Instituto Politécnico Nacional

Centro de Investigación en Computación

Enrutamiento Seguro y Escalable en Redes Móviles Ad Hoc

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS DE LA COMPUTACIÓN

presenta

Ing. Miguel Rodríguez Martínez

Directores de tesis:

Dr. Rolando Menchaca Méndez

Dr. Felipe R. Menchaca García

México, D.F., Agosto de 2011

SIP-14 bis

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 25 del mes de Julio de 2011 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"ENRUTAMIENTO SEGURO Y ESCALABLE EN REDES MÓVILES AD HOC"

Presentada por el alumno:

RODRÍGUEZ	**MARTÍNEZ**	**MIGUEL**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	0	9	1	6	6	7

aspirante de: **MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de Tesis

Dr. Rolando Menchaca Mendez

Dr. Felipe Rolando Menchaca Garcia

Dr. Marco Antonio Moreno Armendáriz

Dr. Marco Antonio Moreno Ibarra

Dr. José Giovanni Guzmán Lugo

PRESIDENTE DEL COLEGIO DE PROFESORES

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
DIRECCIÓN

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México DF el día 8 del mes de agosto del año 2011, el (la) que suscribe Miguel Rodríguez Martínez alumno (a) del Programa de Maestría en Ciencias de la Computación con número de registro B091667, adscrito al Centro de Investigación en Computación, manifiesta que es autor (a) intelectual del presente trabajo de Tesis bajo la dirección del Dr. Rolando Menchaca Méndez y cede los derechos del trabajo intitulado Enrutamiento Seguro y Escalable en Redes Móviles Ad Hoc, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a la siguiente dirección miguelroma1@hotmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Miguel Rodriguez Martínez
Nombre y firma

RESUMEN

Uno de los problemas abiertos más importantes en los protocolos de enrutamiento para redes móviles ad hoc es la implementación de mecanismos de seguridad que garanticen que la comunicación entre los nodos que componen la red sea segura, confiable y eficaz. En general, los ataques a este tipo de redes se pueden clasificar en una de las dos clases siguientes: basados en el consumo de recursos y contra el enrutamiento. En el presente trabajo se presenta el Protocolo de Enrutamiento Seguro Bajo Demanda Basado en Vector de Distancias (OSDV) en el cuál se proporcionan mecanismos de seguridad para protegerse en contra de ataques en contra del enrutamiento como lo son: afirmar tener una distancia más corta hacia el destino, hacerse pasar por el origen o el destino, modificar el número secuencial de un destino, ataque one hop. Tales mecanismos de seguridad son *cadenas hash*, *firmas digitales* y *ordenación en tiempo*. Estas técnicas han sido empleadas por separado siendo hasta ahora su utilización de manera conjunta para proveer un mejor mecanismo de seguridad.

Al utilizar este protocolo, los nodos de una red móvil ad hoc (MANET, por sus siglas en inglés) pueden llevar a cabo comunicaciones seguras al descubrir, establecer y reparar rutas utilizando los mecanismos de seguridad incluidos en el protocolo, aún ante la presencia de un conjunto de nodos adversarios. El protocolo emplea las *cadenas hash* para autenticar el conteo de salto de cada nodo (información mutable), las *firmas digitales* para asegurar que ningún dato haya sido modificado (información no mutable) y *ordenación en tiempo* para detectar que la autenticación de conteo de salto sea llevada a cabo de acuerdo a las políticas del protocolo.

El protocolo OSDV fue simulado en el entorno NS2 (Network Simulator release 2) teniendo como parámetro de comparación al protocolo de enrutamiento bajo demanda basado en vector de distancias (AODV). Esto con el fin de comprobar el funcionamiento de dichos protocolos en presencia de ataques. Nuestros experimentos muestran que OSDV es considerablemente más resistente a ataques que AODV.

ABSTRACT

In the context of routing in mobile ad hoc networks (MANETs), one of the most challenging problems is the implementation of security policies and mechanisms capable of securing the communication among the nodes in the network. In general, the attacks to the MANETs can be classified into one of two classes: Denial of service (consumption of resources) or against the routing protocols. In this work we present the On-demand Secure Distance Vector Protocol (OSDV) which implements a series of security mechanisms aimed to prevent attacks against routing such as nodes reporting a fake shorter distance to the destination, a node pretending that he is the destination, nodes forging destination sequence numbers and a more subtle attack where nodes pretend that they are one hop closer to the destination (one-hop attacks). The security mechanisms implemented in OSDV are hash chains that are used to secure the mutable fields in the control signaling, digital signatures to secure the identity of the nodes as well as the immutable control fields, and time ordering to prevent one-hop attacks. All these techniques have been used in the past, but OSDV integrates all of them in a single protocol.

As shown by our simulation experiments, when using these techniques, the nodes that compose a MANET are capable of discovering, establishing and maintaining secure routes from sources to destination, even in the presence of a set of adversary nodes.

We performed a series of simulation experiments where we compare the performance of our protocol against that of AODV which is a de facto standard for routing in MANETs. Our experiments show that OSDV is much more resilient to attacks than AODV.

DEDICATORIA

A Dios por darme la capacidad y oportunidad de poder alcanzar una meta más en mi vida.

A mi novia y futura esposa Libia Miriam por brindarme su amor, comprensión y apoyo para alcanzar esta gran meta.

A mi padre Sebastián y mi madre Eustolia por quererme tanto y apoyarme en todo momento en mis decisiones.

A mis hermanos Juan, Ángel, Carlos, Rocío y Gabriel por su cariño y apoyo.

AGRADECIMIENTOS

Agradezco a Dios por acompañarme día a día e iluminarme en los momentos difíciles para salir adelante.

Agradezco a mi padre Sebastián, mi madre Eustolia y mis hermanos por creer en mí en todo momento y por apoyarme durante toda mi vida.

Gracias a mi novia Libia Miriam Martínez de la Rosa por apoyarme y creer en mí durante el desarrollo y culminación de este trabajo.

Un especial agradecimiento a mi director de tesis, el Dr. Rolando Menchaca Méndez, por su apoyo incondicional en la asesoría y dirección de esta tesis.

Agradezco a mi director de tesis, el Dr. Felipe R. Menchaca García, por su valioso tiempo y por haber brindado su apoyo incondicional en la culminación de este trabajo.

Agradezco a todos los profesores que me brindaron su valiosa enseñanza y su apoyo académico. ¡Gracias!

A mis amigos, por el apoyo y motivación que de ellos he recibido.

A los sinodales por su dedicación y apoyo en la revisión y recomendaciones del presente trabajo.

Índice

CAPÍTULO 3. Simuladores de eventos discretos

CAPÍTULO 4. Especificación del protocolo OSDV

Índice de figuras

Índice de tablas

Capítulo 1

Definición del problema

Introducción

Las redes móviles ad hoc (o MANET por sus siglas en ingles), son redes autoconfigurables que no necesitan ningún tipo de infraestructura para operar. Las MANETs están compuestas por nodos móviles que se comunican entre sí por medio de transmisiones inalámbricas. Este tipo de redes habilitan la comunicación entre dos nodos que posiblemente se encuentren fuera del rango de una transmisión inalámbrica directa, por lo que en un momento dado cualquier nodo puede actuar como emisor, receptor o enrutador de un flujo de datos.

Por su naturaleza descentraliza, y debido a que no necesitan de la participación de usuarios humanos en su configuración, las MANETs son altamente tolerantes a fallas en los nodos que las componen, así como a cambios en su topología. Estas características las convierten en alternativas ideales en situaciones donde no se cuente con una infraestructura preestablecida o donde la probabilidad de falla de los nodos individuales sea alta. Situaciones como éstas las encontramos en escenarios rurales, durante catástrofes naturales, en operaciones de rescate, así como en una amplia y creciente gama de aplicaciones ubicuas.

Por otro lado, y debido a la naturaleza inalámbrica de las comunicaciones, así como al hecho de que todos los nodos participan en labores de enrutamiento, este tipo de redes pueden sufrir una variedad de ataques que pueden clasificarse en alguna de las dos categorías siguientes: ataques contra el enrutamiento y ataques basados en el consumo de recursos. Estos tipos de ataques afectan el rendimiento, la funcionalidad y la fiabilidad de los protocolos que se encargan del enrutamiento para redes móviles ad hoc. Como se verá más adelante, los ataques a este tipo de redes se pueden llevar a cabo fácilmente, pero por el contrario, la implementación de un protocolo de enrutamiento para poder defenderse de ese tipo de ataques no es trivial.

Planteamiento del problema

En una red móvil de tipo Ad Hoc, para llevar a cabo una comunicación entre un nodo origen y un nodo destino, si el nodo destino está fuera del alcance del radio de propagación del nodo origen, se debe establecer una ruta de comunicación a través de otros nodos, los cuales actuarán como enlace entre el nodo origen y el nodo destino. Cada uno de los nodos que participan en la ruta que conecta al origen con el destino reenvían la información generada por el origen, de forma tal que ésta es recibida por el destino. Por la naturaleza de este tipo de redes, donde las comunicaciones son inalámbricas, y donde cualquier nodo puede formar parte de rutas que conectan a los orígenes con el destino, pueden ser vulnerables a diversos tipos de ataques, por ejemplo:

- Cuando un nodo atacante miente acerca de su identidad (e.g., pretende ser el destino).
- Cuando un nodo miente acerca de su distancia al destino (e.g., afirma tener una ruta más corta hacia el destino de la que en realidad tiene).
- Cuando un nodo atacante sólo reenvía información de control y la información de datos la desecha.
- Cuando un nodo atacante envía o desecha información de datos de manera aleatoria.
- Cuando un nodo atacante envía o desecha información de control de manera aleatoria.

Además de esos tipos de ataques se deben solucionar los problemas inherentes a la movilidad de los nodos de una MANET. Por ejemplo, las rutas se pueden romper continuamente debido a que un nodo intermedio ha dejado de estar activo o se ha movido a otra posición y por lo tanto será necesario calcular otra ruta hacia el destino. Por otro lado, debido a que los recursos disponibles para este tipo de redes son escasos (ancho de banda, energía de las baterías, etc.) es necesario implementar protocolos eficientes, que sean capaces de soportar múltiples flujos de comunicación de manera concurrente, es decir, que sean escalables.

Objetivo general:

Implementar un protocolo de enrutamiento para redes móviles ad hoc, que sea capaz de entregar paquetes de datos de un nodo origen a uno destino, aún y cuando existan nodos adversarios que mientan, ya sea a la hora de informar sobre su distancia hacia el destino o sobre su identidad. Adicionalmente, el protocolo debe ser escalable y hacer uso eficiente de los recursos de la red.

Objetivos específicos:

- Analizar el funcionamiento del protocolo de vector de distancias sobre demanda para redes ad hoc (AODV) y así identificar sus debilidades en cuanto a seguridad.
- Diseñar e implementar en el simulador NS-2 un algoritmo de enrutamiento que sea capaz de tolerar ataques por parte de nodos maliciosos.
- Diseñar e implementar en el simulador NS-2 un modelo de nodo malicioso.
- Diseñar e implementar en NS-2 una serie de escenarios de simulación para verificar de manera cuantitativa las propiedades del protocolo propuesto.
- Analizar los resultados experimentales para determinar las causas de los comportamientos mostrados por los diferentes protocolos de enrutamiento.

Justificación

Las aplicaciones móviles distribuidas como las utilizadas en operaciones militares, ayuda en caso de desastres o emergencias, redes que permiten la interacción de personas (como las usadas en reuniones de estudiantes durante conferencias), la seguridad en el proceso de enrutamiento es necesaria para proteger la red en contra de ataques implementados por nodos maliciosos.

Es de vital importancia tener protocolos de enrutamiento seguro para el uso en aplicaciones destinadas a operaciones militares y aplicaciones donde la seguridad en el proceso de enrutamiento sea un requerimiento.

Como se mencionó en la introducción, las MANETs son relativamente fáciles de atacar, por ejemplo un único nodo que publique una distancia 1 (medida en saltos) a todos los destinos tiene el potencial de inutilizar toda la red. Al publicar esta distancia en descubrimientos de nuevas rutas el nodo malicioso puede ocasionar que: se creen ciclos de enrutamiento, él tenga más posibilidades de ser incluido en la nueva ruta, y como consecuencia poder hacer mal uso de la información que fluye a través de él o eliminar tal información.

Debido a que el problema de la seguridad en la transmisión en las redes móviles ad hoc sigue siendo abierto, es indispensable desarrollar mecanismos que doten a las MANETs de tolerancia a dichos ataques, siendo fiables, funcionales y además que sean escalables.

Organización de la tesis

La presente tesis está organizada de la siguiente manera:

- Capítulo 1. Aquí se hace el planteamiento del problema, se esboza una solución y se justifica la realización de la solución.
- Capítulo 2. En este capítulo se hace un estudio relacionado al enrutamiento en redes móviles ad hoc (MANETs) en el que se analizan algunos de los ataques al desempeño del enrutamiento, así como los esquemas de seguridad para contrarrestar el impacto negativo en el desempeño de los protocolos de enrutamiento.
- Capítulo 3. Debido a que el protocolo propuesto es probado por medio de un simulador de eventos discretos, en este capítulo se presenta la teoría relacionada con los mismos. Se abordan los aspectos más importantes de una simulación de eventos discretos, así como la teoría tanto de diseño de experimentos como de análisis de los resultados. Además se describen los tipos de simulación, el análisis de la simulación y las técnicas para llevar a cabo la validación de un modelo de simulación.
- Capítulo 4. Aquí se presenta el diseño y la implementación del algoritmo de enrutamiento seguro y escalable propuesto, para redes móviles ad hoc.
- Capítulo 5. En este capítulo se hace un análisis de desempeño del algoritmo propuesto con base a los resultados de simulación.
- Capítulo 6. En este último capítulo se proporcionan las conclusiones finales y se realizan recomendaciones generales sobre el algoritmo realizado. Por último, se describen las líneas de investigación futuras que pueden basarse en este trabajo de tesis.

Capítulo 2

Enrutamiento seguro en redes móviles ad hoc (MANETs)

2.1 Estudio de seguridad en enrutamiento inalámbrico ad hoc

Las redes ad hoc usan nodos móviles para habilitar la comunicación entre dos nodos que posiblemente se encuentren fuera del rango de una transmisión inalámbrica directa. Ataques a protocolos de enrutamiento para redes Ad Hoc alteran el rendimiento y rentabilidad.

En una red móvil inalámbrica ad hoc (o MANET por sus siglas en inglés), nodos móviles cooperan entre sí para formar una red sin usar ningún tipo de infraestructura, como puntos de acceso o estaciones base. En su lugar, los nodos móviles reenvían paquetes entre ellos, permitiendo la comunicación entre nodos que posiblemente se encuentren fuera del rango de una transmisión inalámbrica directa. La movilidad de los nodos y la capacidad fundamentalmente limitada de los medios inalámbricos, junto con efectos de transmisión inalámbrica como la atenuación[1], la propagación multicamino[2] y la interferencia[3], se combinan para crear retos significativos para protocolos de enrutamiento operando en redes ad hoc.

Ejemplos de aplicaciones para las redes ad hoc van desde aplicaciones militares y ayuda en caso de desastres o emergencias, hasta redes que permiten la interacción de comunidades de personas como las reuniones de estudiantes durante una conferencia. En esas y otras aplicaciones de las redes ad hoc, la seguridad en el proceso de enrutamiento es necesaria para proteger la red, así como la información que fluye a través de ella en contra de ataques implementado por nodos maliciosos.

2.2 Ataques a redes ad hoc

Los ataques a protocolos de enrutamiento para redes ad hoc generalmente caen dentro de una de las dos categorías siguientes:

---

[1] Atenuación de la señal es la pérdida de potencia sufrida por la misma al transitar por cualquier medio de transmisión.

[2] Multicamino es el fenómeno de propagación que resulta en señales de radio al alcanzar la antena receptora por dos o más caminos.

[3] La interferencia es cualquier proceso que altera, modifica o destruye una señal durante su trayecto en el canal existente entre el emisor y el receptor. Típicamente el término se refiere a la agregación de una señal no deseada a la señal útil.

- Ataques contra el enrutamiento (routing-disruption attacks). El atacante intenta causar que los paquetes de datos legítimos, sean enrutados hacia caminos no funcionales.
- Ataques basados en el consumo de recursos (resource-consumption attacks). El atacante inyecta paquetes dentro de la red en un intento por consumir los recursos valiosos, como ancho de banda, memoria o poder de cómputo.

Desde una perspectiva de la capa de aplicación, ambos ataques son ejemplos de un ataque de negación de servicio (denial-of-service: DoS).

Un ataque de alteración de enrutamiento es llevado a cabo por un atacante al enviar paquetes de enrutamiento falsificados para crear un **ciclo de enrutamiento**, causando que los paquetes atraviesen nodos en un ciclo sin alcanzar sus destinos, consumiendo así energía y ancho de banda disponible. Un atacante podría similarmente crear un **hoyo negro de enrutamiento**, el cual atrae tráfico y desecha paquetes de datos. Un atacante crea un hoyo negro al distribuir información de enrutamiento (esto es, afirmar información falsificada de la distancia más corta); el atacante atrae tráfico y puede entonces descartarlo. Un caso especial de un hoyo negro, es cuando un atacante pudiera crear un **hoyo gris**, en el cual él selectivamente desecha algunos paquetes pero no otros, por ejemplo, al reenviar paquetes de enrutamiento pero no paquetes de datos. Un atacante también puede intentar causar que un nodo use una ruta desviada, o particionar la red al inyectar información de enrutamiento falsificada para impedir que un conjunto de nodos sea alcanzable desde otro. Un atacante podría intentar crear una ruta a través de él mismo, aparentando tener más capacidad al agregar nodos virtuales a la ruta. A este ataque se le conoce como *gratuituos detour* porque en realidad existe una ruta más corta que debería ser usada. En protocolos para redes ad hoc que intentan mantener rastro de los nodos que han sido identificados como maliciosos en una lista negra en cada nodo (como es hecho en los protocolos watchdog y pathrater[1]), un atacante podría volver malo a un nodo bueno, causando que otros nodos buenos agreguen a ese nodo a sus listas negras y así evitar la participación de dicho nodo en rutas futuras.

Un **ataque de apresuramiento** (rushing attack)[2] es un ataque enfocado en contra de protocolos de enrutamiento sobre demanda[3]. La mayoría de estos protocolos utilizan un esquema de eliminación de paquetes duplicados. En este caso, un atacante difunde paquetes de solicitud de ruta (ROUTE REQUESTS) rápidamente a través de la red, reprimiendo cualquier ROUTE REQUEST posterior legítimo debido a la saturación generada por los de ROUTE REQUESTS recibidos con anticipación.

Un tipo más sutil de ataque de alteración de enrutamiento es crear un **hoyo de gusano en la red**, usando un par de nodos atacantes A y B enlazados por medio de una conexión privada de red.

El ataque de hoyo de gusano es un ataque grave en contra de los protocolos de enrutamiento ad hoc ya que es particularmente difícil poder defenderse contra él. En el ataque de hoyo de gusano un atacante graba paquetes (o bits) en un lugar de la red, los tunelea a otros lugares de la red y los retransmite desde ahí dentro de la red. La mayoría de los protocolos de red ad hoc existentes que carecen de un mecanismo para defenderse en contra del ataque de hoyo de gusano serían incapaces de encontrar rutas más grandes que uno o dos saltos, los cuales alteran gravemente la comunicación. Si un atacante de hoyo de gusano tunelea todos los paquetes a través del hoyo de gusano honesta y seguramente, no lleva a cabo ningún daño. Sin embargo, cuando un atacante reenvía sólo mensajes de control de enrutamiento, este ataque podría alterar gravemente el enrutamiento. Por ejemplo, cuando es usado en contra de un protocolo de enrutamiento sobre demanda como DSR[4] o AODV[5], una aplicación poderosa del atacante de hoyo de gusano puede ser montada al tunelear cada paquete de ROUTE REQUEST directamente hacia el nodo objetivo del REQUEST. Este ataque impide que cualquier nodo descubra rutas no más grandes de dos saltos de longitud.

Los protocolos periódicos son vulnerables a este tipo de ataque. Por ejemplo OLSR[6] y TBRPF[7] usan paquetes HELLO para detección de vecinos, entonces si un atacante tunelea a B todos los paquetes HELLO transmitidos por A y tunelea a A todos los paquetes HELLO transmitidos por B, entonces A y B creen que ellos son vecinos, lo cual causaría que el protocolo falle en encontrar rutas cuando en realidad ellos no son vecinos.

El ataque de hoyo de gusano es además peligroso en otras aplicaciones inalámbricas. Un ejemplo es en un sistema de control de acceso inalámbrico que está basado en proximidad, como llaves de carros inalámbricas o sistemas de control de acceso basados en proximidad para PCs. En tales sistemas, un atacante podría reenviar el intercambio de autenticación para obtener acceso no autorizado.

2.2.1 Modelo del atacante.

El modelo del atacante consiste de dos principales clases de **atacantes, pasivos y activos**. Un atacante pasivo no envía mensajes, sólo escucha la información transmitida por la red. Los atacantes pasivos son principalmente amenazas en contra de la privacidad en la

comunicación, más que en contra de la funcionalidad de la red o protocolo de enrutamiento. Y por lo tanto no se discutirá más a fondo aquí.

Un atacante activo inyecta paquetes dentro de la red (y también escucha la información transmitida por la red). A esta clase de atacantes se les caracteriza con base al número de nodos que han comprometido. Se asume que el atacante posee todas las llaves criptográficas de información de nodos comprometidos y las distribuye entre todos sus nodos. Se denota como un atacante-n-m, donde n es el número de nodos que ha comprometido y m es el número de nodos que posee. Se propone la siguiente jerarquía del atacante para medir la seguridad en los protocolos de enrutamiento: Activo-0-1 (el atacante posee un nodo), activo-0-x (el atacante posee x nodos), activo-1-x (el atacante posee un nodo comprometido y distribuye las llaves criptográficas a sus x-1 nodos), y activo-y-x. Adicionalmente se llama a un atacante que ha comprometido nodos como atacante ActivoVC si él posee todos los nodos localizados en un corte que particiona a los nodos en múltiples conjuntos, forzando a estos últimos a permanecer en particiones diferentes para que se comuniquen sólo a través de un nodo atacante. Este ataque es particularmente poderoso porque el atacante controla todo el tráfico entre nodos de particiones disjuntas.

El modelo que se asumirá en los experimentos para la evaluación del protocolo es activo-x-0, ya que un nodo atacante puede comprometer a todos los nodos de la red, pero no posee algún nodo.

2.3 Esquemas de seguridad para MANETs

2.3.1 Lazos de paquetes

La solución al ataque de hoyo de gusano son los **lazos de paquetes** (packet leashes)[8]. Se considera específicamente dos tipos de lazos de paquetes: geográfico y temporal. La idea principal es que al autenticar o un sello de tiempo extremadamente preciso o información de localización combinada con sello de tiempo holgado, un receptor puede determinar si un paquete ha atravesado una distancia irreal dadas tecnologías específicas para acceder a la red. Sin embargo, los lazos de paquetes dependen de tiempo de sincronización extremadamente precisos y sellos de tiempo en cada paquete. Se puede aproximar el tiempo de viaje de un paquete como la diferencia entre el tiempo de la recepción y el sello

de tiempo. Siendo más cuidadosos, un nodo puede escoger sumar el tiempo máximo de error de sincronización, asumiendo que el reloj del emisor pudiera ser más rápido que el del receptor. Opuestamente, para permitir todas las comunicaciones directas entre nodos legítimos, un nodo puede restar el tiempo máximo de error de sincronización, asumiendo que el reloj del emisor pudiera ser más lento que el del receptor.

Dados los tiempos de sincronización precisos requeridos por los lazos temporales, se construyeron algunos autenticadores de transmisión eficientes basados enteramente en primitivas simétricas. En particular se extiende el protocolo de autenticación de transmisión para la autenticación en tiempo eficiente de flujo tolerante a pérdida (Tesla)[9] para permitir la revelación de la llave de autenticación dentro del paquete autenticado. Se usa un árbol Merkle[10] para autenticar esas llaves. Esa investigación mostró que con este mecanismo de autenticación, actualmente los dispositivos disponibles pueden soportar fácilmente líneas de velocidad en autenticación de lazos temporales.

Otro método para construir un lazo es usar información de localización y relojes ligeramente sincronizados. Se llamó a tales lazos, lazos geográficos. Si los relojes del emisor y receptor son sincronizados dentro de $\pm\Delta$, y v es el límite superior sobre la velocidad de cualquier nodo. Así, el receptor puede calcular el límite superior sobre la distancia entre el emisor y él mismo d_{sr}. Basado específicamente en, el sello de tiempo t_s en el paquete el tiempo local del receptor t_r, el error relativo máximo en la localización de la información δ, y las localizaciones de los receptores p_r y el emisor p_s, d_{sr} puede ser limitado por la Ecuación 1

$$d_{sr} \leq |p_s - p_f| + 2v \cdot (t_r - t_s + \Delta) + \delta \quad (1)$$

En ciertas circunstancias, limitar la distancia entre el emisor y el receptor d_{sr} no puede prevenir el ataque de hoyo de gusano; por ejemplo, cuando existen obstáculos que evitan la comunicación entre nodos que de otra manera estarían en rango de transmisión, un escenario basado en distancias todavía permitiría el ataque de hoyo de gusano entre el emisor y el receptor. Una red que usa información de localización como un lazo, puede controlar incluso clases de hoyo de gusanos. Un receptor verifica que cada posible localización del emisor (un radio dado por la Ecuación 2 alrededor de p_s) pueda ser alcanzable por cada posible receptor de localización (un radio dado por la Ecuación 2 alrededor de p_r).

$$\delta + v(t_r - t_s + 2\Delta) \quad (2)$$

2.3.2 Administración de Llaves Públicas y Privadas

2.3.2.1 Configuración de llaves en redes ad hoc

En muchas redes ad hoc, el compromiso de una simple red de nodos y la captura de sus llaves criptográficas es una amenaza evidente. Intuitivamente, un único nodo comprometido es menos poderoso que numerosos nodos comprometidos.

Para lograr este nivel más alto de seguridad en contra de únicos nodos comprometidos, sería posible que el protocolo de enrutamiento distinga entre varios nodos legítimos. Tales nodos pueden ser distinguidos a través del uso de autenticación, pero una única llave compartida no puede proveer autenticación; en su lugar, cada nodo legítimo debe poseer una o más llaves únicas para ese nodo. Adicionalmente, cada nodo debe tener una manera de autenticar a un nodo legítimo. Cómo colocar información de llaves de autenticación, es el problema de la configuración de llave, e investigadores han propuesto un número de soluciones para este problema vistas en las siguientes secciones.

2.3.2.2 Establecer llaves privadas.

Algunos protocolos de enrutamiento para redes ad hoc requieren llaves privadas compartidas entre todos los pares de nodos en la red. La distribución de llaves privadas es substancialmente más difícil que la distribución de llaves públicas porque protocolos para la distribución de llaves deben asegurar que las llaves sean secretas.

La manera obvia de distribuir las llaves privadas es compartirlas con cada par de nodos antes de su utilización, cuando se sabe que todos los nodos se están comportando correctamente. Esta propuesta es más difícil cuando se desea incrementar el uso de nodos en la red. Frank Stajano y Ross Anderson[11] proponen un escenario para establecer confiabilidad y llaves entre dos nodos en una red ad hoc; en su modelo *resurrecting duckling*, dos nodos son ligados para conseguir que un nodo esclavo se una a un nodo maestro. Una vez que una llave es intercambiada a través de su capa física, esa llave puede ser usada para encriptar y auntenticar información posterior, como una lista de llaves compartidas.

Si las llaves públicas han sido establecidas, se pueden establecer llaves privadas al usar un protocolo de intercambio de llaves. Se pueden usar tales llaves para verificar mensajes autenticados pero no para generar autenticadores falsos.

2.3.2.3 Evitar el problema de administración de llaves

Una propuesta para resolver el problema de administración de llaves es asumir que cada nodo tiene una lista de llaves públicas legítimas. Esta propuesta es la más simple; sin embargo, asume que todos los nodos confían en un conjunto común de autoridades y que cada nodo puede descargar una lista de nodos legítimos antes de su utilización.

Otro problema con esta propuesta concierne al incremento de la utilización de la red. Si no se hace uso de los nodos de la red que están cerca simultáneamente, entonces un nodo podría ser usado antes de que un futuro nodo pueda proveer sus llaves a la autoridad. En este caso, la autoridad necesitaría generar llaves para nodos futuros. Cuando un nodo quiere unirse a la red, recibe la lista de nodos legítimos así como su propia llave privada. En este caso, el canal sobre el cual el nodo recibe la llave privada debe ser seguro en contra de escuchadores ocultos. Ordinariamente, el canal sobre el cual el nodo recibe la lista de nodos legítimos necesitaría ser seguro sólo en contra de ataques activos; tal seguridad sería provista al usar un lado del canal para checar la igualdad de una función hash criptográfica calculada sobre la lista.

2.3.2.4 Direcciones SUCV

Gabriel Montenegro y Claude Castellucia proponen que cada nodo escoja una dirección basada en su llave pública[12]. En esta propuesta, llamada direcciones estadísticamente únicas criptográficamente verificables (SUCV, Statically Unique Criptographically verifiable), en el que cada nodo genera un par de llaves públicas y privadas, y entonces escoge su dirección basada en una función hash criptográfica de la llave pública. Los autores propusieron dos escenarios: uno en el cual las direcciones IPv6 del nodo fueron las funciones hash de salida y otro en el cual los 64 bits menos significativos fueron la salida de la función hash. Cada nodo genera su propio par de llave pública y privada. Como resultado, el nodo puede encontrar otra llave a la que aplica la función hash a los mismos 64 bits de dirección en 2^{32} veces. De esta forma cualquier otro nodo debe hacer 2^{63} operaciones sobre la media para encontrar una colisión.

Desafortunadamente, esta propuesta no resuelve completamente el problema de la configuración de llaves. En particular, en una red sin SUCV, el problema es tener una lista de pares legítimos (nodo, llave pública); en una red usando SUCV, el problema es tener una lista de nodos legítimos.

2.3.2.5 Certificados de una autoridad certificadora.

Otra propuesta para evitar el problema de distribución de llaves, es definir una o más autoridades certificadoras (CAs). Cada nodo en la red tiene un certificado que incluye su dirección de nodo, su llave pública, y una firma de la CA (figura 2-1). Si cada nodo incluye su certificado cada vez que firme un mensaje, el receptor puede primero verificar el certificado, y entonces usar la llave pública en el certificado para checar la firma (figura 2-2). Optimizaciones deben permitir al protocolo incluir el certificado menos frecuentemente.

