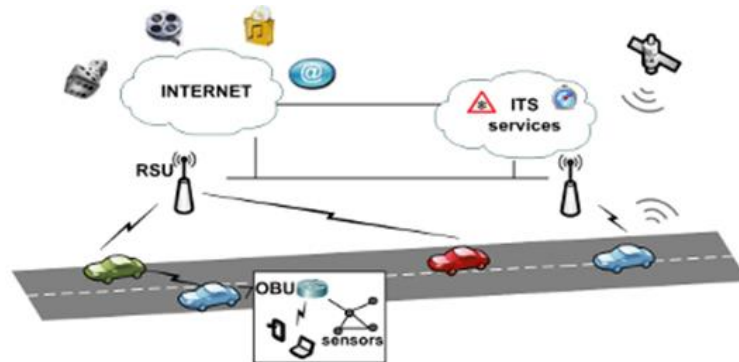


Fig. 4. Representación general de los elementos de una VANET [39]

2.3.2. Características y Consideraciones Importantes de una VANET

Una red vehicular ad-hoc se destaca por aspectos como la autonomía y movilidad de sus nodos. Cada nodo cuenta con la capacidad de recibir, transmitir, y procesar información como mejor lo considere; sin embargo, su movilidad hace que la topología de red sea altamente variable. Dado que los nodos pueden entrar o salir de una región, el protocolo de comunicación establecido entre los miembros de la red debe cubrir eficientemente estas consideraciones con el fin de garantizar una óptima calidad del servicio ofrecido [12].

Una ventaja en este tipo de redes es que la movilidad de los vehículos está limitada por factores como calles, semáforos, límites de velocidad, entre otras. Así mismo, la existencia de sistemas de transporte: buses, trenes, sistemas automáticos, etc., nos permite tener una mayor predictibilidad de la red con respecto a las MANETs. Otro punto importante a considerar, es que el rango de cobertura de los radios usados, usualmente va desde los 100 hasta los 300 metros, lo que ayuda en el intercambio eficiente de información entre vehículos [10].

2.3.3. Frecuencias de Operación de VANET

Las redes vehiculares ad-hoc, al igual que las redes MANET, por lo general trabajan en una determina banda de frecuencia que se localiza en las bandas no licenciadas de 2.4 GHz, 5 GHz y 914 MHz. Un gran número de investigadores enfocan sus estudios sobre el desarrollo de esta tecnología trabajando sobre la banda de 5.9 GHz [10].

En estas bandas se manejan velocidades aceptables para transmisión de información entre vehículos y dentro del perímetro establecido por el alcance y cobertura de la red inalámbrica [10].

2.3.4. Principales Protocolos de Enrutamiento

Los protocolos que se utilizan para controlar el encaminamiento de los paquetes a intercambiar dentro de una red vehicular ad-hoc mantienen ciertas características propias de los protocolos desarrollados para MANET. La diferencia se da en el hecho de que deben soportar una topología escalable y variable [8]. Entre los protocolos que funcionan en las redes MANET y que se consideran apropiados para ser utilizados para la simulación con VANETs destacan los protocolos DSDV, AODV y TORA [22].

a. Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV)

DSDV es un protocolo de vector distancia que se basa en el algoritmo de Bellman-Ford. Cada nodo mantiene su tabla de encaminamiento, la cual contiene un número de secuencia, numero de saltos y los posibles destinos [22].

La actualización de la tabla de encaminamiento se debe hacer en forma constante y difundirse mediante difusión. Al ocurrir un cambio en la topología, cada nodo asigna un número de secuencia que permite distinguir las nuevas rutas de las antiguas rutas, disminuyendo así la posibilidad de creación de bucles. En el caso de existir dos rutas con un mismo destino, se usará la ruta con el número de secuencia más reciente. Si dos rutas tienen un igual valor del número de secuencia se elige la ruta con menor número de saltos. En general, DSDV es un protocolo que se utiliza para redes con un número medio de nodos.

b. Ad-hoc On-Demand Distance Vector (AODV)

AODV es un protocolo basado en DSDV el cual fue diseñado para redes con gran número de nodos [22]. La tabla de encaminamiento mantiene el número de secuencia pero se utiliza un encaminamiento bajo demanda. Así, en la tabla de encaminamiento solo se almacena la información de los nodos que intervengan en la transmisión. En general, AODV disminuye el overhead de control y el gasto de memoria en comparación con DSDV. Además, las rutas se mantienen en cache mientras son necesarias.

c. Temporally Ordered Routing Algorithm (TORA)

TORA es un protocolo que establece los enlaces a través de redes de malla inalámbricas, se les asigna una dirección lógica para mantener un grafo dirigido y sin ciclos. El objetivo principal es minimizar la carga sobre la red al ocurrir un cambio en la topología de la red [22]. La deficiencia de este algoritmo es que no siempre elige la ruta más corta para el destino de comunicación.

2.3.5. Aplicaciones Principales en VANET

Las principales aplicaciones de este tipo de redes están asociadas a tres factores de interés relacionados a la seguridad, gestión de tránsito vehicular y aplicaciones comerciales. Las aplicaciones relacionadas a seguridad, permiten supervisar constantemente el trayecto, informar la proximidad de vehículos, el estado de la

carretera, curvas pronunciadas, entre otras. Las aplicaciones relacionadas a la gestión del tránsito tienen por objetivo mejorar la eficiencia del tránsito e incrementar el grado de comodidad de los conductores. Por su parte, las aplicaciones comerciales proporcionan al conductor y pasajeros entretenimiento mediante servicios como transmisión de audio y video, intercambio de mensajes en la carretera, acceso a Internet, entre otros [14].

Debido a que las VANETs constituyen una tecnología en desarrollo, sus aplicaciones están enfocadas principalmente a dar solución a los servicios de tránsito. Ello se consigue mediante el intercambio continuo de mensajes que permiten la cooperación entre vehículos y proveedores de servicio. La fiabilidad y la latencia son factores vitales en estas aplicaciones [8].

Dependiendo de la forma en la que se establece la comunicación dentro de una red vehicular, las aplicaciones disponibles variarán. Una clasificación se da en función de la capacidad de intercambiar mensajes entre vehículos (Vehículo a Vehículo, V2V) o empleando los nodos estáticos (Vehículo a Infraestructura, V2I) [31]. La Fig. 5, resalta los tipos de servicios que se pueden ofrecer en función de la forma en la cual se comuniquen los nodos que integran la red vehicular.



Fig. 5. Representación de los tipos de comunicación en una VANET [23]

a. Servicios y Aplicaciones entre Vehículos

El intercambio de información de seguridad entre vehículos es uno de los principales tópicos de investigación en la actualidad. Su aplicación permitirá informar a los usuarios sobre congestiones, accidentes, reparación de vías, zonas de riesgo, entre otros [20]. Así los vehículos colaboran en el envío de esta información para mejorar la circulación vehicular. Lo fundamental en este tipo de servicios es garantizar que la información sea en tiempo real, oportuna y confiable [10].

b. Servicios entre Infraestructura-Vehículos

Este tipo de servicio usa radio bases, en puntos estratégicos dentro de la topología de la red, para alcanzar una mayor cobertura. Entre los servicios disponibles se incluye: información sobre clima, zonas de riesgo: inundaciones, desastres geográficos; estado de tránsito, acceso a internet, entre otras [10]. Un tipo de servicio derivado de este tipo de conexiones pero con aplicaciones más complejas como por el ejemplo el acceso a internet tarifado se conoce como servicio basado en pasarelas o portales y está en fase de desarrollo en la actualidad.

2.3.6. Futuras Investigaciones

El interés de investigación en el campo de las VANETs se centra en aspectos como seguridad, entretenimiento, control y gestión del tránsito vehicular y en aplicaciones de tipo comercial. Estos estudios han arrojado diferentes servicios entre los cuales se destacan los siguientes [34]:

- Gestión de tránsito vehicular para evitar congestiones, comunicar sobre advertencias de peligro de colisión y gestionar el libre parqueo.
- Brindar información sobre el estado de la carretera.
- Informar acerca de accidentes ocurridos o vehículos detenidos en la vía.
- Proveer información turística en carreteras, brindar información sobre gasolineras, restaurantes, hoteles próximos y demás servicios

- Solicitar ayuda en carreteras en caso de asaltos o zonas de riesgo y brindar respuestas de emergencia y apoyo a las autoridades.
- Entretenimiento, comodidad y seguridad a los conductores, a todo ello se le suma una conectividad continua a Internet.

2.4. Virtualización en Redes Vehiculares

En [9] Lynch et.al., propone el uso de VMNs para hacer frente a la variabilidad en la topología de las redes MANETs. Los VMNs son nodos abstractos que se mueven de una manera predeterminada y que permiten generar una capa virtual sobre la cual desplegar servicios de comunicaciones. Cada VMN es soportado colectivamente por nodos reales que viajan cerca de su punto de localización aunque la dirección de movimiento de los VMNs no es determinada por la dirección de movimiento de los nodos reales. Así, los VMNs pueden encubrir mucha de la incertidumbre de los ambientes MANETs y por tanto simplificar el diseño y operación de los servicios de comunicaciones. En [9], Lynch et.al., proponen la VNLayer, una capa abstracta para MANET que comprenden dos tipos de entidades:

- **Servidores virtuales predictibles:** ubicados sobre una rejilla en localizaciones fijas.
- **Nodos clientes no predictibles:** estos corresponden a los nodos móviles reales.

La VNLayer divide el escenario en una rejilla, (véase la *Fig. 6*) tal que los desarrolladores pueden escribir algoritmos para ser implementados sobre nodos estables (virtuales) más que sobre nodos no predecibles (reales). Por supuesto, es necesario que existan nodos reales para dar soporte a los nodos virtuales. Además, la VNLayer realiza tareas como la determinación de la ubicación del nodo y revisa el estado de sincronización de los nodos virtuales.

En [42], Wu utiliza a la VNLayer como una herramienta de encaminamiento reactivo. En su trabajo de investigación implementa el protocolo de enrutamiento VNAODV y lo compara con el protocolo AODV. En sus conclusiones establece que VNAODV puede entregar más paquetes, genera menos tránsito sobre la red y crea más rutas de encaminamiento sobre redes MANETs densas.

Otros trabajos de aplicaciones en coordinación de movimiento y en procesos de encaminamiento de paquetes, todos en redes MANETs, son presentados en [9] y [36]. En todos estos trabajos, los investigadores hacen uso de un ambiente de simulación llamado VNSim [43], el cual es basado en el simulador de red ns-2. La dificultad de trabajo con este simulador en ambientes vehiculares, es su falta de disponibilidad por parte de los autores y su uso restringido a modelos de movimiento no adecuados para redes vehiculares.

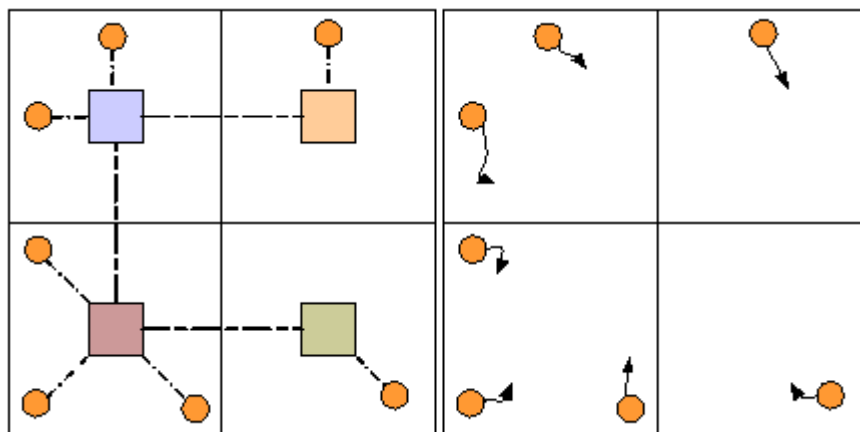


Fig. 6. La capa virtual. La figura de la izquierda muestra la rejilla para la implementación de una capa virtual, en donde los rectángulos representan los nodos virtuales, y los pequeños círculos los nodos reales. La figura derecha nos muestra los nodos físicos reales [36]

Como se indica en [42], [36], la propuesta de [9] ha sido estudiada por varios autores; en todos ellos se hace uso de una disposición de nodos virtuales estáticos y en rejilla. Una nueva propuesta de uso de los VMNs se presenta en [8]. El objetivo es estudiar las potenciales ventajas de tener los VMNs sobre VANETs, considerando la

ventaja de que los movimientos de los vehículos no son aleatorios; por lo contrario, son limitados a las calles, y la variación de la densidad vehicular tiene un comportamiento estadístico en el tiempo y espacio.

Adicionalmente, Bravo, et.al [7], proponen el concepto de circuito virtual virtual (VVC) definido como ruta sobre una secuencia de VMNs, para entregar paquetes entre sus puntos extremos, véase la Fig. 7. Imitando el concepto de circuitos virtuales en las redes tradicionales, estos autores proponen el uso de circuitos virtuales sobre nodos virtuales aprovechando la mayor estabilidad de estos en redes móviles vehiculares.

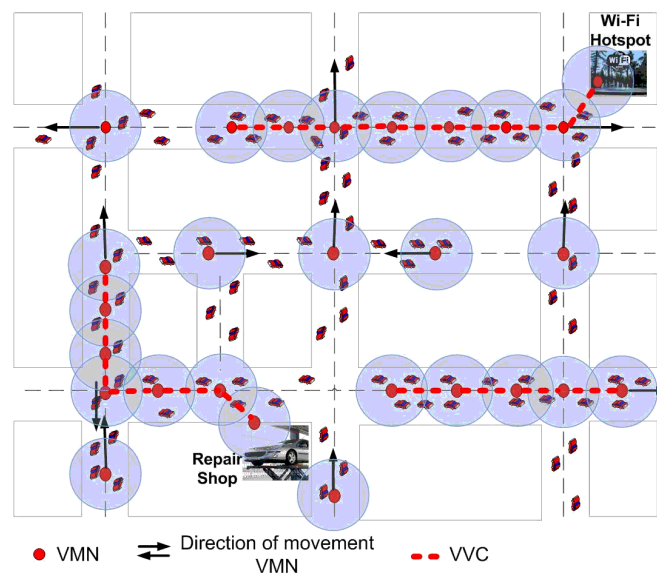


Fig. 7. La figura nos muestra la implementación de los VVC sobre los nodos móviles virtuales [7].

Así mismo, en [6] se propone el concepto de circuito virtualizado (VLC) como una ruta predeterminada sobre una secuencia de VMNs, que puede ser usado para soportar entrega de paquetes de una o varias comunicaciones. El objetivo es usar los VLCs tal que ellos puedan soportar diferentes modos de comunicaciones: anycast, broadcast o multicast; simplex half-duplex o full duplex; con o sin diferenciación de servicios para los diferentes flujos de comunicación. Muchas de las operaciones de los VLCs dependen de los flujos de tránsito y densidades esperadas en diferentes tiempos a través de las áreas por ellos cubiertas. Por ello, la definición de los VLCs

es manejada por modelos de tránsito que capturan los principales flujos de tránsito en una ciudad, las horas de tránsito pico, entre otras.

2.5. Simuladores de Redes de Telecomunicaciones

Al trabajar con una herramienta de simulación existen ciertas consideraciones: la utilización de un modelo de simulación correcto, código fuente libre, uso de licencias, compatibilidad con el sistema operativo de la PC, entre otras. Todas ellas, sin descuidar factores como escalabilidad y rendimiento, ayudan a determinar qué tan útil puede resultar esa herramienta.

Entre los diversos tipos de simuladores, los simuladores de red constituyen una poderosa herramienta para el análisis y mejora de las tecnologías de red. Este tipo de simuladores nos permiten modelar diferentes características de las redes de telecomunicaciones, realizar investigaciones y perfeccionar el aprendizaje en puntos específicos tales como: encaminamiento, calidad de servicio, enlaces inalámbricos o cableados, comparación y análisis de protocolos, simulación de comunicaciones sobre canales de transmisión realistas, evaluar pérdidas, teoría de colas, entre otras. Además, nos ofrecen facilidades para medir la eficiencia de la red. A continuación describiremos algunos simuladores de software libre y sus principales características, con la finalidad de elegir el simulador que se adapte al tema de investigación propuesto.

2.5.1. Network Simulator 2

Network Simulator 2 (ns-2) —originalmente desarrollado dentro del proyecto VINT— es un simulador de eventos discretos de uso extendido en la comunidad académica para la investigación en redes de computadores y telecomunicaciones

[28]. Basado en software libre, ns-2 está disponible en múltiples plataformas como Windows o Linux. Además, esta herramienta soporta simulaciones de redes cableadas e inalámbricas, protocolos de encaminamiento y multidifusión, entre otras [17]. Su estructura se fundamenta en dos lenguajes de programación: C++ como núcleo principal y una versión orientada a objetos del lenguaje Tcl (Tool Command Language) mejor conocida como OTcl (Object-oriented Tool Command Language). El lenguaje OTcl permite interactuar al usuario con el simulador mediante un script en el que detalla los parámetros de la topología a simular. La ventaja de tener un simulador de red de este tipo, radica en el hecho de que a través de las librerías implementadas en C++ se puede modelar el comportamiento de los nodos mientras que usando los scripts en OTcl se puede controlar el proceso de simulación. [4].

2.5.2. Objective Modular Network Testbed in C++ (OMNeT++)

OMNeT++ es un simulador —gratis y sin fines de lucro— de eventos discretos. Se basa en C++ y está orientado a la construcción de ambientes de simulación para redes de telecomunicaciones. Entre las herramientas disponibles se incluyen: simulación para redes cableadas e inalámbricas, análisis de colas, paquetes internos como INET (Integrated Network Enhanced Telemetry) que contiene una extensa colección de modelos de protocolos de Internet. Además, OMNeT++ ofrece un IDE —basado en Eclipse— para la ejecución gráfica. Existen extensiones para simulación en tiempo real, lenguajes alternativos de programación (Java, C#), y varias otras funciones [41].

Las simulaciones con OMNeT++ consisten en llamadas a módulos simples implementados en C++. Varios módulos simples se pueden unir y formar un módulo compuesto. Una simulación de red en OMNeT++ se hace por medio de NED (Network Description) —el lenguaje de descripción de la red— y está implementada como un módulo compuesto que comprende otros módulos [15]. Si bien OMNeT++ es un simulador bien organizado, de fácil uso y bastante flexible; los usuarios deben

implementar una cantidad considerable de código si desean obtener los resultados esperados [32].

2.5.3. Simuladores Escritos en Java

Existen múltiples herramientas de simulación escritas en lenguaje Java debido a las prestaciones y ventajas que ofrece este lenguaje de programación, algunas aplicaciones permiten enlazar simuladores como ns-2 con simuladores de tránsito como SUMO tal es el caso de una aplicación conocida como MOVE, sin embargo se ha optado por describir de manera breve tres simuladores destacados, que son:

- **JiST (Java in Simulation Time):** simulador de red de alto rendimiento orientado a objetos, basado en componentes y de eventos discretos. Se utiliza para la simulación de redes inalámbricas que incorporan dispositivos con sensores reales. Se ejecuta en una máquina virtual Java, su diseño permite superar tiempos de ejecución optimizando recursos como tiempo de simulación y consumo de memoria [2].
- **J-Sim (Java-based Simulator):** simulador gratuito orientado a objetos, basado en componentes y de eventos discretos. Está disponible para múltiples plataformas (Windows, Linux). Sus diferentes componentes: nodos, enlaces, protocolos, entre otros, se conectan mediante puertos. Se puede ampliar su estructura creando nuevos componentes o modificando los existentes redefiniendo los atributos y métodos de las subclases. J-Sim se enfoca en la fisiología y la biomedicina, sin embargo, su motor de cálculo permite que sea aplicable a una amplia gama de dominios científicos como las telecomunicaciones [24].
- **KivaNS (Kiva Network Simulator):** aplicación gratuita y de código abierto. KivaNS funciona en diferentes plataformas, como GNU/Linux o Windows. Se utiliza para simular el encaminamiento de paquetes, analizarlos y

especificar esquemas de redes de datos. Este simulador está pensado para evaluar parámetros de carga, rendimiento, etc. [13].

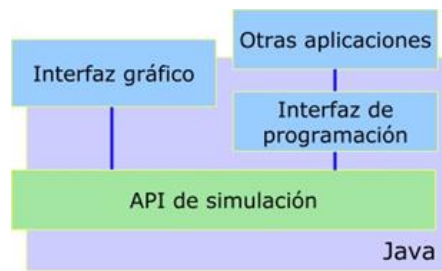


Fig. 8. Estructura del simulador KivaNS [13].

Muchas de estas aplicaciones en Java, al igual que otras programadas en lenguaje Python, manejan una estructura similar a la observada en la *Fig. 8*, que consiste en una interfaz gráfica, código fuente que puede ser modificado, y compatibilidad con aplicaciones externas para dar una mejor performance al simulador. Entre todos estos simuladores nosotros hemos optado para la elaboración el ambiente de simulación ns-2 debido a las múltiples ventajas que ofrece y que fueron antes mencionadas.

2.6. Simuladores de Tránsito Vehicular

Los simuladores de tránsito son programas que permiten la modelación de escenarios de tránsito vehicular liviano y pesado, así se determina el desempeño de las variables involucradas. Por lo que, el simulador de tránsito vehicular para el desarrollo de este trabajo, debe permitirnos operar las siguientes características [16]:

- **Variabilidad de escenarios de tránsito vehicular:** permitir la creación de escenarios de simulación o poder exportar escenarios reales de bases de datos.
- **Modelar a cada vehículo individualmente:** permitir la modelación de velocidad, aceleración, cambio de ruta y localización individual.

- **Proveer accesibilidad a la información generada por las simulaciones:** El simulador de tránsito debe ofrecer los datos de movilidad y estadísticas requeridas para su enlace con el simulador de red.

A continuación describiremos algunos simuladores de software libre, sus principales características, y al final realizaremos la elección del mejor simulador que mejor se adapte a nuestra necesidad.

2.6.1. Simulation of Urban MObility (SUMO)

SUMO es un software de código abierto que simula el tránsito vehicular relacionándolo con un entorno microscópico, véase la *Fig. 9*, esto significa que cada vehículo es modelado individualmente [3]. SUMO se desarrolló en el Instituto de Sistemas de Transporte en el Centro Aeroespacial de Alemania. Entre otras, las características que nos ofrece son:

- Alta portabilidad, se puede realizar diferentes algoritmos dinámicos por parte del usuario utilizando el lenguaje de programación en C++.
- Se modela diferentes tipos de vehículos (autos, camiones, buses). El movimiento de un vehículo se realiza con rutas microscópicas en un espacio continuo y en tiempo discreto.
- Las calles pueden ser de varios carriles con semáforos en las intersecciones y diferentes reglas para los sentidos de las vías.
- Velocidad de ejecución rápida (hasta 100.000 actualizaciones por vehículo/seg en una máquina de 1 GHz).
- Interoperabilidad con otras aplicaciones, se puede importar a VISUM, Vissim, Shapefiles, OSM, RoboCup, MATsim, openDRIVE y XML-Descriptions.
- Paquetes para Windows y Linux.
- Interfaz gráfica para el usuario.

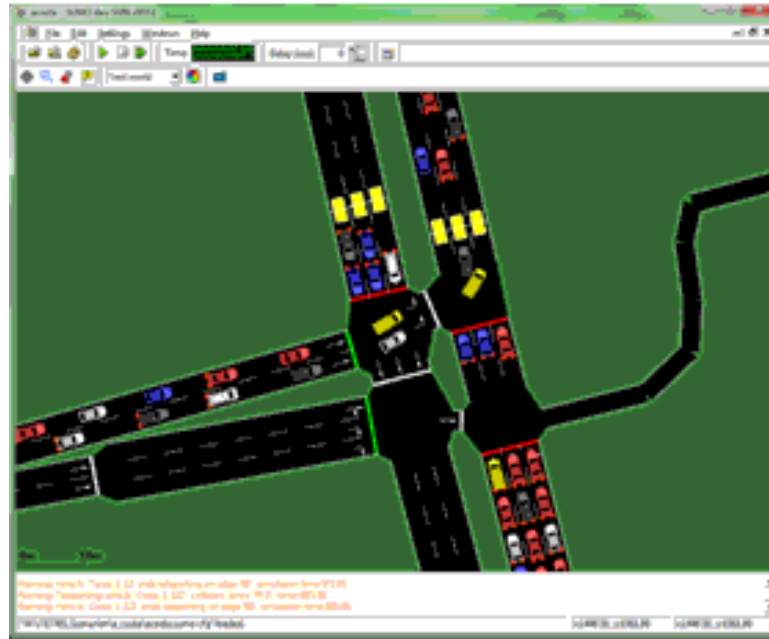


Fig. 9. Entorno gráfico del simulador de tránsito SUMO [3].

2.6.2. VanetMobisim

VanetMobisim es un simulador de tránsito vehicular para entornos macroscópicos y microscópicos, se desarrolla en Java, véase la *Fig. 10*, como una extensión del programa Mobility Simulation Environment (CanuMobiSim) por el Institute of Parallel and Distributed Systems (IPVS) de la Universidad de Stuttgart [18]. Entre otras, algunas de características que nos presenta son:

- En nivel macroscópico puede importar mapas en formato Geographical Data File (GDF) o crear mapas aleatorios.
- Soporte para vías de varios carriles con flujos bidireccionales por separado, variabilidad de velocidad y señales de tránsito para intersecciones de vías.
- Utiliza archivos con formato XML para su simulación.
- Facilidad de integración con simuladores de redes de comunicación como son QualNet y ns-2.

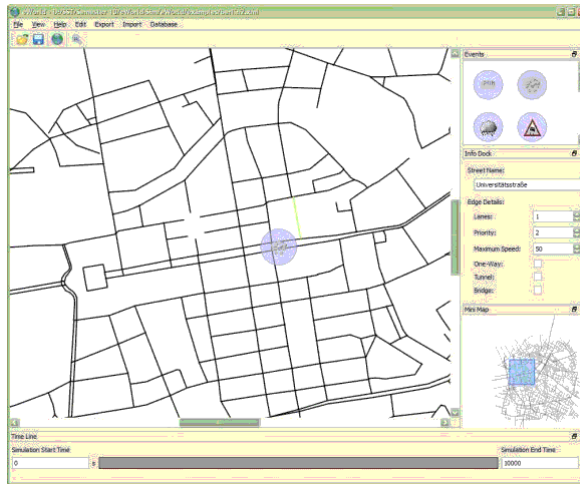


Fig. 10. Entorno gráfico del simulador de tránsito VanetMobisim [18].

2.6.3. Street Randow Waypoint (STRAW)

STRAW es un simulador que contiene constantes de movimiento provenientes del análisis del sistema de tránsito de varias ciudades de E.E.U.U., fue desarrollado por Aqua Lab Project como una parte del proyecto C3 (Car-to-Car cooperation for vehicular ad-hoc networks) para que sea parte del simulador de redes de datos JIST/SWANS, véase la Fig. 11.

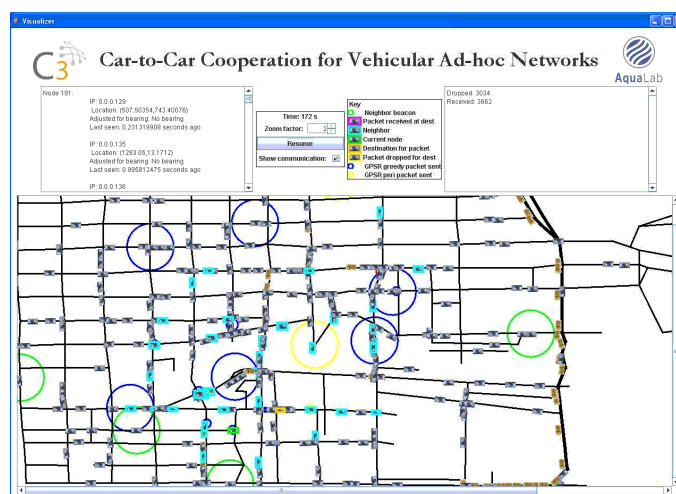


Fig. 11. Entorno gráfico del simulador de tránsito STRAW [1].

Algunas de las características que nos presenta son:

- Uso de la herramienta de simulación JAVA In Simulation Time (JIST), que permite realizar simulación de eventos discretos sobre una máquina virtual de JAVA (JVM).
- SWANS permite que la implementación de protocolos de transporte y enrutamiento se lo junto a STRAW.

En base a las características de los simuladores citados, se analizó a cada uno, para comparar y seleccionar el que se adapte mejor a las características requeridas de nuestro proyecto. Esto nos lleva a realizar una comparación entre los simuladores de tránsito con la *Tabla 1*.

SIMULADORES DE TRÁNSITO	
SIMULADOR	CARACTERISTICAS
SUMO	<ul style="list-style-type: none"> • Software Libre. • Variabilidad de escenarios de tránsito vehicular. • Modelación de vehículos individualmente. • Accesibilidad a la información generada por sus simulaciones • Importación de archivos a simuladores de redes.
VanetMoviSim	<ul style="list-style-type: none"> • Software Libre. • Variabilidad de escenarios de tránsito vehicular. • Accesibilidad a la información generada por sus simulaciones. • Importación de archivos a simuladores de redes. • Nivel de programación más complejo que otros simuladores de tránsito.
STRAW	<ul style="list-style-type: none"> • Software Libre. • Escenarios de tránsito vehicular basado en el sistema de tránsito de ciudades de E.E.U.U. • Importación de información generada por las simulaciones solo al simulador de redes JIST/SWANS.

Tabla 1. Análisis de los Simuladores de Tránsito

El software que se adapta a las necesidades de nuestro proyecto es SUMO, éste nos permite la creación de escenarios de simulación o exportación de escenarios reales de una base de datos; además, nos da a elegir las características de movilidad para cada vehículo, nos provee la accesibilidad a los resultados de las simulaciones y nos da permite importar sus archivos al simulador de redes.

3. ESTRUCTURA DEL SIMULADOR

3.1. Introducción

La sección 2.4, nos presentó a la VNLayer como una capa abstracta para MANETs, la cual comprende dos tipos de entidades: servidores virtuales sobre una rejilla y nodos móviles reales como nodos clientes [9]. Así, la VNLayer realiza tareas como la localización del nodo y el análisis del estado de sincronización de los nodos virtuales.

En [42], Wu utiliza a la VNLayer como una herramienta de encaminamiento reactivo. Para el proceso de simulación, Wu presenta un simulador conocido como VNSim; su estructura está determinada por tres agentes: JOIN, VNS y VNC. El agente JOIN ejerce las funciones de seguimiento y elección del líder. Para ello, utiliza dos tipos de mensaje: REGION y LEADER. El Agente VNS sincroniza el estado de los nodos non-leader con el estado del nodo leader. Por su parte el VNC, en conjunto con el VNS, permite adaptar los paquetes de la capa de aplicación para su ingreso en la capa virtual y su posterior envío a las capas inferiores.

En [5], Bravo propone el estudio de las potenciales ventajas de tener los VMNs sobre VANETs, considerando la ventaja de que los movimientos de los vehículos son limitados a las calles, y la variación de la densidad vehicular tiene un comportamiento estadístico en el tiempo y espacio. Acorde a las propuestas de [5], [42] se implementó un software de simulación basado en ns-2 conformada por tres agentes: Master, Members y Constancy:

- Agente Master realiza un rastreo del movimiento nodo físico y elige entre los nodos físicos de la región un *Líder*. El seguimiento del nodo determina su ubicación, velocidad y dirección, características que pueden ser determinadas

de la plataforma de ns-2. La elección del *Líder* es manejado por el algoritmo diseñado en [42]. Cuando un nodo ingresa a una región emite una solicitud para ser *Líder*. Luego de un tiempo determinado y si no existe respuesta, se elige como líder. El *Líder* envía periódicamente mensajes hacia los otros nodos de la región, tal que una nueva elección puede ser iniciada después de un periodo anormal de actividad del *Líder*. El *Líder* puede renunciar a su condición, si éste probablemente abandone la región.

