
XV Jornadas de Ingeniería Telemática.
JITEL 2021.
Universidad de A Coruña.

Actas de las XV Jornadas de Ingeniería Telemática (JITEL 2021), A Coruña (España), 27-29 de octubre de 2021.

ISBN

Implementación en el Espacio de Usuario del Protocolo de Encaminamiento AODVv2

Sergio Machado, Israel Martín-Escalona, Enrica Zola, Francisco Barceló-Arroyo, Javier Ozón
Departamento de Ingeniería Telemática,
Universidad Politècnica de Catalunya, UPC BarcelonaTECH
Barcelona, Spain.
{sergio.machado, israel.martin, enrica.zola, francisco.barcelo, francisco.javier.ozon}@upc.edu

El protocolo AODV es un protocolo reactivo de encaminamiento para redes MANET que se utiliza frecuentemente como referencia tanto para desarrollar nuevos protocolos de encaminamiento en redes ad hoc como para evaluar su rendimiento. Aunque el protocolo está presente en varios simuladores de red (por ejemplo, ns2, OMNeT++, etc.), existen pocas implementaciones que puedan emplearse en condiciones reales para la investigación de campo o de evaluación. Este documento presenta una implementación en el espacio de usuario de la última versión del protocolo, el AODVv2, que se puede utilizar en cualquier dispositivo capaz de ejecutar Linux. El objetivo del proyecto ha sido el desarrollo de una implementación de AODVv2 de código abierto y fácil mantenimiento, para ser utilizada con fines experimentales por la comunidad científica. El presente documento proporciona una descripción de los principales criterios de diseño y codificación considerados para implementar el protocolo, y expone además las principales pruebas que se han realizado con el fin de verificar su correcto funcionamiento.

***Palabras Clave*—AODVv2, MANET, ad hoc, protocolos de encaminamiento, implementación, Linux, espacio de usuario.**

I. INTRODUCCIÓN

Las Mobile Ad Hoc Networks (MANETs) son redes inalámbricas carentes de infraestructura en las que los nodos móviles se comunican entre sí de forma descentralizada. En una red MANET los nodos desempeñan distintos papeles, puesto que pueden ser tanto emisores como receptores de un determinado flujo de información, así como intermediarios en la comunicación entre otros pares de nodos. Un protocolo de encaminamiento debe determinar en cada momento qué nodos se ocupan de la retransimisión de un determinado flujo de información, i.e. de reenviar una trama al siguiente nodo de la ruta que une el emisor con el receptor.

This work was supported by the Spanish Government and ERDF through CICYT project PGC2018-099945-B-I00.

Los protocolos de encaminamiento de las MANETs pueden clasificarse conforme distintos criterios. Una de las clasificaciones más comunes divide los protocolos en encaminamiento no jerárquico y encaminamiento jerárquico. En el primer caso, las reglas de encaminamiento se aplican uniformemente a todos los nodos de la red. En el segundo, los nodos de la red se dividen en distintos grupos y los protocolos aplican distintos criterios conforme el encaminamiento comunica nodos del mismo grupo o bien nodos de grupos distintos. A su vez, los protocolos de encaminamiento no jerárquico se clasifican en tres categorías: proactivos, reactivos e híbridos. Los protocolos proactivos monitorizan regularmente la topología de red y aplican soluciones clásicas de encaminamiento en redes fijas, tanto de tipo vector distancia como de estado del enlace. Mientras los protocolos proactivos disponen en cada momento de un encaminamiento posible entre cualquier par de nodos, los protocolos reactivos funcionan bajo demanda, es decir, activan un proceso de encaminamiento cada vez que un paquete se ha de enviar desde un nodo fuente a un nodo destino. El encaminamiento conserva su validez durante un determinado periodo de tiempo y se emplea hasta que expira dicho plazo o el protocolo comprueba que ha dejado de funcionar. En este caso, el protocolo determinará un nuevo encaminamiento entre el nodo emisor y el receptor. Dado que ambos mecanismos presentan ventajas y desventajas, los protocolos híbridos tratan de combinar los beneficios de ambas aproximaciones.

Con independencia de la estrategia seguida, i.e. jerárquica o no jerárquica, los protocolos de encaminamiento para MANETs pueden clasificarse a su vez conforme dispongan o no de la posibilidad de obtener y, por tanto, aprovechar la localización de los nodos en la estrategia de encaminamiento. Esta política de localización de los nodos se ha demostrado además imprescindible para un correcto escalado de dichos protocolos [3].

Ad hoc On Demand Distance Vector Routing (AODV) [4] es un protocolo de encaminamiento MANET

que establece y mantiene rutas entre nodos mediante mensajes de petición de ruta (route request) y mensajes de respuesta de ruta (route reply). Cuando se ha de establecer una ruta entre dos nodos, el nodo emisor inunda los nodos vecinos con un mensaje de petición de ruta; cada uno de los nodos vecinos retransmite, a su vez, el mensaje de petición a sus vecinos y así sucesivamente hasta alcanzar el nodo destino. En este punto, el nodo destino responde con un mensaje de respuesta de ruta que recorre en sentido inverso el camino previamente trazado por el mensaje de petición de ruta.