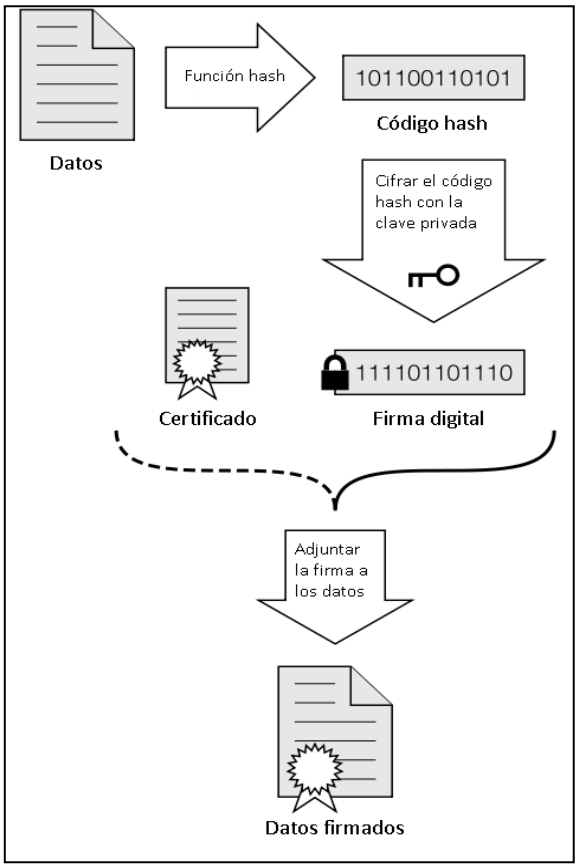


Figura2-1. Firmar un mensaje

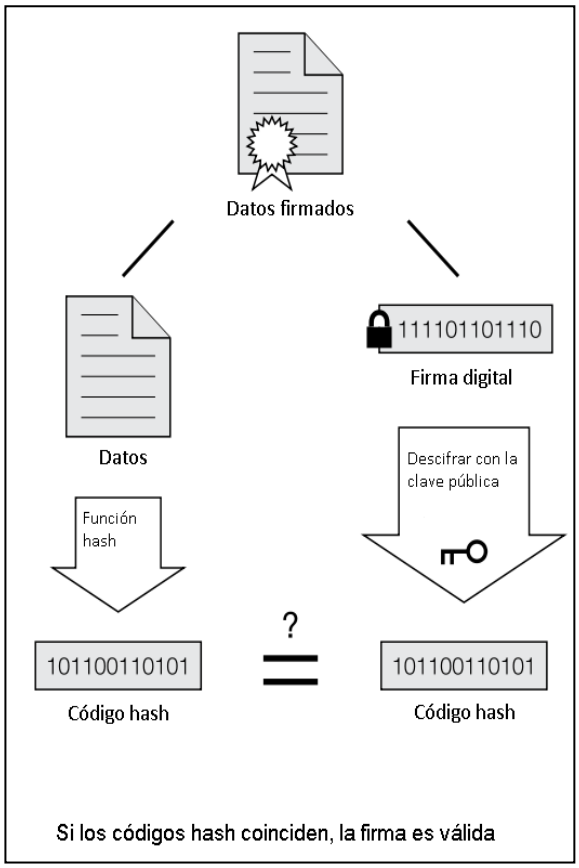


Figura 2-2. Comprobación de la firma digital

Las autoridades certificadoras pueden estar en línea o fuera de línea. La diferencia entre esos tipos es si ellas son alcanzables a través de la red. Debido a que una CA en línea es alcanzable a través de la red, la CA puede participar en el protocolo de verificación de certificado. Sin embargo, incluso con un CA en línea, el protocolo de verificación de certificado no es simple porque hay una dependencia circular entre seguridad y enrutamiento: para correr el protocolo de seguridad, se deben establecer rutas a la CA, pero para establecer esas rutas, primero debe estar en funcionamiento un protocolo de

enrutamiento. En Ariadne[18], se rompe esta dependencia circular al usar la CA, la cual ya es confiable, para establecer esas rutas.

Una CA en línea a menudo es vulnerable para efectuar concesiones. Lidong Zhou y Zygmunt J. Haas proponen usar umbral de criptografía para distribuir la funcionalidad de la CA a varios nodos[13]. Una manera simple de lograr ésto es confiar sólo en llaves que tengan certificados de varias CAs. Esta técnica permite efectuar concesiones de uno o dos nodos que participan en la certificación, sin permitir a un atacante certificar un número arbitrario de nuevos nodos. Podemos usar umbral de criptografía para proveer esta misma semántica, la cual tiene la ventaja de proveer un certificado más pequeño. Seung Yi y Robin Kravets proveen una discusión más completa de protocolos que usan CAs en redes ad hoc[14].

2.3.2.6 Confiabilidad transitiva y grafos confiables pgp

Investigadores han propuesto una alternativa de modelo confiable para redes ad hoc sin una CA en línea. En este modelo cada nodo firma certificados para otros nodos[15]. Un nodo puede buscar en la red una cadena de certificados siguiendo desde el nodo que inicia la pregunta y finalizando en el nodo que intenta autenticar un mensaje. Generalmente, tales escenarios requieren de confiabilidad transitiva –eso es, si A confía en B, y B confía en C, entonces tales escenarios requieren que A confíe en C.

Este requerimiento puede ser un poco ligero al requerir caminos adicionales de nodos disjuntos y limitar la longitud de la cadena. Si existen múltiples caminos de nodos disjuntos entre el firmador y el verificador, entonces un atacante habría tenido que negociar con un nodo en cada camino para insertar una llave falsa. Cuando la longitud del certificado es limitado, el impacto de un nodo comprometido puede ser limitado. Por ejemplo, una red puede limitar cadenas de certificados para un firmador intermedio. En este caso, si ningún vecino de A es comprometido, entonces un atacante no puede generar un certificado falso que A aceptaría.

2.3.2.7 Anulación de llave pública

Idealmente, cuando un nodo está comprometido, alguna autoridad puede anular el certificado para su llave pública. Por lo tanto, cualquier sistema necesita protegerse en contra de un atacante que intenta anular las llaves de nodos legítimos.

Colocar esas anulaciones es un problema incluso más difícil que el problema de configuración de llaves. Una manera para revocar certificados es usar una CA en línea para firmar certificados negativos. Sin embargo un nodo generalmente no distribuye su propio certificado negativo. Como resultado se necesita un mecanismo de distribución alternativo. Una propuesta es en la que una lista negra u otra anulación de información pueden ser inundadas a través de la red cuando se descubre un certificado negativo. Con esta propuesta, se necesita alguna limitación de inundación, o un atacante puede conseguir que su certificado sea anulado y repetidamente inundar la red con su información anulada. Como resultado, se debe verificar que sólo información de anulación nueva sea inundada.

2.3.3 Seguridad en enrutamiento

2.3.3.1 SEAD en redes móviles inalámbricas ad hoc.

El protocolo de enrutamiento con seguridad eficiente para vector de distancias ad hoc (SEAD)[16] es resistente en contra de múltiples atacantes no coordinados creando estados de enrutamiento incorrectos en cualquier otro nodo, a pesar de que existan atacantes activos o nodos comprometidos en la red. El diseño de SEAD está basado en parte sobre el protocolo de enrutamiento de números secuenciales del destino con vector de distancias en una red ad hoc (Destination Sequence-Number Distance Vector, DSDV)[17]. Para apoyar el uso de SEAD en nodos de CPU limitada, y defenderse en contra de ataques del tipo DoS (negación de servicio) en el cual un atacante intenta causar que otros nodos consuman excesivo ancho de banda o tiempo de procesamiento, el protocolo usa funciones hash eficientes y no se usan operaciones de criptografía asimétrica que son altamente costosas en términos de tiempo de ejecución.

2.3.3.2 Enrutamiento con vector de distancias

En un enrutamiento con vector de distancias, cada ruteador mantiene una tabla de enrutamiento listando todos los posibles destinos dentro de la red. Cada entrada en una tabla de enrutamiento de un nodo contiene la dirección (identidad) de algún destino, la distancia más corta conocida de un nodo (llamada métrica, usualmente en número de saltos) al destino, y la dirección de enrutamiento de un nodo vecino, que es el primer salto en la ruta más corta al destino. La distancia al destino es una métrica en la tabla de

entradas. Cada ruteador reenvía un paquete el cual usa su propia tabla de enrutamiento para determinar el siguiente salto el destino.

Para mantener las tablas de enrutamiento, cada nodo transmite periódicamente una actualización de enrutamiento que contiene la información de su propia tabla de enrutamiento. Cada nodo actualiza su propia tabla usando las actualizaciones que él escucha, así su ruta para cada destino usa como siguiente salto el vecino que dijo tener la métrica más pequeña en su actualización para ese destino. El nodo coloca la métrica en su tabla de entrada para ese destino a un salto más que la métrica de la actualización de ese nodo vecino.

La mejora primaria para redes ad hoc hecha en DSDV sobre vector de distancias de enrutamiento estándar es la adición de un número de secuencia en cada tabla de entradas de enrutamiento. Usando este número de secuencia impide ciclos de enrutamiento causados por actualizaciones que son aplicadas cuando están fuera de funcionamiento. Este problema puede ser común sobre transmisiones inalámbricas con multisaltos porque la información de enrutamiento puede esparcirse a lo largo de muchos caminos a través de la red.

2.3.3.3 Cadenas Hash

Una cadena hash de sólo un camino es construida sobre una función hash de sólo un camino (sin inversa). Como una función normal, una función hash de sólo un camino *H* mapea una salida de cualquier longitud a una cadena de bits de longitud compuesta. Así, en la Ecuación 3 ρ es la longitud en bits de la salida de la función hash. La función *H* debe ser simple de calcular y aún así, ser computacionalmente imposible de invertir.

$$H: \{0,1\}^* \rightarrow \{0,1\}^\rho \quad (3)$$

Para crear una cadena hash de sólo un camino, un nodo escoge un número aleatorio (Ecuación 4) y calcula la lista de valores mostrada en la Ecuación 5

$$x \in \{0,1\}^\rho \quad (4)$$

$h_0, h_1, h_2, h_3, \ldots, h_n,$ (5)
donde $h_0 = x$, *y* $h_i = H(h_{i-1})$ *para* $0 < i \leq n$, *para algún n.*

El nodo en la inicialización genera los elementos de la cadena hash usando la recursividad, dicho nodo usa ciertos elementos de la cadena para asegurar sus actualizaciones de enrutamiento.

Dado un elemento autenticado existente de una cadena hash de sólo un camino, se pueden verificar elementos posteriores en la secuencia de uso dentro de la cadena. Por ejemplo, dado un valor h_i autenticado, un nodo puede autenticar h_{i-3} al calcular $H\left(H\left(H(h_{i-3})\right)\right)$ y verificar que el valor resultante es igual a h_i. Para usar cadenas hash de sólo un camino para autenticación, se asume algún mecanismo para que un nodo distribuya un elemento autentico como h_n desde su cadena hash generada.

2.3.3.4 Autenticar actualizaciones de enrutamiento (SEAD)

Cada nodo en SEAD (Secure Efficient Ad hoc Distance vector routing protocol) usa un elemento siguiente único específico de su cadena hash en cada actualización de enrutamiento que él envía a él mismo (métrica 0). Basada en su elemento inicial, la cadena hash de sólo un camino conceptualmente provee autenticación para el límite inferior de la métrica en otras actualizaciones de enrutamiento para este destino. La autenticación provee sólo un límite inferior en la métrica –es decir, un atacante puede incrementar la métrica o afirmar la misma métrica, pero no puede decrementar la métrica.

Se asume que el operador de red puede colocar un límite superior sobre el diámetro de la red, y se usa $m-1$ para denotar este límite. Para autenticar un registro en una actualización de enrutamiento en la que se usa el número secuencial de tal registro para determinar un grupo contiguo de *m* elementos y que están contemplados en la cadena hash del destino, un elemento el cual debe ser usado para autenticar la actualización de la ruta. . El elemento particular de este grupo de elementos que debe ser usado para autenticar el registro se determina por el valor de la métrica que se está enviando en cada registro. Específicamente, si una cadena hash de un nodo es la secuencia de valores $h_0, h_1, h_2, h_3, \ldots, h_n$ y *n* es divisible por *m*, entonces para un número secuencial *i* en algún registro de actualización de enrutamiento, sea un elemento (dado por la Ecuación 6) del grupo de elementos h_{km}, $h_{km-1}, \ldots, h_{km+m-1}$ de esta cadena hash se usa para autenticar el registro; si el valor del registro para esta métrica es $j, \text{donde } 0 \leq j < m$, entonces el valor h_{km-j} se usa para autenticar el registro de actualización de

enrutamiento para ese número secuencial. Los nodos que reciben cualquier actualización de enrutamiento pueden autenticar fácilmente cada registro en la actualización, dado cualquier elemento hash cercano autentico de la misma cadena hash.

$$k = \frac{n}{m} - i\,. \tag{6}$$

2.3.3.5 Protocolo de enrutamiento seguro sobre demanda para redes ad hoc (ARIADNE)

Ariadne[18] es un protocolo de enrutamiento seguro bajo demanda que contrarresta nodos comprometidos y depende sólo de criptografía simétrica eficiente. Ariadne puede autenticar mensajes de enrutamiento usando uno de tres modelos:

- Secretos compartidos entre cada par de nodos.
- Secretos compartidos entre nodos comunicadores combinados con autenticación de transmisión.
- Firmas digitales.

En este apartado se describe a Ariadne usando Tesla[19] (autenticación en tiempo eficiente de flujo tolerante a pérdida), un modelo eficiente de autenticación de transmisión que requiere tiempos de sincronización. Al usar pares de llaves compartidas, evita la necesidad de tiempo de sincronización pero con costo elevado en la configuración de llaves. La idea básica de Ariadne está basada en DSR (Dynamic Source Route): Ariadne descubre rutas bajo demanda a través de un descubrimiento de ruta y las usa para que la fuente envíe paquetes de datos a sus destinos. Cada nodo retransmisor ayuda al utilizar el mantenimiento de ruta para descubrir problemas con cada ruta seleccionada.

2.3.3.5.1 Descubrimiento de rutas de Ariadne básico

El diseño del protocolo Ariadne se presenta en dos fases: primero se presenta un mecanismo que permite al nodo destino verificar la autenticidad del ROUTE REQUEST y entonces utilizar una técnica eficiente la cual consiste en comprobar cadenas hash para cada salto con el fin de verificar que ningún nodo falte en la lista de nodos contenida en el REQUEST. Después, se asume que el origen S utiliza un descubrimiento de ruta para un objetivo D y que ellos comparten las llaves secretas K_{SD} y K_{DS}, respectivamente, para la autenticación de mensajes en cada dirección.

El destino autentica peticiones de ruta. Para validar la legitimidad de cada campo en un ROUTE REQUEST, el origen simplemente incluye un código de autenticación de mensaje (MAC) calculada con la llave K_{SD} sobre datos únicos –por ejemplo, un sello tiempo. El destino puede verificar fácilmente la autenticidad y que tan reciente es una petición de ruta usando la llave compartida K_{SD}.

En un descubrimiento de ruta, el origen debe autenticar cada nodo individual en la lista de nodos del ROUTE REPLY. Un requerimiento secundario es que el objetivo pueda autenticar cada nodo en la lista de nodos del ROUTE REQUEST, así él regresará un ROUTE REPLAY sólo a través de caminos que contienen nodos legítimos. Cada salto autentica la nueva información en el REQUEST usando su llave Tesla actual. El destino almacena el REPLY hasta que nodos intermedios puedan liberar las correspondientes llaves Tesla. La condición de seguridad Tesla es verificada en el destino que incluye un código de autenticación de mensaje (MAC) en el REPLY para certificar que la condición de seguridad fue conocida.

S:	$h_0 = MAC_{K_{SD}}(REQUEST, S, D, id, ti)$
$S \rightarrow^*$:	$\langle REQUEST, S, D, id, ti, h_0, (\,), (\,)\rangle$
A:	$h_1 = H[A, h_0]$
	$M_A = MAC_{K_{A_{ti}}} REQUEST, S, D, id, ti, h_1, (A), (\,)$
$A \rightarrow^*$:	$REQUEST, S, D, id, ti, \boldsymbol{h_1}, (\boldsymbol{A}), \boldsymbol{M_A}$
B:	$h_2 = H[B, h_1]$
	$M_B = MAC_{K_{B_{ti}}}(REQUEST, S, D, id, ti, h_2, (A, B), (M_A))$
$B \rightarrow^*$:	$REQUEST, S, D, id, ti, \boldsymbol{h_2}, (A, \boldsymbol{B}), (M_A, \boldsymbol{M_B})$
C:	$h_3 = H[C, h_2]$
	$M_C = MAC_{K_{C_{ti}}}(REQUEST, S, D, id, ti, h_3, (A, B, C), (M_A, M_B))$
$C \rightarrow^*$:	$REQUEST, S, D, id, ti, \boldsymbol{h_3}, (A, B, \boldsymbol{C}), (M_A, M_B, \boldsymbol{M_C})$
D:	$M_D = MAC_{K_{DS}}(REQUEST, D, S, ti, (A, B, C), (M_A, M_B, M_C))$
$D \rightarrow C$:	$REPLY, D, S, ti, (A, B, C), (M_A, M_B, M_C), \boldsymbol{M_D}, (\,)$
$C \rightarrow B$:	$REPLY, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (\boldsymbol{K_{C_{ti}}})$
$B \rightarrow A$:	$REPLY, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, \boldsymbol{K_{B_{ti}}})$
$A \rightarrow S$:	$REPLY, D, S, ti, (A, B, C), (M_A, M_B, M_C), M_D, (K_{C_{ti}}, K_{B_{ti}}, \boldsymbol{K_{A_{ti}}})$

Figura 2-3. Ejemplo de descubrimiento de ruta en Ariadne. El nodo origen S está intentando descubrir una ruta hacia el nodo destino D. La letra en negrita indica los campos modificados en los mensajes, relacionados a los mensajes previos de ese tipo.

Aplicar la función hash por salto. La autenticación de datos en mensajes de enrutamiento no es suficiente porque un atacante podría eliminar un nodo de la lista de nodos en un REQUEST. Se usan funciones hash de sólo un camino para verificar que ningún salto fue omitido. A esta propuesta se le llamó aplicar la función hash por punto. Para cambiar o eliminar un salto previo, un atacante debe o escuchar un REQUEST o ser capaz de invertir la función de sólo un camino. Por razones de eficiencia, se puede incluir el autenticador en el valor de hash pasado en el REQUEST. La figura 2-3 muestra un ejemplo de descubrimiento de ruta en Ariadne.

2.3.3.5.2 Mantenimiento de rutas de Ariadne básico

EL mantenimiento de rutas en Ariadne está basado en DSR. Un nodo que está reenviando un paquete al siguiente salto a lo largo de la ruta original, regresa un ROUTE ERROR al emisor original del paquete si es incapaz de entregar el paquete al siguiente salto después de un número limitado de intentos de retransmisión.

Para impedir que nodos no autorizados envíen ERRORS, se requiere que el emisor autentique un ERROR. Cada nodo en el camino de regreso a la fuente reenvía el ERROR. Si la autenticación está retrasada –por ejemplo, cuando se usa Tesla– cada nodo almacena el ERROR hasta que sea capaz de autenticarlo.

2.3.3.5.3 Evitar comportamiento indeseado en el enrutamiento

El protocolo Ariadne, como se describió anteriormente, es vulnerable a un atacante que forma parte de la ruta descubierta. Para evitar el uso de rutas maliciosas, se eligen rutas basándose en su eficiencia previa con respecto a la entrega de paquetes. El modelo depende de observaciones acerca de cuáles paquetes fueron entregados exitosamente. Las observaciones pueden ser recibidas o a través de un mensaje extra de la capa de red extremo-a-extremo o al explotar las propiedades de las capas de transporte, tal como lo hace TCP con avisos selectivos (selective acknowledgements)[20]. Esta propuesta de reacción es algo similar a la usada en IPv6 para la detección de nodos inalcanzables[21].

Un nodo con múltiples rutas a un único destino puede asignar una fracción de paquetes que él origina para ser enviados a lo largo de cada ruta. Cuando una fracción substancialmente más pequeña de paquetes enviados a lo largo de cualquier ruta particular es entregada exitosamente, el nodo puede iniciar enviando una fracción más pequeña de su total de paquetes a ese destino a lo largo de esa ruta.

2.3.3.6 Seguridad ligera para DSR

Panagiotis Papadimitratos y Zygmunt Haas[22] proponen el protocolo de enrutamiento seguro (SRP), el cual se puede usar con DSR o con el protocolo de enrutamiento en zonas zonas (ZRP)[23]. Los autores diseñaron SRP como una extensión de encabezado que es adherido a los paquetes de ROUTE REQUEST y ROUTE REPLAY. Debido a que el protocolo SRP no asegura los paquetes ROUTE ERROR usados en el mantenimiento de rutas, éste comisiona a la parte de mantenimiento de ruta segura del protocolo de transmisión de mensajes seguros[24], la función de mantenimiento la ruta. SRP usa un número de secuencia en el REQUEST para asegurar frescura, pero este número de secuencia sólo puede ser checado en el destino. SRP requiere una asociación de seguridad sólo entre nodos de comunicación y usa esta asociación de seguridad sólo para auntenticar ROUTE REQUESTS y ROUTE REPLAYS por medio del uso de códigos de mensajes de autenticación. En el destino, SRP puede detectar la modificación de los ROUTE REQUEST, y en la fuente, SRP puede detectar la modificación de los ROUTE REPLY.

Sin embargo, SRP no intenta impedir la modificación de campos no autorizados que son ordinariamente modificados en el transcurso de reenvío de estos paquetes. Por ejemplo, un nodo puede eliminarse libremente o malear la lista de nodos de un paquete ROUTE REQUEST que él reenvía.

Debido a que SRP requiere una asociación de seguridad sólo entre nodos que se comunican, él usa mecanismos ligeros para evitar otros ataques. Por ejemplo, para limitar la inundación, los nodos graban la tasa en la cual cada nodo reenvía paquetes ROUTE REQUEST y da prioridad a paquetes REQUEST. Tales mecanismos pueden asegurar un protocolo cuando pocos atacantes están presentes; sin embargo, tales técnicas proveen ataques secundarios, como el enviar paquetes ROUTE REQUEST falsificados para reducir la efectividad de un ROUTE REQUEST autentico de un nodo. Adicionalmente, tales técnicas agravan el problema de nodos glotones. Por ejemplo, un nodo que no reenvía paquetes ROUTE REQUEST ordinariamente logra mejor rendimiento porque está generalmente menos congestionado, y no necesita usar el poder de su batería para reenviar paquetes originados por otros nodos. En SRP, un nodo glotón conserva esas ventajas y consigue una prioridad mayor cuando inicia un descubrimiento de ruta.

SRP autentica ROUTE REPLAYS de nodos intermedios usando grupos de llaves compartidas o firmas digitales. Cuando un nodo con una ruta guardada comparte un grupo de llaves con (o puede generar una firma digital verificable por) el origen del REQUEST, puede usar ese un grupo de llaves para autenticar los mensajes REPLAY. El autenticador, es un código de mensaje de autenticación calculado usando el grupo de llaves o una firma y es

conocida como la señal de respuesta del nodo intermedio. La firma o MAC (código de mensaje de autenticación) es calculada durante el almacenamiento del REPLY.

Como se mencionó anteriormente, SRP no intenta direccionar la solicitud del mantenimiento de ruta. En SRP, como en Ariadne, múltiples REPLYS son regresados para cada REQUEST; los nodos usan transmisión de mensajes seguros (SMT)[24] para asegurar la entrega exitosa de paquetes. En SMT, los mensajes de datos son divididos en paquetes usando técnicas de compartición, así que si M es la salida de N entonces tales paquetes son recibidos y el mensaje puede ser reconstruido.

2.3.3.7 Enrutamiento autenticado para redes ad hoc

Kimaya Sanzgiri et-al, desarrollaron el protocolo de enrutamiento autenticado para redes ad hoc (ARAN)[25], el cual está basado en AODV. En ARAN, cada nodo tiene un certificado firmado por una autoridad confiable, el cual asocia su dirección de IP con una llave pública. ARAN es un protocolo bajo demanda, dividido en descubrimiento de rutas y mantenimiento.

La Figura 2-4 muestra un ejemplo de descubrimiento de rutas en ARAN. Para iniciar un descubrimiento de ruta, el origen (S, en este ejemplo) transmite un paquete ROUTE REQUEST firmado que incluye el destino (D en este ejemplo), su certificado $cert_S$, un número N que se usa sólo una vez, y un sello de tiempo t. El número N y el sello de tiempo juntos aseguran frescura cuando son usados en una red con corrimiento de reloj limitado (limited clock skew). Cada nodo que reenvía este REQUEST checa la firma o firmas. En el ejemplo, el nodo C checa el certificado $cert_B$ de B, entonces checa la firma del mensaje saliente. C entonces verifica el certificado $cert_S$ del origen S y usa la llave en el certificado para verificar la firma en el REQUEST. Si las firmas (o firma, cuando el paquete es directamente recibido del origen) son válidas, un nodo intermedio elimina la última firma y certificado del nodo anterior (si aplica), firma el REQUEST original, e incluye su propio certificado. El nodo actual transmite el REQUEST. En el ejemplo, el nodo C elimina la firma del nodo B, firma el REQUEST resultante, e incluye su propio certificado. El nodo C entonces transmite el REQUEST.

Cuando el primer ROUTE REQUEST de un descubrimiento de ruta alcanza al destino, el destino firma un ROUTE REPLAY y lo envía hacia el nodo del cual recibió el ROUTE REQUEST. En el ejemplo, el objetivo regresa un ROUTE REPLAY firmado al salto previo C. El ROUTE REPLAY es reenviado de igual manera que el REQUEST, excepto que cada nodo transmite de forma unicast el REPLY al nodo del cual recibió el REQUEST. En particular, cada que un nodo recibe un REPLY checa la firma o firmas. En el ejemplo, el nodo B checa

el certificado $cert_C$ del nodo C, entonces checa la firma del mensaje saliente. B entonces verifica el certificado $cert_D$ de D y usa la llave en el certificado para verificar la firma del REQUEST. Si las firmas (o firma, cuando el paquete es recibido directamente del destino) son válidas, el nodo intermedio elimina la última firma y el certificado del nodo anterior (si aplica), firma el REPLY original, e incluye su propio certificado. Entonces transmite el REPLY de manera unicast al nodo del cual recibió el REQUEST asociado. En el ejemplo, el nodo B elimina la firma del nodo C, firma el REPLY resultante, e incluye su propio certificado. El nodo B entonces transmite el REPLY resultante a A, del cual había escuchado previamente el REQUEST.

$$S \rightarrow^*: (ROUTE\ REQUEST, D, cert_S, N, t)_{K_{\bar{S}}}$$
$$A \rightarrow^*: \left((ROUTE\ REQUEST, D, cert_S, N, t)_{K_{\bar{S}}}\right)_{K_{\bar{A}}}, cert_A$$
$$B \rightarrow^*: \left((ROUTE\ REQUEST, D, cert_S, N, t)_{K_{\bar{S}}}\right)_{K_{\bar{B}}}, cert_B$$
$$C \rightarrow^*: \left((ROUTE\ REQUEST, D, cert_S, N, t)_{K_{\bar{S}}}\right)_{K_{\bar{C}}}, cert_C$$
$$D \rightarrow C: \left((ROUTE\ REPLY, S, cert_D, N, t)_{K_{\bar{D}}}\right)$$
$$C \rightarrow B: \left((ROUTE\ REPLY, S, cert_D, N, t)_{K_{\bar{D}}}\right)_{K_{\bar{C}}}, cert_C$$
$$B \rightarrow A: \left((ROUTE\ REPLY, S, cert_D, N, t)_{K_{\bar{D}}}\right)_{K_{\bar{B}}}, cert_B$$
$$A \rightarrow S: \left((ROUTE\ REPLY, S, cert_D, N, t)_{K_{\bar{D}}}\right)_{K_{\bar{A}}}, cert_A$$

Figura 2-4. Descubrimiento de ruta en ARAN. En esta figura, el nodo S está descubriendo una ruta hacia el nodo D. Cada nodo retransmite el primer paquete route request que recibe de cada descubrimiento de ruta. Cuando el route request alcanza el objetivo, el destino regresa un route reply al nodo del cual escuchó ese route request. Cada nodo que escucha un route reply reenvía el reply al nodo del cual escuchó el request.

Cuando un nodo B retransmite un paquete REPLY que recibió del nodo C hacia el nodo previo A, él también establece un registro en la tabla de enrutamiento para el destino D, indicando que el destino del siguiente salto para paquetes destinados hacia D es el nodo C. Cuando los paquetes dirigidos para el destino D son reenviados al nodo B, B entrará en turno reenviándolos al nodo C. Si el nodo B descubre que el enlace de él hacia el nodo C está roto, y por lo tanto no puede reenviar paquetes al nodo C, inicia un mantenimiento

de ruta. La figura 2-5 muestra un ejemplo de mantenimiento de ruta en ARAN. Los nodos intermedios envían un ROUTE ERROR al salto previo, indicando que la ruta está rota. Este ROUTE ERROR incluye la fuente, el destino, el certificado del nodo intermedio, y un número N que se usa sólo una vez y sello de tiempo generado por el nodo intermedio para tener frescura. Este paquete es reenviado sin cambios a la fuente.

$$B \rightarrow A: \langle (ROUTE\ ERROR, S, cert_B, N, t)_{K_{\bar{B}}} \rangle$$
$$A \rightarrow S: \langle (ROUTE\ ERROR, S, cert_B, N, t)_{K_{\bar{B}}} \rangle$$

Figura 2-5. Mantenimiento de ruta en ARAN. Cuando un nodo B determina que su siguiente salto a D es inalcanzable, él retransmite un mensaje de error firmado indicando que su siguiente salto a D es inalcanzable. Cada nodo que usa B como un siguiente salto para D retransmite este route error pero no lo firma.

Debido a que ARAN usa criptografía de llaves públicas para autenticar, es particularmente vulnerable a ataques de negación de servicio (DoS) basados en inundación de la red con control de paquetes falsos para lo cual se requiere la verificación de firmas. Así como un nodo no puede verificar firmas en tiempo real, un atacante puede forzar ese nodo a desechar alguna fracción de los paquetes que recibe.

