Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Aplicación de las redes tolerantes a retardos e interrupciones (DTN) en entornos tácticos

Máster en Gestión y Dirección de Sistemas y Tecnologías de la Información y las Comunicaciones y de Seguridad de la Información

ALUMNO: Jose María Cuenca Caballero

DIRECTORES: Felipe Gil Castiñeira

CURSO ACADÉMICO: 2017-2018

(Página dejada intencionadamente en blanco)

Centro Universitario de la Defensa
en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Aplicación de las redes tolerantes a retardos e interrupciones (DTN) en entornos tácticos

Máster en Gestión y Dirección de Sistemas y Tecnologías de la Información y las Comunicaciones y de Seguridad de la Información

(Página dejada intencionadamente en blanco)

RESUMEN

Las comunicaciones en entornos tácticos están basadas en tecnologías de radio que permiten conectar a las unidades móviles, ya sean vehículos o soldados desmontados, con el correspondiente puesto de mando. Este tipo de entorno de trabajo, además de la movilidad, tiene otro tipo de características, como la orografía y la vegetación, los escenarios urbanos, el entorno radioeléctrico ruidoso debido a interferencias, o el ancho de banda reducido, que dificultan las comunicaciones por radio, provocando situaciones de pérdida de la conectividad con el elemento responsable de cada grupo jerárquico.

En este tipo de escenarios, el funcionamiento de aplicaciones informáticas convencionales se ve muy limitado, o incluso imposibilitado, debido al reducido ancho de banda disponible, a la elevada latencia y a la pérdida de conectividad de los nodos.

Algunos sistemas tácticos de información y comunicaciones han intentado resolver este problema mediante soluciones propietarias que pueden dar lugar a problemas de interoperabilidad. Las redes tolerantes a retardos e interrupciones (DTN) son una tecnología basada en estándares que puede ser una solución a esta problemática.

Este TFM busca estudiar la aplicación de las DTN en entornos tácticos para proporcionar una solución a las interrupciones que pueden sufrir las comunicaciones. Para ello, por una parte se han estudiado los casos en los que, de forma experimental, se han realizado desarrollos para integrar implementaciones de DTN en entornos tácticos, y por otra se ha modificado el simulador “The ONE” para modelar un escenario similar al de un entorno táctico en el que poner a prueba el comportamiento de las DTNs.

PALABRAS CLAVE

DTN, simulación, The ONE, radio táctica, interoperabilidad.

(Página dejada intencionadamente en blanco)

CONTENIDO

ÍNDICE DE FIGURAS

(Página dejada intencionadamente en blanco)

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

Las redes telecomunicaciones utilizadas en la vida forma cotidiana, a pesar de estar basadas en tecnologías de naturaleza muy diferente (canales de comunicaciones basados en soporte físico consistente en cableado de metales conductores o fibra óptica, o sistemas inalámbricos que funcionan en diferentes bandas de frecuencia, con modulaciones, técnicas de acceso al medio y compartición de recursos entre usuarios y codificaciones variadas) comparten, desde un punto de vista genérico, valores de los parámetros de ancho de banda, latencia, tasa de errores y disponibilidad del enlace, dentro de unos rangos más o menos amplios, que permiten el correcto funcionamiento de los protocolos y aplicaciones desarrollados desde el origen de la red Internet, como son TCP/UDP, IP, y las tecnologías de la capa de aplicación.

Sin embargo hay escenarios para los que los valores de los parámetros antes mencionados difieren en gran medida de los de las redes de uso cotidiano, lo que impide el funcionamiento de los protocolos y aplicaciones convencionales. Es necesario contar con soluciones adaptadas a estos entornos de comunicaciones extremas. El paradigma DTN es una de estas soluciones. Algunos ejemplos de escenarios [1] donde las DTNs son de aplicación son:

- Comunicaciones en entornos militares tácticos donde, en función de las circunstancias, los efectivos pueden estar cierto tiempo en zonas sin conectividad con la red de operación.
- Comunicaciones en entornos rurales que carecen de conectividad convencional a Internet, y esta se debe conseguir mediante “mulas de datos” (elementos que se encargan del transporte de los datos entre los puntos sin conectividad).
- Comunicaciones para redes de sensores en escenarios de reducida conectividad convencional, como los casos de seguimiento de fauna salvaje.
- Comunicaciones con naves espaciales en entornos de espacio profundo donde, en función de las circunstancias (interrupción de las comunicaciones por interposición de planetas, satélites u otros astros, o por ruido cósmico; elevado retardo debido a la distancia, etc.) no es viable la comunicación basada en protocolos y aplicaciones convencionales.
- Comunicaciones basadas en tecnología acústica submarina (utilizada con vehículos submarinos no tripulados, sistemas de monitorización de canalizaciones submarinas, etc.), que se caracteriza por tener un ancho de banda muy reducido y elevado retardo debido a la baja velocidad de propagación de las ondas acústicas.

Estos escenarios, si bien difieren mucho entre sí, comparten ciertas características comunes [2] que condicionan las comunicaciones:

- Conexión intermitente de los nodos de la red: debido, bien a la movilidad de los nodos (por ejemplo, nodos en movimiento en entornos urbanos o montañosos), bien a las circunstancias del entorno (por ejemplo, zonas con una elevada densidad de nodos utilizando tecnologías de transmisión por radio en las mismas bandas).
- Niveles bajos de seguridad: asociados al uso predominante de tecnologías de transmisión inalámbricas, lo que aumenta la posibilidad de problemas como elevados niveles de ruido, interferencias o datos destinados a provocar una denegación de servicio en el interfaz de radio provocado por *jammers* o escuchas por terceras partes.
- Topología de red cambiante: asociada al movimiento de los nodos, lo que puede conllevar que la aparición de un nodo en otra ubicación altere las rutas de encaminamiento.
- Conexiones de muy bajo rendimiento: caracterizadas por tiempos de transmisión y tiempos de "encolado" muy elevados (en comunicaciones espaciales podrían ser de horas o días), eficiencia de transmisión muy baja debido a errores y bajas tasas de transmisión provocadas por el uso de modulaciones robustas.
- Nodos con bajas capacidades, e incluso tiempos de vida muy limitados: debido, bien a la limitación de tamaño y disponibilidad de energía a la que está sometido el nodo (por ejemplo, dispositivos muy pequeños instalados en collares de seguimiento de fauna o dispositivos instalados en satélites o naves cuya única alimentación provenga de paneles solares), bien por la corta duración de la vida del dispositivo como consecuencia de las circunstancias ambientales en las que debe trabajar (por ejemplo, dosis elevadas de radiación o temperaturas extremas).
- Tecnologías de conexión heterogéneas: ya que, en cada zona, o incluso dentro de una misma zona, las tecnologías de transmisión disponibles entre los nodos, pueden ser diferentes (por ejemplo, WiFi, satélite, comunicaciones de espacio profundo, ondas submarinas acústicas, etc.).

A la vista de los condicionantes que afectan a las comunicaciones en los escenarios planteados, sin necesidad de entrar en el detalle de los protocolos utilizados en las redes convencionales, es evidente que la baja fiabilidad y disponibilidad de los enlaces y los elevados retardos, provocarían el vencimiento de temporizadores (bien a nivel TCP, bien a nivel de capa de aplicación aunque se utilizase UDP) que darían lugar a permanentes retransmisiones de los paquetes, lo que impide el uso directo de las tecnologías convencionales.

Las comunicaciones militares en entornos tácticos utilizan tecnologías de transmisión vía radio y resulta evidente que están afectadas por varios de estos problemas, lo cual da lugar a pérdida de la conectividad de los nodos con el elemento responsable de cada grupo jerárquico, lo que limita, o incluso imposibilita el uso de aplicaciones informáticas convencionales. Algunos sistemas tácticos de información y comunicaciones[1] han intentado resolver este problema mediante soluciones propietarias. Las DTNs son una tecnología basada en estándares que puede ser una solución a esta problemática, por lo que el estudio de su aplicación resulta conveniente y motiva el presente trabajo.

[1] Sistema SYNAPS de Thales Group (https://www.thalesgroup.com/es/synaps).

El principio de funcionamiento de las DTNs se basa en la transmisión, de forma asíncrona, de mensajes (*asynchronous message oriented*) entre los nodos de la red. Salvando las diferencias, la transmisión de mensajes en estas redes se asemeja al servicio de correo electrónico.

Debido a las especiales características de conectividad entre los nodos de este tipo de redes (la disponibilidad de un enlace puede tardar segundos, minutos, o incluso horas o días), el envío de este tipo de mensajes utiliza una estrategia de transferencia con custodia, en la que los nodos que reciben un mensaje y aceptan la custodia del mismo, lo almacenan hasta que reciben la confirmación de custodia del mensaje por otro nodo de la ruta o la recepción del mensaje por el destinatario, por lo que los nodos deben disponer de un sistema de almacenamiento persistente.

1.2 Objetivos del trabajo

Como se recoge en el epígrafe anterior, las comunicaciones vía radio utilizadas en los entornos militares tácticos se ven afectadas por una serie de problemas que dificultan, o incluso impiden, el uso de aplicaciones convencionales.

Hasta la actualidad, lo más habitual es que la comunicación en este tipo de escenarios se haya basado en voz, salvo escenarios específicos como las compañías de carros de combate con el sistema BMS-Lince o las baterías de artillería con el sistema TALOS[2]. Pero la evolución del combate lleva a conceptos como el "combate colaborativo" o el "ComFut", que requiere de la comunicación de datos.

La suma de estos 2 hechos (la necesidad de comunicación de datos en escenarios donde la conectividad entre nodos se puede ver muy degradada) hace que sea necesario buscar soluciones que hagan viable este tipo de comunicaciones. Por otra parte, es deseable que estas soluciones se basen en estándares que eviten al cliente ser cautivo de soluciones propietarias de los fabricantes.

Teniendo en cuenta estas necesidades, y considerando que los principios de funcionamiento de las DTNs parecen dar solución a la problemática planteada, los objetivos que este TFM se propone alcanzar son los siguientes:

- Revisar los estándares y documentación existente para exponer con claridad y elevado nivel de detalle la arquitectura y fundamentos técnicos de la tecnología DTN.
- Realizar una búsqueda de información para identificar el uso de DTNs en entornos militares tácticos, que es donde, a priori, esta tecnología es aplicable, y el estado de madurez de dichos proyectos.
- Utilizar un simulador para modelar un escenario en el que poner a prueba la aplicabilidad de las DTNs en entornos militares tácticos.
- Analizar los resultados de las simulaciones realizadas para verificar la aplicabilidad de las DTNs en entornos militares tácticos y definir parámetros que optimicen el rendimiento de la red simulada.
- En el caso de que se compruebe que las DTNs pueden constituir una solución a las comunicaciones en entornos militares tácticos, identificar las siguientes líneas de trabajo para profundizar en el conocimiento necesario para plantear una implementación de la tecnología DTN en equipamiento real.

[2] https://www.gmv.com/es/Productos/Talos/

1.3 Organización del trabajo

El presente TFM tiene 2 partes claramente diferenciadas.

El Capítulo 2 constituye la primera de estas partes. Debido al desconocimiento generalizado de las DTNs, este capítulo comienza exponiendo la arquitectura, protocolos y tecnologías en las que se basan las DTNs para, a continuación, presentar con un elevado nivel de detalle las implementaciones conocidas para entornos militares tácticos.

La segunda parte tiene un carácter práctico y se desarrolla en 2 capítulos. En el Capítulo 3 se expone el desarrollo realizado con el simulador ONE (*Opportunistic Networking Environment*) para definir un escenario que recree un entorno militar táctico en el que realizar las simulaciones. En el Capítulo 4 se presentan los resultados de dichas simulaciones y se recogen las conclusiones de carácter técnico extraídas.

El Capítulo 5 contiene las conclusiones del TFM y una propuesta de líneas de trabajo para avanzar en el conocimiento de las DTNs aplicadas a entornos militares tácticos.

2 ESTADO DEL ARTE

2.1 Orígenes y evolución de las DTNs

2.1.1 Orígenes de las DTNs

El origen de las redes tolerantes a retardos e interrupciones está en las investigaciones sobre comunicaciones interplanetarias [3] lideradas por Vinton Cerf a principios de la década de los '00.

Las primeras referencias [4] al concepto DTN fueron hechas por Kevin Fall en 2002. Al año siguiente, en 2003, V. Cerf y K. Fall propusieron de manera informal una arquitectura [5] para las DTNs. La utilidad e interés de este tipo de redes desembocó en la creación, dentro de la *Internet Research Task Force* (IRTF) del DTN *Research Group* (DTNRG), basado en el *Interplanetary Network Research Group* (IPNRG).

Como resultado de los trabajos de investigación realizados, en 2007 se publicaron las RFCs que establecen una arquitectura para las DTN y especifican el *Bundle Protocol* (BP), y en 2008 se publicó la RFC con la especificación del *Licklider Transmission Protocol* (LTP).

Una vez especificada la arquitectura y los protocolos necesarios para el funcionamiento básico de las DTNs, se desarrolló un simulador de este tipo de redes [6], y posteriormente han ido apareciendo implementaciones plasmadas en proyectos reales [2], como un sistema de monitorización de la congestión en situaciones de desastes en Reino Unido, una red experimental para dotar de conectividad a la comunidad Sami en el norte de Suecia, un sistema de seguimientote fauna salvaje en África, o el sistema americano CONDOR para comunicaciones en entornos tácticos. Existe un proyecto de software libre que ofrece una implementación de DTN para sistemas operativos Linux y POSIX sobre microprocesadores ARM Cortex [7].

2.1.2 Evolución de las DTNs: Estándares

El origen de las DTNs tiene casi 2 décadas pero, puesto que los casos de uso no están asociados a la vida cotidiana de la mayor parte de la población, este tipo de redes no ha tenido un despliegue real masivo. Aún así, su interés y utilidad están fuera de toda duda, por lo que la investigación y el desarrollo de estas redes han continuado.

Puesto que la investigación y desarrollo de las DTNs se lleva a cabo en el ámbito del IRTF, las especificaciones desarrolladas se plasman en forma de RFCs.

Las principales RFCs que especifican la arquitectura, protocolos, capas de convergencia y otras extensiones a las especificaciones originales son las siguientes:

- RFC 4838: Delay-Tolerant Networking Architecture. 2006-12-15. Informativa.
- RFC 5050: Bundle Protocol Specification. 2007-07-05. Experimental.
- RFC 5325: Licklider Transmission Protocol – Motivation. 2008-06-25. Informativa.
- RFC 5326: Licklider Transmission Protocol – Specification. 2008-06-25. Experimental.
- RFC 5327: Licklider Transmission Protocol - Security Extensions. 2008-06-25. Experimental.
- RFC 6255: Delay-Tolerant Networking Bundle Protocol IANA Registries. 2011-02-25. Informativa.
- RFC 6256: Using Self-Delimiting Numeric Values in Protocols. 2011-02-23. Informativa.
- RFC 6257: Bundle Security Protocol Specification. 2011-03-11. Experimental.
- RFC 6258: Delay-Tolerant Networking Metadata Extension Block. 2011-02-18. Experimental.
- RFC 6259: Delay-Tolerant Networking Previous-Hop Insertion Block. 2010-05-17. Experimental.
- RFC 6260: Compressed Bundle Header Encoding (CBHE). 2011-02-28. Experimental.
- RFC 6693: Probabilistic Routing Protocol for Intermittently Connected Networks. 2012-05-22. Experimental.
- RFC 7116: Licklider Transmission Protocol (LTP), Compressed Bundle Header Encoding (CBHE), and Bundle Protocol IANA Registries. 2014-02-. Informativa.
- RFC 7122: Datagram Convergence Layers for the Delay- and Disruption-Tolerant Networking (DTN) Bundle Protocol and Licklider Transmission Protocol (LTP). 2013-10-17. Experimental.
- RFC 7242: Delay-Tolerant Networking TCP Convergence-Layer Protocol. 2014-03-10. Experimental.

A la vista de este listado se comprueba que el estado de estas RFCs es Experimental o Informativo, lo que indica que este tipo de redes aún no ha alcanzado un grado de madurez elevado.

2.2 Introducción a la tecnología DTN

Como ya se ha indicado, si bien la utilidad de las DTNs está fuera de toda duda, la singular naturaleza de los entornos en los que son de aplicación hace que sea un tipo de red poco conocido incluso para personas con una cierta cultura en el campo de las redes de comunicaciones.

Por este motivo se considera conveniente incluir en el TFM, además de las referencias a las RFCs incluidas en el epígrafe anterior, una breve descripción de los aspectos más importantes de las DTNs.

2.2.1 Arquitectura

La arquitectura de las DTNs está definida en la RFC 4838 de IETF [8]. Esta arquitectura define una DTN como una red de redes, donde cada una de estas redes (denominadas regiones DTN) puede tener una naturaleza diferente. Las regiones DTN están formadas por *hosts* y *routers* y, la conexión con otras regiones DTN se lleva a cabo por medio de *gateways*. La Figura 2-1 muestra un ejemplo de una DTN formada por 3 regiones.

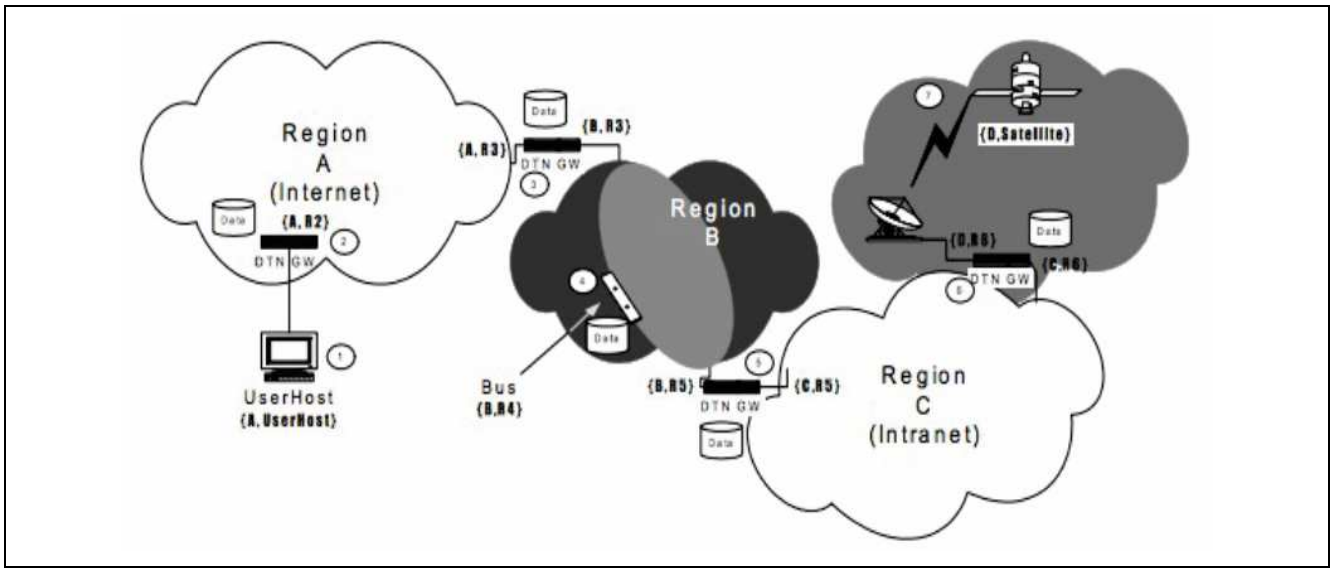

Figura 2-1 Ejemplo de DTN (tomada de [2])

Tipos de nodos de una DTN

En las DTN, como en la mostrada en la Figura 2-1, se distinguen 3 tipos de entidades:

- Host: actúan como dispositivos de los extremos de la comunicación que envían y reciben mensajes, pero no los reenvían. Estos dispositivos implementan la capa Bundle, diseñada para entornos en los que es necesario operar con retrasos largos, por lo que deben disponer de almacenamiento persistente para poder almacenar los mensajes en cola hasta que hay un enlace disponible.
- Routers: llevan a cabo el reenvío de mensajes dentro de una región DTN. Estos dispositivos también implementan la capa Bundle, por lo que deben disponer de almacenamiento persistente. Estos dispositivos, además pueden actuar como hosts.
- Gateways: llevan a cabo el reenvío de mensajes entre 2 o más regiones DTN, para ello, implementan las capas inferiores de las arquitecturas de red de las diferentes regiones que conectan. Estos dispositivos también implementan la capa Bundle, por lo que deben disponer de almacenamiento persistente, pero además deben soportar la funcionalidad de custodia de mensajes en transferencia. Estos dispositivos, además pueden actuar como hosts.

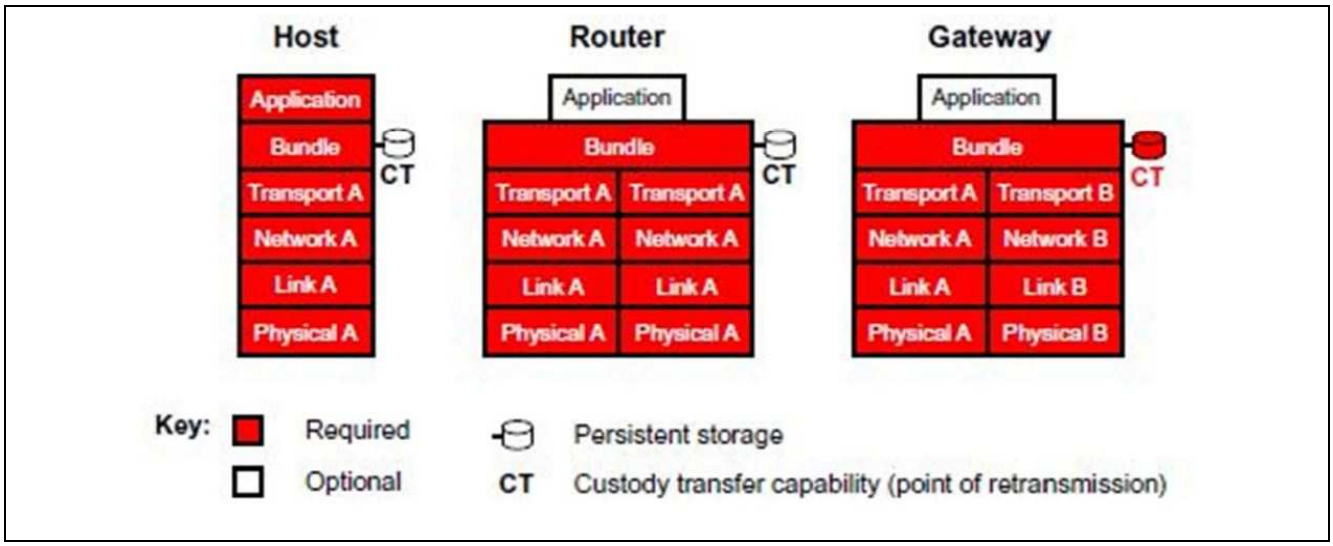

Figura 2-2 Arquitectura de los nodos DTN (tomada de [9])

Tipos de contactos entre nodos

Debido al bajo perfil de conectividad que caracteriza a las DTNs, la conexión entre nodos se considera positiva cuando el retardo y la capacidad pueden ser considerados constantes. El periodo de tiempo en el que se dan estos condicionantes se denomina "contacto" [8].