- Agente Constancy coordina los agentes de la capa virtual. El agente inicia su función al momento de recibir un paquete desde el Master. Una vez que un *Líder* es seleccionado en la región, éste coordina las acciones del agente Members [42].
- Agente Members inicia su función al momento en que el Constancy notifique la elección del leader [42].

Es así que cada nodo móvil contendrá tres agentes. La localización de los nodos seguirá una estructura de rejilla similar a la utilizado por [42] para la conformación de la capa virtual. Para el escenario de simulación en redes vehiculares se hará uso del simulador SUMO, que contempla la herramienta TraceExporter para su integración con ns-2.

El capítulo está estructurado de la siguiente manera. La sección 3.2 describe la arquitectura del software implementado. La sección 3.3 identifica las clases, atributos y métodos del simulador basado en el algoritmo de [42]. La sección 3.4 detalla el enlace del simulador de red y el simulador de tránsito vehicular.

3.2. Bloques Funcionales

El objetivo de esta tesis es implementar un ambiente de simulación² de nodos móviles virtuales en escenarios de redes vehiculares. Para este propósito, nos

²A lo largo de todo este documento nos referiremos como ambiente de simulación, a nuestro simulador desarrollado.

basaremos en las investigaciones preliminares desarrolladas por nuestro grupo de investigación [5], [8] y en el diseño propuesto en [42], véase la *Fig. 12*. Nuestro trabajo se enfoca en implementar esta propuesta a través de software libre de simuladores de redes de comunicaciones y de tránsito vehicular. Sin embargo, su alcance se limita a la capa de virtualización más no al desarrollo de protocolos de comunicación entre nodos móviles virtuales. En lo que sigue de esta sección, se detalla los bloques funcionales del simulador implementado.

3.2.1. Módulo de Control de Tránsito Vehicular: sirve para generar topologías con vehículos, simular su comportamiento y analizar los resultados. Para ello, tres sub elementos son necesarios:

- a. **Generador de Tránsito Vehicular:** Este módulo es el encargado de adecuar los patrones de movimiento generados por el simulador de tránsito vehicular (SUMO) al formato propio del simulador de red (ns-2).
- b. **Simulador de Tránsito Vehicular:** Este módulo, basado en SUMO, permite generar las diferentes topologías para ambiente vehiculares. El archivo que resulta de éste módulo es remitido al generador de tránsito vehicular para su adecuación a los formatos necesarios.
- c. **Analizador de Tránsito Vehicular:** Analiza las principales características y estadísticas del tránsito vehicular generado por el simulador (SUMO), con la finalidad de proporcionar información suficiente a los diferentes elementos de la capa de virtualización.

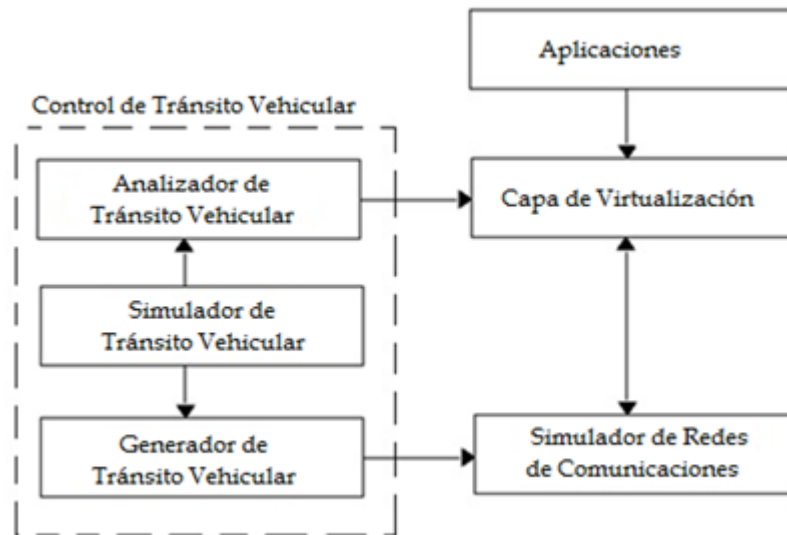


Fig. 12. Diagrama de bloques funcionales del ambiente de simulación propuesto [8]

3.2.2. Módulo de Aplicación: basada en el simulador de red, es el encargado de generar las diferentes aplicaciones a desplegarse sobre la capa virtual. Este módulo se basa completamente en la plataforma ns-2, nos permite simular aplicaciones como: transferencia de archivos, entretenimiento, conectividad a internet, operaciones de emergencia y seguridad, conectividad instantánea simulada por DHCP, enviar información sobre el estado del tránsito, entre otras.

3.2.3. Capa de Virtualización: Implementada sobre la plataforma de ns-2, es la encargada de gestionar las diferentes funcionalidades de los nodos móviles virtuales. Esta capa de virtualización interactúa con la capa de aplicación y la plataforma ns-2 [43]. Su función es crear y gestionar los nodos móviles virtuales y coordinar su relación con la capa de aplicación y el enlace con los nodos físicos. Para su implementación, tal como se aprecia en la Fig. 13, se requiere tres agentes [42]: Join, VNServer y VNClient.

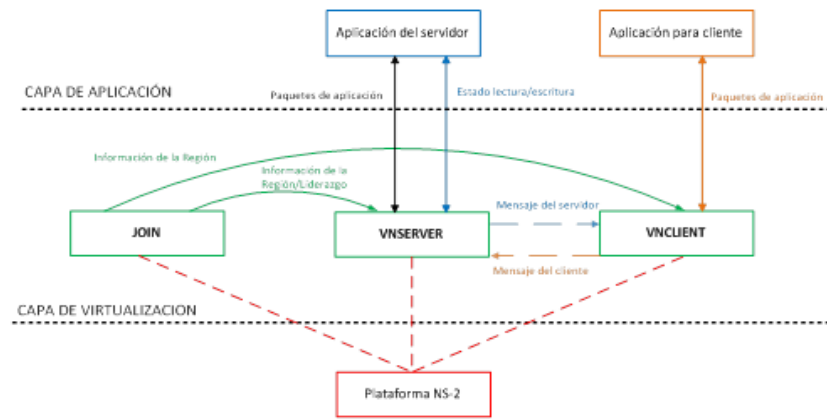


Fig. 13. Diagrama de bloques funcionales para describir la capa de virtualización [43]

El agente Join implementa un módulo para determinar la ubicación del nodo y un módulo de control para la elección del nodo *Líder*. Comunica cambios de región y los cambios de estado del nodo *Líder* a los otros dos tipos de agentes, utilizando para ello dos tipos de mensajes: mensaje con información de la región y mensajes con información del estado del *Líder* [42].

El agente VNServer (VNS) actúa como interfaz para las aplicaciones y procesos del servidor. También almacena en un búfer los mensajes de respuesta del servidor y mantiene el estado de la aplicación sincronizada con el estado del nodo del servidor [42].

El agente VNClient (VNC) actúa como interfaz entre una aplicación y el proceso de un cliente [42].

3.2.4 El Simulador de Redes de Comunicaciones: ns-2 ha sido seleccionado, entre otras cosas, porque permite realizar modificaciones de su estructura interna con el fin de adecuarla a las situaciones que deseamos simular.

Tal como se aprecia en la Fig. 14, ns-2 está escrito en C++ y OTcl. Un usuario implementa el escenario de simulación a través de fichero tipo script,

escrito en Tcl, que ingresa al simulador. Ns-2 mediante su intérprete y sus librerías internas entrega resultados como producto de la simulación. En función de la información que se desea analizar, existen varias herramientas que ayudan al procesamiento de los datos de salida, entre ellas NAM, Xgraph, GNUPlot, Matlab, AWK, entre otros [30].

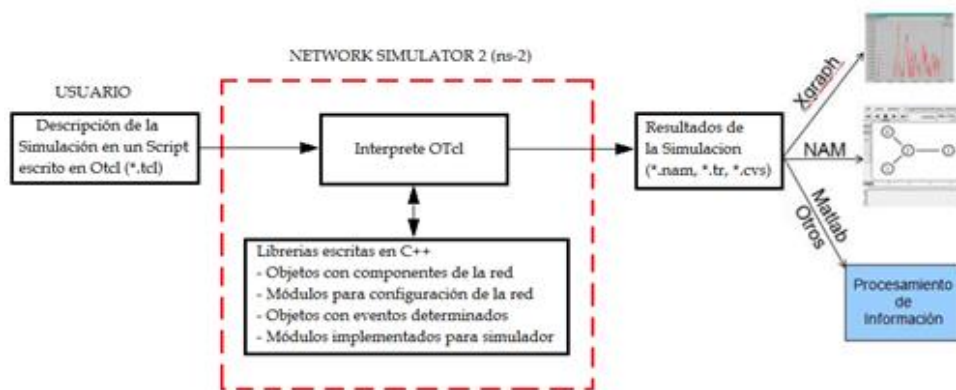


Fig. 14. Bloques funcionales para describir el funcionamiento de ns-2 [30]

3.3. Identificación y Diseño de las Clases, Atributos y Métodos del Ambiente de Simulación.

El ambiente de simulación para la capa de virtualización se implementó con las propuestas de [5] y [42]. El simulador está implementado en la plataforma ns-allinone-2.34, que data del 18 de Junio de 2009. Ns-2 proporciona los bloques de simulación para las capas física y enlace de datos. De igual forma, facilita las características para la simulación de nodos móviles.

En la Fig. 15, se muestra la estructura del ambiente de simulación implementado sobre ns-2. La capa virtual realiza la tarea de colocar en cada nodo físico tres agentes de ns-2: agente Master, agente Constancy y agente Members.

El Master es el encargado de la gestión del *Líder*: selecciona un *Líder* de entre los nodos físicos que están en la región cubierta por el nodo virtual y hace un

seguimiento a los cambios de región y estado. Para ello, asigna un número a cada nodo físico para identificarlos en la capa virtual. Mediante esta identificación, el Máster puede intercambiar información del proceso de selección y de los cambios de región del nodo *Líder* a los otros dos agentes Constancy y Members. Todo ello, para permitir la sincronización entre los tres agentes [42].

El agente Constancy sincroniza a los tres agentes. Para ello en cada nodo este agente emite un mensaje de solicitud de sincronización (Solicita_sincronia) y el Members responde con un mensaje de confirmación de sincronización (Confirma_sincronia). Por su parte, el agente Master envía dos tipos de mensajes al agente Constancy: el primer tipo de mensaje es relacionado a la región, este mensaje (SMS_REGION) informa sobre el cambio de región del nodo actual. El segundo tipo de mensaje (SMS_LIDER) al Constancy, que hace referencia a mensajes de la confirmación de nuevo líder.

El agente Members empieza a cumplir su función cuando recibe un mensaje con la información de la Región en la cual se encuentra el nodo y que fue enviado desde el agente Master.

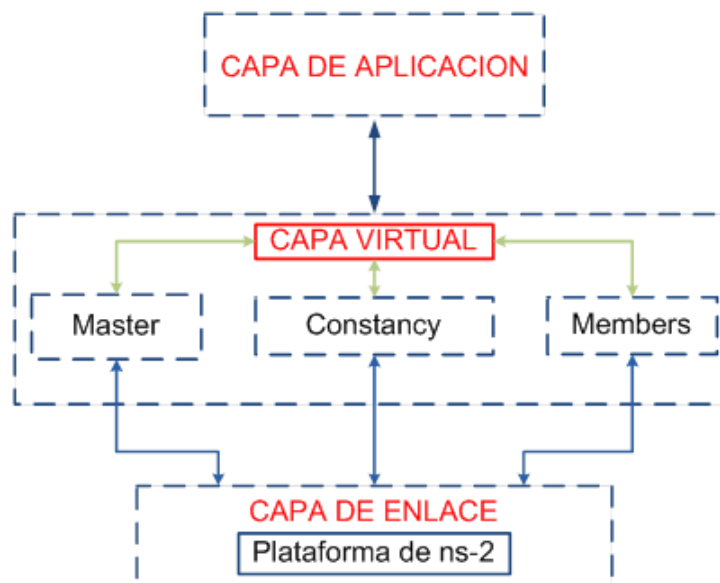


Fig. 15. Arquitectura del Ambiente de Simulación en la plataforma ns-2 [2].

Para la implementación de estos agentes se ha definido varias clases con sus atributos y métodos, las cuales están contenidas en diez archivos:

- **vncaza_pkt.h** archivo que contiene las cabeceras de paquetes a ser intercambiados entre agentes.
- **variables.h** archivo que contiene las variables a ser utilizadas en los agentes.
- **master.h / master.cc** archivos que contienen: los contadores de tiempo para los periodos de revisión de la ubicación del nodo, control del periodo de mensajes de elección del nodo *Líder* y control de envío de mensajes de confirmación de liderazgo. Enlaces entre el intérprete (TCL) y compilador (C++). Las clases para el funcionamiento del Master: *MasterAgent* que permite el control del agente.
- **constancy.h / constancy.cc** archivos que contienen: enlaces entre el intérprete y compilador, clases de Constancy: *ConstancyAgent* que realiza el control del agente.
- **members.h / members.cc** archivos que contienen: enlaces entre el intérprete y compilador, clases de Members: *MembersAgent* que realiza el control del agente.
- **resultados.h / resultados.cc** archivos que contiene las funciones de publicación de resultados de la capa virtual en un archivo digital.

La estructura física descrita por los archivos se puede apreciar de manera gráfica en los anexos [1], en donde se utiliza los diagramas Unified Modeling Language (UML). Además, en los anexos [2], se presenta el código en forma de documento. A continuación se describirá las estructuras, clases, funciones y constantes de cada uno de las clases desarrolladas.

3.3.1. Paquetes de la Capa Virtual

Los objetos en la clase *packet.cc* de ns-2 constituyen la unidad fundamental de intercambio entre los objetos de la simulación. Esta clase proporciona suficiente información para vincular un paquete a una lista, trabajar con colas de paquetes y referirse a un búfer de paquetes de datos [40].

Tal como se observa en la *Fig. 16*, un paquete de ns-2 está compuesto por una pila de cabeceras y un espacio opcional para datos [21]. Esto significa que cuando se crea un paquete por un agente se añade una pila compuesta por todas las cabeceras registradas. Para acceder a cualquiera de ellas se usa un valor de desplazamiento.

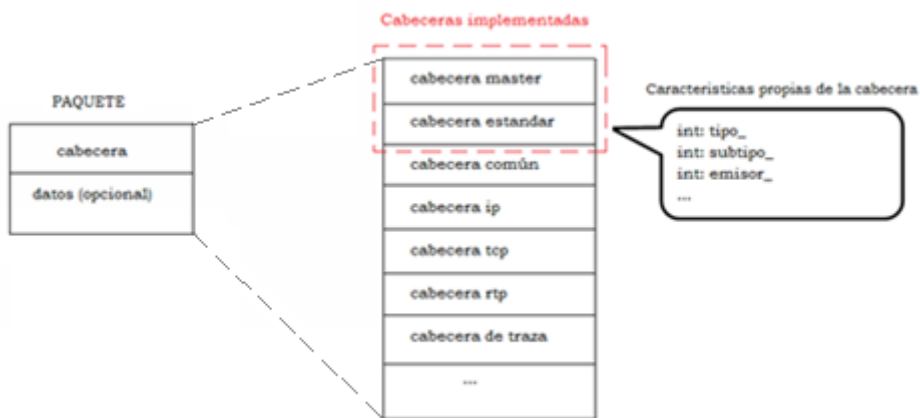


Fig. 16. Formato de un paquete de ns-2 [21].

Conforme a lo expuesto por Jiang Wu en [42], se definen las siguientes características propias de una cabecera común para un mensaje. La cabecera debe ofrecer información acorde con las siguientes especificaciones:

- *Tipo*: Permite identificar el tipo de paquete.
- *Subtipo*: Permite identificar el subtipo de paquete.
- *Región ID*: Identifica la región del emisor del paquete.
- *Emisor*: contiene la dirección del nodo físico que emite el mensaje.
- *Destino*: contiene la dirección del nodo físico que recibe el mensaje.

- *Tiempo_envio*: tiempo de envío que demoró la transmisión del paquete.
- *Hash*: un hash³ del estado del nodo virtual asignado al paquete en el momento del envío.

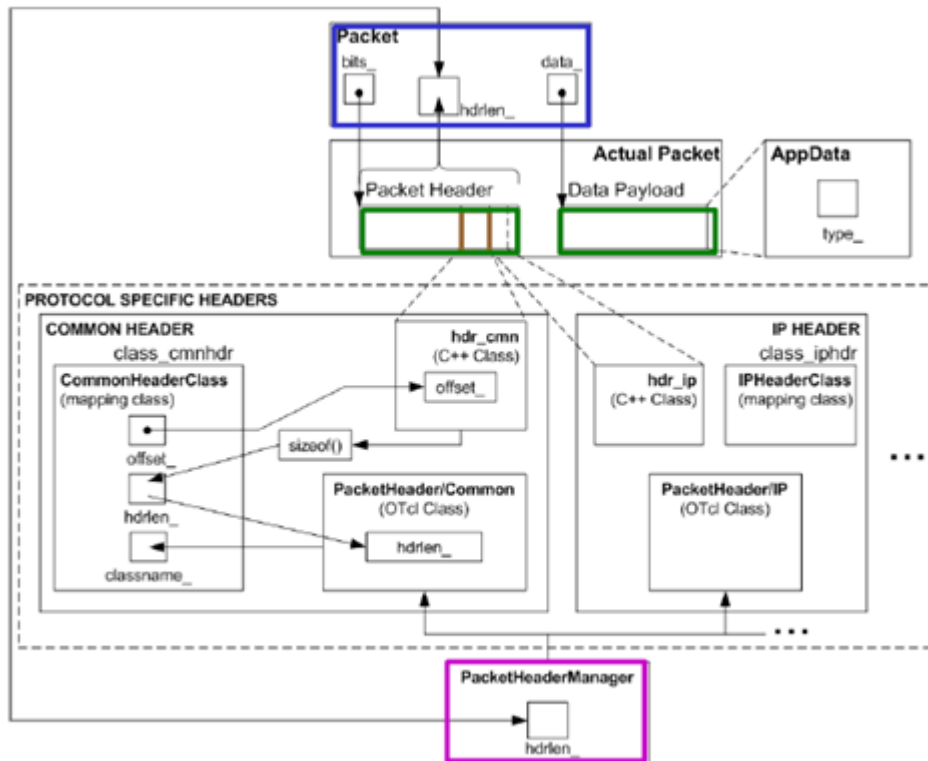


Fig. 17. Arquitectura interna de un paquete en ns-2 [19]

Un paquete en ns-2, está limitado por una longitud fija (*hdrlen_*) y se compone de dos elementos: una cabecera y un conjunto de datos. La cabecera está representada por una estructura implementada en C++ con atributos fijos que poseen un tamaño limitado [19]. Por ejemplo, en la Fig. 17, se muestra la cabecera de un nuevo agente (Packet Header), éste se conforma de un encabezado propio (Common Header) que incluye funciones propias de ns-2 tales como *offset()* para separación de elementos dentro de la cabecera y *sizeof()* para limitación de tamaño.

³ Hash es un valor también conocido como "message digest", está representado por un número generado a partir de una cadena de texto y es utilizado para identificar de manera íntegra la información contenida en un mensaje, texto, etc. evitando que la información pueda modificarse sin previamente haber modificado este valor [11].

Así mismo, se tiene una cabecera común para cada paquete (*hdr_cmn*), que se utiliza para agregar al paquete implementado por el agente, una cabecera con información acerca del tamaño total del paquete, tipo y el ID único asignado a cada paquete. En complemento a la cabecera del ejemplo en la *Fig. 17*, se añade una cabecera IP (IP Header) útil para encaminamiento, recalando que cada cabecera contiene sus características propias y en conjunto forman la cabecera principal del paquete, a la cual se le adjuntaran los datos a transmitir [19].

De igual forma, dentro de la cabecera principal pueden existir cabeceras de menor tamaño y diferente tipo. Por ejemplo, al transmitir datos de un servidor FTP mediante TCP/IP, un paquete contiene una cabecera global compuesta por la cabecera de datos y por la cabecera encargada del encaminamiento, tal como se aprecia en la *Fig. 17* [19].

Para nuestro ambiente de simulación, las cabeceras implementadas son las siguientes:

- a. **Cabecera Estándar:** Adicional a los parámetros establecidos, esta cabecera requiere el tipo de envío: monodifusión o difusión. Esta cabecera es utilizada por los tres agentes para intercambiar información sobre la localización del nodo en la región, información sobre el estado del nodo líder e información del estado de sincronización.
- b. **Cabecera Agente Master:** Adicional a los parámetros estándar, requiere información acerca del tipo de paquete, tiempo de inicio de establecimiento de *Líder*, secuencia, respuesta del nodo *Líder*, y ubicación de un antiguo nodo *Líder* en caso de nuevo establecimiento de *Líder*. Esta cabecera es utilizada por el agente Master para funciones de envío de mensajes de loopback para afirmar el estado del nodo.

En la *Fig. 18*, se presentan todos los elementos y funciones principales de las dos cabeceras implementadas en nuestro ambiente de simulación.

CABECERA ESTANDAR	CABECERA AGENTE MASTER
int & tipo ()	int & tipo ()
int & subtipo ()	double & tiempo_envio ()
double & tiempo_envio ()	double & comienzo_ldr ()
int & region_ID_X ()	double & tiempo_salida ()
int & region_ID_Y ()	int & emisor ()
int & emisor ()	int & destino ()
int & destino ()	int & region_ID_X ()
int & tipo_envio ()	int & region_ID_Y ()
unsigned int & hash ()	int & secuencia ()
char * visualiza_header ()	int & respuesta ()
	int & version ()
	int & antiguo_posicion_x ()
	int & antiguo_posicion_y ()
	char * visualiza_header ()

Fig. 18. Principales elementos de las cabeceras de los paquetes implementadas en el ambiente de simulación.

En cuanto al tipo de banderas (de acuerdo a [42]), se pueden identificar cuatro específicos, los mismos que son descritos a continuación:

- 1) **Banderas de aplicación (SMS_APLICACION):** son banderas enviadas y recibidas desde la capa de aplicación. Este tipo de banderas se utilizan en los mensajes emitidos por el servidor de aplicación, los mensajes solicitados por un cliente y los mensajes enviados/recibidos por un nodo virtual que estén relacionados con la aplicación. [42]
- 2) **Banderas de Elección de Líder (SMS_MASTER):** comprende los tipos de banderas utilizados para realizar el algoritmo de elección de *Líder*, se distinguen los siguientes subtipos de banderas [42]:
 - **LEADER_REQUEST:** Solicitud de liderazgo de un nuevo nodo cuando entra en una región.

- **LEADER_REPLY:** Respuesta a un nodo *Líder* confirmando su existencia
- **MSG_HEARTBEAT:** bandera utilizada en mensajes periódicos para reafirmar el liderazgo de un nuevo *Líder*.
- **MSG_LEADERLEFT:** bandera utilizada en el mensaje de notificación que emite un nodo *Líder* cuando abandona la región.

Como medio de comprobación para la recepción de los mensajes con banderas de estos tipos, se implementaron los siguientes tipos de banderas:

- **LEADER_CONFIRM:** Afirmación de liderazgo
- **RELECTION_LEADER:** Inicio de nueva elección del liderazgo.

3) **Banderas de Sincronización (SMS_SINCRONIA):** son banderas para controlar la sincronización entre los agentes de un mismo nodo. Se implementaron dos subtipos de banderas: *SOLICITA_SINCRONIA* implementado para solicitar sincronización y el *CONFIRMA_SINCRONIA* que sirve para dar respuesta y confirmar la sincronización solicitada.

4) **Banderas que contienen información relacionada al estado de la región actual (SMS_REGION):** banderas utilizadas en mensajes para comunicar cambios de región o para reafirmar que el nodo se mantiene en la región actual luego de transcurrido un tiempo determinado [42].

3.3.2. Algoritmo para elección de un Nodo Líder

En [42], se propone un algoritmo para la elección del nodo *Líder* en una región. El algoritmo contiene la variabilidad de estados del nodo para convertirse en un *Líder* o

en un *No Líder*. El algoritmo se puede ver en la *Fig. 19*, las etapas numeradas del 1 al 10 son los pasos para que un nodo configure su estado dentro de una región.

- 1) El inicio del algoritmo es la ubicación del nodo en la rejilla. Si el nodo no se encuentra dentro de la rejilla asumirá el estado de nodo *Muerto*.
- 2) Al ingresar a una nueva región, el nodo configura su estado a *Desconocido*, puesto que ignora qué rol desempeña en la región.
- 3) Un nodo *Desconocido* cambia a *Solicitante*, para emitir un mensaje de solicitud de liderazgo en tiempo aleatorio —*LeaderRequest*— a todos los nodos de la región.
- 4) Si el nodo *Solicitante* no recibe un mensaje de respuesta por un tiempo determinado (*LeaderReply* o *Heartbeat*) cambia su estado a *Líder*. Un nodo afirma su liderazgo periódicamente con el envío de mensajes, con la bandera tipo *Heartbeat*, hacia todos los nodos de la región.
- 5) Un nodo *Líder* renuncia a su liderazgo por dos factores: i) cambio de región, ii) duplicado de liderazgo⁴.
- 6) Un nodo se mantiene en ese estado *No Líder* al recibir periódicamente mensajes *Heartbeat* transmitidos por el *Líder* de la región. Al no recibir, por un periodo determinado, dos mensajes de afirmación del líder (desbordamiento de colas o pérdidas de paquetes), cambia su estado a *Inestable*.
- 7) Un nodo en estado *Inestable*, activa un temporizador en espera de un mensaje con la bandera *Heartbeat*. Si no recibe este mensaje establece su estado a *Desconocido* para nuevamente reiniciar el algoritmo.
- 8) Si un nodo en estado *Inestable* recibe un mensaje del *Líder* con la bandera *Heartbeat* cambia a *No Líder* y reconoce al *Líder*. Si un nodo en estado *Desconocido* recibe un mensaje del *Líder* con la bandera *Heartbeat* o *LeaderReply* cambia a estado *No Líder*.
- 9) Un nodo en estado *Solicitante*, cambia directamente a *No Líder* si recibe un mensaje del *Líder* con la bandera *Heartbeat* o *LeaderReply*.
- 10) Por último, de color rojo en la *Fig. 19*, se observa que en cualquier estado (*Solicitante*, *Líder*, *No Líder*, *Inestable* o *Muerto*), el nodo al ingresar en una

⁴ Duplicado de liderazgo: Un nuevo *Líder* recibe mensajes *Heartbeat* de un *Líder* anterior en la región.

nueva de región se establece como *Desconocido*, reiniciando el algoritmo de elección del líder.

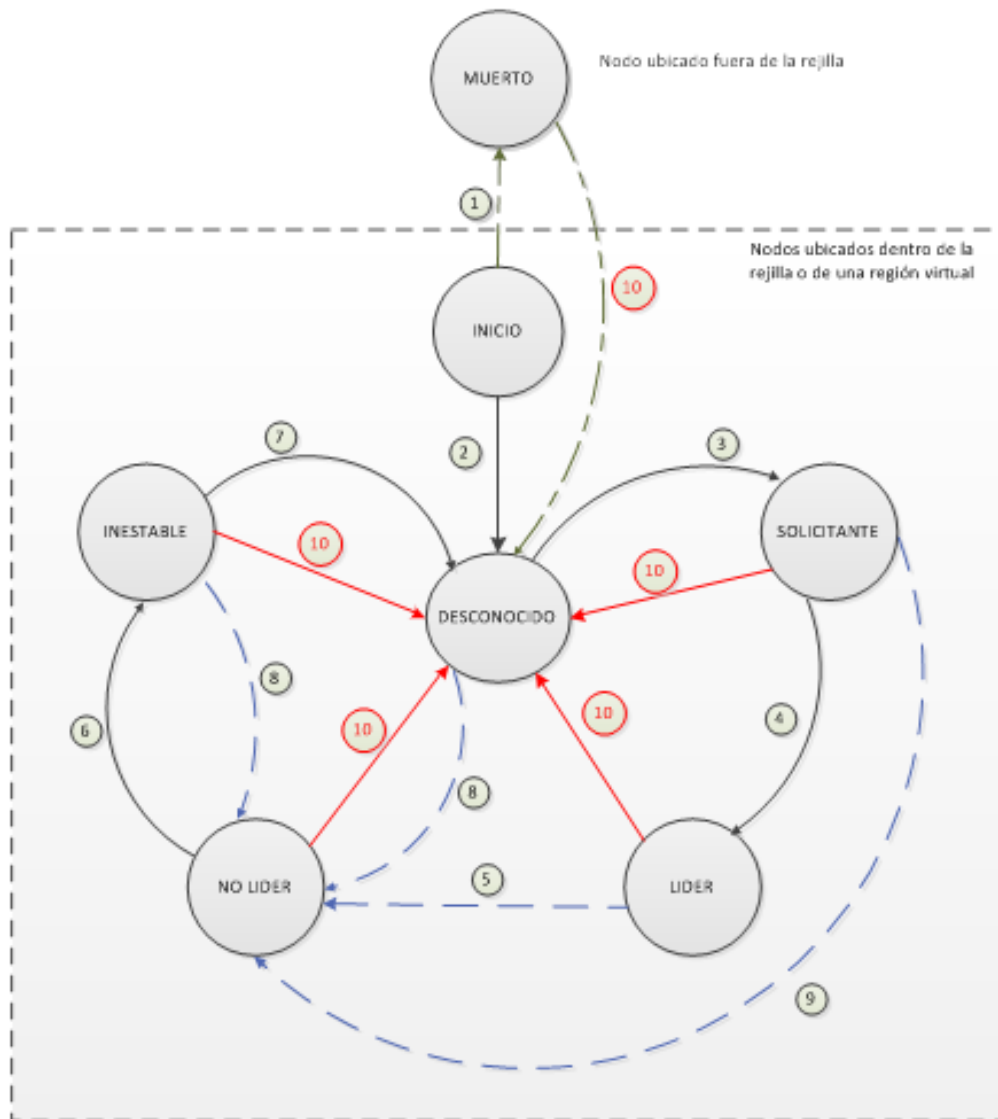


Fig. 19. Algoritmo para elección de Líder [42].

3.3.3. Agentes

En ns-2 Un agente representa los puntos finales de la capa de red. Un agente procesa las unidades de datos (paquetes) con la tarea de crear o destruir información sobre estos [19], y se utiliza en la aplicación de protocolos en distintas capas. Un agente tiene un desarrollo en C++ y OTcl.

3.3.3.1. Agente Master

El agente Master construye la trama cabecera del paquete y envía los paquetes con el formato de la sección 3.3.1. Así, en acorde a lo que se establece en [42], el agente Master desarrolla funciones para: i) la elección del *Líder*, ii) rastreo de los nodos y iii) envío de mensajes de notificación para a los agentes Constancy y Members. En esta sección se describirá las clases que contiene el agente Master.

3.3.3.1.1. Clases del Agente Máster

Las clases del Agente Master están contenidas de acuerdo a [42] y [5] por lo que se tiene: *MasterClass* que nos permite el enlace del compilador y el intérprete. Contadores de tiempo *SolicitudLiderTemporizador*, *AntiguoLiderTemporizador* y *MasterTemporizador* que permiten la gestión del período de los mensajes de elección del nodo *Líder*. *MasterAgent* que contiene el desarrollo del algoritmo de gestión del nodo *Líder* y Las clases *EstandarHeaderClass* y *MasterHeaderClass* para la declaración de la cabecera de los paquetes.

a. Enlace con el intérprete: *MasterClass*

La clase *MasterClass* permite el enlace del intérprete y el compilador y el intérprete. Esta clase es derivada de la clase *TclClass*, véase la *Fig. 20*. Su función es la de crear un objeto OTcl *Master*. Para ello, El constructor de la clase *TclClass* en C++, toma como argumento “*Agent/Master*” para con la función *create()*, entregar un el objeto OTcl *Master* [19].