2.3.3.8 SAODV

Manel Guerrero Zapata y N. Asokan[26] proponen AODV seguro (SAODV), otro protocolo diseñado para proporcionar seguridad a AODV. La idea de SAODV es usar una firma para autenticar la mayoría de los campos de un route request (RREQ) y un route reply (RREP) y usar cadenas hash para autenticar el conteo de saltos. SAODV estructura extensiones de firma para AODV. La red de nodos autentica paquetes de enrutamiento de AODV con una extensión de firma de SAODV, el cual impide ciertos ataques como falsificación de identidad. En SOADV, un paquete RREQ incluye una extensión única de route request

(RREQ-SSE). El origen escoge un conteo máximo de salto, basado en el diámetro de red esperado, y genera una cadena hash de sólo un camino de longitud igual al conteo máximo de saltos más uno. Esta cadena de hash de sólo un camino es usada como un autenticador de métrica. El iniciador firma el RREQ y el ancla de su cadena hash; esta firma y el ancla son incluidas en el RREQ-SSE. Adicionalmente, el RREQ-SSE incluye un elemento de la cadena hash basada en el conteo de salto actual en el encabezado de RREQ. A este valor se le llama el autenticador de conteo de salto. Por ejemplo, si los valores de la cadena hash $h_0, h_1, \dots, h_N$ fueron generados tal que se cumple la Ecuación 7 entonces el autenticador de conteo de salto h_i corresponde al conteo de salto $N - i$.

$$h_i = H[h_{i+1}] \tag{7}$$

Con la excepción del campo de conteo de salto y autententicador de conteo, los campos de los encabezados de RREQ Y RREQ-SSE son inmutables y por lo tanto pueden ser autenticados al verificar la firma en la extensión del RREQ-SSE. Para verificar el campo de conteo de salto en el encabezado del RREQ, un nodo puede seguir la cadena hash al ancla. Por ejemplo, si el campo de conteo de salto es *i*, entonces el autenticador de conteo de salto hca debería ser $H^i[h_N]$. Debido a que la longitud (N) y ancla (h_N) de esta cadena hash es incluida en el RREQ-SSE y autenticado por la firma, un nodo puede seguir la cadena hash y asegurar el ancla usando la Ecuación 8.

$$h_N = H^{N-i}[hca]. \tag{8}$$

Cuando se reenvía un RREQ en SAODV, un nodo primero autentica el RREQ para asegurar que cada campo es válido. El nodo entonces realiza eliminación de duplicidad para asegurar que reenvía sólo un único RREQ para cada descubrimiento de ruta. El nodo entonces incrementa el campo de conteo de salto en el encabezado del RREQ, aplica la función hash al autenticador de conteo de salto, y retransmite el RREQ, junto con su extensión de RREQ-SSE.

Cuando el RREQ alcanza el destino, el destino checa la autenticación en el RREQ-SSE. Si el RREQ es válido, el destino regresa un RREP como en AODV. Una extensión de firma única del route replay (RREP-SSE) provee autenticación para el RREP. Como en el RREQ, el campo mutable es sólo el contador de salto; como resultado, el RREP es seguro de igual manera que el RREQ. En particular, un RREP-SSE tiene una firma que cubre el ancla de la cadena hash junto con todos los campos del RREP, excepto el contador de salto. El contador de salto es autenticado por un atenticador de contador de salto, el cual es también un elemento de la cadena hash. Un autenticador de contador de salto de h_i corresponde a un contador de salto de $N - i$.

Un nodo que reenvía un RREP checa la extensión de firma. Si la firma es válida, entonces este nodo intermedio coloca una entrada en su tabla de enrutamiento para los RREPs de la fuente original, especificando que paquetes a ese destino deberían de ser reenviados por el nodo intermedio anterior de quien escuchó el RREP.

SAODV permite que nodos intermedios respondan a través del uso de una extensión de firma doble de un route reply (RREP-DSE). Un nodo intermedio que responde a un RREQ incluye un RREP-DSE. La idea aquí es que para establecer una ruta hacia el destino, un nodo intermedio tuvo que haber previamente reenviado un RREP desde el destino. Si el nodo intermedio ya había almacenado el RREP y la firma, puede entonces regresar el mismo RREP si el número de secuencia en ese RREP es mayor al número de secuencia especificado en el RREQ. Sin embargo, algunos de los campos de ese RREP, en particular el campo tiempo de vida, no es válido mucho tiempo. Como resultado, una segunda firma, calculada por el nodo intermedio, se usa para autenticar este campo.

Para permitir respuestas basadas en información de enrutamiento de un paquete RREQ, el iniciador incluye una firma apropiada para un paquete RREP a través del uso de un RREQ-DSE. Conceptualmente, el RREQ-DSE es un RREQ y RREP contenidos dentro de un paquete. Para reducir el uso de paquetes adicionales (overhead), SAODV usa la observación de que los campos de RREP y RREP substancialmente coincidan. En particular, el RREQ-DSE necesita sólo incluir algunas banderas, un tamaño de prefijo, y algunos campos reservados, junto con una firma válida para un RREP usando esos valores. Cuando un nodo reenvía un RREQ-DSE, guarda la ruta y firma de la misma manera como si él hubiera reenviado un RREP.

SAODV además usa firmas para proteger el mensaje route error (RERR) usado en el mantenimiento de ruta. En SAODV, cada nodo firma el RERR que transmite, ya sea que él lo esté originando o reenviando. Los nodos que implementan SAODV no cambian su información del número de secuencia del destino cuando reciben un RERR debido a que el destino no autentica el número de secuencia de destino.

Bajo un ataque, ARAN sólo necesita verificar una firma en un paquete del atacante al poner en la lista negra a un nodo que no verifica correctamente la firma interna (la firma del origen en el caso de un RREQ o la firma del destino en el caso de un RREP). Un atacante, entonces, es improbable que incluya una firma válida de salida con una firma interna inválida. Como resultado, cualquier paquete falso tendría sólo una firma saliente falsa y, por lo tanto, tiene un costo de verificación igual al del protocolo SAODV.

2.3.3.9 TORP

En el pasado, protocolos de enrutamiento seguro para redes móviles ad hoc han dependido de la diseminación o de estado de enlaces, distancias o caminos publicados en los cuales se puede confiar. Mientras esto puede funcionar en redes estaticas, depender de dicha información provoca numerosos ataques en una MANET. Actualizaciones en caminos o los estados de los enlaces no pueden ser completamente asegurados a pesar de que una autoridad confiable informe a todos los nodos de una MANET que el par de nodos para los enlaces publicados son válidos, los cuales son todavía vulnerables a ataques debido a que requieren de una inundación confiable de la información transmitida a través de ellos. Por otro lado, se sabe que la publicación de distancias es vulnerable a ataques, en particular los que involucran la colusión. Por lo que en [39] se propone el protocolo de enrutamiento ordenado en tiempo (TORP) para asegurar el enrutamiento en una MANET. Con TORP, se establecen múltiples caminos libres de ciclos entre un origen y destino sin la necesidad de verificar el estado de los enlaces, publicar distancias o caminos establecidos hacia los destinos. Los nodos siguen pasos de fin a fin para detectar si paquetes de datos enviados a través de caminos existentes se entregaron correctamente. Para establecer caminos hacia un destino, un nodo usa sólo el tiempo local en el que él recibió los paquetes de control, más que alguna métrica publicada. Con TORP se mostró que es tan eficiente como los protocolos de enrutamiento sin mecanismos de seguridad (proactivos y reactivos) en la ausencia de atacantes y se probó que proporciona seguridad en contra de varios ataques.

Capítulo 3

Simulación de eventos discretos

3.1 SIMULACIÓN[27]

3.1.1 Introducción a la simulación

La simulación es una técnica útil para realizar análisis a sistemas de computadora. Si la caracterización del sistema no está disponible, como es el caso a menudo durante el diseño o la obtención del mismo, un modelo de simulación provee una manera fácil para predecir el rendimiento o comparar varias alternativas. Incluso si un sistema está disponible, los simuladores permiten caracterizarlo bajo una amplia variedad de cargas de trabajo y ambientes.
Sin embargo, los modelos de simulación a menudo fallan; esto es, producen resultados no útiles o resultados erróneos. Esto es debido o a que los desarrolladores de modelos son muy competentes en desarrollo de software pero carecen de fondo estadístico o porque son muy competentes en técnicas estadísticas pero no están conscientes de las buenas técnicas de desarrollo de software. Muchas simulaciones se terminan prematuramente antes de que se hayan completado. Esto se debe a que los modelos de simulación toman un gran tiempo de desarrollo (mucho más grande que el inicialmente anticipado).

3.1.2 Errores comunes en simulación

1. *Nivel de detalle inapropiado*: La simulación permite que un sistema sea estudiado más detalladamente que en una modelación analítica. EL análisis requiere varias suposiciones y simplificaciones. En otras palabras, un modelo analítico es menos detallado. En un modelo de simulación, el nivel de detalle es limitado sólo por el tiempo disponible para el desarrollo de la simulación. Una simulación más detallada requiere más tiempo para su desarrollo. La probabilidad de problemas se incrementa y es más difícil encontrarlos. La eliminación de fallas en tiempo se incrementa. Una simulación más detallada además requiere un mayor tiempo de cómputo para su ejecución. Esto es particularmente importante para simulaciones grandes donde el tiempo de ejecución puede ser del orden de varias horas o días. Generalmente se asume que un modelo más detallado es un mejor modelo, desde que toma menos suposiciones. Esto no siempre es verdad. Un modelo detallado podría requerir más conocimiento detallado o parámetros de entrada, los cuales si no están disponibles, podrían hacer al modelo inexacto. Por ejemplo, al simular un sistema de tiempo compartido, supone gastar tiempo en peticiones de servicio de disco necesarias para poder llevar a cabo la simulación. Una alternativa es generarlas usando una distribución exponencial. Una alternativa más detallada es simular el movimiento de la cabeza y rotación del disco. Si se escoge la segunda alternativa, se conseguirían resultados más exactos sólo si el sector y referencias de pista son conocidos.
Otro problema con modelos detallados es que toman mucho tiempo para su desarrollo. Es mejor iniciar con un modelo menos detallado, conseguir algunos resultados, estudiar las cosas delicadas, y presentar detalles en las áreas que tienen el mayor impacto sobre los resultados.

2. *Lenguaje incorrecto*: La opción del lenguaje de programación tiene un impacto significativo sobre el tiempo de desarrollo del modelo, los lenguajes de simulación de propósito especial requieren menos tiempo para el desarrollo del modelo y facilita varias tareas comunes como la verificación (usando bitácoras) y análisis estadístico. Los lenguajes de propósito general, por otro lado, son más portátiles y proveen mayor control sobre la eficiencia y tiempo de ejecución de la simulación.

3. *Modelos no verificados*: Los modelos de simulación son generalmente programas de computadora grandes, y se toman precauciones menos especiales, es posible tener varias fallas o errores de programación, los cuales harían que las conclusiones no tengan sentido.

4. *Modelos inválidos*: Incluso un programa de simulación que no tenga errores, podría no representar correctamente el sistema real debido a sus suposiciones incorrectas acerca del comportamiento del sistema. Es esencial que los modelos sean validados para asegurar que la conclusión obtenida sea la misma como la obtenida por el sistema real. Todos los resultados del modelo de simulación deberían ser conjeturados hasta ser confirmados por modelos analíticos, medidas, o por medio de la intuición.

5. *Condiciones iníciales manejadas inadecuadamente*: La parte inicial de una trayectoria de simulación es generalmente no representativa del comportamiento del sistema en un estado estable. La parte inicial debería por lo tanto ser descartada.

6. *Simulaciones demasiado cortas*: Los analistas a menudo tratan de ahorrarse tiempo, tanto como el tiempo que la computadora ahorra al correr simulaciones que son demasiado cortas. Los resultados en tales casos son fuertemente dependientes de las condiciones iníciales y podrían no ser representativas del sistema real. La longitud correcta para simulaciones depende de la precisión (amplitud de los intervalos de confianza) deseada y de la inconsistencia de las cantidades observadas.

7. *Generador de números aleatorios pobre*: Los modelos de simulación requieren cantidades aleatorias, y los procedimientos para generar los números aleatorios son llamados generadores de números aleatorios. Es más seguro usar un generador bien conocido que haya sido analizado extensamente que el que uno desarrolle. Incluso generadores que han sido bien conocidos tienen problemas.

8. *Selección Inapropiada de semillas*: Los generadores de números aleatorios son procedimientos de computadora que dando un número aleatorio generan otro. El primer número aleatorio en la secuencia es llamado semilla y tiene que ser suministrado por el analista. La semilla para diferentes flujos de números aleatorios debe ser escogida cuidadosamente para mantener independencia entre los flujos. A menudo, los analistas o comparten un flujo entre varios procesos diferentes o usan la misma semilla (generalmente inicializada a ceros) para todos los flujos. Esto presenta una correlación entre varios procesos en el sistema y lleva a conclusiones que podrían no ser

representativas del sistema real.

3.1.3 Terminología

A continuación se presentan un número de términos que son comúnmente usados al modelar un sistema por medio de una simulación.

1. Variables de estado: Las variables que definen el estado del sistema son llamadas variables de estado. Si una simulación es detenida a la mitad, puede ser reiniciada después si y sólo si los valores de todas las variables de estado son conocidas.
2. Evento: Un cambio en el estado del sistema es llamado evento. En la simulación de un protocolo de red un evento es la expiración de un temporizador o la transmisión o recepción de un paquete de datos.
3. Modelos de tiempo continuo y tiempo discreto: Un modelo en el cual el estado del sistema está definido en todos los tiempos es llamado un modelo de tiempo continuo. Si el estado del sistema es definido sólo en instantes particulares en el tiempo, el modelo es llamado modelo de tiempo discreto. Como ejemplo, considera una clase que se reúne cada viernes. Supóngase un modelo, en donde el estado de la clase es especificado por el número de estudiantes quienes toman la clase. Nótese que el tamaño de la clase puede ser determinado sólo en viernes. En otros días, el estado de la clase no está definido. Esto es por lo tanto un ejemplo de un modelo de tiempo discreto.
4. Modelos de estado continuo y estado discreto: Un modelo es llamado modelo de estado continuo o discreto dependiendo de si las variables de estado son continuas o discretas. Recordemos que las variables continuas pueden tomar valores infinitos incontables. Por ejemplo, si en un modelo de clase semanal, donde el estado es definido como el tiempo pasado por los estudiantes en la materia, eso sería un modelo estado continúo. De otra manera, si el estado es definido como el número de estudiantes, eso sería un modelo de estado discreto. En el modelo de un protocolo re red, la variable de estado (la longitud de la cola) puede asumir sólo valores enteros. Es por lo tanto un modelo de estado discreto. Un modelo de estados discretos es llamado también modelo de eventos discretos. Similarmente, un modelo de estados continuos es llamado modelo de eventos continuos. Nótese que la continuidad de tiempo no implica continuidad de estado y viceversa. Por lo tanto, se pueden encontrar ejemplos para todas las cuatro posibles combinaciones: estado discreto/tiempo discreto, estado discreto/tiempo continuo, estado continuo/tiempo discreto, y estado continuo/tiempo continuo.
5. Modelos determinísticos y probabilísticos. Si la salida (resultados) de un modelo puede ser predicha con certeza, eso es un modelo determinístico. De otra manera, un modelo probabilístico, da un resultado diferente en repeticiones para el mismo conjunto de parámetros de entrada.
6. Modelo estático y dinámico: Un modelo en el cual el tiempo no es una variable es llamado estático. Si el estado del sistema cambia con el tiempo, el modelo es dinámico. EL modelo de un protocolo de comunicaciones, por ejemplo, es un

modelo dinámico. Un ejemplo de un modelo estático es el siguiente modelo de transformación de masa a energía: $E = mc^2$.

7. Modelo abierto y cerrado: Si la salida es externa al modelo y es independiente de él, eso es llamado un modelo abierto. En un modelo cerrado, no hay entrada externa.
8. Modelos estables e inestables: Si el comportamiento dinámico del modelo se establece a un estado fijo, i.e., independiente del tiempo, es llamado estable. Un modelo del cual su comportamiento está continuamente cambiando es llamado inestable.

Los modelos de sistemas de computadoras son generalmente de tiempo continuo, estado discreto, probabilísticos, dinámicos, y no lineales. Algunos son abiertos; otros son cerrados. Adicionalmente, se usan los modelos de sistemas de computadora estables así como inestables.

En nuestro caso, el modelo de las MANETs es de tiempo discreto, estado discreto, probabilístico, dinámico, no lineal y cerrado.

3.1.4 Seleccionar un lenguaje para la simulación

Seleccionar un lenguaje apropiado es probablemente el paso más importante en el proceso de desarrollar un modelo de simulación. Una decisión incorrecta durante este paso podría guiar a grandes tiempos de desarrollo, estudios incompletos, y fallas.

Hay cuatro opciones: un lenguaje de simulación, un lenguaje de propósito general, extensión de un lenguaje de propósito general, y un paquete de simulación. Cada opción tiene sus propias ventajas y desventajas.

Lenguajes de simulación como SIMULA y SIMSCRIPT ahorran al analista tiempo considerable cuando se desarrolla una simulación. Esos lenguajes tienen herramientas que facilitan el avance de tiempo, planificación de evento, manipulación de entidad, generación de variables aleatorias, recolección estadística de datos, y generación de reportes. Ellos permiten a los analistas pasar más tiempo en temas específicos al sistema que está siendo simulado y no preocuparse acerca de temas que son generales a todas las simulaciones. Además, esos lenguajes permiten tener un código modular legible con buena detección de errores.

Un analista escoge primordialmente un lenguaje de propósito general como Java o C debido a la familiaridad que tenga con el lenguaje. La mayoría de los diseñadores de sistemas de computadoras y nuevos analistas no están familiarizados con los lenguajes de simulación. Además, los requerimientos para los límites de entrega no dejan tiempo para que ellos aprendan un lenguaje de simulación. Más allá, a menudo los lenguajes de simulación no están disponibles en sus sistemas de computadora. Esto es porque las personas escriben su primera simulación en un lenguaje de propósito general.

Incluso para principiantes, el tiempo de compensación entre un lenguaje de simulación y un lenguaje de propósito general no es realmente lo que parece. Si ellos escogen un lenguaje de simulación, ellos tienen que pasar tiempo aprendiendo el lenguaje. En algunos

casos, ellos podrían tener incluso que instalarlo en su sistema de computadora y ver que no haya piezas faltantes. Si ellos escogen un lenguaje de propósito general, ellos pueden iniciar inmediatamente. Pero ellos pasan tiempo desarrollando rutinas para manejo de eventos, generación de números aleatorios, etcétera. Se puede gastar tiempo considerable aprendiendo a cerca de estos temas y redescubrir problemas conocidos.

Esto no dice que los analistas deberían usar siempre lenguajes de simulación. Hay otras consideraciones, como la eficiencia, flexibilidad, y portabilidad, las cuales podrían hacer que un lenguaje de propósito general sea la única opción. Un lenguaje de propósito general da a los analistas más flexibilidad al permitirles tomar atajos prohibidos en un lenguaje de simulación. Más allá, un modelo desarrollado en un lenguaje de propósito general puede ser fácilmente convertido para su ejecución en diferentes sistemas de computadora.

Tener como objetivo escoger como opción entre un lenguaje de simulación y un lenguaje de propósito general, sugiere que el analista aprenda al menos un lenguaje de simulación para que adicionalmente se familiarice con otros factores que le ayuden en la selección del lenguaje.

Una extensión de un lenguaje de propósito general como GASP (para FORTRAN) es otra alternativa. Estas extensiones consisten de una colección de rutinas para manejar tareas que son comúnmente requeridas en simulaciones. Su intención es proveer un compromiso en términos de eficiencia, flexibilidad, y portabilidad.

Paquetes de simulación como NS2[28], QNET4[29] y RESQ[29] permiten al usuario definir un modelo usando un dialogo. Los paquetes tienen una biblioteca de estructura de datos, rutinas, y algoritmos. Sus ventajas más grandes son el tiempo de ahorro que proveen. Al usar un paquete de simulación, por ejemplo, uno podría desarrollar un modelo, resolverlo, y conseguir resultados en un día. Desarrollar una simulación usando un lenguaje, de otra manera, podría tomar varios días (o meses) dependiendo de la complejidad del sistema.

El principal problema con los paquetes de simulación es su inflexibilidad. Ellos proveen sólo las flexibilidades que fueron concebidas por sus desarrolladores. En la mayoría de las situaciones prácticas, los analistas caen dentro de uno u otro problema que no puede ser modelado por el paquete. Esto podría forzar a un analista hacer simplificaciones. A pesar de eso, ahorra mucho tiempo que si un sistema no se pudiera modelar analíticamente, se debería mirar en la posibilidad de usar un paquete de simulación antes de iniciar a desarrollar un nuevo modelo de simulación.

Los lenguajes de simulación pueden ser clasificados en dos categorías, lenguajes de simulación continua y lenguajes de simulación de eventos discretos, basados en los tipos de eventos que ellos simulan. Los lenguajes de simulación continua son diseñados para manejar modelos de eventos continuos que son generalmente descritos por ecuaciones diferenciales. Ejemplos de esos lenguajes son CSMP[30] y DYNAMO[31]. Estos lenguajes son populares en sistemas de modelaje químico. De otra manera, los lenguajes de simulación de eventos discretos son diseñados para manejar cambios de estados discretos. Dos ejemplos de estos lenguajes son SIMULA[32] y GPSS[33]. Algunos lenguajes, como SIMSCRIPT[34] y GASP[35], permiten simulaciones discretas, continuas, así como simulaciones combinadas. Las últimas cuatro son los lenguajes mayormente usados por

analistas del rendimiento de sistemas de computadora.

3.1.5 Tipos de simulación

Los tipos de simulación de mayor del interés para los científicos de la computación son la emulación, la simulación Monte Carlo, la simulación trace-driven, y la simulación de eventos discretos.
Una simulación usando hardware o firmware es llamada emulación. Una terminal emuladora, por ejemplo, simula una clase de terminal sobre otra. Un emulador de procesador emula un conjunto de instrucciones de un procesador sobre otro. Aunque la emulación es un tipo de simulación, las tareas de diseño para emulación son mayormente tareas de diseño de hardware. Por lo tanto, la emulación no será discutida.
Los otros tres tipos de simulación son descritos a continuación.

3.1.5.1 Simulación de Monte Carlo

Una simulación estática o sin un eje de tiempo es llamada simulación de Monte Carlo. Tales simulaciones son usadas para modelar fenómenos probabilísticos en los que no cambian sus características con el tiempo. Como en una simulación dinámica, ellos requieren la generación de números pseudoaleatorios. Además las simulaciones Monte Carlo son usadas para la evaluación de expresiones no probabilísticas usando métodos probabilísticos.

3.1.5.2 Simulación Trace-Driven

Una simulación usando una bitácora como entrada es una simulación trace-driven. Una bitácora es una grabación de eventos ordenados por tiempo en un sistema real. Las simulaciones trace-driven son muy comunes en análisis de sistemas de computadora. Son generalmente usadas en análisis o para ajustar algoritmos de administración de recursos. Dividir algoritmos en secciones, análisis de caché, algoritmos de planificación de CPU, algoritmos de prevención de interbloqueo, y algoritmos para asignación dinámica de almacenamiento son ejemplos de casos donde la simulación trace-driven ha sido usada y documentada satisfactoriamente en los trabajos escritos. En tales estudios, una bitácora de demanda de algún recurso se usa como entrada para la simulación, la cual modela diferentes algoritmos. Por ejemplo, con el propósito de comparar diferentes modelos de administración de memoria, se puede obtener en un sistema una bitácora de patrones de referencia de página de programas clave. Esta bitácora puede ser usada para encontrar el conjunto óptimo de parámetros para un algoritmo dado de administración de memoria o para comparar diferentes algoritmos.

Se debe notar que las bitácoras deben ser independientes del sistema bajo estudio, por ejemplo, una bitácora de páginas que van y regresan de un disco dependen del tamaño del conjunto de trabajo, así como de la política de reemplazo de página usada. Esta bitácora no puede ser usada para estudiar otras políticas de reemplazo de página. Similarmente, una bitácora de instrucción obtenida en un sistema operativo no debería ser usada para analizar otro sistema operativo.

Sherman y Brown (1973)[39] señalan las siguientes ventajas de simulaciones trace-driven:

1. Credibilidad: Los resultados de una simulación trace-driven son fáciles de vender a otros miembros el equipo de diseño. Por ejemplo, una bitácora de referencias a páginas, tiene más credibilidad que referencias generadas aleatoriamente usando una distribución supuesta.
2. Fácil validación: El primer paso en una simulación trace-driven es monitorear un sistema real para conseguir la bitácora. Durante este monitoreo, uno puede además medir el rendimiento de características del sistema. Al comparar la medida del rendimiento con el obtenido por la simulación, uno puede validar fácilmente un modelo trace-driven.
3. Carga de trabajo exacta: Una bitácora mantiene los efectos de correlación e interferencia en la carga de trabajo. No se requiere alguna simplificación como las que se necesitan al conseguir un modelo analítico de la carga de trabajo.
4. Compensaciones detalladas: Debido al alto nivel de detalles en el trabajo a realizar, es posible estudiar el efecto de pequeños cambios en el modelo o algoritmos.
5. Menos aleatorización: Una bitácora es una entrada determinística. Si la simulación es repetida otra vez, la bitácora de entrada es todavía la misma pero la salida podría ser diferente debido a la aleatorización en otras partes del modelo. Sobre todo, la salida de un modelo trace-driven tiene menos variación, lo cual significa que se requiere un número menor de repeticiones para conseguir una confiabilidad deseada en el resultado. Además, si otras partes del sistema no son aleatorios, es posible conseguir resultados absolutos en una ejecución del modelo.
6. Comparación legítima: Una bitácora permite diferentes alternativas para que sean comparadas bajo el mismo flujo de entrada. Esto es, una comparación más legítima que otros modelos de simulación en el cual la entrada es generada desde un flujo aleatorio y es diferente para varias alternativas que están siendo simuladas.
7. Similitud a la implementación actual: Un modelo trace-driven es generalmente muy similar al sistema que está siendo modelado. Así, cuando se está implementando, uno puede obtener muy buenas conjeturas de la complejidad al implementar un algoritmo propuesto.

Desventajas de simulaciones trace-driven:

1. Complejidad: Un modelo trace-driven requiere una simulación más detallada del sistema. Algunas veces, la complejidad del modelo sobrepasa la importancia del algoritmo que está siendo modelado.
2. Representatividad: Las bitácoras tomadas de un sistema podrían no ser

representativas del trabajo a realizar sobre otro sistema. Incluso en un trabajo a realizar en un único sistema podrían cambiar con el tiempo, y entonces las bitácoras se vuelven obsoletas más rápido que otras formas de modelos de carga de trabajo que pueden ser ajustados con el tiempo.

3. Finalización: Una bitácora es una secuencia grande. Una bitácora detallada de unos cuantos minutos de actividad en un sistema podría ser suficiente para llenar un paquete de discos. Un resultado basado en esos cuantos minutos podría no ser aplicable a las actividades durante el resto del día.
4. Punto de validación único: Mientras se usan bitácoras para validación, se debe tener cuidado ya que las bitácoras dan sólo un punto de validación para cada una. Un algoritmo que es el mejor para una bitácora podría no ser el mejor para otra. Se debe usar varias bitácoras para validar los resultados.
5. Detalle: El principal problema con la simulación trace-driven es el alto nivel de detalle. Las bitácoras son generalmente secuencias muy largas que tienen que ser leídas desde un disco y entonces el cálculo computacional se tiene que hacer para cada elemento.
6. Compensación: Con bitácoras, es difícil cambiar las características del trabajo a realizar. Se requiere una bitácora diferente para la carga de trabajo a realizar con el propósito de hacer conclusiones acerca del impacto de los cambios en el trabajo a realizar. Similarmente, si una bitácora contiene características de demanda de recursos de varios trabajos, es difícil estudiar los efectos sobre trabajos individuales.

3.1.5.3 Simulaciones de eventos discretos

Una simulación usando un modelo de estados discretos del sistema es llamada una simulación de eventos discretos. Esto es opuesto a simulaciones de eventos continuos en el cuál el estado del sistema toma valores continuos. Los modelos de estados continuos son usados, por ejemplo, en simulaciones químicas donde el estado del sistema se describe por la concentración de substancia química. En sistemas de computadoras, se usan modelos de eventos discretos debido a que el estado del sistema se describe por el número de trabajos en varios dispositivos. Nótese que el término "discreto" no se aplica a los valores de tiempo usados en la simulación. Una simulación de eventos discretos podría usar valores de tiempo continuo o discreto.

Todas las simulaciones tienen una estructura común. Sin considerar el sistema que está siendo simulado, la simulación tendrá algunos de los componentes descritos a continuación. Si se usa un lenguaje de propósito general, todos los componentes tienen que ser desarrollados por el analista. Los componentes son:

1. Planificador de eventos: Guarda una lista enlazada de eventos esperando que sucedan. El planificador permite que los eventos sean manipulados en varias maneras. Algunas de esas actividades de manipulación son:
    a. Planifica el evento X en el tiempo T
    b. Mantiene el evento X por un intervalo de tiempo dt.

c. Cancela un evento X previamente programado.
d. Mantiene el evento X indefinidamente (hasta que éste es planificado por otro evento).
e. Planifica un evento guardado indefinidamente.

El planificador de eventos es uno de los componentes usados más frecuentemente en la simulación. Es ejecutado antes de cada evento, y podría ser llamado varias veces durante un evento para planificar otros nuevos eventos.

2. Reloj de Simulación y mecanismos de avance de tiempo: Cada simulación tiene una variable global representando el tiempo simulado. Existen dos formas de actualizar el tiempo de la simulación y en general, el planificador es el encargado de hacerlo. La primera manera, llamada enfoque de unidad de tiempo, incrementa el tiempo por pequeños incrementos y entonces verifica si hay otros eventos que puedan ocurrir. El segundo enfoque, llamado el enfoque de manejador de eventos, incrementa el tiempo automáticamente al tiempo del siguiente evento más cercano que está ocurriendo. El enfoque de la unidad de tiempo no es generalmente usado en simulaciones de computadora.

3. Variables de estado del sistema: Éstas son variables globales de estado que describen el estado del sistema. Por ejemplo, en la simulación de una red de computadoras, una variable de estado del sistema es el número de nodos activos. Ésta, es una variable de estado global que es distinta de variables locales como el tiempo de CPU requerido por un trabajo, el cual sería almacenado en la estructura de datos representando el trabajo.

4. Rutinas de eventos: Cada evento es simulado por su rutina. Estas rutinas actualizan las variables de estado del sistema y programan otros eventos. Por ejemplo, al simular un protocolo que trabaja en la capa de red, se deben implementar rutinas para manejar los eventos de llegadas de paquetes de datos desde la capa de transporte, llegadas de paquetes de datos desde la capa MAC, y llegada de paquetes de control desde la capa MAC.

5. Rutinas de entrada: Éstas consiguen los parámetros del modelo, es decir, número de nodos, tipo de tráfico, patrones de movilidad, configuración de hardware de los nodos, etc. En general, es mejor preguntar por todas las entradas al inicio de una simulación y entonces liberar al usuario, debido a que las simulaciones generalmente toman un tiempo prolongado para completarse. Las rutinas de entrada típicamente permiten variar a un parámetro en una manera específica. Por ejemplo, la simulación podría ejecutada con diferentes volúmenes de tráfico para verificar el comportamiento de la red ante distintas cargas de trabajo. Cada conjunto de valores de entrada define una interacción que podría tener que ser repetida varias veces con diferentes semillas. Así, cada única ejecución de la simulación consiste de varias iteraciones, y cada iteración consiste de varias repeticiones.

6. Generador de reporte: Son las rutinas de salida ejecutadas al final de la simulación. Ellas calculan el resultado final e imprimen en un formato especificado.

7. Rutinas de inicialización: Éstas establecen el estado inicial de las variables de estado del sistema e inicializan varios flujos generadores de números aleatorios. Se sugiere que haya rutinas separadas para inicializar el estado al inicio de la simulación, al inicio de una iteración, y al inicio de una repetición.