Según sus características, se distinguen 5 tipos de contactos:

- Contactos persistentes: son los que están siempre disponibles, por lo que no es necesario el establecimiento de conexión. Ejemplos de este tipo de contactos son las conexiones mediante routers ADSL o cable-modems.
- Contactos bajo demanda: son los que requieren una acción para iniciar la conexión pero, una vez establecida, se comportan como un contacto persistente. Un ejemplo de este tipo de contactos son las conexiones mediante modems telefónicos clásicos.
- Contactos intermitentes "planificados": son aquellos que se establecen, según un acuerdo previo, a una hora y con una duración determinados.
- Contactos intermitentes "oportunísticos": son aquellos que se establecen de forma imprevista, sin una planificación o expectativa previa. Un ejemplo de este tipo de contactos sería una conexión provista por un vehículo dotado de un nodo DTN que casualmente pasa por un punto donde hay otro nodo DTN.
- Contactos intermitentes "predichos": son aquellos que, sin estar planificados en base a un acuerdo, por la experiencia previa se espera que se produzcan a una hora y con una duración determinados.

Direccionamiento en DTNs

Los mensajes enviados por una DTN puede entregarse, bien a un único nodo, bien a un conjunto de nodos, denominado "DTN *endpoint*" [8]. Según se defina el conjunto de nodos que forma parte del "DTN *endpoint*", y el conjunto mínimo de nodos del "DTN *endpoint*" que debe recibir el mensaje para considerar que la entrega se ha realizado con éxito, se hablará de diferentes tipos de tráfico:

- *Unicast*: el mensaje es entregado al único nodo al que es enviado.
- *Anycast*: el mensaje es entregado a un nodo de un "DTN *endpoint*".
- *Multicast* o *Broadcast*: el mensaje es entregado a todos los nodos de un "DTN *endpoint*".

En sus orígenes, en las DTNs se utilizaba un direccionamiento del tipo: **{<regionID> , <entityID>}**, como por ejemplo, **{earth.sol.int , src.someclient.com:1131}**.

Posteriormente la RFC 4838 definió que la forma de especificar a los DTN *endpoints* con lo que se designó EID (*Endpoint IDentifier*), sería mediante la sintaxis genérica basada en URI (*Uniform Resource Identifier*). De este modo, cada nodo tiene, al menos, un EID único que lo identifica de forma única. La primera parte de este URI está definida por IANA dentro de su esquema de nombres[3] como "dtn".

[3] https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

Siguiendo este esquema de direccionamiento basado en URIs (esto es: **[URL:URN]**), el direccionamiento de los nodos podrá estar basado en su nombre de dominio o incluso en las aplicaciones soportadas [9]:

dtn://mars.sol.int/src.habitat.com

dtn:http://www.uo.edu.cu

La estrategia seguida para resolver el direccionamiento en las DTNs recibe el nombre de "*late binding*" (podría traducirse como "traducción tardía"), y consiste en que la resolución del URI de un destino no tiene porque realizarse en el origen del mensaje, si no que puede ser realizada en la región de destino.

Esto difiere de la estrategia "*early binding*", que es la utilizada en las redes que forman parte del Internet convencional, donde debe hacerse una resolución del DNS antes de enviar el paquete.

En DTNs, donde la disponibilidad de los enlaces puede ser muy baja y hay que optimizar el tráfico y tener en cuenta que el envío de mensajes puede requerir mucho tiempo (incluso horas o días), la estrategia "*late binding*" mejora el comportamiento respecto a "*early binding*", ya que requiere menor intercambio de información entre nodos.

Enrutamiento

En las DTNs, el concepto clásico de enrutamiento no es directamente aplicable, porque este concepto descansa sobre la idea de algoritmos y protocolos que trabajan de forma conjunta para encontrar un camino que se mantiene, al menos durante un plazo de tiempo, entre origen y destino y, una vez encontrado el camino, los datos se envían por este camino. Sin embargo, en el campo de las DTNs, dadas las condiciones de conectividad ya explicadas, de forma genérica se puede asegurar que no habrá un camino entre origen y destino que esté activo durante un periodo de tiempo, si no una sucesión de contactos que pueden ser de diferente tipo (lo que lleva asociada diferente certidumbre), por lo que los protocolos clásicos de enrutamiento no tienen validez.

Por otra parte, la diversidad de tipos de contactos entre nodos, y la certeza y duración de estos, hace que las estrategias de enrutamiento que dan buen resultado para un tipo de DTNs no sean buenas para otro tipo.

Por este motivo, la RFC 4838 no especifica una estrategia de enrutamiento para la arquitectura DTN, si no que deja la solución abierta a nuevos protocolos que se desarrollen.

Las estrategias de enrutamiento utilizadas en el ámbito de las DTNs son [6], [10], [11] y [12]:

- Estrategia *single-copy*: en esta estrategia cada mensaje tiene una copia en la red, lo que minimiza la sobrecarga en la misma pero, salvo escenarios muy específicos, obtiene una eficacia muy reducida.

  Ejemplos de protocolos:

  - *Direct delivery*: consiste en que el nodo origen espera a contactar con el destino para entregarle el mensaje.
  - *First contact*: consiste en que cada nodo va entregando el mensaje al primer nodo con el que logra un contacto y, una vez entregado, lo borra.

- Enrutamiento basado en replicación de mensajes: la estrategia que sigue este tipo de protocolos consiste en que, el nodo que tiene un mensaje, envía réplicas de ese mensaje a los nodos con los que entra en contacto. Se distinguen 2 variantes: inundación y *n-copy*.

En inundación se entrega a los nodos de la red copias de los mensajes según diferentes estrategias. Si bien esta variante adolece de algunos problemas (no es escalable con el tamaño de la red, da lugar a congestión en el caso de regiones con muchos nodos, y requiere una gran cantidad de recursos), la experiencia pone de manifiesto que el rendimiento obtenido en los escenarios de DTNs habituales, es mejor que en el caso de protocolos de tipo "*forwarding*". Ejemplos de protocolos:

- *Epidemic routing*: es el más sencillo. Se trata de un protocolo por inundación que consiste en que los nodos replican el mensaje a todos los otros nodos con los que entran en contacto y no tienen ya una réplica. Tiene una eficiencia muy baja, pero garantiza que la entrega se realiza, y ocurre en el menor tiempo posible.
- *Probabilistic Routing Protocol using History of Encounters and Transitivity* (PRoPHET): incorpora una importante mejora respecto al anterior, ya que no realiza réplicas del mensaje a todos los nodos con los que entra en contacto si no que, basándose en el histórico de contactos y los datos intercambiados con otros nodos, mantiene información sobre la probabilidad de éxito en la entrega a destinos conocidos de la DTN.
- MaxProp: la mejora de este protocolo consiste en definir la prioridad con la que los mensajes enviados a cada destino deben ser replicados a los nodos con los que entra en contacto, según la probabilidad de que aparezca una ruta al destino desde ese nodo.

En *n-copy* se limita el número de copias de mensajes que circulan por la red, para evitar la saturación de la misma. Con el objeto de evitar la congestión de la red, se puede incorporar la técnica de "reenvío en 2 saltos", en la que la fuente sólo replica el mensaje en los N primeros nodos con los que contacta, y estos deben entregar el mensaje al destinatario (2 saltos). Ejemplo de protocolo:

- *Spray&Wait*: este protocolo tiene 2 variantes. Por un lado el modo normal, en la que el nodo emisor entrega una copia a cada nodo con el que tiene contacto hasta completar las N copias, y por otro el modo binario, en el que el nodo emisor entrega [N/2] copias del mensaje al primer nodo con el que tiene contacto, que a su vez, entrega la mitad de las copias a los nodos con los que contacta, y así sucesivamente hasta que se agota el número de copias, momento en el que se pasa al estado de espera, en el que los nodos que tienen una copia del mensaje esperan a entrar en contacto con el destinatario para entregarle el mensaje.

- Enrutamiento basado en "*forwarding*": esta estrategia de enrutamiento requiere que los *routers* tengan un conocimiento de la topología de la red, pero, para el cálculo de rutas los nodos que aparecen en un momento dado como no disponibles, pueden tenerse en cuenta en función del tipo de contacto con el que suelan ser visibles (contactos planificados y contactos predichos). Dado que la disponibilidad de estos contactos no es determinística, puede ocurrir que el mensaje se envié por una ruta que no vuelva a estar disponible y se produzca un error.

  Ejemplos de protocolos:

  - *Delay-Tolerant Link State Routing* (DTLSR): como su nombre indica, es un protocolo de estado de enlace que tiene la particularidad de que, cuando un nodo deja de estar activo, no es eliminado del algoritmo, si no que se indica en su métrica, de forma que la ruta se calcule considerando la probabilidad de que vuelva a estar activo. La métrica del salto en el que está involucrado el nodo inactivo va empeorando con el paso del tiempo en que continúa en este estado.
  - *Schedule-Aware Bundle Routing* (SABR): este protocolo utiliza un "*contact plan*", que contiene información sobre los contactos actuales y los previstos (planificados o predichos) para calcular la ruta por la que el envío del mensaje tardará menos tiempo en llegar al destino.

Aplicaciones

Como se ha puesto de manifiesto en epígrafes anteriores, las DTNs adolecen de importantes carencias en la calidad de conexión que resulta habitual en las redes convencionales, para las que se han desarrollado las aplicaciones que se utilizan de forma cotidiana. Esta circunstancia hace que, de forma genérica, las aplicaciones convencionales no puedan ser utilizadas en las DTNs.

El uso de aplicaciones sobre DTNs no puede ofrecer una experiencia de interactividad tal y como se entiende de forma convencional. Incluso muchas aplicaciones no funcionarán debido a que los retardos habituales en los escenarios de aplicación exceden los *timeouts* de ejecución de las mismas.

La solución a este problema pasa, bien por adaptar aplicaciones existentes o desarrollar otras nuevas para su uso sobre este tipo de redes, bien por utilizar *proxies* para las aplicaciones cuya naturaleza lo permita (como la navegación web, el correo electrónico o la transferencia de ficheros) [13].

La opción basada en la adaptación de aplicaciones existentes requiere el rediseño y compilación de las mismas para ser utilizadas con la API de DTN. Esto, al igual que el desarrollo de aplicaciones nuevas, requiere una gran cantidad de recursos, e impide la actualización de dichas aplicaciones al mismo ritmo que las desarrolladas para entornos convencionales.

El uso de *proxies* de nivel de aplicación consiste en que el *proxy* convierte el protocolo utilizado por la aplicación para intercambiar información en tráfico adecuado para ser transmitido por redes DTN. De esta forma el proxy se comporta como un servidor, que responde a la aplicación con los mensajes esperados. En [14] se estudian diferentes escenarios de interconexión de redes y se plantean 3 aproximaciones para resolver el problema de la adaptación del tráfico de las aplicaciones convencionales a las redes DTN:

- Funcionamiento *offline*: consiste en la replicación de contenidos de un servidor en almacenamiento local para que estos puedan ser accedidos *offline*.
- *Cache prefill*: consiste en alimentar un sistema caché con información que, de forma heurística, se determina que es muy probable que sea demandada por los clientes antes de que su validez expire.
- *On-demand (predictive) prefetching*: consiste en enviar de forma anticipada contenidos directamente vinculados a la petición realizada por el usuario (por ejemplo, objetos de enlaces incluidos en la respuesta enviada al usuario).

2.2.2 Bundle Protocol

Como se ha expuesto en epígrafes anteriores, el núcleo de las funcionalidades que dan lugar al concepto DTN está en una capa que se inserta entre las tradicionales capas de aplicación y transporte, denominada "*bundle layer*" (ver Figura 2-2), y que se materializa en el Bundle Protocol (BP). La RFC 5050 de la IETF [15] contiene la especificación de BP.

La función principal de BP es implementar el mecanismo de almacenamiento y reenvío ("*store & forward*") de *bundles*, que es como se denomina a la unidad de información o mensaje de la "*bundle layer*". La implementación del mecanismo "*store & forward*" requiere que el protocolo realice diversas funciones, entre las que destacan:

- Capacidad de almacenamiento persistente, necesaria para conservar los *bundles* hasta que estos son, bien entregados, bien su custodia transferida.

- Gestión de la custodia (transferencia, aceptación, rechazo y retención) de los *bundles*: como se expuso en un epígrafe anterior, este aspecto es la base del mecanismo "*store & forward*" que sostiene las DTNs.
- Gestión de la prioridad de los *bundles*: la RFC 4838 especifica 3 niveles de prioridad para el procesado de bundles. De mayor a menor prioridad son: *Expedited*, *Normal* y *Bulk*.
- Fragmentación y reensamblado de *bundles* para ajustar el tamaño de los mismos a las restricciones existentes en contactos con esta capacidad limitada.
- Gestión del esquema de direccionamiento basado en URIs, descrito en un epígrafe anterior, para determinar a qué nodo se realiza el envío del *bundle*.
- Seguridad: un aspecto esencial que no se ha tratado anteriormente está en la forma en la que BP provee seguridad a las DTNs, y que se introduce en el siguiente apartado.

Para llevar a cabo estas funciones, el formato de un bundle contiene, además de la información de aplicación, datos necesarios para la implementación de las funciones:

- Cabecera: contiene las direcciones de origen y destino del *bundle*, y la información de control necesaria para realizar las tareas de fragmentación y reensamblado, gestión de la custodia del *bundle*, *timestamps* y *lifetime* del *bundle*, datos sobre el diccionario usado en el *bundle*, entre otros campos [16].
- Control de la información: con este campo la aplicación que genera los datos que contiene el *bundle* indica la manera de manipular los mismos.
- Unidad de datos de aplicación (ADU – *Application Data Unit*): son los datos generados por la aplicación.

Seguridad

Además de la necesidad de seguridad requerida por redes de comunicación convencionales, las DTNs necesitan medidas adicionales para proteger sus recursos:

- Evitar que entidades no autorizadas puedan hacer uso de la red, para evitar ocupar los, ya de por sí, escasos recursos (tiempo de conectividad, ancho de banda, etc.).
- Garantizar la integridad de los datos transmitidos, para evitar desperdiciar la disponibilidad de contactos enviando *bundles* con datos corrompidos.
- Garantizar la confidencialidad de los datos transmitidos.

El BP aporta seguridad a las DTNs en 3 dimensiones diferentes mediante el uso, en los nodos, de certificados (con el correspondiente par de claves pública y privada) y firma de los mensajes. Estas 3 dimensiones de seguridad son provistas mediante la implementación de 3 bloques:

- *Payload Security Block* (PSB): este bloque permite al "*security-destination*" comprobar la integridad del *bundle* enviado desde el "*security-source*", y la autenticidad del propio "*security-source*". Es decir, se aplica extremo a extremo.
- *Bundle Authentication Block* (BAB): este bloque permite a cada nodo comprobar la integridad del *bundle* en cada salto y la autenticidad del nodo que ha realizado la transmisión, antes de procesarlo y, en su caso, reenviarlo. Es decir, se aplica salto a salto.

Como el PSB se aplica extremo a extremo, y el BAB se aplica salto a salto, en el caso de que ambas dimensiones de protección estén presentes, el BAB se aplica como capa más externa que contiene al PSB.

- *Confidentiality Block* (CB): este bloque aporta una funcionalidad de cifrado sobre el *payload* del mensaje para proveer confidencialidad. Se aplica extremo a extremo, es decir, entre el "*security-source*" y el "*security-destination*".

2.2.3 Capas de convergencia

Como especifica la RFC 5050, con el objeto de que BP pueda ser transportado mediante los protocolos de capas inferiores, las DTNs requieren la existencia de un adaptador de capas de convergencia (CLA – *Convergence Layer Adapter*), que determina como se encapsulan los *bundles* en protocolos de la capa de transporte (TCP, UDP, LTP) o incluso en tecnologías de capa de enlace.

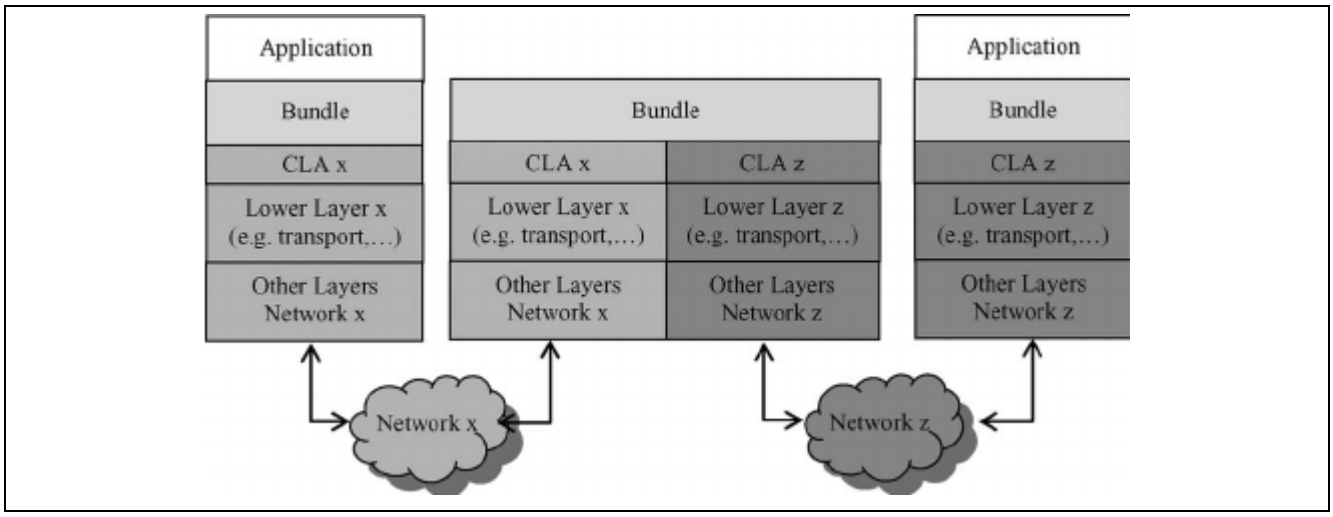

Figura 2-3 Convergente Layer Adapter - CLA (tomada de [17])

Las principales capas de convergencia definidas hasta la actualidad son [9]:

- *Licklider Transmission Protocol* (LTP): este protocolo, cuya especificación se encuentra en la RFC 5326 [18] tiene por objeto proveer una comunicación fiable en enlaces de elevada latencia y con interrupciones frecuentes, realizando un mapeo directo de los *bundles* sobre una tecnología de nivel de enlace, motivo por el que no tiene en consideración los aspectos de enrutamiento o control de congestión. Se trata, por lo tanto, de un protocolo diseñado para conexiones punto a punto, o de un solo salto.
- *Licklider Transmission Protocol - Transport* (LTP-T): este protocolo es una mejora del LTP original, que permite utilizarlo en escenario con varios saltos. El funcionamiento consiste en establecer, para cada contacto, una sesión LTP y, en el caso de que se produzca un error en la transmisión de un segmento, se solicita su retransmisión al nodo anterior.
- Capa de convergencia para TCP/IP: como su nombre indica, este protocolo realiza la encapsulación de *bundles* en paquetes TCP/IP, por lo que es de aplicación en contactos en los que los nodos establezcan este tipo de conexión (por ejemplo, redes WiFi), lo que suele ocurrir en escenarios con baja latencia y baja tasa de errores. Al utilizar TCP/IP se beneficia de las funcionalidades provistas por estos protocolos, como el control de flujo, el modo de funcionamiento orientado a conexión (entrega fiable, recepción de paquetes en orden, etc.), entre otras.

- Capa de convergencia Saratoga: este protocolo permite el uso de BP sobre los enlaces provistos por la red de satélites de órbita heliosincrónica de la *Disaster Monitoring Constellation* (DMC). Esta capa de convergencia trabaja sobre UDP/IP, por lo que no ofrece un servicio de entrega fiable ni garantiza la entrega ordenada de los datagramas.
- Otras capas de convergencia son UDP/IP y *Bluetooth*.

2.3 Implementaciones de la tecnología DTN

En los epígrafes anteriores se han presentado las funcionalidades y protocolos que sustentan el concepto de DTN. Con el objeto de facilitar la puesta en marcha de proyectos que utilicen estas redes se han desarrollado varias implementaciones de DTN, entre las que destacan 2: DTN2 e IBR-DTN.

La implementación de referencia es la denominada DTN2[4], desarrollada originalmente (implementación DTN1) por el DTNRG. En [19] se recoge el detalle de las decisiones de diseño, módulos, funcionalidades e interfaces de este desarrollo, así como resultados de su evaluación.

Unos años después se desarrollo la implementación IBR-DTN [20]. Esta implementación está especialmente desarrollada para sistemas embebidos, por lo que requiere menos recursos y conlleva un menor consumo de energía.

El estudio comparativo de ambas implementaciones llevado a cabo en [21] arroja los siguientes resultados:

- DTN2 logra un mayor *throughput* para grandes transferencias de información, pero para cantidades pequeñas el rendimiento es similar en ambas implementaciones.
- La transferencia de pequeñas cantidades de información requiere un porcentaje de uso de CPU similar en ambas implementaciones, pero para la transferencia de grandes cantidades de información el comportamiento de IBR-DTN es mejor, ya que requiere menos procesado.

2.4 Antecedentes de uso de las DTNs en entornos militares

2.4.1 CONDOR

En [22] se expone cómo se ha aplicado la tecnología DTN al proyecto CONDOR (*Command and Control On-The-Move Network Digital Over-The-Horizon Relay*) del cuerpo de Marines del ejército de Estados Unidos, que tiene por objeto el desarrollo de un sistema que extienda la conectividad de datos a entornos militares tácticos caracterizados por la interrupción de las conexiones de las tropas, convoys, etc., debido a la alta movilidad de los efectivos, los obstáculos del terreno, las interferencias y las zonas de sombra [9].

La red CONDOR tiene por objeto proveer comunicaciones fiables en escenarios en los que nodos ubicados en un entorno táctico con interrupciones en la conectividad y con comunicación entre ellos mediante radio táctica, se comunican con otro nodo remoto (*over-the-horizon*) mediante enlaces satelitales, bajo la premisa de que la información generada (por ejemplo imágenes, información logística, correo electrónico y, en función de los retardos de la red, localización de las tropas) tiene un periodo de validez superior a la duración de las interrupciones en las conexiones [22].