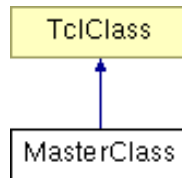


Fig. 20. Enlace del Agente Master con OTcl.

b. Contadores de tiempo: SolicitudLiderTemporizador, AntiguoLiderTemporizador y MasterTemporizador.

Los contadores de tiempo utilizan el método *expire()* de ns-2 que es derivado de la clase *TimerHandler* [19]. El diagrama de colaboración entre Master y los contadores de tiempo se presenta en la *Fig. 21*. Para el desarrollo de los distintos procesos del agente Máster: envío de paquetes, elección de *Líder*, supervisión de nodos, entre otros, hemos implementados tres contadores: *MasterTemporizador*, *SolicitudLiderTemporizador* y *AntiguoLiderTemporizador*.

- **MasterTemporizador:** establece el tiempo para revisar la ubicación de un nodo.
- **SolicitudLiderTemporizador:** indica el control del tiempo para el envío aleatorio de los mensajes de elección de un nodo líder.
- **AntiguoLiderTemporizador:** indica el tiempo para el envío periódico de banderas *Heartbeat*.

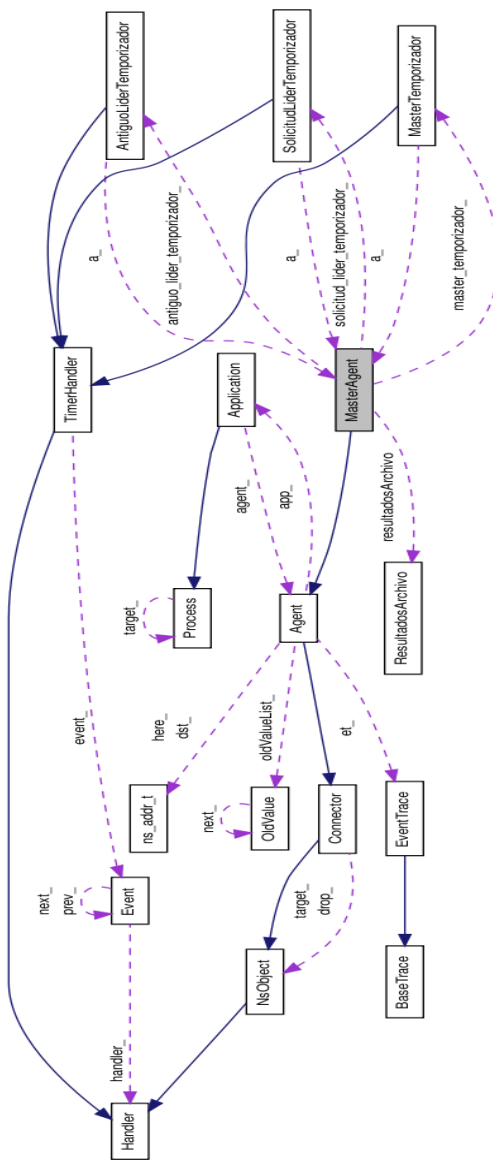


Fig. 21. Diagrama de Colaboración entre Master y los contadores de tiempo para el control de la elección del Líder.

CONTADORES DE TIEMPO DEL AGENTE MASTER	
CONTADORES DE TIEMPO	ATRIBUTOS
AntiguoLiderTemporizador	antiguo_lider_temporizador_
MasterTemporizador	master_temporizador_
SolicitudLiderTemporizador	solicitud_lider_temporizador_

Tabla 2. Contadores de tiempo del Agente Master

c. El agente: Clase MasterAgent

La clase *MasterAgent* contiene las funciones necesarias para la elección del nodo líder. En la Fig. 22 se muestra la ubicación del agente Master en la jerarquía de ns-2. Esta clase está constituida por las siguientes funciones:

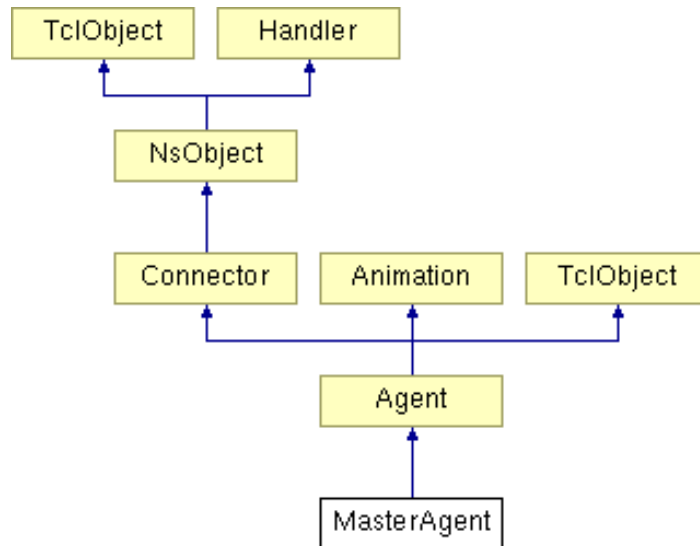


Fig. 22. Ubicación del Agente Master en ns-2.

- **El constructor de la clase: MasterAgent()**

El constructor de la clase está definido para que reciba un argumento PT_MASTER (el nombre de las variables está definido de acuerdo a la nomenclatura de [42]), este argumento sirve para identificar los paquetes de control enviados y recibidos por el agente [19]. Además, se crean los objetos para los contadores de tiempo (véase *Tabla 2*) e inicializa algunos atributos como *estado_sincronización_*, *tiempo_salida_*, *tiempo_empieza_lider_*. Además, El constructor contempla el método *bind()* que permite la correspondencia entre las variables de los objetos C++ con el objeto OTCL [4].

- **Vincular los métodos OTcl a C++: *command()***

La función *command()* —propia de ns-2— se ejecuta cada vez que un comando Tcl invoca un proceso de Master. El método evalúa los parámetros entregados, en forma de cadena de texto. Así, para nuestro caso, si se encuentra *start*, se convoca a la función *datoRegion()* que ubica la región a la que pertenece el nodo, además si el nodo se localiza fuera de la rejilla, este método le coloca en estado Muerto (ver sección 3.3.2). Si la cadena de texto es *revisa* convoca a la función *revisaLocalizacion()*. Por último, si la cadena de texto es *set-trazaArchivoNombre*, convoca a *función_imprimir()* de la clase resultados.

- **Recepción de paquetes: *recv()***

Esta función —que es propia de ns-2— es llamada cada vez que el Master recibe un paquete. Al recibir los paquetes de datos con la información de los eventos de la capa virtual, las funciones *recv()*, *seleccionarLider()* y *verificaEstadoNodo()* realizan el control de la elección del *Líder*.

Al recibir un paquete estándar con la bandera tipo puesto en SMS_MASTER (definida en la sección 3.3.1), la función comienza con el control de la elección del líder, para ello la función llama a la función *seleccionarLider()*.

- **Selección de nodo líder: *void seleccionarLider(Packet *)***

Todos los nodos deben establecer siempre su estado de *Líder* o *No Líder*. Para ello, la función *seleccionarLider()* coloca el estado a un nodo. La función recibe el paquete con la información de la capa virtual y procede a establecer el estado de un nodo:

- El nodo en estado *Solicitante* difunde un mensaje con la banderas de solicitud de liderazgo *LeaderRequest* (el mensaje se envía en un tiempo aleatorio para evitar colisión entre paquetes, procedentes de distintos nodos de la región). Si hay respuesta de existencia de un *Líder* en la región (*LeaderReply*), la función coloca al nodo en estado *No Líder*, caso contrario lo coloca en estado *Líder*.
- Si un nodo recibe la notificación *LeaderLeft* —se emite cuando el *Líder* abandona el liderazgo— la función permite una nueva elección de un nodo *Líder*.
- Además, el nodo virtual debe prevenir estados anormales del *Líder*. Para ello, el nodo *Líder* confirmar su estado de liderazgo con el envío de banderas *Heartbeat*, la función permite el control del estado del *Líder*.

Así, si al no haber perdida de paquetes la función establece el estado de un nodo. Para esta tarea se utiliza el envío de paquetes en modo monodifusión y difusión.

- **Determinación del estado del nodo: void verificaEstadoNodo ()**

La función comprueba el estado del nodo, para junto con el algoritmo de la sección 3.3.2, determinar la acción a tomarse: Si el estado del nodo es *Líder*, se emite de forma periódica los mensajes con las banderas *Heartbeat*, Si el nodo es *No Líder*, se verifica las notificaciones del *Líder*, caso contrario realiza el algoritmo de elección de *Líder*.

- **Envío de paquetes:**

Un objetivo del agente Master es la permanente notificación de eventos a los otros dos agentes. El agente Master obtiene el paquete de la capa virtual, lo procesa y

envía las notificaciones de los eventos del nodo *Líder* a los otros agentes. La implementación de los paquetes utiliza las funciones provenientes de las clases *Agent* y *Packet* de ns-2.

El ambiente de simulación utiliza la función *allocpkt()* —proveniente de la clase *Agent*—, ésta función permite configurar los valores por defecto para las estructuras cabeceras *hdr_cmn*, *hdr_ip* y las cabeceras de los paquetes de la sección 3.3.1 [19]. Las tramas cabeceras de los paquetes tienen varios campos a ser procesados, para ello utilizamos las siguientes funciones:

void send_loopback_pkt(int): Esta función informa los cambios de estados de los nodos a los otros agentes. En su algoritmo, establece los cambios por i) elección y permanencia de un nodo *Líder*, ii) cambio de región, iii) permanencia de un nodo *No Líder*. Para ello utiliza valores enteros como ingreso para definir el mensaje de salida

void send_monodifusion_pkt(int, int, int, int): Esta función notifica al nodo solicitante la existencia de un nodo *Líder*. Utiliza un valor de entrada, para establecer el mensaje como *LeaderReply*; además, recibe las direcciones del emisor y destino del paquete, y la secuencia. Con estos valores transmite el paquete en modo monodifusión.

void send_difusion_pkt(int, int): Esta función permite a un nodo realizar la solicitud de liderazgo en modo difusión (*LEADER_REQUEST*). Para ello, dentro de la estructura del paquete se utiliza la variable *IP_BROADCAST* —variable propia de ns-2— que permite entregar la información a todos los nodos.

void send_virtual_difusion(int, int, int, int, int): Esta función notifica eventos del nodo *Líder*. Para esto, en sus valores de entrada contiene, los valores de la ubicación

actual y estado del nodo *Líder* en la capa virtual. Además, la función entrega dos valores a ser enviados, i) *MSG_LEADERLEFT* que notificar a todos los otros nodos el inicio/reinicio del algoritmo de elección de *Líder*, iii) confirmación de existencia y categoría de nodo *Líder*.

- **Funciones iniciales:** Estas funciones permiten llamar a funciones específicas del agente. Estas inician cuando existe un cambio de estado en algún temporizador utilizado por el Master.

virtual void inicio(int): Función inicial para estar al tanto de la ubicación del nodo mediante la función *revisaLocalizacion()*. La función se llama al cambiar de estado el temporizador *MasterTemporizador*.

virtual void inicioSolicitudLider(int): Función que proporciona el tiempo de espera para la solicitud de liderazgo mediante *envioSolicitudLider()*. La función se llama al cambiar de estado el temporizador *SolicitudLiderTemporizador*.

virtual void inicioAntiguoLider (int): Función que proporciona el tiempo de espera aleatorio para el envío de mensajes de afirmación de liderazgo (*Heartbeat*) mediante la función *retardoHeartbeat()*. La función se llama al cambiar de estado el temporizador *AntiguoLiderTemporizador*.

- **Ubicación del nodo:**

Esta función que nos permite conocer la ubicación del nodo en el escenario de simulación. La función utiliza *getLoc()* y *getVelo()* de la clase *MobileNode* de ns-2,

métodos que permiten conocer la ubicación y velocidad de un nodo [40]. Así, ésta función informa eventos de posición y velocidad del nodo en la región virtual.

El nodo virtual conoce el estado del nodo *Líder* para prevenir estados anormales, si esto sucede se debe lanzar una nueva elección de *Líder*. Si el nodo *Líder* va a salir de su actual región, renuncia a su liderazgo y permite una nueva elección de un nodo *Líder*. Para ello, utiliza el mensaje *Relection_Leader*. Por otra parte, si un nodo *No Líder* abandona la región, este cambiara su estado a desconocido y solicitará el liderazgo de la región mediante el envío del mensaje de liderazgo *LeaderRequest*.

Además, esta función mantiene informada a la capa virtual con una constante revisión de la ubicación del nodo en la región, para ello utiliza las funciones *datoRegion()* y *datoTiempoArribo()*. Esto con el propósito de comprobar si el nodo se encuentra en movimiento, el tiempo de salida del nodo de su región actual y el tiempo de espera para la notificación del evento de salida del nodo. Además, la función permite determinar si el nodo se encuentra dentro de la rejilla, si este no fuera el caso, la función coloca al nodo en estado Muerto.

double revisaLocalizacion(): La función realiza cuatro tareas, i) comprueba si el nodo cambia de región, ii) entrega el tiempo en que el nodo *Líder* cambia de región. iii) determina si el nodo se moverá dentro de la región a la que pertenece, iv) determina si el nodo se encuentra en movimiento.

- **Determinación del nodo dentro de la rejilla:**

El movimiento de los nodos físicos (vehículos) de VANETs se restringe en un cierto rango de coordenadas (calles). La función permite desplazar la ubicación de la rejilla para permitir contener (dentro de este rango de coordenadas) los movimientos de los

nodos físicos. Además, determina la región a la que el nodo pertenece, si el nodo no se encuentra dentro de la rejilla, la función notifica este evento mediante un impreso de pantalla y coloca al nodo en estado Muerto.

$$\text{Región del Nodo} = \text{INT} \left(\frac{\text{Ubicación del nodo en el escenario de simulación}}{\text{Tamaño de la region en el escenario de simulación}} \right) \quad \text{Ec. 1}$$

int datoRegion(double, double, int *, int *): La función recibe la ubicación del nodo para determinar la región del nodo en el escenario de simulación, si esta abandona la rejilla, nos presenta una mensaje de alarma para advertir, este evento⁵.

- **Estimación del tiempo de cambio de ubicación en la rejilla:**

Para la estimación del tiempo en que un nodo arribará a un punto, se utiliza una ecuación simple de cinemática (véase Ec.2). El nodo puede tomar dos casos en la dirección de su movimiento del nodo: i) el nodo se encuentra en un movimiento paralelo a uno de los ejes cartesianos, ii) el nodo toma un movimiento con inclinación de los ejes cartesianos.

$$\text{Tiempo de Arribo} = \frac{(\text{Posición actual} - \text{Posición a Ubicarse})}{\text{Velocidad del Nodo}} \quad \text{Ec. 2}$$

double datoTiempoArribo(double, double, double, double): La función tiene como parámetros de ingreso las posiciones en el eje cartesiano del nodo, y su velocidad. Así, la función considera los casos de movimiento del nodo, para presentar un tiempo de arribo a su próxima ubicación. Si el movimiento del nodo no

⁵ En la Ec.1 la notación INT indica la parte entera de la ecuación.

es paralelo con respecto a los ejes cartesianos, la función entrega el tiempo menor de una de las componentes en X y Y para determinar la región más próxima.

- **Forma de la región en la rejilla:**

La forma de la región de la rejilla se determina, para ubicar las coordenadas de inicio y fin de cada una de las regiones. Así, esta función junto con *datoRegion()* y *datoTiempoArribo()* ubican a las rejilla dentro del escenario de simulación. Las formas de las regiones son paralelogramos, por lo que utilizando una división en tamaño de la rejilla, se puede determinar el número de regiones.

int formaRegion(int , int, double, double, double, double): La función tiene como parámetros de ingreso el número de filas y columnas en las que se divide a la rejilla, además, se ingresa el tamaño de la rejilla para seccionarlo en filas y columnas. Así devuelve el tamaño (largo y ancho) de cada una de las regiones.

- **Reinicio del estado de un nodo líder:**

El agente Master entrega una función para restablecer el estado inicial de un nodo (estado desconocido), si este abandona una región.

void reiniciaEstadoNodo(): La función restable el estado inicial y contadores de tiempo de un nodo.

d. Declaración de la cabecera de los paquetes: EstandarHeaderClass y MasterHeaderClass.

Estas clases permiten la interconexión de las cabeceras de los paquetes con TCL [19]. Estas clases son derivadas de la clase ns-2 *PacketHeaderClass*. Por ello, su

constructor es invocado tomando dos argumentos de entrada: el nombre de la clase en OTcl y el tamaño del paquete. Además, *bind_offset()* —función perteneciente a *PacketHeaderClass*— debe ser llamada para que la nueva cabecera sea inserta en el paquete utilizado. En la Fig. 23 se muestra la ubicación de las cabeceras de los paquetes implementados en ns-2.

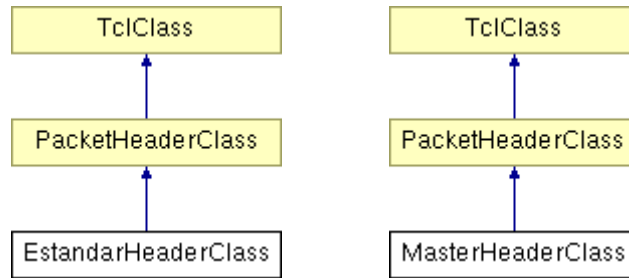


Fig. 23. Ubicación de las cabeceras de los paquetes MasterHeaderClass y EstandarHeaderClass en ns-2

3.3.3.2. Agente Constancy

El agente Constancy coordina las acciones de los tres agentes. En esta sección se describirá las clases que contiene el agente Constancy.

3.3.3.2.1. Clases del Agente Members

a. Enlace con el intérprete: ConstancyClass.

La clase *ConstancyClass* permite establecer el enlace entre el compilador y el intérprete. Esta clase es derivada de la clase *TclClass*, véase Fig. 24. Su función es crear un objeto del tipo Constancy escrito en OTcl [19]. Contiene dos funciones: el constructor de la clase *ConstancyAgent()* que recibe como argumento “Agent/Constancy” e inicializa variables propias del agente Constancy y la función *create()* que entrega el objeto solicitado.

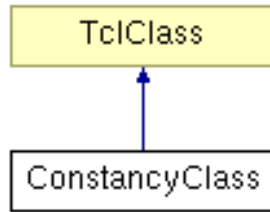


Fig. 24. Enlace del Agente Constancy con OTcl

b. El agente: Clase ConstancyAgent

La clase *ConstancyAgent* contiene las funciones necesarias del Agente Constancy. En la *Fig. 25*, se muestra la ubicación del agente Constancy en la jerarquía de ns-2. Esta clase está constituida por las siguientes funciones:

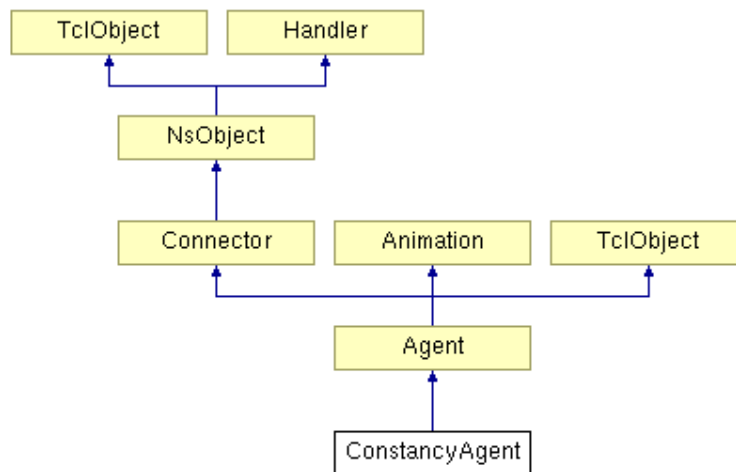


Fig. 25. Ubicación del Agente Constancy en ns-2.

- **El constructor de la clase: ConstancyAgent()**

El constructor de la clase *ConstancyAgent* está definido para que reciba un argumento `PT_ESTANDAR`, el mismo que permite identificar los paquetes de enviados y recibidos por este agente [19].

El constructor emplea el método *bind()* para permitir la correspondencia entre las variables de los objetos escritos en C++ con los objetos escritos en OTcl [19]. En relación a dicha correspondencia definimos las variables para identificar la ubicación del nodo *Líder*, las características de la topología y las variables necesarias para establecer la sincronización.

El constructor además inicializa algunos atributos propios del agente Constancy como son: ubicación del nodo en *X* y *Y*, *secuencia_*, *numero_reintentos*, *nueva_sinc_*, *primer_nodo_*, *tfin_primer_nodo*. De igual manera, como parte del constructor, se establece un espacio de memoria destinado para que el agente pueda desempeñar sus funciones sin problemas registrando sus respuestas en un fichero denominado resultados.

- **Vincular los métodos OTcl a C++: *command()***

La función *command()* se ejecuta cada vez que un comando Tcl invoca un proceso del agente Constancy. En función de los parámetros de tipo string entregados, se ejecuta un evento específico. Así, para nuestro caso, si la cadena recibida es *set-trazaArchivoNombre* se convoca a la *funcion_imprimir()* de la clase resultados para visualizar mensajes de respuesta.

- **Recepción de paquetes: *recv()***

Esta función —que es propia de ns-2— se invoca cada vez que el agente Constancy recibe un paquete. Al recibir un paquete con información de eventos de la capa virtual, la función *recv()* publica el mensaje en el fichero de respuesta. Además, el agente Constancy realiza una acción específica de acuerdo a la información contenida en la cabecera del paquete.

Es así que, si el paquete de entrada contiene una cabecera de tipo `hdr_estandar`, ésta función analiza los parámetros de la cabecera del paquete entrante con el fin de determinar si el mensaje recibido contiene información del estado del nodo *Líder* (`SMS_LIDER`), información de la región (`SMS_REGION`), es un mensaje para solicitar sincronización (`SMS_SINCRONIA`) o es un mensaje de aplicación (`SMS_APLICACION`) todos estos mensajes antes definidos en la sección 3.31.

Si el mensaje proviene es del tipo `SMS_LIDER`, `recv()` analiza el valor del atributo subtipo, el cual puede estar configurado en *no-líder*, *antiguo_líder* o un *nuevo líder* (usados en modo loopback). Por otra parte, si el mensaje entrante contiene información acerca de la región o indica el cambio de localización del nodo a una región diferente, una notificación de ese evento es impreso en el fichero de resultado. Por último, si el mensaje es de aplicación lo único que se hace es notificarlo y liberar el paquete si proviene de un nodo servidor o de respaldo (en caso de que se desee desarrollar alguna aplicación se debe generar procesos específicos para el tratamiento de estos paquetes).

- **Mensajes de difusión de sincronización: `broadcast_servidor(int,int,int, u_char *)`**

Esta función al ser invocada crea un paquete con un mensaje de sincronización `SMS_SINCRONIA` que además contiene información del nodo emisor, de la región y otros parámetros dada por la cabecera `hdr_estandar` detallada en la sección 3.3.1 y lo emite por difusión a todos los nodos vecinos para dar a conocer el estado de sincronía.

- **Envío de mensajes loopback para sincronización: `enviar_msgloopback(int , int)`**

Esta función crea un paquete de cabecera estándar, lo complementa con información de las cabeceras `hdr_cmn` y `hdr_ip` propias de ns-2. El paquete a enviar será de tipo

sincronía, mientras que el subtipo puede ser solicitud o confirmación. Además entrega información con la ubicación e identificación del nodo emisor, el nodo de destino, el tiempo y el tipo de envío.

3.3.3.3. Agente Members

El agente Members notifica la recepción de paquetes provenientes de los agentes Master y Constancy. En esta sección se describirá las clases que contiene el agente Members.

3.3.3.3.1. Clases del Agente Members

a. Enlace con el intérprete: MembersClass.

La clase *MembersClass* permite el enlace del compilador y el intérprete. Esta clase es derivada de la clase *TclClass* de ns-2, véase *Fig.26*. Su función es la de crear un objeto OTcl *Members*. Para ello, el constructor de la clase *TclClass* en C++, toma como argumento “*Agent/Members*” para con la función *create()*, entregar el objeto [19].

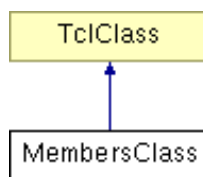


Fig. 26. Enlace del Agente Master con OTcl

b. El agente: Clase MembersAgent

La clase *MembersAgent* contiene las funciones necesarias del Agente Members. En *Fig. 27* se muestra la ubicación del agente Members en la jerarquía de ns-2. Esta clase está constituida por las siguientes funciones:

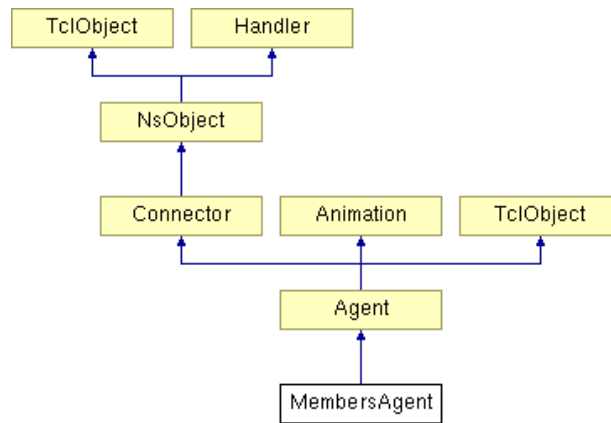


Fig. 27. Ubicación del Agente Members en ns-2.

- **El constructor de la clase: MembersAgent()**

El constructor de la clase está definido para que reciba un argumento PT_CONSTANCY; este argumento sirve para identificar los paquetes de control enviados y recibidos por el agente [19]. Inicia algunos atributos del agente como la identificación de la región. Además, este constructor contempla el método *bind()* que permite la correspondencia entre las variables de los objetos C++ con el objeto OTCL [17].

- **Vincular los métodos OTcl a C++: command()**

La función *command()* —propia de ns-2— se ejecuta cada vez que un comando Tcl invoca un proceso de Members. El método evalúa los parámetros entregados, en forma de string. Así, para nuestro caso, sí se encuentra la cadena *set-trazaArchivoNombre*, convoca a *función_imprimir()* de la clase resultados.

- **Recepción de paquetes: recv()**

Esta función —que es propia de ns-2— es llamada cada vez que Members recibe un paquete. Al recibir los paquetes de datos con la información de los eventos de la capa virtual, la función *recv()* describe la publicación de resultados en el fichero de

respuesta. La acción del agente depende de los mensajes de respuesta de los agentes Master y Constancy. Así, Members toma acción cuando recibe una bandera del tipo *SMS_MASTER*, *SMS_REGION*, *SMS_APLICACION* y *SMS_SINCRONIA* (definidas en la sección 3.3.1).

Al recibir las banderas tipo *SMS_MASTER* y *SMS_REGION*, el agente Members revisa las banderas subtipo: *NOLIDER*, *NUEVO_LIDER* y *NUEVA_REGION*, con lo que determina el mensaje de respuesta de recepción de paquetes por el agente Master en el fichero creado. Al recibir los mensajes de: *SMS_APLICACION* y *SMS_SINCRONIA*, provenientes del agente Constancy, el agente Members publicara el mensaje de recepción en el fichero creado.

3.3.4. Archivo de Resultados de la Capa Virtual

Esta clase (resultados.cc) permite plasmar los resultados de la capa virtual sobre un fichero digital. La clase está compuesta por las siguientes funciones [42]:

- **void creamosArchivo(char *):** Esta función utiliza la función *fopen* de C++ cual permite la creación de un fichero.
- **char* estadoLider(int):** La función convierte el estado del líder de un número entero a cadena de caracteres para ser utilizadas en las funciones *imprimirMensajes()* y *imprimirEventos()*.
- **void imprimirMensajes() / void imprimirEventos():** Estas funciones escriben los mensajes de respuesta provenientes de los agentes Master, Constancy y Members sobre un fichero.

3.4. Enlace del Simulador de Red y El Simulador de Tránsito Vehicular

La simulación de redes VANET requiere la utilización de dos software. Primero, el simulador de tránsito (SUMO) que genera la topología de red, movimiento vehicular y los escenarios de simulación de tránsito. Segundo, el simulador de redes ns-2 que analiza el proceso de comunicación de los nodos (vehículos) en base al escenario de movilidad. SUMO contempla la herramienta TraceExporter que genera la integración de los dos simuladores [37].

3.4.1. TraceExporter

Esta sección refiere a la herramienta TraceExporter para vincular los archivos de traza de SUMO a ns-2. Esta herramienta utiliza la plataforma Java 1.5 para su uso. TraceExporter genera tres archivos de salida (config.tcl, activity.tcl y mobility.tcl) cuales realizan la conversión de coordenadas negativas y positivas de SUMO, a únicamente coordenadas positivas, para ns-2. El archivo de configuración config.tcl es usado para configurar algunas variables y describir el escenario de simulación. Los archivos de tránsito vehicular son dados por activity.tcl y mobility.tcl que describen el primer y último movimiento de los nodos. De [37], nos entrega los comandos de TraceExporter:

COMANDO	TIPO	DESCRIPCIÓN
<i>-n</i>	*.net.xml	Nombre del archivo de red.
<i>-t</i>	*.xml	Nombre del archivo de traza.
<i>-a</i>	*.tcl	Archivo de actividad.
<i>-m</i>	*.tcl	Archivo de movilidad.
<i>-c</i>	*.tcl	Archivo de configuración.
<i>-p</i>	Float [0,1]	Tasa de filtrado de vehículos; Penetración de 1 equivale a sin filtrado. Penetración de 0 equivale a sin vehículos
<i>-s</i>		La semilla para la generación de números aleatorios. El default es 0.
<i>-b</i>		Tiempo de inicio de la simulación.
<i>-e</i>		Tiempo de finalización de la simulación.

Tabla 3. Comandos de la herramienta TraceExporter

De manera que el comando de TraceExporter será:

```
java -jar traceExporter.jar ns2 -n *.net.xml -t *.xml -a activity.tcl -m mobility.tcl  
-c config.tcl -p 1
```

Así, ns-2 incluiría los archivos de salida de TraceExporter con el comando *source*, véase la *Fig. 28*.

```
1 #  
2 # Definimos las características para el enlace inalámbrico  
3 #  
4 set val(chan) Channel/WirelessChannel ;  
5 set val(prop) Propagation/TwoRayGround ;  
6 set val(netif) Phy/WirelessPhy ;  
7 set val(mac) Mac/802_11 ;  
8 set val(ifq) Queue/DropTail/PriQueue ;  
9 set val(ll) LL ;  
10 set val(ant) Antenna/OmniAntenna ;  
11 set val(ifqlen) 50 ;  
12 set val(nn) 4 ;  
13 set val(rp) AODV ;  
14  
15 set val(stop) 50 ;  
16 #  
17 # Definimos la Exportacion de archivos de traza de SUMO a ns2.  
18 #  
19 set opt(config-path) "/home/" ;#Direccion de los archivos  
20 ;#de salida de TraceExporter  
21 source config.tcl ;#Importacion de Archivo de  
22 ;#Configuracion  
23 source $opt(mf) ;#Importacion de Archivo de  
24 ;#Transito  
25  
26 #  
27 # Programa principal  
28 #  
29 set ns [new Simulator]  
30 set tracefd [open respuesta21.tr w]  
31 set windowVsTime2 [open respuesta21.tr w]  
32 set namtrace [open respuesta21.nam w]  
33 .  
34 .  
35 .  
36 .
```

Fig. 28. Captura de un fichero de simulación para el enlace de SUMO y ns-2

4. IMPLEMENTACIÓN Y PRUEBAS

Este capítulo contiene la implementación del ambiente de simulación en el simulador de redes. Nosotros presentamos experimentos, resultados y análisis del ambiente de simulación en redes ad hoc.

4.1. Implementación sobre ns-2 y SUMO

Para realizar la simulación de redes VANET es necesario la utilización de dos programas. Primero, el simulador de tránsito vehicular —SUMO— que genera la topología de red, movimiento vehicular y los escenarios de simulación de tránsito. Segundo, el simulador de redes —ns-2—, donde se analiza el proceso de comunicación de los vehículos (nodos físicos) en base al escenario de movilidad.