8. Rutinas de bitácora: Éstas imprimen variables intermedias conforme proceda la simulación. Ayudan a depurar al programa de la simulación. Es recomendable que la bitácora tenga una característica de encendido/apagado tal que pueda ser apagada para ejecuciones de producción final del modelo. Un modelo podría incluso permitir la habilidad de interrumpir la ejecución del modelo desde el teclado y encender o apagar la bitácora.

9. Administración de memoria dinámica: El número de entidades en una simulación cambia continuamente tanto como nuevas entidades sean generadas y viejas sean destruidas. Esto requiere recolección periódica de basura. La mayoría de los lenguajes de simulación y muchos lenguajes de propósito general proveen esto automáticamente. En otros casos, el programador tiene que realizarlo o escribir códigos para administración dinámica de memoria.

10. Programa principal: Esto trae todas las rutinas juntas. Llama a todas las rutinas de entrada, inicializa la simulación, ejecuta varias iteraciones, y finalmente, llama las rutinas de salida.

3.1.6 Algoritmos de colocación de eventos

En simulaciones de eventos discretos, es necesario asegurar que los eventos ocurran en el orden y tiempo apropiado. La mayoría de los lenguajes de simulación proveen facilidades para esto. Sin embargo, para simulaciones escritas en lenguajes de propósito general, el programador debe implementar esta capacidad. Algunas veces, incluso mientras se usa un lenguaje de simulación, los analistas podrían preferir usar su propio algoritmo de colocación de eventos. Por ejemplo, la implementación eficiente de un algoritmo de colocación de evento ha resultado, en algunos casos, 30% de ahorro en el tiempo total de procesador.

La planificación de eventos usualmente se hace al guardar una lista enlazada ordenada de notas de eventos. Cada nota contiene el tiempo en el cual el evento debería de ocurrir y un puntero al código que debe ser ejecutado en ese tiempo. Hay dos operaciones que son realizadas frecuentemente sobre este conjunto: una para insertar nuevos eventos en el conjunto y una para encontrar el siguiente (más cercano) evento y removerlo del

conjunto. La opción de la estructura de datos usada para mantener este conjunto afecta al tiempo de procesador requerido para las dos operaciones. Algunas estructuras de datos tienen muy pequeña sobrecarga para la inserción pero requieren procesamiento considerable para encontrar el siguiente evento. Otras estructuras de datos hacen todo el trabajo a la hora de la inserción así que encontrar el siguiente evento es directo. La opción de estructura de datos por lo tanto está basada en la frecuencia de inserción y eliminación y en el número promedio de eventos que almacenara el conjunto. Algunas de las estructuras de datos que han sido propuestas son:

1. Lista enlazada ordenada: La propuesta más usada comúnmente en lenguajes de simulación como SIMULA, GPSS, y GASP IV es la que guarda una lista ordenada doblemente enlazada. La primer entrada en la lista es el siguiente evento más cercano. Así, la eliminación es directa. Para insertar un nuevo evento, la lista es recorrida para encontrar el lugar correcto para la nueva entrada. Algunas alternativas para buscar una dirección han sido propuestas. El método más común es insertar atrás del valor de tiempo más grande. Alternativamente, la lista podría ser recorrida hacia adelante desde la primer entrada. Algunos incluso han intentado guardar un apuntador en la mitad de la entrada, determinar primero la mitad que contendría el lugar correcto y entonces buscar hacia adelante o hacia atrás para determinar el lugar.

2. Lista lineal indexada: En esta propuesta, el conjunto de eventos futuros es dividido en varios subconjuntos. Cada subconjunto abarca un intervalo fijo "t del intervalo de tiempo y es mantenido como una sublista. Se guarda un arreglo de índices tal que la iésima entrada del índice que apunta a la iésima sublista que contiene eventos programados para el intervalo [(i-1)"t,i"t], que está, en o después del tiempo (i-1)"t pero antes del tiempo i"t. Aquí, "t es un intervalo especificado por el usuario. Así, dado un nuevo evento para ser insertado, la sublista requerida puede ser determinada sin ninguna búsqueda. La sublista apropiada entonces es recorrida hacia atrás para encontrar la posición de la nueva entrada.
   Un número de variaciones a este modelo han sido propuestas basadas en el argumento de que los tiempos de retención de evento (el tiempo entre la planificación de un evento y su ocurrencia) no son uniformemente distribuidos. En una variación, se hace un intento para guardar todas las listas de la misma longitud, así como el intervalo cubierto por cada entrada de índice; eso es, "t es variable. La búsqueda binaria se usa para encontrar la entrada apropiada del índice. En otra variación, sólo la primera lista es organizada; otras listas son guardadas desordenadas. Una sublista es organizada sólo cuando se convierte en la primer sublista. De ese modo, se trata de reducir la sobrecarga por ordenar.
   Una variación interesante de esta propuesta es llamada colas de calendario. Están basadas en los calendarios de escritorio usados por los humanos para planificar eventos. Un calendario de escritorio típico tiene 365 páginas (una página para cada día del año). Todos los eventos para un único día son escritos abajo de la página correspondiente a ese día. Los eventos para el mismo día del siguiente año pueden

además ser escritos en esa página. Esto no causará ninguna confusión si el año que transcurre es también escrito con el evento y los eventos son eliminados después de que ellos hayan tomado lugar. Esta idea puede ser fácilmente implementada usando una lista lineal indexada. El intervalo "t corresponde a un día humano, y el tamaño del índice corresponde al número de días en el año. Ambos parámetros se deben escoger cuidadosamente debido a que el número de eventos por página es pequeño (cerca a 0 ó 1). Un procedimiento para ajustar dinámicamente estos dos parámetros fue descrito por Brown (1988) quien además mostró que el algoritmo toma una cantidad fija de tiempo por evento sin importar el número de eventos.

3. Estructuras de árbol. Estructuras de datos de árbol también han sido usadas para la simulación de conjuntos de eventos. Usualmente se usa en árbol binario. El tiempo para buscar a través de n eventos es entonces $log_2 n$.
   Un caso especial del árbol binario es la pila, donde cada evento es almacenado como un nodo en el árbol binario. Cada nodo puede tener arriba dos hijos, y el tiempo del evento para cada nodo es más pequeño que el de sus hijos. Esto implica que la raíz siempre tiene el tiempo de evento más pequeño. La ventaja de las pilas es que los árboles pueden ser almacenados en un arreglo (opuesto a una lista enlazada) al poner la raíz en la posición 1 del arreglo y sus hijos en posiciones 2 y 3. Los nodos en el siguiente nivel son guardados en un arreglo de posiciones 4, 5, 6, 7 y así. El recorrido de una pila es simple porque es fácil encontrar los padres y los hijos de cualquier nodo en particular. Los dos hijos de un nodo en posición i están en posiciones $2i$ y $2i + 1$. El padre de un nodo en posición i es el que está en la posición $[i/2]$. Aquí, $[\cdot]$ representa el truncamiento al siguiente entero más bajo. El arreglo tiene que ser reordenado parcialmente después de cada inserción o eliminación.

La opción de la estructura de datos apropiada depende de la distribución de los tiempos del evento retenido y el número de eventos en el futuro conjunto de eventos. Además depende de la facilidad con la cual varias estructuras de datos puedan ser implementadas en el lenguaje de programación dado. En un estudio por Reeves (1984), la lista enlazada simple fue selecionada como la alternativa más eficiente si el número de eventos fuera pequeño (menos de 20 eventos). Para conjuntos de eventos de tamaños 20 a 120, la mejor opción son las listas lineales de índices, mientras para conjuntos más grandes, las pilas resultaron ser las más eficientes. McCormack y Sargent (1979) obtuvieron conclusiones similares.

3.2 Análisis de los resultados de simulación

Durante el desarrollo del modelo de simulación, se debe asegurar que el modelo esté correctamente implementado y que es representativo del sistema real. Estos dos pasos son llamados verificación del modelo y validación del modelo, respectivamente. Después

de que el desarrollo del modelo esté completado, las siguientes dos tareas que se enfrentarán son la de decidir cuantas de las observaciones iníciales deberían ser descartadas para asegurar que el modelo ha alcanzado un estado balanceado y que tan larga será la simulación. Estas tareas son llamadas eliminación de efectos transitorios y criterio de paro, respectivamente.

3.2.1 Técnicas de verificación del modelo

Lo calidad de un modelo de simulación es medida por la cercanía de la salida del modelo a la de los sistemas reales. Debido a que se hace un número de suposiciones acerca del comportamiento de sistemas reales en el desarrollo del modelo, hay dos pasos para medir su calidad. El primer paso es verificar que las suposiciones sean razonables, y el segundo paso es verificar que el modelo implemente esas suposiciones correctamente. Hay dos pasos llamados validación y verificación, respectivamente. La validación está relacionada con la representatividad de las suposiciones, y la verificación está relacionada a la corrección de la implementación. La verificación puede además ser llamada depuración, eso es, asegurar que el modelo haga lo que se espera que haga.
Validación y verificación son conceptos diferentes en un modelo que podría estar en una de las cuatro posibles categorías: inválido y no verificado, inválido y verificado, válido y no verificado, o válido y verificado. Un modelo inválido y verificado, por ejemplo, es el que implementa correctamente las suposiciones, pero las suposiciones están lejos de la realidad. Si la modelación y programación de un modelo de simulación se están llevando a cabo separadamente por dos personas (o equipos), la persona que hace la modelación sería responsable de la validación y la persona que hace la programación estaría relacionada con la verificación.
Se pueden encontrar muchas técnicas para depurar en la documentación de los lenguajes de programación. Cualquier combinación de estas técnicas se puede usar para verificar el modelo. Algunas de estas técnicas, junto con unas cuantas técnicas aplicables especialmente a modelos de simulación, se describen a continuación.

3.2.1.1 Diseño modular de arriba a abajo

Los modelos de simulación son programas de computadora grandes. Todas las técnicas que ayudan a desarrollar, depurar, o mantener grandes programas de computadora son además útiles para modelos de simulación. Dos importantes técnicas son la modularidad y el diseño de arriba a abajo.
La modularidad requiere que el modelo esté estructurado en módulos que se comunican entre sí vía interfaces bien definidas. Estos módulos son comúnmente llamados subrutinas, subprogramas, procedimientos, etcétera. La interfaz consiste de un número de variables de entrada y salida o estructuras de datos. Una vez que la interfaz y la función de un modulo han sido especificadas, pueden ser desarrolladas, depuradas, y mantenidas

independientemente. Entonces la modularidad permite la verificación de la simulación al ser dividida en problemas más pequeños de verificación de los módulos y sus interfaces.
El diseño de arriba a abajo consiste en desarrollar una estructura jerárquica para el modelo tal que el problema es recursivamente dividido en un conjunto de problemas más pequeños. Primero, el modelo es dividido en un número de módulos con diferentes funciones. Cada uno de esos módulos es subdividido en módulos. El proceso es repetido hasta que los módulos sean los suficientemente pequeños para ser fácilmente depurados y mantenidos.

3.2.1.2 Anti-errores de software

El anti-error de software consiste en incluir revisiones y salidas adicionales en el programa que apuntará hacia los errores, si existen. Por ejemplo, si las probabilidades para ciertos eventos se supone que suman uno, el programa podría revisar esto e imprimir un mensaje de error si la suma no está dentro de un límite de tolerancia permitido. Otro ejemplo, es el de contar las entidades generadas y atendidas. En la simulación de una red de computadoras, por ejemplo, el modelo cuenta el número de paquetes enviados por un número de nodos fuente, así como el número de paquetes recibidos por los nodos destino. El número de paquetes perdidos en la ruta y los paquetes recibidos deberían ser iguales al número de paquetes enviados. Una diferencia no igual a cero indicaría un error de programación. De hecho, para cada entidad definida en la simulación, es una buena práctica tener una rutina para generar una entidad que cuente el número de entidades generadas y tener una rutina explícita para la destrucción de entidades que decremente el número de entidades antes de liberar su estructura de datos para la recolección de basura. Los números son mantenidos en un área global (común), la cuál es revisada al final de la simulación para ver que todas las entidades estén contadas apropiadamente. Por ejemplo, en una simulación de red, paquetes, nodos, y conexiones son contadas cuando son generadas así como destruidas. Es muy útil desde que pequeños cambios en el modelo a menudo resultan en errores que son descubiertos por la rutina de conteo de entidad al final de la simulación.

3.2.1.3 Camino estructurado

Consiste en explicar el código a otra persona o grupo. El desarrollador del código explica qué hace cada línea de código, las cuales pueden ser de utilidad. Aclara lo que los oyentes no entienden del modelo, los desarrolladores descubren errores simplemente al leer cuidadosamente el código, intentando explicarlo, y encontrando el código que no concuerda exactamente con la expectativa.

3.2.1.4 Modelos determinísticos

El problema clave al depurar un modelo de simulación es la aleatoriedad de las variables. Es obvio que un programa determinístico es más fácil de depurar que un programa con variables aleatorias. Una técnica común de verificación, permite al usuario especificar cualquier distribución. Por su puesto, los valores por omisión de los parámetros deberían ser establecidos para representar el comportamiento en sistemas reales. Pero al especificar distribuciones constantes (determinísticas), el usuario puede determinar fácilmente las variables de salida y así depurar los módulos.

3.2.1.5 Ejecutar casos simplificados

El modelo podría ser ejecutado con casos simples, por ejemplo, sólo un paquete, o sólo una fuente, o sólo un nodo intermedio. Estos casos pueden ser analizados fácilmente y los resultados de la simulación pueden ser comparados con el análisis. Por su puesto un modelo que trabaja para casos simples no se garantiza que trabaje para casos más complejos. Por lo tanto, la prueba de casos debe ser tan compleja como pueda ser analizada fácilmente.

3.2.1.6 Bitácora

Una bitácora consiste de una lista de eventos en un tiempo ordenado y sus variables asociadas. Ésta puede ser presentada en varios niveles de detalle: bitácora de eventos, bitácora de procedimientos, o bitácora de variables. Las salidas de la bitácora son útiles en la depuración del modelo. Realizar la bitácora causa sobrecarga en el procesamiento adicional, y por lo tanto, el modelo debería tener interruptores que permitieran encender o apagar las bitácoras. Una muestra de una bitácora de evento para la simulación de una red de computadoras, lista el tiempo, el código del evento, y varias características del paquete asociadas con el evento.
Es posible tener mucho más detalles en una bitácora. El usuario debería ser capaz de seleccionar el nivel de detalles en la bitácora, incluyendo no llevar un registro por completo, lo cual debería ser por omisión. El usuario podría tener permitido sólo registrar algunos eventos seleccionados como los pertenecientes a un nodo en particular en una red de computadoras o a un tipo particular de paquete, etcétera.

3.2.1.7 Despliegue de gráficas en línea

Las simulaciones toman un largo tiempo para correr. Las capturas de gráficas en línea y las bitácoras ayudan a mantener informado al usuario sobre el estado de la simulación. Además hacen la simulación interesante y son de ayuda al exponer los resultados de la

simulación a otros. Mayormente, los despliegues de gráficas ayudan a depurar la simulación; ellas pueden presentar la misma información que en las bitácoras pero en una forma más comprensible. Es difícil mirar en una bitácora grande, mientras que es fácil inspeccionar los despliegues de gráficas para el mismo periodo.

3.2.1.8 Prueba de continuidad

Las pruebas de continuidad consisten en correr varias veces la simulación para valores ligeramente diferentes de los parámetros de entrada. Para cualquier parámetro, un ligero cambio en la entrada generalmente debería producir solo un ligero cambio en la salida. Los cambios espontáneos en la salida deben ser investigados. A menudo, se deben a un error de modelación.

3.2.1.9 Pruebas de degradación

Las pruebas de degradación consisten en verificar que el modelo trabaje para valores extremos del sistema, configuración, o parámetros de la carga de trabajo del sistema. A pesar de que esos casos extremos podrían no representar un caso típico, ellos ayudan a descubrir errores que el analista no habría pensado. Es útil incorporar puntos de verificación para valores de parámetros de entrada y verificar que ellos están dentro de los límites permitidos. El analista debería haber verificado que el modelo trabaje para cualquier combinación de esos límites permitidos.
Por ejemplo, el modelo de la simulación de red trabaja para un sistema sin fuentes, sin ruteadores, ruteadores con cero tiempos de servicio, o fuentes con ventanas de control de flujos infinitos.

3.2.1.10 Pruebas de consistencia

Estas pruebas consisten en verificar que el modelo produzca resultados similares para los valores de los parámetros de entrada. Por ejemplo, dos fuentes con una tasa de llegada de 100 paquetes por segundo deberían cargar la red para aproximar el mismo nivel como cuatro fuentes con una tasa de llegada de 50 paquetes por segundo cada una. Si la salida del modelo muestra una salida significativa, o si se tiene una diferencia, la diferencia debería ser explicable o podría ser debida a errores de programación.
Los casos de prueba usados en continuidad, degradación, y pruebas de consistencia deberían ser guardados en una biblioteca de pruebas así que siempre que el modelo es cambiado, las pruebas pueden ser repetidas para verificar el nuevo modelo.

3.2.1.11 Independencia en la semilla

Las semillas usadas en la generación de números aleatorios no debería afectar la conclusión final. Así, el modelo debería producir resultados similares para valores de semillas diferentes. El analista debe verificar esto al correr la simulación con diferentes valores de semillas.

3.2.2 Técnicas de validación del modelo

La validación se refiere a asegurar que las suposiciones usadas en el desarrollo del modelo son razonables, y si son correctamente implementadas, el modelo produciría resultados cercanos a los observados en sistemas reales. Las técnicas de validación dependen de las suposiciones y, por lo tanto, de sistemas que están siendo modelados. Así, a diferencia de las técnicas de verificación que son generalmente aplicables, las técnicas de validación usadas en una simulación podrían no aplicarse a otra.

El modelo de validación consiste en validar los tres aspectos claves del modelo:

1. Suposiciones
2. Valores de los parámetros de entrada
3. Valores de salida y conclusiones

Cada uno de esos tres aspectos podrían estar sujetados a una prueba de validación al compararlos con los obtenidos desde las siguientes tres posibles fuentes:

1. Intuición del experto
2. Medidas de sistemas reales
3. Resultados teóricos

Lo anterior guía a nueve posibles pruebas de validación. Por supuesto, podría no ser conveniente usar alguna de esas posibilidades. En la mayoría de situaciones reales, de hecho, ninguna de las nueve posibilidades podría ser conveniente.
Las tres fuentes de información comparable se describen en las siguientes secciones:.

3.2.2.1 Intuición del experto

Es la manera más práctica y común para validar un modelo. Se podría convocar a una reunión de la gente bien informada del sistema. Las personas son involucradas por los desarrolladores del diseño, arquitectura, implementación, análisis, promoción, o mantenimiento del sistema. Por supuesto, la selección depende de la fase del ciclo de vida del sistema que está siendo modelado. Para sistemas ya en uso, el campo de servicio y la gente de promoción podrían tener un mejor conocimiento que los implementadores. Las

suposiciones, valores de entrada y distribuciones, así como sus salidas son presentadas y discutidas en la reunión.
En la práctica, es mejor validar los tres aspectos (suposiciones, entrada, y salida) separadamente tanto como el modelo progrese más que esperar hasta el final. Las suposiciones del modelo deberían ser puestas a discusión tan pronto como un diseño preliminar del modelo de simulación ha sido preparado. Los valores de entrada y distribución deberían ser discutidos y validados durante el desarrollo del modelo. La salida debería ser validada tan pronto como un modelo ejecutable exista y haya sido verificado.
Una técnica para juzgar la validación de las salidas es presentar a los expertos los resultados del modelo así como las medidas de sistemas reales y ver si ellos pueden distinguir los resultados.
Los analistas expertos pueden señalar errores en la simulación simplemente con mirar en la salida de la simulación.

3.2.2.2 Medidas del sistema real

La comparación con sistemas reales es la manera más confiable y preferida de validar un modelo de simulación. En la práctica, sin embargo, esto es a menudo inconveniente o porque los sistemas reales podrían no existir o porque las medidas podrían ser demasiado caras para obtenerlas. Aunque una o dos medidas pueden ser suficientes para la validación de la simulación.
Se deben comparar las suposiciones, valores de entrada, valores de salida, carga de trabajo, configuraciones, y el comportamiento del sistema con los observados en el mundo real.

3.2.2.3 Resultados teóricos

En algunos casos, es posible modelar analíticamente el sistema simplificación por medio de simplificaciones basadas en suposiciones. Podría además ser posible analíticamente determinar qué distribuciones de entrada deben ser utilizadas. En tales casos, la similitud de los resultados teóricos y resultados de simulación son usados para validar el modelo de simulación. Similarmente, al desarrollar modelos de colas de sistema de computadora, los analistas a menudo encuentran que el comportamiento de los sistemas reales es demasiado difícil para modelar exactamente. En tales casos, ellos desarrollan una técnica aproximada y lo validan al comparar los resultados con los obtenidos de una simulación.
La validación al comparar resultados teóricos y resultados de simulación debe realizarse con cuidado debido a que ambos podrían ser inválidos en el sentido de que podrían no representar el comportamiento de un sistema real. A pesar de eso, es una técnica útil que provee puntos que han sido validados usando la intuición del experto o medidas del sistema real. La modelación analítica permite validación para casos más complejos. Por ejemplo, un modelo de simulación de un multiprocesador podría ser validado usando algunos sistemas reales de uno o dos procesadores, y entonces se puede usar la teoría

para validar un gran número de procesadores.
Antes de concluir esta sección de validación, vale la pena señalar que un "modelo completamente validado" es un mito. En realidad, es posible sólo mostrar que el modelo no es inválido para algunas de las situaciones comparadas. Para probar que el modelo produce el mismo resultado que el sistema original bajo todas las circunstancias requeriría generalmente una cantidad excesiva de experimentos. El ejercicio de validación es por lo tanto limitado a pocos escenarios, y se hace un intento para cubrir todos los casos importantes. Esto ayuda a incrementar el grado de confiabilidad en los resultados del modelo.

3.2.3 Eliminación de transitorios

En la mayoría de las simulaciones es interesante sólo el comportamiento del estado estable, esto es, el comportamiento después de que el sistema ha alcanzado un estado estable. En tales casos, los resultados de la parte inicial de la simulación no deberían ser incluidos en los cálculos finales. Esta parte inicial es también llamada el estado transitorio. El problema de identificar el fin de un estado temporal es llamado eliminación de estados transitorios.
La principal dificultad con la eliminación de estados transitorios es que no es posible definir exactamente qué conforma el estado transitorio y cuándo termina. Todos los métodos para eliminación de transitorios son por lo tanto heurísticas.
Métodos:

1. Ejecuciones grandes
2. Inicialización apropiada
3. Truncamiento

3.2.3.1 Ejecuciones grandes

El primer método es simplemente usar corridas muy largas; es decir, corridas que son suficientemente grandes para asegurar que la presencia de condiciones iníciales no afectaran el resultado. Hay dos desventajas de este método. La primera, desperdicia recursos; si los recursos son caros, una simulación no debería ser corrida por mucho tiempo si no es absolutamente necesaria. La segunda, aunque se generasen nuevas observaciones que no consuman algún recurso significativo, es difícil asegurar que la longitud de la corrida escogida es suficiente grande. Por esas dos razones, es recomendado que este método no se use.

3.2.3.2 Inicialización apropiada

Requiere iniciar la simulación en un estado cercano al estado estable esperado. Por ejemplo, una simulación de planificación de CPU podría iniciar con algunos trabajos en la

cola al inicio. El número de trabajos podría ser determinado desde simulaciones anteriores o por análisis simple. Este método impacta al reducir la longitud de periodos transitorios reduciendo así su efecto sobre todo el cálculo.

3.2.3.3 Truncamiento

Este y todos los métodos siguientes están basados en la suposición de que la variabilidad durante el estado estable es menor que durante el estado transitorio, lo cual generalmente es cierto. En el método de truncamiento, la variabilidad es medida en términos de rango (el mínimo y máximo de las observaciones). Si una trayectoria muestra observaciones exitosas se graba en un grafo de papel, el rango de observaciones se puede usar para estabilizar, tanto como la simulación entre en la fase del estado estable.

3.2.4 Simulaciones terminando (TERMINATING SIMULATIONS)

Aunque la importancia de la mayoría de las simulaciones es cuando alcanzan un estado estable, hay sistemas que nunca alcanzan un estado estable. Estos sistemas siempre operan bajo condiciones temporales. Por ejemplo, si el trafico de red consiste en transferir pequeños archivos (uno a tres paquetes), las simulaciones con un estado estable que usan archivos grandes no darán resultados de interés a un usuario típico. En tales casos, es necesario estudiar el sistema en estado transitorio. Tales simulaciones son llamadas simulaciones terminando. Otros ejemplos de simulaciones terminando son sistemas que se apagan, por instancia, a las 5 pm todos los días, o sistemas que tienen parámetros que cambian con el tiempo. Tales simulaciones no requieren eliminación de transitorios.

Otra tarea relacionada a las simulaciones terminando, es la de condiciones finales (condiciones al final de la simulación). El estado del sistema al final de la simulación podría no ser de estado estable. En tales casos, es necesario excluir la parte final de la respuesta de los cálculos del estado estable. Los métodos para esto, son similares a los usados para determinar periodos temporales iníciales.
Finalmente, el analista debe tener cuidado al manejar las entidades obtenidas al final de la simulación.

3.2.5 Criterio de paro: estimación de la varianza

Es importante que la longitud de la simulación se escoja adecuadamente. Si la simulación es demasiado corta, los resultados podrían ser altamente variables. De otra manera, si la simulación es demasiado larga, los recursos de computación y personas podrían ser innecesariamente desperdiciados. Por lo tanto se determinó que la simulación debería ser

ejecutada hasta que el intervalo de confianza para la respuesta principal se redujera a una amplitud deseada. Si la muestra de la media es $\bar{x}$ y su varianza es $Var(\bar{x})$, un intervalo de confianza $100(1 - \mp)\%$ para la media está dada por la Ecuación 9.

$$\bar{x} \pm z_{1-\alpha/2} Var\,(\bar{x}) \tag{9}$$

Recordemos que $z_{1-\alpha/2}$ es la $(1 - \pm / 2)$ ésima por debajo de una unidad normal variada.
La varianza de la muestra media de n independientes observaciones puede ser obtenida fácilmente de la varianza de las observaciones a través de la Ecuación 10.

$$Var(\bar{x}) = \frac{Var\,(x)}{n} \tag{10}$$

Esta fórmula sólo es válida si las observaciones son independientes. Desafortunadamente, las observaciones en la mayoría de las simulaciones no son independientes. Por ejemplo, en una simulación de encolamiento, si el tiempo de espera para el i-ésimo trabajo es grande, el tiempo de espera para el $(i + 1)$-ésimo trabajo sería grande también y viceversa. En este caso, los tiempos de espera sucesivos están altamente correlacionados y la fórmula anterior no puede ser usada para estimar la varianza del tiempo de espera. Para observaciones correlacionadas, la varianza de la media podría ser varias veces más grande que la obtenida de la fórmula. En un caso extremo, la varianza actual fue 300 veces que la obtenida calculada usando la formula. Ignorando este hecho podría resultar en intervalos de confianza angostos y la terminación prematura de la simulación.

3.3 Simulador de eventos discretos NS-2

NS[28] (más conocido como ns-2 por su versión actual) es un simulador de redes de eventos discretos. Utilizado principalmente en ambientes académicos debido a que está escrito en código abierto y a la abundancia de documentación en línea. Se pueden simular tanto protocolos unicast como multicast y se utiliza intensamente en la investigación de redes móviles ad-hoc. Puede simular una amplia gama de protocolos tanto para redes cableadas o redes inalámbricas, así como mixtas.
El código de NS se ofrece bajo la versión 2 de la Licencia Pública General GNU.

NS fue construido en C++ y proporciona una interfaz de simulación a través de OTcl, un dialecto orientado a objetos de Tcl. El usuario describe una topología de red escribiendo scripts Otcl y, a continuación, el programa principal de NS simula la topología con los parámetros especificados.

NS comenzó a desarrollarse en 1989 como una variante del simulador de red REAL. En 1995, NS había ganado el apoyo de DARPA, el proyecto Vint de LBL, Xerox PARC, UCB y USC/ISI.

NS ahora es desarrollado en colaboración entre una serie de investigadores e instituciones, incluida la SAMAN (con el apoyo de DARPA), CONSER (a través de la NSF), y ICIR (antes ACIRI). Sun Microsystems y la UCB Daedelus y Carnegie Mellon (citado por la página de inicio de NS por la adicción de código wireless), también han aportado grandes contribuciones.

La generación 3 de NS comenzó su desarrollo el 1 de julio de 2006 y se prevé una duración de cuatro años.

Capítulo 4

Especificación del protocolo

4.1 Introducción

El algoritmo de enrutamiento propuesto, denominado Protocolo de Enrutamiento Seguro Bajo Demanda Basado en Vector de Distancias (On Demand Secure Distance Vector Routing Protocol, OSDV) es capaz de implementar redes ad hoc dinámicas, seguras, auto iniciadas, además de proveer enrutamiento multisalto entre nodos móviles que participan en el establecimiento y mantenimiento de la red. OSDV permite a nodos móviles obtener rutas de una manera rápida y segura para nuevos destinos, y no requiere de nodos de propósito particular para mantener rutas hacia destinos que no están activos en la comunicación. OSDV permite a nodos responder a rupturas de enlaces y cambios en la topología de la red de manera oportuna. La operación de OSDV es libre de ciclos, y al evitar el problema de conteo al infinito del algoritmo Bellman-Ford[60] ofrece convergencia rápida cuando la topología de la red ad hoc cambie (comúnmente, cuando un nodo se nueve en la red). Cuando se presenta la ruptura de enlaces, OSDV hace que el conjunto de nodos afectados sean notificados de la ruptura de dichos enlaces, y así el conjunto de nodos invaliden las rutas que utilizan el enlace roto. La seguridad en la publicación de distancias por parte de los nodos se lleva a cabo por medio de cadenas hash y la seguridad de los datos inmutables, como la identidad de los nodos, se realiza por medio de firmas digitales.

4.2 Perspectiva general

Los tipos de mensajes definidos por OSDV son solicitudes de ruta (*RREQs*), respuesta a solicitud de ruta (*RREPs*), y errores de ruta (*RRERs*). Estos tipos de mensajes son recibidos vía UDP y, se aplica el procesamiento normal del encabezado IP. Un nodo solicitante usa su dirección IP como la dirección IP para los mensajes. Para mensajes broadcast, se usa la dirección IP limitada broadcast (255.255.255.255). La operación de OSDV requiere que ciertos mensajes sean diseminados ampliamente (e.g., RREQ). El rango de diseminación está indicado por el TTL en el encabezado IP.

Mientras existan rutas válidas entre un nodo origen y un destino, OSDV no interviene en la comunicación. Cuando se necesita una ruta hacia un nuevo destino, un nodo transmite de manera broadcast un *RREQ* para encontrar una ruta hacia el destino. Una ruta puede ser determinada cuando el *RREQ* alcanza al destino o a un nodo intermedio con una ruta actual hacia ese destino. Una ruta actual es determinada mediante el número secuencial del destino y es válida si al menos es tan grande como lo es el número secuencial del

mensaje *RREQ*. La ruta se establece mediante la transmisión unicast de un mensaje *RREP* de regreso al originador del *RREQ*. Cada nodo que recibe una solicitud de ruta almacena una ruta inversa hacia el originador del *RREQ*, la cual es usada para transmitir de manera unicast el RREP desde el destino a través del camino hacia el originador, o desde cualquier nodo intermedio que sea capaz de responder a una solicitud de ruta.