[4] https://github.com/delay-tolerant-networking/DTN2

http://dtn.sourceforge.net/DTN2/doc/manual/

En un despliegue de la red CONDOR se pueden distinguir 3 tipos de nodos complejos [22]:

- *Gateway*: es un vehículo que conecta redes EPLRS (*Enhanced Position Location Reporting System*) locales con otras remotas (*over-the-horizon*) donde se encontraría el *Tactical Operation Center* (TOC) mediante enlaces vía satélite. Está dotado de equipamiento de radio EPLRS, un *gateway* DTN, un equipo de comunicación vía satélite (TACSAT, INMARSAT o Iridium) y un cifrador para dotar de seguridad a la comunicación vía satélite. La Figura 2-4 muestra un esquema simplificado del sistema que compone el *gateway*, donde se puede ver que el agente DTN está ubicado en el *router* Cisco 3725. Para llevar a cabo la integración del agente DTN (consistente en la implementación de referencia DTN2 del DTNRG) en el sistema se consideraron 3 posibilidades:
  - Integrar la implementación DTN de forma nativa en el *firmware* del router. Si bien está es la opción ideal, resulta complejo a nivel técnico y puede afectar seriamente al rendimiento del dispositivo.
  - Añadir al sistema, conectándolo a uno de los puertos Ethernet del *router*, un ordenador sobre el que se despliegue la implementación de DTN. Esta es la solución más sencilla de implementar, pero requiere nuevo hardware e incrementa el consumo del sistema.
  - Incluir en uno de los *slots* vacantes del *router* Cisco 3725 un módulo NM-CIDS-K9. Este módulo, si bien incluye de origen un IDS, en realidad es un ordenador conectado al *backplane* del *router* con un sistema operativo Linux. Sobre este Linux se despliega la implementación DTN, así como los *proxies* de las aplicaciones. Conceptualmente esta opción es similar a la anterior, pero con una implementación más compacta, lo cual es un aspecto crítico en los vehículos tácticos, motivo por el que fue la opción implementada.
- PoP-V (*Point of Presence Vehicle*): es un vehículo que da conectividad a nodos cercanos dotados de equipos de radio táctica convencional (pueden ser otros vehículos o pequeñas unidades de soldados a pie) con el vehículo *gateway*, mediante EPLRS (incluso plantea la posibilidad de utilizar un enlace satelital TACSAT para situaciones *over-the-horizon*).
- JC2-V (*Jump Command and Control Vehicle*): es un vehículo que sirve de puesto de mando móvil, que mantendría la sincronización de servidores y bases de datos durante las maniobras u operaciones mediante un enlace satelital permanente.

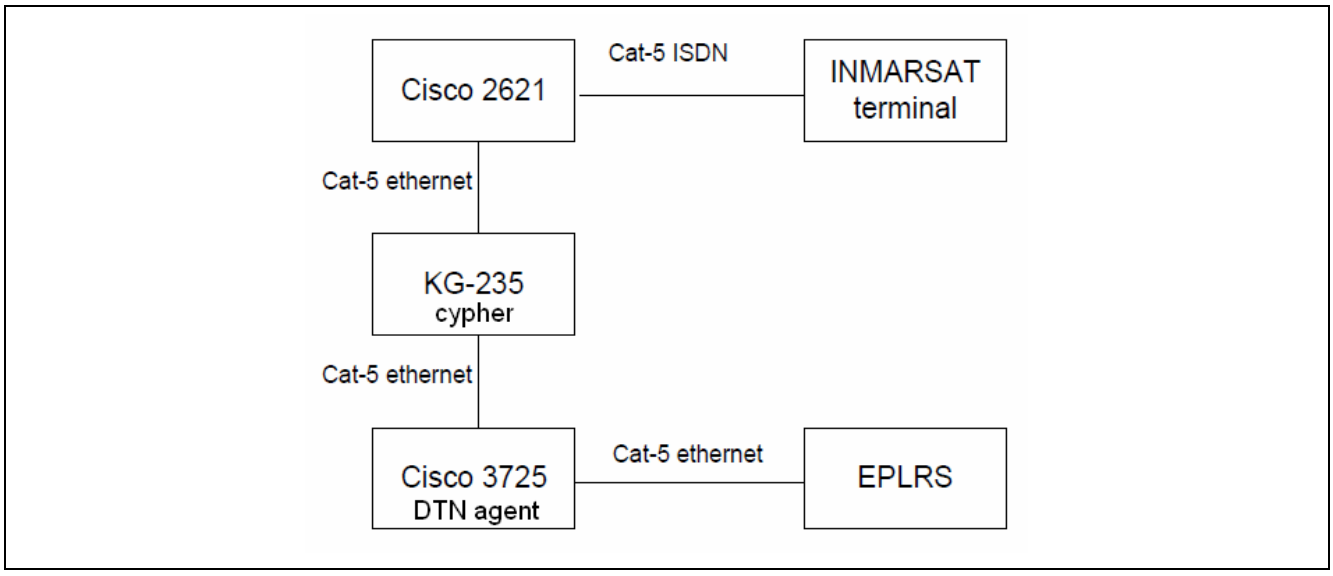

Figura 2-4 Esquema de un CONDOR *gateway* (tomada de [22])

La Figura 2-5 representa un escenario en el que están presentes los vehículos *gateway* y PoP-V. En esta representación se muestra como, cuando la comunicación entre vehículos *gateway* no puede ser llevada a cabo mediante radio EPLRS, el agente DTN establece el enlace satelital.

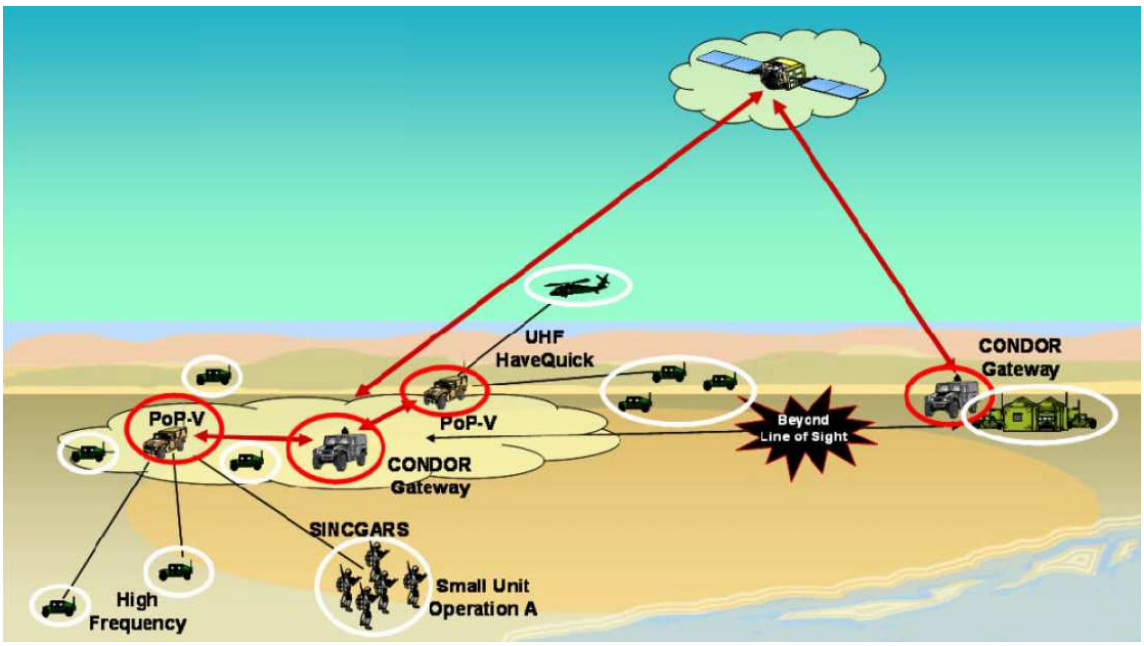

Figura 2-5 Escenario de una red CONDOR (tomada de [23])

El sistema CONDOR también basa el funcionamiento de las aplicaciones en el uso de *proxies* (desplegados en el módulo NM-CIDS-K9 del *router* Cisco 3725) que convierten el tráfico generado por las aplicaciones en bundles adecuados para ser enviados por la DTN, además de realizar las funciones necesarias para evitar los problemas generados por retardos o pérdidas de conectividad.

En [13] se explica de forma detallada el desarrollo y funcionamiento del *proxy* web utilizado en la red CONDOR, consistente en una modificación del *proxy* WWWOFFLE. El funcionamiento consiste en implementar la funcionalidad cliente en los nodos ubicados en el entorno táctico (donde se producen los problemas de conectividad), mientras que la funcionalidad servidor se despliega en la parte de la red que dispone de una conectividad estable. La Figura 2-6 muestra el tráfico resultante para una consulta realizada desde un nodo ubicado en el entorno táctico.

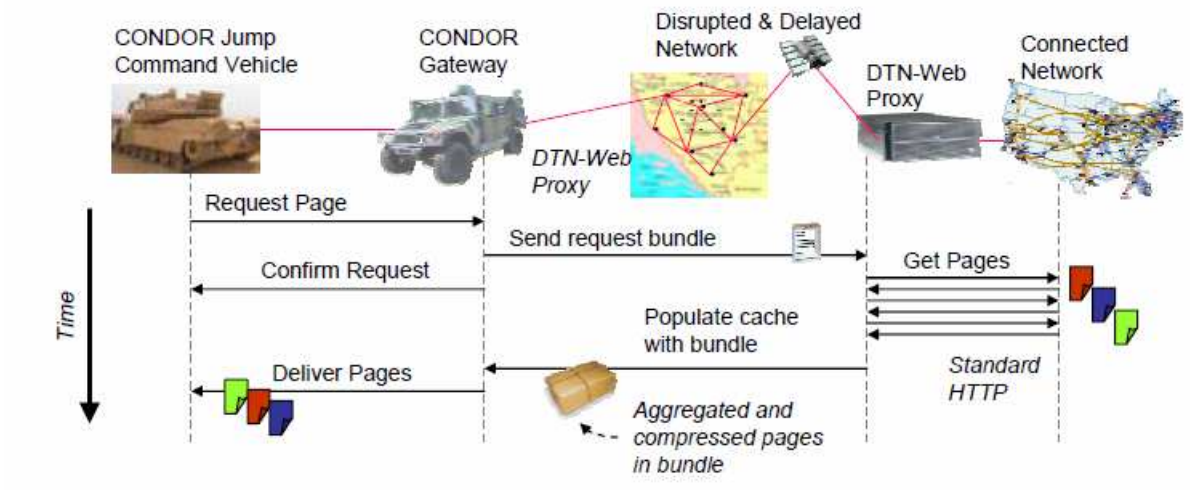

Figura 2-6 Modelo de comunicación de aplicación web en CONDOR mediante el uso de proxy DTN (tomada de [13])

Además de los resultados presentados en este epígrafe, en [22] se enumeran nuevas líneas de trabajo que se plantearon al concluir la parte del proyecto reflejada en este documento, entre ellas están el desarrollo de *proxies* para otras aplicaciones y trasladar la implementación de DTN a un *router* basado en chasis rugerizado.

2.4.2 Proyecto MIDNet de la Agencia Europea de Defensa

La Agencia Europea de Defensa (EDA, *European Defense Agency*) estableció en 2013 el proyecto "B 0882 IAP4 GC [MIDNet]" (*Military Disruption Tolerant Networks*) dentro del Capítulo IAP-4 (*Information Acquisition & Processing – CIS and Networks*) de su Programa de Investigación y Tecnología [24].

Este proyecto, con un presupuesto aproximado de 7.2M€ [24] y que finalizó en 2016, fue desarrollado por 9 compañías y organizaciones de I+D de Holanda, Finlandia, Portugal, Suecia, Polonia, Alemania y Francia, que lo lideró [25].

Como parte de los resultados de este proyecto se publicaron los artículos [25] y [26].

Como se ha expuesto en un epígrafe anterior, para lograr el correcto funcionamiento de aplicaciones existentes sobre DTNs hay 2 soluciones: por una parte modificar las aplicaciones, y por otra desarrollar un *proxy* que adapte convenientemente el tráfico de la aplicación original para que el funcionamiento de esta a través de una DTN sea correcto.

En [26] se da respuesta a esta decisión, optando por desarrollar, para las aplicaciones objeto del estudio, *proxies* que hagan transparente a las aplicaciones el funcionamiento de la DTN. Estos *proxies* funcionan entre la capa de aplicación y los *drivers* DTN, interceptando el tráfico generado por la aplicación y empaquetándolo en *bundles*. Las aplicaciones objeto del proyecto, y para las que se han desarrollado los *proxies*, han sido correo electrónico con protocolos SMTP y POP3, y *Blue Force Tracking*[5] (BFT).

Como se indica en [25], la opción elegida en este proyecto para desplegar la solución DTN ha sido la implementación IBR-DTN, en concreto, la versión 1.0.1. Además, la implementación IBR-DTN original fue mejorada con el desarrollo de una solución de QoS basada en 32 clases, definidas en base a 5 *flags*:

- 2 bits para definir 4 niveles de prioridad de transmisión, establece el orden en que se envían los *bundles*.
- 2 bits para definir 4 niveles de prioridad de retención: establece el orden en que se eliminan los *bundles* en caso de saturación de la capacidad de almacenamiento de los nodos.
- 1 bit *Keep Most Recent* (KMR): utilizado en aplicaciones con requisitos de tiempo real (como BFT) para indicar que los nuevos *bundles* sobrescriben los anteriores en el almacenamiento de los nodos.

Los equipos de radio con los que se planificaron las pruebas de campo descritas en [25] son habituales en comunicaciones tácticas en UHF y VHF (Transbit R-450C, Rohde&Schwarz SVFuA, Radmor RRC 9311 de la familia PR4G), si bien durante las pruebas sólo se utilizaron equipos Transbit en modo IP.

[5] Seguimiento de fuerzas amigas.

Las pruebas de campo se llevaron a cabo en Alemania en 2016 utilizando un escenario habitual en las operaciones llevadas a cabo por la coalición (varias patrullas de vigilancia en un entorno hostil).

Aunque durante las pruebas se produjeron varios problemas y no se recopiló toda la información que estaba planificada, sí que se obtuvieron los siguientes resultados:

- Las tasas de transmisión obtenidas durante las pruebas del servicio de correo electrónico con VHF fueron iguales a la capacidad nominal tanto al trabajar con los niveles de máxima potencia de las radios como al hacerlo con el mínimo nivel de potencia, aunque en este caso hubo varios instantes de pérdida de conectividad.
- Las pruebas de envío de correo electrónico se llevaron a cabo con paquetes de tamaño medio 500B y, aunque se produjeron retardos variables e importantes en la entrega, todos los correos fueron recibidos.
- La aplicación BFT funcionó de forma continua a pesar de las interrupciones en las comunicaciones.

Además de estos resultados, [25] identifica una serie de líneas de trabajo con las que continuar el proyecto MIDNet y profundizar en el estudio de aplicabilidad de las DTNs en entornos tácticos militares.

3 DESARROLLO DEL TFM

3.1 Introducción

Según se expuso en el Capítulo 1, dos de los objetivos de este TFM son, por un lado, validar la conveniencia del uso de DTNs en entornos militares tácticos con diferentes patrones de tráfico, y por otro, proponer un conjunto de parámetros que maximicen el rendimiento de la red en este tipo de escenarios. El contenido de los Capítulos 3 y 4, de marcado carácter técnico, da respuesta a estos objetivos.

La validación de la aplicabilidad de la tecnología de DTNs a los entornos militares tácticos puede hacerse siguiendo dos enfoques. El primero de los enfoques consistiría en adaptar una de las implementaciones de los estándares de DTNs a los equipos de comunicaciones militares y realizar pruebas de campo. El segundo de los enfoques consiste en utilizar una herramienta de simulación que permita un elevado nivel de similitud con un escenario real. Como se ha expuesto en el capítulo anterior, el primer enfoque supone una elevada complejidad y, por ello, está asociado a importantes inversiones económicas y de tiempo. Sin embargo, existen herramientas de uso libre y código abierto que permiten llevar a cabo el segundo de los enfoques sin necesidad de inversión económica y con una reducida inversión de tiempo.

Para realizar la validación, la metodología que se ha seguido ha consistido en la realización de simulaciones con diferentes patrones de tráfico en un escenario modelado con características propias de un entorno militar táctico real.

El análisis de los resultados obtenidos de estas simulaciones permite obtener conclusiones de carácter técnico en base a las que definir parámetros que permiten maximizar el rendimiento de la red modelada en el escenario definido.

Aunque existen multitud de simuladores de redes de datos, para el caso de las redes móviles ad-hoc el catálogo es muy limitado. En el campo de las DTNs, el catálogo se reduce aún más, y la principal herramienta de simulación disponible es el simulador ONE[6] (*Opportunistic Networking Environment*), que ha sido desarrollado específicamente para evaluar el comportamiento de los principales componentes de estas redes.

[6] http://akeranen.github.io/the-one/

3.2 El simulador ONE

El simulador ONE, cuyo principal desarrollador es el finlandés Ari Keränen, ha sido específicamente creado para el estudio de las DTNs.

En [6] se encuentra una detallada descripción del funcionamiento del simulador ONE y sus funcionalidades. Las características de mayor interés del simulador ONE para el desarrollo de este TFM son las siguientes:

- Puede ser ejecutado en modo gráfico o en modo consola [27]. El modo gráfico permite comprobar visualmente el comportamiento de los nodos y los contactos entre los mismos según diferentes modelos de movimiento. El modo consola permite lanzar lotes de simulaciones basadas en diferentes combinaciones de parámetros mediante un único comando, y ejecutar las mismas con una velocidad mayor que en el modo gráfico, dado que evita la sobrecarga del sistema asociada a la representación de los nodos en movimiento y sus enlaces.
- Evita la complejidad de la implementación de la capa de nivel de enlace e inferiores mediante la abstracción de las mismas en base a dos de los parámetros característicos, a saber, la tasa de transmisión y el radio de cobertura nominal de la tecnología [27].
- Está desarrollado en Java, lo que permite editar el código fuente con facilidad y modificar el comportamiento de cualquier funcionalidad del simulador.
- Permite definir diferentes interfaces de red para ejecutar las simulaciones.
- Permite definir diferentes grupos de nodos y, para cada grupo, permite definir los siguientes parámetros, comunes para todos sus nodos:
  - El modelo de movimiento.
  - El tamaño del buffer de los nodos.
  - El protocolo de enrutamiento.
  - El tiempo de vida de los bundles.
  - El inferfaz o interfaces con que están dotados los nodos.
  - El número de nodos.
- Permite utilizar, bien un generador automático de tráfico en tiempo real de simulación, bien especificar un fichero con un perfil de tráfico previamente establecido por el usuario.
- Genera diversos reportes con la información de cada simulación ejecutada, Estos reportes incluyen tanto información estadística del resultado de la simulación, como *logs* de los diferentes eventos que han ido ocurriendo durante la simulación.
- Incorpora diversos modelos de movimiento sintético para los nodos [27]. Estos modelos son de dos tipos:
  - Independientes de mapas: no requieren ninguna cartografía. En este caso los nodos se mueven en un área delimitada mediante parámetros siguiendo una determinada estrategia. Algunos de estos modelos son: *Random Walk*, *Random Waypoint* y *Cluster Movement*.
  - Basados en mapas: hacen uso de mapas convertidos a formato WKT (*Well Known Text*). En este caso los nodos se mueven siguiendo rutas representadas en un mapa. Estos modelos de movimiento ofrecen un comportamiento más cercano al de nodos que se desplazan en un entorno urbano. Los modelos incluidos en el simulador son MBM (*Random Map-Based Movement*), SPMBM (*Shortest Path Map-Based Movement*) y RMBM (*Routed Map-Based Movement*). Puesto que los nombres de los modelos son autoexplicativos y no son utilizados en las simulaciones ejecutadas en este TFM, no se dará mas detalle sobre los mismos.

- Incorpora diversos protocolos de enrutamiento utilizados en DTNs [27]. Los protocolos implementados en el simulador se pueden clasificar en 3 categorías:
  - *Single-copy*: en la red sólo existe una copia del *bundle* enviado. Los protocolos de esta categoría son *Direct Delivery* y *First Contact*.
  - *N-copy*: el *bundle* enviado se replica N veces en la red. El protocolo implementado es *Spray&Wait*, con las variantes modo normal y modo binario.
  - Inundación: estos protocolos siguen diferentes estrategias de inundación de la red con el *bundle* que se debe enviar. Los protocolos de esta categoría son *Epidemic*, *PRoPHET* y *MaxProp*.
- La versión actualmente disponible del simulador (v1.6.0-21e086d), y con la que se ha realizado esta parte del TFM, sólo soporta tráfico de tipo *unicast* en modo *half duplex*.

3.3 Preparación del entorno de trabajo

Para la realización de esta parte práctica del TFM se ha utilizado una máquina con sistema operativo Windows 10 y, en consecuencia, el software utilizado y las configuraciones realizadas que se describen a continuación son las propias de esta plataforma.

3.3.1 Descripción del software utilizado

La realización de las diferentes fases de esta parte del TFM ha requerido las siguientes herramientas software:

- Simulador ONE: la versión más actualizada del código fuente del simulador ONE (v1.6.0-21e086d) se puede descargar del correspondiente proyecto[7] en GitHub. Como ya se ha expuesto, el código fuente de este simulador está escrito en Java. En concreto, el fichero **README.txt** del proyecto indica que debe compilarse con la versión 6 de Java JDK o posteriores.
- Java JDK: se optó por utilizar la versión 6 de Java JDK[8] adecuada para la arquitectura de la máquina en la que se iba a ejecutar el simulador, en concreto, la versión "Windows x86".
- Intérprete Perl: se ha utilizado la herramienta DWinPerl[9], que es un intérprete válido para plataformas Microsoft Windows. Se ha utilizado la versión v5.14.2.1-v7-32, que es la última disponible. Este intérprete se ha utilizado para ejecutar el *script* que genera los ficheros de tráfico que, convenientemente adaptados, se utilizaron para realizar las simulaciones.

3.3.2 Instalación y configuración del software

Java JDK:

Para instalar el JDK (*Java Development Kit*) simplemente hay que seguir las indicaciones facilitadas por el asistente de instalación. Para lograr un correcto funcionamiento con el simulador ONE es suficiente seleccionar la ruta de instalación deseada para el JDK y aceptar las opciones ofrecidas por defecto por el asistente.

[7] https://github.com/akeranen/the-one

[8] https://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html

[9] http://dwimperl.szabgab.com/windows.html

Una vez instalado el software, es necesario añadir la ruta de la carpeta en la que se encuentra el ejecutable **javac.exe** a la Variable de Entorno **Path**, de tipo Variable de Sistema, del sistema Windows que se ha utilizado para desarrollar esta parte práctica del TFM.

Simulador ONE:

El código fuente del simulador que se descarga del proyecto en GitHub consiste en un fichero comprimido (extensión **.zip**) que contiene la estructura de carpetas y ficheros necesarios para compilar y ejecutar el programa.