4.1.1. Implementación sobre Sumo

Esta sección contendrá, la generación de los archivos de SUMO, en base de un modelo de movilidad microscópico.

4.1.1.1. Características de Sumo

SUMO se diseñó en base a dos objetivos principales: visión de una ejecución rápida y altamente portátil [3]. Debido a esto, el software se ejecuta en línea de comandos para asegurar la velocidad de simulación. Además, se puede hacer el uso optativo de la interfaz gráfica. Por otra parte, el software se divide en bloques funcionales para permitir el uso de estructuras de datos más rápidas. Ésta característica puede resultar en un inconveniente al convertir el software en una herramienta de no tan fácil uso [16].

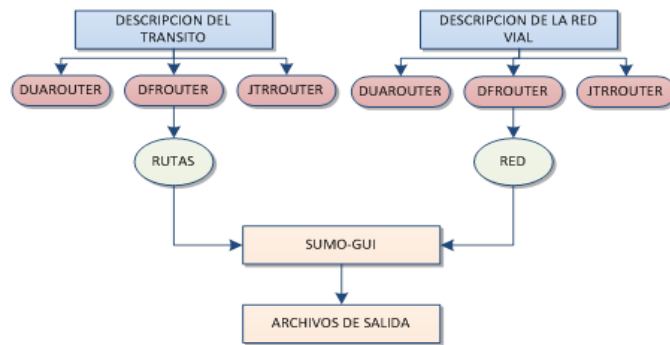


Fig. 29. Aplicaciones del paquete de instalación de SUMO 0.16.0

El paquete de instalación de SUMO que utilizaremos en este trabajo, es la versión 0.16.0 que data del 04 de Diciembre de 2012; ésta contiene varias aplicaciones que son necesarias para generar la simulación (véase Fig. 29). De [37], podemos obtener la característica de las aplicaciones, con sus respectivas extensiones de fichero, las cuales se muestran en la *Tabla 4*⁶:

APLICACION	DESCRIPCION	TIPO DE ARCHIVO
SUMO	Genera la simulación microscópica sin visualización gráfica.	*.sumo.cfg
SUMO-GUI	Genera la Interfaz gráfica para la simulación microscópica.	*.sumo.cfg
NETCONVERTER	Importador y generador de redes viales; lee las redes desde diferentes formatos y las convierte en un formato aceptado para SUMO.	*.netc.cfg
NETGEN	Genera redes abstractas para la simulación de SUMO.	*.netg.cfg
DUAROUTER	Calcula las redes más rápidas a través de la red, importando diferentes tipos de demandas.	*.rou.cfg
JTRROUTER	Calcula las rutas usando los porcentajes de giro de las intersecciones.	*.jtr.cfg
DFROUTER	Calcula las rutas obtenidas por los bucles de inducción.	*.df.cfg
OD2TRIPS	Descompone las matrices origen/destino (matrices O/D), en viajes únicos de vehículos.	*.od2t.cfg
POLYCONVERTER	Importa los puntos de interés y polígonos de diferentes formatos para traducirlos en una descripción visible para SUMO-GUI.	*.net
ACTIVITYGEN	Genera una demanda basada en los requisitos de movilidad de un modelo poblacional.	*.act.cfg

Tabla 4. Características de las aplicaciones del paquete de instalación de SUMO 0.16.0

⁶ En la tercera columna de la Tabla 4 el símbolo * representa el nombre del archivo.

4.1.1.2. Escenarios de Simulación

En el proceso de creación de un escenario de simulación necesitamos establecer los siguientes pasos. En primera instancia, es necesaria la creación de una red vial que puede ser definida por el usuario o ser importa desde una base de datos. El segundo paso es la definición del tránsito vehicular sobre la red vial. El tercer paso recae sobre la ejecución de simulación. A partir de este punto describiremos paso a paso cómo generar escenarios de simulación.

4.1.1.2.1. Creación de la Red Vial

La red vial es el esqueleto del escenario de simulación, para su generación se puede utilizar las aplicaciones NETGEN y NETCONVERT. NETGEN contempla la generación automática de redes. NETCONVERT ofrece al usuario la elección de crear su propia red o importar desde una base de datos externa.

a. NETGEN

En [37], nos indica que la generación automática de redes con NETGEN puede ser de tres tipos: i) redes tipo Grid, que tienen forma de rejilla. ii) redes tipo Spider, que se forman en base a círculos concéntricos. iii) redes Aleatorias, donde se generan de manera aleatoria redes viales. Estos tipos de redes son definidos parcialmente por el usuario, el cual establece parámetros de medida en base al Sistema Internacional de medida (SI) para la generación de la red vial.

- **Redes Grid**

La generación de la red vial tipo rejilla se realiza con los comandos mostrados en la *Tabla 5*. Las distancias se encuentran en metros y los valores por defecto se muestran en la cuarta columna:

Comando	Tipo	Descripción	Default
<i>--grid.x-number</i>	Int	Número de cruces en el eje X.	5
<i>--grid.y-number</i>	Int	Número de cruces en el eje Y.	5
<i>--grid.x-length</i>	Float	Distancia entre los cruces en la coordenada X.	100
<i>--grid.y-length</i>	Float	Distancia entre los cruces en la coordenada Y.	100
<i>--grid.number</i>	Int	El número de cruces en ambas coordenadas.	5
<i>--grid.length</i>	Float	El número de calles en ambas direcciones.	100
<i>--grid.attach-length</i>	Float	La longitud de las calles añadidas en la frontera; 0 significa que no hay calles unidas.	0

Tabla 5. Comandos de generación de la red tipo Grid

La señalización de las intersecciones de vías en la red puede tener tres tipos: *traffic_light*, *priority* y *right_before_left*, los cuales pueden ser asignados con el comando *-junction* o de forma abreviada *-j*, *Tabla 6*.

Comando	Tipo	Descripción	
<i>--j</i>	STRING	<i>traffic_light</i>	La intersección es controlada por un semáforo.
		<i>priority</i>	Los vehículos de la izquierda esperan al paso de los de la derecha.
		<i>right_before_left</i>	Los vehículos de la derecha esperan al paso de los de la izquierda.

Tabla 6. Comandos de generación de cruces de la red tipo Grid

Una vez ejecutamos los comandos de la *Tabla 5* y *Tabla 6* se obtiene la siguiente captura (véase *Fig. 30*), el cual contiene siete cruces, con semáforos, en las coordenadas X y Y , distanciados por 100 m.

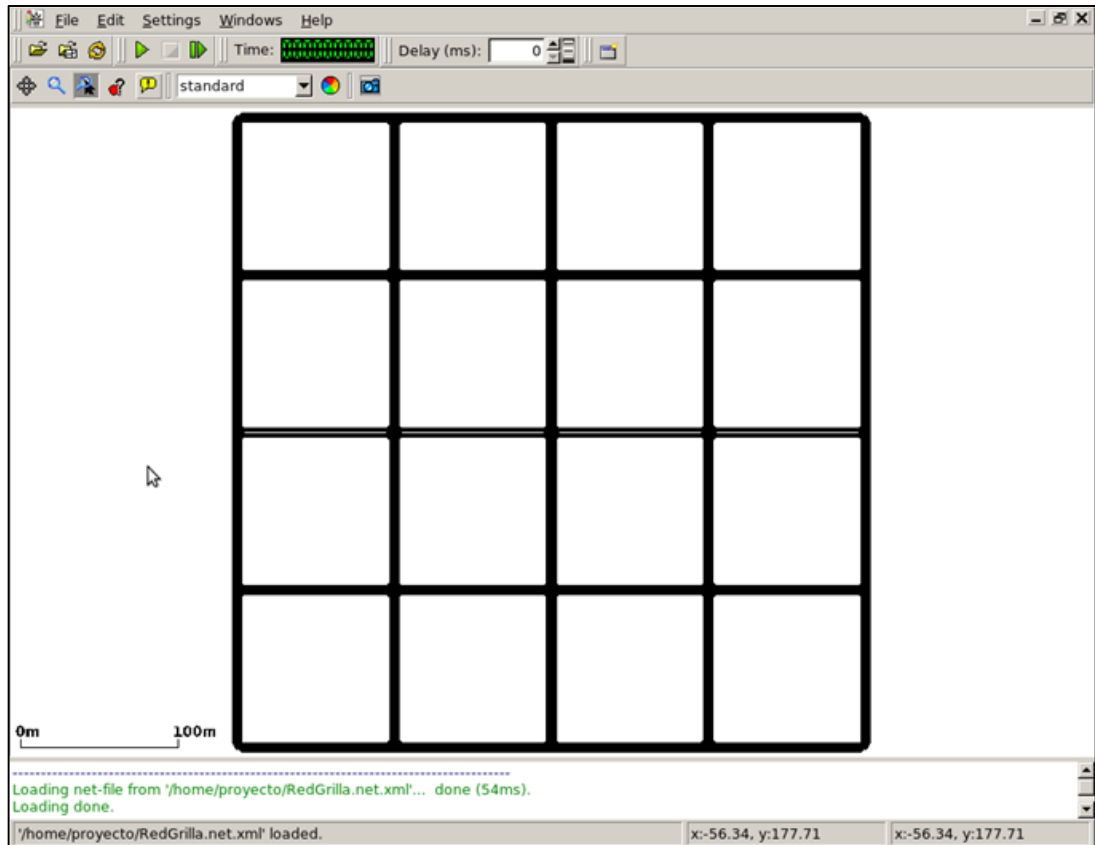


Fig. 30. Red vial tipo Grid

- **Redes Spider**

La red vial tipo spider se genera en base a un punto de central, del cual las vías estarán en forma concéntrica. Los comandos y valores por defecto se presentan en la siguiente tabla:

Comando	Tipo	Descripción	Default
<code>--spider.arm-number</code>	INT	Número de ejes que contiene la red	13
<code>--spider.circle-number</code>	INT	Número de círculos de la red	20
<code>--spider.space-radius</code>	FLOAT	Distancia entre los círculos de la red	100
<code>--spider.omit-center</code>		Omitir el punto central de la red	false

Tabla 7. Comandos de generación de cruces de la red tipo Spider

Una vez que ejecutamos los comandos de la *Tabla 7* se obtiene la *Fig. 31*,

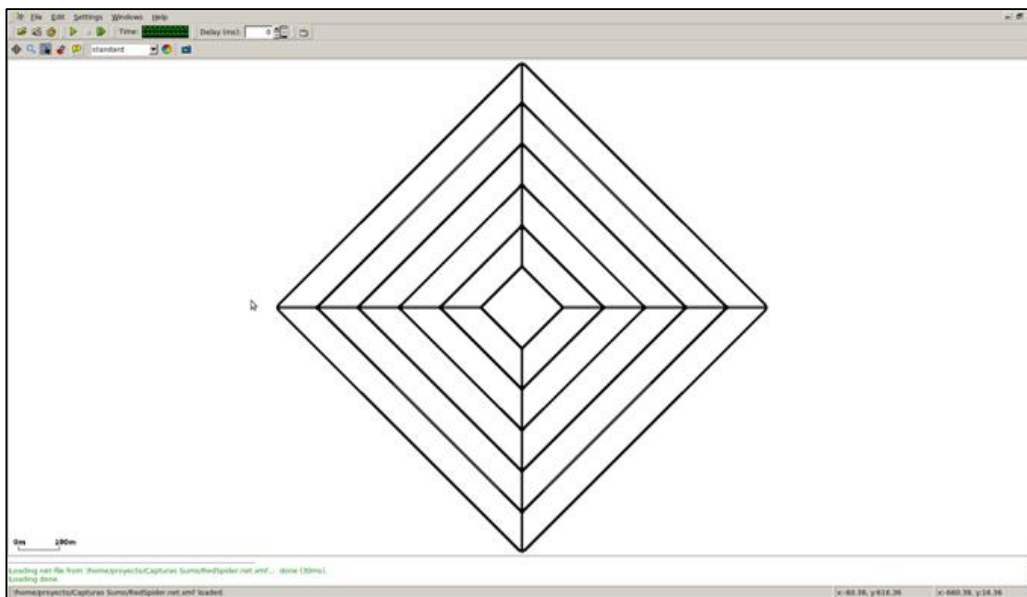


Fig. 31. Red vial tipo Spider

- **Redes Aleatorias**

La red vial tipo Aleatoria, como en las redes tipo Grid y Spider, es generada en base a parámetros establecidos por el usuario e iteraciones para determinar la red vial. Los comandos y valores por defecto se presentan en la *Tabla 8*.

Comando	Tipo	Descripción	Default
<i>--rand.max-distance</i>	FLOAT	Máxima distancia	250
<i>--rand.min-distance</i>	FLOAT	Máxima distancia	100
<i>--rand.min-angle</i>	FLOAT	Mínimo Angulo	0.785398
<i>--rand.num-tries</i>	INT	Número de intentos	50
<i>--rand.connectivity</i>	FLOAT	Conectividad	0.95
<i>--rand.neighbor-dist1</i>	FLOAT	Distancia 1 entre vecinos	0
<i>--rand.neighbor-dist2</i>	FLOAT	Distancia 2 entre vecinos	0
<i>--rand.neighbor-dist3</i>	FLOAT	Distancia 3 entre vecinos	10
<i>--rand.neighbor-dist4</i>	FLOAT	Distancia 4 entre vecinos	10
<i>--rand.neighbor-dist5</i>	FLOAT	Distancia 5 entre vecinos	2
<i>--rand.neighbor-dist6</i>	FLOAT	Distancia 6 entre vecinos	1

Tabla 8. Comandos de generación de cruces de la red tipo Aleatoria

Una vez que ejecutamos los comandos de *Tabla 8* se obtiene la *Fig. 32*.

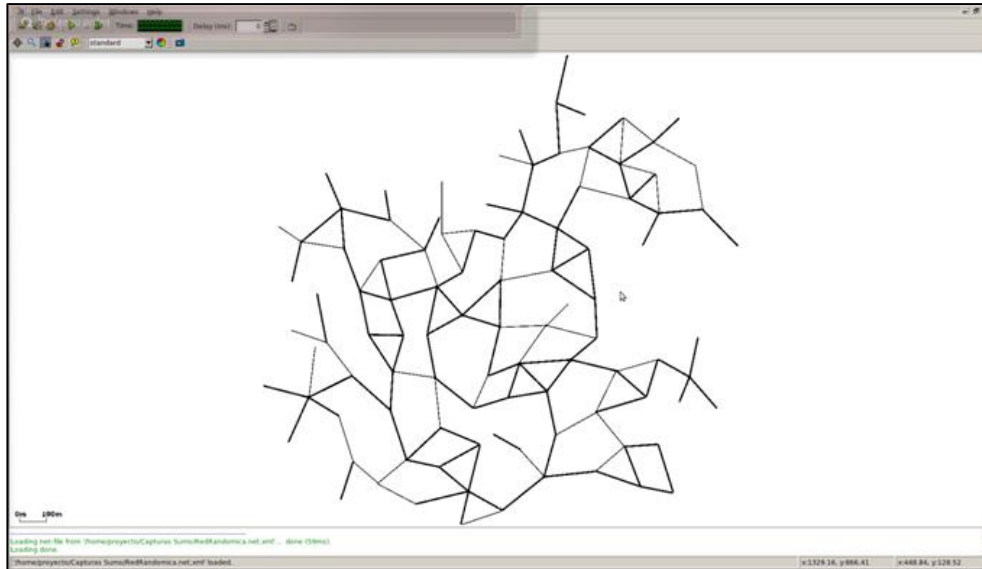


Fig. 32. Red vial tipo Aleatoria

b. NETCONVERT

NETCONVERTER lee archivos *.xml para convertirlos en un formato entendible para SUMO. El usuario puede generar paso a paso las características de una red vial o puede importar desde una base de datos un mapa de red vial.

- **Generación de archivos XML manualmente**

En [37], nos indica que para la generación manual de una red vial, se debe ingresar cuatros archivos *.xml en NETCONVERT, véase *Fig. 33*. Así, SUMO integra los siguientes conceptos para la descripción de la red:

- i. **Node:** representa las intersecciones y uniones existentes de la red vial.
- ii. **Edge:** representa las vías de la red vial.

iii. **Type:** determina las propiedades de las vías, como prioridades, número de carriles, velocidad, entre otras.

iv. **Connection:** indica la unión de los vértices de las vías, para la red final.

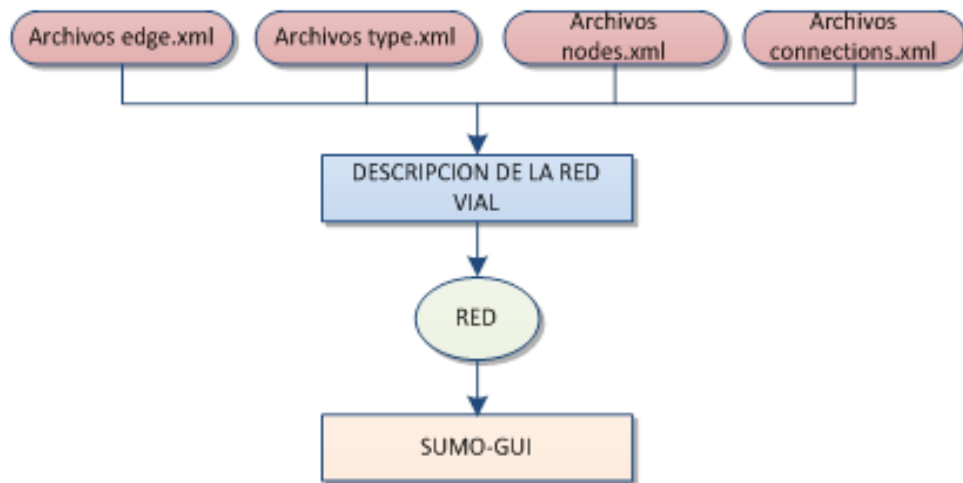


Fig. 33. Esquema para la generación manual de una red vial

ARCHIVO Node (*.nod.xml): Es el archivo que contendrá las instrucciones para las intersecciones y uniones de la red. Éste deberá tener la siguiente línea de comando con el siguiente formato:

```
< node id="<STRING>" x="<FLOAT>" y="<FLOAT>" type="<TYPE>" />
```

En [37], nos presenta el significado de cada uno de los atributos con su respectivo rango de valores, los tenemos en *Tabla 9*:

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>id</i>	El nombre del nodo
FLOAT	<i>x</i>	Determina la posición exacta en un plano cartesiano con origen en x=0
FLOAT	<i>y</i>	Determina la posición exacta en un plano cartesiano con origen en y=0
<i>priority</i>	<i>type</i>	Intersección con semáforos.
<i>traffic_light</i>		Intersección en la cual, el vehículo con más derecho a circular tenga prioridad.

Tabla 9. Comandos de generación de un archivo node

ARCHIVO Edge (*.edg.xml): Es el archivo que contendrá las vías de la red, este deberá tener la siguiente línea de comando con el siguiente formato:

```
<edge id="<STRING>" (fromnode="< NODE_ID1 >"
tonode="<NODE_ID2>" | xfrom="< FLOAT >" yfrom="< FLOAT >" xto="<
FLOAT >" yto="< FLOAT >" ) [type= "<tipo>"]/> | priority="<INT>"
nolanes="<INT>" speed="<FLOAT>" [lenght="<FLOAT>"]
[shape="<2D_POINT>"] [spread_type="<DIRECTION>"] />
```

El significado de cada uno de los atributos con su respectivo rango de valores, los tenemos la *Tabla 10*:

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>id</i>	El nombre del edge.
	<i>from</i>	Indica en nombre del node de donde nace el edge.
	<i>To</i>	Indica en nombre del node de donde finaliza el edge.
	<i>type</i>	Indica el nombre de un tipo de edge definido en un archivo *.typ.xml.
INT	<i>nolanes</i>	Número de carriles.
FLOAT	<i>speed</i>	Máxima velocidad permitida en la vía en m/s.
INT	<i>priority</i>	La prioridad del edge.
FLOAT	<i>length</i>	La longitud del edge en metros.

Tabla 10. Comandos de generación de un archivo node

ARCHIVO Type (*.typ.xml): Determina, de manera global, las propiedades de los edge. La línea de comando deberá contener el siguiente formato:

```
<type id="<STRING>" nolanes="<INT>" speed="<FLOAT>"
priority="<UINT>"/>
```

El significado de cada uno de los atributos con su respectivo rango de valores, los tenemos la *Tabla 11*:

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>id</i>	El nombre del type.
INT	<i>nolanes</i>	Número de carriles del edge.
FLOAT	<i>speed</i>	Máxima velocidad permitida en la vía en m/s.
INT	<i>priority</i>	La prioridad del edge.

Tabla 11. Comandos de generación de un archivo type

ARCHIVO Connection (*.con.xml): Determina las propiedades de las conexiones vías entre edge. La línea de comando deberá contener el siguiente formato:

```
<connection      from="<FROM_EDGE_ID>"      to="<T0_EDGE_ID>"
lane="<INT_1>:<INT_2>"/>
```

El significado de cada uno de los atributos con su respectivo rango de valores, los tenemos la *Tabla 12*:

Nombre del Atributo	Descripción
<i>from</i>	Nombre del edge que el vehículo abandona.
<i>to</i>	Nombre del edge que el vehículo ingresa.
<i>lane</i>	Número de carriles que serán conectados, estas son contadas de derecha a izquierda.

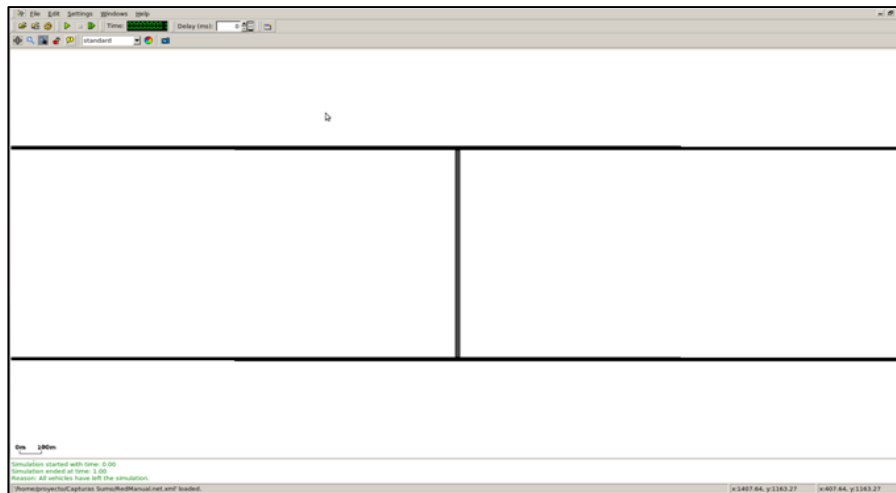
Tabla 12. Comandos de generación de un archivo connection

Al definir todos los componentes necesarios para la creación de una red vial, se los agrupa en un archivo de configuración (véase *Fig. 34*), de tal manera que NETCONVERT genere un archivo final para utilizarlo en sumo. En la *Fig. 35* se puede observar un ejemplo de una red vial generada manualmente.

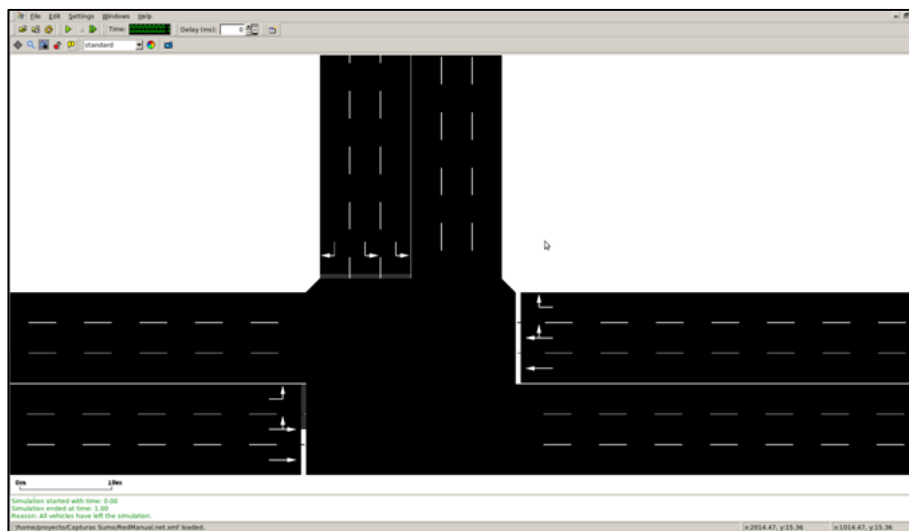
```

1 <configuration>
2   <input>
3     <edge-files value="RedManual.edg.xml"/>
4     <node-files value="RedManual.nod.xml"/>
5     <type-files value="RedManual.typ.xml"/>
6     <connection-files value="RedManual.con.xml"/>
7   </input>
8   <output>
9     <output-file value="RedManual.net.xml"/>
10  </output>
11  <processing>
12    <no-turnarounds value="true"/>
13  </processing>
14 </configuration>
15
```

Fig. 34. Archivo de configuración de una red vial generada manualmente



(a)



(b)

Fig. 35. Red vial generada manualmente: a) Zoom 100m, b) Zoom 10m

- **Generación de Archivos XML mediante importación.**

SUMO es capaz de importar mapas viales de diferentes bases de datos. En nuestro trabajo utilizamos OpenStreetMap creado por Wiki contributors [25]. Lo que nos motivó a utilizar esta herramienta es su facilidad y su uso sin restricciones por su licencia Open Database License (ODbL). Al poner como ejemplo la importación del mapa alrededor de la Universidad Politécnica Salesiana Sede Cuenca, tendríamos a la Fig. 36.

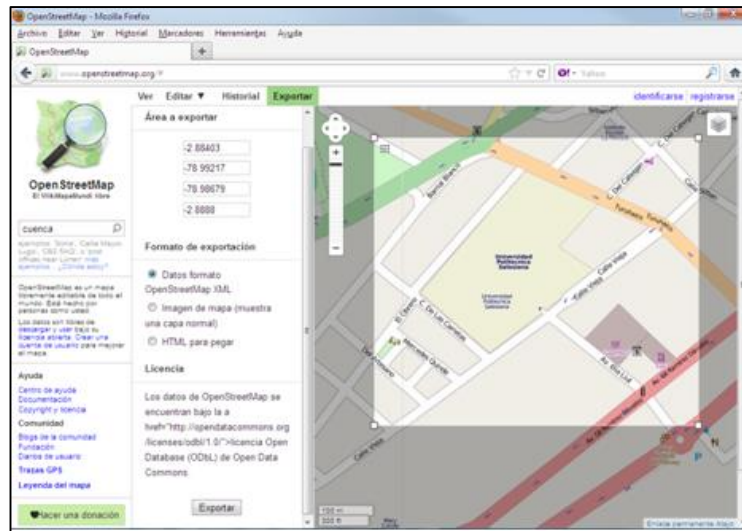


Fig. 36. Zona de la Universidad Politécnica Salesiana sede Cuenca con OpenStreetMap

Al momento de seleccionar la zona deseada del mapa vial de Cuenca, debemos seleccionar el menú Exportar, el cual nos permitirá la exportación del archivos en formato *.xml, lo que nos resulta un archivo en formato *.osm. (véase Fig. 37). Con el programa Java OpenStreetMap Editor (JOSM) se puede modificar el mapa importado como se puede observar en Fig. 37.

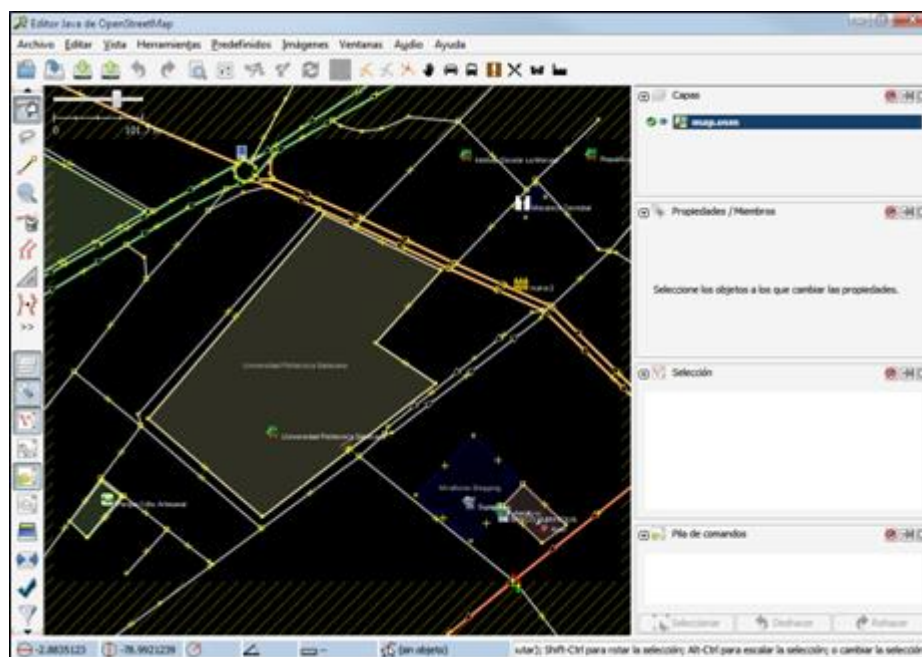


Fig. 37. Vista del mapa de la Universidad Politécnica Salesiana en JOSM

Una vez tengamos el archivo *.osm⁷ en nuestro computador, podemos utilizar a NETCONVERT⁸ para que el mapa vial sea compatible con SUMO, para ello utilizamos el siguiente comando:

```
netconvert -osm *.osm.xml -o *.net.xml
```

Una vez que con NETCONVERT o NETGEN se han encargado de generar un archivo *.net.xml”, el resultado puede ser visualizado a través de la aplicación SUMO-GUI.

4.1.1.2.2. Creación del Tránsito Vehicular

En [37], se enseña que la generación de tránsito debe ser contenida sobre un archivo *.rou.xml (véase Fig. 38). SUMO da a conocer los siguientes conceptos para la creación de tránsito vehicular:

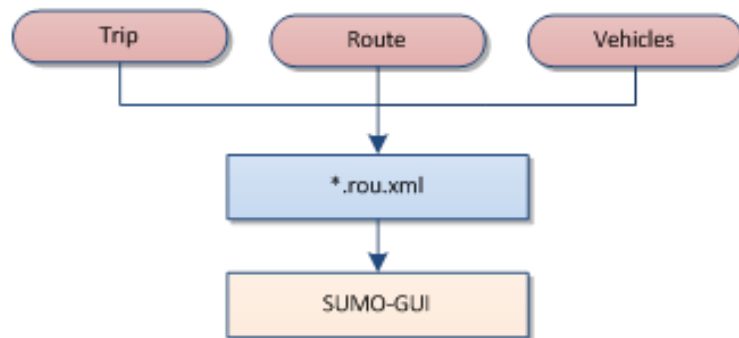


Fig. 38. Esquema para la generación del archivo *.rou.xml

- i. **Trip:** indica el movimiento del vehículo de un punto a otro definido por un edge de origen y uno de destino, con el tiempo de simulación.

⁷ En el comando de NETCONVERTER el símbolo * significa el nombre del archivo.

⁸ Las coordenadas de OpenStreetMap se encuentran en WGS84, NETCONVERT entrega la conversión automática a UTM.

- ii. **Route:** determina el recorrido del vehículo a través de distintos edge.
- iii. **Vehicles:** describe el tiempo de inicio y final de la simulación; además determina el nombre del vehículo.