Aunque el protocolo AODV version 2 (AODVv2) [5] introduce algunas mejoras sobre AODV, el mecanismo de encaminamiento es fundamentalmente el mismo en ambos casos. Algunas de las mejoras más relevantes contempladas por AODVv2 son: inundación optimizada de los mensajes de petición de ruta con el fin de reducir la sobrecarga (overhead) debida al encaminamiento; ampliación del número de métricas que pueden considerarse; empleo de una estructura Type-Length-Value (TLV) para la codificación de los campos de datos, que añade flexibilidad a la información que ha de enviarse, tal y como define el RFC Generalized MANET Packet/Message Format [6]; secuenciación de los mensajes Route Request (RREQ) y Route Reply (RREP); mecanismo que evita que los nodos intermedios respondan a los mensajes RREQ cuando conocen una ruta al destino solicitado; y uso de mensajes Route Error (RERR) para avisar de la caída de un enlace así como de mensajes Route Reply Acknowledgment (RREP_ACK) para comprobar el funcionamiento de los enlaces bidireccionales de la red.

AODVv2 es un protocolo que se toma con frecuencia como referencia en la evaluación del rendimiento de nuevos protocolos de encaminamiento. Además, los investigadores suelen tomar AODVv2 como punto de partida para desarrollar sus propios protocolos [7]. Actualmente se encuentran disponibles en la literatura varias implementaciones del algoritmo; sin embargo, la mayoría de dichas implementaciones están concebidas únicamente con fines de simulación [8] [9], han quedado obsoletas [10] [11] o bien se restringen a ciertas configuraciones antiguas de software. En [12], los autores presentan una implementación de AODV en Python, mientras que en [13] se implementa AODV mediante Exata Emulator y Matlab. Con respecto a las implementaciones de AODVv2, los autores de [9] detallan la implementación del protocolo en el programa de simulación OMNeT++ dentro del marco INET. Sin embargo, dado que el protocolo AODVv2 ha evolucionado desde 2008, se ha hecho necesaria una actualización para garantizar la fiabilidad de futuras pruebas. Más recientemente, los autores en [14] presentan una adaptación de la implementación de AODV-UU [10] a las versiones más recientes del kernel. En el presente trabajo, hemos adoptado el enfoque de [14], que usa una implementación del protocolo en el espacio de usuario para ganar independencia del kernel. Nuestro objetivo es desarrollar una nueva implementación, de código abierto y fácil de mantener, del protocolo AODVv2 dirigido a todos los investigadores que trabajan en protocolos de encaminamiento MANET.

El objetivo de este artículo es proporcionar una descripción completa de esta nueva implementación del protocolo AODVv2. Este trabajo está dividido de la siguiente manera. En la Sección II se describen los mensajes y el funcionamiento del protocolo AODVv2. La sección III presenta los criterios de diseño de nuestra implementación de AODVv2 en el espacio de usuario de Linux. La sección IV describe las pruebas y los casos que se han planteado con el fin de comprobar el correcto funcionamiento de la implementación. Finalmente, en la Sección V se exponen las principales conclusiones del proyecto y se plantean posibles mejoras futuras.

II. DESCRIPCIÓN GENERAL DEL PROTOCOLO

El protocolo AODVv2 se ocupa de la gestión dinámica de una red inalámbrica ad hoc con el objeto de garantizar, cuando es físicamente posible, la disponibilidad de una ruta entre cualquier par de dispositivos conectados a la red. AODVv2 sustituye al protocolo AODV e incluye algunas mejoras, como una mayor fiabilidad de las pruebas de los enlaces inalámbricos, la transmisión y procesado de respuestas a las peticiones de ruta y un mayor rango de búsqueda de rutas. Como su predecesor, AODVv2 es un protocolo de encaminamiento de vector distancia, en el que los datos de ruta incluyen el siguiente nodo al que se debe reenviar un paquete y el coste (métrica) de dicho salto. De forma predeterminada, la métrica considera el número de saltos de la ruta. De este modo, cuando existe más de un camino entre el emisor y el receptor, el protocolo escoge aquel que contiene un menor número de enlaces.

En Fig. 1 y Fig. 2 se describe la operativa básica del protocolo. La Fig. 1 muestra un escenario donde el nodo emisor *S* pretende enviar un mensaje al nodo destino *D*, que no está conectado directamente a *S*. En la figura, las líneas discontinuas representan transmisiones broadcast. AODVv2 es un protocolo reactivo. De este modo, cuando un nodo emisor no dispone de una ruta al nodo destino, envía un mensaje broadcast RREQ con el objeto de descubrir una ruta que comunique el origen con el destino. El mensaje RREQ será recibido, procesado y reenviado por los nodos intermedios (llamados también nodos retransmisores) a lo largo del camino entre el nodo emisor y el receptor. En Fig. 1 el mensaje broadcast RREQ enviado por *S* es recibido por los nodos *A* y *E*. Dado que ninguno de los dos es el nodo destino del mensaje RREQ, *A* y *E* volverán a enviar el mensaje RREQ a sus nodos vecinos. Si un nodo recibe el mismo RREQ en más de una ocasión, solo lo reenviará la primera vez y lo descartará en las ocasiones sucesivas. El mensaje RREQ incluye un número de secuencia que permite identificar los mensajes RREQ duplicados.

En Fig. 1, el nodo *A* descarta el mensaje RREQ reenviado por el nodo *E*, puesto que se habrá identificado como mensaje duplicado gracias al número de secuencia. Cuando un nodo recibe un mensaje RREQ útil, i.e. que no debe ser descartado, el nodo almacena la ruta inversa que

lo comunica con el nodo que ha emitido inicialmente el RREQ, en este caso el nodo *S*. En Fig. 1, el nodo *A* sabrá que el nodo *S* es uno de sus nodos adyacentes, si es que no lo sabía previamente, y que por tanto puede alcanzar dicho nodo mediante un único salto. De modo análogo, el nodo *B* reconocerá que puede llegar al nodo *S* a través del nodo *A*. Asimismo, el nodo *C* descubrirá que puede llegar a *S* a través del nodo *B*. Finalmente, el mensaje RREQ llegará al nodo destino *D*, que sabrá que puede comunicarse con el nodo *S* a través del nodo *C*. Todos los nodos implicados almacenan localmente la correspondiente información en sus respectivas tablas de encaminamiento.