El proyecto incluye en su directorio raíz un fichero **compile.bat** que compila el código fuente al ser ejecutado desde una ventana de Símbolo de Sistema. Para compilar el código fuente es necesario que el JDK esté instalado y configurado en el sistema.

```
C:\the-one-master-1.6.0-21e086d>compile.bat

C:\the-one-master-1.6.0-21e086d>set targetdir=target

C:\the-one-master-1.6.0-21e086d>IF NOT EXIST "target" mkdir target

C:\the-one-master-1.6.0-21e086d>javac -sourcepath src -d target -extdirs lib/ src/core/*.java src/movement/*.java src/report/*.java
src/routing/*.java src/gui/*.java src/input/*.java src/applications/*.java src/interfaces/*.java

C:\the-one-master-1.6.0-21e086d>IF NOT EXIST "target\gui\buttonGraphics" (
mkdir target\gui\buttonGraphics
 copy src\gui\buttonGraphics\* target\gui\buttonGraphics\
)

C:\the-one-master-1.6.0-21e086d>
```

Además, el proyecto trae por defecto un escenario configurado en el fichero **default_settings.txt** que permite ejecutar con éxito el simulador, sin necesidad de realizar ninguna configuración, con tan sólo ejecutar el fichero **one.bat** desde una ventana de Símbolo de Sistema.

El aspecto del interfaz gráfico del simulador ONE es el mostrado en la Figura 3-1. En [6] se recoge una descripción del contenido mostrado en cada parte de la ventana y manejo de las opciones básicas.

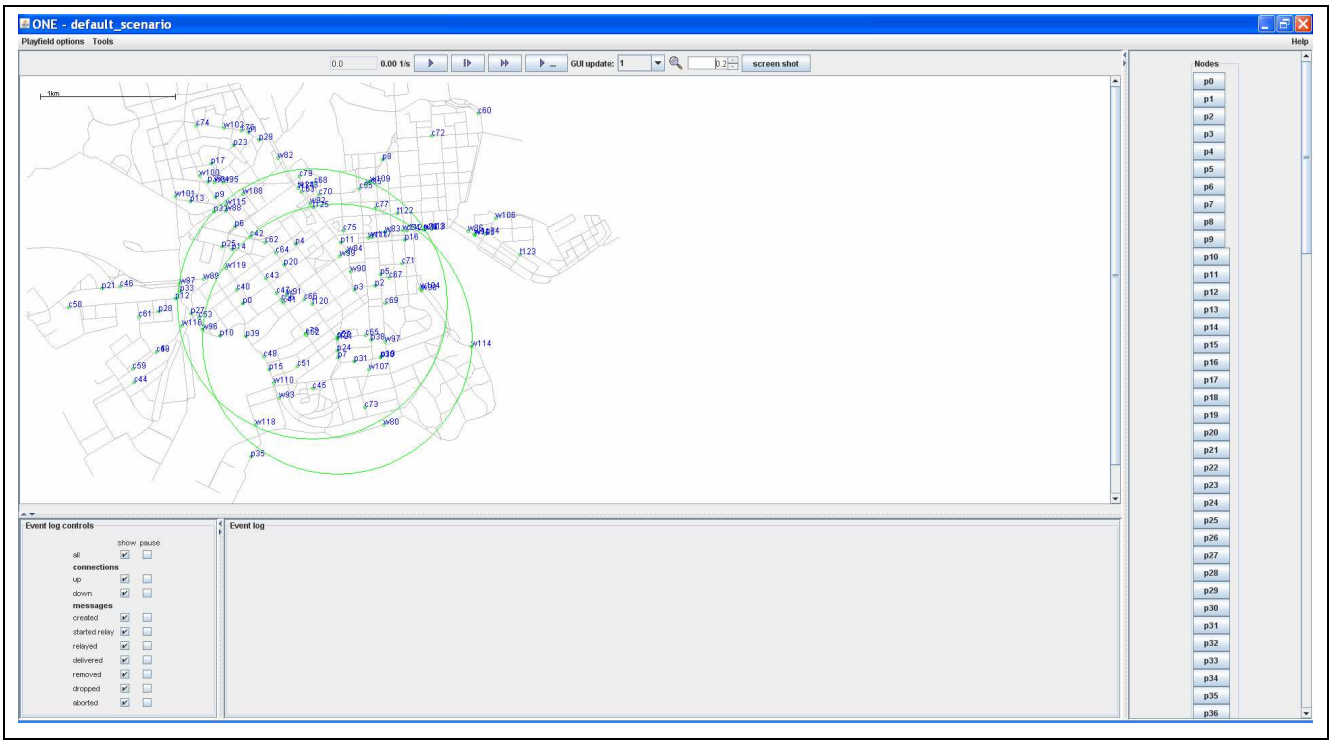

Figura 3-1 Interfaz gráfico del simulador ONE

DWinPerl:

Para instalar el software DWinPerl simplemente hay que seguir las indicaciones facilitadas por el asistente de instalación. Para lograr un correcto funcionamiento con los *scripts* PERL incluidos en una de las carpetas del proyecto del simulador ONE es suficiente seleccionar la ruta de instalación deseada para el software y aceptar las opciones ofrecidas por defecto por el asistente.

El asistente de instalación añade las variables de entorno para que el software funcione correctamente.

3.4 Adaptación del simulador y generación de ficheros

3.4.1 Adaptación del código fuente del simulador

El simulador ONE es plenamente operativo y, las características implementadas funcionan correctamente. No obstante, es un proyecto que continúa en desarrollo para adaptar y extender las características y opciones disponibles actualmente.

Como se ha indicado al principio de este capítulo, el simulador ONE está desarrollado en Java, por lo que se basa en un estilo de programación orientado a objetos.

Como ya se expuso en el Capítulo 2, una de las características de las comunicaciones por radio en entornos militares tácticos es la reducción de la cobertura nominal de los equipos de comunicaciones debido a diversas circunstancias, e incluso la interrupción de las comunicaciones. Considerando las características de los interfaces de radio utilizados en este tipo de escenarios (el radio de cobertura nominal es mayor que la distancia a la que pueden llegar a estar los nodos), y teniendo en cuenta el funcionamiento del simulador ONE (que no incorpora opciones para simular desvanecimientos o interrupciones de la comunicación), se comprobó que, para realizar simulaciones que se asemejasen a las circunstancias reales, era necesario modificar el código fuente del programa para que considerase situaciones de disminución de la cobertura y de interrupción de las comunicaciones.

El primer paso para conseguir implementar esta funcionalidad fue estudiar los diferentes ficheros con el código del programa y de las clases que lo forman para identificar el punto en el que se asociaba el parámetro ***interfaz*.transmitRange** del fichero de configuración **default_settings.txt** a un atributo de una clase, y cuál era el uso que hacían de el los diferentes métodos.

Tras estudiar el código del programa, se comprobó que el fichero **NetworkInterface.java**, que contiene el código de la clase **NetworkInterface** era el lugar donde debía incluirse la nueva porción de código que implementase la funcionalidad requerida. En concreto, el método cuyo código se ha modificado para soportar esta nueva funcionalidad es **getTransmitRange**.

El Anexo I contiene el código del fichero **NetworkInterface.java** después de la modificación. En el se ha resaltado el código añadido para implementar la nueva funcionalidad.

El uso de esta funcionalidad se lleva a cabo desde el fichero de configuración de la simulación, mediante la definición de los valores de los atributos que se han creado y de otro existente en el código original:

- Simulación de la reducción de la cobertura nominal definida para cada tipo de interfaz: se realiza en base a dos parámetros que, según sus valores, pueden simular diferentes tipos de terrenos:
  - ***interfaz*.transmitRangeDegradadoRatioUbicaciones**: especifica el tanto por ciento de ubicaciones en las que los nodos verán reducida la cobertura nominal de ese tipo de interfaz.
  - ***interfaz*.transmitRangeDegradadoRatioMaximo**: especifica el tanto por ciento en el que se verá reducida la cobertura nominal de ese tipo de interfaz en los casos en los que esta se produce.
- Simulación de la pérdida de cobertura para los interfaces de los nodos de un grupo: se lleva a cabo haciendo uso del parámetro ***group*.activeTimes** de forma que se considera que, cuando un nodo se queda sin cobertura, es indiferente que se mueva, hasta el momento en que la recupera. De esta manera, en el escenario simulado se puede definir en qué momentos los nodos pierden la conectividad con los nodos de otros grupos, aunque no necesariamente con los del propio grupo, ya que se encontrarían en una posición muy próxima.

El código original del simulador no realiza cálculo alguno para calcular la cobertura para cada interfaz de cada nodo en cada ciclo de simulación, ya que utiliza valores constantes, sin embargo el nuevo código sí debe hacerlo, lo que conlleva una mayor carga computacional, que da lugar a ralentización de 1:6 en la ejecución de las simulaciones.

3.4.2 Elaboración del fichero de configuración del escenario

La definición del escenario objeto de la simulación se toma, por defecto, del fichero **default_settings.txt**, si bien puede especificarse otro fichero al lanzar la ejecución del simulador.

Por sencillez, la configuración del escenario utilizado para las simulaciones realizadas en este TFM se ha llevado a cabo en el fichero **default_settings.txt**, cuyo contenido se puede ver en el Anexo II.

El escenario recreado, en la medida que el simulador lo permite, se asemeja a una columna de transporte en zona de operaciones compuesta por varios vehículos de transporte de tropas y material, un vehículo de transmisiones, un vehículo de puesto de mando, un vehículo con equipos ligeros de guerra electrónica, un vehículo de zapadores y un vehículo de exploración de caballería. Si bien el *layout* mostrado en el simulador no muestra el avance de los nodos al ritmo de una columna de transporte, el modelo es válido porque los movimientos de los nodos de los diferentes grupos representan el movimiento relativo de unos nodos respecto a otros, que es lo relevante en la simulación, bien a la velocidad de la columna de transporte, bien en paradas que fuera necesario realizar por las circunstancias encontradas en la ruta. Además, la temporización de la activación y

desactivación de los diferentes grupos de nodos, simula 3 detenciones de la columna en la que se despliegan soldados a pie. En la primera detención se activan 3 grupos de nodos: uno, por su patrón de movimiento, simula ser un UAV, mientras que los otros dos grupos podrían ser soldados de caballería y zapadores. En las dos siguientes detenciones también se activan 3 grupos de nodos: de nuevo el UAV, y otros dos grupos que podrían ser soldados de caballería y equipos ligeros de guerra electrónica.

Salvo el nodo que simula ser el vehículo de puesto de mando, que se ha configurado en modo estático al considerarse la referencia para el movimiento relativo del resto de grupos (a excepción del UAV), estos grupos se han configurado con un modelo de movimiento de tipo **ClusterMovement**, en el que los nodos se mueven dentro de un área circular específico. El UAV se ha configurado con un modelo de movimiento **RandomWaypoint** por todo el escenario.

La Figura 3-2 muestra la distribución de los grupos de nodos y las zonas de movilidad de cada uno de los nodos dentro del área definida para su grupo.

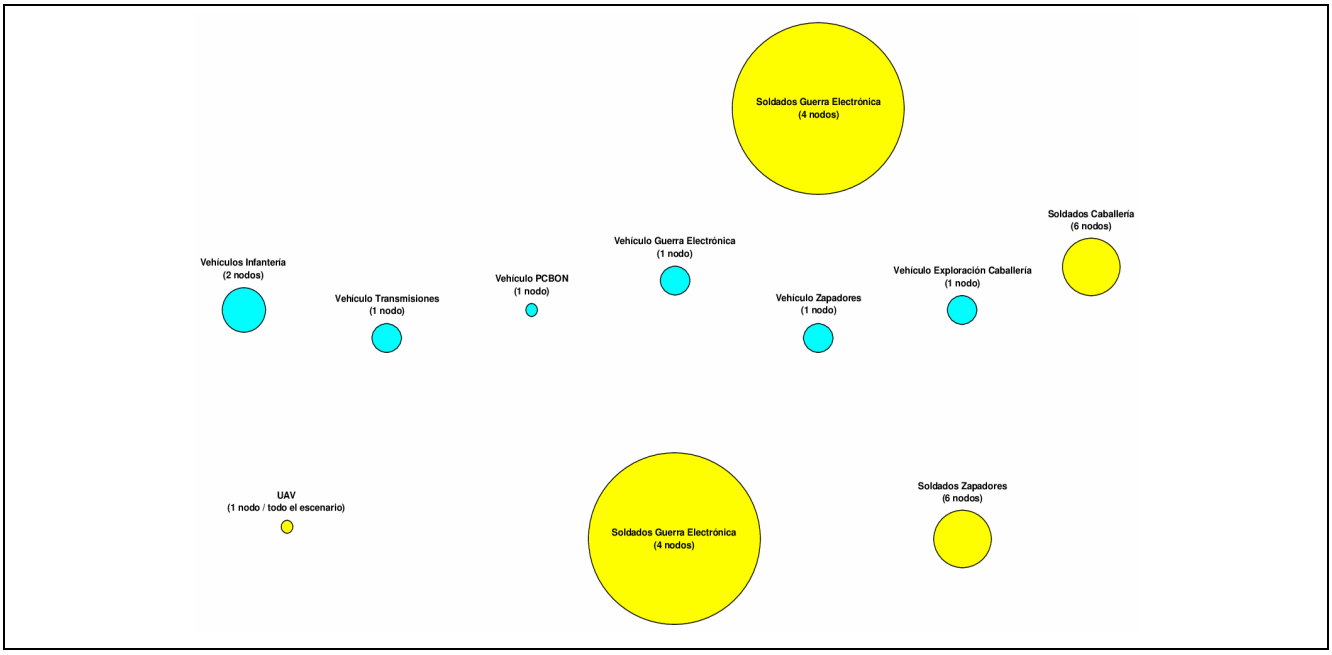

Figura 3-2 Representación de la posición relativa de los grupos de nodos en el escenario simulado. En azul las zonas de movilidad de los vehículos de los diferentes grupos. En amarillo la zona de movilidad de los soldados durante las detenciones de la columna

El fichero de configuración define dos interfaces de radio que se distinguen en el radio de cobertura nominal, asemejándose a radios de tipo vehicular con una cobertura de 7000m para el ancho de banda binario considerado, y portátil con una cobertura de 1000m para el mismo ancho de banda.

Los nodos móviles, que se corresponden con soldados a pie, estarían equipados con equipos de radio portátil, mientras que los vehículos, según su tipología, dispondrían sólo de la radio de tipo vehicular, o bien de ambas.

El simulador simplifica la gestión del tráfico, permitiendo la transmisión de un paquete por un tipo de interfaz y después por otro, de manera similar a como gestionaría el tráfico un gestor de comunicaciones.

Además de las reducciones de cobertura de los interfaces según los porcentajes definidos con los parámetros creados para este fin en el nuevo código fuente, en el Anexo II se puede comprobar que se han definido multitud de intervalos de tiempo con pérdidas de cobertura de todos los nodos de los diferentes grupos.

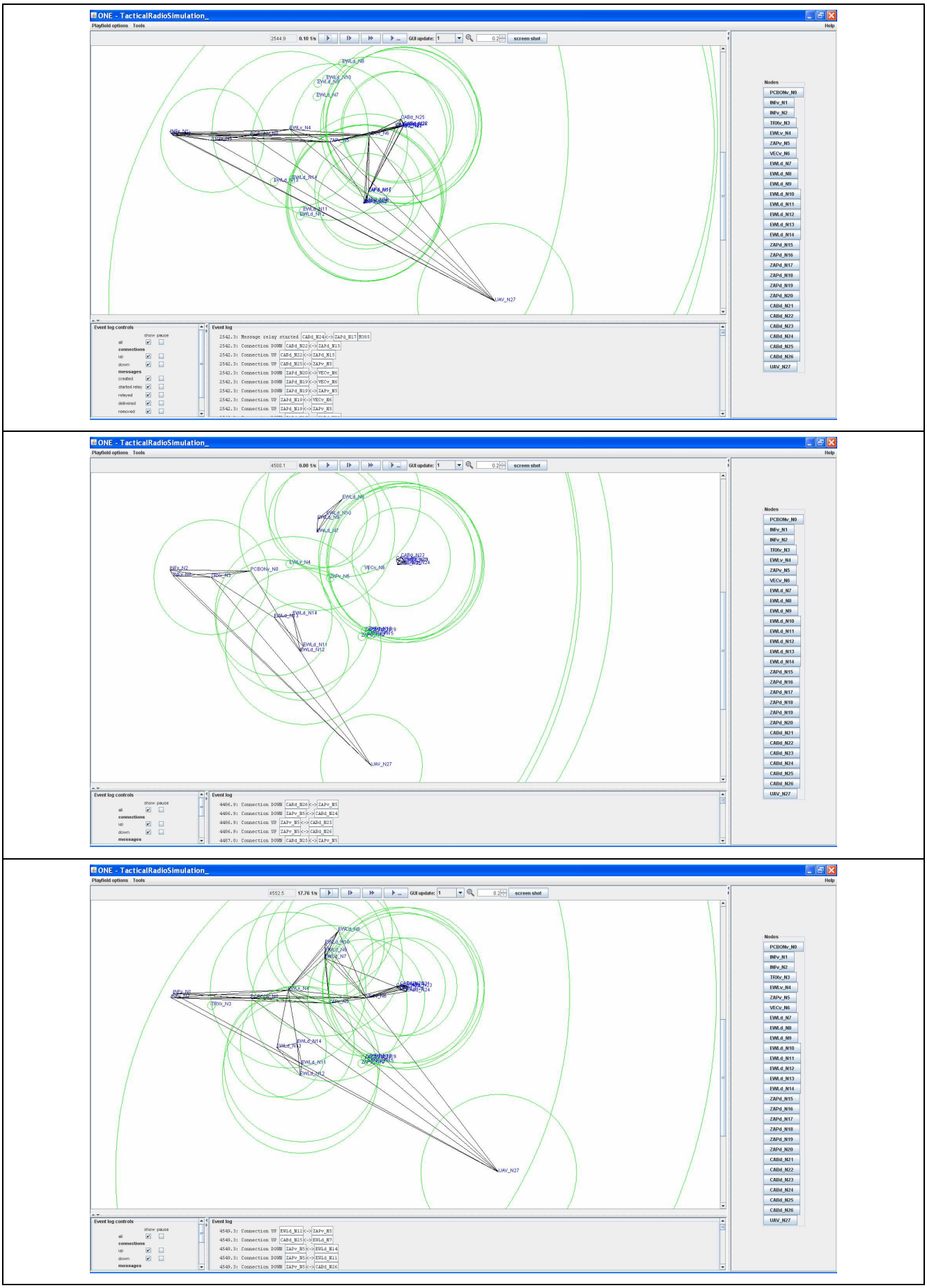

Figura 3-3 Capturas de pantalla de diferentes instantes de simulaciones realizadas con el interfaz gráfico

Para comprobar el comportamiento del sistema en diferentes situaciones de tráfico, en el fichero de configuración se ha especificado que el tráfico utilizado en las simulaciones no sea generado aleatoriamente por el simulador, si no que sea tomado de ficheros creados ex profeso para cada perfil objeto de estudio.

La simulación se realiza sobre un tiempo total de 11100 segundos, de los que los 300 primeros segundos se consideran para entrenamiento de los protocolos de enrutamiento de los nodos, por lo que en este primer periodo no se genera tráfico.

La Figura 3-3 muestra diferentes instantes de una de las simulaciones realizadas con el interfaz gráfico.

3.4.3 Generación de ficheros de tráfico

El simulador ONE soporta que el tráfico inyectado en el escenario para realizar las simulaciones provenga de diversas fuentes. Si bien durante la realización de este TFM se han utilizado las funcionalidades de simulador automático de eventos y de generación de mensajes definidos en un fichero, el simulador también soporta la inyección de tráfico proveniente de otros simuladores y aplicaciones mediante el uso de una API [6].

Inicialmente se utilizó el generador automático de eventos, (***Events*.class = MessageEventGenerator**). Aunque este generador de mensajes permite definir el rango de tamaños de mensajes que se van a enviar y el rango de segundos en que se irán generando esos mensajes, se comprobó que no era una solución válida para las simulaciones que se iban a realizar por dos motivos: en primer lugar, este generador de mensajes envía mensajes tomando como origen y destino nodos aleatorios de entre todos los disponibles en la simulación, lo cual no es un perfil de tráfico que se corresponda con el que tiene lugar en los escenarios objeto de estudio; en segundo lugar, en este generador no se pueden definir intervalos de tiempo en los que crear mensajes entre nodos específicos, de forma que se simule el tráfico generado en las paradas en las que se despliegan los soldados a pie.

Debido a lo anterior, se optó por generar los mensajes de las simulaciones a partir de un fichero previamente creado (***Events*.class =ExternalEventsQueue**). De este modo ha sido posible especificar patrones de envío de mensajes definiendo:

- Origen y destino de los mensajes en base a un perfil que simula el de este tipo de escenarios.
- La temporización de cada perfil de mensajes entre nodos del mismo grupo y de diferentes grupos, de forma que se corresponda con el comportamiento simulado en el escenario (detenciones, despliegue de los diferentes grupos de nodos que simulan soldados a pie, pérdidas de cobertura asociadas al avance de los vehículos).
- La cadencia en la creación de los mensajes.
- El tamaño de los mensajes.

Para la generación de estos ficheros se ha utilizado el *script* **createCreates.pl** escrito en Perl que están incluidos en el proyecto ONE. Este *script* permite definir los siguientes parámetros en base a los que se genera el fichero que, desde el simulador ONE, se utiliza para generar los mensajes de la simulación:

- Los tiempos de inicio y final del intervalo de simulación en el que se van a generar los mensajes.
- El número de mensajes que se van a generar en ese intervalo.
- El rango de *host* entre los que se van a generar los mensajes con una distribución aleatoria.
- El rango del tamaño de los mensajes.

- El rango del tamaño de las respuestas, si las hubiera.
- Un número que se utiliza como semilla para la generación pseudoaleatoria de los mensajes.

```
C:\the-one-master-1.6.0-21e086d\toolkit> perl createCreates.pl -time 300:6500 -nrof 500 -hosts 0:6 -sizes 900:1100 -rsizes 20:40 -seed 4 > trafico.txt
```

Para cada uno de los intervalos medios de tiempo de creación de mensajes utilizados en las simulaciones, el proceso seguido para crear los ficheros ha sido el siguiente:

1. Crear un fichero de tráfico para cada intervalo de interés (distinguiendo los periodos en los que se simula el despliegue de soldados a pie), con el intervalo medio de tiempo de creación de mensajes y el tamaño de mensajes deseados.
2. Editar cada uno de los ficheros creados para definir el envío de mensajes entre nodos con el patrón deseado de orígenes y destinos. Este paso es necesario porque el fichero generado por el *script* pone nodos de origen y destino de forma aleatoria, y este perfil de tráfico no se ajusta al del escenario simulado. Puesto que el fichero generado por el *script* tiene sus campos separados por tabuladores, esta edición se ha llevado a cabo con Microsoft Excel para, una vez definidos diversos patrones para una serie de mensajes, poder replicarlos por bloques para el resto de mensajes del fichero.
3. Concatenar el contenido de los ficheros de tráfico de los diferentes intervalos en el orden adecuado.
4. Renumerar de forma consecutiva los mensajes del fichero resultante en el paso anterior.