Los nodos de la red monitorean el estado de los enlaces de los siguientes saltos en rutas activas. Cuando se detecta un enlace roto en una ruta activa, se usa un mensaje *RRER* para notificar a otros nodos que tal enlace se ha roto. El mensaje *RRER* indica cuáles son los destinos que no son alcanzables debido al enlace roto.

OSDV es un protocolo de enrutamiento seguro, y trata con la administración de tablas de enrutamiento. La información de una tabla de enrutamiento es guardada incluso para rutas con tiempo de vida corto, como las creadas para almacenar temporalmente caminos inversos hacia nodos que origen *RREQs*. OSDV usa los siguientes campos con cada entrada en la tabla de rutas:

- Dirección IP del destino
- Número secuencial del destino
- Bandera del número secuencial válido del destino
- Otro estado y banderas de enrutamiento (e.g., válido, no válido, reparado, en reparación)
- Conteo de salto (número de saltos necesarios para alcanzar el destino)
- Dirección del siguiente salto
- Lista de precursores
- Tiempo de vida (tiempo de expiración o eliminación de la ruta)

La administración de números secuenciales es crucial para evitar ciclos de enrutamiento, incluso cuando se presenten rupturas de enlaces y un nodo no sea alcanzable para proporcionar su propia información sobre su número secuencial. Un destino es inalcanzable cuando se rompe un enlace o es desactivado. Cuando estas condiciones ocurren, la ruta es invalidada por operaciones que involucran el número secuencial y el marcar la entrada de la ruta en la tabla como inválida.

4.3 OSDV terminología

En esta sección se define otra terminología usada en OSDV que no está definida en [37].

ruta segura activa

Una ruta segura hacia un destino que tiene una entrada en la tabla de enrutamiento y que está marcada como válida. Sólo se pueden usar rutas seguras activas para reenviar paquetes.

broadcast

Broadcasting quiere decir transmitir paquetes a la dirección de broadcast limitada, 255.255.255.255. Un paquete broadcast podría no ser reenviado ciegamente, pero bradcasting es útil para habilitar la diseminación de mensajes OSDV en toda la red ad hoc.

destino

Una dirección IP a la cual paquetes de dato serán transmitidos. Un nodo sabe que él es el destino cuando su dirección aparece en el campo apropiado del encabezado IP de un paquete de datos. Las rutas para nodos destino son suministradas por el funcionamiento del protocolo OSDV, el cual acarrea la dirección IP del nodo destino deseado un mensajes de descubrimiento de ruta.

nodo retransmisor

Un nodo que acepta reenviar paquetes destinados a otro nodo, al retransmitirlos al siguiente salto más cercano al destino de manera unicast, esto se hace a lo largo de la ruta que se está estableciendo usando los mensajes de control para enrutamiento.

ruta establecida

Una ruta establecida para enviar paquetes de datos desde un nodo que está originando una operación de descubrimiento de ruta hacia su destino deseado

ruta inválida

Una ruta que ha expirado, denotada por un estado inválido en la entrada de la tabla de enrutamiento. Una ruta inválida no puede ser usada para reenviar

paquetes de datos, pero puede proveer información útil para reparaciones de rutas, y además para mensajes RREQ futuros.

nodo origen

Un nodo que crea un mensaje OSDV de descubrimiento de ruta segura para ser procesado y posiblemente retransmitido por otros nodos en la red ad hoc. Por ejemplo, el nodo que origina un proceso de descubrimiento de ruta y trasmite de manera broadcast el mensaje *RREQ* es llamado nodo origen del mensaje *RREQ*.

ruta inversa

Una ruta establecida para reenviar un paquete de respuesta (*RREP*) de regreso al origen desde el destino o desde un nodo intermedio que tiene una ruta para ese destino.

número secuencial

Un número incrementado de manera monótona mantenido por cada nodo originador. En los mensajes de control de enrutamiento del protocolo OSDV, el número secuencial es usado por otros nodos para determinar la actualidad (o frescura) de la información contenida en un paquete de control.

ruta segura válida

Ver ruta segura activa

función hash

Función para resumir o identificar de manera unívoca a un dato.

hash

Resultado de aplicar la función hash a un dato.

cadena hash

Es la acción de aplicar repetidamente una función hash a un dato.

4.4 Características Funcionales de OSDV

El protocolo de enrutamiento OSDV está diseñado para redes móviles ad hoc con cantidades de nodos de 10 hasta cientos de nodos móviles. OSDV puede manejar tasas de movilidad baja, moderada y relativamente alta, así como una variedad de niveles de tráfico. OSDV está diseñado para usarse en redes donde los nodos no pueden confiar entre ellos, o donde se sabe que existe el riesgo potencial de posibles ataques, además, se asume que existe un subsistema administrador de llaves el cual hace posible que los nodos dentro de una MANET obtengan sus respectivas llaves. Esto garantizará que los nodos puedan verificar que un nodo sea quien afirma ser (autenticación de fuente), adicionalmente los nodos tienen que ser capaces de verificar que la información no mutable de enrutamiento que reciben no haya sido alterada (integridad). OSDV está diseñado para proveer rutas seguras y debido a esto no siempre se proporcionan la ruta más corta hacia el destino, esto da a lugar a otras consecuencias como un mayor retardo de origen a destino y el empleo de más información de control diseminada en la red, sin embargo ofrece un mejor desempeño en la entrega de información de nodo origen a un nodo destino.

En la siguiente sección se muestra el formato de los mensajes de control empleados por el protocolo OSDV para establecer rutas seguras.

4.5 Formato de los mensajes

4.5.1 Formato del mensaje de solicitud de ruta (*RREQ*)

El formato del mensaje de solicitud de ruta (*RREQ*) se muestra en la figura 4-6, y contiene los siguientes campos:

Tipo 1

J Bandera de unión; reservado para multicast.

R Bandera de reparación; reservada para multicast.

G Bandera de *RREP* injustificado; indica si un *RREP* injustificado debería ser transmitido de manera unicast al nodo especificado en el campo de dirección IP.

D Bandera sólo para el destino; indica que sólo el destino puede responder a este *RREQ*.

U Número de secuencia desconocido; indica que el número de secuencia del destino es desconocido.

Reservado Enviado como 0; ingnorado en recepción.

Conteo de salto

El número de saltos desde la dirección IP del originador hasta el nodo que mantiene el *RREQ*.

Dirección IP Destino

La dirección IP del destino para el cual se desea una ruta.

Número secuencial del Destino

El último número secuencial recibido en el pasado por el originador para cualquier ruta hacia el destino.

Dirección IP Origen

La dirección IP del nodo el cual originó el *RREQ*.

Número secuencial del Origen

El número secuencial actual que será usado en el registro de la ruta que apunta hacia el originador del *RREQ*.

Top hash La hash tope para la autenticación del conteo de salto (*Hop Count*). Este campo tiene una longitud fija de 128 bits.

Hash La hash correspondiente al actual conteo de salto. Este campo tiene una longitud fija de 128 bits.

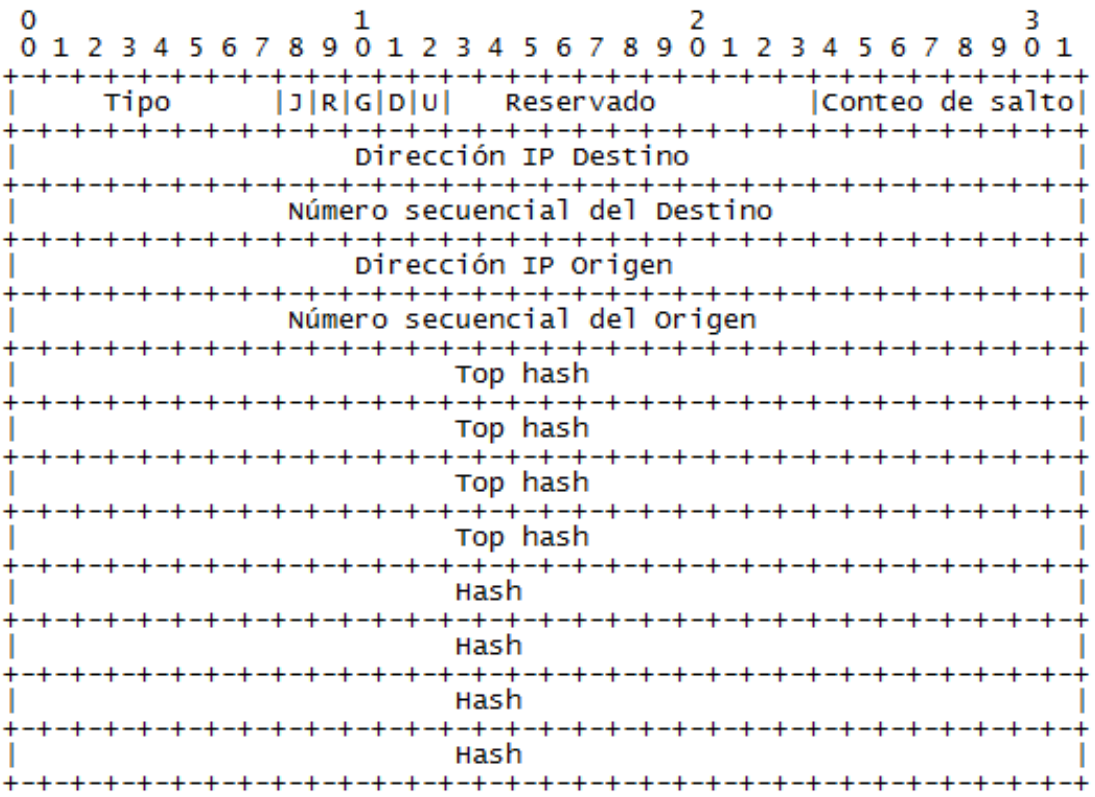


Figura 4-6. Formato del mensaje RREQ.

4.5.2 Formato del mensaje de respuesta de ruta (*RREP*)

El formato del mensaje de respuesta de ruta se muestra en la figura 4-7, y contiene los siguientes campos:

Tipo 2

R Bandera de reparación; usada para multicast.

A Reconocimiento requerido

Reservado Enviado como 0; ignorado en recepción.

Prefix Sz (Tamaño de prefijo)

Si no es cero, el quinto bit tamaño del prefijo especifica que el siguiente salto indicado podría ser usado por cualquier nodo con el mismo prefijo como el destino solicitado.

Conteo de salto

El número de saltos desde la dirección IP del origen hasta la dirección IP del destino.

Dirección IP Destino

La dirección IP del destino para el cuál una ruta es suministrada.

Número secuencial del Destino

El número secuencial asociado a la ruta.

Dirección IP Origen

La dirección IP del nodo el cuál originó el *RREQ* para el cual la ruta es suministrada.

Tiempo de vida

El tiempo en milisegundos que es considerado por los nodos que reciben el *RREP* para validar una ruta.

Top hash La hash tope para la autenticación del conteo de salto (*Hop Count*). Este campo tiene una longitud fija de 128 bits.

Hash La hash correspondiente al actual conteo de salto. Este campo tiene una longitud fija de 128 bits.

El bit 'A' es usado para indicar que el enlace sobre el cual el mensaje *RREP* es enviado podría ser no confiable o unidireccional. Cuando el mensaje *RREP* contiene el bit 'A' activo, se espera que el receptor del *RREP* regrese un mensaje *RREP-ACK*.

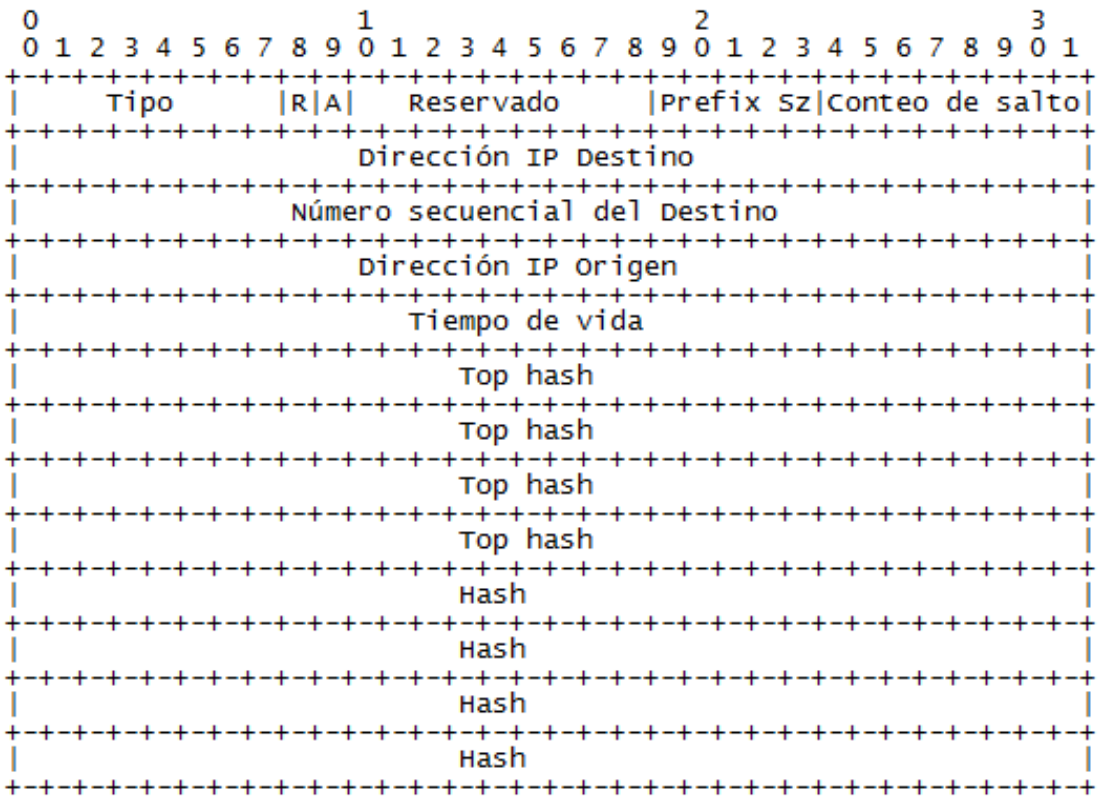


Figura 4-7. Formato del mensaje RREP.

4.5.3 Formato del mensaje de error de ruta (*RERR*)

El formato del mensaje de error de ruta se muestra en la figura 4-8, y contiene los siguientes campos:

Tipo 3

N Bandera de no eliminación; activada cuando un nodo ha realizado una reparación local de un enlace, y los nodos del flujo arriba no deberían de eliminar la ruta.

Reservado Enviado como 0; ignorado en la recepción.

Conteo Dest El número de los destinos inalcanzables incluidos en el mensaje. Es al menos 1.

Dirección IP del destino inalcanzable

La dirección IP del destino que se ha vuelto inalcanzable debido a una ruptura de enlace.

Número secuencial del destino inalcanzable

El número secuencial en el registro de la tabla de rutas para el destino listado en el campo previo Unreacheable Destination IP Address.

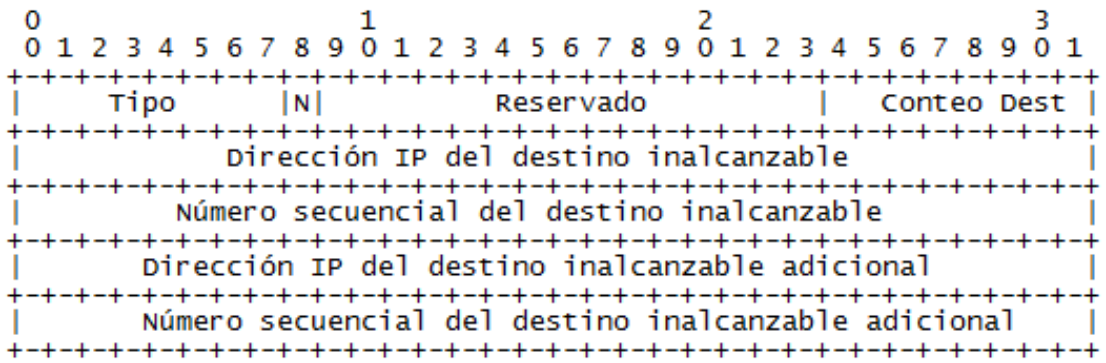


Figura 4-8. Formato del mensaje RERR.

El mensaje *RERR* es enviado siempre que un enlace roto causa que uno o más destinos se vuelvan inalcanzables desde alguno de los nodos vecinos.

4.5.4 Formato del mensaje de reconocimiento de respuesta de ruta (RREP-ACK)

El mensaje de reconocimiento de respuesta de ruta (*RREP-ACK*) es enviado en respuesta a un mensaje *RREP* con el bit 'A' activado. Eso es hecho cuando existe peligro de no completar correctamente el ciclo de un descubrimiento de ruta (se presentan enlaces unidireccionales).

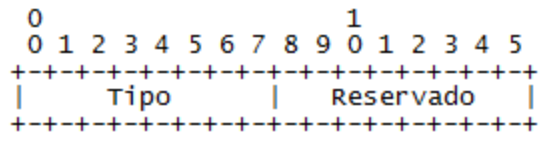


Figura 4-9. Formato del mensaje RREP-ACK

Tipo 4

Reservado Enviado como 0; ignorado en recepción.

4.6 Operación del protocolo OSDV

Esta sección describe los escenarios bajo los cuales los nodos generan mensajes de solicitudes de ruta (*RREQ*), respuestas de ruta (*RREP*) y error de ruta para llevar a cabo una comunicación unicast hacia un destino, y cómo son manejados los mensajes de datos. Con el propósito de procesar los mensajes correctamente, cierta información de estado tiene que ser mantenida en los registros de la tabla de enrutamiento para el destino. Además con el propósito de proporcionar niveles de seguridad a la operación del protocolo, se clasifica en información mutable e inmutable. La información inmutable es protegida por medio de firmas digitales mientras que la información mutable es protegida mediante el uso de cadenas hash.

Con el propósito de simplificar nuestra notación, en las siguientes subsecciones utilizaremos $h^x(y)$ para denotar que la función hash h es aplicada x veces al parámetro y. Por ejemplo, la expresión $h^3(5)$ es equivalente a la expresión $h(h(h(5)))$. Adicionalmente, para hacer más clara la especificación de nuestro protocolo, usaremos el operador flecha (←), para indicar asignación, y el operador igual (=) para indicar comparación.

4.6.1 Mantenimiento de Números Secuenciales

Cada registro de la tabla de enrutamiento de cada nodo incluye la última información disponible acerca del número secuencial para la dirección IP del nodo destino. Este número secuencial es llamado el número secuencial del destino. Éste es actualizado siempre que un nodo recibe nueva información acerca del número secuencial desde el mensaje *RREQ*, *RREP*, o *RERR* que podría ser recibido relacionado a ese destino. OSDV depende de cada nodo en la red que posee y mantiene su número secuencial del destino para garantizar que las rutas sean libres de ciclos hacia ese nodo. Un nodo destino incrementa su propio número secuencial bajo dos circunstancias:

- Inmediatamente antes de que un nodo origine un descubrimiento de ruta, debe incrementar su propio número secuencial. Esto previene conflictos con rutas inversas previamente establecidas hacia el originador de una *RREQ*.

- Inmediatamente antes de que un nodo destino origine un *RREP* en respuesta a un *RREQ*, debe actualizar su propio número secuencial al máximo del número secuencial actual y el número secuencial del destino en el paquete *RREQ*.

Cuando el destino incrementa su número secuencial, trata al valor del número secuencial como si fuera un número sin signo. Teniendo esto en cuenta, el valor del número secuencial máximo es el número más grande que se pueda representar con un entero sin singo de 32 bits (i.e., 4294967295), entonces cuando se incrementa se asignará un valor de cero (0).

Con el propósito de actualizar la información no reciente acerca del destino no reciente, el nodo compara su valor numérico actual con el número secuencial obtenido del mensaje entrante OSDV. Esta comparación se realiza usando aritmética con signo de 32 bits, esto es necesario para conseguir renovar el número secuencial. Si el resultado de restar el número secuencial actual almacenado del valor del número secuencial entrante es menor a cero, entonces la información relacionada a ese destino en el mensaje OSDV es descartada, ya que la información no es actual comparada con la información almacenada del nodo.

La otra circunstancia en la cual un nodo podría cambiar el número secuencial del destino en uno de sus registros en su tabla de enrutamiento, es al responder a una pérdida o expiración de un enlace al siguiente salto hacia ese destino.

Un nodo cambia el número secuencial en el registro de su tabla de enrutamiento de un destino sólo si:

- Él es el nodo destino, y ofrece una nueva ruta a él mismo, o
- Recibe un mensaje OSDV con información nueva acerca del número secuencial para un nodo destino, o
- El camino hacia el nodo destino se rompe o expira

4.6.2 Registros de la tabla de enrutamiento y listas de precursores

Cuando un nodo recibe un paquete de control OSDV de un vecino, o crea o actualiza una ruta para un destino en particular, él busca un registro de ruta en su tabla de enrutamiento para el destino. Cuando no exista un registro en la tabla de enrutamiento para ese destino, se crea uno. El número secuencial es determinado desde la información contenida en el paquete de control. La ruta es actualizada sólo si el nuevo número secuencial es:

I. Más grande que le número secuencial de destino almacenado en la ruta, o
II. Los números secuenciales son iguales, pero el conteo de salto (de la nueva información) más uno, es más pequeño que el conteo de salto existente en la tabla de enrutamiento, o
III. El número de secuencia es desconocido.

El campo de tiempo de vida de un registro en la tabla de enrutamiento es determinado desde el paquete de control, o es inicializado a ACTIVE_TIME_ROUTE. Esta ruta puede ser usada para enviar cualquier paquete de datos encolado y cumplir con cualquier solicitud de ruta (*RREQ*) pendiente.

Cada vez que una ruta es usada para reenviar paquetes de datos, se actualiza el campo de tiempo de vida de la ruta activa de la fuente, del destino y del siguiente salto en el camino hacia el destino. Este campo no tiene que ser menor que el tiempo actual más ACTIVE_ROUTE_TIMEOUT. Ya que se espera que la ruta entre cada origen y destino sea simétrica, el tiempo de vida de la ruta activa para el salto previo, junto con el tiempo de vida del camino inverso hacia la IP fuente, son actualizados, este tiempo no tiene que ser menor que el tiempo actual más ACTIVE_ROUTE_TIMEOUT. El tiempo de vida para una ruta activa es actualizo cada vez que la ruta es usada.

Para cada ruta válida mantenida por un nodo en el registro de esa ruta en la tabla de enrutamiento, el nodo también mantiene una lista de precursores que puede reenviar paquetes en esa ruta. Estos precursores recibirán notificación desde el nodo en el evento de detección de pérdida del enlace del siguiente salto. La lista de precursores en un registro de la tabla de enrutamiento contiene aquellos nodos del vecindario a los cuales se les envió o generó una respuesta de ruta.

4.6.3 Generación de solicitudes de ruta (*RREQ*)

Un nodo disemina un *RREQ* cuando determina que él necesita una ruta para un destino y no tiene una disponible. Esto puede suceder si el destino es desconocido para el nodo, o si una ruta válida previa hacia un destino expira o si es marcada como inválida. El campo del número secuencial del destino en el mensaje *RREQ* es el último número secuencial conocido para ese destino y es copiado del campo de número secuencial registrado en la tabla de enrutamiento. Si no se conoce el número secuencial, la bandera de número secuencial desconocido debe ser activada. El número secuencial del originador en el mensaje *RREQ* es su propio número secuencial, el cual es incrementado previamente antes de su inserción en un *RREQ*.

Un nodo no debe originar más *RREQ* por segundo que los especificados en el campo *RREQ_RATELIMIT*. Después de transmitir un *RREQ*, un nodo espera por un RREP (u otro mensaje de control relacionado al destino apropiado). Si una ruta no es recibida dentro de los milisegundos especificados en NET_TRVERSAL_TIME, el nodo puede intentar de nuevo descubrir una ruta al transmitir otro *RREQ*, hasta alcanzar el máximo permitido.

Los paquetes de datos que están esperando ser enviados a través de una ruta (es decir, esperando por un *RREP* después de haber enviado un *RREQ*) deben ser almacenados. El almacenamiento se realiza con semántica de "el primero en entrar es el primero en salir" (FIFO). Si no se recibió algún *RREP* para ese descubrimiento de ruta, todos los paquetes destinados para ese destino son eliminados del área de almacenamiento temporal y se envía un mensaje a la aplicación indicando que ese destino es inalcanzable.

Para reducir la congestión en una red, se utiliza un retardo exponencial por cada intento repetido de descubrimiento de ruta hecho por un nodo hacia otro nodo destino. La primera vez que un nodo fuente transmite un *RREQ*, espera el tiempo en milisegundos contenido en el campo NET_TRAVERSAL_TIME para la recepción de un *RREP*. Si no se recibe un *RREP* dentro de ese tiempo, el nodo fuente envía un nuevo *RREQ*. Para calcular el tiempo de espera por parte de un nodo fuente después de enviar el segundo *RREQ*, el nodo fuente debe un retardo exponencial binario. Entonces, el tiempo de espera para la llegada del *RREP* correspondiente al segundo *RREQ* es 2 * NET_TRAVERSAL_TIME milisegundos. Si no se recibe un *RREP* en este tiempo se puede enviar otro *RREQ* hasta alcanzar el límite permitido. Para cada intento adicional, el tiempo de espera para la recepción del *RREP* es multiplicado por 2.

4.6.3.1 Seguridad en la generación de RREQ

Con el propósito de evitar que la información mutable contenida en los *RREQ* sea modificada conforme a las políticas del protocolo, se usan cadenas hash para autenticar el campo *Hop Count* (*conteo de salto*). Las cadenas hash son aplicadas para verificar que el conteo de salto no haya sido decrementado por algún atacante y con ello prevenir el ataque de hoyo negro. Una cadena hash se construye al aplicar una función hash de sólo una dirección repetidamente a una semilla.

Para aplicar cadenas hash en la autenticación del conteo de salto, el proceso que se sigue cada vez que se genera un mensaje *RREQ* es el siguiente:

- Se genera un valor aleatorio (*semilla*)

- Se establece
    - Máximo conteo de salto $\leftarrow$ Diámetro de la red (Campo *network diameter*, que generalmente especifica al camino más largo posible)
- Se aplica la función hash al valor de *semilla* y se asigna al campo *Hash*
    - $Hash \leftarrow h(semilla)$
- Se calcula Top_Hash al aplicar la función hash a *semilla* tantas veces como valga el campo máximo conteo de salto
    - $Top\_Hash \leftarrow h^{máximo\ \ conteo\ \ de\ salto}\ (semilla)$

El procedimiento anterior se ejemplifica en la figura 4-10. En la figura se muestra que para un valor de semilla igual a uno, y un máximo conteo de saltos igual a 8, *Top_hash* es igual al resultado de aplicar ocho veces la función hash *h* a la semilla

$$semilla \leftarrow 1$$

$$Máximo\_conteo\_de\_salto \leftarrow 8$$

$$Hash \leftarrow h(semilla)$$

$$Top\_Hash \leftarrow h^{8}(semilla)$$

$$Top\_Hash \leftarrow h\left(h\left(h\left(h\left(h\left(h\left(h\left(h(semilla)\right)\right)\right)\right)\right)\right)\right)$$

Figura 4-10. Generación de cadena hash

4.6.4 Control de la diseminación de mensajes de solicitud de ruta (*RREQ*)

Para prevenir diseminación innecesaria de *RREQ* a lo ancho de la red, el nodo originador puede usar una técnica de búsqueda por expansión de anillo. En una búsqueda por

expansión de anillo, nodo origen inicialmente usa un TTL = TTL_START en el encabezado IP de un paquete *RREQ* y establece el tiempo límite igual a los milisegundos contenidos en el campo RING_TRAVERSAL_TIME para la recepción de un *RREP*. El tiempo almacenado en el campo RING_TRAVERSAL_TIME se calcula como se describe en la sección de parámetros de configuración (sección 4.7). El valor del campo TTL_VALUE usado para calcular el tiempo del campo RING_TRAVERSAL_TIME es asignado igual al valor del campo TTL contenido en el encabezado IP. Si el *RREQ* alcanza el límite de tiempo permitido sin recibir un RREP, el originador transmite de manera broadcast una vez más el *RREQ* con el TTL incrementado por TTL_INREMENT. Esto continúa hasta que el TTL establecido en el *RREQ* alcanza el límite contenido en el campo TTL_THRESHOLD.

El *conteo de salto* (*Hop Count*) almacenado en un registro de una ruta inválida indica el último conteo de salto conocido para ese destino en la tabla de enrutamiento. Cuando se requiere una nueva ruta para ese destino en un tiempo posterior (es decir, cuando se pierde la ruta), el TTL en el encabezado IP del *RREQ* inicialmente se establece con el valor del conteo de salto más TTL_INREMENT.

Un registro de una ruta que ha expirado almacenado en la tabla de enrutamiento no debe ser desechado antes del tiempo calculado por (current_time + DELETE_PERIOD) ver sección 4.6.9. De otra manera, se perderá el “soft state” correspondiente a la ruta (es decir, el último conteo de salto conocido). Cualquier registro de ruta que está esperando por un *RREP* no debe ser desechado antes de (current_time + 2 * NET_TRAVERSAL_TIME).

4.6.5 Procesamiento y reenvío de solicitudes de ruta (*RREQs*) de manera segura

Cuando un nodo recibe un *RREQ*, y con el propósito de autenticar el conteo de salto lleva a cabo la siguiente operación:

- Se aplica la función hash *h* al campo Hash tantas veces como sea necesario. Para llevar a cabo dicha acción primero se debe restar el valor del campo *conteo de salto* (*Hop_Count*) al valor del campo *máximo conteo de salto*, posteriormente se aplica la función hash tantas veces como valga el resultado de la operación anterior al valor contenido en el campo *Hash* y se verifica que el siguiente predicado sea verdadero:

$$Top\_hash = h^{máximo\ \ conteo\ \ de\ salto - conteo\ \ de\ salto}(Hash)$$

En este punto se verifica que el valor resultante sea igual al valor contenido en el campo *Top_Hash* para asegurar de esta manera que el conteo de salto publicado sea el correcto (ver ejemplo en la figura 4-11). En caso de no satisfacer la igualdad anterior se procede a eliminar silenciosamente el *RREQ* entrante.