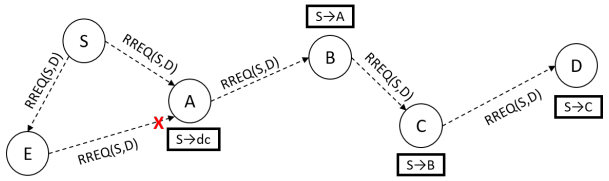

Fig. 1. Ejemplo de inundación RREQ.

Fig. 2 muestra el mensaje RREP enviado por el nodo destino con el fin de completar el proceso de creación de la ruta. Tras recibir el mensaje RREQ creado por el nodo *S*, el nodo *D* responde con un mensaje RREP. A diferencia de lo que sucede con RREQ, los mensajes RREP no se envían en modo broadcast sino en modo unicast, representado en la figura mediante flechas continuas. Esto es así puesto que los nodos intermedios han almacenado localmente el camino de vuelta al nodo emisor conforme el mensaje RREQ recorría el camino de *S* a *D*. De este modo, el nodo receptor *D* enviará el mensaje RREP únicamente al nodo *C*, el cual, a su vez, lo reenviará únicamente al nodo *B* y así sucesivamente hasta que el mensaje RREP llegue al nodo emisor *S* que comenzó todo el proceso. Finalmente, una vez el nodo *S* haya recibido el mensaje RREP, se considerará que la ruta ha sido correctamente establecida y los paquetes almacenados en la cola del emisor *S* empezarán a ser enviados a su destino *D*.

Complementariamente, AODVv2 permite comprobar el funcionamiento de los enlaces cuando crea una ruta. Para ello, cuando un nodo envía o retransmite un mensaje RREP, debe enviar también un mensaje RREP_ACK al siguiente nodo, quien a su vez debe responderle con un mensaje RREP_ACK. Este mecanismo, que debe omitirse cuando se alcanza el límite del ancho de banda del enlace reservado para mensajes de control, aparece en Fig. 2 en el enlace que une los nodos *D* y *C* y se obvia en el resto por simplicidad.

AODVv2 emplea asimismo mensajes RERR para informar sobre la caída de enlaces en la red. Existen tres casos en los que el protocolo transmite un mensaje RERR. En primer lugar, cuando un nodo recibe un paquete que ha de retransmitir pero no dispone de una ruta válida a la dirección de destino. En este caso, el nodo envía un mensaje unicast RERR a la dirección origen del paquete. En segundo lugar, cuando un nodo recibe un mensaje RREP pero no dispone de una ruta al nodo que creó

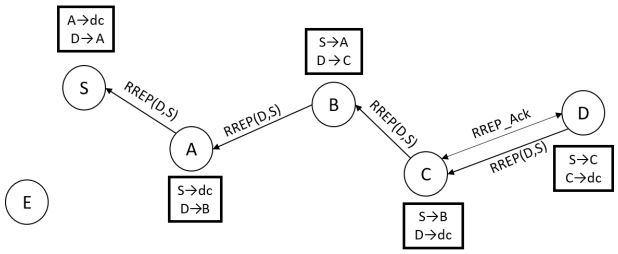

Fig. 2. Ejemplo de RREP.

el correspondiente mensaje RREQ, i.e. una ruta al nodo emisor. En este caso, el nodo envía un mensaje unicast RERR al nodo que creó el mensaje RREP, i.e. al nodo destino de la comunicación. Nótese que tanto el paquete de información en el primer caso como el mensaje RREP en el segundo, son descartados por el nodo intermedio que crea el mensaje RERR. El tercer supuesto en que se transmite un mensaje RERR ocurre cuando un nodo detecta la caída de un enlace que puede desactivar distintas rutas entre nodos. En este caso, el nodo envía un mensaje multicast RERR a todos sus nodos adyacentes.

Con el fin de evitar congestiones, el tráfico de control de AODVv2 solo puede consumir el 10% de la capacidad de cada enlace. Cuando supera este umbral, el protocolo prioriza la detección y notificación de la caída de enlaces y también de la presencia de rutas incorrectas, con el objeto de encontrar rápidamente nuevas rutas disponibles para la transmisión de datos.

III. IMPLEMENTACIÓN DEL PROTOCOLO AODVV2

Esta sección presenta las decisiones de diseño más relevantes en la implementación del protocolo AODVv2. Dicha implementación ha sido concebida para ser ejecutada en dispositivos con el sistema operativo Linux. Hay dos aproximaciones para implementar un protocolo de encaminamiento para Linux según el tipo de espacio de memoria donde se va a ejecutar la implementación: el espacio del usuario y el espacio del kernel. En el espacio de usuario el código tiene un acceso limitado a los recursos del sistema y siempre a través de una Application Programming Interface (API) del kernel que le proporciona acceso tanto a memoria como a hardware. El código que se ejecuta en el espacio del kernel el acceso tanto a memoria como a hardware cerece de restricciones. El espacio del kernel está reservado para las funciones de más bajo nivel y confiables dado que tiene un acceso ilimitado a los recursos del sistema. Si bien una implementación en el espacio de usuario tiene un menor rendimiento que una en el espacio del kernel, se ha elegido la primera puesto que simplifica el diseño, el desarrollo, las pruebas y la portabilidad del código fuente.