Para llevar a cabo las simulaciones deseadas, se han creado 16 ficheros de tráfico con las combinaciones de 4 valores de intervalo medio de tiempo de creación de mensajes y 4 tamaños de mensaje.

El Anexo III contiene, a modo de ejemplo, un fragmento de uno de los ficheros de tráfico utilizado en las simulaciones.

3.5 Comportamiento de redes convencionales

Según se ha indicado anteriormente, los protocolos de comunicaciones como TCP/UDP sobre IP, por sí solos, limitan o incluso impiden el funcionamiento de aplicaciones convencionales en entornos con características de conectividad como las descritas en el Capítulo 1 (ancho de banda reducido, pérdidas de la conectividad entre nodos, latencia elevada, entre otras). Los entornos militares tácticos son un ejemplo del tipo de escenarios donde las tecnologías convencionales no funcionarían correctamente.

El escenario definido para estudiar la viabilidad del uso de las DTNs mediante el simulador ONE ha sido configurado para asemejarse a una situación real con las características de ancho de banda de equipamiento de radio táctica existente en la actualidad y en circunstancias de conectividad degradadas. La degradación de las condiciones de conectividad entre los nodos del escenario se implementa haciendo uso de los 2 recursos desarrollados en la adaptación del código fuente del simulador:

- Pérdida de cobertura para los interfaces de los nodos de un grupo: como se explicó en un epígrafe anterior, se implementa haciendo uso del parámetro ***group*.activeTimes**. Con este recurso se especifica, mediante la definición de intervalos de segundos, los momentos en que los nodos de un grupo pierden la conectividad con los nodos de otros grupos, aunque no necesariamente con los del propio grupo, ya que se encontrarían en una posición muy próxima. Esto se asemejaría a pérdidas de conectividad debidas a la orografía o a efectos prolongados de *jamming*, y se han definido en intervalos entre los 10 y los 50 segundos. La Figura 3-4 muestra las desconexiones introducidas con este recurso para 3 periodos de la simulación.
- Reducción de la cobertura nominal definida para cada tipo de interfaz: para los periodos en los que los nodos de un grupo están en una situación de actividad, la cobertura de cada interfaz de cada nodo se puede ver reducida según una distribución pseudoaleatoria obtenida a partir de los parámetros ***interfaz*.transmitRangeDegradadoRatioUbicaciones** e ***interfaz*.transmitRangeDegradadoRatioMaximo**. Esta reducción de cobertura puede, según la distancia entre los nodos, dar lugar a pérdidas de conectividad de corta duración entre los mismos.

Considerando las pérdidas de conectividad introducidas por cada uno de estos recursos, las tecnologías de red convencionales verían su funcionamiento muy limitado, o incluso imposibilitado, por los siguientes motivos:

- Pérdida de conectividad durante la transmisión de los mensajes en alguno de los contactos entre nodos de la ruta entre origen y destino: puede afectar de 2 formas diferentes:
  - Si el tamaño de los paquetes es elevado, debido a que el ancho de banda disponible es muy reducido, aumenta la probabilidad de que se produzca una pérdida de contacto en alguno de los saltos de la ruta, lo que da lugar a un elevado número de intentos de retransmisión.
  - Si el tamaño de los paquetes es pequeño, además de un incremento del *overhead* de datos de control, el número de paquetes en la red se multiplica, y también lo haría el número de paquetes con error de transmisión, lo que provocaría el incremento de intentos de retransmisión.
- Indisponibilidad de rutas activas entre origen y destino al intentar enviar los mensajes: este problema esta causado principalmente por las pérdidas prolongadas de conectividad (una gran parte de entre 30 y 50 segundos) entre los nodos involucrados en una ruta, lo cual, además de incrementar el tráfico de control de los protocolos de enrutamiento para actualizar el estado de las rutas, impediría la transmisión de los paquetes, ya que el nodo de destino se consideraría inaccesible.
- En el caso de utilizar TCP puro (sin implementar la tecnología DTN) como protocolo de nivel de transporte, los 2 problemas anteriores, además, tendrían como consecuencia el vencimiento de temporizadores utilizados por el protocolo. Los temporizadores más afectados serían los siguientes [28]:
  - Temporizador de retransmisión: utilizado para medir el tiempo que el origen espera el ACK (*acknowledgement*) del destinatario con la confirmación de la recepción por este de un paquete con un número de secuencia. En caso de vencer, el origen retransmite los paquetes.
  - Temporizador de persistencia: utilizado para recuperar situaciones en las que se pierde el paquete con el que el extremo que había cerrado la ventana de transmisión, la vuelve a abrir.
  - Temporizador 2MSL (*Maximum Segment Lifetime*): utilizado en el mecanismo de cierre activo de una conexión TCP para recuperarse de la pérdida de alguno de los paquetes del

proceso. El tiempo que los extremos deben estar en el estado **TIME_WAIT** es el doble del valor MSL.

- Temporizador *keepalive*: utilizado para comprobar si en una conexión TCP establecida en la que los extremos no están recibiendo paquetes, los otros extremos siguen activos.

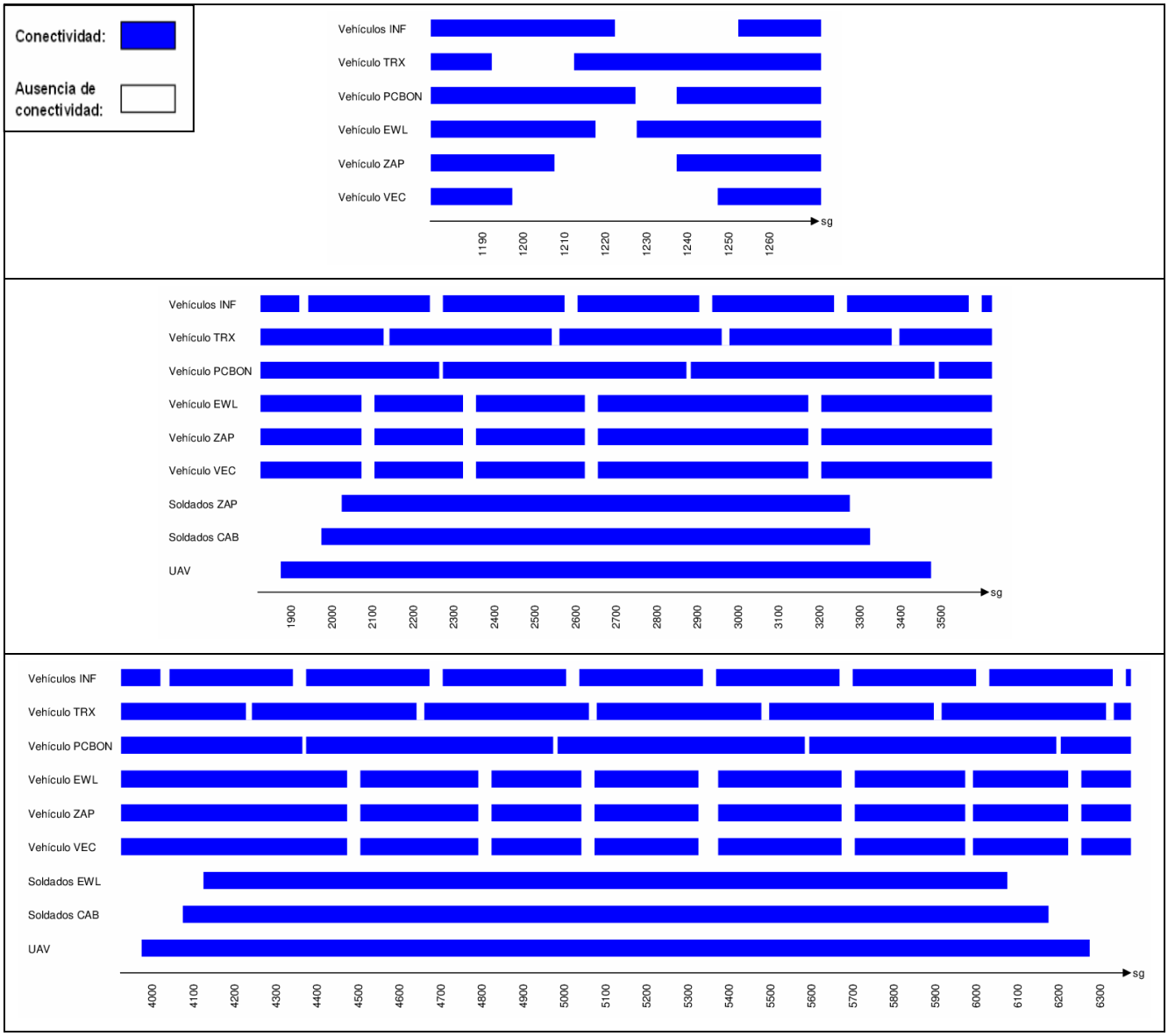

Figura 3-4 Representación de las situaciones de desconexión definidas en el fichero de configuración default_settings.txt para cada grupo de nodos. No incluye desconexiones por reducciones de cobertura. (Arriba: un intervalo de la situación que simula el avance de la columna. Medio: el intervalo de la primera parada. Abajo: el intervalo de la segunda parada)

3.6 Ejecución de las simulaciones

La parte práctica de este TFM tiene como objetivo comprobar el funcionamiento del escenario táctico simulado con una variedad de perfiles de tráfico y de protocolos de enrutamiento.

El conjunto de simulaciones se ha ejecutado combinando los siguientes parámetros:

- Protocolo de enrutamiento: el estudio se ha realizado con los siguientes protocolos:

  - Epidemic.
  - PRoPHET, considerando unidades de tiempo de 30 segundos.
  - *Spray&Wait*, en modo binario, con 6 copias por mensaje.

- TTL de los mensajes: el estudio se ha realizado con los valores 1, 3 y 5 minutos. Debido a un *bug* del simulador asociado al modo en que se van ejecutando las simulaciones dentro de un lote que, en el caso de que el número de valores de un parámetro sea múltiplo de los de otro parámetro, provoca que ciertas simulaciones se repitan y otras no se ejecuten, ha sido necesario añadir un valor más de TTL (2 minutos), aunque los resultados de estas simulaciones no se han considerado en el análisis de los datos realizado en el siguiente capítulo.
- Tamaño de los mensajes: el estudio se ha realizado con mensajes de tamaños medios de 100B, 1KB, 10KB y 100KB. Con la finalidad de dotar a las simulaciones de mayor realismo, en vez de utilizar mensajes de tamaño constante, para cada uno de los cuatro tamaños objeto de estudio, el *script* **createCreates.pl** se ha configurado para generar mensajes con valores variables en un rango de ±10%.
- Intervalo medio de tiempo de creación de mensajes: el estudio se ha realizado creando mensajes cada 1, 5, 10 y 15 segundos. La creación de mensajes por el *script* **createCreates.pl** se lleva a cabo de forma que los mensajes se generan en instantes de tiempo aleatorios con una distribución final que arroja un valor medio según los valores definidos al ejecutar el *script*.

Las variables que se han combinado en cada ejecución por lotes han sido el protocolo de enrutamiento y el TTL de los mensajes, resultando un total de 12 combinaciones. De esta forma, para cada uno de los 16 ficheros de tráfico (caracterizados por un intervalo medio de tiempo de creación de mensajes y un tamaño de mensaje, y cargados en el escenario mediante el parámetro ***Events*.filePath=*ruta_fichero*/*nombre_fichero.txt*** del fichero de configuración), se ha ejecutado un proceso por lotes mediante el siguiente comando:

```
C:\the-one-master-1.6.0-21e086d> one.bat -b 12 default_settings.txt
```

La combinación de las 12 simulaciones de cada lote, con los 16 ficheros de tráfico, ha resultado en un total de 192 simulaciones.

(Página dejada intencionadamente en blanco)

4 RESULTADOS DE LAS SIMULACIONES

4.1 Simulaciones realizadas

En el capítulo anterior se ha explicado cuales han sido los pasos para:

1. Instalar y configurar el software necesario para realizar las simulaciones deseadas.
2. Adaptar el comportamiento del simulador ONE.
3. Definir el escenario objeto de simulación y los perfiles de tráfico.
4. Simular las diferentes situaciones objeto de estudio mediante la ejecución de lotes de simulaciones.

La parte práctica de este TFM se ha llevado a cabo en una estación de trabajo con las siguientes características:

- Procesador: Intel Core i7-6700HQ (4 núcleos, 8 subprocesos), a 2,60GHz.
- RAM: 16GB.
- Almacenamiento: HDD de estado sólido.
- Sistema operativo: Microsoft Windows 10 Home.

Dedicando este equipo exclusivamente a la realización de las 192 simulaciones ya descritas, se ha requerido un tiempo de ejecución superior a 4 horas y media. El resultado de la compilación del código fuente sólo ha sido capaz de extraer un rendimiento de entre el 17,0% y el 17,5% de la capacidad del procesador.

Como ya se indicó en el capítulo anterior, las simulaciones asociadas al parámetro TTL=2 no se incluirán en el estudio de este capítulo por resultar irrelevantes, pero fue necesario realizarlas para evitar un *bug* del simulador. Por lo que el número final de simulaciones analizadas es de 144.

De todos los ficheros generados en cada simulación, la principal fuente de datos utilizada para ser analizada en este capítulo, y obtener los resultados de esta parte práctica, ha sido el contenido del reporte **MessageStatsReport** que contiene la siguiente información:

```
Message stats for scenario TacticalRadioSimulation__GroupRouter-EpidemicRouter__GroupMsgttl-2__

sim_time: 11100.0000
```

```
created: 1668
started: 22632
relayed: 22508
aborted: 124
dropped: 21649
removed: 0
delivered: 1668

delivery_prob: 1.0000
response_prob: 0.0000

overhead_ratio: 12.4940

latency_avg: 1.3053
latency_med: 0.1000

hopcount_avg: 1.0683
hopcount_med: 1

buffertime_avg: 88.3026
buffertime_med: 88.8000

rtt_avg: NaN
rtt_med: NaN
```

Como se puede observar, el reporte **MessageStatsReport** contiene información de tipo estadístico sobre la ejecución de la simulación. El resto de reportes contienen información detallada de los diferentes eventos ocurridos durante la simulación (establecimientos y pérdidas de contactos entre nodos, creación y transmisión entre nodos de cada uno de los mensajes, etc.). El tamaño del conjunto de reportes generados en cada una de las 144 simulaciones es de aproximadamente 100MB.

4.2 Resultados obtenidos

La cantidad de información obtenida de las 144 simulaciones analizadas es ingente. De toda esta información, los datos cuyo análisis aporta la mejor información para el objeto de este TFM, son los valores de los parámetros **delivery_prob**, **overhead_ratio**, **latency_avg**, y **latency_med** del reporte **MessageStatsReport**.

Para cada uno de estos parámetros se analizará el comportamiento del escenario en las simulaciones realizadas con las diferentes combinaciones de las variables anteriormente descritas:

- Protocolo de enrutamiento: para los protocolos Epidemic, PRoPHET (considerando unidades de tiempo de 30 segundos) y *Spray&Wait* (en modo binario, con 6 copias por mensaje).
- TTL de los mensajes: con los valores 1, 3 y 5 minutos.
- Tamaño de los mensajes: con los valores medios 100B, 1KB, 10KB y 100KB.
- Intervalo medio de tiempo de creación de mensajes: creando mensajes cada 1, 5, 10 y 15 segundos.

El objetivo es representar el valor de cada uno de los 4 parámetros relevantes del reporte **MessageStatsReport** en función de estas 4 variables para, posteriormente analizar el comportamiento en función de las mismas y, finalmente, obtener conclusiones que permitan definir las condiciones que maximizan el rendimiento de la red definida en el escenario.

La solución adoptada para realizar estas representaciones consiste en, para cada parámetro, generar 3 gráficas (una por cada valor de la variable TTL) y, en cada gráfica, representar el valor del parámetro objeto de estudio en función de las otras 3 variables (protocolo de enrutamiento, tamaño medio de mensaje e intervalo medio de tiempo de creación de mensajes), agrupando la representación de estos valores en función del intervalo medio de tiempo de creación de mensajes.

*4.2.1 Parámetro "**delivery_prob**"*

El parámetro **delivery_prob** del fichero **MessageStatsReport** contiene la probabilidad de entrega satisfactoria de los mensajes creados en la simulación realizada. Es decir, arroja un valor cuantitativo que define la fiabilidad de la red en cuanto a la capacidad de entrega a los destinatarios de los mensajes creados por los emisores.

Como se observa en las gráficas de la Figura 4-1, de las simulaciones realizadas, se extrae que, respecto a las variables consideradas, la probabilidad de entrega se comporta de la siguiente manera:

- El protocolo de enrutamiento utilizado no tiene una incidencia significativa.
- El TTL de los mensajes tiene un impacto apreciable, pero no excesivo.
- El intervalo entre mensajes sí tiene un impacto importante.
- El tamaño de mensaje es la variable que más afecta a la probabilidad de entrega.

Un ejemplo que pone de manifiesto la dependencia de la probabilidad de entrega respecto del tamaño de mensaje es que, observando el comportamiento de la red para cada protocolo de enrutamiento dentro de cada agrupación de las variables tamaño de paquete, intervalo entre mensajes y TTL, se comprueba que la probabilidad de entrega puede tener variaciones que van del 98,05% para 100B con [Epidemic, 1sg/msg, TTL=1] a 9,81% para 100KB con [Epidemic, 1sg/msg, TTL=1]. Este comportamiento, aunque en menor medida, se repite para el resto de protocolos de enrutamiento.

Para mensajes de tamaño igual a 10KB, cuando el intervalo entre mensajes es de 5sg/msg o superior, la probabilidad de entrega es superior al 86%. Mientras que para tamaños de mensaje de 100KB, la probabilidad de entrega es inferior al 75% para la mayor parte de los casos (salvo TTL=5 e intervalos entre mensajes de 15sg/msg.

Se observa que, de forma generalizada, mientras el tamaño del mensaje sea igual o inferior a 1KB, las probabilidades de entrega son iguales o cercanas al 100% para cualquiera de los valores de intervalo entre mensajes, TTL de los mensajes y protocolo de enrutamiento utilizados en la simulación.

Así mismo, se observa que, para los diferentes valores de TTL de mensajes y de protocolos de enrutamiento, cuando el tamaño de mensaje es igual a 10KB, se produce un incremento relevante de la probabilidad de entrega al pasar de 1sg/msg a 5sg/msg.

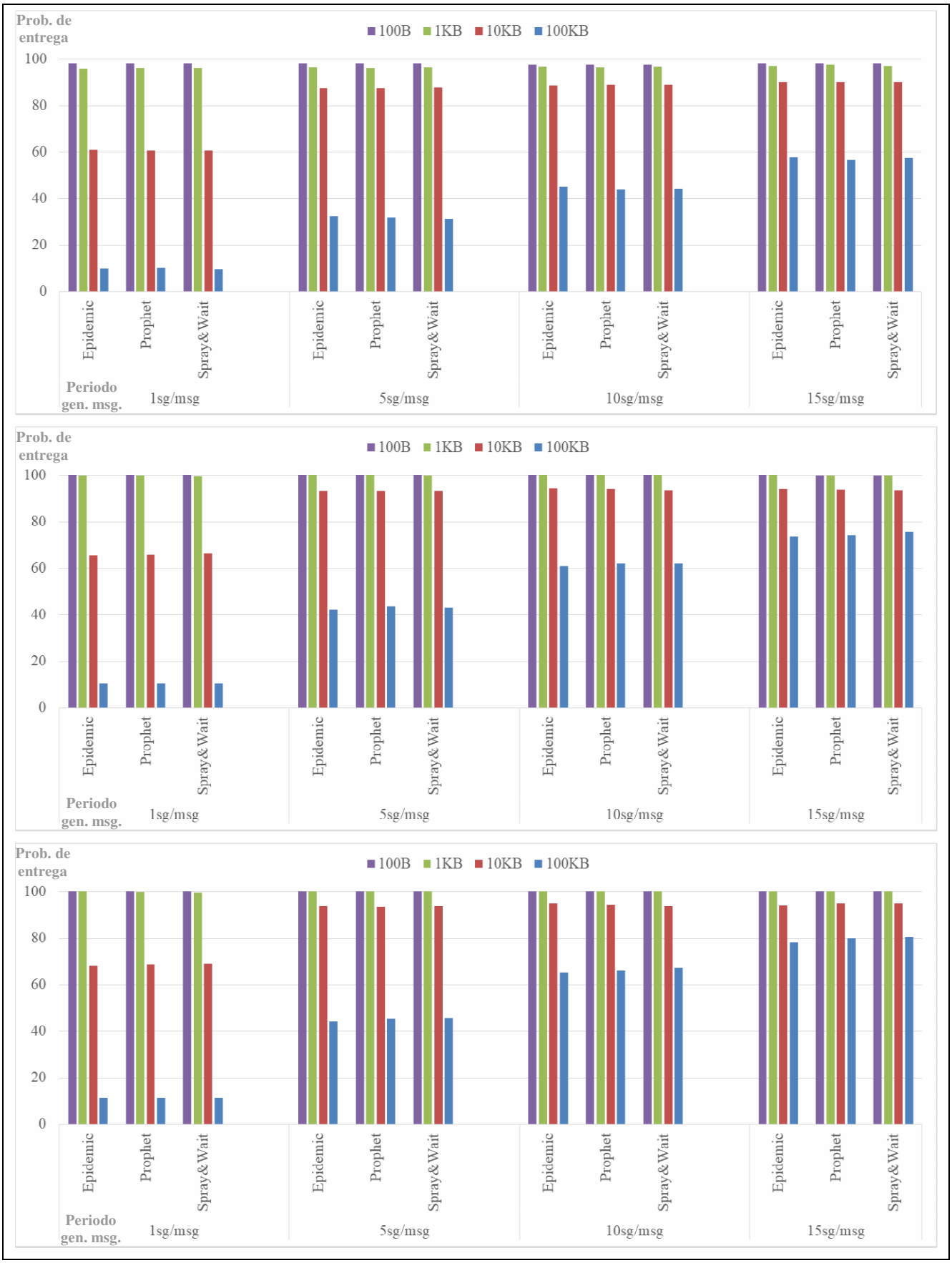

Figura 4-1 Representación del parámetro delivery_prob en función de las 4 variables consideradas en las simulaciones (arriba TTL=1, medio TTL=3, abajo TTL=5)

*4.2.2 Parámetro "**overhead_ratio**"*

El parámetro **overhead_ratio** del fichero **MessageStatsReport** expresa cuanto se sobrecarga la red con mensajes clonados por los nodos según el protocolo de enrutamiento utilizado.

El valor del parámetro **overhead_ratio** responde a la siguiente ecuación:

$$overhead\_ratio = \frac{relayed - delivered}{delivered}$$

donde: - **relayed**: es el número total de mensajes transmitidos por la red.

- **delivered**: es el número de mensajes entregados al destinatario.