SUMO establece varias formas o mecanismos para que los vehículos transiten por la red vial, entre las cuales podemos mencionar:

- i) Usando flujos que puede generar un flujo de vehículos entre un periodo o tiempo establecido.
- ii) Usando flujos y probabilidad de giro que permite definir la probabilidad de que un vehículo tome cualquiera de los caminos en una intersección, de esta manera se puede tener un ambiente más real y aleatorio.
- iii) Importando rutas de otros simuladores —al igual que las redes viales— [16]. Además, destacamos los siguientes mecanismos, por su facilidad de implementación:

- a) **Usando Trips (Viajes):** Como se mencionó, cada trip contiene un origen, un destino y el tiempo de salida. La sintaxis básica para la definición es la siguiente [37]:

```

<tripdef id="<ID>" depart="<TIME>" from="<ORIGIN_EDGE_ID>"
to="<DESTINATION_EDGE_ID>" [type="<VEHICLE_TYPE>"]
[period="<INT>" repno="<INT>"] [color="<COLOR>"]/>.
```

El significado de cada comando se contiene en la *Tabla 13*.

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>id</i>	El nombre de los vehículos.
INT	<i>depart</i>	El tiempo de salida del vehículo.
<EDGE ID>	<i>from</i>	El nombre del edge de inicio.
<EDGE ID>	<i>to</i>	El nombre del edge de fin.
INT	<i>period</i>	El tiempo de frecuencia de que otro vehículo con la misma ruta se inicie (opcional).
INT	<i>repro</i>	El número de vehículos que se insertan en una misma ruta (opcional)
COLOR	<i>color</i>	El color del vehículo generado.

Tabla 13. Comandos de generación de un archivo de rutas usando trips

Este archivo se debe introducir en la herramienta DUAROUTER, con el siguiente comando:

```
duarouter --trip-files [nombre_del_archivo_viajes].xml --net-file
[nombre_del_archivo_red_vial].net.xml --output-file
[nombre_del_archivo_salida].rou.xml
```

b) Usando rutas aleatorias: Esta opción utiliza un archivo en python para generar las rutas aleatorias sobre una red vial dada. La sintaxis básica para la definición es la siguiente [37]:

```
randomTrips.py -n [nombre_del_archivo_red_vial].net.xml -b 0 -e 300 -p 10 -
o [nombre_del_archivo_salida].rou.xml
```

El significado de cada comando se contiene en la *Tabla 14*.

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>-n</i>	El nombre de la red donde queremos generar las rutas.
INT	<i>-b</i>	Inicio del intervalo de viajes.
<EDGE ID>	<i>-e</i>	Fin del intervalo de viajes.
<EDGE ID>	<i>-p</i>	El número de viajes se define por la tasa de repetición.

Tabla 14. Comandos de generación de rutas aleatorias

- c) **Manualmente:** A diferencia del archivo de red (*.net.xml) que no se puede generar manualmente, el archivo de rutas (*.rou.xml) si se puede generar. Este archivo utiliza el lenguaje xml. La sintaxis básica para la definición es la siguiente [37]:

```
<routes>
  <vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5"
  maxSpeed="70"/>
</routes>
```

El significado de cada comando se contiene en la *Tabla 15*.

Valor/Tipo	Nombre del Atributo	Descripción
STRING	<i>id</i>	El nombre del vehículo.
FLOAT	<i>accel</i>	La aceleración del vehículo.
FLOAT	<i>decel</i>	La desaceleración del vehículo.
FLOAT	<i>sigma</i>	La imperfección conductor (entre 0 y 1)
FLOAT	<i>length</i>	El tamaño del vehículo
FLOAT	<i>maxSpeed</i>	La máxima velocidad del vehículo

Tabla 15. Comandos de generación de rutas manualmente

4.1.2. Implementación del Ambiente de Simulación Diseñado en ns-2.34

Ns-2 es un simulador orientado a objetos basado en dos lenguajes de programación: C++ y OTcl. El lenguaje OTcl es utilizado por el usuario para indicar las especificaciones del escenario que desea analizar; mientras que por otro lado, la implementación de los nuevos agentes se logra al modificar las librerías internas de ns-2 escritas en C++ [30]. Los tres agentes Master, Constancy y Members desarrollados en nuestro ambiente de simulación utilizan C++ como lenguaje de programación para describir todas sus funciones y módulos implementados. A continuación detallaremos las modificaciones que se deben realizar en la estructura interna de ns-2 para que el ambiente de simulación funcione adecuadamente.

4.1.2.1 Modificar fichero ns-default.tcl: Valores por defecto de parámetros de los agentes implementados

El fichero ns-default.tcl se encuentra en el directorio “.../ns-allinone-2.34/ns-2.34/tcl/lib/ns-default.tcl”. En este fichero se configuran todos los valores por defecto para los diferentes parámetros que definimos en los constructores de los agentes Master, Constancy, y Members respectivamente, utilizando el método bind() para el enlace de las variables entre C++ y OTcl.

Estos parámetros establecen un valor por defecto que se almacena en la plantilla de clase OTcl y que es copiado de ésta clase en la instancia cuando se crea el objeto que este asociado al agente específico y que solicite los valores de sus parámetros por defecto correspondientes. Para añadir una variable por defecto en el fichero ns-default.tcl se utiliza la siguiente nomenclatura [40]:


```
<Nombre de la clase> set <parámetro a configurar> <valor por defecto>
```

Para la implementación de nuestro ambiente de simulación, se han definido todos los valores por defecto de las variables correspondientes a cada agente en el fichero `ns-default.tcl`, véase la *Fig. 39*.

4.1.2.2 Modificar fichero `packet.h`: Configuraciones para las cabeceras implementadas

El fichero `packet.h` se encuentra en el directorio “`.../ns-allinone-2.34/ns-2.34/common/packet.h`”. En este fichero se deben realizar las siguientes modificaciones:

- **Modificar Clase: `p_info()`**

La primera modificación, véase *Fig. 40*, dentro de la clase `p_info()`, para la implementación del agente consiste en definir los identificadores para las cabeceras a ser utilizados. Por medio de la función `access(p)` accedemos a la cabecera específica del agente para un objeto de tipo `Packet * p`. [40]. La nomenclatura adecuada para realizar esta acción se expresa a continuación:

```
#define <identificador de clase>(p) <identificador cabecera> :: access(p)
```

```

1466 # ***** modificar aqui para tesis vncaza *****
1467
1468 # Defaults for Master
1469 Agent/Master set nodo_ID_ 0
1470 Agent/Master set dimension_X_ 1000
1471 Agent/Master set dimension_Y_ 1000
1472 Agent/Master set areatotal_X_ 1000
1473 Agent/Master set areatotal_Y_ 1000
1474 Agent/Master set inicio_X_ 0
1475 Agent/Master set inicio_Y_ 0
1476 Agent/Master set numero_filas_ 10
1477 Agent/Master set numero_columnas_ 10
1478 Agent/Master set region_ID_X_ 0
1479 Agent/Master set region_ID_Y_ 0
1480 Agent/Master set secuencia_ 0
1481 Agent/Master set cinematica_nodo_ 1
1482 Agent/Master set intervalo_estatico_ 0.1
1483 Agent/Master set tiempo_minimo_cambio_ 0.05
1484 Agent/Master set lider_ 0
1485 Agent/Master set estado_lider_ -1
1486 Agent/Master set tiempo_espera_lider_ 0.1
1487 Agent/Master set intervalos_heartbeat_ 0.1
1488 Agent/Master set intervalo_solicitud_lider_ 0.1
1489 Agent/Master set packetSize_ 10000
1490
1491 # Defaults for Constancy
1492 Agent/Constancy set puerto_cliente_ 17921
1493 Agent/Constancy set puerto_master_ 17923
1494 Agent/Constancy set numero_filas_ 8
1495 Agent/Constancy set numero_columnas_ 8
1496 Agent/Constancy set nodo_ID_ 0
1497 Agent/Constancy set lider_ 0
1498 Agent/Constancy set estado_lider_ -1
1499 Agent/Constancy set tespera_sinc_ 0.01
1500 Agent/Constancy set tintervalo_sinc_ 0.01
1501 Agent/Constancy set max_reintentos_ 3
1502 Agent/Constancy set habilitar_sinc_ 1
1503 Agent/Constancy set tretardo_sinc_ 0.01
1504 Agent/Constancy set tespera_envia_ 0.01
1505 Agent/Constancy set temp_vecino_ 0.01
1506 Agent/Constancy set packetSize_ 1000
1507
1508 # Defaults for Members
1509 Agent/Members set PUERTO_SERVIDOR_ 17923
1510 Agent/Members set nodo_ID_ 0
1511 Agent/Members set tiempo_espera_transmitir_ 0.01
1512 Agent/Members set estado_cliente_ -1
1513 Agent/Members set estado_lider_ -1
1514 Agent/Members set packetSize_ 1000
1515
1516 # ***** fin de las modificaciones *****

```

Fig. 39. Definiciones de variables por defecto para los agentes Master, Constancy y Members

```

71
72 #define HDR_LMS(p)          (hdr_lms::access(p))
73 ...
74 ...
75 ...
76 //***** Funciones requeridas para acceder a las cabeceras implementadas
77 #define HDR_ESTANDAR(p) (hdr_estandar::access(p))
78 #define HDR_MASTER(p) (hdr_Master::access(p))
79
80 //***** Fin de las modificaciones *****
81 ...

```

Fig. 40. Definición de PT_MASTER y PT_ESTANDAR en la clase p_info()

La clase *p_info()*, cuyas funciones internas permiten identificar al agente implementado mediante su nombre simbólico [19]. El constructor de un nuevo

agente detallado en la sección 3.3.3, necesita este parámetro para identificar a un paquete, PT_ESTANDAR y PT_MASTER parámetros asociados a las cabeceras `hdr_estandar` y `hdr_master` respectivamente. Estos identificadores al ser implementados para cada cabecera deben ser asociados a un determinado valor numérico tal como se aprecia en la Fig. 41. Además, en la Fig. 42, dentro del constructor de la clase `p_info()`, se debe agregar el identificador de las cabeceras.

```

194 // ANTNET packet
195 static const packet_t PT_ANT = 62;
196 ...
197 ...
198 ...
199 //***** modificar aqui *****
200 static const packet_t PT_ESTANDAR = 63;
201 static const packet_t PT_MASTER = 64;
202 //***** fin de las modificaciones *****
203 ...
204

```

Fig. 41. Definiciones de identificadores para PT_ESTANDAR y PT_MASTER

```

248 class p_info {
249 public:
250 //Realizar modificaciones dentro del constructor de la clase pinfo()
251 p_info()
252 {
253     initName();
254 }
255     ...
256     ...
257     ...
258     if (type == PT_ICP ||
259         type == PT_TELNET ||
260         type == PT_CBR ||
261         type == PT_AUDIO ||
262         type == PT_VIDEO ||
263         type == PT_ACK ||
264 //***** modificar aqui *****
265         type == PT_ESTANDAR ||
266         type == PT_MASTER ||
267 // ***** fin de las modificaciones *****
268         type == PT_SCTP ||
269         type == PT_SCTP_APP1 ||
270         type == PT_HDLC)
271         return DATApkt;
272     if (pc_)
273         return pc_->classify(type);
274     return UNCLASSIFIED;
275 }
276
277

```

Fig. 42. Definiciones de PT_ESTANDAR y PT_MASTER dentro de clase `p_info()`

Dentro de la clase *p_info()*, se debe modificar la función *initName()*, véase Fig. 43, con la finalidad de relacionar el nombre del paquete con el identificador de cada cabecera definida en la sección 3.3.1.

```

286         static void initName()
287         {
288             if(nPkt_ >= PT_NTTYPE+1)
289                 return;
290             char **nameNew = new char*[PT_NTTYPE+1];
291             for(unsigned int i = (unsigned int)PT_SMAC+1; i < nPkt_;
292             {
293                 nameNew[i] = name_[i];
294             }
295             if(!nPkt_)
296                 delete [] name_;
297             name_ = nameNew;
298             nPkt_ = PT_NTTYPE+1;
299
300
301             name_[PT_TCP]= "tcp";
302             name_[PT_UDP]= "udp";
303             name_[PT_CBR]= "cbr";
304             ...
305             ...
306             ...
307             //***** modificar aqui *****
308             name_[PT_ESTANDAR] = "Estandar";
309             name_[PT_MASTER] = "Master";
310             // ***** fin de las modificaciones *****
311             ...
312
313             // ANINET patch
314             name_[PT_ANT] = "Ant";
315
316         }

```

Fig. 43. Modificaciones dentro de la función *initName()*

Dada la diferencia entre un agente de datos y un agente de encaminamiento, se ubica a *PT_ESTANDAR* y *PT_MASTER* en datos, véase Fig. 44.

```

338 //***** modificar aqui *****
339 #define DATA_PACKET(type) ( (type) == PT_TCP || \
340                             (type) == PT_MYPING || \
341                             (type) == PT_ESTANDAR || \
342                             (type) == PT_MASTER || \
343                             (type) == PT_TELNET || \
344                             (type) == PT_CBR || \
345                             (type) == PT_AUDIO || \
346                             (type) == PT_VIDEO || \
347                             (type) == PT_ACK || \
348                             (type) == PT_SCTP || \
349                             (type) == PT_SCTP_APP1 \
350                             )
351 // ***** fin de las modificaciones *****
352

```

Fig. 44. Especificación de *PT_ESTANDAR* y *PT_MASTER* como paquete de datos

4.1.2.3 Modificar archivo ns-packet.tcl

El fichero ns-packet.tcl se encuentra en el directorio /ns-allinone-2.34/ns-2.34/tcl/lib/ns-packet.tcl”. En este fichero se registra a las nuevas cabeceras creadas en la sección 3.3.1. En la Fig. 45, se aprecia las modificaciones que se deben realizar para agregar nuestros paquetes en las lista de redes ad-hoc.

```
163 # Mobility, Ad-Hoc Networks, Sensor Nets:
164     AODV    # routing protocol for ad-hoc networks
165     Diffusion # diffusion/diffusion.cc
166     IMEP    # Internet MANET Encapsulation Protocol, for ad-hoc networks
167     MIP     # Mobile IP, mobile/mip-reg.cc
168     Smac    # Sensor-MAC
169     TORA    # routing protocol for ad-hoc networks
170     # AOMDV patch
171     AOMDV
172     Protoname
173
174     # ***** modificar aqui para tesis vncaza *****
175     Estandar # Referencia para el agente Estandar
176     Master   # Referencia para el agente Master
177     # ***** fin de las modificaciones *****
178
179     # ANTNET patch
180     Antnet
181 # Other:
182     Encap    # common/encap.cc
183     IPinIP   # IP encapsulation
184     HDLC     # High Level Data Link Control
185 } {
186     add-packet-header $prot
187 }
```

Fig. 45. Modificaciones sobre el fichero ns-packet.tcl

4.1.2.4 Modificar archivos cmu-trace.h y cmu-trace.cc

Los objetos *cmu-trace* se pueden utilizar para rastrear paquetes en escenarios con enlaces inalámbricos [19]. Estos objetos se utilizan para el seguimiento de los paquetes que son recibidos, descartados o enviados por los agentes, routers, capas mac o colas de interfaz en ns-2.

Dentro del fichero cmu-trace.h, localizado dentro del directorio de ns-2 en “.../ns-allinone-2.34/ns-2.34/trace/cmu-trace.h”, véase la Fig. 46, se debe añadir en la clase *CMUTrace Class* las funciones que definen los formatos para las trazas

correspondientes a las cabeceras de los paquetes de nuestro ambiente de simulación [33].

```
115 class CMUTrace : public Trace {
116 public:
117     //----- definiciones de argumentos publicos -----
118
119 private:
120     char    tracename[MAX_ID_LEN + 1];
121     int     nodeColor[MAX_NODE];
122     int     tracetype;
123     MobileNode *node_;
124     int     newtrace_;
125     /*----- codigo omitido declaraciones de variables y cabeceras-----*/
126     void    format_imep(Packet *p, int offset);
127     void    format_aadv(Packet *p, int offset);
128     //***** modificar aqui para tesis vncaza *****
129     void    format_estandar(Packet *p, int offset);
130     void    format_master(Packet *p, int offset);
131
132     //***** fin de modificaciones realizadas *****
133
134     // This holds all the tracers added at run-time
135     static PacketTracer *pktTrc_;
136 };
137
```

Fig. 46. Modificaciones sobre el fichero cmu-trace.h

Una vez definidas las funciones en la cabecera del objeto *cmu-trace*, se modifica las clases del archivo *cmu-trace.cc* ubicado en el directorio “*../ns-allinone-2.34/ns-2.34/trace/cmu-trace.cc*”. En este fichero se debe realizar las siguientes modificaciones:

- Se debe agregar la cabecera implementada dentro de ambiente de simulación *vncaza_pkt.h*, tal como se aprecia en la *Fig. 47*.

```
58 #include <simulator.h>
59 #include <antnet/ant_pkt.h>
60 //***** modificar aqui para tesis vncaza *****
61 #include <vncaza/vncaza_pkt.h>
62 //***** fin de modificaciones *****
63 #include <protoname/protoname_pkt.h>
```

Fig. 47. Incluimos el fichero *vncaza_pkt.h* que contiene las cabeceras en *cmu-trace.cc*

- En la función `void CMUTrace::format(Packet* p, const char *why){...}`, tal como se aprecia en la *Fig. 48*, se debe agregar los argumentos `PT_ESTANDAR` y `PT_MASTER` definidos como parámetros de entrada en los constructores de los agentes implementados, estos son asociándolos con un tipo determinado de paquetes para que sea reconocido por ns-2.

```

1408 void CMUTrace::format(Packet* p, const char *why)
1409 {
1410     hdr_cmn *ch = HDR_CMN(p);
1411     int offset = 0;
1412
1413     /*
1414      * Log the MAC Header
1415      */
1416     format_mac_common(p, why, offset);
1417
1418     if (pt_>namchannel())
1419         nam_format(p, offset);
1420     offset = strlen(pt_>buffer());
1421     switch(ch->ptype()) {
1422     case PT_MAC:
1423     case PT_SMAC:
1424         break;
1425     ...
1426     ...
1427     ...
1428     //***** modificar aqui para tesis vncaza *****
1429     case PT_ESTANDAR:
1430         format_estandar(p, offset);
1431         break;
1432     case PT_MASTER:
1433         format_master(p, offset);
1434         break;
1435     //***** fin de la modificacion realizada *****
1436     ...

```

Fig. 48. Llamamos a las funciones agregadas en `cmu-trace.h`

- De acuerdo a las cabeceras creadas para cada agente y descritas en el capítulo 3, se seleccionan los parámetros que se desean visualizar en el fichero de trazas (`*.tr`) definido en el script (`*.tcl`).

Existen tres formatos de trazas diferentes que se han programado: trazas etiquetadas, nuevas trazas y trazas de formato clásico [33]. La sintaxis para cada traza es diferente, para trazas etiquetadas y nuevas trazas existen identificadores para cada campo de información que se imprime. De acuerdo a [33], se aconseja utilizar los siguientes parámetros para visualización de

trazas "o" como dirección de origen, "s", como número de secuencia y "l" como la longitud del paquete correspondiente.

En caso de utilizar un paquete con el identificador PT_ESTANDAR, véase la Fig. 49, se crea un puntero con la cabecera `hdr_estandar` y se establecen los parámetros que se desean visualizar en el fichero de trazas, el mismo procedimiento se realiza con el identificador `PT_MASTER`.

```
1331 //***** modificar aqui para tesis vncaza *****
1332 //modificacion para agente estandar
1333 void
1334 CMUTrace::format_estandar(Packet *p, int offset)
1335 {
1336     struct hdr_estandar *ah = HDR_ESTANDAR(p);
1337
1338     if (pt_>tagged()) {
1339         sprintf(pt_>buffer() + offset,
1340             "-estandar:o %d -estandar:s %d -estandar:l %d ",
1341             ah->emisor(), ah->destino(), ah->tipo());
1342     }
1343     else if (newtrace_) {
1344         sprintf(pt_>buffer() + offset,
1345             "-P estandar -Po %d -Ps %d -Pl %d ",
1346             ah->emisor(), ah->destino(), ah->tipo());
1347     }
1348     else {
1349         sprintf(pt_>buffer() + offset,
1350             "[estandar %d %d %d] ",
1351             ah->emisor(), ah->destino(), ah->tipo());
1352     }
1353 }
1354 //modificacion para agente master
1355 void
1356 CMUTrace::format_master(Packet *p, int offset)
1357 {
1358     struct hdr_Master *ah = HDR_MASTER(p);
1359
1360     if (pt_>tagged()) {
1361         sprintf(pt_>buffer() + offset,
1362             "-master:o %d -master:s %d -master:l %d ",
1363             ah->emisor(), ah->secuencia(), ah->destino());
1364     }
1365     else if (newtrace_) {
1366         sprintf(pt_>buffer() + offset,
1367             "-P master -Po %d -Ps %d -Pl %d ",
1368             ah->emisor(), ah->secuencia(), ah->destino());
1369     }
1370     else {
1371         sprintf(pt_>buffer() + offset,
1372             "[master %d %d %d] ",
1373             ah->emisor(), ah->secuencia(), ah->destino());
1374     }
1375 }
1376
```

Fig. 49. Modificaciones en fichero `cmu-trace.cc` para visualización de trazas

4.1.2.5 Compilación y detección de errores en la Implementación

Para asegurar que el ambiente de simulación funcione de forma adecuada es importante considerar las siguientes observaciones:

- Los ficheros del simulador (anexo [3]) deben estar colocados dentro del directorio “.../ns-allinone-2.34/ns-2.34/vncaza/”. En el fichero Makefile se agrega los archivos de nuestro ambiente de simulación. Makefile se ubica en el directorio “.../ns-allinone-2.34/ns-2.34/” [33], véase la Fig. 50.

```
161 # WIN32: uncomment the following line to include specific make for VC++
162 # !include <conf/makefile.win>
163
164 OBJ_CC = \
165     tools/random.o tools/rng.o tools/ranvar.o common/misc.o common/timer-handler.o \
166     common/scheduler.o common/object.o common/packet.o \
167     common/ip.o routing/route.o common/connector.o common/ttl.o \
168     trace/trace.o trace/trace-ip.o \
169     ...
170     ...
171     ...
172     xcp/xcpq.o xcp/xcp.o xcp/xcp-end-sys.o \
173     wpan/p802_15_4csmaca.o wpan/p802_15_4fail.o \
174     wpan/p802_15_4hlist.o wpan/p802_15_4mac.o \
175     wpan/p802_15_4nam.o wpan/p802_15_4phy.o \
176     wpan/p802_15_4sscs.o wpan/p802_15_4timer.o \
177     wpan/p802_15_4trace.o wpan/p802_15_4transac.o \
178     apps/pbc.o \
179     antnet/antnet_common.o antnet/antnet_rtable.o antnet/antnet.o \
180
181     #----- modificar aqui-----
182     vncaza/constancy.o vncaza/members.o vncaza/resultados.o vncaza/master.o \
183     # ----- fin de las modificaciones -----
184
185     linkage/ex-linkage.o \
186     apps/myping.o \
187     protoname/protoname.o protoname/protoname_rtable.o)
188 $(OBJ_STL)
189
```

Fig. 50. Modificaciones en el fichero Makefile

- Debido a que se modificó el fichero packet.h, en el terminal se debe ejecutar el comando “*touch common/packet.cc*” (esto cuando exista inserción de nuevos paquetes). Además, se ejecuta el comando *Make* desde el terminal para compilar toda la implementación del ambiente de simulación y descartar cualquier error. En la Fig. 51, se muestra un ejemplo de la compilación.



Fig. 51. Compilación y Detección de errores.

4.1.3. Elaboración de un script para Simulación de Enlaces Inalámbricos en ns-2

Ns-2, es una herramienta muy potente dentro del campo de la simulación de redes, ns-2 utiliza el lenguaje OTcl para realizar los scripts⁹ de simulación. En dichos scripts se especifican los parámetros necesarios para los escenarios como: topología, protocolos, características de los enlaces, modelos de tránsito y la planificación de los eventos. Este lenguaje permite efectuar cambios en el escenario de simulación de manera rápida e interactiva.

Una descripción básica del contenido interno de un script Otcl se muestran en la *Fig. 52*, en la cual se distinguen tres partes principales en el diseño del script que son: la cabecera de inicialización, el cuerpo del script y el proceso de finalización [29]. Como se observa en la *Fig. 52*, cada subconjunto en la descripción del script contiene elementos que permiten establecer condiciones específicas para la simulación. La descripción de cada uno de estos elementos se realizará a continuación:

⁹ Script: es un fichero de extensión *.tcl en el cual se configura el escenario de simulación por parte del usuario.

CABECERA	Objetos Clase Simulador
	Archivo de Traza (*.tr)
	Archivo para NAM (*.nam)
CUERPO	Definición de variables globales y constantes
	Definición del escenario de simulación
	Configuración de las capas físicas y MAC de los nodos inalámbricos
	Creación de la topología de la red
	Ubicación de los nodos y generación de movimiento
	Definición de agentes y aplicaciones
	Organizador de Eventos
FINALIZACIÓN	Procedimiento de finalización

Fig. 52. Descripción interna de un script para simulación en ns-2 [29].

4.1.3.1. Cabecera de inicialización del script:

Esta sub sección del script sirve para crear un objeto de la clase simulador a partir del cual se podrán invocar los procedimientos de esta clase [29]. También en esta cabecera del script se definen los archivos que almacenan los datos generados como resultados luego de la simulación siendo estos del tipo *.tr para manejo de trazas, *.nam para análisis con la herramienta, The Network Animator (NAM), o en nuestro caso crear un fichero llamado resultados para imprimir eventos. En la Fig. 53, se observan las líneas de programación necesarias para implementar los elementos de la cabecera del script. Los elementos trace-all y namtrace-all-wireless permiten almacenar las trazas y los datos para dar solvencia al enlace inalámbrico [19].

```

31 # =====
32 # Programa principal (Elementos de la cabecera)
33 # =====
34 file delete $val(fichero)
35 #Creacion de objeto ns y ficheros de visualizacion de resultados
36 set ns [new Simulator] ;#Creamos la instancia del simulador
37 set tracefd [open respuestap4.tr w] ;#Creamos el archivo de la traza
38 set namtrace [open respuestap4.nam w] ;#Creamos el archivo para el NAM
39 #Procedimientos para almacenar las trazas en un formato
40 $ns trace-all $tracefd
41 $ns namtrace-all-wireless $namtrace $val(x) $val(y)

```

Fig. 53. Descripción de los principales elementos que forman parte de la cabecera de un script para un enlace inalámbrico.

4.1.3.2. Cuerpo del script:

El cuerpo del script incluye los elementos que permiten dar soporte al diseño de la red a simular, podemos mencionar a los siguientes:

- **Definición de variables globales y constantes para los agentes implementados**

En esta sección se define las características para el enlace inalámbrico que se utilizará en la simulación, entre estas variables tenemos: el tipo de canal, el modelo de propagación, tipo de MAC, definición de manejo de colas de paquetes, capa de enlace, modelo de antena, número de nodos móviles que intervienen en la simulación, protocolo de encaminamiento, dimensión de la topología y máximo tiempo de simulación. El resultado de la implementación de estas variables se aprecia en la *Fig. 54*.

```
1 # -----
2 # Definimos las características para el enlace inalámbrico
3 # -----
4 set val(chan)          Channel/WirelessChannel    ;# tipo de canal
5 set val(prop)         Propagation/TwoRayGround   ;# modelo de radio-propagation
6 set val(netif)        Phy/WirelessPhy           ;# tipo de interfaz de red
7 set val(mac)          Mac/802_11                ;# tipo de MAC
8 set val(ifq)          Queue/DropTail/PriQueue   ;# tipo de interfaz para colas
9 set val(ll)           LL                         ;# tipo de capa de enlace
10 set val(ant)          Antenna/OmniAntenna       ;# modelo de antena
11 set val(ifqlen)      50                         ;# max paquetes en fq
12 set val(nn)          20                         ;# numero de nodos mobiles
13 set val(rp)          AODV                       ;# protocolo de enturamiento
14 set val(x)           1000                       ;# dimension de la topologia en X
15 set val(y)           2000                       ;# dimension de la topologia en Y
16 set val(stop)        50                        ;# tiempo de simulacion
```

Fig. 54. Descripción de las características del medio de transmisión

Además, en esta sección, tal como se aprecia en la *Fig. 55*, se define los parámetros que caracterizan al nodo virtual por medio de las variables que representan longitud del paquete, fichero para visualización de resultados, dimensión de la rejilla y número de filas y columnas para elección de nodo líder, y habilitación de sincronización.

```

18 # =====
19 # Definimos las características para el nodo virtual
20 # =====
21 set val(filas)          5           ;# numero de filas en la region
22 set val(columnas)      4           ;# numero de columnas en la region
23 set val(fichero)       "respuestap4" ;# fichero para visualizar resultados
24 set val(paquete)       10000      ;# longitud del paquete enviado
25 set val(sinc_nodo)     1           ;# habilitamos la sincronizacion o no
26 set val(inicio_x)      0           ;# dimension de la topologia en X
27 set val(inicio_y)      0           ;# dimension de la topologia en Y
28 set val(dimension_x)   800        ;# dimension de la topologia en X
29 set val(dimension_y)   1600       ;# dimension de la topologia en Y
30

```

Fig. 55. Descripción de las características del nodo

- **Creación de la topología de la red**

Esta sección define el espacio de trabajo para el ambiente de simulación y la dimensión física para el escenario (delimita la ubicación y movimiento de los nodos), véase *Fig. 56*.

```

43 #Definimos un objeto para definir la topogia utilizada
44 set topo [new Topography]
45 $topo load_flatgrid $val(x) $val(y)

```

Fig. 56. Definimos el espacio de trabajo para el ambiente de simulación

Dado que en la topología de la simulación, el elemento fundamental son los nodos. Se debe definir un objeto denominada god (Director General de Operaciones), véase *Fig. 48*. El god es el objeto que se utiliza para almacenar información global sobre el estado del medio, la red e indicar el número de nodos que participaran dentro de la simulación. Este elemento además sirve para gestionar los detalles de las operaciones que suministramos como son los patrones de movimiento en nuestras simulaciones.

```

47 #Creamos el objeto GOD
48 create-god $val(nn) ;#donde $val(nn) representa numero total de nodos
49

```

Fig. 57. Creación de un objeto GOD

- **Definición de los elementos de las capas físicas y MAC para configurar el nodo virtual**

Crear un nodo implica una serie de características que se deben definir detalladamente, mediante la función *node-config()* (propia de ns-2), se pueden asignar los parámetros de configuración necesarios para cada nodo implementado (ver Fig. 58): protocolo de encaminamiento de los paquetes AODV, capa de enlace, tipo de MAC definido para 802.11, tipo de cola Droptail para descartar paquetes que llegan al buffer cuando sobrepasa la capacidad del mismo, longitud del buffer, tipo de antena utilizada, modelo de propagación TwoRayGround muy cercano a la realidad, tipo de canal, elementos para topología definida y parámetros para habilitar o denegar la visualización de datos en las trazas como el movimiento del nodo, el agente utilizado y el encaminamiento [29].

```

50 # =====
51 # Configuramos las características de los nodos virtuales
52 # =====
53 $ns node-config -adhocRouting $val(rp) \
54                 -llType $val(ll) \
55                 -macType $val(mac) \
56                 -ifqType $val(ifq) \
57                 -ifqLen $val(ifqlen) \
58                 -antType $val(ant) \
59                 -propType $val(prop) \
60                 -phyType $val(netif) \
61                 -channelType $val(chan) \
62                 -topoInstance $topo \
63                 -agentTrace ON \
64                 -routerTrace ON \
65                 -macTrace OFF \
66                 -movementTrace ON
67
68 # Creamos un numero especifico de nodos dado por el parametro establecido en la cabecera
69 for {set i 0} {$i < $val(nn)} { incr i } {
70     set node_($i) [$ns node]
71 }
72

```

Fig. 58. Configuración y definición de todos los nodos definidos para la topología.

Al tratar un gran número de simulaciones con una cantidad representativa de nodos se usa un bucle para su creación *set <nombre del nodo> [\$ns node]*, para el cual el nombre del nodo debe contener el identificador de iteraciones del bucle de tal manera que cada vez que el bucle se repita permita crear un elemento único, tal como se aprecia en las líneas de programación 68 a la 71 de la Fig. 58.