El acceso a los paquetes que llegan a una interfaz del dispositivo en el espacio de usuario se hace a través de Netfilter [16], que proporciona un conjunto de puntos de enganche dentro del kernel que permiten registrar funciones *callback*. Estas funciones *callback* serán requeridas

cuando un paquete llegue al punto donde la función ha sido registrada. Hay cinco puntos de enganche: PREROUTING, por donde pasan todos los paquetes que entran en la pila de red de Linux; INPUT, por donde pasan todos los paquetes con dirección IP destino igual a cualquiera de las configuradas en el dispositivo; FORWARD, por donde pasan todos los paquetes que se reenvían a otro dispositivo; OUTPUT, por donde pasan todos los paquetes localmente generados; y finalmente, POSTROUTING, por donde pasan todos los paquetes salientes, tanto si han sido generados localmente, como si son paquetes para reenviar. La configuración de Netfilter se realiza utilizando la herramienta `iptables` o la más reciente `nftables`.

El procesado de paquetes dentro de Netfilter se realiza a través de reglas. Una regla incluye una condición de coincidencia que define a qué paquetes se le aplicará la regla y una acción que define qué se debe hacer con un paquete que cumple la condición de coincidencia. La mayor parte de las acciones se pueden reducir a dos: continuar con el procesado del paquete o bien descartarlo. Cuando a un paquete que cumple la condición se le aplica una de las dos acciones anteriores, no se siguen evaluando más reglas.

Adicionalmente a las dos acciones básicas anteriores, existe una tercera acción denominada NFQUEUE que se ajusta al diseño en el espacio de usuario de la implementación de AODVv2 propuesta en este trabajo. Así pues, cuando a un paquete se le aplica una acción NFQUEUE, se almacena en una lista enlazada que es accesible desde un código que se está ejecutando en el espacio de usuario. La comunicación entre el kernel y el espacio de usuario se realiza mediante un protocolo basado en mensajes llamado netlink [17]. Cuando un paquete se encola, el kernel notifica al software en el espacio de usuario este evento mediante un mensaje que contiene el paquete y una información adicional de asistencia. Es entonces cuando el software procesa el paquete y determina un veredicto que notifica al kernel, que actúa conforme la decisión tomada en el espacio de usuario.

La implementación de AODVv2 propuesta en este trabajo modifica tanto la Forwarding Information Base (FIB) como la Routing Information Base (RIB). Como cualquier otro protocolo de encaminamiento, AODVv2 define su propia estructura para la RIB asociada al plano de control, que utiliza para determinar la mejor ruta para un destino y que es su candidata para instalarse en la FIB. En ese caso, esa será la ruta que se utilizará para encaminar los paquetes hacia el destino.

La Fig. 3 muestra el procedimiento seguido por una aplicación Linux que envía un paquete a un nodo MANET para el que se necesita descubrir la ruta. En primer lugar, la aplicación abre un socket para enviar datos. Es entonces cuando se necesita tomar la decisión de encaminamiento. La configuración de una dirección IP en una interfaz de red instala una ruta que asume que todos los nodos de su misma red son directamente alcanzables. Sin embargo, en el caso de las redes MANET esto no es siempre cierto. De todos modos, esta ruta instalada garantiza que los paquetes destinados a la red MANET entrarán dentro del sistema Netfilter para ser procesados (i.e., la FIB tiene una ruta y el paquete IP atravesará el punto de enganche OUTPUT). En cualquier otro caso, el paquete se descartará.

Se configura una ruta en OUTPUT de tal modo que se encolan los paquetes con dirección IP destino perteneciente a la red MANET. Hay una cola distinta para cada uno de los destinos. Una cola se crea en el momento en que se detecta un destino sin cola asociada. Un hilo que se ejecuta en el espacio de usuario comprueba si existe en la RIB de AODVv2 (denominada `Local Route Set` en el RFC [5]). Si existe, se acepta el paquete y se pasa al punto de enganche POSTROUTING para finalmente ser transmitido. En caso contrario, se inicia el proceso de descubrimiento de ruta.

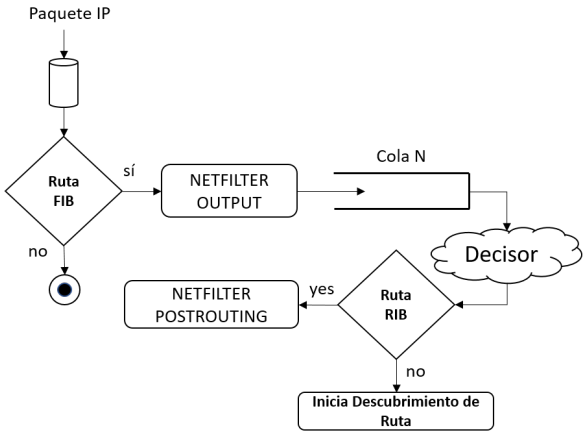

Fig. 3. Captura y procesado de un paquete IP en el espacio de usuario

Las rutas almacenadas en la RIB incluyen una única dirección destino, i.e., el tamaño del prefijo es igual al número de bits de la dirección IP destino: 32 bits en el caso de direcciones IPv4 y 128 bits en el caso de direcciones IPv6. AODVv2 define también un *Router Client Set* en cada router, i.e. un conjunto de nodos que utilizan al router como *relay*. El objetivo de este conjunto es permitir a los routers AODVv2 descubrir rutas para aplicaciones que se ejecutan en los nodos que pertenecen al *Router Client Set*, además de las que descubre para sus propias aplicaciones. En este punto, nótese que no es necesaria una ruta a la dirección destino en la FIB ya que la decisión de encaminamiento se tomará utilizando las rutas almacenadas en la RIB AODVv2. Así pues, la RIB pertenece tanto al plano de control como al de encaminamiento.