Como se observa en las gráficas de la Figura 4-2, de las simulaciones realizadas, se extrae que, respecto a las variables consideradas, la sobrecarga de la red se comporta de la siguiente manera:

- El intervalo entre mensajes no tiene, en general, una incidencia significativa.
- El protocolo de enrutamiento utilizado tiene un impacto importante cuando, para los protocolos de tipo inundación, la sobrecarga de red aumenta en un orden de magnitud.
- El tamaño de mensaje también tiene un impacto muy importante.
- El TTL de los mensajes se pone de manifiesto cuando su valor es mayor que un umbral para el que, según las condiciones de la red (ancho de banda, intervalos de pérdida de contacto entre los nodos, tamaño de mensajes e intervalo entre mensajes), los mensajes comienzan a propagarse entre los nodos. En el caso de las simulaciones realizadas este valor es TTL=3.

Como se puede comprobar, para TTL=1 la sobrecarga en la red es prácticamente inexistente, mientras que para valores iguales o superiores a TTL=3, la sobrecarga se ve incrementada aproximadamente en dos órdenes de magnitud para los mensajes de tamaño igual o inferior a 10KB.

El tamaño de los mensajes tiene un impacto importante, de forma que, cuando el tamaño del mensaje es igual o inferior a 10KB, se facilita su transmisión en los intervalos de contacto entre nodos y, en caso de que el protocolo de enrutamiento no contenga el incremento de mensajes clonados entre nodos, la sobrecarga de la red se ve fuertemente incrementada.

El protocolo de enrutamiento utilizado tiene una incidencia clara en la sobrecarga de la red. En el caso de protocolos de inundación puros, para tamaños de mensaje de 10KB o inferiores, el número de mensajes clonados se ve fuertemente incrementado, por lo que la sobrecarga de la red lo hace en la misma proporción. En estas circunstancias es cuando protocolos como *Spray&Wait*, con limitación del número de copias que se generan de los mensajes ponen de manifiesto su rendimiento.

Figura 4-2 Representación del parámetro overhead_ratio en función de las 4 variables consideradas en las simulaciones (arriba TTL=1, medio TTL=3, abajo TTL=5)

*4.2.3 Parámetro "**latency_med**"*

El parámetro **latency_med** del fichero **MessageStatsReport** contiene la mediana[10] de los valores de latencia de los mensajes recibidos por los destinatarios.

Como se observa en las gráficas de la Figura 4-3, de las simulaciones realizadas, se extrae que, respecto a las variables consideradas, la mediana de la latencia de los mensajes se comporta de la siguiente manera:

- El intervalo entre mensajes no tiene, en general, una incidencia significativa.
- El protocolo de enrutamiento utilizado tiene un impacto apreciable, pero no excesivo.
- El TTL de los mensajes tiene un impacto importante.
- El tamaño de mensaje, para valores superiores a 1KB, es la variable que más afecta.

Como ya se ha puesto de manifiesto en el epígrafe anterior, el tamaño de los mensajes tiene un impacto importante. Para este parámetro se observa que, cuando el tamaño del mensaje es igual o inferior a 1KB, la mediana de la latencia arroja valores iguales o inferiores a 0,5 segundos en todas las simulaciones, mientras que para mensajes de tamaño igual a 10KB la mediana de la latencia toma valores a partir de 2,2 segundos. Como se puede observar, para mensajes de tamaño igual a 100KB, la dificultad en la transmisión de los mismos conlleva valores muy superiores de la mediana de la latencia.

El TTL de los mensajes tiene una incidencia clara en el valor de la mediana de la latencia, ya que permite que mensajes que tienen mayor dificultad en alcanzar a su destinatario debido a las condiciones de pérdida de contacto entre los nodos, se mantengan en la red un mayor tiempo, con lo que se incrementa la posibilidad de entrega y, con ello, la mediana de la latencia de la simulación.

La incidencia del protocolo de enrutamiento en el comportamiento de la mediana de la latencia está directamente relacionada con la sobrecarga que sufre la red, por lo que, para tamaños de mensaje iguales o inferiores a 10KB e intervalos entre mensajes iguales o inferiores a 15sg/msg, el uso de protocolos de enrutamiento más eficientes como *Spray&Wait* puede suponer mejoras en el valor de la mediana de la latencia de hasta el 50%.

La influencia de la variable intervalo entre mensajes sólo se pone de manifiesto para el tamaño de mensajes de 10KB, produciendo una importante reducción de la mediana de la latencia al pasar de 1sg/msg a 5sg/msg.

[10] La mediana de un conjunto de números es el valor del número que ocupa la posición central de dicho conjunto una vez que los números han sido ordenados de forma ascendente o descendente.

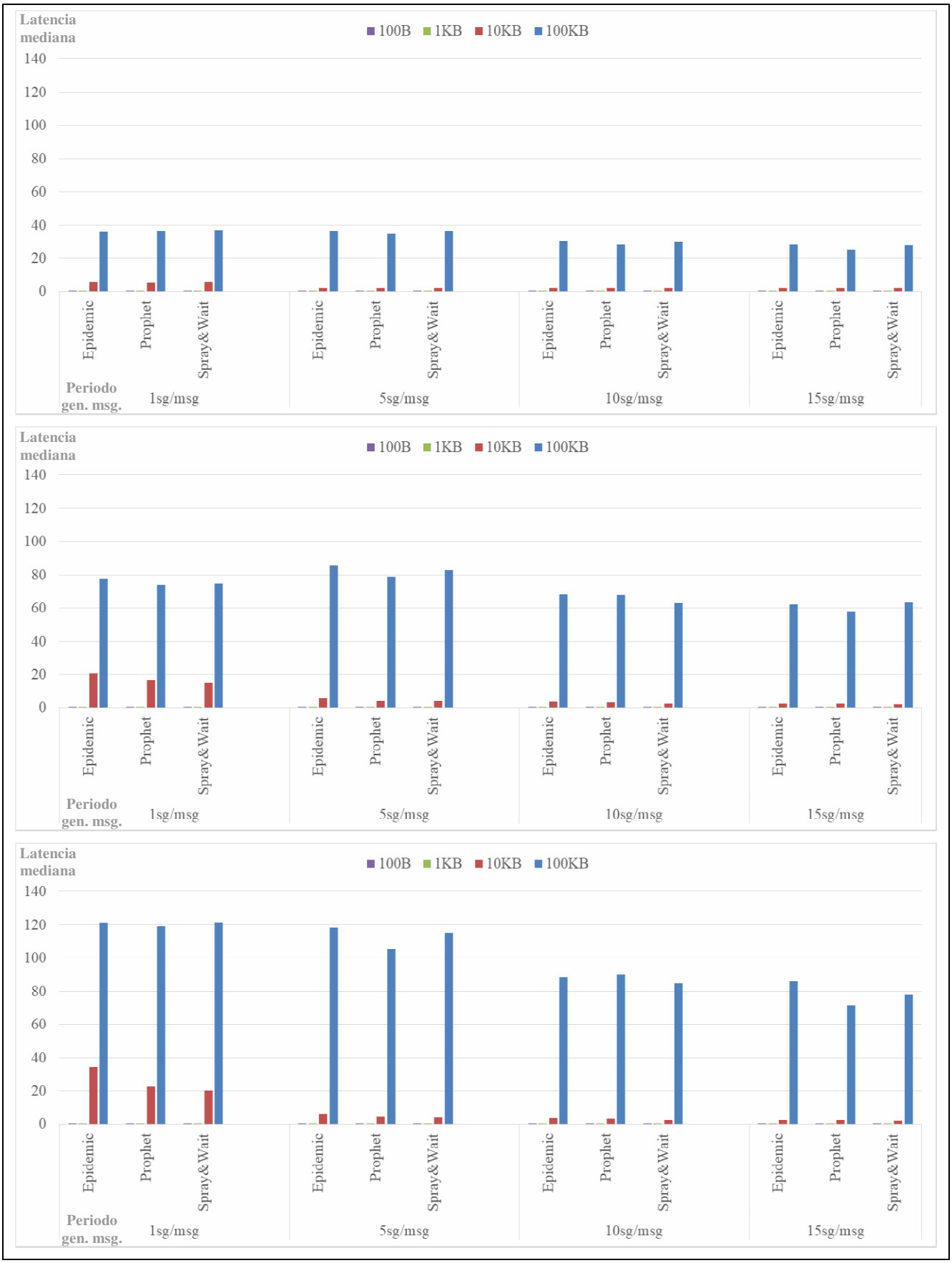

Figura 4-3 Representación del parámetro latency_med en función de las 4 variables consideradas en las simulaciones (arriba TTL=1, medio TTL=3, abajo TTL=5)

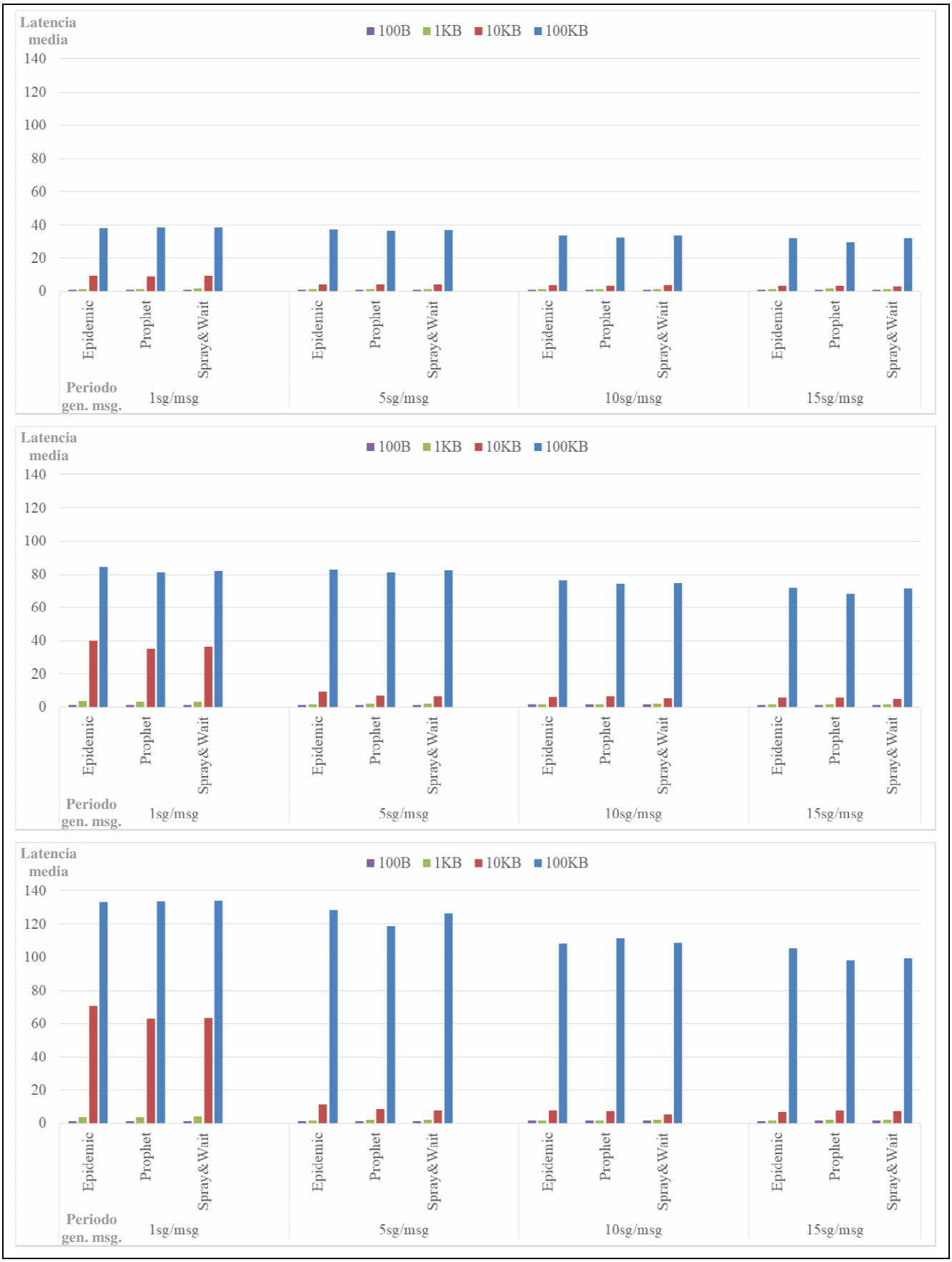

Figura 4-4 Representación del parámetro latency_avg en función de las 4 variables consideradas en las simulaciones (arriba TTL=1, medio TTL=3, abajo TTL=5)

*4.2.4 Parámetro "**latency_avg**"*

El parámetro **latency_avg** del fichero **MessageStatsReport** contiene el valor medio, o media, de los valores de latencia de los mensajes recibidos por los destinatarios.

Como se observa en las gráficas de la Figura 4-4, de las simulaciones realizadas, se extrae que, respecto a las variables consideradas, la media de la latencia de los mensajes se comporta de forma análoga a la de la mediana (parámetro **latency_med**). Únicamente reseñar que los valores obtenidos para el valor medio son mayores a los obtenidos para la mediana debido que la función mediana absorbe mejor que la función media los elevados valores de latencia aportados por los mensajes que, a medida que se eleva el valor del TTL de los mensajes, sufren un elevado retraso en ser entregados al destinatario.

4.3 Conclusiones técnicas obtenidas de las simulaciones

La principal conclusión que se obtiene del estudio llevado a cabo es que los valores obtenidos para los parámetros latencia y probabilidad de entrega de los mensajes, permitirían el uso de aplicaciones informáticas en el escenario utilizado para realizar las simulaciones que, por sus características de tasas de transferencia, degradación y pérdida de la conectividad entre nodos, se asemeja a lo que sería un entorno militar táctico.

De los 4 parámetros analizados en el epígrafe anterior, los 2 que arrojan información directa sobre el rendimiento ofrecido a las aplicaciones que hacen uso de la red son la probabilidad de entrega satisfactoria de mensajes y la latencia de dichos mensajes, en particular la mediana de dicho parámetro, ya que, como se ha señalado, absorbe mejor los valores elevados de latencia de los mensajes que sufren un retraso significativamente elevado a medida que se incrementa el TTL de los mismos.

Por su parte, el parámetro sobrecarga de red permite identificar especialmente el rendimiento del protocolo de enrutamiento y la influencia del TTL de los mensajes en la ocupación de la red.

Las principales conclusiones técnicas a las que se llega tras analizar en el epígrafe anterior los resultados de las simulaciones en el entorno previamente definido, son las siguientes:

- La probabilidad de entrega de los mensajes está impactada en un alto grado por el tamaño de los mensajes. Esto es debido principalmente a 2 factores: por un lado el reducido ancho de banda disponible en los interfaces de radio utilizados en este tipo de escenarios, y por otro lado, las interrupciones en los contactos entre nodos provocadas por la orografía, los entornos urbanos, los efectos del *jamming*, entre otros efectos. Se ha comprobado que cuando el tamaño de mensaje es igual o inferior a 1KB la probabilidad de entrega de los mensajes es cercana al 100% para cuales quiera valores de los parámetros TTL de los mensajes, intervalo entre mensajes y protocolo de enrutamiento. Cuando el tamaño de mensaje es de 10KB, se observa un incremento importante en la probabilidad de entrega de mensajes al pasar de un intervalo entre mensajes de 1sg/msg a 5sg/msg y superiores.
- Aunque en una medida mucho menor que el tamaño de los mensajes, la probabilidad de entrega de mensajes también está influenciada por el TTL de los mismos. En particular se observa una mejora al pasar de TTL=1 a TTL=3.
- La latencia (tanto el valor medio como la mediana) tiene una fuerte dependencia del tamaño de los mensajes y su comportamiento respecto a esta variable sigue la misma línea que para la probabilidad de entrega de mensajes, pero con carácter inverso.

- La latencia depende en gran medida del TTL de los mensajes cuando estos son de gran tamaño (100KB), pero para tamaños de mensaje de 10KB o inferior la latencia se mantiene aproximadamente constante para los diferentes valores de tamaño de mensaje e intervalo de mensaje a medida que se varía el TTL, con la excepción de los mensajes de 10KB e intervalo entre mensajes de 1sg/msg.
- El protocolo de enrutamiento utilizado se pone de manifiesto en la sobrecarga de la red, especialmente cuando el tamaño de los mensajes es pequeño (de 10KB o inferiores) y el TTL de los mensajes es lo suficientemente grande (TTL=3 o superior) como para que los mensajes clonados tengan tiempo suficiente de extenderse por la red. En estas circunstancias es cuando los protocolos basados en inundación pura, como Epidemic, muestran un rendimiento peor (dando lugar a una mayor sobrecarga de la red) que otros protocolos más elaborados. Dentro de estos protocolos más elaborados, en las simulaciones se ha utilizado PRoPHET y *Spray&Wait*. Considerando el escenario simulado, en el que los nodos están habitualmente dentro de la cobertura nominal de los otros nodos, y las pérdidas de contacto se deben a efectos como la orografía, las interferencias o los desvanecimientos, el protocolo PRoPHET (se basa en considerar el histórico de contactos anteriores de cada nodo) no presenta un nivel de mejora, respecto a los protocolos de inundación puros, tan elevado como el observado al utilizar *Spray&Wait*, que explícitamente limita el número de copias de cada mensaje en la red.
- El tamaño de los mensajes también afecta a la sobrecarga de la red de forma drástica cuando el TTL de los mensajes es lo suficientemente grande como para que los mensajes clonados se extiendan por la red. Este comportamiento es tanto más acusado cuanto menor sea el tamaño del mensaje, ya que tu transmisión completa resulta más sencilla. En este caso el uso del protocolo Spray&Wait es capaz de mantener el nivel de sobrecarga asociado a mensajes de pequeño tamaño en un orden de magnitud similar al de los mensajes de gran tamaño.

Resulta evidente que los objetivos perseguidos son:

- Maximizar la probabilidad de entrega de mensajes, para evitar retransmisiones o pérdidas de información.
- Minimizar la sobrecarga de la red, para aprovechar la capacidad de esta con más cantidad de tráfico sin que la probabilidad de entrega y la latencia se vean comprometidos.
- Minimizar la latencia de los mensajes, para mejorar todo lo posible la experiencia de usuario y abrir la puerta al uso de aplicaciones que tengan requisitos de temporización más estrictos.

Con la finalidad de alcanzar estos objetivos, y a la vista de la información obtenida del análisis de los resultados de las simulaciones en el escenario definido, se puede concluir que los valores óptimos de las variables estudiadas son:

- Como protocolo de enrutamiento la mejor opción es utilizar estrategias que limiten el número de copias de los mensajes en la red, como *Spray&Wait*.
- Sea cual sea el intervalo entre mensajes y el TTL, los mensajes de tamaño igual o inferior a 1KB tienen un buen comportamiento.
- Si el intervalo entre mensajes es igual o superior a 5sg/msg, el comportamiento para mensajes de tamaño igual a 10KB, aunque no es tan bueno como el de mensajes de menor tamaño, también sería admisible.

4.4 Información de interés en entornos tácticos

Los resultados arrojados por el conjunto de simulaciones realizadas y las conclusiones técnicas obtenidas de las mismas, junto con los 2 desarrollos presentados en el Capítulo 2 (los proyectos CONDOR y MIDNet), ponen de manifiesto que las DTNs constituyen una solución viable para las comunicaciones de datos en entornos militares tácticos que, debido a las actuales tendencias, como el combate colaborativo o el ComFut, han cobrado una especial importancia.

La información requerida en este tipo de escenarios depende de la naturaleza de las unidades involucradas en cada despliegue, su equipamiento y su misión pero, a modo de ejemplo, se pueden mencionar las siguientes necesidades:

- Posición, velocidad y rumbo en tiempo real de tropas.
- Comunicación de órdenes y tareas.
- Información meteorológica y oceanográfica: por ejemplo, temperatura ambiental, precipitaciones, viento, oleaje y temperatura del agua.
- Información ambiental: por ejemplo, contaminación NBQ.
- Información de objetivos.
- Información sobre infraestructuras: por ejemplo, planos, materiales de construcción y estado.
- Control de fuegos de artillería (planeamiento, ejecución, integración).
- Información para gestión de los planes de frecuencias.
- Configuración y uso de equipamiento de guerra electrónica.
- Información logística: por ejemplo, combustible, repuestos, armamento, munición, agua y alimentos.
- Información de carácter médico: por ejemplo, imágenes, instrucciones y tratamientos.
- Transmisión de fotografías e incluso video: por ejemplo, imágenes aéreas y de sistemas optrónicos.
- Información generada por sensores: por ejemplo, los instalados en los uniformes del ComFut y en infraestructuras.

El acceso a esta, u otra información, puede llevarse a cabo mediante cualquier tipo de interfaz, como pueden ser los siguientes:

- Clientes de mensajería militar, correo electrónico convencional y chat.
- Clientes GIS: para visualización de cartografía e información georreferenciada (por ejemplo, simbología y eventos).
- Interfaces web.
- Clientes ligeros o pesados de aplicaciones específicas.
- Sistemas de videoconferencia.

5 CONCLUSIONES Y LÍNEAS FUTURAS

5.1 Conclusiones

Este TFM tiene dos partes claramente diferenciadas. En el Capítulo 2 se han presentado las 2 implementaciones de DTN en entornos militares tácticos de las que se han encontrado referencias. En los Capítulos 3 y 4 se ha descrito el desarrollo práctico llevado a cabo con el simulador ONE y se han analizado los resultados de las simulaciones realizadas. De cada una de estas partes se extraen conclusiones particulares y, además, otras de carácter global.

En lo que respecta a la primera parte del TFM se pueden extraer las siguientes conclusiones:

- Las DTNs constituyen una solución para la comunicación en escenarios con perfiles muy bajos de conectividad en los que las tecnologías convencionales (IP sobre TCP o UDP) no funcionan.
- Los escenarios a los que están dirigidas las DTNs son escasos, pero de gran relevancia (por ejemplo, exploración espacial, entornos militares tácticos, intervención en situaciones de catástrofes y emergencias, seguimiento de fauna en peligro de extinción, entre otros), motivo por el que la investigación en este campo ha continuado y se han publicado RFCs que extienden las especificaciones originales.
- Si bien el origen de las DTNs tiene casi 2 décadas, el escaso uso de los escenarios para los que estas redes son una solución, ha provocado que el número de implementaciones reales sea muy reducido.
- El reducido número de implementaciones reales ha llevado a un bajo grado de madurez de la tecnología.
- Las aplicaciones convencionales no funcionan correctamente en este tipo de redes. Las aplicaciones que se pueden utilizar sobre DTNs requieren, bien desarrollos específicos, bien el uso de gateways para acomodar las aplicaciones convencionales a las características de conectividad de las DTNs.
- Los proyectos estudiados (red CONDOR y proyecto MIDNET) demuestran que las DTNs ofrecen a las fuerzas armadas una solución de comunicación para escenarios tácticos, incluso con características de conectividad muy degradadas.