- **Ubicación de los nodos y generación de movimiento**

Una vez creados todos los nodos y configurados sus principales parámetros con las características deseadas en la simulación, se debe especificar la posición que ocupará cada uno de ellos en un sistema de referencia. Para establecer la posición inicial de los nodos se hace uso de las siguientes funciones:

```
$<identificador del nodo> set X_ <coordenada inicial en X>  
$<identificador del nodo> set Y_ <coordenada inicial en Y>  
$<identificador del nodo> set Z_ <coordenada inicial en Z>
```

En muchos casos se suele considerar al escenario de simulación como un sistema bidimensional, con lo cual la ubicación en la coordenada Z no suele considerarse asignándole un valor referente de Z=0. Un ejemplo de ubicación para tres nodos se aprecia en la *Fig. 50*.

```
73 # =====  
74 # Posiciones iniciales de los nodos  
75 # =====  
76 #Nodo 0: primero en fila  
77 $node_(0) set X_ 50.0  
78 $node_(0) set Y_ 1550.0  
79 $node_(0) set Z_ 0.0  
80 #Nodo 1: segundo en fila  
81 $node_(1) set X_ 50.0  
82 $node_(1) set Y_ 1400  
83 $node_(1) set Z_ 0.0  
84 #Nodo 2: tercero en fila  
85 $node_(2) set X_ 50.0  
86 $node_(2) set Y_ 1000.0  
87 $node_(2) set Z_ 0.0  
88
```

Fig. 59. Configuración y definición de todos los nodos definidos para la topología.

Dado que el análisis y pruebas están enfocadas a nodos móviles es importante definir la movilidad que tendrán los diferentes nodos durante el tiempo de simulación especificándose este mediante la siguiente función:

```
<$ns at $time $node setdest < coordenada X > <coordenada Y> <velocidad  
movimiento>
```

En donde la variable \$time define el tiempo de movimiento del nodo (\$node) desde una posición inicial definida con anterioridad hasta la nueva posición dada por las coordenadas X y Y a una velocidad limitada y fija expresada en metros por segundo, tal como se aprecia en el ejemplo de movilidad de tres nodos dado en la *Fig. 60*.

```
89 # -----  
90 #           Generacion de movimiento de los nodos  
91 # -----  
92 #Movimientos del nodo 0  
93 $ns at 2.0 "$node_(0) setdest 50.0 400.0 5.0"  
94 $ns at 11.0 "$node_(0) setdest 50.0 350.0 5.0"  
95 #Movimientos del nodo 1  
96 $ns at 2.0 "$node_(1) setdest 50.0 420.0 5.0"  
97 $ns at 11.0 "$node_(1) setdest 50.0 370.0 5.0"  
98 #Movimientos del nodo 2  
99 $ns at 2.0 "$node_(2) setdest 50.0 430.0 5.0"  
100 $ns at 11.0 "$node_(2) setdest 50.0 390.0 5.0"  
101
```

Fig. 60. Configuración de patrones de movimiento de todos los nodos definidos para la topología.

- **Definición de agentes y aplicaciones**

Ésta sección del script (véase *Fig. 61*), contiene a los tres agentes implementados Master, Constancy y Members, Le agregamos a cada nodo los tres agentes implementados asignando un puerto para la comunicación haciendo uso de la función *attach*. Además, inicializamos las actividades de los agentes y habilitamos el fichero resultados para visualización de respuestas de los agentes.

4.1.3.3. Procedimiento de Finalización:

Esta sección del script (véase Fig. 62) se describe las siguientes acciones: reiniciar el estado de los nodos utilizados en la simulación a su estado original (al concluir la simulación), detener con el tiempo límite para la simulación los datos que llegan al NAM, y mediante la función `stop()` creación de datos en los archivos de la trazas.

```
102 # =====
103 # Configuramos de agentes y procesos
104 # =====
105 for {set i 0} {$i < $val(nn)} {incr i} {
106
107 # Establecemos una comunicacion de la capa virtual entre los nodos
108 set master_($i) [new Agent/Master]
109 set constancy_($i) [new Agent/Constancy]
110 set members_($i) [new Agent/Members]
111
112 #Configuramos parametros para el agente Master
113 $master_($i) set nodo_ID_ $i
114 $master_($i) set intervalo_estatico_ 0.001
115 $master_($i) set tiempo_minimo_cambio_ 0.001
116 ...
117
118 #Configuramos parametros para el agente Constancy
119 $constancy_($i) set nodo_ID_ $i
120 $constancy_($i) set habilitar_sinc_ $val(sinc_nodo)
121 $constancy_($i) set numero_filas_ $val(filas)
122 ...
123
124 #Configuramos parametros para el agente Members
125 $members_($i) set nodo_ID_ $i
126
127 #Configuramos los puertos usados por los agentes implementados
128 $node_($i) attach $master_($i) [$master_($i) set num_puerto_]
129 $node_($i) attach $constancy_($i) [#[$constancy_($i) set num_puerto_]
130 $node_($i) attach $members_($i) [#[$members_(0) set num_puerto_]
131 ...
132
133 #Habilitamos la visualizacion de ficheros para los tres agentes implementados
134 $ns at 0.0 "$master_($i) set-trazaArchivoNombre $val(fichero)"
135 $ns at 0.0 "$constancy_($i) set-trazaArchivoNombre $val(fichero)"
136 $ns at 0.0 "$members_($i) set-trazaArchivoNombre $val(fichero)"
137 $ns at 0.0 "$constancy_($i) start"
138 $ns at 0.0 "$members_($i) start"
139
140 }
```

Fig. 61. Configuración de agentes Master, Constancy y Members

```
178 #Reiniciamos el estado de los nodos cuando la simulacion termina
179 for {set i 0} {$i < $val(nn)} {incr i} {
180     $ns at $val(stop) "$node_($i) reset";
181 }
182
183 # Finalizamos el nam y la simulacion
184 $ns at $val(stop) "$ns nam-end-wireless $val(stop)"
185 $ns at $val(stop) "stop"
186 $ns at 50.01 "puts \"end simulation\" ; $ns halt"
187
188 proc stop {} {
189     global ns tracefd namtrace
190     $ns flush-trace
191     close $tracefd
192     close $namtrace
193 }
194
195 $ns run
```

Fig. 62. Definición de procesos para finalización de simulación en el script

4.2. Diseño de Experimentos sobre un Nodo Virtual

Esta sección describe el desempeño del ambiente de simulación diseñado. Para ello, se implementaron cuatro pruebas. La primera prueba consta de un escenario de simulación con nodos estáticos. La segunda prueba es un escenario de simulación MANET. La tercera es un escenario de simulación VANET y la cuarta prueba realiza la revisión de comunicación de los tres agentes implementados. Las tres primeras pruebas nos servirán para realizar un análisis del algoritmo de elección del nodo líder en la una región.

a) Escenario de simulación con nodos estáticos

El escenario de simulación estacionario está compuesto por nodos sin movimiento, véase *Fig. 63*. El escenario tiene una topografía de 700x700m. Además, está compuesto por 10 nodos en un tiempo de simulación de 0s a 120s. Los nodos se equipan con una interfaz de red inalámbrica IEEE 802.11. Las características del canal inalámbrico se establecen como se muestran en la *Tabla 16*.

Parámetros	Valores
Tiempo de simulación	0-120s
Numero de nodos	10
Tamaño de la topología	700x700m
Propagación	TwoRayGround
Tipo de Interfaz de Cola	DropTail/PriQueue
Modelo de Antena	OmniAntenna
Potencia de Transmisión	0.28183815
Frecuencia	914 MHz
Ganancia de Antena de Transmisión	1
Ganancia de Antena de Recepción	1
Perdidas del sistema	1
Altura de la Antena de Transmisión	1.5
Altura de la Antena de Recepción	1.5
Umbral de Potencia de Recepción (RXThresh)	7.6911e-8 (50m)
Umbral de Potencia de Transmisión (CSThresh)	1.9228e-8 (100m)

Tabla 16. Valores de la simulación del canal inalámbrico IEEE 802.11 para el escenario de simulación de nodos Estáticos

El script de simulación está formado por el agente Master descrito en la sección 3.3. El agente Master contiene las características presentadas en la Tabla 17, las cuales son las mismas para la simulación MANET y VANET, esto nos servirá para realizar una comparación entre las tres pruebas de redes ad-hoc. De la Fig. 63, se ubicó a la rejilla en las coordenadas de inicio [0;0]. El tamaño de rejilla es de 200x200m con 4 columnas y 4 filas. Así, el tamaño de cada región cuadrada será de 50x50m.

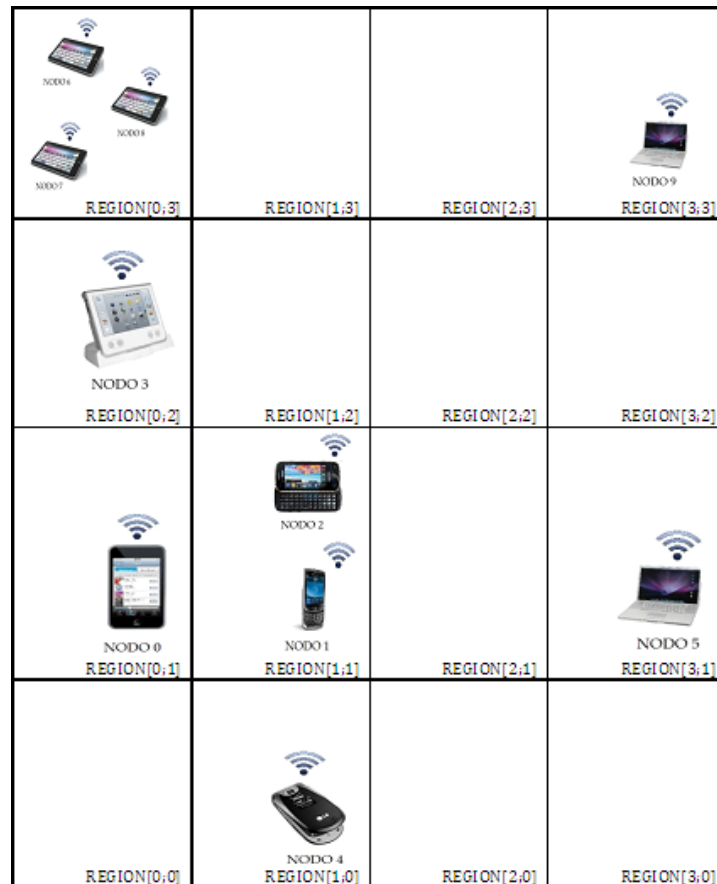


Fig. 63. Escenario de simulación de nodos estáticos

Parámetros	Valores
Intervalo Estático	1 s
Intervalo entre Heartbeat	0.5 s
Intervalos de Solicitud de liderazgo	1 s
Tiempo de Espera de Mensaje del líder	0.5 s

Tabla 17. Valores de la Simulación del Agente Master para el escenario de simulación de nodos Estáticos

El modelo de propagación implementado TwoRayGround (Modelo de dos rayos) en el escenario de simulación, es usado para predecir la potencia de la señal recibida por cada paquete. En la capa física de cada nodo existe un umbral de potencia de recepción. Ns-2 realiza la tarea de comprobar la potencia que contiene el paquete receptado, para considerar a dicho paquete como valido o invalido, ns-2 compara los niveles de umbral de potencia RXThresh (Receive power threshold) y CSThresh (Carrier Sense Threshold) [19]. Al considerar, que si la potencia recibida es mayor que RXThresh, el paquete se recibirá correctamente. Cuando la potencia se encuentra entre RXThresh y CSThresh, el paquete se detecta pero no puede ser interpretado. Si la potencia del paquete recibido se encuentra por debajo del límite inferior CSThresh, el mensaje no es detectado [19], la interpretación grafica se presenta en la Fig. 64.

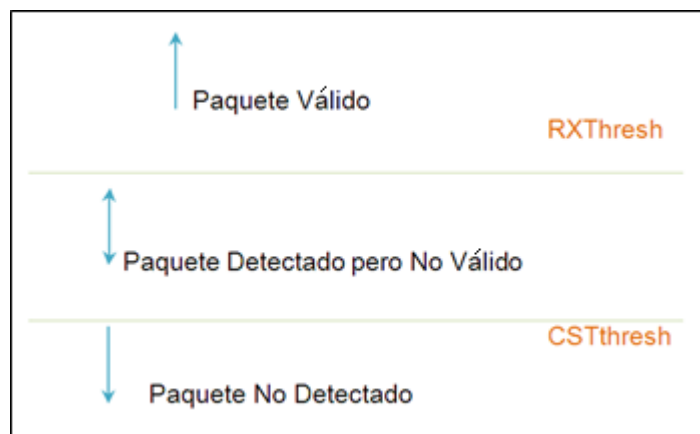


Fig. 64. Umbrales de detección de transmisión y recepción de paquetes para un nodo [19]

Para calcular la potencia recibida se utiliza la ecuación de propagación Ec. 3. Esta ecuación entrega la potencia recibida en función de la distancia entre nodos. Así, en la Ec. 3. P_t es la Potencia de transmisión, G_t y G_r son las ganancias de las antenas de transmisión y recepción, respectivamente. Además, h_t y h_r son las alturas de las antenas de transmisión y recepción, respectivamente. L es la pérdida del sistema, y d la distancia de los nodos. Así se determina los valores de RXThresh en 50m y CSThresh en 100m, véase Tabla 16.

$$\Pr(d) = \frac{Pt*Gt*Gr*h_t^2*h_r^2}{d^4L}$$

Ec. 3

- **Resultados de la simulación.**

Esta sección muestra los resultados de la simulación del escenario con nodos estáticos. En este escenario todos los nodos se intercambian información de acuerdo a la región a la cual pertenecen. En la Fig. 65, se presenta el estado de los nodos que actuaron en la región [0; 3], en la figura se observa que los nodos 7, 8 y 9, mantuvieron un liderazgo en la región. Además, en la Tabla 18, se presenta el diagrama de cada uno de los estados de los nodos: 7, 8 y 9 a lo largo de la simulación.

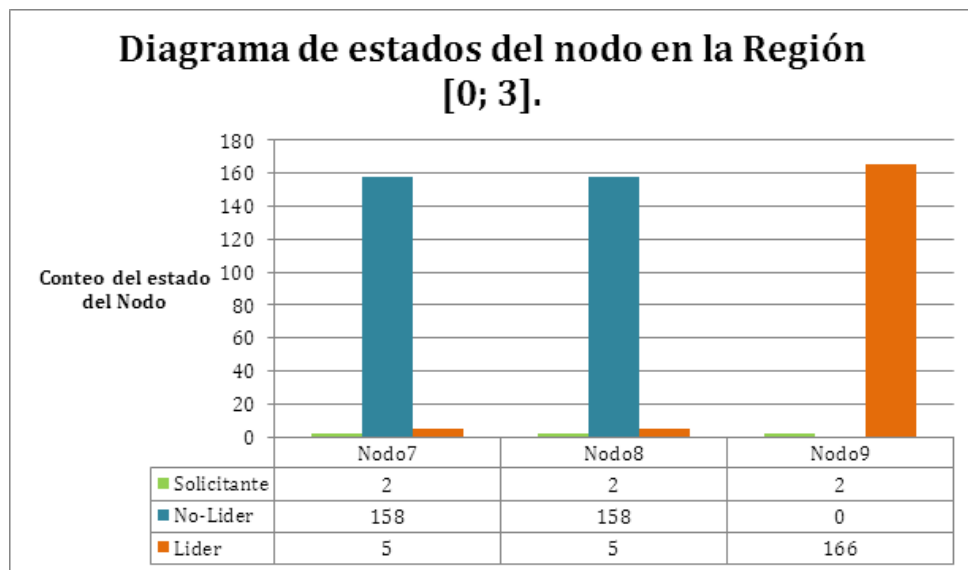


Fig. 65. Diagrama de estados del nodo en el escenario de simulación con nodo estáticos en la Región [0; 3]

La Tabla 18, determina los tiempos en que los nodos 7, 8 y 9 se establecen como *Líder*, *No Líder* y *Solicitante* en la región de análisis [0; 3]. El estado inicial de los tres nodos es *Solicitante*, para lo cual envían un mensaje tipo Master con la bandera

LeaderRequest. Al no haber un mensaje de respuesta de un nodo *Líder*, los nodos 7, 8 y 9 establecen su liderazgo en los tiempos de 0.16s, 0.16s y 0.08s respectivamente. Al establecer el liderazgo de los nodos 7, 8 y 9, estos comienzan a transmitir mensajes tipo Master con las banderas *Heartbeat* y *comienza_ldr*, ver sección 2.3. El nodo 9 se estableció como *Líder* en un tiempo menor al nodo 7 y al nodo 8. Con lo que, en 1.08s el nodo 7 receipta un paquete Master, con las banderas *Heartbeat* y *comienza_ldr*, y cambia su estado a *No Líder*. Además, en 1.16s el nodo 8 receipta un paquete Master, con las banderas *Heartbeat* y *comienza_ldr*, y cambia su estado a *No Líder*. Estos estados de *Líder* para el nodo 9 y *No Líder* para los nodos 7 y 8 permanecen hasta el final de la simulación. En Fig. 66, se presenta de manera gráfica cada uno de los estados de los nodos 7, 8 y 9 en el tiempo de simulación de 0s a 120s. En la gráfica, el nodo 7 corresponde al color rojo, el nodo 8 el color verde, el nodo 9 al color azul.

NODO	MUERTO	DESCONOCIDO	SOLICITANTE	LIDER	NO-LIDER	INESTABLE	REGIÓN
■ /			[0,00;0,16]	[0,16;1,16]	[2,00;120,00]		[0;3]
■ 8			[0,00;0,16]	[0,16;1,16]	[2,00;120,00]		[0;3]
■ 9			[0,00;0,08]	[0,08;120,00]			[0;3]

Tabla 18. Estado de los nodos 7, 8 y 9 en el tiempo de simulación [0s; 120s] en el escenario de simulación de nodos Estáticos

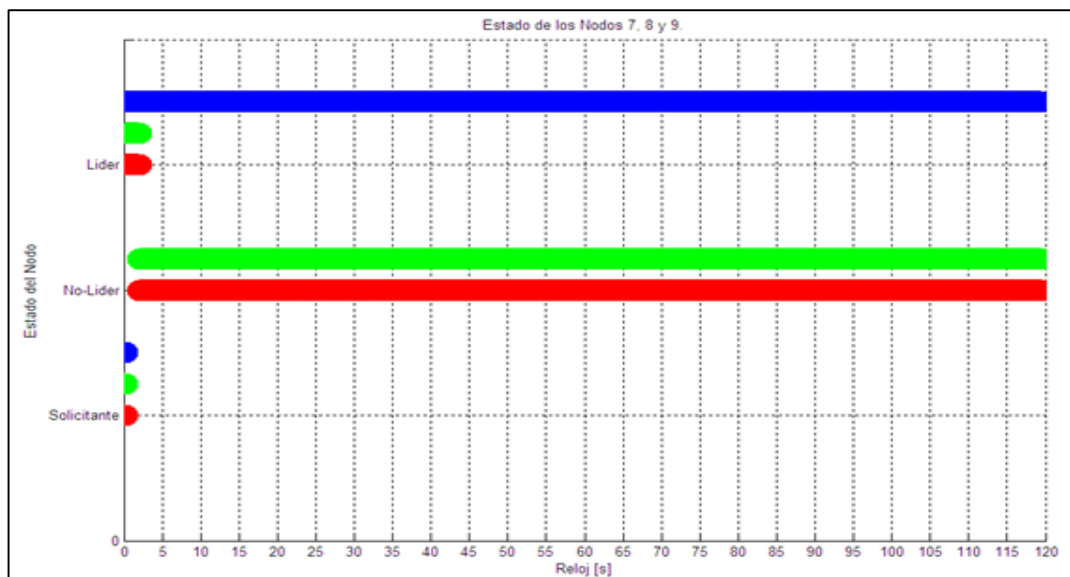


Fig. 66. Estado de los nodos 7, 8 y 9 en el tiempo de simulación [0s; 120s] en el escenario de simulación de nodos Estáticos

b) Escenario de simulación MANET

El escenario de simulación MANET está compuesto por dos partes: La primera consiste en la descripción de la topología propuesta en la cual se configuran todas las características de los nodos; mientras que la segunda únicamente contiene información del movimiento que pueden realizar los nodos durante la simulación.

Para este experimento, el escenario de simulación está compuesto por un total de 15 nodos distribuidos tal como se aprecia en la *Fig 67*. El tiempo total de simulación es de 150 segundos. Se ha establecido que la velocidad de los nodos se ubique entre 1 m/s y 5 m/s dado que esos datos son los valores promedios para un persona que se desplaza caminando y llevando consigo un dispositivo portátil como un celular¹⁰. Todos los nodos en el escenario de simulación han sido identificados con un número del 0 al 14, se han colocado estratégicamente con el fin de poder evaluar todos los posibles cambios de estado de un nodo al desplazarse de una región a otra. Además, los movimientos de los diferentes nodos para ir de un punto a otro son aleatorios, a excepción del nodo 0, el cual ha sido preseleccionado para tener un movimiento lineal, observar todas las transiciones en los estados que puede tomar y servir de objeto de estudio para los análisis posteriores.

Para el escenario de simulación detallado en el script `pruebasm1.tcl` (anexo [4.2]), se ha considerado una rejilla rectangular de 3 filas y 4 columnas, lo cual nos da una rejilla con un total de 12 posibles regiones (nodos virtuales) por las cuales un nodo se podría desplazar. La dimensión total de la rejilla es de 200 x 150 metros y cada región a su vez es cuadrada y tiene una dimensión de 50x50m. En la Tabla 23, se presentan las principales características de la topología para el experimento con MANETS.

Dado que nuestro ambiente de simulación utiliza una rejilla, compuesta por varias regiones rectangulares, ver Tabla 20, se delimitó el rango de transmisión para que los nodos no procesen todos los paquetes provenientes de todos los nodos desde los

¹⁰ BBC Ciencia: La velocidad al andar predice la longevidad.

cuales puedan recibir información dentro del escenario de simulación. Esto previene la sobrecarga innecesaria de las colas de paquetes de los nodos, y descartar de paquetes receptados.

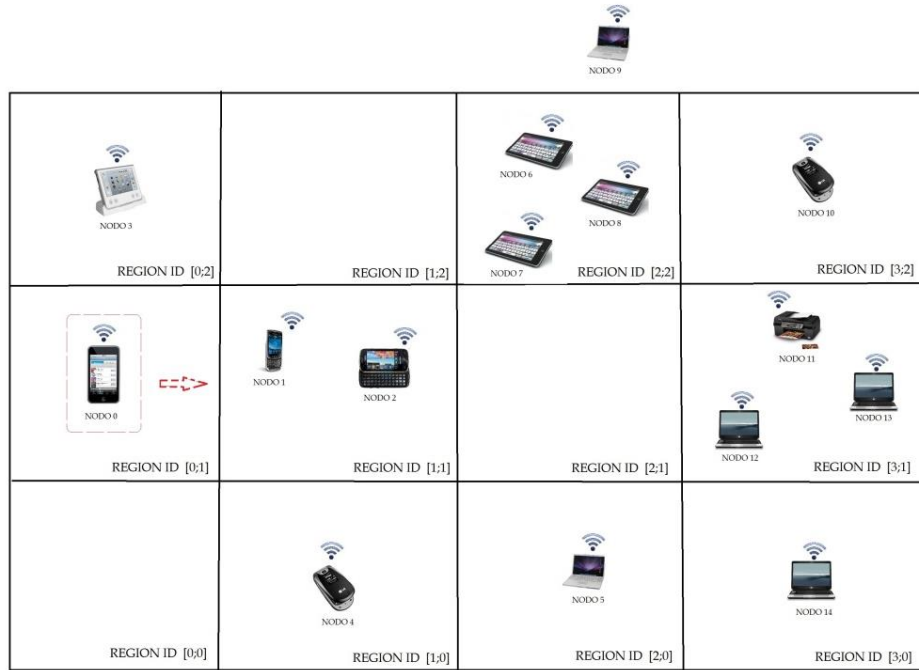


Fig. 67. Escenario de simulación de MANET

Parámetros	Valores
Tiempo de simulación	0-150s
Numero de nodos	15
Tamaño del área de trabajo	700x500 m
Tamaño de la rejilla	200x150 m
Propagación	TwoRayGround
Tipo de Interfaz de Cola	DropTail/PriQueue
Modelo de Antena	OmniAntenna
Potencia de Transmisión	0.28183815
Frecuencia	914 MHz
Ganancia de Antena de Transmisión	1
Ganancia de Antena de Recepción	1
Perdidas del sistema	1
Altura de la Antena de Transmisión	1.5
Altura de la Antena de Recepción	1.5
Umbral de Potencia de Recepción (RXThresh)	7.6911e-8 (50m)
Umbral de Potencia de Transmisión (CSThresh)	1.9228 e-8 (100m)

Tabla 19. Valores de la simulación del canal inalámbrico IEEE 802.11 para el escenario de simulación MANET

Parámetros	Valores
Numero de Filas	3
Numero de Columnas	4
Tamaño del paquete	1000
Coordenada X de Inicio de la rejilla	0
Coordenada Y de Inicio de la rejilla	0
Tamaño X de la rejilla	200
Tamaño Y de la rejilla	150

Tabla 20. Valores del Nodo Virtual en el escenario de simulación MANET

El script de simulación está formado por los agentes descritos en la sección 3.3. El agente Master contiene las características presentadas en la tabla Tabla 21, las cuales son las mismas para la simulación VANET, esto nos servirá para realizar una comparación entre los dos tipos de redes ad-hoc.

Parámetros	Valores
Intervalo Estático	1 s
Intervalo entre Heartbeat	0.5 s
Intervalos de Solicitud de liderazgo	1 s
Tiempo de Espera de Mensaje del líder	0.5 s

Tabla 21. Valores de la Simulación del Agente Master para el escenario de simulación MANET

- **RESULTADOS DE LA SIMULACION.**

Esta sección se detalla los resultados de la simulación del escenario MANET. Se seleccionó tres posibles gráficas para describir los resultados: la primera permite evaluar los cambios de estado de un nodo al desplazarse por la rejilla, la segunda nos brinda información únicamente de todo lo que sucede en una región en particular, y la tercera nos permite comparar entre algunos nodos las transiciones de estados sucedidas a lo largo del tiempo de simulación.

En la Fig. 68, en la cual se detalla todos los posibles estados que el nodo 0 (nodo preseleccionado para el análisis) toma a lo largo de la simulación. Al observar la gráfica y los anexos [4.2], se determina que el nodo 0, se ubica desde un inicio de la simulación en la región [1;0], en la cual al no haber ningún otro nodo presente, cambia de un estado *Desconocido* a *Solicitante* de acuerdo al algoritmo descrito en la sección 3.3.2 del capítulo 3. Luego de un tiempo de espera, asume levemente el liderazgo de la región por un periodo de [2,00s; 5,00s]. Luego de los 5 segundos se le pide al nodo que se traslade a la región vecina [1; 1] con lo cual este se configura automáticamente como *No Líder* para abandonar la región. Este nodo se mantiene dentro de la región [1; 0] por un tiempo de 6 segundos adicionales [5,00s; 11,00s] hasta que abandona la región actual.

El nodo 0 al ingresar a la región [1; 1] se mantiene en su interior por un periodo de tiempo de [11,50s; 34,16s], en esta región se aprecia que el nodo 0 tiene una clara tendencia a actuar como *No Líder*. Esto se justifica ya que en la región [1; 1], se encuentran dos nodos con sus estados definidos como *Líder* y *No Líder*. Al existir un *Líder* reconocido, el nodo 0 tiene que asumir un estado de *No Líder* hasta que el *Líder* decida renunciar al liderazgo.

En la región [2; 1], el nodo 0 presenta un estado de *Líder*, puesto que el nodo está ingresando a una región sin ningún nodo. En un inicio, al ingresar a la región se establece como *Desconocido* y luego de un corto periodo de tiempo cambia a *Solicitante*, emite mensajes para solicitar el liderazgo (*LeaderRequest*) y al no obtener respuesta se establece como *Líder*. Al abandonar la región lo hace en estado de *No Líder* permitiendo que otro nodo que ingrese pueda asumir el liderazgo de la región.

En la región [3; 1], por su parte, el nodo 0 presenta tres cambios de estado (el movimiento para el nodo 0 en la simulación al entrar a esta región se configuró de tal manera que el nodo presente estos estados): i) El nodo 0 ingresa a una región con un

Líder definido con anterioridad, por lo tanto asumirá un papel de nodo *No Líder*. ii) Al continuar con la simulación, intencionalmente se hace que el líder actual abandone la región cambiando su estado para verificar que el nodo 0, en ese momento, al no recibir mensajes *Heartbeat* para confirmación del liderazgo, cambia a un estado *Inestable* y luego a *Desconocido* para solicitar el liderazgo en la región.

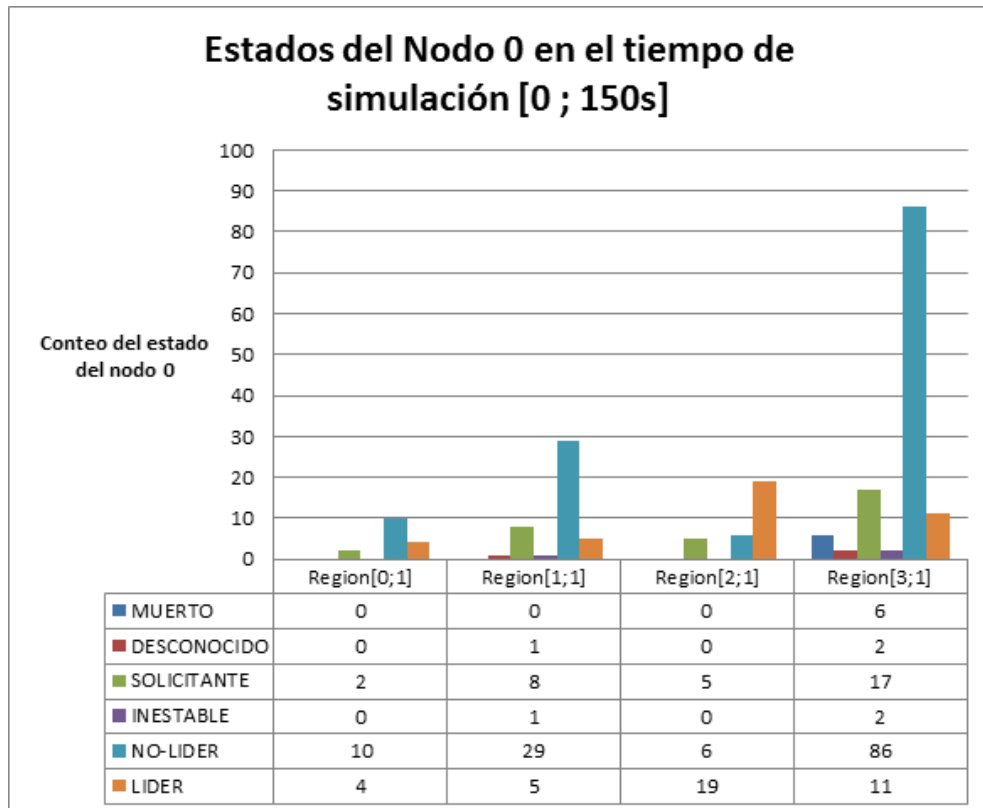


Fig. 68. Estado del nodo 0 en el tiempo de simulación [0s; 50s] en el escenario de simulación MANET

En esta etapa se produce un leve duplicado de liderazgo dado que existen varios nodos en la región [3; 1]; sin embargo, dado que los nodos emiten mensajes *Heartbeat* de forma aleatoria, esto permite que el nodo que primero envió su mensajes de confirmación de líder obtendrá dicha estado y los demás nodos se establecerán como *No Líderes*, tal como le sucede al nodo 0. iii) Por último, se optó por provocar que el nodo cambie su estado ha *Muerto* abandonando temporalmente la rejilla y luego volviendo a ingresar a la región de la cual partió para reiniciar el

algoritmo para selección de *Líder* propuesto en la sección 3.3.2 del capítulo 3, para finalmente configurarse nuevamente como *No Líder* dado que ya existe un *Líder* definido en la región.