En el resto del artículo, el término nodo hace referencia a cualquier participante en la red AODVv2 y el término router hace referencia a los nodos que participan en el encaminamiento de los paquetes de datos.

A. Estructuras de Datos

La especificación del protocolo define varias estructuras de datos que tienen que ser implementados. A continuación se entra en los detalles de estas estructuras.

Como se ha mencionado anteriormente, un router AODVv2 puede descubrir rutas tanto para sus aplicaciones locales como para aquellas que se ejecutan en los nodos que pertenecen a su *Router Client Set*. Actualmente, la implementación sólo ha sido probada para aplicaciones locales, así que el *Router Client Set* solo incluye la dirección local de router y no hay una manera de añadir más nodos al conjunto. Sin embargo, la implementación de la estructura facilita la futura implementación de esta característica del protocolo.

El *Neighbor Set* es una estructura de datos que permite a los routers AODVv2 recopilar información de otros routers AODVv2 vecinos. Los enlaces con un router AODVv2 vecino se clasifican de acuerdo a su estado: CONFIRMED, HEARD y BLACKLISTED. Cuando un router AODVv2 es consciente de la presencia de un vecino, inicializa su estado a HEARD y comprueba si el enlace permite una comunicación bidireccional. Si lo permite, el router actualiza el estado a CONFIRMED. En caso contrario, el estado del router cambia a BLACKLISTED. Esto tiene su importancia ya que solo los vecinos con estado CONFIRMED pueden ser seleccionados como siguiente salto para un determinado destino.

Todas las rutas descubiertas por el protocolo se almacenan en la estructura de datos *Local Route Set*. Todas las rutas tienen una propiedad de estado con cuatro valores posibles: UNCONFIRMED, IDLE, ACTIVE e INVALID. Sólo se pueden utilizar para encaminar rutas activas. Una ruta UNCONFIRMED no se considera activa porque el enlace aún no ha sido confirmado como bidireccional. Una ruta INVALID es aquella que ha expirado o que ha detectado la caída de un enlace y, por tanto, no es una ruta válida. Las rutas con estado IDLE o ACTIVE son activas y se pueden utilizar para encaminar paquetes. Una ruta IDLE es aquella que no se ha utilizado durante un determinado tiempo configurable. Si se utiliza una ruta IDLE para encaminar un paquete, inmediantamente su estado pasa a ACTIVE.

Existen tres estructuras de datos más para completar la implementación del protocolo AODVv2. The *Multicast Message Set* se utiliza para evitar el procesado y reenvío innecesario de mensajes de control. Un mensaje RREQ se almacena en el *Multicast Message Set* y tiene asociado un temporizador de expiración (RREQ_WAIT_TIME), cuyo valor por defecto es 2. Sólo se generará un nuevo RREQ si este temporizador ha expirado.

La estructura *Route Error Set* almacena datos relacionados con direcciones inalcanzables. Finalmente, la estructura *Interface Set* almacena los detalles de todas las interfaces de red configuradas para enviar y recibir mensajes AODVv2.

La implementación se ejecuta como un demonio `systemd`, cuyo rendimiento se puede ajustar a través de los valores de un fichero de configuración. Este fichero incluye los valores de varios parámetros de AODVv2 que están agrupados en tres categorías: temporizadores, constantes del protocolo y parámetros de administración y control. Los valores de estos parámetros están inicialmente establecidos en los valores definidos por defecto en [5].

B. Mensajes del protocolo

AODVv2 define cuatro tipos de mensajes: RREQ, RREP, RERR y RREP_ACK. El primero se utiliza en el proceso de descubrimiento de rutas. Un mensaje RREP lo envía un router AODVv2 como respuesta a un mensaje RREQ destinado a cualquier dirección incluida en el *Router Client Set*. Como ya se ha dicho, en la implementación actual el *Router Client Set* solo contiene la dirección del propio router. Los mensajes RREP_ACK se utilizan para comprobar la bidireccionalidad con un vecino y pasar la ruta, en caso positivo, a estado ACTIVE. Cuando un router reenvía un mensaje RREP_ACK, a su vez envía un mensaje RREP_ACK al candidato a siguiente salto. Los mensajes RERR se generan para notificar rutas que han dejado de estar disponibles para un router.

C. Operación del protocolo

La Fig. 4 muestra el diseño del diagrama de flujo de nuestra implementación. El hilo `decisor_thread` es el responsable de interceptar los paquetes que pasan por el punto de enganche OUTPUT. Para ello se crea una regla iptables que almacena en una cola NFQUEUE todos los paquetes cuya dirección destino pertenece a la red MANET. Una vez encolados, en el espacio de usuario, el hilo utiliza la librería `libnetfilter_queue` para conectarse a la cola, la cual se caracteriza por un identificador de 16 bits (el identificador por defecto es 0), y para obtener los mensajes del kernel tan pronto como un paquete sea encolado. Esto se hace a través de una función de callback definida por el hilo `decisor_thread`. Esta función se llama cada vez que el kernel encola un paquete. La función de callback comprueba si existe una ruta activa en el *Local Route Set*. Si así ocurre, el veredicto será aceptar el paquete y este será reinyectado para continuar su camino en Netfilter pasando al punto de enganche POSTROUTING. En caso contrario, la implementación utiliza un mapa múltiple (multimap) en que cada clave corresponde a una dirección destino y tiene como valor asociado una cola en la que todos los paquetes con dicha dirección destino serán encolados y esperarán a que acabe el proceso de descubrimiento de ruta. Si no se encontrase una ruta dentro de un determinado período de expiración, todos los paquetes de la cola serían descartados y la cola, borrada.