Proceso realizado para la autenticación del conteo de salto

$Máximo\_conteo\_de\_salto \leftarrow 8$

$conteo\_de\_salto \leftarrow 3$

$Hash \leftarrow h\left(h\left(h(semilla)\right)\right)$: Es el Valor de la cadena *Hash* recibida en el nodo y vemos que corresponde al conteo de salto publicado en el campo *conteo de salto* (se aplicó la función hash 3 veces)

Para autenticar el conteo de salto verificamos que los siguientes campos sean iguales:

$Top\_Hash = h^{Máximo\ \_conteo\ \_de\ \_salto\ -conteo\ \_de\ \_salto}\ (Hash)$

$h^{8-3}(Hash)$

$h^{5}(Hash)$

$h\left(h\left(h\left(h\left(h(Hash)\right)\right)\right)\right)$

$h\left(h\left(h\left(h\left(h\left(h\left(h(semilla)\right)\right)\right)\right)\right)\right)$

El campo *Top_Hash* posee el valor: $h\left(h\left(h\left(h\left(h\left(h\left(h(semilla)\right)\right)\right)\right)\right)\right)$

Comparando las dos ecuaciones tenemos:

$$h\left(h\left(h\left(h\left(h\left(h\left(h(semilla)\right)\right)\right)\right)\right)\right) = h\left(h\left(h\left(h\left(h\left(h\left(h(semilla)\right)\right)\right)\right)\right)\right)$$

Al ser iguales auntenticamos el conteo de salto y se continúa con el procesamiento del *RREQ*.

Figura 4-11. Autenticación de conteo de salto.

Una vez auntenticado el conteo de salto, se graba el tiempo en que se recibió el *RREQ*, con el fin de comparar el tiempo de llegada de los *RREQ* posteriores y con ello prevenir el ataque llamado “one hop”. Este ataque se lleva a cabo cuando un nodo malicioso no publica su distancia en el mensaje *RREQ* que está reenviando y sólo reenvía el *RREQ* sin modificarlo, y con ello, aumentar sus posibilidades de estar en la nueva ruta.

Posteriormente, un nodo crea o actualiza una ruta al salto previo sin un número secuencial válido entonces aplica un criterio de eliminación de *RREQ*, el cual consiste en verificar si el *RREQ* entrante posee un número secuencial más reciente o verifica que se tenga un valor de conteo de salto menor para el mismo número secuencial. Si no se cumplen las condiciones anteriores se procede a eliminar silenciosamente el *RREQ*. El resto de este apartado describe las acciones que se toman para *RREQs* que no son eliminados.

Posteriormente, el nodo busca una ruta inversa hacia la IP del origen. Si se necesita, se crea la ruta o se actualiza en su tabla de enrutamiento usando el número secuencial del origen contenido en el *RREQ*. La ruta inversa se necesita si el nodo recibe un *RREP* de regreso hacia el nodo que originó el *RREQ* (identificado por la dirección IP del origen). Cuando la ruta inversa se crea o actualiza, se llevan a cabo las siguientes acciones sobre la ruta:

1. El número secuencial del origen del *RREQ* es comparado con el correspondiente número secuencial del destino en el registro de la ruta en la tabla de enrutamiento y se copia si es más grande que el valor existente
2. El campo del número secuencial válido es activado
3. El nodo del cual se recibió el *RREQ* se convierte en el siguiente salto en la tabla de enrutamiento.
4. El conteo de salto es copiado del campo de *conteo de salto (Hop Count)* del mensaje *RREQ*.

Siempre que se recibe un mensaje *RREQ* el valor del campo *Lifetime* del registro de la ruta inversa para la dirección IP del origen es colocado al máximo de (*ExistingLifetime*, *MinimalLifetime*), donde se cumple la Ecuación 11.

$$MinimalLifetime = (current\ time + 2 * NET\_TRAVERSAL\_TIME - 2 * HopCount * NODE\_TRAVERSAL\_TIME) \quad (11)$$

El nodo actual puede usar la ruta inversa para enviar paquetes de datos de la misma manera como lo hace con cualquier otra ruta registrada en la tabla de enrutamiento.

Si un nodo no genera un *RREP* (siguiendo las reglas de procesamiento del protocolo), y si el encabezado IP del mensaje entrante tiene un valor de TTL mayor a 1, el nodo actualiza y transmite de manera broadcast el *RREQ*. Para actualizar el *RREQ* , el TTL o el campo de salto límite en el encabezado IP saliente es decrementado en uno, para contar un nuevo salto a través de un nodo intermedio. El número secuencial del destino para el destino solicitado es establecido del correspondiente valor recibido en el mensaje *RREQ*.

Además, antes de reenviar el mensaje de solicitud de ruta (*RREQ*), un nodo aplica un vez más la función hash para contar el nuevo salto. Es decir: *Hash←h(Hash)*

4.6.6 Generar respuestas de ruta (*RREP*)

Un nodo genera un *RREP* cuando:

I. Él es el destino
II. Él tiene una ruta activa hacia el destino, el número secuencial del destino existente en el registro de la ruta para el destino es válido y más grande o igual que el número secuencial contenido en el *RREQ*, y la bandera de "sólo destino" ('D') no está activada.

Cuando se genera un mensaje *RREP*, u nodo copia la dirección IP del destino y el número secuencial del origen del mensaje *RREQ* dentro de los campos correspondientes en el mensaje *RREP*. El procesamiento es ligeramente diferente, dependiendo si el nodo que responde es el destino o si es algún nodo intermedio que tiene una ruta actual hacia ese destino.

4.6.6.1 Generación de respuestas de ruta (*RREP*) por parte del destino

Si el nodo que genera el *RREP* es el destino, debe de incrementar su número secuencial en uno si el número secuencial contenido en el paquete *RREQ* es igual al valor incrementado anterior. De otra manera, el destino no cambia su número secuencial antes de generar el mensaje *RREP*. El destino coloca su número secuencial dentro del campo Destination Sequence Number del *RREP*, e ingresa el valor cero en el campo de conteo de salto del RREP.

Además, con el propósito de evitar que la información mutable contenida en los campos del mensaje *RREP* no sea modificada de manera acorde a las políticas del protocolo, se usan cadenas hash para autenticar el campo *Hop Count (conteo de salto).*

Entonces, para aplicar cadenas hash en la autenticación del conteo de salto, el proceso que se sigue cada vez que se genera un mensaje *RREP* es el siguiente:

- Se genera un valor aleatorio (*semilla*)
- Se establece
  - $\text{Máximo conteo de salto} \leftarrow \text{Diámetro de la red}$
- Se aplica la función hash al valor de *semilla* y se asigna al campo Hash
  - $Hash \leftarrow h(semilla)$
- Se calcula Top_Hash al aplicar la función hash a *semilla* tantas veces como el valor del campo "máximo conteo de salto"
  - $Top\_hash = h^{máximo\ \ conteo\ \ de\ \ salto}(semilla)$

4.6.6.2 Generación de respuestas de ruta (RREP) por parte de nodos intermedios

Si el nodo que está generando un RREP no es el destino, sino un nodo intermedio que se encuentra a lo largo de ruta del origen al destino, ese nodo copia su propio número secuencial para el destino dentro del campo Destination Sequence Number en el mensaje *RREP*.

El nodo intermedio actualiza el registro de la ruta de reenvío al colocar el último salto (desde donde recibió el *RREQ*, como se indica en el campo de dirección IP de la fuente en el encabezado IP) dentro de la lista de precursores para el registro de la ruta de reenvío. Además, el nodo intermedio actualiza su registro de la ruta inversa en la tabla de enrutamiento para el nodo originador del *RREQ* (el registro para la dirección IP del origen del mensaje *RREQ*).

El campo tiempo de vida del RREP es calculado al restar el tiempo actual del tiempo de expiración en su registro de ruta en la tabla de enrutamiento.

Para publicar la información referente a su distancia en el campo *Hop Count* del mensaje *RREP*, el nodo coloca su distancia en saltos en dicho campo y para autenticar la distancia se usan cadenas hash siguiendo estos pasos:

- Se genera un valor aleatorio (*semilla*)
- Se establece
- $\text{Máximo conteo de salto} \leftarrow \text{Diámetro de la red}$
- Se aplica la función hash al valor de *semilla* tantas veces como el valor del campo de conteo de salto extraído de la tabla de enrutamiento y se asigna al campo *Hash*
    - $Hash \leftarrow h(semilla)$
- Se calcula *Top_Hash* al aplicar la función hash a *semilla* tantas veces como el valor del campo "máximo conteo de salto"
    - $Top\_hash \leftarrow h^{máximo\ \ conteo\ \ de\ salto}\ (semilla)$

4.6.7 Recibir y reenviar respuestas de ruta (RREP)

Cuando un nodo recibe un *RREP*, con el propósito de autenticar el conteo de salto lleva a cabo la siguiente operación:

- Se aplica la función hash *h* al campo Hash. Para llevar a cabo dicha acción primero se debe restar el valor del campo *conteo de salto* (*Hop_Count*) al valor del campo *máximo conteo de salto*. Posteriormente se aplica la función hash tantas veces como valga el resultado de la operación anterior al valor contenido en el campo Hash y se verifica que el siguiente predicado sea verdadero:

$$Top\_hash\ =\ h^{máximo\ \ conteo\ \ de\ salto\ -conteo\ \ de\ salto}\ (Hash)$$

En este punto se verifica que el valor resultante de aplicar la función hash al campo *Hash* sea igual al valor contenido en el campo *Top_hash* para asegurar así que el conteo de salto publicado sea el correcto. En caso de no satisfacer la igualdad anterior se procede a eliminar silenciosamente el *RREP* entrante.

Después de auntenticar el campo *Hop Count*, el nodo busca una ruta para el salto previo. Si es necesario se crea una ruta para dicho salto previo, pero sin un número de secuencia

válido (ver sección 4.6.2). Posteriormente, el nodo incrementa el valor del conteo de salto en uno, para contar el nuevo salto a través del nodo intermedio. A este valor incrementado se le llama “nuevo conteo de salto”. Entonces se crea la ruta de reenvío para este destino si aún no existe. De otra manera, el nodo compara el número secuencial del destino contenido en el mensaje *RREP* con el número secuencial del mismo destino almacenado en su tabla de enrutamiento. Una vez hecha la comparación, el registro de la ruta es actualizado bajo las siguientes condiciones:

i. El número secuencial en la tabla de enrutamiento es marcada como inválida en el registro de la ruta.
ii. El número secuencial del destino en el *RREP* es mayor que la copia del número secuencial del destino que posee el nodo y el valor desconocido es válido, o
iii. Los números secuenciales son iguales, pero la ruta es marcada como inactiva, o
iv. Los número secuenciales son iguales, pero el nuevo conteo de salto es menor que el conteo de salto del registro de la ruta.

Si el registro de la ruta para el destino es creado o actualizado, entonces se llevan a cabo las siguientes acciones:

- La ruta es marcada como activa,
- El número secuencial del destino es marcado como válido
- Se asigna el salto siguiente en el registro de la ruta al nodo del cual se recibió el *RREP*, el cuál es indicado por el campo de dirección IP de la fuente en el encabezado IP.
- Se asigna el valor del nuevo conteo de salto al campo conteo de salto.
- Se asigna el tiempo actual más el valor del tiempo de vida en el mensaje *RREP* al campo tiempo de expiración.
- Se asigna el número secuencial del destino contenido en el RREP al campo número secuencial del destino.

Si el nodo actual no es el destino del mensaje *RREP* indicado por la dirección IP del origen en el mismo mensaje, el nodo busca en su tabla de enrutamiento una ruta para el nodo origen para determinar el siguiente salto para el paquete *RREP*, y entonces retransmitir el *RREP* hacia el origen usando la ruta obtenida de la tabla de enrutamiento.

Cuando cualquier nodo transmite un *RREP*, la lista de precursores para el correspondiente nodo destino es actualizada al agregarle el siguiente nodo para el cual el *RREP* es reenviado.

4.6.8 Mantenimiento de la conectividad local

Cada nodo intermedio debe monitorear continuamente la conectividad con sus siguientes saltos activos. Un nodo puede mantener información precisa acerca de su conectividad hacia esos siguientes saltos activos, usando uno o más mecanismos de la capa de red, como se describe a continuación:

- Cualquier notificación de la capa de enlace, como los que provee el IEEE 802.11, los cuales pueden ser usados para determinar la conectividad cada vez que un paquete es transmitido hacia un siguiente salto activo. Por ejemplo, la ausencia de un ACK de la capa de enlace o falla para conseguir un CTS después de nviar un RTS, incluso después de un número máximo de intento de retransmisiones, indica la pérdida del enlace para este salto activo.
- Si la notificación de la capa 2 no está disponible, el reconocimiento pasivo debería de usarse cuando se espera que el siguiente salto retransmita el paquete, el reconocimiento se lleva a cabo al escuchar un intento de transmisión en el canal hecho por el siguiente salto. Si no se detecta la transmisión dentro de los milisegundos indicados en el campo *NEXT_HOP_WAIT* o el siguiente salto es el destino (y por lo tanto se supone que no se reenviará el paquete) uno de los siguientes métodos se usa para determinar la conectividad:
    - Recibir cualquier paquete desde el siguiente salto.
    - Transmitir de manera unicast un *RREQ* al siguiente salto, preguntando por una ruta hacia el siguiente salto.
    - Transmitir de manera unicastun mensaje de solicitud ICMP ECHO al siguiente salto.

Si no se puede detectar un enlace hacia el siguiente salto con los métodos anteriores, el nodo intermedio debe asumir que el enlace se ha perdido y debe tomar acciones correctivas al seguir los métodos especificados en la sección 4.6.9.

4.6.9 Mensajes de error de ruta (RERR), expiración de ruta y eliminación de ruta

Generalmente, el procesamiento de errores de ruta y ruptura de enlaces requiere los siguientes pasos:

- Invalidar rutas existentes
- Listar destinos afectados
- Determinar cuáles vecinos podrían ser afectados
- Entregar un REER apropiado a dichos vecinos

Un mensaje de error de ruta (*RERR*) puede ser enviado de manera broadcast (en caso de existir muchos precursores), unicast (si hay sólo un precursor), o de forma iterativa al enviar de manera unicast a todos los precursores (en caso de que el envío broadcast no sea apropiado). Incluso cuando se envía un mensaje *RRER* de manera unicast de forma iterativa a varios precursores, ese mensaje se considera un único mensaje de control. Un nodo no genera más mensajes RRER por segundo que los indicados por el campo *RERR_RATELIMIT*.

Un nodo inicia el proceso de un mensaje RERR en tres situaciones:

(i) Si detecta un enlace roto para el siguiente salto de una ruta activa en su tabla de enrutamiento mientras transmite datos (y, si se intentó una reparación de ruta pero no fue exitosa) o
(ii) Si recibe un paquete de datos destinado a un nodo para el cual no posee una ruta activa ni está reparándola, o
(iii) Si recibe un RERR de un vecino para una o más rutas activas.

Para el caso (i), el nodo primero crea una lista de los destinos inalcanzables que consiste del vecino inalcanzable y cualquier otro destino en la tabla de enrutamiento local que usa el vecino inalcanzable como siguiente salto.

Para el caso (ii), hay sólo un destino inalcanzable, el cual es el destino del paquete de datos que no puede ser entregado. Para el caso (iii), la lista consiste de los destinos en el RERR para los cuales existe un registro correspondiente en la tabla de enrutamiento local que tiene el transmisor del *RERR* recibido.

Algunos de los destinos inalcanzables en la lista podrían ser usados por los nodos del vecindario, y por lo tanto podría reenviarse un nuevo *RERR*. El *RERR* debe contener aquellos destinos que son parte de la lista de destinos inalcanzables creada y no debe tener una lista vacía de precursores.

El nodo o nodos del vecindario que deberían recibir el *RERR* son todos aquellos que pertenecen a una lista de precursores de al menos uno. En caso de que existe sólo un único vecino que necesita recibir el *RERR*, el *RERR* se envía de manera unicast. De otra manera el *RERR* es típicamente enviado de manera broadcast con los destinos inalcanzables, y sus correspondientes números secuenciales de los destinos, incluidos en

el paquete. El campo *DestCount* del paquete *RERR* indica el número de destinos inalcanzables.

Antes de transmitir el RERR, se realizan ciertas actualizaciones en la tabla de enrutamiento que podrían afectar los números secuenciales de los destinos para los destinos inalcanzables. Para cada uno de esos destinos, el registro correspondiente en la tabla de enrutamiento se actualiza de la siguiente forma:

1. El número secuencial del destino de este registro de ruta (si existe y es válido), es incrementado para las situaciones (i) y (ii) y para la situación (iii) se copia el del RERR entrante.
2. El registro de la ruta es invalidado al marcarlo como invalido en la tabla de enrutamiento.
3. Se actualiza el campo Lifetime asignándole el tiempo actual más el tiempo contenido en el campo *DELETE_PERIOD*. Antes de que transcurra este tiempo, el registro de la ruta no debe ser eliminado.

Se debe notar que el campo *Lifetime* en la tabla de enrutamiento posee dos roles, para una ruta activa es el tiempo de expiración y para una ruta inválida es el tiempo de eliminación.

4.6.10 Reparación local

Cuando se tiene un enlace roto en una ruta activa, un nodo que es parte del enlace roto puede escoger reparar el enlace localmente si el destino no está a una distancia en saltos mayor a la indicada por el campo MAX_REPAIR_TTL. Para reparar el enlace roto, el nodo incrementa el número secuencial del destino y entonces transmite de manera broadcast un RREQ para ese destino. El TTL del RREQ inicialmente debe ser establecido con el valor de la Ecuación 12:

$$\max(MIN\_REPAIR\_TTL, 0.5 * \#hops) + LOCAL\_ADD\_TTL \quad (12)$$

donde #hops es el número de saltos del emisor (nodo que está intentando reparar el enlace). Los intentos de reparación local a menudo son invisibles para el nodo origen, y siempre tendrán TTL >= MIN_REPAIR_TTL + LOCAL_ADD_TTL. El nodo que inicia la reparación espera durante el descubrimiento de ruta para recibir *RREPs* como respuesta al *RREQ*. Durante la reparación local los paquetes son almacenados temporalmente. Si la

reparación local no tuvo éxito, el nodo procede de a cuerdo a la sección 4.6.9 al transmitir un mensaje *RERR* para ese destino.

Por otra parte, si el nodo recibe uno o más *RREPs* durante el periodo de descubrimiento, primero compara el conteo de salto de la nueva ruta con el valor del conteo de salto almacenado en el registro de la ruta inválida para ese destino. Si el conteo de salto de la nueva ruta para ese destino es mayor que el previamente almacenado en el registro de la ruta conocida, el nodo transmite un mensaje *RERR* con el bit 'N' activado para ese destino.

Un nodo que recibe un mensaje *RERR* con la bandera 'N' activada no debe eliminar la ruta para ese destino. La única acción que debe tomar un nodo que recibe un RERR es la retransmisión del mensaje si se recibió del siguiente salto. Cuando el nodo origen recibe un mensaje *RERR* con la bandera 'N' activada, y si este mensaje proviene de su siguiente salto a lo largo de su ruta hacia el destino, entonces el nodo origen inicia el descubrimiento de ruta.

La reparación local de enlaces rotos para rutas algunas veces tiene como resultado el incremento de la longitud de los caminos para sus destinos. Reparar el enlace localmente tiene como objetivo incrementar el número de paquetes de datos que son capaces de ser entregados a los destinos ya que los paquetes de datos no serán eliminados mientras el *RERR* viaja hacia el nodo origen. Enviar un *RERR* al nodo origen después de una reparación local de un enlace roto puede permitir al nodo origen reconstruir una ruta.

4.6.11 Acciones tomadas después de un reinicio

Un nodo que participa en la red ad hoc lleva a cabo ciertas acciones después de reiniciarse, como eliminar todos sus números secuenciales guardados para todos los destinos, incluyendo su propio número secuencial. Sin embargo, podría haber nodos vecinos que estén usando a este nodo como un siguiente salto activo. Lo anterior podría crear ciclos de enrutamiento. Para prevenir esta posibilidad de ciclos, cada nodo después de su reinicio espera un periodo de tiempo contenido en el campo *DELETE_PERIOD* antes de transmitir cualquier mensaje de descubrimiento de ruta. Si el nodo recibe un paquete de control *RREQ*, *RREP* o *RERR*, debe crear registros de ruta así como la información de los números secuenciales proporcionada por los paquetes de control, pero no debe reenviar paquetes de control. Si el nodo recibe un paquete de datos para algún otro destino, debe de transmitir un *RERR* y restablecer el temporizador de espera para que expire después del tiempo actual más el tiempo contenido en el campo *DELETE_PERIOD*.

Mientras el nodo espera para ser nuevamente un nodo activo, ninguno de sus vecinos lo podrá usar como un salto siguiente activo. Su número secuencial se actualiza una vez que reciba un *RREQ* de cualquier otro nodo, así como el *RREQ* siempre lleva el máximo número secuencial del destino usado en la ruta. Si no recibe un *RREQ*, el nodo debe inicializar su número secuencial a cero.

4.7 Parámetros de configuración

Esta sección proporciona valores por omisión para algunos parámetros importantes asociados con las operaciones del protocolo OSDV. La modificación de estos parámetros puede afectar el rendimiento del protocolo. Cambiar el parámetro NODE_TRAVERSAL_TIME también cambia la estimación del nodo para el parámetro NET_TRAVERSAL_TIME, y por lo tanto deber hacerse con conocimiento adecuado acerca del comportamiento de otros nodos en la red ad hoc. El valor configurado para el parámetro MY_ROUTE_TIMEOUT debe ser al menos 2 * PATH_DISCOVERY_TIME.

Nombre del parámetro	Descripción	Valor
ACTIVE_ROUTE_TIMEOUT	Tiempo de vida para una ruta activa	3,000 Milliseconds
ALLOWED_HELLO_LOSS	Usado para calcular el tiempo de vida de un mensaje HELLO	2
BLACKLIST_TIMEOUT	Cantidad de RREQ permitidos para ser enviados por un nodo que intenta descubrir una ruta hacia un destino	RREQ_RETRIES * NET_TRAVERSAL_TIME
DELETE_PERIOD	Ver explicación en la parte de abajo	
HELLO_INTERVAL	Intervalo para el envío de mensajes HELLO	1,000 Millisegundos
LOCAL_ADD_TTL	Campo usado en la reparación local	2
MAX_REPAIR_TTL	Utilizado en el proceso de reparación de ruta	0.3 * NET_DIAMETER
MIN_REPAIR_TTL		Ver explicación en la parte de abajo
MY_ROUTE_TIMEOUT	Tiempo de vida para una nueva ruta active	2 * ACTIVE_ROUTE_TIMEOUT
NET_DIAMETER		Ver explicación en la parte de abajo
NET_TRAVERSAL_TIME	Tiempo estimado permitido para que un paquete *RREQ* alcance su destino, cuando no es el primer intento	2 * NODE_TRAVERSAL_TIME * NET_DIAMETER
NEXT_HOP_WAIT		NODE_TRAVERSAL_TIME +

		10
NODE_TRAVERSAL_TIME	Tiempo estimado permitido para que un paquete *RREQ* alcance su destino, cuando es el primer intento	40 milliseundos
PATH_DISCOVERY_TIME	Tiempo estimado en descubrir una ruta	2 * NET_TRAVERSAL_TIME
RERR_RATELIMIT	Usado para indicar la cantidad de mensajes *RRER* enviados por un nodo en un segundo	10
RING_TRAVERSAL_TIME	Tiempo límite para la recepción de un mensaje *RREP*	2 * NODE_TRAVERSAL_TIME * (TTL_VALUE + TIMEOUT_BUFFER)
RREQ_RETRIES	Número de mensajes *RREQ* enviados despues de no haber recibido un *RREP*	2
RREQ_RATELIMIT	Usado para indicar la cantidad de mensajes *RREQ* enviados por un nodo en un segundo	10
TIMEOUT_BUFFER	Tiempo de espera adicional en la espera de la llegada de un mensaje *RREP*	2
TTL_START	Usados en la técnica de búsqueda por expansión de anillo ver sección 4.6.4	1
TTL_INCREMENT		2
TTL_THRESHOLD		7
TTL_VALUE	Ver explicación en la parte de abajo	

El parámetro MIN_REPAIR_TTL debe ser el último conteo de salto conocido al destino.

El parámetro TTL_VALUE es el valor del campo TTL contenido en el encabezado IP mientras se hace búsqueda empleando la técnica de expansión de anillo. El parámetro TIMEOUT_BUFFER es configurable. Su propósito es proveer un buffer para el tiempo límite, así que si un RREP se retrasó debido a la congestión, es menos probable que el tiempo límite de espera se agote mientras el RREP está todavía en camino de regreso a la fuente. Para omitir el buffer, se puede establecer el parámetro TIMEOUT_BUFFER = 0.

El parámetro DELETE_PERIOD tiene como objetivo proveer un límite superior sobre el tiempo para el caso donde un nodo A cercano a la fuente tenga un vecino B como un siguiente salto activo para un destino D, mientras B ha invalidado la ruta hacia D. Posteriormente B puede eliminar la ruta hacia D. La determinación del límite depende de algunas de las características de la capa de enlace. Si se usa la retroalimentación de la

capa de enlace para detectar la pérdida del enlace, el parámetro DELETE_PERIOD debe ser al menos el valor contenido en ACTIVE_ROUTE_TIMEOUT.

Finalmente, NET_DIAMETER mide el número máximo posible de saltos entre dos nodos en la red.

Capítulo 5

Pruebas y Resultados

5.1 Pruebas

En esta sección se presentan varias pruebas realizadas al protocolo de enrutamiento OSDV. Para esto, se definieron un conjunto de escenarios para evaluar el desempeño y la escalabilidad del protocolo. Estos escenarios se describen a continuación.

Escenario 1: Se varía la cantidad de adversarios.

Escenario 2: Se varía la cantidad de nodos (densidad)

Escenario 3: Se varía el tiempo de pausa de cada nodo

Escenario 4: Se varía la cantidad de flujos

Escenario 5: Se varía el área de simulación (densidad – longitud del camino más largo posible)

Los escenarios anteriores se simularon con el propósito de medir el desempeño del protocolo en cuanto a la capacidad del protocolo para entregar paquetes de datos, el tiempo promedio de entrega que toman los paquetes de datos y la cantidad de paquetes de control empleados en labores de enrutamiento desde el origen al destino,.

El parámetro de comparación usado para evaluar el desempeño del protocolo OSDV es el protocolo AODV. Esto debido a que AODV es un estándar de facto en protocolos de enrutamiento para MANETs por lo que es generalmente usado para realizar análisis comparativos de resultados experimentales. Adicionalmente, a diferencia de lo que sucede con la mayoría de los protocolos de enrutamiento descritos en el Capítulo 2, la distribución estándar de NS-2 cuenta con una implementación robusta y optimizada de AODV. Finalmente, el desarrollo del protocolo OSDV está basado en AODV, es decir, ambos usan vector de distancias, números secuenciales, mensajes *RERR*, mensajes *RREQ* y mensajes *RREP* para el establecimiento de rutas. Lo anterior nos permite identificar el impacto que tienen en el desempeño de nuestro protocolo los mecanismos de seguridad propuestos, es decir, nos permite cuantificar las ganancias y costos en cuanto a desempeño que se obtienen al utilizar firmas digitales, cadenas hash y ordenación en tiempo. Dichos mecanismos son usados para encontrar rutas seguras en una MANET.

5.1.1 Modelo del atacante

El modelo del atacante para las pruebas realizadas consiste en atacantes activos. Este atacante activo para los escenarios simulados tiene la capacidad de modificar el campo de

conteo de salto con el fin de crear un hoyo negro y así atraer el tráfico hacia él. Además, se clasificó a este tipo de atacante con base al número de nodos que ha comprometido como *Atacante-y-0* ya que el atacante puede comprometer a todos los nodos de la red, pero no posee algún nodo.

5.1.2 Parámetros de simulación

De a cuerdo a los a los escenarios descritos anteriormente se varían los parámetros indicados en el tipo de escenario.

Escenario1: Número de adversarios

Sistema operativo	Ubuntu Karmic 9.10	Sistema operativo utilizado para instalar y ejecutar el simulador ns-2
Simulador	NS-2.34	Simulador de eventos discretos empleado para simular los escenarios propuestos en su versión 2.34
Número de nodos	100	Cantidad de nodos empleados en el escenario de simulación
Número de adversarios	0, 10, 20, 30, 40, 50	Cantidad de adversarios. Para este escenario se variaron desde 0 hasta 50 adversarios
Área de simulación	1500 x 1500 metros	Área delimitada para que los nodos puedan interactuar entre ellos
Tiempo de simulación	500 segundos	El tiempo total empleado en la simulación
Radio de propagación	250 metros	Radio de propagación empleado por un nodo al emitir o recibir paquetes
Modelo de movilidad	Random waypoint	Tipo de movilidad usado en la simulación el cuál proporciona tiempo de movimiento aleatorio y tiempo de pausa
Tiempo de pausa de los nodos	5 segundos	Tiempo de espera por parte de un nodo después de un movimiento aleatorio
Velocidad máxima de los nodos	10 m/s	Velocidad máxima con la que los nodos se mueven de un punto a otro
Velocidad mínima de los nodos	1 m/s	Velocidad mínima con la que los nodos se mueven de un punto a otro
Número de flujos concurrentes	20	Cantidad de flujos existentes en la simulación en los cuales su inicio y fin son aleatorios
Tipo de tráfico	CBR (UDP)	Tipo de trafico utilizado para simular el envío de datos desde la capa de aplicación a través de protocolo de

		transporte UDP
Cantidad de semillas	10	Cantidad de semillas empleadas en la generación de tráfico.
Número de paquetes de datos	1000	Cantidad de paquetes de datos empleados para los flujos de la simulación
Tamaño de los paquetes de datos	512 bytes	Tamaño de cada paquete de datos

Tabla 5-1. Parámetros de simulación para escenario 1.

Escenario2: Número de nodos (densidad)

Sistema operativo	Ubuntu Karmic 9.10	Sistema operativo utilizado para instalar y ejecutar el simulador ns-2
Simulador	NS-2.34	Simulador de eventos discretos empleado para simular los escenarios propuestos en su versión 2.34
Número de nodos	50, 100, 150	Cantidad de nodos empleados en el escenario de simulación. Para este escenario se variaron desde 0 hasta 150
Número de adversarios	20%	El escenario se simuló con 20% de adversarios en relación con la cantidad de nodos
Área de simulación	1500 x 1500 metros	Área delimitada para que los nodos puedan interactuar entre ellos
Tiempo de simulación	500 segundos	El tiempo total empleado en la simulación
Radio de propagación	250 metros	Radio de propagación empleado por un nodo al emitir o recibir paquetes
Modelo de movilidad	Random waypoint	Tipo de movilidad usado en la simulación el cuál proporciona tiempo de movimiento aleatorio y tiempo de pausa
Tiempo de pausa de los nodos	5 segundos	Tiempo de espera por parte de un nodo después de un movimiento aleatorio
Velocidad máxima de los nodos	10 m/s	Velocidad máxima con la que los nodos se mueven de un punto a otro
Velocidad mínima de los nodos	1 m/s	Velocidad mínima con la que los nodos se mueven de un punto a otro
Número de flujos concurrentes	20	Cantidad de flujos existentes en la simulación en los cuales su inicio y fin son aleatorios
Tipo de tráfico	CBR (UDP)	Tipo de trafico utilizado para simular el

		envío de datos desde la capa de aplicación a través de protocolo de transporte UDP
Cantidad de semillas	10	Cantidad de semillas empleadas en la generación de tráfico.
Número de paquetes de datos	1000	Cantidad de paquetes de datos empleados para los flujos de la simulación
Tamaño de los paquetes de datos	512 bytes	Tamaño de cada paquete de datos

Tabla 5-2. Parámetros de simulación para escenario 2.