Si bien las conclusiones de la segunda parte del TFM tienen un carácter eminentemente técnico (motivo por el que se han incluido en el Capítulo 4), también es posible extraer otras de carácter más general, que se presentan a continuación:

- La primera y más importante es que las DTNs, seleccionando los parámetros de funcionamiento adecuados, constituyen una solución a las comunicaciones de datos en entornos militares tácticos.
- Existen multitud de simuladores de redes telemáticas, pero el escaso desarrollo de las DTNs en comparación con otras tecnologías de red reduce el catálogo de simuladores para este tipo de entornos a unos pocos desarrollos experimentales y, principalmente, al simulador ONE.
- El simulador ONE está desarrollado en Java y su código fuente está disponible de forma libre, lo que facilita la adaptación de su funcionamiento o el desarrollo de nuevas funcionalidades.
- La variedad de parámetros de configuración del simulador ONE permiten modelar escenarios que permiten estudiar el comportamiento de las DTNs en entornos militares tácticos.
- La información contenida en los reportes generados por las simulaciones realizadas incluye, tanto datos estadísticos sobre el comportamiento general de la red, como *logs* detallados de los diferentes eventos ocurridos durante las simulaciones (establecimientos y pérdidas de contactos entre nodos, creación y transmisión entre nodos de cada uno de los mensajes, etc.). Esto permite entender el funcionamiento de la red, tanto desde un punto de vista de rendimiento general, como estudiar en detalle el comportamiento de los nodos y la transmisión de mensajes en circunstancias específicas.
- Los resultados arrojados por las simulaciones realizadas con ONE permiten definir cuantitativamente los valores óptimos de diversas variables con el objeto de maximizar el rendimiento de la red en los escenarios definidos.

Desde un punto de vista global, e integrador de ambas partes del TFM, se puede concluir lo siguiente:

- Los resultados obtenidos de las simulaciones realizadas están alineadas con el éxito de los proyectos CONDOR y MIDNET, que demuestran la aplicabilidad para las Fuerzas Armadas de las DTNs en entornos tácticos.
- La simulación constituye un medio idóneo para avanzar en el estudio de las DTNs en entornos militares tácticos por el coste económico y temporal de:
  - Integrar alguna de las implementaciones de DTNs sobre equipamiento real de radio táctica.
  - Testear un hipotético desarrollo de DTN sobre radio táctica en tests de campo, que requieren personal, vehículos, organización, etc.
- Es conveniente ampliar los casos de estudio con: trazas de los nodos (posicionamiento y contactos entre nodos) obtenidas de maniobras reales, otros protocolos de enrutamiento, valores de las variables, y dotar al simulador ONE de nuevas funcionalidades que permitan profundizar en el comportamiento de la tecnología DTN antes de iniciar desarrollos reales, para que estos vayan dirigidos a soluciones previamente validadas mediante simulación.

A la vista del contenido del TFM y, en particular, de las conclusiones presentadas en este capítulo, se puede afirmar que se ha alcanzado los objetivos establecidos al iniciar el trabajo.

5.2 Líneas futuras

Tras la elaboración del presente TFM, y en base a los conocimientos adquiridos, se han identificado varias líneas de trabajo encaminadas a ampliar el trabajo realizado. Una descripción somera de estas líneas de trabajo es la siguiente:

- Realizar desarrollos sobre el código del simulador ONE para que incorpore las siguientes funcionalidades:
  - Comportamiento full-duplex de los interfaces de red.
  - Soporte para tráfico *multicast* (actualmente no implementado) de forma simultanea a tráfico unicast.
  - Permitir definir el valor de TTL por mensaje (actualmente el TTL se define para todo el tráfico de un grupo), de forma que se puedan simular los diferentes tiempos de caducidad de la información de diferentes aplicaciones usadas por nodos de un mismo grupo, o de tipos de información gestionados por una misma aplicación.
- Realizar simulaciones utilizando, en vez de modelos de movimiento sintéticos, trazas reales de movimiento y contactos (establecimiento y pérdidas de contacto) de los nodos, obtenidas de maniobras reales de las Fuerzas Armadas.
- Realizar simulaciones utilizando tráfico real de aplicaciones utilizadas en los escenarios objeto de estudio.

Estos trabajos darían lugar a un conocimiento del comportamiento de las DTNs en entornos militares tácticos adecuado para plantear la conveniencia de dedicar recursos humanos, económicos y de tiempo a una integración en equipamiento de radio táctica de una implementación de DTN, que permitiera llevar a cabo pruebas de campo.

(Página dejada intencionadamente en blanco)

6 ABREVIATURAS Y ACRÓNIMOS

ACK: Acknowledgement.

ADU: Application Data Unit.

API: Application Program Interface.

ARM: Advanced RISC Machine.

BAB: Bundle Authentication Block.

BFT: Blue Force Tracking.

BP: Bundle Protocol.

CB: Confidentiality Block.

CBHE: Compressed Bundle Header Encoding.

CLA: Convergence Layer Adapter.

ComFut: Combatiente del Futuro.

CONDOR: Command and Control On-The-Move Network Digital Over-The-Horizon Relay.

DMC: Disaster Monitoring Constellation.

DNS: Domain Name System.

DTLSR: Delay-Tolerant Link State Routing.

DTN: Delay/Disruption Tolerant Network.

DTNRG: DTN Research Group.

EDA: European Defense Agency.

EID: Endpoint IDentifier.

EPLRS: Enhanced Position Location Reporting System.

GIS: Geographic Information System.

HTTP: Hypertext Transfer Protocol.

IANA: Internet Assigned Numbers Authority.

IAP: Information Acquisition & Processing.

IDS: Intrusion Detection System.

IETF: Internet Engineering Task Force.

IP: Internet Protocol.

IPNRG: Interplanetary Network Research Group.

IRTF: Internet Research Task Force.

JC2-V: Jump Command and Control Vehicle.

KMR: Keep Most Recent.

LTP: Licklider Transmission Protocol.

LTP-T: Licklider Transmission Protocol Transport.

MIDNet: Military Disruption Tolerant Networks.

MSL: Maximum Segment Lifetime.

NBQ: Nuclear, Bactereológico, Químico.

ONE: Opportunistic Networking Environment.

POSIX: Portable Operating System Interface.

POP3: Post Office Protocol v3.

PoP-V: Point od Presence Vehicle.

PRoPHET: Probabilistic Routing Protocol using History of Encounters and Transitivity.

PSB: Payload Security Block.

QoS: Quality od Service.

RFC: Request For Comments.

SABR: Schedule-Aware Bundle Routing.

SMTP: Simple Mail Transfer Protocol.

TCP: Transmission Control Protocol.

TOC: Tactical Operation Center.

TTL: Time To Live.

UDP: User Datagram Protocol.

UHF: Ultra High Frequency.

URI: Uniform Resource Identifier.

URL: Uniform Resource Locutor.

URN: Uniform Resource Name.

VHF: Very High Frequency.

7 BIBLIOGRAFÍA

[1] Vaishali S. Raj, Dr. R. Manicka Chezian. "Delay-Disruption Tolerant Network (DTN), its network characteristics and core applications". International Journal of Computer Science and Mobile Computing. Septiembre 2013.

[2] Wei Sun, Congmin Liu, Dan Wang. "On delay tolerant networking and its application". 2011 International Conference on Computer Science and Information Technology. IPCSIT vol. 51. 2012.

[3] Vinton Cerf, Scott Burleigh, A. Hooke, Jordan Torgerson, Robert Durst, Keith Scott, E. Travis, H. Weiss. "Interplanetary internet (IPN): architectural definition". 2001.

[4] Kevin Fall. "Delay-Tolerant Networking for challenged Internets". Presentation to ICIR. Junio 2002.

[5] Vinton Cerf, Scott Burleigh, A. Hooke, Jordan Torgerson, Robert Durst, Kevin Fall; H. Weiss. "Delay-Tolerant Network architecture". Marzo 2003.

[6] Ari Keränen, Jörg Ott, Teemu Kärkkäinen. "The ONE simulator for DTN protocol evaluation". SIMUTools. 2009.

[7] Web de Micro Planetary Communication Network (µPCN). https://upcn.eu/ . [En línea]. [Disponible]. [Último acceso: 31 octubre 2018].

[8] Vinton Cerf, Scott Burleigh, A. Hooke, Jordan Torgerson, Robert Durst, Keith Scott, Kevin Fall, H. Weiss. "RFC 4838 - Delay-Tolerant Networking architecture". https://tools.ietf.org/html/rfc4838 . Diciembre 2006.

[9] María Irene Páez Bencomo. "Análisis y evaluación de prestaciones de protocolos de encaminamiento en redes tolerantes al retardo". Trabajo fin de Máster. Universidad Politécnica de Madrid. Escuela Técnica Superior de Ingenieros de Telecomunicación. 2013.

[10] Web de Wikipedia "Routing in Delay-Tolerant Networking". https://en.wikipedia.org/wiki/Routing_in_delay-tolerant_networking . [En línea]. [Disponible]. [Último acceso: 31 octubre 2018].

[11] Khalil Massri, Andrea Vitaletti, Alessandro Vernata, Ioannis Chatzigiannakis. "Routing protocols for Delay Tolerant Networks: A reference architecture and a thorough quantitative evaluation". Journal on Sensor and Actuator Networks. 2016.

[12] Ilir Shinko, Tetsuya Oda, Evjola Spaho, Vladi Kolici, Makoto Ikeda, Leonard Barolli. "A simulation system based on ONE and SUMO simulators: Performance evaluation of First Contact, Prophet and Spray-and-Wait DTN protocols". International Conference on Broadband and Wireless Computing, Communication and Applications. Marzo 2016.

[13] Keith Scott. "Disruption Tolerant Networking proxies for on-the-move tactical networks". The MITRE Corporation. 2005.

[14] Jörg Ott, Dirk Kutscher. "Bunding the Web: HTTP over DTN". WNEPT. Agosto 2006.

[15] Scott Burleigh, Keith Scott. "RFC 5050 - Bundle Protocol Specification". https://tools.ietf.org/html/rfc5050 . Noviembre 2007.

[16] Kevin Fall, Stephen Farrel. "DTN: An architectural retrospective". IEEE Journal on Selected Areas in Communications, Vol.26, No.5. Junio 2008.

[17] Carlo Caini, H. Cruickshank, Stephen Farrell, Mario Marcheseo. "Delay and Disruption Tolerant Networking (DTN): An alternative solution for future satellite networking applications". Proceedings of the IEEE. 2011.

[18] M. Ramadas, Scott Burleigh, Stephen Farrel. "RFC 5326 - Licklider Transmission Protocol - Specification". https://tools.ietf.org/html/rfc5326 . Septiembre 2008.

[19] Michael Demmer, Eric Brewer, Kevin Fall, Sushant Jain, Melissa Ho, Rabin Patra. "Implementing Delay Tolerant Networking". Intel Corporation. Diciembre 2004.

[20] Sebastian Schildt, Johannes Morgenroth, Wolf-Bastian Pöttner, Lars Wolf. "IBR-DTN: A lightweight, modular and highly portable Bundle Protocol implementation". ECEASST. 2011.

[21] Marius Georgescu, Takemi Sahara, Muhammad Ashar, Hiroki Izumikawa, Yuta Onogi, Morihiko Tamai, Shigeru Kashihara. "Performance analisys of file transmission in DTN2 and IBR-DTN". IEICE Technical Report. 2012.

[22] Salil Parikh, Robert Durst. "Disruption Tolerant Networking for Marine Corps CONDOR". The MITRE Corporation. 2005.

[23] Robert Durst, Salil Parikh, Keith Scott, Jason Andresen, Karl Tritchler, Tom Ullrich, Tim Bultman. "Disruption Tolerant Networking for CONDOR". Presentation. MITRE Corporation. 2006.

[24] European Defence Agency. "Research &Technology". 2013.

[25] M. Małowidzki, P. Kaniewski, R. Matyszkiel, P. Bereziński. "Standard tactical services in a military Disruption-Tolerant Network: Field tests". IEEE Milcom 2017 Track 2 - Networking Protocols and Performance. 2017.

[26] M. Małowidzki, T. Dalecki, P. Bereziński, M. Mazur, P. Skarżyński. "Adapting standard tactical applications for a military Disruption-Tolerant Network". ICMCIS. 2016.

[27] Ari Keränen, Teemu Käkkäinen, Jörg Ott. "Simulating mobility and DTNs with the ONE". Journal of Communications, Vol.5, No.2. Febrero 2010.

[28] Defense Advanced Research Projects Agency (DARPA) - Information Processing Techniques Office. “RFC 793 - Transmission Control Protocol”. https://tools.ietf.org/html/rfc793 . Septiembre 1981.

(Página dejada intencionadamente en blanco)

ANEXO I: CÓDIGO FUENTE DE LA CLASE NetworkInterface

A continuación se muestra el contenido del fichero **NetworkInterface.java**, en el que se ha resaltado en tipografía negrita el código añadido.

```
/*
 * Copyright 2010 Aalto University, ComNet
 * Released under GPLv3. See LICENSE.txt for details.
 */
package core;

import interfaces.ConnectivityGrid;
import interfaces.ConnectivityOptimizer;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

import routing.util.EnergyModel;

import util.ActivenessHandler;

/**
 * Network interface of a DTNHost. Takes care of connectivity among hosts.
 */
abstract public class NetworkInterface implements ModuleCommunicationListener {
	/** tanto por ciento máximo de reducción de la cobertura -setting id ({@value})*/
	public static final String TRANSMIT_RANGE_DEGRADADO_RATIO_MAXIMO_S = "transmitRangeDegradadoRatioMaximo";
	/** tanto por ciento de ubicaciones en las que la cobertura seré inferior a la nominal -setting id ({@value})*/
	public static final String TRANSMIT_RANGE_DEGRADADO_RATIO_UBICACIONES_S = "transmitRangeDegradadoRatioUbicaciones";
	/** transmit range -setting id ({@value})*/
	public static final String TRANSMIT_RANGE_S = "transmitRange";
	/** transmit speed -setting id ({@value})*/
	public static final String TRANSMIT_SPEED_S = "transmitSpeed";
	/** scanning interval -setting id ({@value})*/
	public static final String SCAN_INTERVAL_S = "scanInterval";

	/**
	 * Sub-namespace for the network related settings in the Group namespace
	 * ({@value})
	 */
	public static final String NET_SUB_NS = "net";

	/** Activeness offset jitter -setting id ({@value})
	 * The maximum amount of random offset for the offset */
	public static final String ACT_JITTER_S = "activenessOffsetJitter";

	/** {@link ModuleCommunicationBus} identifier for the "scanning interval"
	variable. */
	public static final String SCAN_INTERVAL_ID = "Network.scanInterval";
	/** {@link ModuleCommunicationBus} identifier for the "radio range"
	variable. Value type: double */
	public static final String RANGE_ID = "Network.radioRange";
	/** {@link ModuleCommunicationBus} identifier for the "transmission speed"
	variable. Value type: integer */
	public static final String SPEED_ID = "Network.speed";
```

```
private static final int CON_UP = 1;
private static final int CON_DOWN = 2;

private static Random rng;
protected DTNHost host = null;

protected String interfacetype;
protected List<Connection> connections; // connected hosts
private List<ConnectionListener> cListeners = null; // list of listeners
private int address; // network interface address
protected double transmitRangeDegradadoRatioMaximo;
protected double transmitRangeDegradadoRatioUbicaciones;
protected double transmitRange;
protected double oldTransmitRange;
protected int transmitSpeed;
protected ConnectivityOptimizer optimizer = null;
/** scanning interval, or 0.0 if n/a */
private double scanInterval;
private double lastScanTime;

/** activeness handler for the node group */
private ActivenessHandler ah;
/** maximum activeness jitter value for the node group */
private int activenessJitterMax;
/** this interface's activeness jitter value */
private int activenessJitterValue;

static {
        DTNSim.registerForReset(NetworkInterface.class.getCanonicalName());
        reset();
}

/**
 * Resets the static fields of the class
 */
public static void reset() {
        rng = new Random(0);
}

/**
 * For creating an empty class of a specific type
 */
public NetworkInterface(Settings s) {
        this.interfacetype = s.getNameSpace();
        this.connections = new ArrayList<Connection>();

        this.transmitRangeDegradadoRatioMaximo = s.getDouble(TRANSMIT_RANGE_DEGRADADO_RATIO_MAXIMO_S);
        this.transmitRangeDegradadoRatioUbicaciones = s.getDouble(TRANSMIT_RANGE_DEGRADADO_RATIO_UBICACIONES_S);
        this.transmitRange = s.getDouble(TRANSMIT_RANGE_S);
        this.transmitSpeed = s.getInt(TRANSMIT_SPEED_S);
        ensurePositiveValue(transmitRangeDegradadoRatioMaximo, TRANSMIT_RANGE_DEGRADADO_RATIO_MAXIMO_S);
        ensurePositiveValue(transmitRangeDegradadoRatioUbicaciones, TRANSMIT_RANGE_DEGRADADO_RATIO_UBICACIONES_S);
        ensurePositiveValue(transmitRange, TRANSMIT_RANGE_S);
        ensurePositiveValue(transmitSpeed, TRANSMIT_SPEED_S);
}

/**
```

```
	 * For creating an empty class of a specific type
	 */
	public NetworkInterface() {
		this.interfacetype = "Default";
		this.connections = new ArrayList<Connection>();
	}

	/**
	 * copy constructor
	 */
	public NetworkInterface(NetworkInterface ni) {
		this.connections = new ArrayList<Connection>();
		this.host = ni.host;
		this.cListeners = ni.cListeners;
		this.interfacetype = ni.interfacetype;
		this.transmitRangeDegradadoRatioMaximo = ni.transmitRangeDegradadoRatioMaximo;
		this.transmitRangeDegradadoRatioUbicaciones = ni.transmitRangeDegradadoRatioUbicaciones;
		this.transmitRange = ni.transmitRange;
		this.transmitSpeed = ni.transmitSpeed;
		this.scanInterval = ni.scanInterval;
		this.ah = ni.ah;

		if (ni.activenessJitterMax > 0) {
			this.activenessJitterValue = rng.nextInt(ni.activenessJitterMax);
		} else {
			this.activenessJitterValue = 0;
		}

		this.scanInterval = ni.scanInterval;
		/* draw lastScanTime of [0 -- scanInterval] */
		this.lastScanTime = rng.nextDouble() * this.scanInterval;
	}

	/**
	 * Replication function
	 */
	abstract public NetworkInterface replicate();

	/**
	 * For setting the host - needed when a prototype is copied for several
	 * hosts
	 * @param host The host where the network interface is
	 */
	public void setHost(DTNHost host) {
		this.host = host;
		ModuleCommunicationBus comBus = host.getComBus();

		if (!comBus.containsProperty(SCAN_INTERVAL_ID) &&
		    !comBus.containsProperty(RANGE_ID)) {
			/* add properties and subscriptions only for the 1st interface */
			/* TODO: support for multiple interfaces */
			comBus.addProperty(SCAN_INTERVAL_ID, this.scanInterval);
			comBus.addProperty(RANGE_ID, this.transmitRange);
			comBus.addProperty(SPEED_ID, this.transmitSpeed);
			comBus.subscribe(SCAN_INTERVAL_ID, this);
			comBus.subscribe(RANGE_ID, this);
			comBus.subscribe(SPEED_ID, this);
		}
```

```
		if (transmitRange > 0) {
			optimizer = ConnectivityGrid.ConnectivityGridFactory(
					this.interfacetype.hashCode(), transmitRange);
			optimizer.addInterface(this);
		} else {
			optimizer = null;
		}
	}

	/**
	 * Sets group-based settings for the network interface
	 * @param s The settings object using the right group namespace
	 */
	public void setGroupSettings(Settings s) {
		s.setSubNameSpace(NET_SUB_NS);
		ah = new ActivenessHandler(s);

		if (s.contains(SCAN_INTERVAL_S)) {
			this.scanInterval =  s.getDouble(SCAN_INTERVAL_S);
		} else {
			this.scanInterval = 0;
		}
		if (s.contains(ACT_JITTER_S)) {
			this.activenessJitterMax = s.getInt(ACT_JITTER_S);
		}

		s.restoreSubNameSpace();
	}

	/**
	 * For checking what interface type this interface is
	 */
	public String getInterfaceType() {
		return interfacetype;
	}

	/**
	 * For setting the connectionListeners
	 * @param cListeners List of connection listeners
	 */
	public void setClisteners(List<ConnectionListener> cListeners) {
		this.cListeners = cListeners;
	}


	/**
	 * Returns the transmit range of this network layer
	 * @return the transmit range
	 */
	public double getTransmitRange()
	{

		Random random = new Random();
		int randomInt = random.nextInt(100)+1;
		double actualTransmitRange = this.transmitRange;
		double MaxDegradationRatio = (this.transmitRangeDegradadoRatioMaximo)/100;
```

```
                        if (host.isMovementActive() == false)
                        {
                                actualTransmitRange = 50.0;
                        }

                        else
                        {

                                if (randomInt > this.transmitRangeDegradadoRatioUbicaciones)
                                {
                                        actualTransmitRange = this.transmitRange;
                                }

                                if (this.transmitRangeDegradadoRatioUbicaciones >= randomInt)
                                {
                                        actualTransmitRange = this.transmitRange * ((1 - MaxDegradationRatio) +
MaxDegradationRatio*randomInt/100);
                                }

                        }

                        return actualTransmitRange;
                        //return this.transmitRange;
                }


                /**
                 * Returns the transmit speed of this network layer with respect to the
                 * another network interface
                 * @param ni The other network interface
                 * @return the transmit speed
                 */
                public int getTransmitSpeed(NetworkInterface ni) {
                        return this.transmitSpeed;
                }

                /**
                 * Returns a list of currently connected connections
                 * @return a list of currently connected connections
                 */
                public List<Connection> getConnections() {
                        return this.connections;
                }

                /**
                 * Returns true if the interface is on at the moment (false if not)
                 * @return true if the interface is on at the moment (false if not)
                 */
                public boolean isActive() {
                        boolean active;

                        if (ah == null) {
                                return true; /* no handler: always active */
                        }

                        active = ah.isActive(this.activenessJitterValue);
```

```
		if (active && host.getComBus().getDouble(EnergyModel.ENERGY_VALUE_ID,
							1) <= 0) {
			/* TODO: better way to check battery level */
			/* no battery -> inactive */
			active = false;
		}

		if (active == false && this.transmitRange > 0) {
			/* not active -> make range 0 */
			this.oldTransmitRange = this.transmitRange;
			host.getComBus().updateProperty(RANGE_ID, 0.0);
		} else if (active == true && this.transmitRange == 0.0) {
			/* active, but range == 0 -> restore range  */
			host.getComBus().updateProperty(RANGE_ID,
						this.oldTransmitRange);
		}
		return active;
	}

	/**
	 * Checks if this interface is currently in the scanning mode
	 * @return True if the interface is scanning; false if not
	 */
	public boolean isScanning() {
		double simTime = SimClock.getTime();

		if (!isActive()) {
			return false;
		}

		if (scanInterval > 0.0) {
			if (simTime < lastScanTime) {
				return false; /* not time for the first scan */
			}
			else if (simTime > lastScanTime + scanInterval) {
				lastScanTime = simTime; /* time to start the next scan round */
				return true;
			}
			else if (simTime != lastScanTime ){
				return false; /* not in the scan round */
			}
		}
		/* interval == 0 or still in the same scan round as when
		   last time asked */
		return true;
	}

	/**
	 * Returns true if one of the connections of this interface is transferring
	 * data
	 * @return true if the interface transferring
	 */
	public boolean isTransferring() {
		for (Connection c : this.connections) {
			if (c.isTransferring()) {
				return true;
			}
		}
```

```
                return false;
        }

        /**
         * Connects the interface to another interface.
         *
         * Overload this in a derived class.  Check the requirements for
         * the connection to work in the derived class, then call
         * connect(Connection, NetworkInterface) for the actual connection.
         * @param anotherInterface The interface to connect to
         */
        public abstract void connect(NetworkInterface anotherInterface);

        /**
         * Connects this host to another host. The derived class should check
         * that all pre-requisites for making a connection are satisfied before
         * actually connecting.
         * @param con The new connection object
         * @param anotherInterface The interface to connect to
         */
        protected void connect(Connection con, NetworkInterface anotherInterface) {
                this.connections.add(con);
                notifyConnectionListeners(CON_UP, anotherInterface.getHost());

                // set up bidirectional connection
                anotherInterface.getConnections().add(con);

                // inform routers about the connection
                this.host.connectionUp(con);
                anotherInterface.getHost().connectionUp(con);
        }

        /**
         * Disconnects this host from another host.  The derived class should
         * make the decision whether to disconnect or not
         * @param con The connection to tear down
         */
        protected void disconnect(Connection con,
                        NetworkInterface anotherInterface) {
                con.setUpState(false);
                notifyConnectionListeners(CON_DOWN, anotherInterface.getHost());

                // tear down bidirectional connection
                if (!anotherInterface.getConnections().remove(con)) {
                        throw new SimError("No connection " + con + " found in " +
                                        anotherInterface);
                }

                this.host.connectionDown(con);
                anotherInterface.getHost().connectionDown(con);
        }

        /**
         * Returns true if another interface is within radio range of this interface
         * and this interface is also within radio range of the another interface.
         * @param anotherInterface The another interface
         * @return True if the interface is within range, false if not
         */
```

```
protected boolean isWithinRange(NetworkInterface anotherInterface) {
        double smallerRange = anotherInterface.getTransmitRange();
        double myRange = getTransmitRange();
        if (myRange < smallerRange) {
                smallerRange = myRange;
        }

        return this.host.getLocation().distance(
                        anotherInterface.getHost().getLocation()) <= smallerRange;
}

/**
 * Returns true if the given NetworkInterface is connected to this host.
 * @param netinterface The other NetworkInterface to check
 * @return True if the two hosts are connected
 */
protected boolean isConnected(NetworkInterface netinterface) {
        for (int i = 0; i < this.connections.size(); i++) {
                if (this.connections.get(i).getOtherInterface(this) ==
                        netinterface) {
                        return true;
                }
        }
        return false;
}

/**
 * Makes sure that a value is positive
 * @param value Value to check
 * @param settingName Name of the setting (for error's message)
 * @throws SettingsError if the value was not positive
 */
protected void ensurePositiveValue(double value, String settingName) {
        if (value < 0) {
                throw new SettingsError("Negative value (" + value +
                                ") not accepted for setting " + settingName);
        }
}

/**
 * Updates the state of current connections (ie tears down connections
 * that are out of range, recalculates transmission speeds etc.).
 */
abstract public void update();

/**
 * Notifies all the connection listeners about a change in connections.
 * @param type Type of the change (e.g. {@link #CON_DOWN} )
 * @param otherHost The other host on the other end of the connection.
 */
private void notifyConnectionListeners(int type, DTNHost otherHost) {
        if (this.cListeners == null) {
                return;
        }
        for (ConnectionListener cl : this.cListeners) {
                switch (type) {
                case CON_UP:
                        cl.hostsConnected(this.host, otherHost);
```

```
                                        break;
                                case CON_DOWN:
                                        cl.hostsDisconnected(this.host, otherHost);
                                        break;
                                default:
                                        assert false : type;        // invalid type code
                                }
                        }
        }

        /**
         * This method is called by the {@link ModuleCommunicationBus} when/if
         * someone changes the scanning interval, transmit speed, or range
         * @param key Identifier of the changed value
         * @param newValue New value for the variable
         */
        public void moduleValueChanged(String key, Object newValue) {
                if (key.equals(SCAN_INTERVAL_ID)) {
                        this.scanInterval = (Double)newValue;
                }
                else if (key.equals(SPEED_ID)) {
                        this.transmitSpeed = (Integer)newValue;
                }
                else if (key.equals(RANGE_ID)) {
                        this.transmitRange = (Double)newValue;
                }
                else {
                        throw new SimError("Unexpected combus ID " + key);
                }
        }

        /**
         * Creates a connection to another host. This method does not do any checks
         * on whether the other node is in range or active
         * (cf. {@link #connect(NetworkInterface)}).
         * @param anotherInterface The interface to create the connection to
         */
        public abstract void createConnection(NetworkInterface anotherInterface);

        /**
         * Disconnect a connection between this and another host.
         * @param anotherInterface The other host's network interface to disconnect
         * from this host
         */
        public void destroyConnection(NetworkInterface anotherInterface) {
                DTNHost anotherHost = anotherInterface.getHost();
                for (int i=0; i < this.connections.size(); i++) {
                        if (this.connections.get(i).getOtherNode(this.host) == anotherHost){
                                removeConnectionByIndex(i, anotherInterface);
                        }
                }
                // the connection didn't exist, do nothing
        }

        /**
         * Removes a connection by its position (index) in the connections array
         * of the interface
         * @param index The array index of the connection to be removed
```

```
	 * @param anotherInterface The interface of the other host
	 */
	private void removeConnectionByIndex(int index,
				NetworkInterface anotherInterface) {
		Connection con = this.connections.get(index);
		DTNHost anotherNode = anotherInterface.getHost();
		con.setUpState(false);
		notifyConnectionListeners(CON_DOWN, anotherNode);

		// tear down bidirectional connection
		if (!anotherInterface.getConnections().remove(con)) {
			throw new SimError("No connection " + con + " found in " +
					anotherNode);
		}

		this.host.connectionDown(con);
		anotherNode.connectionDown(con);

		connections.remove(index);
	}

	/**
	 * Returns the DTNHost of this interface
	 */
	public DTNHost getHost() {
		return host;
	}

	/**
	 * Returns the current location of the host of this interface.
	 * @return The location
	 */
	public Coord getLocation() {
		return host.getLocation();
	}

	/**
	 * Returns a string representation of the object.
	 * @return a string representation of the object.
	 */
	public String toString() {
		return this.address + " of " + this.host +
			". Connections: " +	this.connections;
	}

}
```