Otro hilo, llamado `msg_thread`, es el responsable del procesado de los mensajes de control de AODVv2. Tiene un socket escuchando en la dirección multicast de enlace local 224.0.0.109, puerto 269, es decir, en la dirección multicast y puerto estándar asignada a los protocolos MANET [18]. A través de este socket se reciben y procesan los mensajes de control de AODVv2 según las especificaciones del protocolo. Así pues, el hilo `msg_thread` es el responsable del procesado de los mensajes de control de AODVv2 y el hilo `decisor_thread` es el responsable de iniciar el proceso de descubrimiento de rutas en caso de que no exista una ruta al prefijo destino. Cuando el hilo `msg_thread` descubre una ruta

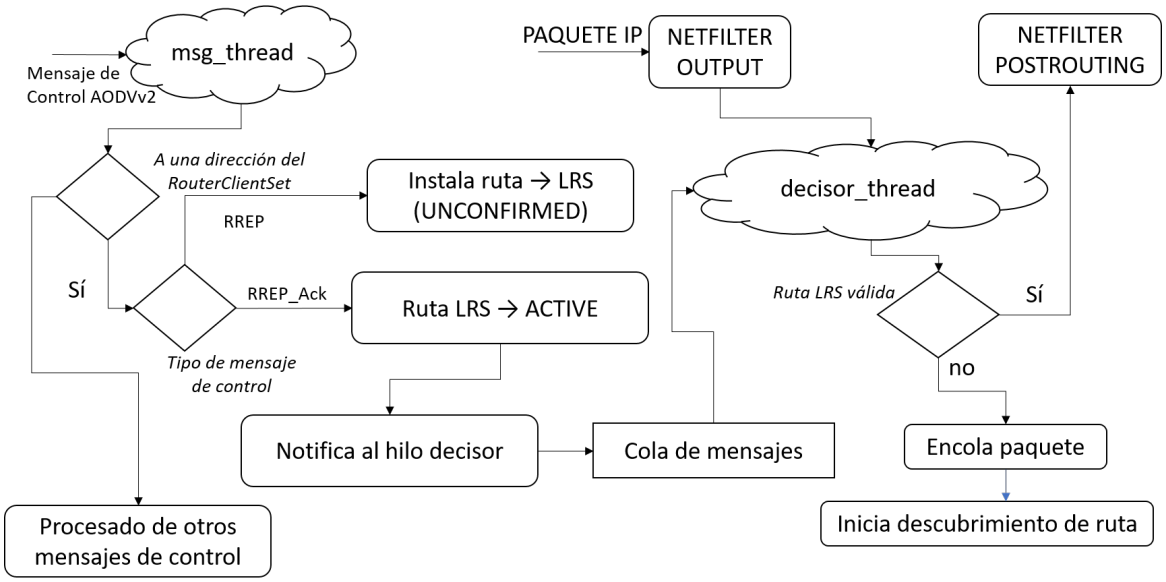

Fig. 4. Diagrama de flujo del diseño de la implementación de AODVv2. LRS se refiere a *Local Route Set*.

válida, i.e. recibe el mensaje RREP que corresponde al prefijo destino, la instala en la *Local Route Set* con estado UNCONFIRMED y envía un mensaje RREP_ACK. Al recibirse el correspondiente mensaje RREP_ACK desde el siguiente salto, la ruta pasa a tener estado VALID y el hilo `msg_thread` señaliza este evento a través de una cola de mensajes al hilo `decisor_thread` que aceptará el paquete para que continúe atravesando Netfilter.

Tanto la recepción como el procesado de los mensajes de control AODVv2 se realizan en el hilo `msg_thread`. Estas operaciones incluyen la monitorización de la disponibilidad de los siguientes saltos de una ruta, el mantenimiento del *Neighbor Set*, la transmisión de mensajes de control del protocolo en respuesta a los enviados por otros routers y el mantenimiento actualizado del *Local Route Set*.

IV. VALIDACIÓN Y PRUEBAS DE LA IMPLEMENTACIÓN

En última instancia la implementación debe ser validada y probada. En primer lugar, se ha definido una suite de pruebas completa a fin de verificar que la implementación del protocolo AODVv2 satisface el comportamiento definido en las especificaciones [5]. Finalmente, se ha ejecutado la implementación en una red sencilla con el objeto de verificar su correcto funcionamiento en dispositivos reales.

El propósito de la suite de pruebas es validar que la implementación cumple con las especificaciones del protocolo. Para ello, es preciso verificar que la generación, recepción y reenvío de cada mensaje de control AODVv2 satisface las acciones que se indican en la especificación. Para cada uno de los *casos de uso* (e.g., generación de una RREQ, reenvío de una RREQ, recepción de una RREP, etc.), se ejecuta tanto la implementación como la suite de pruebas que fuerza el *caso de uso* que queremos verificar.

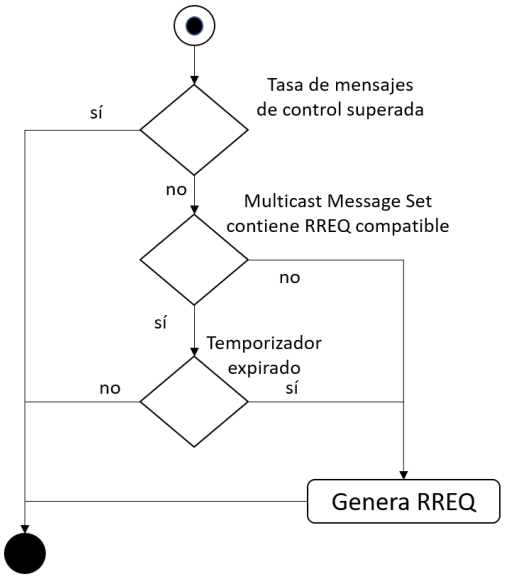

Fig. 5. Diagrama de flujo de una generación de RREQ.