Escenario3: Tiempo de pausa

Sistema operativo	Ubuntu Karmic 9.10	Sistema operativo utilizado para instalar y ejecutar el simulador ns-2
Simulador	NS-2.34	Simulador de eventos discretos empleado para simular los escenarios propuestos en su versión 2.34
Número de nodos	100	Cantidad de nodos empleados en el escenario de simulación
Número de adversarios	20%	El escenario se simuló con 20% de adversarios en relación con la cantidad de nodos
Área de simulación	1500 x 1500 metros	Área delimitada para que los nodos puedan interactuar entre ellos
Tiempo de simulación	500 segundos	El tiempo total empleado en la simulación
Radio de propagación	250 metros	Radio de propagación empleado por un nodo al emitir o recibir paquetes
Modelo de movilidad	Random waypoint	Tipo de movilidad usado en la simulación el cuál proporciona tiempo de movimiento aleatorio y tiempo de pausa
Tiempo de pausa de los nodos	0, 5, 10, 20, 40 segundos	Tiempo de espera por parte de un nodo después de un movimiento aleatorio. Para este escenario se variaron los tiempos de pausa de 0 a 40 segundos
Velocidad máxima de los nodos	10 m/s	Velocidad máxima con la que los nodos se mueven de un punto a otro
Velocidad mínima de los nodos	1 m/s	Velocidad mínima con la que los nodos se mueven de un punto a otro
Número de flujos concurrentes	20	Cantidad de flujos existentes en la simulación en los cuales su inicio y fin son aleatorios
Tipo de tráfico	CBR (UDP)	Tipo de trafico utilizado para simular el

		envío de datos desde la capa de aplicación a través de protocolo de transporte UDP
Cantidad de semillas	10	Cantidad de semillas empleadas en la generación de tráfico.
Número de paquetes de datos	1000	Cantidad de paquetes de datos empleados para los flujos de la simulación
Tamaño de los paquetes de datos	512 bytes	Tamaño de cada paquete de datos

Tabla 5-3 Parámetros de simulación para escenario 3.

Escenario4: Número de flujos

Sistema operativo	Ubuntu Karmic 9.10	Sistema operativo utilizado para instalar y ejecutar el simulador ns-2
Simulador	NS-2.34	Simulador de eventos discretos empleado para simular los escenarios propuestos en su versión 2.34
Número de nodos	100	Cantidad de nodos empleados en el escenario de simulación
Número de adversarios	20%	El escenario se simuló con 20% de adversarios en relación con la cantidad de nodos
Área de simulación	1500 x 1500 metros	Área delimitada para que los nodos puedan interactuar entre ellos
Tiempo de simulación	500 segundos	El tiempo total empleado en la simulación
Radio de propagación	250 metros	Radio de propagación empleado por un nodo al emitir o recibir paquetes
Modelo de movilidad	Random waypoint	Tipo de movilidad usado en la simulación el cuál proporciona tiempo de movimiento aleatorio y tiempo de pausa
Tiempo de pausa de los nodos	0, 5, 10, 20, 40 segundos	Tiempo de espera por parte de un nodo después de un movimiento aleatorio
Velocidad máxima de los nodos	10 m/s	Velocidad máxima con la que los nodos se mueven de un punto a otro
Velocidad mínima de los nodos	1 m/s	Velocidad mínima con la que los nodos se mueven de un punto a otro
Número de flujos concurrentes	1, 10, 20, 30	Cantidad de flujos. Para este escenario se varia la cantidad de flujos de 1 hasta 30
Tipo de tráfico	CBR (UDP)	Tipo de trafico utilizado para simular el envío de datos desde la capa de aplicación a través de protocolo de transporte UDP

Cantidad de semillas	10	Cantidad de semillas empleadas en la generación de tráfico.
Número de paquetes de datos	1000	Cantidad de paquetes de datos empleados para los flujos de la simulación
Tamaño de los paquetes de datos	512 bytes	Tamaño de cada paquete de datos

Tabla 5-4 Parámetros de simulación para escenario 4.

Escenario5: Área de simulación (densidad – longitud del camino más largo posible)

Sistema operativo	Ubuntu Karmic 9.10	Sistema operativo utilizado para instalar y ejecutar el simulador ns-2
Simulador	NS-2.34	Simulador de eventos discretos empleado para simular los escenarios propuestos en su versión 2.34
Número de nodos	100	Cantidad de nodos empleados en el escenario de simulación
Número de adversarios	20%	El escenario se simuló con 20% de adversarios en relación con la cantidad de nodos
Área de simulación	1500 x 1500, 1500 x 3000, 3000 x 3000 metros	Área delimitada para que los nodos puedan interactuar entre ellos. Para este escenario se variaron las áreas de simulación
Tiempo de simulación	500 segundos	El tiempo total empleado en la simulación
Radio de propagación	250 metros	Radio de propagación empleado por un nodo al emitir o recibir paquetes
Modelo de movilidad	Random waypoint	Tipo de movilidad usado en la simulación el cuál proporciona tiempo de movimiento aleatorio y tiempo de pausa
Tiempo de pausa de los nodos	5 segundos	Tiempo de espera por parte de un nodo después de un movimiento aleatorio
Velocidad máxima de los nodos	10 m/s	Velocidad máxima con la que los nodos se mueven de un punto a otro
Velocidad mínima de los nodos	1 m/s	Velocidad mínima con la que los nodos se mueven de un punto a otro
Número de flujos concurrentes	20	Cantidad de flujos existentes en la simulación en los cuales su inicio y fin son aleatorios
Tipo de tráfico	CBR (UDP)	Tipo de trafico utilizado para simular el envío de datos desde la capa de aplicación a través de protocolo de transporte UDP

Cantidad de semillas	10	Cantidad de semillas empleadas en la generación de tráfico.
Número de paquetes de datos	1000	Cantidad de paquetes de datos empleados para los flujos de la simulación
Tamaño de los paquetes de datos	512 bytes	Tamaño de cada paquete de datos

Tabla 5-5 Parámetros de simulación para escenario 5.

5.1.3 Métricas de desempeño

Las métricas empleadas para la evaluación del desempeño del protocolo OSDV son las siguientes:

- Porcentaje de paquetes entregados a los receptores (Delivery ratio).. Se refiere a la cantidad de paquetes de datos entregados de manara exitosa desde el origen al destino.
- Sobre carga de control (control overhead). Se refiere a la cantidad de paquetes de control empleados para la operación del protocolo OSDV, es decir, para el descubrimiento, establecimiento y mantenimiento de rutas.
- Retardo extremo a extremo (End-to-end delay). Se refiere al tiempo tomado en la entrega de paquetes de datos desde el origen al destino.

5.2 Resultados

A continuación se muestra un resumen de los resultados que se obtuvieron con base a las pruebas que se realizaron. En las pruebas realizadas se emplearon scripts de simulación escritos en el lenguaje OTcl[41] y en la figura 5-12 se muestra el resultado arrojado en la terminal al correr un script de simulación para el protocolo OSDV.

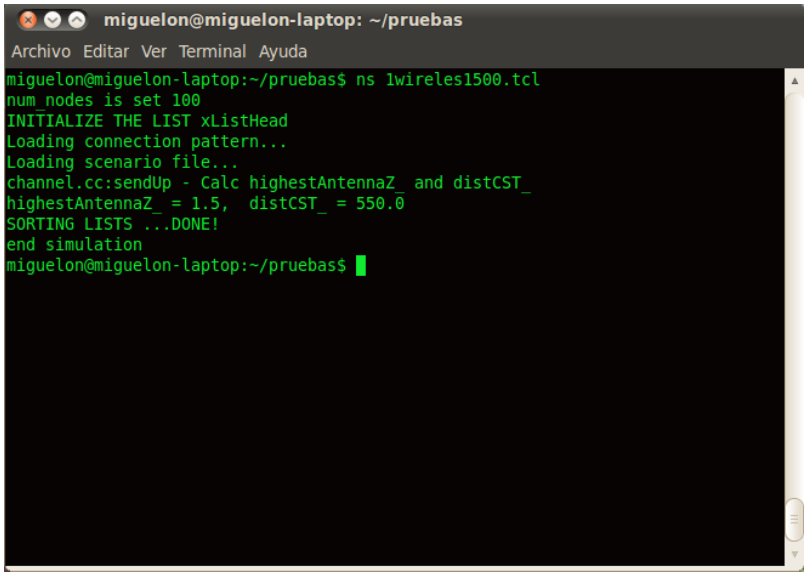


Figura 5-12. Imagen de la terminal al ejecutar un script de simulación.

En las figuras 5-13, 5-14 y 5-15 podemos encontrar los resultados arrojados para el escenario1 (número de adversarios), donde podemos observar que el desempeño del protocolo OSDV variando la cantidad de nodos maliciosos. Primeramente podemos ver el resultado del porcentaje de paquetes entregados a los receptores (figura 5-13). Aquí se puede visualizar que el desempeño del protocolo OSDV es superior, ya que es capaz de entregar una cantidad mayor de paquetes de datos a los receptores. En la gráfica podemos observar que el desempeño del protocolo se mantiene aún y cuando se han agregado más adversarios.

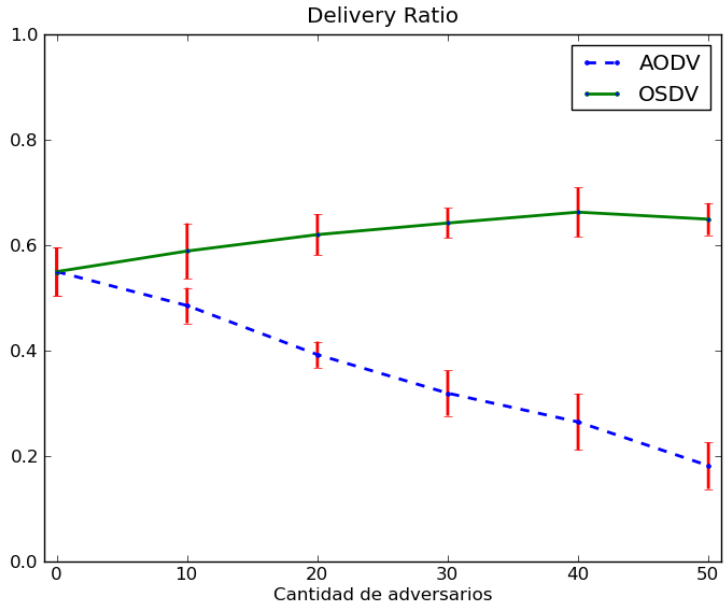


Figura 5-13. Delivery ratio para escenario 1.

Por otra parte encontramos que el protocolo OSDV envía más paquetes de control (figura 5-14) y el retardo de entrega de paquetes (figura 5-15) es mayor al protocolo AODV, esto se debe a que con la detección de nodos maliciosos, se tiene que establecer rutas que no son las más cortas hacia los destinos, provocando dicho incremento en el tiempo de entrega de los paquetes y los paquetes de control enviados.

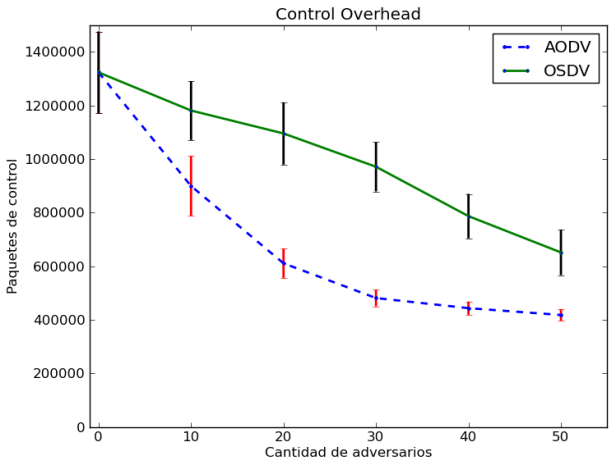


Figura 5-14. Control overhead para escenario 1.

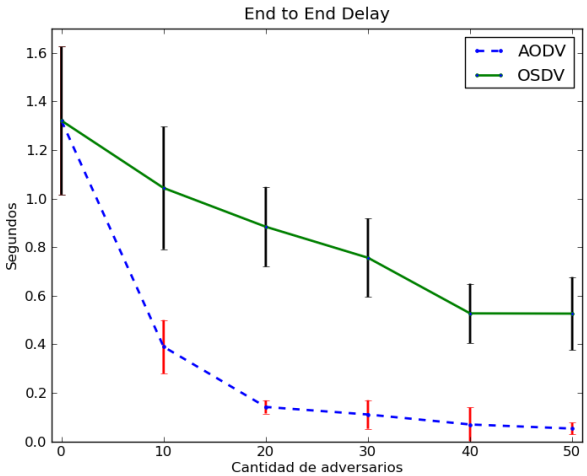


Figura 5-15. End to end delay para escenario 1.

Los resultados para el escenario número 2 (número de nodos) se muestran en las figuras 5-16, 5-17 y 5-18. Variando la cantidad de nodos en este escenario encontramos que la entrega de paquetes de datos a los destinos es mejor con el protocolo OSDV. Además se puede ver que el desempeño de los dos protocolos decae cuando se agregan más nodos al escenario, pero aún con estas condiciones el protocolo OSDV se mantiene por arriba del protocolo AODV. El desempeño del protocolo OSDV decae debido a la incorporación de más nodos y nodos maliciosos a la misma área de simulación, entonces el área se satura provocando que los nodos destinos sean inalcanzables.

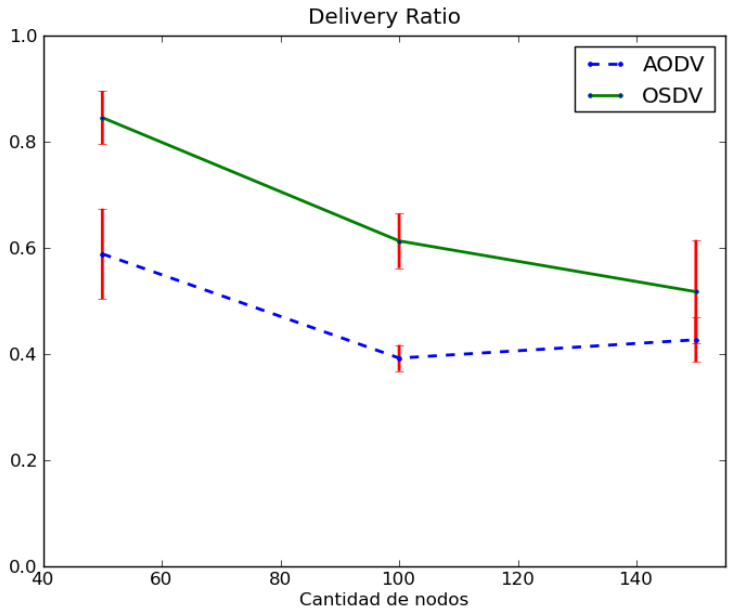


Figura 5-16. Delivery ratio para escenario 2.

Las figuras 5-17 y 5-18 muestran el comportamiento del protocolo en presencia de más nodos, éstas presentan un aumento tanto de paquetes de control (figura 5-17) como del tiempo en la entrega de paquetes (figura 5-18). Esto es debido a que los nodos destinos se vuelven inalcanzables, entonces se tienen que establecer rutas más largas (tiempos más grandes) y emplear más paquetes de control.

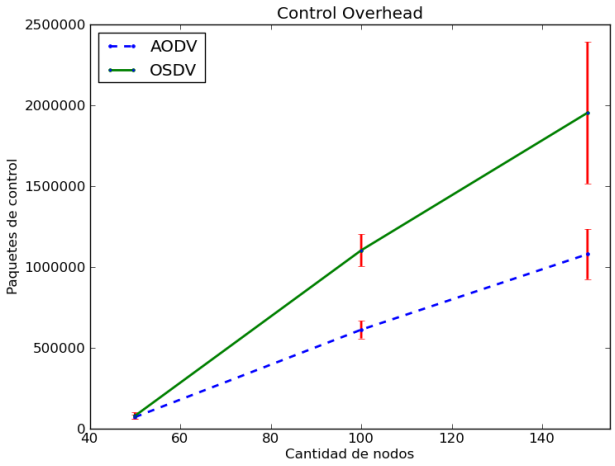


Figura 5-17. Control overhead para escenario 2.

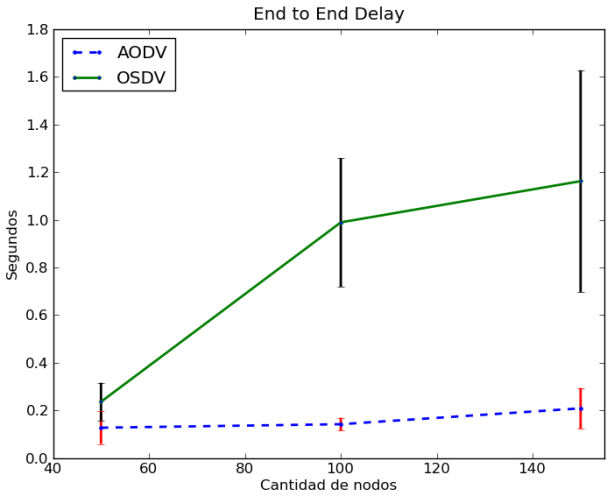


Figura 5-18. End to end delay para escenario 2.

Los resultados obtenidos para el escenario número 3 (tiempo de pausa) los podemos ver en las figuras 5-19, 5-20 y 5-21. Aquí podemos encontrar que la entrega de paquetes a los destinos por parte del protocolo OSDV se mantiene por encima de la entrega de paquetes proporcionada por el protocolo AODV (figura 5-19). La cantidad de adversarios en el escenario sigue siendo el 20% y cuando se incrementa el tiempo de pausa se puede apreciar que el protocolo OSDV entrega más paquetes de datos que el protocolo AODV, porque a pesar de la existencia de los nodos maliciosos, puede establecer rutas hacia los destinos.

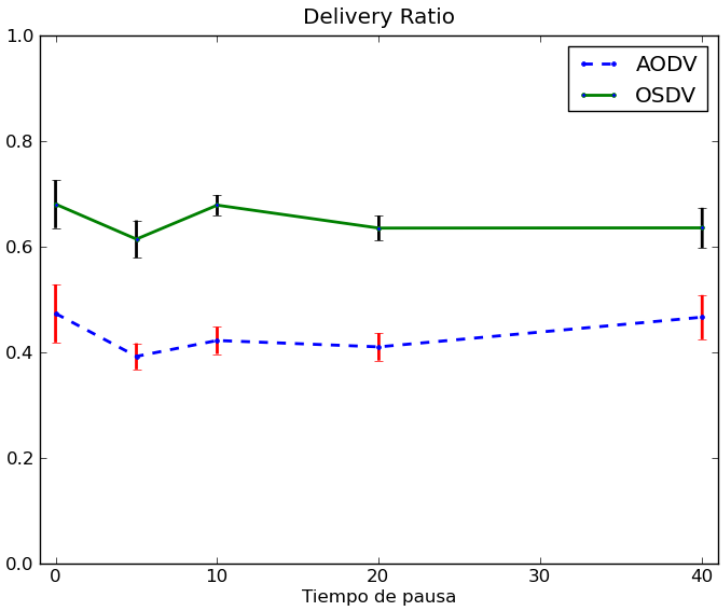


Figura 5-19. Delivery ratio para escenario 3.

Se puede apreciar que para llevar a cabo una entrega mayor de paquetes de datos en el escenario número 3 se emplean más paquetes de control como lo muestra la figura 5-20, y por lo tanto el tiempo de entrega se incrementa (figura 5-21). Las rutas hacia los nodos siguen siendo más grandes en el protocolo OSDV que en el protocolo AODV aún y cuando se incremente el tiempo de pausa de cada nodo, lo que provoca que el protocolo OSDV envíe más paquetes de control y tarde más tiempo en entregar los paquetes de datos.

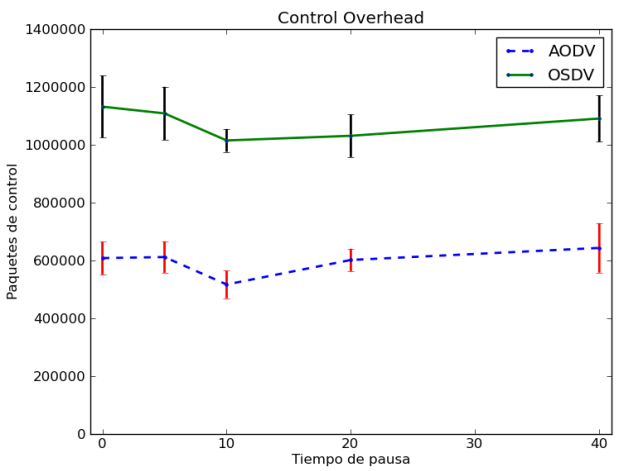


Figura 5-20. Control overhead para escenario 3.

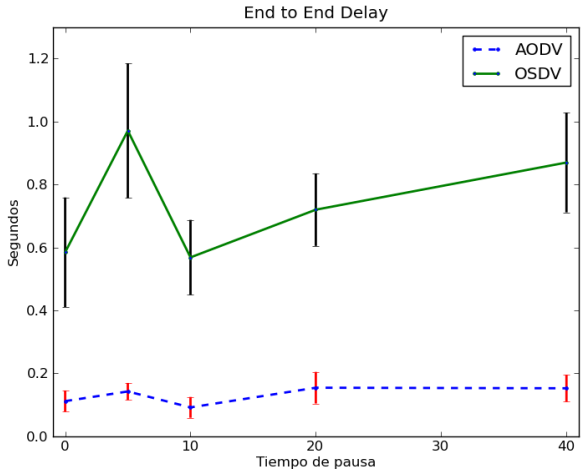


Figura 5-21. End to end delay para escenario 3.

Para el escenario donde varía la cantidad de flujos en la red móvil, los resultados se muestran en las figuras 5-22, 5-23 y 5-24. Se puede observar que conforme se incrementa la cantidad de flujos en la MANET (mostrado en la figura 5-22) el rendimiento de ambos protocolos decae debido a que los nodos de la red tienen que mantener paquetes de datos y control en sus colas de prioridad, provocando que en algún momento se saturen y no sean capaces de procesar todos los paquetes recibidos. El protocolo OSDV sigue teniendo la capacidad de entregar una mayor cantidad de paquetes de datos que el protocolo AODV, debido a que establece rutas seguras hacia los destinos mientras que el protocolo AODV no evita los nodos maliciosos (20% del total de nodos).

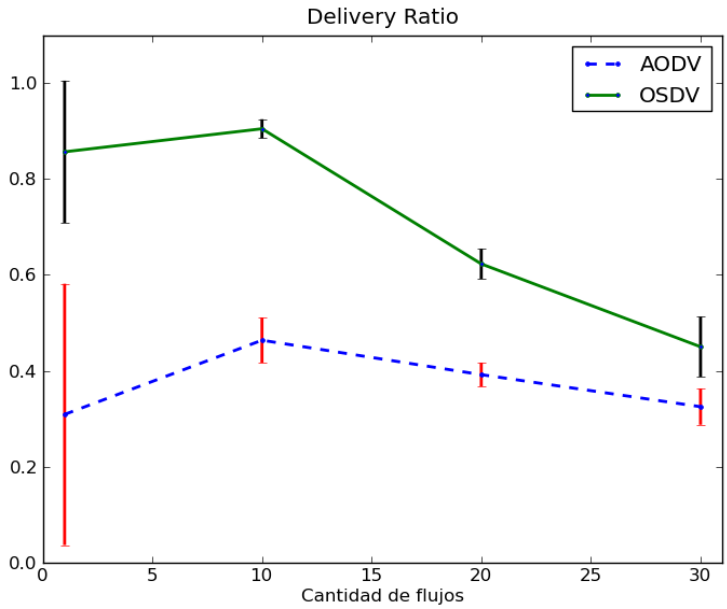


Figura 5-22. Delivery ratio para escenario 4.

Como se puede observar en la figura 5-23 el protocolo OSDV emplea más paquetes de control conforme se aumenta la cantidad de flujos debido a que se tienen que establecer más rutas hacia más destinos, así mismo, el retardo en la entrega de paquetes de datos se incrementa (figura 5-24) al seguir estableciendo rutas seguras que no son las más cortas hacia los destinos.

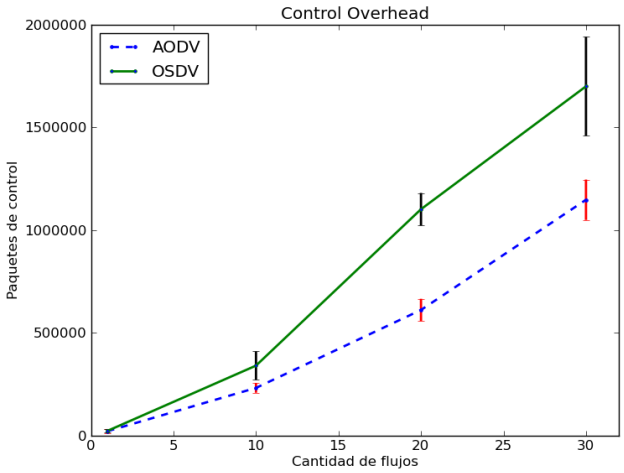


Figura 5-23. Control overhead para escenario 4.

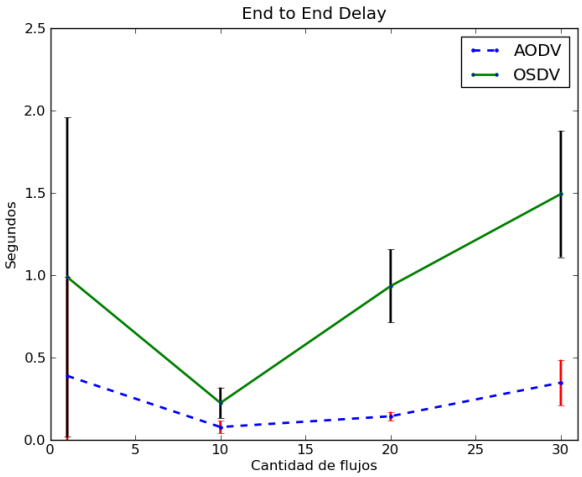


Figura 5-24. End to end delay para escenario 4.

Finalmente, en el escenario 5 se varió el área de simulación en conjunto con la cantidad de nodos que interactúan en tal área con el objetivo de mantener constante la densidad de nodos por unidad de área pero permitiendo que existan caminos cada vez más largos. Los resultados de dichos cambios los podemos apreciar en las figuras 5-25, 5-26 y 5-27. Con forme se amplía el área de la red se nota un decremento en la entrega de paquetes de datos por parte del protocolo OSDV pero siendo mejor que la del protocolo AODV.

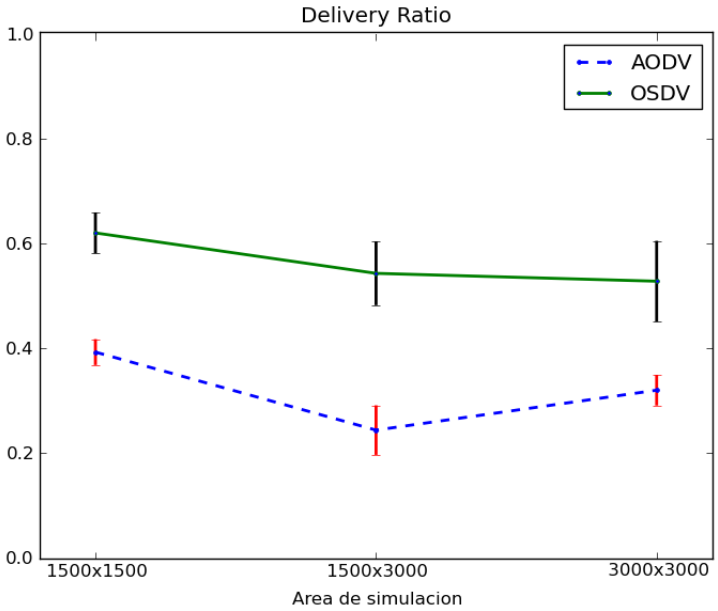


Figura 5-25. Delivery ratio para escenario 5.

Con respecto a la cantidad de paquetes de control, se sigue observando que el protocolo OSDV emplea más paquetes de control (figura 5-26) para ofrecer caminos seguros hacia los destinos y debido a ello los caminos proporcionados no siempre son los más cortos, esto se ve reflejado en el tiempo que es gastado en la entrega de los paquetes de datos en la red móvil (figura 5-27).

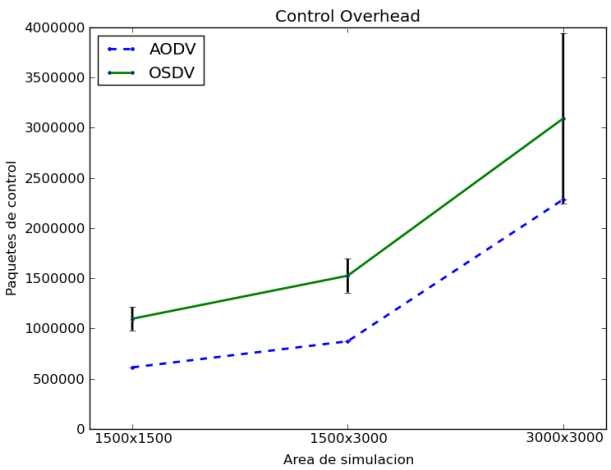


Figura 5-26. Control overhead para escenario 5.

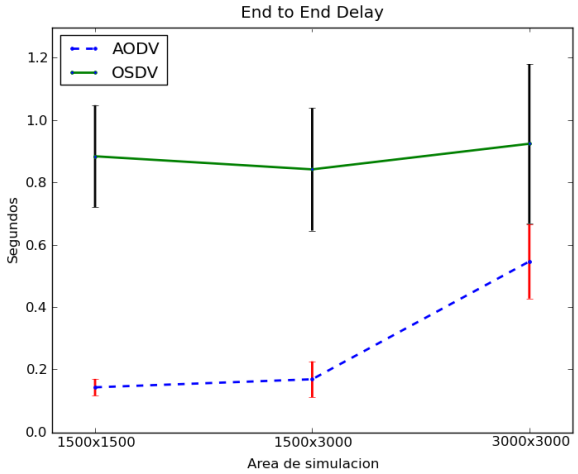


Figura 5-27. End to end delay para escenario 5.