Un punto importante a analizar son los cambios de estados de todos los nodos que se encuentran en una región en particular. En nuestro experimento se ha seleccionado la región [3; 1]. En la *Fig. 69* se observa que los nodos 11, 12, y 13 se disputaron y mantuvieron el liderazgo en la región a lo largo de la simulación. Sin embargo, al momento de que un nodo *Líder* abandona la región se produce el mismo efecto de duplicado de liderazgo analizado en el escenario anterior. Los nodos 0, 7, 9, 12 y 14 por su parte mantuvieron un predominio en el estado del no liderazgo a lo largo de toda la simulación, ocasionado como producto de un desplazamiento momentáneo hacia la región [3; 1], donde ya existe un *Líder* o debido al no permanecer por un tiempo considerable dentro de la región.

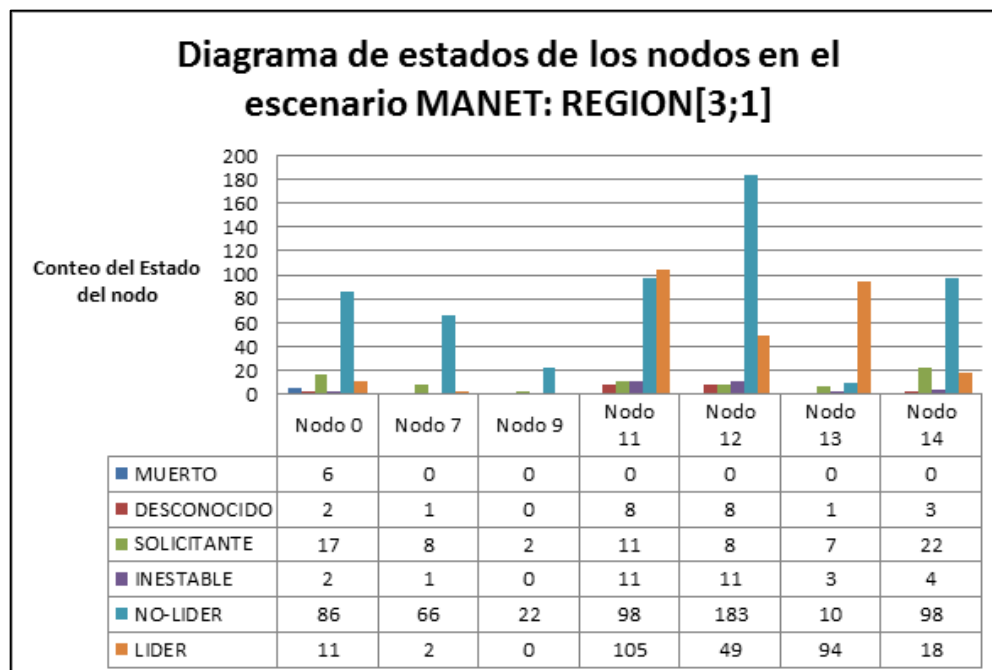


Fig. 69. Diagrama de estados del nodo en el escenario MANET: Región [3; 1]

En la Tabla 22, se presentan todos los posibles estados que los nodos 0,7, 9, 11, 12 y 13 toman a lo largo de todo el periodo de simulación. En la tabla se detalla el tiempo de duración del estado que toma el nodo y la región en la cual se encuentra en ese instante. En la Fig. 70, se realizó un análisis de los diferentes nodos que se encuentran asociados a una región particular [3; 1] sea por ingresar, abandonar o mantenerse en la región en algún instante dado, los resultados obtenidos son los siguientes:

NODO	MUERTO	DESCONOCIDO	SOLICITANTE	LIDER	NO-LIDER	INESTABLE	REGIÓN
0			[0,00;0,5]	[2,00;5,00]	[5;11]		[0;1]
		[22,50;23,00]	[12,50;13,08]		[14,00;21,50]	[21,00;21,50]	[1;1]
			[24,50;24,82]	[24,82;26,74]	[27,5;34,16]		[2;1]
			[35,00;35,4]	[35,40;45,5]	[45,50;48,50]		[3;1]
		[78,50;80,00]	[50,00;52,24]		[53,00;77,00]	[77,00;78,5]	[3;1]
			[80,00;80,08]	[80,08;82,16]	[82,24;85,24]		[3;1]
		[86,00;119,00]			[120,5;123,5]		[3;1]
7			[125,00;125,74]		[126,5;149,83]		[2;2]
			[0,00;0,81]	[2,00;30,50]	[30,50;33,50]		[2;1]
			[34,50;35,00]		[35,00;38,83]		[3;1]
		[78,50;78,60]	[39,50;42,50]	[42,5;43,08]	[44,00;75,58]	[76,80;77,00]	[2;1]
			[86,00;86,08]		[86,10;90,50]		[2;2]
9			[92,00;95,00]	[95,00;149,5]			[1000;1000]
	[0,5;29,00]				[30,5;35,00]		[3;2]
			[36,50;36,60]		[36,60;44,00]		[3;1]
			[45,50;45,60]		[45,50;55,16]		[3;0]
11			[56,00;56,10]	[56,10;149,5]			[3;1]
			[0,00;0,24]	[0,25;2,66]		[3,50;5,66]	[3;1]
					[6,50;8,00]	[9,50;9,60]	[3;1]
		[11,00;11,66]			[12,50;12,58]	[14,00;14,10]	[3;1]
		[15,50;15,60]	[17,00;17,66]		[17,74;27,50]	[29,00;29,10]	[3;1]
		[29,10;29,58]			[30,50;77,00]	[77,00;78,50]	[3;1]
12			[78,50;80,00]	[80,00;80,08]	[80,08;82,32]	[83,00;86,08]	[87,50;87,60]
			[89,00;89,10]	[90,50;90,66]	[90,66;149,5]		
			[0,00;0,16]	[0,16;17,66]	[18,50;27,50]	[27,50;29,00]	[3;1]
		[29,00;29,58]			[30,50;35,00]	[35,00;36,50]	[3;1]
		[36,50;37,08]			[38,00;75,58]	[77,00;77,50]	[3;1]
13			[78,50;78,60]	[80,00;80,16]	[80,16;82,08]	[82,16;93,50]	[98,00;98,50]
			[96,50;98,60]			[99,50;103,33]	[3;1]
					[104;149,83]		[3;1]
			[0,00;1,08]		[1,16;8,00]	[9,50;9,60]	[3;1]
13		[11,00;11,10]	[12,50;14,00]	[15,50;75,50]	[75,50;75,58]		[3;1]
			[77,00;77,66]		[78,50;93,50]	[95,00;95,10]	[2;0]
		[96,50;96,60]	[98,00;99,50]	[101,00;104]		[105,5;105,6]	[2;0]
		[107,0;107,1]	[108,5;108,58]	[108,58;149,5]			[2;0]

Tabla 22. Estado de los nodos 0, 7, 9, 11, 12 y 13 en el tiempo de simulación [0s; 150s] en el escenario de simulación MANET

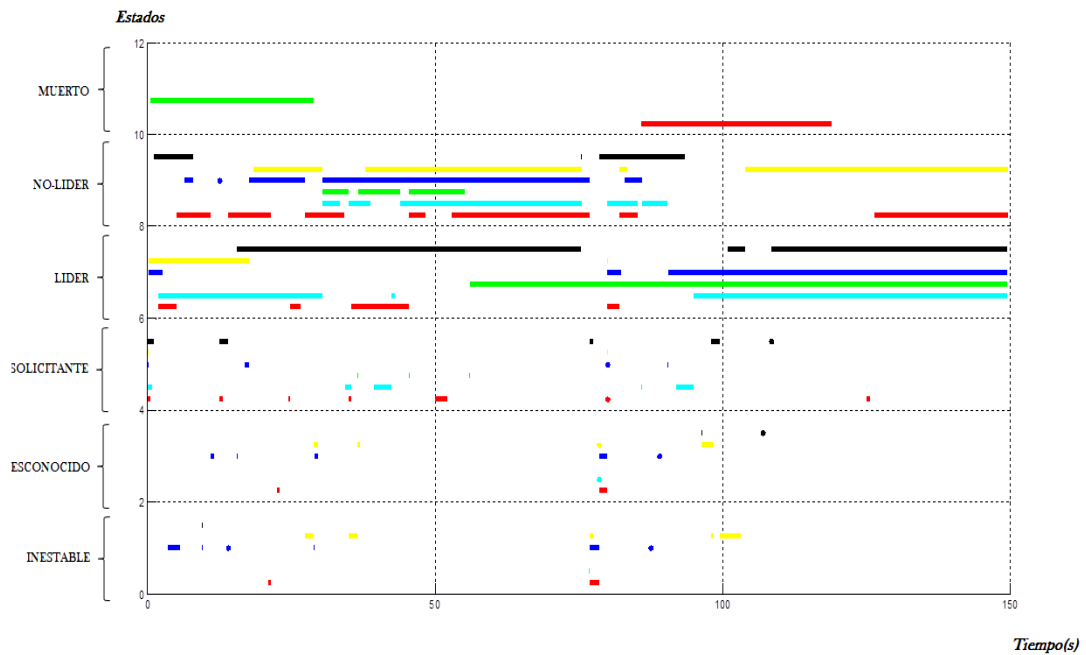


Fig. 70. Estado de los nodos 0, 7, 9, 11, 12 y 13 en el tiempo de simulación [0s; 150s] en el escenario de simulación VANET

Desde el inicio de la simulación se encuentran 3 nodos en la región [3; 1] dichos nodos son el nodo 11, nodo 12 y el nodo 13. Sus estados iniciales cuando parte la simulación son: nodo 12 asume el liderazgo y los nodos 11 y 13 se configuran como *No Líderes* en la región mencionada.

El primer nodo que asume el liderazgo en la región [3; 1] es el nodo 12 y lo hace por un periodo de tiempo de [0,16 s; 17,66 s] que es el tiempo antes del cual el nodo comienza a desplazarse libremente por la región.

Existe una leve pérdida de mensajes *Heartbeat* emitidos por el nodo 12 que actúa como *Líder* de la región, si los nodos *No Líderes* no reciben estos mensajes para reconocer al *Líder* cambian sus estados para solicitar el liderazgo de la región, estos cambios de estados se observan al analizar los datos del nodo 13, que pasa de un

estado *Inestable* a *Desconocido* y luego a *Solicitante* para finalmente asumir el liderazgo de la región [3; 1] por un periodo de tiempo de [15,50; 75,50].

En el segundo 75 de la simulación el nodo 13 se configura para abandonar la región [3; 1] con el fin de que este nodo renuncié a su liderazgo actual y entre los dos únicos nodos en la región el nodo 11 y el nodo 12 se disputen el liderazgo. Este evento se refleja en los estados de estos nodos ya que ambos se encuentran en calidad de *Solicitantes* aproximadamente en el segundo 80 de la simulación y luego de intercambiar mensajes y estabilizar sus estados, el nodo 11 (nodo establecido sin movimiento para efectos de simulación) se convierte en el nuevo Líder de la región [3; 1] desde el segundo 90 hasta que la simulación culmina luego de los 150 segundos establecidos para el experimento.

En la *Fig. 70*, se presenta de manera gráfica cada uno de los estados de los nodos en el tiempo de simulación de 0s a 150s. En la gráfica, el nodo 0 corresponde al color rojo, el nodo 7 el color cian, el nodo 9 al color verde, el nodo 11 al azul, el nodo 12 amarillo y el nodo 13 al color negro.

c) Escenario de simulación VANET

El escenario de simulación VANET está compuesto por dos partes. La primera es la generación del tránsito vehicular a través de SUMO, y la segunda es la entrega del movimiento de los nodos (vehículos) hacia el simulador de redes ns-2.

El escenario de tránsito vehicular, está compuesto por la red vial, de la Av. Huayna-Cápac de la ciudad de Cuenca, véase *Fig. 71*. En base a la sección 4.1, la red vial de JOSM se exportó a SUMO. Además, las herramientas DUAROUTER y

randomTrips.py, nos sirvieron para la generación de la demanda de tránsito vehicular, véase Fig. 72.

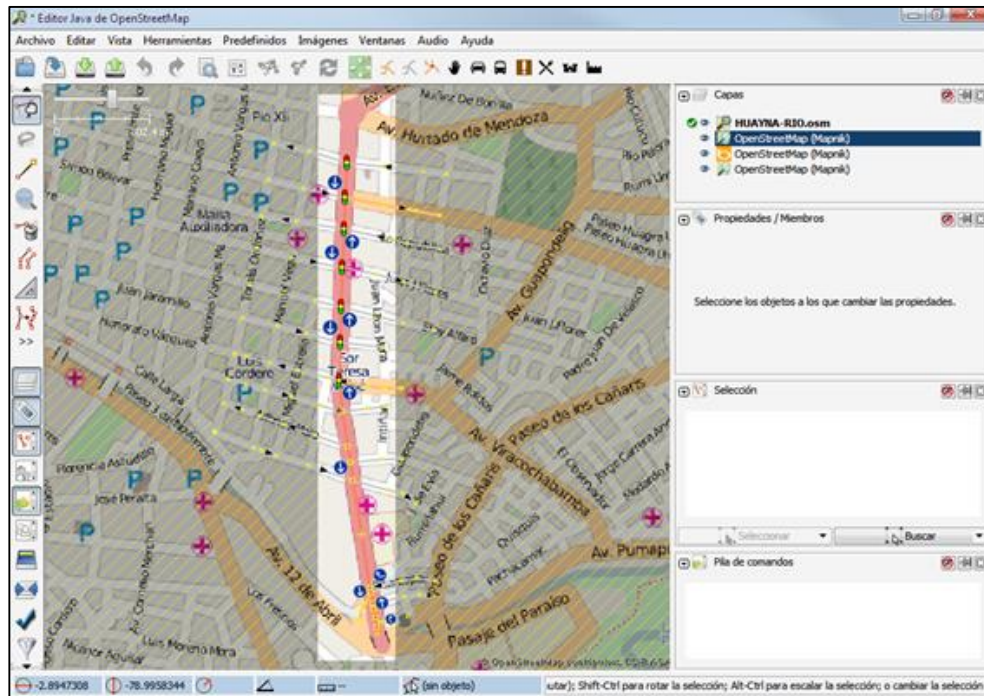


Fig. 71. Vista del mapa de la Av. Huayna-Cápac en JOSM

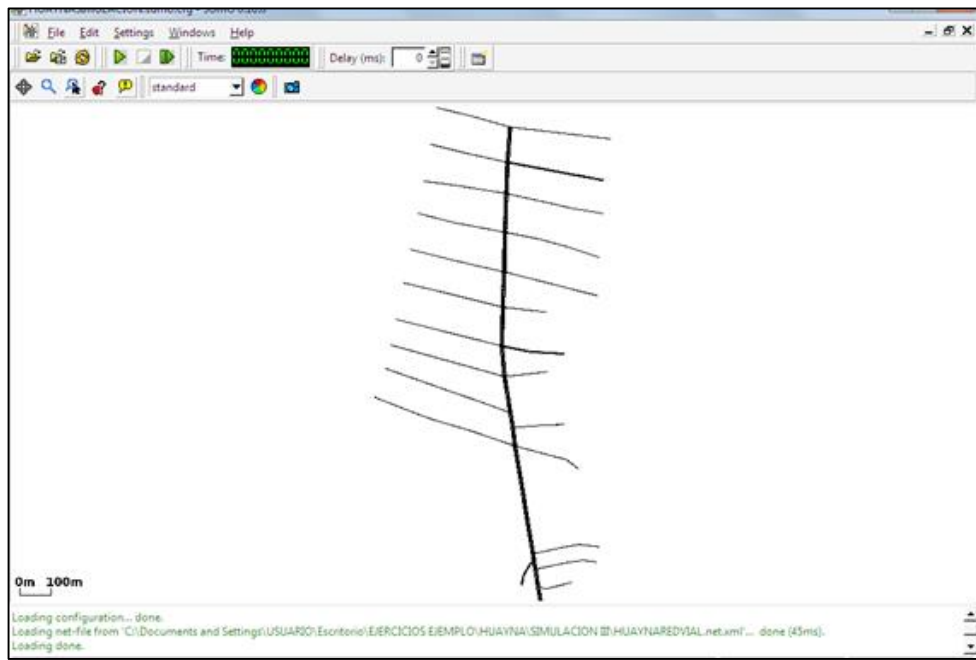
El escenario de tránsito está compuesto por 25 vehículos que transitan en la red vial de manera aleatoria, el movimiento de los vehículos respeta el sentido de las vías. La simulación se realiza en un rango de 0s a 100s. Además, en base a la sección 3.4, se configuró a la herramienta TraceExporter, con las siguientes características: el tiempo de simulación se estableció en [0-100s] para el inicio y fin de la simulación, respectivamente, la penetración en 1 para que no haya el filtrado de vehículos.

Así, TraceExporter nos entregó los archivos config.tcl, mobility.tcl y activity.tcl, Estos archivos se integran al fichero simulación de redes de ns-2. Estos tres archivos nos entregan el movimiento de los nodos físicos (vehículos) sobre el escenario de simulación de ns-2.

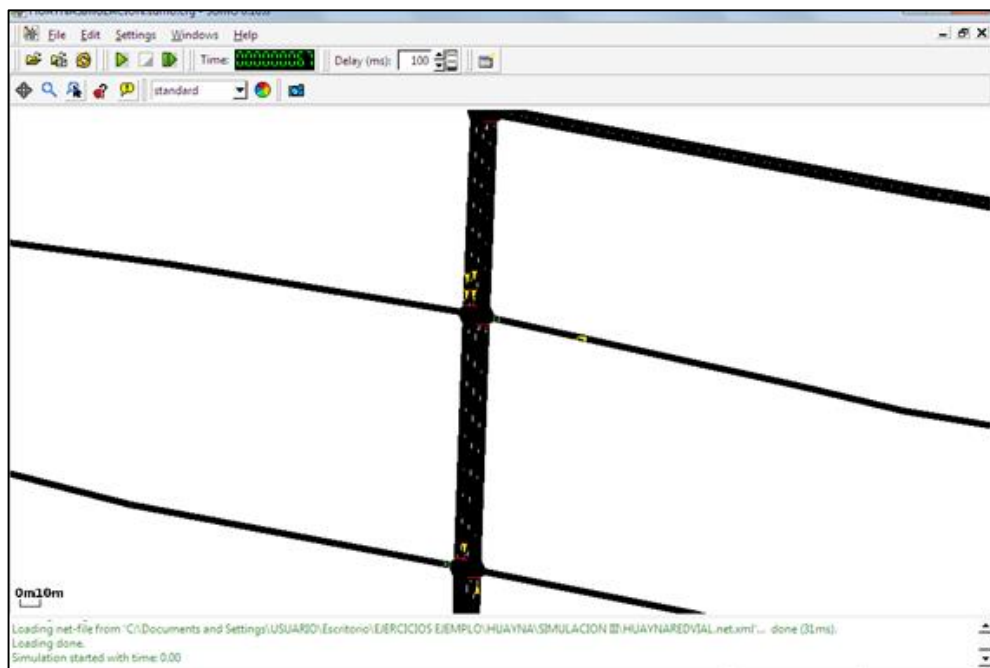
El escenario de simulación exportado de SUMO a ns-2 tiene un tamaño de la topografía en 736 x1539m. La simulación se estable con un tiempo de 100s y 25 nodos. Los nodos se equipan con una interfaz de red inalámbrica de IEEE 802.11. Las características del canal inalámbrico se establecen en la Tabla 23:

Parámetros	Valores
Tiempo de simulación	0-100s
Numero de nodos	25
Tamaño de la topología	736x1539m
Propagación	TwoRayGround
Tipo de Interfaz de Cola	DropTail/PriQueue
Modelo de Antena	OmniAntenna
Potencia de Transmisión	0.28183815
Frecuencia	914 MHz
Ganancia de Antena de Transmisión	1
Ganancia de Antena de Recepción	1
Perdidas del sistema	1
Altura de la Antena de Transmisión	1.5
Altura de la Antena de Recepción	1.5
Umbral de Potencia de Recepción (RXThresh)	8.5457e-9 (150m)
Umbral de Potencia de Transmisión (CSThresh)	4.8070e-9 (200m)

Tabla 23. Valores de la Simulación del canal Inalámbrico IEEE 802.11 en el escenario de simulación VANET



(a)



(b)

Fig. 72. Generación de la demanda de tránsito vehicular sobre la Av. Huayna-Cápac;
a) La Red Vial de la Av. Huayna-Cápac, b) Captura del movimiento de los vehículos sobre la red vial

Como nuestro ambiente de simulación es una rejilla, compuesta por varias regiones rectangulares, véase Tabla 24, se delimito el rango de transmisión para que los nodos no procesen todos los paquetes provenientes de todos los nodos en el escenario de simulación. Esto previene la sobrecarga innecesaria de las colas de paquetes de los nodos, y el descarte de paquetes receptados, véase *Fig. 73*. En base al tamaño de la región se determinó con la Ec. 3 los valores de CSTthresh/ RXTthresh en 200m y 150m.

Parámetros	Valores
Numero de Filas	6
Numero de Columnas	1
Tamaño del paquete	1000
Coordenada X de Inicio de la rejilla	390
Coordenada Y de Inicio de la rejilla	750
Tamaño X de la rejilla	30
Tamaño Y de la rejilla	630

Tabla 24. Valores del Nodo Virtual en el escenario de simulación VANET

De la *Fig. 72*, se observa que la Av. Huayna-Cápac no inicia en las coordenadas de origen, esta tiene un desplazamiento vertical y horizontal, por lo que los nodos físicos restringen su movimiento a las coordenadas contenidas por la Avenida. Así, se ubicó a la rejilla en las coordenadas de inicio [390; 750], que corresponde a la intersección de la Avenida Huayna- Cápac y Avenida Viracochabamba. El final de la rejilla se ubicó en [391; 1380] que corresponde a la intersección de la Avenida Huayna- Cápac y Presidente Vicente Rocafuerte. El ancho de la rejilla es el ancho de la Avenida Huayna- Cápac.

El script de simulación está formado por los agentes descritos en la sección 3.3. El agente Master contiene las características presentadas en la Tabla 25, las cuales son las mismas para la simulación MANET, esto nos servirá para realizar una comparación entre los dos tipos de redes Ad-hoc.

Parámetros	Valores
Intervalo Estático	1 s
Intervalo entre Heartbeat	0.5 s
Intervalos de Solicitud de liderazgo	1 s
Tiempo de Espera de Mensaje del líder	0.5 s

Tabla 25. Valores de la Simulación del Agente Master en el escenario de simulación VANET

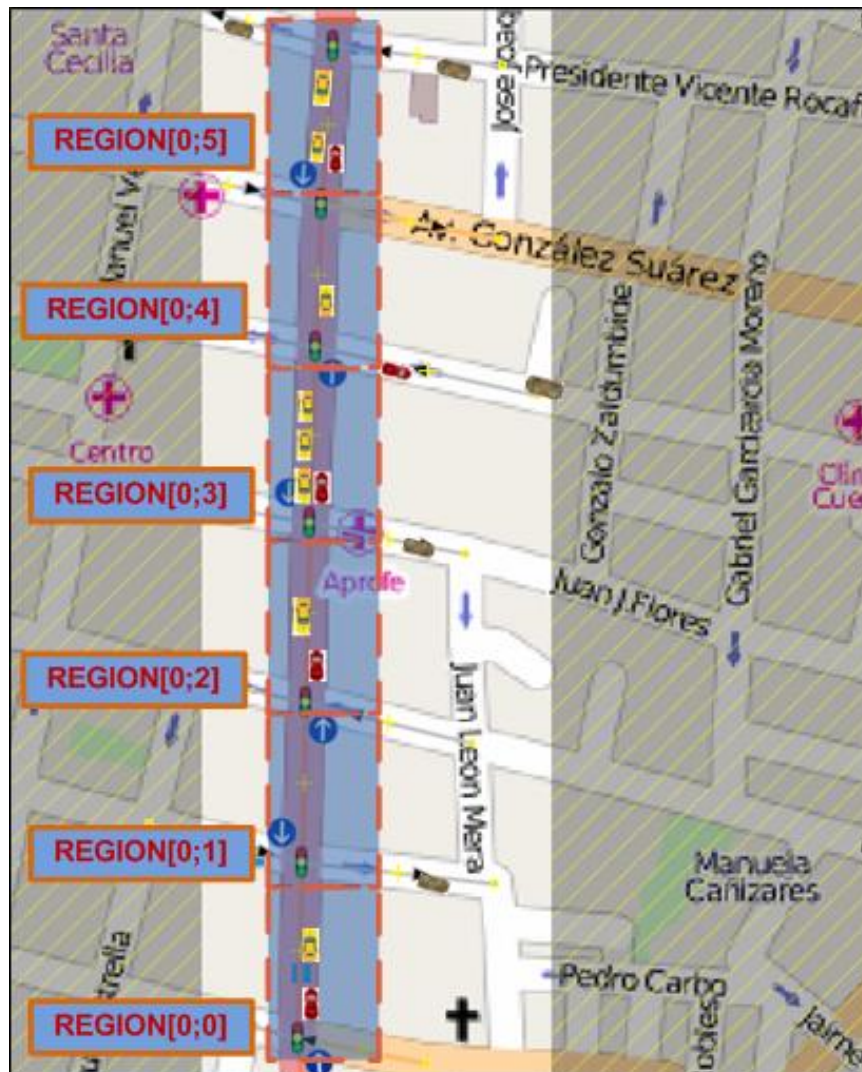


Fig. 73. Escenario de simulación VANET

- **Resultados de la simulación.**

Esta sección muestra los resultados de la simulación del escenario VANET. En este escenario todos los nodos se intercambian información de acuerdo a la región a la cual pertenecen. En la Fig. 74, se presenta el estado de los nodos que actuaron en la región [0,3], se observa que los nodos 1, 2, 9, 15, 17, mantuvieron un liderazgo en la región. Los nodos 3, 4, 10, 12 y 22 mantuvieron un predominio en el estado del *No Líder*.

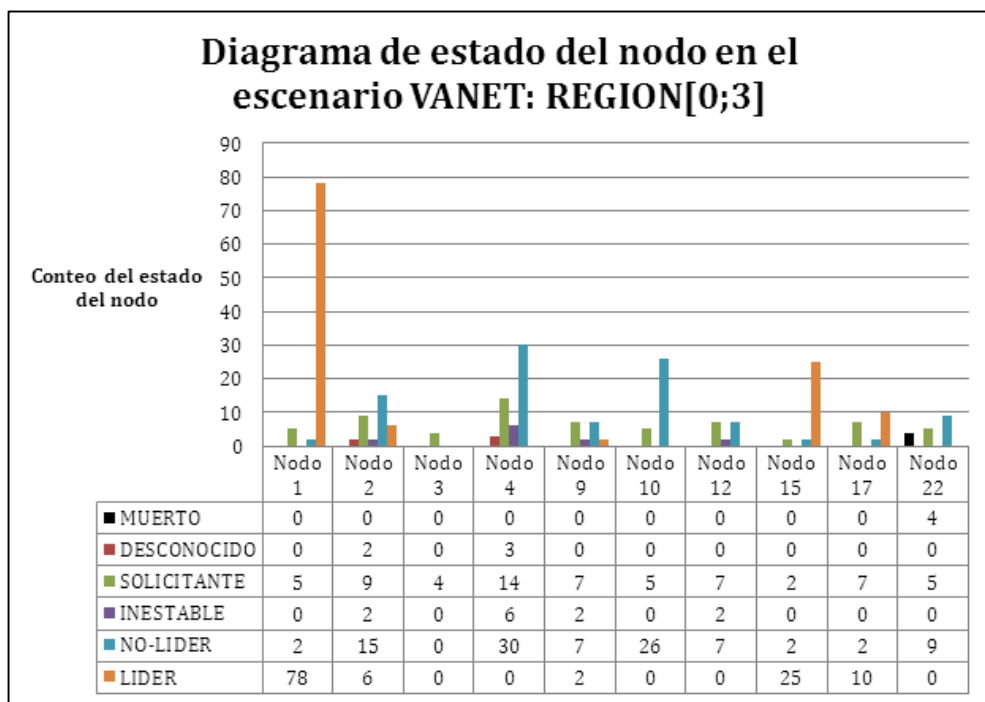


Fig. 74. Diagrama de estados del nodo en el escenario VANET: Región [0; 3]

En la Tabla 25, se presenta el diagrama de estados de los nodos: 1, 2, 9, 15 y 17. En esta tabla se muestra el tiempo de duración del estado del nodo y la región a la que pertenecieron. Así, se determina los tiempos en que los nodos mencionados se establecen como *Líder* en la región de análisis [0; 3]. El primer nodo que asume el liderazgo en la región [0; 3] es el nodo 15, sobre un rango de tiempo de [2,00s; 17,00s].

En el rango de tiempo [17,00s; 41,00s], la región no contiene nodos, por lo cual se establece un tiempo de inactividad. En 41.00s los nodos 1 y 2 ingresan en la región [0; 3], al ingresar a una nueva región los nodos 1 y 2 tratan de determinar su papel en la región, para lo cual envían un mensaje tipo Master con la bandera *LeaderRequest*. Al no haber un mensaje de respuesta de un nodo *Líder*, los nodos 1 y 2 establecen su liderazgo en los tiempos de 41.08s y 41.16s respectivamente. Al establecer el liderazgo de los nodos 1 y 2, estos comienzan a transmitir mensajes tipo Master con las banderas *Heartbeat* y *comienza_ldr*, ver sección 2.3. El nodo 1 se estableció como *Líder* en un tiempo menor al nodo 2. Con lo que, en 43.08s el nodo 2 recibe un paquete Master, con las banderas *Heartbeat* y *comienza_ldr*, y cambia su estado a *No Líder*. En el rango de tiempo [45,50; 47,00], el nodo 2 no recibe un mensaje Master con la bandera *Heartbeat* del nodo 1, *Líder* de la región [0; 3], con lo cual, el nodo 2 cambia de estado a *Inestable*, por el conteo de dos banderas *Heartbeat* perdidos, al contar un tercer *Heartbeat* perdido, el nodo 2 cambia de estado a *Desconocido*, y posteriormente a *Solicitante*. En el tiempo de 48.66s, el nodo 2 recibe una bandera *Heartbeat* del nodo 1 *Líder* de la región, con lo cual establece su estado en *No Líder*. La transmisión y recepción de los paquetes de los nodos se presenta en el anexo [4.3].

El nodo 1 establece su liderazgo sobre la región [0; 3] en un rango de [41.08s; 84.50s]. En el tiempo de 93.66 el nodo 17 asume el liderazgo de la región [0; 3]. Además el nodo 17 recibe un paquete tipo Master con la bandera *LeaderRequest* del nodo 9, y emite un paquete Master con la bandera *LeaderReply* para el nodo 9, (este paquete no llega a ser recibido por el nodo 9, debido a factores como desbordamiento de la cola o por choque de paquetes debido al canal inalámbrico). En el tiempo de 95,08 el nodo 9 establece un momentáneo liderazgo, hasta que en el tiempo de 95.58 recibe un mensaje tipo Master con la bandera *Heartbeat* del nodo 17, que hace prevalecer su liderazgo.

En la *Fig. 75*, se presenta de manera gráfica cada uno de los estados de los nodos en el tiempo de simulación de 0s a 100s. En la gráfica, el nodo 1 corresponde al color

rojo, el nodo 2 el color verde, el nodo 9 al color azul, el nodo 15 al negro y el nodo 17 al color amarillo.