Considérese, por ejemplo, el *caso de uso* de la generación de una RREQ mostrado en el diagrama de flujo de la Fig. 5. Una RREQ se genera solo si la tasa de mensajes de control no supera un cierto umbral y aplica una de las siguientes condiciones: a) el *Multicast Message Set* no contiene una RREQ compatible; o b) la contiene pero el temporizador ha expirado. Al verificar la implementación, la herramienta de pruebas prepara un estado para cada una de las posibles ramas del diagrama de flujo. En el caso que queramos comprobar si un paquete IP se envía solo si existe una ruta en la *Local Route Set* (como se explicó en la Sección III, ver Fig. 3), la herramienta instala esa ruta *antes* de intentar enviar un paquete. Con la ruta instalada, la herramienta envía un ICMP Request a través del sistema Linux y con la ayuda de una herramienta

de monitorización de tráfico de red verifica su envío. Similarmente, en el caso que queramos verificar si un paquete IP se encola cuando ya hay una RREQ compatible en curso (ver Fig. 4), la herramienta añade esta RREQ en el *Multicast Message Set* con un temporizador que aún no ha expirado y traza los elementos que hay en la cola para comprobar que el paquete está allí. Después espera hasta que el temporizador expira y vuelve a intentar enviar un ICMP Request para verificar que en esta ocasión sí se envía una RREQ.

Resumiendo, para cada *caso de uso* la herramienta genera un mensaje que satisface los requisitos y, con la ayuda de trazas y captura de los posibles mensajes generados por el router AODVv2, se verifica que la implementación es correcta.

Con la implementación comprobada localmente mediante la herramienta de verificación, se ha procedido a probar la implementación bajo condiciones más realistas. Para ello, se ha configurado una red de portátiles con arquitectura Intel x86 sobre el kernel Linux 4.4; además, para verificar su portabilidad, la implementación también ha sido probada en Raspberry Pi bajo arquitectura ARM con un kernel 4.15. En este escenario, las pruebas se han realizado sobre una sencilla red donde uno de los dispositivos, que actúa de emisor, genera tráfico ICMP destinado a otro dispositivo situado en el extremo opuesto de la red. Este escenario sirve como prueba de concepto para verificar el establecimiento de una ruta multisalto entre emisor y receptor. Todos los dispositivos tienen una interfaz de red IEEE 802.11 Wireless Network Interface Card (WNIC) que funciona en la banda de 2.4Ghz. Debido al limitado espacio físico del laboratorio donde se realizaron las pruebas, de unos 30 metros de largo, así como a la amplia cobertura de la tecnología radio utilizada, se redujo artificialmente la cobertura mediante las siguientes restricciones: primero, se configuró la tarjeta para que transmitiese a la mínima potencia posible; y, segundo, se incluyó una regla iptables en la cadena PREROUTING de cada dispositivo (ver Listado 1) para forzar la eliminación de todos los paquetes con dirección MAC origen igual a la de cada dispositivo que se desea "situar" fuera de cobertura.

Listing 1. Regla utilizada para eliminar paquetes según la dirección MAC origen

```
iptables \
  -A PREROUTING \
  -m mac --mac-source [MAC_ADDRESS] \
  -j DROP
```

La Fig. 6 muestra un escenario de pruebas con cuatro dispositivos, tanto portátiles como Raspberry Pi, etiquetados como *A*, *B*, *C* y *D*. Las flechas dobles que conectan un par de nodos representan el intercambio mutuo de datos entre dicho par de nodos. Los nodos que no están conectados por una flecha doble, no intercambian datos. En el ejemplo, el dispositivo *A* configura dos reglas iptables: una para eliminar los paquetes con dirección MAC origen igual a la dirección MAX de la interfaz del dispositivo *C* y otra

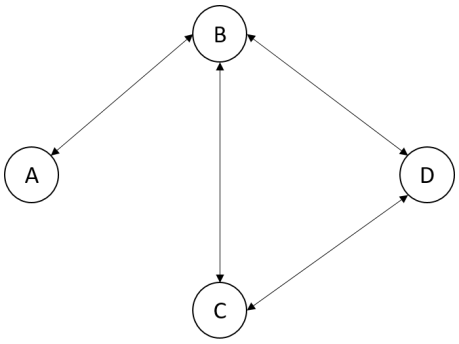

Fig. 6. RREQ Ejemplo de escenario de pruebas.

para la dirección MAX del dispositivo *D*. A su vez, los dispositivos *C* y *D* configuran sus reglas iptables con la dirección MAC del dispositivo *A* para descartar las tramas destinadas a este dispositivo.

Nótese que aunque la implementación se ha probado para las arquitecturas x86 y ARM, podría ejecutarse sin dificultades sobre otros dispositivos con sistema operativo Linux.