Capítulo 6

Conclusiones y trabajo a futuro

6.1 Conclusiones

Con la creación de tecnologías inalámbricas y aplicaciones distribuidas, el uso de las redes móviles ad hoc (MANETs) han tenido un incremento considerable en su uso, ya sea de manera comercial, educativa, personal o gubernamental. Por lo tanto, primeramente fue necesario ofrecer protocolos de enrutamiento para redes móviles que ofrecieran buen desempeño en tareas de enrutamiento en dichas redes, teniendo como principales retos la movilidad, la reparación de rutas en tiempo real, ser libres de ciclos, proporcionar rutas actuales, mantenimiento de enlaces rotos, etc. Con el fin de cubrir tales necesidades se desarrollaron protocolos de enrutamiento proactivos y reactivos (ambos con ventajas y desventajas), los cuales son empleados en escenarios donde uno se desempeña mejor que el otro. Sin embargo, en los últimos años ha surgido un reto muy importante, como lo es la seguridad en MANETs. Estudios recientes han clasificado a los ataques a redes móviles en dos tipos: ataques contra el enrutamiento y basados en el consumo de recursos. Por lo que ha surgido la necesidad de implementar protocolos que además de ofrecer un buen desempeño ofrezcan mecanismos de seguridad en contra de ataques al enrutamiento como los vistos en la sección 2.2.

En una red ad hoc, desde el punto de vista de un protocolo de enrutamiento, existen dos clases de paquetes: los paquetes de enrutamiento y los paquetes de datos. Ambos tienen diferentes requerimientos de seguridad. Los paquetes de datos son de extremo a extremo y pueden ser protegidos con cualquier sistema de seguridad de extremo a extremo como IPSec[38]. Por otro lado, los paquetes de enrutamiento son enviados a vecinos inmediatos, procesados, modificados, y reenviados.

Otra consecuencia de la naturaleza de la transmisión de paquetes de enrutamiento es que, en muchos casos, existen algunas partes de esos paquetes que cambian durante su propagación. Esto se puede apreciar en el protocolo propuesto, ya que está basado en vector de distancias, en donde los paquetes de control contienen un conteo de salto de la ruta que se está estableciendo y que va cambiando conforme se propaga en la red. Entonces, en un paquete de control se distinguen dos tipos de información: mutable y no mutable. Por lo tanto, el presente trabajo de tesis se enfocó en implementar un protocolo reactivo basado en vector de distancias llamado OSDV, que ofrezca mecanismos de seguridad para asegurar tanto la información mutable como la no mutable, como lo son: cadenas hash (para asegurar información mutable) y firmas digitales(para asegurar información no mutable), ordenación en tiempo.

El objetivo principal que planteó alcanzar esta tesis, fue el desarrollar un protocolo de enrutamiento para una red móvil que sea capaz de entregar paquetes de datos a los

destinos aun y cuando existan nodos maliciosos que mientan al informar su distancia o su identidad, con el propósito de hacer mal uso de la información. Con este fin en el capítulo 4 se describe la especificación del protocolo OSDV, en el que se ve a detalle el uso de cadenas hash para asegurar la información mutable y el uso de firmas digitales para asegurar la información inmutable.

Por lo que después de desarrollar dicho protocolo, se procedió a realizar pruebas con diferentes escenarios, tales pruebas fueron llevadas a cabo en el simulador NS-2, con las que podemos concluir lo siguiente:

- Con el protocolo OSDV se incrementó la posibilidad de entregar los paquetes de datos al destino en todos los escenarios (Deliver Ratio).
- Detecta a nodos maliciosos y no permite que participen en la ruta.
- Bajo escenarios sin algún atacante el desempeño del protocolo seguro es igual al del protocolo AODV.
- Debido a que no siempre se usa la ruta más corta hacia el destino, el tiempo de entrega de los paquetes es mayor y también se envían más paquetes de control.
- Además, basándose en los resultados experimentales obtenidos en la sección 5-2, se puede afirmar que el protocolo mantiene su operación aún y cuando se variaron parámetros de simulación. Por lo tanto, el protocolo es escalable.

Finalmente podemos afirmar que con la implementación del protocolo OSDV se logró el objetivo de la tesis.

6.2 Trabajo a futuro

Debido a que existen otras maneras de atacar a los protocolos de enrutamiento reactivos, ya sea con base al consumo de recursos o al enrutamiento, como posibles trabajos futuros que se pueden llevar a cabo con base al presente trabajo desarrollado se tienen:

- Implementar mecanismos de seguridad para protegerse en contra de atacantes que intenten enviar muchos mensajes RREQ o RREP con el fin de saturar la red.
- En caso de que algún nodo malicioso envíe mensaje RERR publicando información falsa sobre destinos inalcanzables, proporcionar un mecanismo de seguridad.
- El ataque de hoyo gris consiste en un atacante que aleatoriamente elimina paquetes de datos o control, por lo tanto es necesario proporcionar un mecanismo de seguridad para este tipo de ataque.
- Utilizar regiones de interés, con el propósito de aumentar la escalabilidad del protocolo. Las regiones de interés consisten en dividir la red móvil en secciones en las que el enrutamiento se hace primero en cada sección y en caso de que el nodo destino buscado no se encuentre en la primer región, se procederá con la búsqueda en otra región de interés.

REFERENCIAS

[1] S. Marti et al., “Mitigating Routing Misbehaviour in Mobile Ad Hoc Networks,” Proc. 6th Ann. Int’l Conf. Mobile Computing and Networking (MobiCom 2000), ACM Press, 2000, pp. 255–265.

[2] Y.-C. Hu, A. Perrig, and D.B. Johnson, “Rushing Attacks and Defense in Wireless Ad Hoc Network Routing Protocols,” Proc. 2003 ACM Workshop on Wireless Security (WiSe 2003), ACM Press, 2003, pp. 30–40.

[3] Dynamic MANET On-demand Routing Protocol (DYMO), Ad hoc On-demand Distance Vector (AODV) Routing, Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks, etc.

[4] D.B. Johnson, “Routing in Ad Hoc Networks of Mobile Hosts,” Proc. IEEE Workshop on Mobile Computing Systems and Applications (WMCSA’94), IEEE Press, 1994, pp. 158–163.

[5] C.E. Perkins and E.M. Royer, “Ad-Hoc On-Demand Distance Vector Routing,” Proc. 2nd IEEE Workshop Mobile Computing Systems and Applications (WMCSA’99), IEEE Press, 1999, pp. 90–100.

[6] Qayyum, L. Viennot, and A. Laouiti, “Multi-Point Relaying: An Efficient Technique for Flooding in Mobile Wireless Networks,” tech. report RR-3898, INRIA, Feb. 2000.

[7] Bellur and R.G. Ogier. “A Reliable, Efficient Topology Broadcast Protocol for Dynamic Networks,” Proc. 18th Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM ’99), IEEE Press, 1999, pp. 178–186.

[8] Y.-C. Hu, A. Perrig, and D.B. Johnson, “Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks,” Proc. 22nd Ann. Joint Conf. IEEE Computer and Communications Societies (INFOCOM 2003), IEEE Press, 2003, pp. 1976–1986.

[9] Perrig et al., “Efficient Authentication and Signing of Multicast Streams over Lossy Channels,” *Proc. IEEE Symp. Security and Privac*y, IEEE Press, 2000, pp. 56–73.

[10] R.C. Merkle, “Protocols for Public Key Cryptosystems,” *Proc. IEEE Symp. Research in Security and Privac*y, IEEE Press, 1980, pp. 122–133.

[11] F. Stajano and R. Anderson. “The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks,” *Proc. 7th Int’l Workshop on Security Protocol*s, Lecture Notes in Computer Science 1796, Springer-Verlag, 1999, pp. 172–194.

[12] G. Montenegro and C. Castelluccia. “Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses,” *Proc. Symp. Network and Distributed Systems Security* (NDSS 2002), Internet Society, 2002, pp. 87–99.

[13] L. Zhou and Z.J. Haas, “Securing Ad Hoc Networks,” *IEEE Network Magazin*e, IEEE Press, vol. 13, no. 6, 1999, pp. 24–30.

[14] S. Yi and R. Kravets, *Key Management for Heterogeneous Ad Hoc Wireless Networks*, tech. report UIUCDCS-R-2002-2290, Dept. Computer Science, Univ. of Illinois at Urbana-Champaign, July 2002.

[15] J.-P. Hubaux, L. Buttyán, and S. Cˇ apkun, “The Quest for Security in Mobile Ad Hoc Networks,” *Proc. 2nd Symp. Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), ACM Press, 2001, pp. 146–155.

[16] Y.-C. Hu, D.B. Johnson, and A. Perrig, “SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad Hoc Networks,” *Proc. 4rth IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA 02), IEEE Press, 2002, pp. 3–13.

[17] C.E. Perkins and P. Bhagwat, “Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers,” *Proc. SIGCOMM ’94 Conf. Communications Architectures, Protocols and Application*s, ACM Press, 1994, pp. 234–244.

[18] Y.-C. Hu, A. Perrig, and D.B. Johnson, “Ariadne: A Secure On-Demand Routing Protocol for Ad Hoc Networks,” *Proc. 8th Ann. Int’l Conf. Mobile Computing and Networking* (MobiCom 2002), ACM Press, 2002, pp. 12–23.

[19] A. Perrig et al., “Efficient Authentication and Signing of Multicast Streams over Lossy Channels,” *Proc. IEEE Symp. Security and Privac*y, IEEE Press, 2000, pp. 56–73.

[20] M. Mathis et al., *TCP Selective Acknowledgment Options*,” RFC 2018, Oct. 1996.

[21] T. Narten, E. Nordmark, and W. Allen Simpson, “*Neighbor Discovery for IP Version 6* (IPv6),” RFC 2461, Dec. 1998.

[22] P. Papadimitratos and Z.J. Haas, “Secure Routing for Mobile Ad Hoc Networks,” *Proc. SCS Communication Networks and Distributed Systems Modeling and Simulation Conf.* (CNDS 2002), Jan. 2002.

[23] Z.J. Haas and M.R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," *Proc. ACM SIGCOMM 98 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM Press, 1998, pp. 167–177.

[24] P. Papadimitratos and Z.J. Haas, "Secure Message Transmission in Mobile Ad Hoc Networks," *Elsevier Ad Hoc Networks J.*, Elsevier, vol. 1, no. 1, 2003, pp. 193–209.

[25] K. Sanzgiri et al., "A Secure Routing Protocol for Ad Hoc Networks," Proc. 10th IEEE Int'l Conf. Network Protocols (ICNP '02), IEEE Press, 2002, pp. 78–87.

[26] M. Guerrero Zapata and N. Asokan, "Securing Ad Hoc Routing Protocols," *Proc. ACM Workshop on Wireless Security* (WiSe), ACM Press, 2002, pp. 1–10.

[27] Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling by Raj Jain
Wiley Computer Publishing, John Wiley & Sons, Inc. ISBN: 0471503363.

[28] The Network Simulator - ns-2.
http://www.isi.edu/nsnam/ns/ (Consultado en Julio de 2011)

[29] QNET4 y RESQ
http://domino.watson.ibm.com/comm/research.nsf/pages/r.performance.history.html (Consultado en Julio de 2011)

[30] Continuous System Modeling Program CSMP
http://hopl.murdoch.edu.au/showlanguage.prx?exp=714 (Consultado en Julio de 2011)

[31] DYNAmic MOdels simulation
http://hopl.murdoch.edu.au/showlanguage.prx?exp=61&language=DYNAMO (Consultado en Julio de 2011)

[32] Lenguaje de programación orientada a objetos SIMULA
http://staff.um.edu.mt/jskl1/talk.html (Consultado en Julio de 2011)

[33] General Purpose Simulation System (GPSS)
http://www.webgpss.com/ (Consultado en Julio de 2011)

[34] General-purpose simulation language SIMSCRIPT
http://www.simscript.net/products/products.html (Consultado en Julio de 2011)

[35] General Activity Simulation Program GASP

http://hopl.murdoch.edu.au/showlanguage.prx?exp=408 (Consultado en Julio de 2011)

[36] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.1: The Bellman-Ford algorithm, pp.588–592.

[37] Manner, J., et al., "Mobility Related Terminology", June 2004, Network Working Group, http://www.faqs.org/rfcs/rfc3753.html (consultado en Julio de 2011)

[38] IPsec Internet Protocol security
http://tools.ietf.org/html/rfc2367 (Consultado en Agosto de 2011)

[39] Stephen Dabideen et al “Secure Routing in MANETs without Verifiable Distances or Link States” University of California, Santa Cruz

[40] S. W. Sherman and J. C. Browne "Trace driven modeling: Review and overview", ANSS '73 Proceedings of the 1st symposium on Simulation of computer

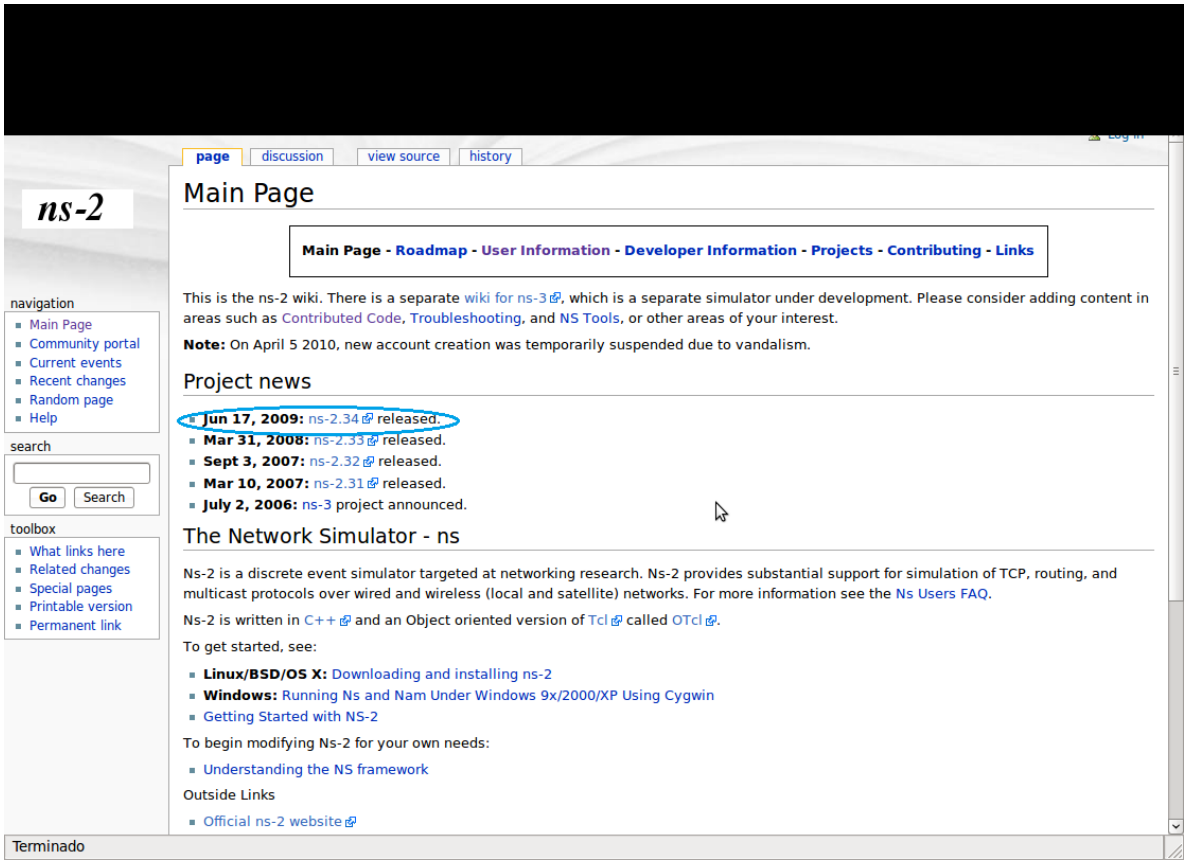
[41] OTcl Project Page http://otcl-tclcl.sourceforge.net/otcl/ (consultado en Agosto de 2011)

APÉNDICE A: Instalación de NS-2.34

Características de la instalación:

Sistema operativo: Ubuntu Karmic 9.10
Versión de GCC: 4.1
Versión del simulador: 2.34.

Se descargó ns-allinone-2.34 de la página oficial de NS-2:
http://nsnam.isi.edu/nsnam/index.php/Main_Page



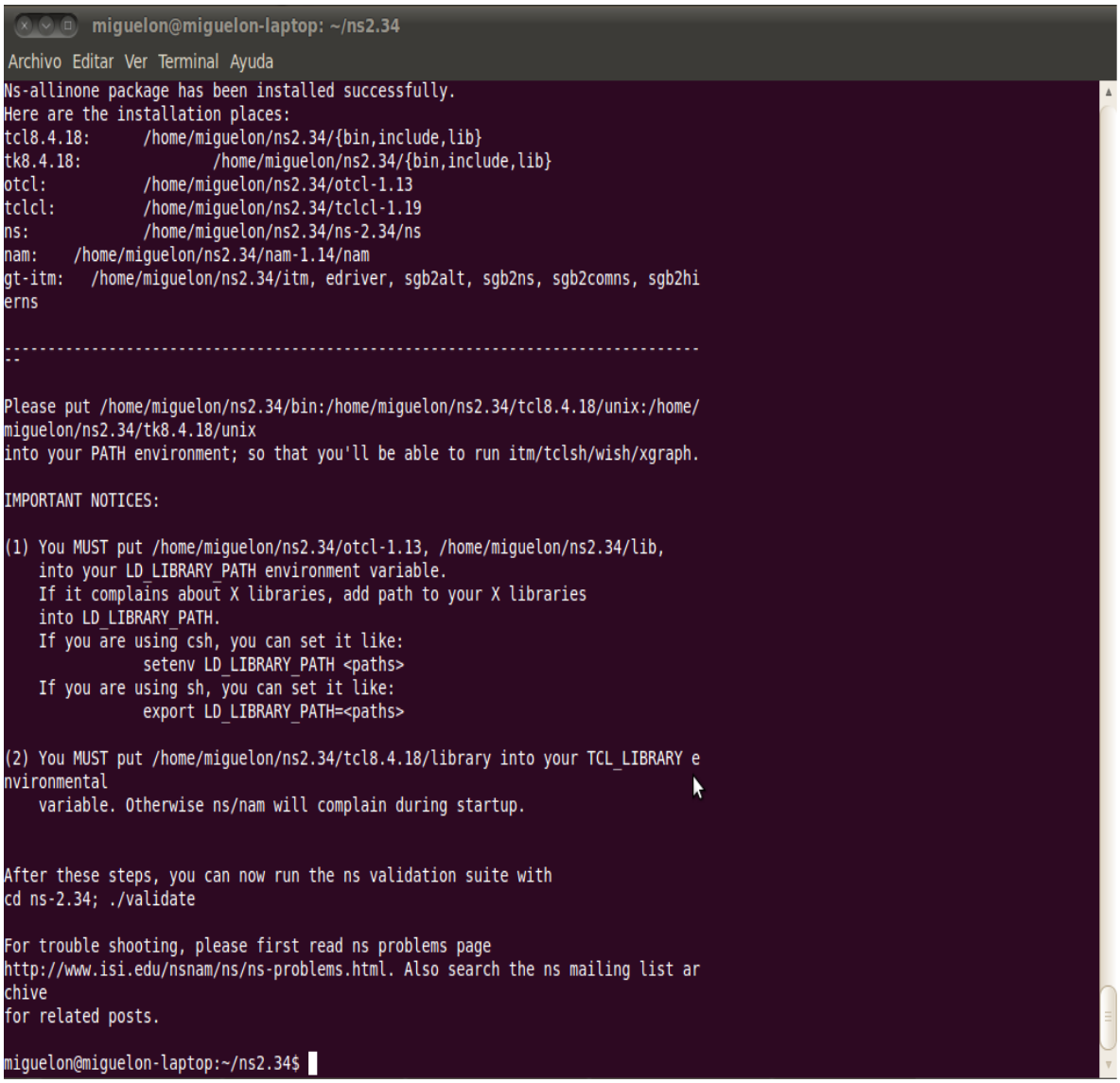
Para instalar ns-allinone-2.34 se debe corregir el error en el enlace del OTcl que se muestra a continuación:

```
otcl.o: In function `OTclDispatch':
/home/ns/ns-allinone-2.34/otcl/otcl.c:495: undefined reference to `__stack_chk_fail_local'
otcl.o: In function `Otcl_Init':
/home/ns/ns-allinone-2.34/otcl/otcl.c:2284: undefined reference to `__stack_chk_fail_local'
ld: libotcl.so: hidden symbol `__stack_chk_fail_local' isn't defined
ld: final link failed: Nonrepresentable section on output
make: *** [libotcl.so] Error 1
```

Este error es debido a que el enlace usado es "`ld -shared`" en lugar de "`gcc -shared`". Para corregirlo tenemos que editar una línea en el archivo otcl-1.13/configure.orig, y volver a correr el instalador, la parte del código se muestra a continuación:

```
--- configure.orig      2009-11-02 12:14:52.556167945 -0800
+++ configure   2009-11-02 12:17:28.966706099 -0800
@@ -6301,7 +6301,7 @@
          ;;
      Linux*)
          SHLIB_CFLAGS="-fpic"
-         SHLIB_LD="ld -shared"
+         SHLIB_LD="gcc -shared"
          SHLIB_SUFFIX=".so"
          DL_LIBS="-ldl"
          SHLD_FLAGS=""
```

Al final de la instalación se debe obtener el siguiente resultado:

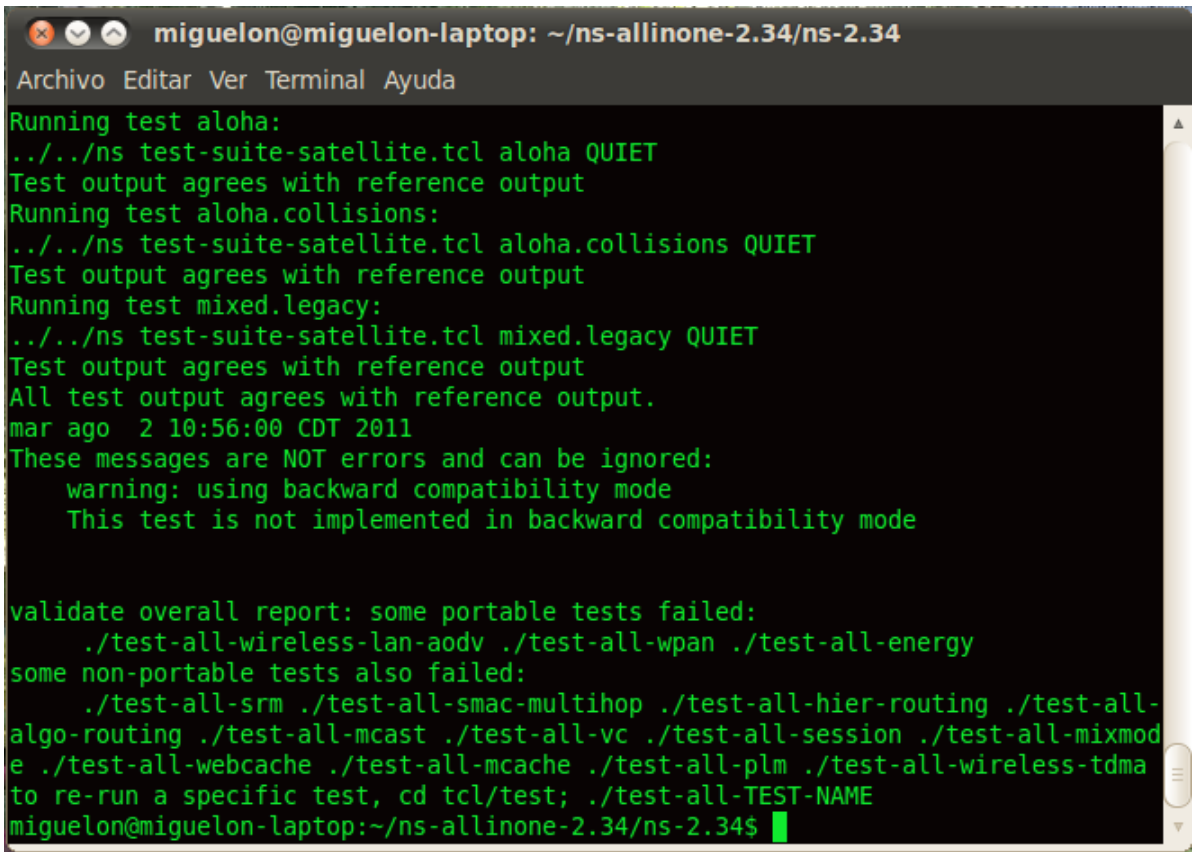


Y para tener funcionando todas las características del simulador solo bastará con modificar las variables de entorno que se indican en el resultado.

Se puede validar la instalación con los siguientes comandos:

```
cd ns-2.34
./validate
```

Al ejecutar el comando anterior se obtuvo el resultado mostrado en la figura siguiente:



APÉNDICE B: Adición de un nuevo protocolo a NS-2

Una vez que se tiene el protocolo desarrollado es necesario colocar la carpeta que contiene los archivos de dicho protocolo en el directorio base de NS-2, una vez hecho esto se pueden seguir los siguientes pasos para integrar dicho protocolo al simulador:

1. En el protocolo desarrollado se tuvo que haber declarado una constante el cual indicará el nuevo tipo de paquete, éste se definirá dentro del archivo *common/packet.h*.
   En el archivo se debe encontrar la enumeración *packet_t*, en donde se enlistan todos los tipos de paquetes, entonces agregar PT_OSDV (nombre del nuevo protocolo) a la lista como se muestra a continuación:

```
1: enum packet_t {
2: PT_TCP,
3: PT_UDP,
4: PT_CBR,
5: /* ... más tipos de paquetes ... */
6: PT_OSDV,
7: PT_NTYPE // Este tipo de paquete debe ser el último
8: };
```

   En el mismo archivo (*packet.h*) dentro del constructor de la clase *p_info* se debe proporcionar un nombre para el nuevo tipo de paquete (línea 6).

```
1: p_info() {
2: name_[PT_TCP]= "tcp";
3: name_[PT_UDP]= "udp";
4: name_[PT_CBR]= "cbr";
5: /* ... más nombres ... */
6: name_[PT_OSDV]= "osdv";
7: }
```

2. Para registrar información referente al nuevo tipo de paquete se debe implementar la función *format_osdv()* dentro de la clase **CMUTrace**.
   En este paso se tiene que editar el archivo *trace/cmu-trace.h*. Se debe agregar la nueva función como se muestra en la línea 6 del ejemplo.

```
1: class CMUTrace : public Trace {
2: /* ... definiciones ... */
3: private:
```

```
4: /* ... */
5: void format_aodv(Packet *p, int offset);
6: void format_osdv(Packet *p, int offset);
7: };
```

Ahora agregar la implementación de la función en *trace/cmu-trace.cc*

```
1: #include <osdv/osdv_pkt.h> //incluir el nuevo
2: //protocolo
3: /* ... */
4:
5: void
6: CMUTrace::format_osdv(Packet *p, int offset)
7: {
8: struct hdr_osdv_pkt* ph = HDR_OSDV_PKT(p);
9:
10: if (pt_->tagged()) {
11: sprintf(pt_->buffer() + offset,
12: "-osdv:o %d -osdv:s %d -osdv:l %d ",
13: ph->pkt_src(),
14: ph->pkt_seq_num(),
15: ph->pkt_len());
16: }
17: else if (newtrace_) {
18: sprintf(pt_->buffer() + offset,
19: "-P osdv -Po %d -Ps %d -Pl %d ",
20: ph->pkt_src(),
21: ph->pkt_seq_num(),
22: ph->pkt_len());
23: }
24: else {
25: sprintf(pt_->buffer() + offset,
26: "[osdv %d %d %d] ",
27: ph->pkt_src(),
28: ph->pkt_seq_num(),
29: ph->pkt_len());
30: }
31: }
```

Con el propósito de tener acceso a la función recién creada se debe cambiar el *format()* en *trace/cmu-trace.cc*.

```
1: void
2: CMUTrace::format(Packet* p, const char *why)
```

```
3: {
4: /* ... */
5: case PT_PING:
6: break;
7:
8: case PT_OSDV:
9: format_osdv(p, offset);
10: break;
11:
12: default:
13: /* ... */
14: }
```

3. En este paso se necesita hacer cambios en los archivos Tcl. Se agregará el nuevo tipo de paquete, dar valores por omisión para atributos enlazados y proveer la infraestructura necesaria para crear nodos desde los scripts de simulación del nuevo protocolo de enrutamiento.
   En *tcl/lib/ns-packet.tcl* localizar el siguiente código y agregar el nuevo protocolo a la lista (línea 2).

```
1: foreach prot {
2: OSDV
3: AODV
4: ARP
5: # ...
6: NV
7: } {
8: add-packet-header $prot
9: }
```

   Se deben dar valores por omisión en el archivo *tcl/lib/ns-default.tcl*, como se muestra a continuación:

```
1: # ...
2: # Valores por omisión para el nuevo protocolo
3: Agent/osdv set accessible_var_ true
```

   Finalmente se debe modificar *tcl/lib/ns-lib.tcl* para agregar procedimiento con el fin de poder crear un nodo.

```
1: Simulator instproc create-wireless-node args {
2: # ...
3: switch -exact $routingAgent_ {
```

```
4: osdv {
5: set ragent [$self create-osdv-agent $node]
6: }
7: # ...
8: }
9: # ...
10: }
```

Después, dentro del mismo archivo, programar el procedimiento *create-osdv-agent* como se muestra a continuación:

```
1: Simulator instproc create-osdv-agent { node } {
2: # Crear agente de enrutamiento para el Nuevo protocolo
3: set ragent [new Agent/osdv [$node node-addr]]
4: $self at 0.0 "$ragent start"
5: $node set ragent_ $ragent
6: return $ragent
7: }
```

4. Para este paso se definirá el nuevo paquete como un paquete de mayor prioridad debido a que es un paquete de enrutamiento, esto se hace en la clase **PriQueue** la cual es la encargada de manejar las colas de prioridades.
   Se debe modificar la funciona *recv()* en el archivo *queue/priqueue.cc*. Sólo se modifica la línea 13 en el siguiente código:

```
1: void
2: PriQueue::recv(Packet *p, Handler *h)
3: {
4: struct hdr_cmn *ch = HDR_CMN(p);
5:
6: if (Prefer_Routing_Protocols) {
7:
8: switch(ch->ptype()) {
9: case PT_DSR:
10: case PT_MESSAGE:
11: case PT_TORA:
12: case PT_AODV:
13: case PT_OSDV:
14: recvHighPriority(p, h);
15: break;
16:
17: default:
18: Queue::recv(p, h);
19: }
```

```
20: }
21: else {
22: Queue::recv(p, h);
23: }
24: }
```

5. Una vez hechas las modificaciones de los pasos anteriores sólo falta compilar el nuevo protocolo. Para hacerlo se debe editar el archivo *Makefile* para agregar los archivos objeto del nuevo protocolo como se muestra a continuación.

```
1: OBJ_CC = \
2: tools/random.o tools/rng.o tools/ranvar.o common/misc.o \
3: common/timer-handler.o \
4: # ...
5: osdv/osdv.o \
6: osdv/osdv_rtable.o \
7: # ...
8: $(OBJ_STL)
```

Como no se editó el archivo *common/packet.cc* se debe tocar para tener una fecha de modificación actual para ser tomado en cuenta en la re compilación.

```
[ns-2.34]$ touch common/packet.cc
[ns-2.34]$ make
```