NODO	MUERTO	DESCONOCIDO	SOLICITANTE	LIDER	NO-LIDER	INESTABLE	REGIÓN
1			[0,00;0,24]	[0,24;1,16]	[2,00;39,50]	[39,50;39,50]	[0;2]
			[41,00;41,08]	[41,08;84,50]	[84,50;84,50]		[0;3]
			[86,00;86,58]		[86,66;92,00]		[0;4]
			[93,50;93,58]	[93,58;100,00]			[0;5]
2	[0,50;39,50]						[1000;1000]
			[41,00;41,16]	[41,16;43,08]	[44,00;45,50]	[45,5;47,00]	[0;3]
		[47,00;48,50]	[48,50;49,08]		[50,00;55,08]		
			[56,00;59,00]	[59,00;88,00]			[0;2]
			[89,00;92,00]	[92,00;95,50]			[0;1]
		[96,5;97,08]		[98,00;100,08]		[0;0]	
9	[0,5;47,00]						[1000;1000]
			[48,50;49,08]		[50,00;83,00]		[0;5]
			[84,50;84,66]	[84,66;86,58]	[86,66;92]		[0;4]
15			[93,50;95,08]	[95,08;95,58]	[96,50;99,50]	[99,50;99,66]	[0;3]
			[0,00;0,50]	[2,00;17,00]	[17,00;17,00]		[0;3]
			[18,50;19,08]		[20,00;25,08]		[0;2]
			[26,00;29,00]	[29,00;32,50]	[33,50;33,50]		[0;1]
17			[35,00;38,00]	[38,00;100,00]			[0;0]
			[0,00;0,50]	[2,00;80,00]	[80,00;80,00]		[0;5]
			[81,50;84,50]	[84,50;90,50]	[90,50;90,50]		[0;4]
			[92,00;93,66]	[93,66-98,00]	[98,00;98,00]		[0;3]
		[99,50-99,50]				[0;2]	

Tabla 26. Estado de los nodos 1, 2, 9, 15, 17 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET

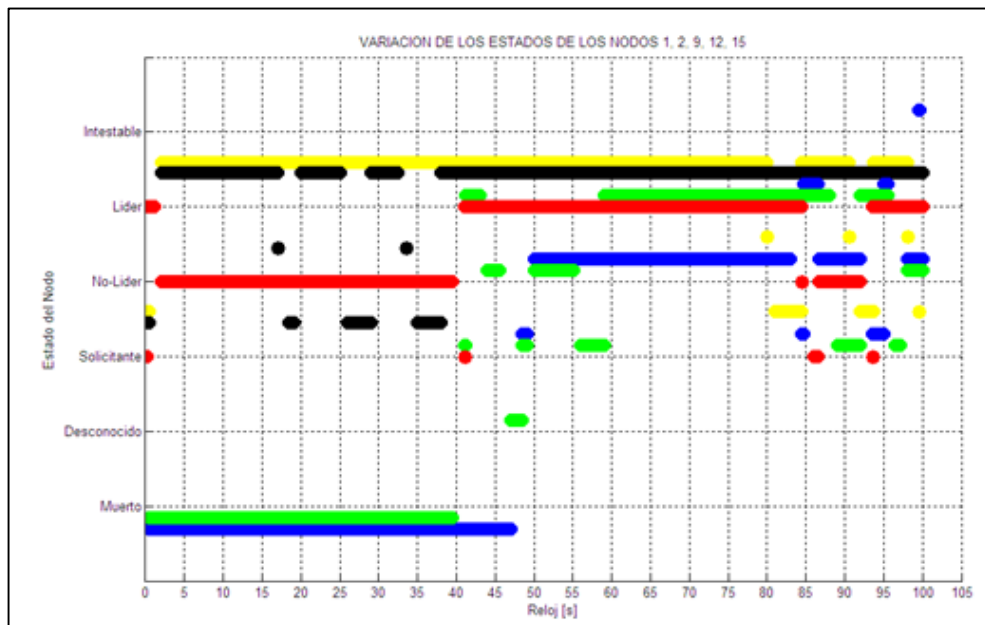


Fig. 75. Estado de los nodos 1, 2, 9, 15, 17 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET

En la Fig. 76, se presenta el diagrama de barras de los estados del nodo 15, a lo largo de la simulación. Al observar la gráfica, la Tabla 26 y los anexos [4.3], se determina que el nodo 15, asume un liderazgo de la región [0; 3] en el rango de simulación de [2,00s; 17,00s]. En el tiempo 17.00s, el nodo 15 cambia de región (se establece como nodo *No Líder*) a la región [0; 2]. Al ingresar en la región [0; 2], el nodo 15 se establece como nodo *No Líder*, por la existencia de un nodo *Líder* (nodo 13). En el tiempo 26,00s el nodo 15 ingresa en la región [0; 1], al ingresar en una nueva región el nodos 15 determinar su papel en la región, para lo cual envían un mensaje tipo Master con la bandera *LeaderRequest*. Al no haber un mensaje de respuesta de un nodo *Líder*, el nodo 15 asume un momentáneo liderazgo en el tiempo de 29,00s y comienza a transmitir mensajes tipo Master con las banderas *Heartbeat* y *comienza_ldr* (ver sección 2.3). El nodo 15 constituye su liderazgo sobre la región [0; 1] en un rango de [29.00s; 32.50s]. En el tiempo 32.50s, el nodo 15 cambia de región (cambia a nodo *No Líder*) a la región [0; 0]. En la región [0; 0] el nodo permanece como líder hasta el fin de la simulación.

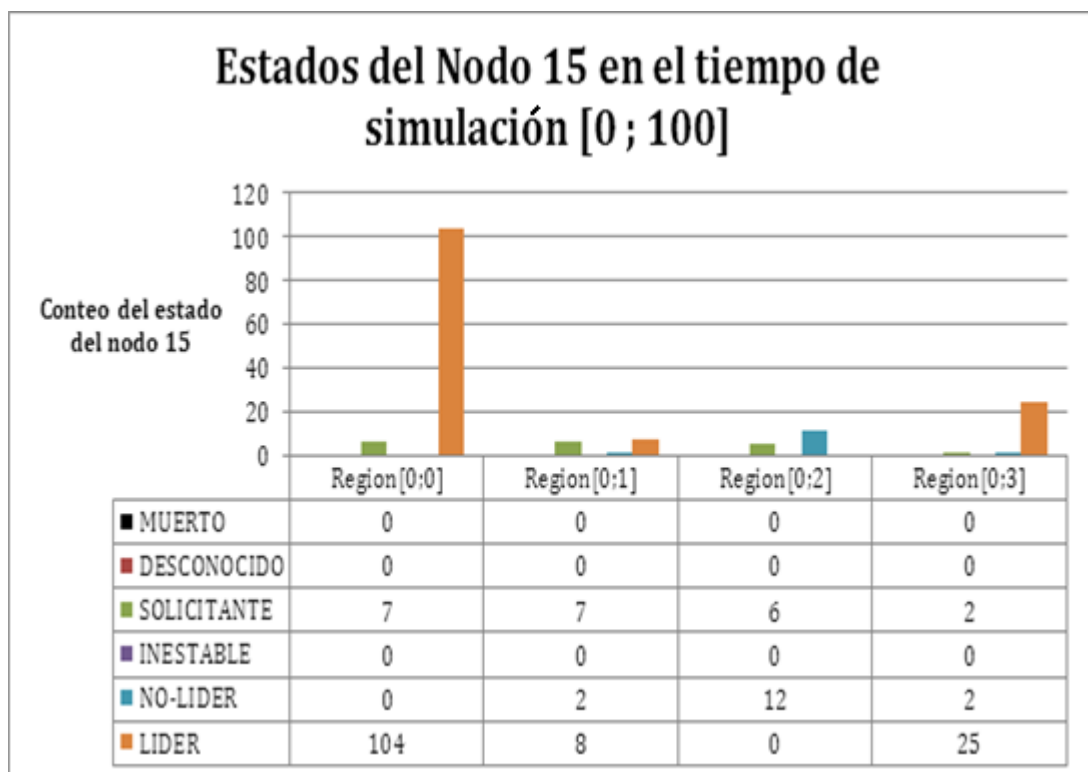


Fig. 76. Estado del nodo 15 en el tiempo de simulación [0s; 100s] en el escenario de simulación VANET

d) Escenario de simulación de Agentes Master, Constancy y Members

Este escenario de simulación está compuesto por nodos sin movimiento, véase *Fig. 77*. El escenario tiene una topografía de 400x400m. Además, está compuesto por 9 nodos en un tiempo de simulación de 0s a 50s. Los nodos se equipan con una interfaz de red inalámbrica IEEE 802.11. Las características del canal inalámbrico se establecen con los valores de la Tabla 27.

Parámetros	Valores
Tiempo de simulación	0-120s
Numero de nodos	10s
Tamaño de la topología	700x700m
Propagación	TwoRayGround
Tipo de Interfaz de Cola	DropTail/PriQueue
Modelo de Antena	OmniAntenna
Potencia de Transmisión	0.28183815
Frecuencia	914 MHz
Ganancia de Antena de Transmisión	1
Ganancia de Antena de Recepción	1
Perdidas del sistema	1
Altura de la Antena de Transmisión	1.5
Altura de la Antena de Recepción	1.5
Umbral de Potencia de Recepción (RXThresh)	7.6911e-8 (50m)
Umbral de Potencia de Transmisión (CSThresh)	1.9228e-8 (100m)

Tabla 27. Valores de la Simulación del canal Inalámbrico IEEE 802.11 en el escenario de simulación de los tres agentes Master, Constancy y Members



Fig. 77. Escenario de simulación de los tres agentes Master, Constancy y Members

El script de simulación está formado por los agentes descritos en la sección 3.3. El agente Master contiene las características presentadas en la Tabla 28. De la Fig. 77, se ubicó a la rejilla en las coordenadas de inicio [0; 0]. El tamaño de rejilla es de 200x200m con 4 columnas y 4 filas. Así, el tamaño de cada región cuadrada será de 50x50m.

Parámetros	Valores
Intervalo Estático	1 s
Intervalo entre Heartbeat	0.5 s
Intervalos de Solicitud de liderazgo	1 s
Tiempo de Espera de Mensaje del líder	0.5 s

Tabla 28. Valores de la Simulación del Agente Master en el escenario de simulación de los tres Agentes: Master, Constancy y Members.

Como nuestro ambiente de simulación es una rejilla, compuesta por varias regiones cuadradas, véase *Tabla 29*, se delimito el rango de transmisión para que los nodos no procesen todos los paquetes provenientes de todos los nodos en el escenario de simulación. Esto previene la sobrecarga innecesaria de las colas de paquetes de los nodos, y el descarte de paquetes receptados, véase Fig. 77. En base al tamaño de la región se determinó con la ecuación [3] los valores de CSTthresh/ RXThresh en 100m y 50m.

Parámetros	Valores
Numero de Filas	6
Numero de Columnas	1
Tamaño del paquete	1000
Coordenada X de Inicio de la rejilla	390
Coordenada Y de Inicio de la rejilla	750
Tamaño X de la rejilla	30
Tamaño Y de la rejilla	630

Tabla 29. Valores del Nodo Virtual en el escenario de simulación de los tres Agentes: Master, Constancy y Members.

- **Resultados de la simulación.**

Esta sección muestra los resultados de la simulación del escenario con los tres agentes. En este escenario todos los nodos se intercambian información de acuerdo a la región a la cual pertenecen.

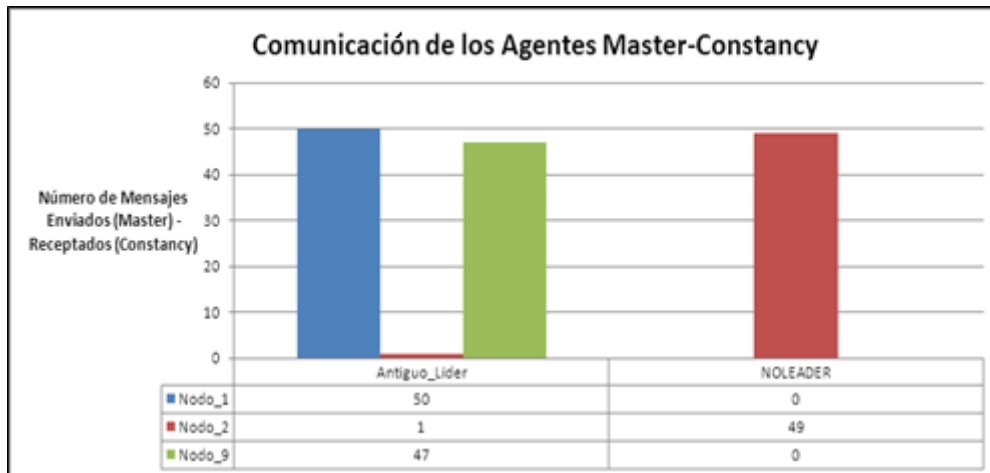


Fig. 78. Comunicación de los Agentes Master-Constancy

En la Fig. 78, se presenta a los nodos 1, 2 y 9. Los nodos 1 y 2 se localizan en la región $[0,0]$, y el nodo 9 en la región $[1,1]$. En la figura se observa el envío de la información de región y estado del nodo del Master al agente Constancy, para ello utiliza la cabecera ESTANDAR, (ver sección 3.3.1.). Así, el nodo 1 y 9, se encuentran en estado *Líder* a lo largo de la simulación en sus respectivas regiones, el nodo 2 se encuentran en estado de *No Líder* en la región $[0, 0]$. En la Fig. 79, se presenta la comunicación de los Agentes Constancy y Members de los nodos 1, 2 y 9.

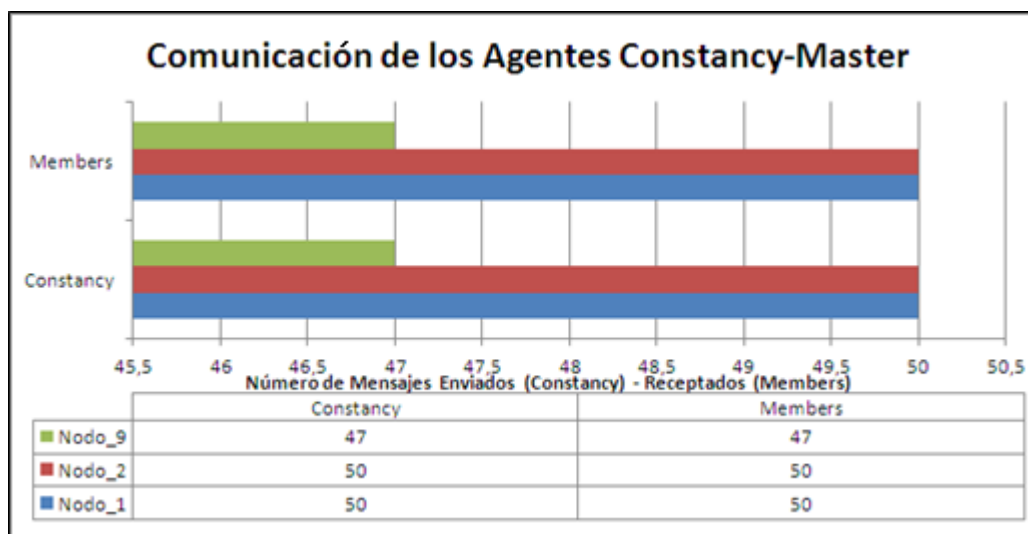


Fig. 79. Comunicación de los Agentes Constancy-Members

4.3. Análisis de los resultados.

En el experimento con nodos estáticos, se comprueba el correcto funcionamiento del algoritmo diseñado en [42] e implementado por el ambiente de simulación. Así por un rango de tiempo de $[0, 2s]$, el nodo virtual coloca a un nodo líder y lo mantiene hasta el final de la simulación.

Al desplazarse los nodos en una MANET a una velocidad inferior que en una VANET, al nodo le tomará más tiempo en desplazarse de un punto a otro por lo tanto su tiempo de pertenencia en una región es relativamente más elevado en función del tamaño de la región. Esto contribuye a que el intercambio de paquetes entre nodos de una misma región sea más constante, esto facilita que los nodos se establezcan como *Líderes* y *No Líderes* respectivamente. En caso de pérdidas de paquetes por factores de colas o del enlace inalámbrico, se tiene una respuesta favorable frente a una nueva elección de *Líder*, tal como se aprecia en la Tabla 22, el nodo *Líder* en la región [3,1] abandona la región aproximadamente en el segundo 80 de la simulación. En ese instante, existen tres nodos en la misma región y por un lapso de 2 segundos todos se establecen como *Líderes*, sin embargo, al emitir de forma aleatoria los mensajes para establecerse en calidad de *Líderes* y ser reconocidos como tales, como resultado se tiene un solo *Líder* en la región y los otros nodos actualizan sus estados como *No Líderes*. En caso de tener áreas más amplias por región y mantener la misma velocidad promedio de una persona al caminar (1 m/s) con un dispositivo móvil en sus manos, el nodo podría permanecer por más tiempo como *Líder* o establecerse como *No Líder* siempre y cuando entre todos los nodos en la región exista conectividad ya que si esto no fuera posible se presentaría un duplicado de liderazgo.

En el experimento con VANET, se realiza la importación de la red vial de la Av. Huayna-Cápac de la ciudad de Cuenca-Ecuador (véase Fig. 71), Así, nuestro escenario se compuso por seis filas y una columna, (véase Tabla 24). Además de la Fig. 72, se observa que la Av. Huayna-Cápac no inicia en las coordenadas de origen.

Así, los nodos físicos restringen su movimiento a las coordenadas contenidas por la Avenida, por lo que se ubicó a la rejilla en las coordenadas de inicio [390; 750]. Así, el algoritmo implementado, trato de ubicar siempre un nodo líder en una región. Al analizar las *Fig. 75*, *Fig. 76* y *Fig. 77*, se ubica breves rangos de tiempos de *Líderes* duplicados en la regiones, esto se debe por perdidas de paquetes (Enlace Inalámbrico), o por eliminación de paquetes por colas de recepción de paquetes llenas.

En el experimento con los tres agentes, se comprueba la comunicación de los tres agentes. Esto se realiza mediante una comunicación con loopback, permitiendo un intercambio de información entre los agentes Master, Constancy y Members.

5. CONCLUSIONES Y RECOMENDACIONES

En esta tesis se implementó un ambiente de simulación de nodos virtuales tanto para escenarios MANETS y VANETS mediante software libre. Nuestra propuesta de tesis se basó en las investigaciones [5], [8] y en el diseño propuesto en [42]. En esta sección se revisa las contribuciones, análisis y futuras investigaciones.

5.1. Conclusiones

Las redes de infraestructura se caracterizan por el empleo de un punto de acceso central que se encarga de direccionar la información entre los dispositivos terminales. Las redes ad-hoc transmiten información entre los dispositivos terminales de forma directa, sin la necesidad de un punto de acceso central. Así, las redes MANETs son una clase de red ad-hoc, cuyos nodos o dispositivos terminales se encuentran en constante movimiento. Por otra parte, las redes VANETs son una clase de red ad-hoc, que se destacan por aspectos como la autonomía y movilidad de sus nodos. La movilidad de sus nodos hace que la topología de red sea altamente variable, dado que los nodos pueden entrar o salir de una región. Por este motivo, el protocolo de comunicación establecido entre los miembros de la red debe cubrir eficientemente estas consideraciones con el fin de garantizar una óptima calidad del servicio ofrecido.

Los protocolos que se utilizan para controlar el encaminamiento de los paquetes a intercambiar dentro de una red VANET mantienen ciertas características propias de los protocolos desarrollados para MANET. La diferencia se da en el hecho de que deben soportar una topología escalable y variable. Entre los protocolos que

funcionan en las redes MANET y que se consideran apropiados para ser utilizados para la simulación con VANETs destacan los protocolos DSDV, AODV y TORA. No obstante, la alta movilidad de los nodos ha hecho que estos protocolos muestren deficiencias en su implementación en redes vehiculares, por lo que se requiere el estudio de nuevos protocolos que consideren las características propias de las VANETs.

5.2. Contribuciones.

Nuestro trabajo se ha enfocado en desarrollar un ambiente de simulación para nodos móviles virtuales en redes vehiculares. Nuestras contribuciones se pueden enmarcar en la implementación de este ambiente partiendo de las investigaciones desarrolladas por Lynch et.al. [9] y los trabajos en redes vehiculares de Bravo, et.al [7]. Este ambiente de simulación dará el soporte para la implementación de los nuevos constructos que se proponen desde el Grupo de Investigaciones de la Universidad Politécnica Salesiana con respecto a los nodos virtuales en redes vehiculares.

Así, nuestro simulador, se desarrolló acorde a las propuestas de [7] y [42]. Se implementaron tres agentes: Master, Members y Constancy, en ns-2. El agente Master realiza un rastreo del movimiento del nodo y elige entre los nodos de la región un líder. El rastreo del nodo determina su ubicación, velocidad y dirección. El Agente Constancy inicia su función al momento de recibir un paquete desde el Master. El Constancy coordina las acciones de los tres agentes. El Agente Members inicia su función al momento en que el Constancy notifique de elección del líder, y se comprobó su funcionamiento en ambientes vehiculares.

5.3. Futuras Investigaciones

El interés de nuestra investigación en el campo de las VANETs, se centra en desarrollar protocolos para controlar el encaminamiento de los paquetes a intercambiar en una región virtual. Además, asignar funciones adicionales a cada uno de los nodos físicos dentro de un VMN, tales como: nodo servidor, nodo de respaldo del servidor y nodo cliente. Esto con el propósito de entregar un servicio de alta calidad en aplicaciones como: entretenimiento, control y gestión del tránsito vehicular, considerando, las características de una red VANET: i) alta variabilidad de la topología de red, ii) la limitada movilidad de los vehículos por factores externos (calles, semáforos, límites de velocidad, entre otras), iii) el rango de cobertura de las antenas (usualmente de 100m a 300 m).

Por otro lado se desarrollará el Analizador de Tránsito Vehicular, que analiza las principales características y estadísticas del tránsito vehicular generado por el simulador de tránsito vehicular, con la finalidad de proporcionar información suficiente a los diferentes elementos de la capa de virtualización.

Así, nuestro trabajo servirá como soporte para futuras investigaciones en el campo de la virtualización y las redes vehiculares. Destacamos el hecho de que actualmente se está desarrollando un segundo trabajo de grado en este campo como continuación de este trabajo en el encaminamiento de paquetes entre nodos virtuales.

BIBLIOGRAFIA

- [1] AquaLab. Straw <http://www.aqualab.cs.northwestern.edu/resources/9-projects/144-straw-street-random-waypoint-vehicular-mobility-model-for-network-simulations-e-g-car-networks>, April 2013.
- [2] Rimón Barr, Zygmunt J. Haas, and Robbert van Renesse. Java in simulation time (jist) <http://jist.ece.cornell.edu/>, April 2013.
- [3] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo - simulation of urban mobility: An overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 63–68, Barcelona, Spain, October 2011.
- [4] Oscar Martínez Bonastre and Carlos Palau Salvador. *Introducción a la programación de protocolos de comunicaciones con Network Simulator 2*. 2012.
- [5] J.F. Bravo-Torres, M. Lopez-Nores, and Y. Bianco-Fernandez. On the use of virtual mobile nodes with real-world considerations in vehicular ad hoc networks. In *Communications (COMM), 2012 9th International Conference on*, pages 193–196, 2012.
- [6] J.F. Bravo-Torres, M. Lopez-Nores, and Y. Bianco-Fernandez. Supporting more efficient communications in vehicular ad hoc networks with new constructs based on a virtualization layer. In *HotMobile Doctoral Consortium*, San Diego, USA, feb 2012.
- [7] J.F. Bravo-Torres, M. Lopez-Nores, Y. Blanco-Fernandez, and J.J. Pazos-Arias. Virtual virtual circuits: One step beyond virtual mobile nodes in vehicular ad-hoc networks. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–2, 2012.
- [8] J.F. Bravo-Torres, M. Lopez-Nores, Y. Blanco-Fernandez, S. Servia-Rodriguez, and J. Garcia-Duque. A virtualization layer for mobile consumer devices to support demanding communication services in vehicular ad-hoc networks. In *Consumer Electronics (ICCE), 2012 IEEE International Conference on*, pages 225–226, 2012.

- [9] Matthew Brown, Seth Gilbert, Nancy Lynch, Calvin Newport, Tina Nolte, and Michael Spindel. The virtual node layer: a programming abstraction for wireless sensor networks. *SIGBED Rev.*, 4(3):7–12, July 2007.
- [10] Jorge David Alba Cruz. *Estudio y simulación de una red AD-HOC vehicular VANET*. Escuela Superior Politécnica del Ejercito, <http://repositorio.espe.edu.ec/handle/21000/317>, 2009.
- [11] Instituto Nacional de Tecnologías de la comunicación. Inteco http://www.inteco.es/wikiAction/Seguridad/Observatorio/area_juridica_seguridad/Enciclopedia/Articulos_1/Funcion_Hash, Abril 2013.
- [12] Helene Doumenc. *Estudio comparativo de protocolos de encaminamiento en redes VANET*. Universidad Politécnica de Madrid, 2008.
- [13] Teresa Fabuel, Antonio Zaragoza, José María Díaz, Oscar Ferrer, and Francisco Candelas. Kivans <http://www.disclab.ua.es/kiva/>, Abril 2013.
- [14] Mainak Ghosh and Sumit Goswami. Intelligent transportation using vanet <http://www.pcquest.com/pcquest/news/181591/intelligent-transportation-vanet>, Abril 2013.
- [15] Pablo Gil, Jorge Pomares, and Francisco Candelas. Main page from sumo <http://blogs.ua.es/redesitis/recursos-didacticos/simuladores-open-free/>, Abril 2013.
- [16] Rocío Murcia Hemández, Joan García Haro, and Esteban Egea López. Evaluación de herramientas de simulación de redes vehiculares. Master's thesis, Universidad Politécnica de Cartagena. Escuela Técnica Superior de Ingeniería de Telecomunicación., 2011.
- [17] Juan Vicente Capella Hernández. Introducción al simulador de redes ns-2. *Universidad Politécnica de Valencia*, page 10.
- [18] J. Härri, M. Fiore, F. Fethi, and C. Bonnet. Vanetmobisim <http://vanet.eurecom.fr/>, Abril 2013.
- [19] T. Issariyakul and E. Hossain. *An Introduction to Network Simulator NS2*. Springer London, Limited, 2009.

- [20] Raul Santos Leiva. *Simulación de VANETS (Vehicular Ad-Hoc Networks)*. Universidad Politécnica de Catalunya, Noviembre del 2007.
- [21] Hernán Vinicio Barba Molina and Juan Francisco Chafla Altamirano. *Simulación de una red VSAT full-duplex para acceso a internet usando la plataforma DVB-S y DVB-RCS*. Escuela Politécnica Nacional, Marzo 2006.
- [22] S. Navarra-Gavira. *Algoritmos Cross-Layer para la Optimización de las prestaciones del TCP en Redes Wireless Ad-hoc*. Escuela Técnica Superior De Ingenieros, Departamento de Teoría de la Señal y Comunicaciones, 2006.
- [23] Vanet's Vehicular Adhoc Networks. Main page from sumo http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2010_2/lemons/introducao.html, Abril 2013.
- [24] Physiome Project NSR, the National Simulation Resource at the University of Washington Department of Bioengineering. Java-based simulator (jsim) <http://www.physiome.org/jsim/>, Abril 2013.
- [25] OpenStreetMap. Openstreetmap <http://www.openstreetmap.org/copyrightE>, Abril, 2013.
- [26] P. Papadimitratos, A. La Fortelle, K. Evenssen, R. Brignolo, and S. Cosenza. Vehicular communication systems: Enabling technologies, applications, and future outlook on intelligent transportation. *Communications Magazine, IEEE*, 47(11):84–95, 2009.
- [27] Dieter Pfoser, Sotiris Brakatsoulas, Petra Brosch, Martina Umlauf, Nektaria Tryfona, and Giorgos Tsironis. Dynamic travel time provision for road networks. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, GIS '08, pages 68:1–68:4, New York, NY, USA, 2008. ACM.
- [28] The VINT project. Network simulator (ns-2) <http://isi.edu/nsnam/ns/>, Abril 2013.

- [29] Juan Carlos Andrade Quinga and Soffía Veónica Naranjo Culqui. *Análisis del comportamiento de la tecnología WIMAX (IEEE 802.16) y WIMAX MOBILE (IEEE 802.16E) con tráfico de voz y datos en varios escenarios, usando el simulador NS-2*. Escuela Politécnica Nacional, Publicado: 5 de junio de 2012 2012.
- [30] Daniel Camilo Rojas Quirós. *Herramientas de control en redes de computadoras con el simulador NS-2*. Universidad de Costa Rica, Diciembre de 2010.
- [31] Muhammad Abdul Aleem Zulfiqar Ali Bhutto Asad Ali Shaikh Muhmaad Aslam Kumbhar Rahat Ali Khan, Shakeel Ahmed Shah. Wireless sensor networks: A solution for smart transportation. *Institute of Information and Communication Technology, University of Sindh Jamshoro Pakistan*, page 6, Abril de 2012.
- [32] Fernanda RODRÍGUEZ. Elección simulador de red http://iie.fing.edu.uy/investigacion/grupos/artes/csic_mac_2012/Elecci%C3%B3nNS3.pdf, Abril 2013.
- [33] Francisco J. Ros and Pedro M. Ruiz. Implementing a new manet unicast routing protocol in ns2. *Dept. of Information and Communications Engineering, University of Murcia*, December, 2004.
- [34] Hernán Samaniego-Armigos. *Simulación de redes vehiculares / VANET en entornos reales*. Universidad de Málaga, Escuela Técnica Superior De Ingeniería de Telecomunicación, Publicado: 5 de junio de 2012 2012.
- [35] Pablo Picazo Sánchez. Cooperación en redes vehiculares. estado de la cuestión y propuesta de mecanismo basado en incentivos. Master's thesis, Universidad Carlos III de Madrid Especialidad: Sistemas distribuidos, multimedia y seguros, Agosto 2011.
- [36] M.C. Spindel, Massachusetts Institute of Technology. Dept. of Electrical Engineering, and Computer Science. *Simulation and Evaluation of the Reactive Virtual Node Layer*. Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2008.

- [37] Main Page From sumo. Main page from sumo http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page, Abril 2013.
- [38] C.K. Toh. *Ad hoc mobile wireless networks: protocols and systems*. Prentice Hall PTR, 2002.
- [39] Ontario Canada University of Waterloo, Waterloo. Vanet/dtn - description <http://bbcr.uwaterloo.ca/slcesped/vanet/index.html>, Abril 2013.
- [40] Kannan Varadhan. The ns manual (formerly ns notes and documentation, 2003).
- [41] András Varga. Omnet++ <http://www.omnetpp.org/>, Abril 2013.
- [42] J. Wu. *A Simulation Study on Using the Virtual Node Layer to Implement Efficient and Reliable Manet Protocols*. City University of New York, 2011.
- [43] Jiang Wu, Nancy Griffeth, Nancy Lynch, Calvin Newport, and Ralph Droms. Simulating fixed virtual nodes for adapting wireline protocols to manet. In *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, NCA '09, pages 12–19, Washington, DC, USA, 2009. IEEE Computer Society.
- [44] S. Yousefi, M.S. Mousavi, and M. Fathy. Vehicular ad hoc networks (vanets): Challenges and perspectives. In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pages 761–766, 2006.

ANEXOS

ANEXOS

Los anexos, son adjuntados en formato digital, (véase el disco adjunto).

Anexos [1]: contiene a la estructura física del ambiente de simulación en diagramas UML.

Anexos [2]: contiene al ambiente de simulación en forma de documento, se utilizó la herramienta Doxygen.

Anexos [3]: contiene los diez archivos del ambiente de simulación

Anexos [4.1]: contiene los ficheros de respuesta de simulación del experimento con nodos estáticos.

Anexos [4.2]: contiene los ficheros de respuesta de simulación del experimento con MANET.

Anexos [4.3]: contiene los ficheros de respuesta de simulación del experimento con VANET.

Anexos [4.4]: contiene los ficheros de respuesta de simulación del experimento con los agentes Master, Constancy y Members.