V. CONCLUSIONES Y TRABAJO FUTURO

El presente artículo describe una implementación del protocolo de encaminamiento AODVv2 para redes ad hoc, programado sobre el espacio de usuario del sistema operativo Linux, que aumenta la independencia del kernel y permite una fácil portabilidad sobre diferentes dispositivos. A lo largo del artículo se han discutido las decisiones de diseño de implementación y se han presentado los escenarios sobre los que se ha verificado el funcionamiento del programa. Para ello se ha creado un banco de pruebas sobre tráfico ICMP. A pesar de que existen otras implementaciones del algoritmo en la literatura sobre AODVv2, la mayoría de dichas versiones están diseñadas únicamente con fines de simulación, han quedado obsoletas o bien se circunscriben a ciertas versiones antiguas de software. La implementación propuesta en el presente artículo, que es de código abierto y mantenimiento simple, se dirige a todos los investigadores que trabajan en protocolos de encaminamiento MANET.

Actualmente, la implementación se encuentra en una etapa inicial de desarrollo, ya que precisa ser probada en escenarios más amplios y complejos para ser correctamente perfilada y, por tanto, optimizada. También contiene trazas de desarrollo con el fin de facilitar el trabajo de depuración y ajuste. Estos fragmentos de código, que se pueden desactivar en términos de funcionalidad, se eliminarán cuando se considere que el código está listo para su libre distribución.

Como se señala en la Sección III, la funcionalidad *Router Client Set* queda pendiente de implementar. Además, aunque no resulta necesario para el proceso actual de prueba y validación, está previsto ampliar la configuración de la aplicación para permitir múltiples interfaces inalámbricas en cada dispositivo. Una vez haya concluido esta labor de creación de perfiles, se llevará a cabo una prueba más ambiciosa, con varios dispositivos Raspberry

Pi distribuidos a lo largo de un mismo edificio, con el fin de evaluar el funcionamiento de la implementación del protocolo en condiciones más realistas. Esto supondrá el diseño de encaminamientos más complejos así como el empleo de patrones de tráfico más realistas que el tráfico ICMP empleado en el presente trabajo.

REFERENCIAS

[1] A. Mishra, S. Singh, A. Tripathi, "Comparison of Manet Routing Protocol," International Journal of Computer Science and Mobile Computing, vol. 8, pp. 67-74, 2019.

[2] P. Shailaja, C.V. Guru Rao, A.Nagaraju, "A Parametric Oriented Research on Routing Algorithms in Mobile Adhoc Networks," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 9, no. 1, November 2019.

[3] I. Snigdh, D. Gosain, (2015). Analysis of scalability for Routing Protocols in Wireless Sensor Networks. Optik - International Journal for Light and Electron Optics. 127. 10.1016/j.ijleo.2015.11.077.

[4] Perkins, C., Belding-Royer, E., and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, DOI 10.17487/RFC3561, July 2003, https://www.rfc-editor.org/info/rfc3561.

[5] C. Perkins, S. Ratliff, J. Dowdell, L. Steenbrink, and V. Mercieca, "Ad Hoc On-demand Distance Vector Version 2 (AODVv2) Routing," Work in Progress, draft-perkins-manet-aodvv2, May 2016.

[6] T. Clausen, C. Dearlove, J. Dean, and C. Adjih, "Generalized Mobile Ad Hoc Network (MANET) Packet/Message Format", RFC 5444, DOI 10.17487/RFC5444, February 2009.

[7] T. Kumar-Saini and S.C. Sharma, "Recent advancements, review analysis, and extensions of the AODV with the illustration of the applied concept," Ad Hoc Networks, 103, 2020, https://doi.org/10.1016/j.adhoc.2020.102148.

[8] P. Rajankumar, P. Nimisha and P. Kamboj, "A comparative study and simulation of AODV MANET routing protocol in NS2 & NS3," 2014 International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2014, pp. 889-894, doi: 10.1109/IndiaCom.2014.6828091.

[9] C. Sommer, I. Wagner, F. Dressler, "A simulation model of DYMO for ad hoc routing in OMNeT++," Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops, SimuTools 2008, Marseille, France, March 3-7, 2008.

[10] AODV-UU source from GitHub, April 13, 2011. https://github.com/erimatnor/aodv-uu (last accessed on March 2021)

[11] W. Backes and J. Cordasco, "MoteAODV – An AODV Implementation for TinyOS 2.0.," Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices. WISTP 2010. Lecture Notes in Computer Science, 6033. Springer, doi: 10.1007/978-3-642-12368-9_11.

[12] E. Gaona-García, S. Palechor-Mopán, L. Murcia-Sierra, P. Gaona-García, "Implementation of the AODV Routing Protocol for Message Notification in a Wireless Sensor Microgrid," 5th Workshop on Engineering Applications, WEA 2018, Medellín, Colombia, October 17-19, 2018, Proceedings, Part II. 10.1007/978-3-030-00353-1_32.

[13] J. S. Awati, S. A. Patil and M. R. Patil, "Implementation of AODV Routing Protocol of Wireless Sensor Network in Agriculture," 2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, 2018, pp. 1-9, doi: 10.1109/ICOEI.2018.8553730.

[14] S. Jung, B. Kim, K. Kim, B. Roh and J. Ham, "Implementation of AODV-UU on Linux 4.15 Kernel," 2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), Monterey, CA, USA, 2019, pp. 160-161, doi: 10.1109/MASSW.2019.00039.

[15] P. García López, R. García Tinedo, and J. M. Banús Alsina, "Moving routing protocols to the user space in MANET middleware," 2010 Journal of Network and Computer Applications, pp. 588-602.

[16] http://netfilter.org. Visited the 21th December 2020.

[17] Salim, J., Khosravi, H., Kleen, A., and A. Kuznetsov, "Linux Netlink as an IP Services Protocol", RFC 3549, DOI 10.17487/RFC3549, July 2003, ¡https://www.rfc-editor.org/info/rfc3549¿.

[18] Chakeres, I., "IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols", RFC 5498, DOI 10.17487/RFC5498, March 2009.