ANEXO II: FICHERO DE CONFIGURACIÓN DEL ESCENARIO default_settings.txt

A continuación se muestra el contenido del fichero **default_settings.txt** con el que se define el escenario utilizado en las simulaciones.

```
##############################################################################


#
# Settings for the simulation
#


##############################################################################


## Scenario settings


# one.bat y después ejecutar la simulación en entorno gráfico
#Scenario.name = TacticalRadioSimulation_

# one.bat -b 12 default_settings.txt
Scenario.name = TacticalRadioSimulation__GroupRouter-%%Group.router%%__GroupMsgttl-%%Group.msgTtl%%__

Scenario.simulateConnections = true
Scenario.updateInterval = 0.1

# 1h = 3600s
# 2h = 7200s
# 3h = 10800s
# 4h = 14400s
# 6h = 21600s
# 8h = 28800s
# 10h = 36000s
# 12h = 43200s
# Sumar los 300sg que dejo como tiempo de warmup (Report.warmup) para que los
# algoritmos de enrutamiento que lo precisen, aprendan los patrones de contactos
# 10800+300=11100sg
Scenario.endTime = 11100

# Define different node groups
Scenario.nrofHostGroups = 11


##############################################################################


## Interface-specific settings:


# type : which interface class the interface belongs to
# For different types, the sub-parameters are interface-specific
# For SimpleBroadcastInterface, the parameters are:
```

```
# transmitSpeed : transmit speed of the interface (bytes per second)
# transmitRange : range of the interface (meters)

# Radio tactical interface for all nodes
# Transmit speed of 38,4 Kbps = 5kBps

# TacticalRadio1: vehicular
TacticalRadio1.type = SimpleBroadcastInterface
TacticalRadio1.transmitSpeed = 5k
TacticalRadio1.transmitRange = 7000

# TacticalRadio2: handheld
TacticalRadio2.type = SimpleBroadcastInterface
TacticalRadio2.transmitSpeed = 5k
TacticalRadio2.transmitRange = 1000


# Parámetros creados para simular reducciones de cobertura
# transmitRangeDegradadoRatioUbicaciones : tanto por ciento de ciclos en los que,
#   siguiendo una distribución uniforme, el nodo trandrá el transmitRange con un
#   valor degradado
# transmitRangeDegradadoRatioMaximo : tanto por ciento máximo de reducción de la
#   cobertura, respecto al valor nominal, cuando se produzca una reducción de la
#   cobertura, sin llegar a ser una pérdida de cobertura
TacticalRadio1.transmitRangeDegradadoRatioUbicaciones = 20
TacticalRadio1.transmitRangeDegradadoRatioMaximo = 30
TacticalRadio2.transmitRangeDegradadoRatioUbicaciones = 40
TacticalRadio2.transmitRangeDegradadoRatioMaximo = 40


##############################################################################


## Group-specific settings:


# groupID : Group's identifier. Used as the prefix of host names
# nrofHosts: number of hosts in the group
# movementModel: movement model of the hosts (valid class name from movement package)
#    Example: RandomWaypoint, RandomWalk, ClusterMovement, StationaryMovement, ...
# waitTime: minimum and maximum wait times (seconds) after reaching destination
# speed: minimum and maximum speeds (m/s) when moving on a path
#    30Km/h = 8m/s
#    50Km/h = 14m/s
#    120Km/h = 33m/s
# bufferSize: size of the message buffer (bytes)
# router: router used to route messages (valid class name from routing package)
# activeTimes: Time intervals (seconds) when the nodes in the group are
#    active (start1, end1, start2, end2, ...)
# activePeriods: Defines the activity and inactivity periods (seconds).
#    Example: 2000,500. Here node is periodically first active for 2000 seconds,
#    followed by 500 seconds of inactiveness, and 2000 seconds of activeness, 500
#    seconds of inactiveness, etc.
# activePeriodsOffset: Defines how much the activity periods are offset from
#    sim time 0. Value X means that the first period has been on for X seconds at
#    sim time 0. Default = 0 (i.e., first active period starts at 0)
# msgTtl : TTL (minutes) of the messages created by this host group, default=infinite
```

```
#-----------------


## Common settings for all groups


#Group.router = EpidemicRouter
Group.router = [EpidemicRouter; ProphetRouter; SprayAndWaitRouter;]

#Group.msgTtl = 5
Group.msgTtl = [1; 2; 3; 5;]

# Las variantes asociadas al tamaño y a la frecuencia de los paquetes, las genero
#    con diferentes ficheros de tráfico

Group.bufferSize = 1024M

# Interfaces por defecto de los grupos
#Group.nrofInterfaces = 1
#Group.interface1 = TacticalRadio1
#Group.interface2 = TacticalRadio2


#-----------------


## Group2 specific settings
Group2.groupID = INFv_N
Group2.nrofHosts = 2
Group2.nrofInterfaces = 1
Group2.interface1 = TacticalRadio1
Group2.speed = 8 , 14
Group2.movementModel = ClusterMovement
Group2.clusterCenter = 1000 , 5000
Group2.clusterRange = 75
Group2.activePeriods = 300 , 30
Group2.activePeriodsOffset = 0


#-----------------


## Group3 specific settings
Group3.groupID = TRXv_N
Group3.nrofHosts = 1
Group3.nrofInterfaces = 2
Group3.interface1 = TacticalRadio1
Group3.interface2 = TacticalRadio2
Group3.speed = 8 , 14
Group3.movementModel = ClusterMovement
Group3.clusterCenter = 1500 , 5100
Group3.clusterRange = 50
Group3.activePeriods = 400 , 20
Group3.activePeriodsOffset = 50


#-----------------
```

```
## Group1 specific settings
Group1.groupID = PCBONv_N
Group1.nrofHosts = 1
Group1.nrofInterfaces = 1
Group1.interface1 = TacticalRadio1
Group1.speed = 8 , 14
Group1.movementModel = StationaryMovement
Group1.nodeLocation = 2000 , 5000
Group1.activePeriods = 600 , 10
Group1.activePeriodsOffset = 80


#-----------------


## Group4 specific settings
Group4.groupID = EWLv_N
Group4.nrofHosts = 1
Group4.nrofInterfaces = 2
Group4.interface1 = TacticalRadio1
Group4.interface2 = TacticalRadio2
Group4.speed = 8 , 14
Group4.movementModel = ClusterMovement
Group4.clusterCenter = 2500 , 4900
Group4.clusterRange = 50
Group4.activeTimes = 1 , 120,130 , 520,530 , 1220,1230 , 1820,1830 , 2100,2130 , 2350,2380 , 2650,2680 , 3200,3210 , 3520,3530 , 3720,3730 , 3920,3930 ,
4500,4530 , 4820,4850 , 5075,5100 , 5350,5400 , 5700,5730 , 6000,6010 , 6250,6280 , 6520,6530 , 7320,7330 , 8060,8090 , 8280,8300 , 8570,8600 , 8880,8900 ,
9180,9200 , 9410,9440 , 9750,9770 , 10000,10040 , 10300,10330 , 10610,10620 , 11100
# Group4.activeTimes: Agrupados por periodos de inactividad


#-----------------


## Group7 specific settings
Group7.groupID = EWLd_N
Group7.nrofHosts = 4
Group7.nrofInterfaces = 1
Group7.interface1 = TacticalRadio2
Group7.speed = 0.5 , 2.5
Group7.waitTime = 60, 300
Group7.movementModel = ClusterMovement
Group7.clusterCenter = 3000 , 4300
Group7.clusterRange = 300
Group7.activeTimes = 4500,6100 , 8000,9900
# Group7.activeTimes: Agrupados por periodos de actividad


#-----------------


## Group8 specific settings
Group8.groupID = EWLd_N
Group8.nrofHosts = 4
Group8.nrofInterfaces = 1
Group8.interface1 = TacticalRadio2
Group8.speed = 0.5 , 2.5
```

```
Group8.waitTime = 60, 300
Group8.movementModel = ClusterMovement
Group8.clusterCenter = 2500 , 5800
Group8.clusterRange = 300
Group8.activeTimes = 4500,6100 , 8000,9900
# Group8.activeTimes: Agrupados por periodos de actividad


#-----------------


## Group5 specific settings
Group5.groupID = ZAPv_N
Group5.nrofHosts = 1
Group5.nrofInterfaces = 2
Group5.interface1 = TacticalRadio1
Group5.interface2 = TacticalRadio2
Group5.speed = 8 , 14
Group5.movementModel = ClusterMovement
Group5.clusterCenter = 3000 , 5100
Group5.clusterRange = 50
Group5.activeTimes = 1 , 110,140 , 510,540 , 1210,1240 , 1810,1840 , 2100,2130 , 2350,2380 , 2650,2680 , 3200,3210 , 3510,3540 , 3710,3740 , 3910,3940 ,
4500,4530 , 4820,4850 , 5075,5100 , 5350,5400 , 5700,5730 , 6000,6010 , 6250,6280 , 6510,6540 , 7310,7340 , 8060,8090 , 8280,8300 , 8570,8600 , 8880,8900 ,
9180,9200 , 9410,9440 , 9750,9770 , 10000,10040 , 10300,10330 , 10610,10620 , 11100
# Group5.activeTimes: Agrupados por periodos de inactividad


#-----------------


## Group9 specific settings
Group9.groupID = ZAPd_N
Group9.nrofHosts = 6
Group9.nrofInterfaces = 1
Group9.interface1 = TacticalRadio2
Group9.speed = 0.5 , 2.5
Group9.waitTime = 60, 300
Group9.movementModel = ClusterMovement
Group9.clusterCenter = 3500 , 5800
Group9.clusterRange = 100
Group9.activeTimes = 2060,3300
# Group9.activeTimes: Agrupados por periodos de actividad


#-----------------


## Group6 specific settings
Group6.groupID = VECv_N
Group6.nrofHosts = 1
Group6.nrofInterfaces = 2
Group6.interface1 = TacticalRadio1
Group6.interface2 = TacticalRadio2
Group6.speed = 8 , 14
Group6.movementModel = ClusterMovement
Group6.clusterCenter = 3500 , 5000
Group6.clusterRange = 50
Group6.activeTimes = 1 , 100,150 , 500,550 , 1200,1250 , 1800,1850 , 2100,2130 , 2350,2380 , 2650,2680 , 3200,3210 , 3500,3550 , 3700,3750 , 3900,3950 ,
4500,4530 , 4820,4850 , 5075,5100 , 5350,5400 , 5700,5730 , 6000,6010 , 6250,6280 , 6500,6550 , 7300,7350 , 8060,8090 , 8280,8300 , 8570,8600 , 8880,8900 ,
```

```
9180,9200 , 9410,9440 , 9750,9770 , 10000,10040 , 10300,10330 , 10610,10620 , 11100
# Group6.activeTimes: Agrupados por periodos de inactividad


#-----------------


## Group10 specific settings
Group10.groupID = CABd_N
Group10.nrofHosts = 6
Group10.nrofInterfaces = 1
Group10.interface1 = TacticalRadio2
Group10.speed = 0.5 , 2.5
Group10.waitTime = 60, 300
Group10.movementModel = ClusterMovement
Group10.clusterCenter = 3950 , 4850
Group10.clusterRange = 100
Group10.activeTimes = 2000,3350 , 4300,6200 , 7800,10000
# Group10.activeTimes: Agrupados por periodos de actividad


#-----------------


## Group11 specific settings
Group11.groupID = UAV_N
Group11.nrofHosts = 1
Group11.nrofInterfaces = 2
Group11.interface1 = TacticalRadio1
Group11.interface2 = TacticalRadio2
Group11.speed = 30 , 40
Group11.movementModel = RandomWaypoint
Group11.activeTimes = 1900,3500 , 4000,6300 , 7500,10100
# Group11.activeTimes: Agrupados por periodos de actividad


##############################################################################


## Default settings for some routers settings


ProphetRouter.secondsInTimeUnit = 30

SprayAndWaitRouter.nrofCopies = 6
SprayAndWaitRouter.binaryMode = true


##############################################################################


## Message creation parameters


# How many event generators
Events.nrof = 1

# Class of the first event generator
```

```
#Events1.class = MessageEventGenerator
Events1.class = ExternalEventsQueue

#-----------------

## (following settings are specific for the MessageEventGenerator class)

# Creation interval in seconds (one new message every 25 to 35 seconds)
#Events1.interval = 25,35
#Events1.interval = 120, 300

# Message sizes (500kB - 1MB)
#Events1.size = 500k,1M
#Events1.size = 50k, 200k

# range of message source/destination addresses
#Events1.hosts = 0,5

# Message ID prefix
#Events1.prefix = M

#-----------------

Events1.filePath = Z_mis-ficheros-de-trafico/2-final/Z_trafico_Completo15_mod_100KB.txt


##############################################################################


## Movement model settings


# seed for movement models' pseudo random number generator (default = 0)
MovementModel.rngSeed = 2
# World's size for Movement Models without implicit size (width, height; meters)
MovementModel.worldSize = 8000, 10000
# How long time to move hosts in the world before real simulation
#MovementModel.warmup = 1000


##############################################################################


## Reports - all report names have to be valid report classes


# how many reports to load
Report.nrofReports = 8
# length of the warm up period (simulated seconds)
Report.warmup = 300
# default directory of reports (can be overridden per Report with output setting)
Report.reportDir = reports/
# Report classes to load
Report.report1 = ConnectivityONEReport
Report.report2 = CreatedMessagesReport
Report.report3 = DeliveredMessagesReport
Report.report4 = DistanceDelayReport
Report.report5 = EventLogReport
```

```
Report.report6 = MessageGraphvizReport
Report.report7 = MessageReport
Report.report8 = MessageStatsReport


##############################################################################


## Optimization settings -- these affect the speed of the simulation
## see World class for details.


Optimization.cellSizeMult = 5
Optimization.randomizeUpdateOrder = true


##############################################################################


## GUI settings


# how many events to show in the log panel (default = 30)
GUI.EventLogPanel.nrofEvents = 1000
# Regular Expression log filter (see Pattern-class from the Java API for RE-matching details)
#GUI.EventLogPanel.REfilter = .*p[1-9]<->p[1-9]$


##############################################################################
```

ANEXO III: FRAGMENTO DE UN FICHERO DE TRÁFICO

A continuación se muestra un fragmento del contenido del fichero de tráfico correspondiente a un perfil de mensajes de 1KB con un intervalo medio de tiempo de creación de mensajes de 1 mensaje cada 10 segundos.

```
.....
1809.9  C  M150  0  6  1012
1810.7  C  M151  1  0  977
1819.4  C  M152  2  0  982
1839.2  C  M153  3  0  906
1853.9  C  M154  4  0  923
1854.0  C  M155  5  0  996
1856.0  C  M156  6  0  964
1872.8  C  M157  0  1  921
1883.3  C  M158  0  2  1019
1893.5  C  M159  0  3  1062
1894.7  C  M160  0  4  994
2060.0  C  M161  27  0  962
2061.9  C  M162  0  5  1042
2065.6  C  M163  0  6  911
2077.0  C  M164  5  0  1048
2090.8  C  M165  6  0  973
2100.7  C  M166  27  0  974
2101.4  C  M167  6  21  946
2101.8  C  M168  6  22  948
2106.2  C  M169  6  23  1061
2119.1  C  M170  6  24  1048
2129.2  C  M171  6  25  1004
2133.0  C  M172  6  26  1068
2143.2  C  M173  5  15  1034
2148.5  C  M174  5  16  1047
2151.8  C  M175  5  17  1052
2158.8  C  M176  5  18  949
2164.2  C  M177  5  19  958
2195.1  C  M178  5  20  983
2199.1  C  M179  27  0  993
2199.9  C  M180  21  6  1084
2203.9  C  M181  15  5  1036
2209.3  C  M182  5  0  961
2210.9  C  M183  6  0  925
2238.4  C  M184  5  15  994
2250.6  C  M185  6  21  901
2281.1  C  M186  27  0  1042
2286.7  C  M187  0  5  940
2287.6  C  M188  0  6  963
2296.8  C  M189  5  0  1066
2310.0  C  M190  6  0  1064
2321.0  C  M191  27  0  1077
.....
```