***UNIVERSITAT OBERTA DE CATALUÑA***
***MASTER EN SOFTWARE LIBRE***

# *PROYECTO FIN DE MASTER (INVESTIGACIÓN)*

## *Memoria Final*

Pedro J. Ruiz Yelo

# Contenidos

# Título

Estudio general de Redes de Sensores Inalámbrica e Implantación del protocolo RIP sobre una plataforma hardware y basándose en el proyecto de Software Libre OpenWSN

# Resumen

Actualmente el proyecto OpenWSN dispone de una pila completa de protocolos para la comunicación de la red pero se ha visto interesante la dotar de una alternativa a la capa de enrutamiento dentro de la pila de protocolos implementada actualmente. Esta capa está actualmente implementada con RPL y se pretende implementar RIP, un protocolo de enrutamiento IGP (Internal Gateway Protocol) presente en los orígenes de la interconexión de redes y que ha ido evolucionando y se ha adaptado a la nueva pila IPv6.

Para la realización del proyecto se ha llevado a cabo una labor de investigación de este protocolo a través de los diferentes documentos técnicos (RFC) que detallan los aspectos técnicos a tener en cuenta para la implementación del mismo.

El **modelo de investigación** para la implentación del protocolo se ha utilizado la aproximación a los “ciclos de vida tradicionales del desarrollo de aplicaciones”, concretamente en cada uno de los apartados:

- **Análisis**. Se han analizado las redes WSN, los protocolos implementados en OpenWSN (EEE802.15.4-2015, IEEE802.15.4e, IETF 6LoWPAN, IETF RPK,CoAP) enfocando el detalle en el protocolo RPL y el protocolo RIP.
- **Diseño**. Definición de estructuras e intercambio de mensajes para el protocolo RIP sobre OpenWSN.
- **Implementación**. Implementación sobre la pila OpenWSN.
- **Pruebas**. Especificación y realización de pruebas sobre ambos entornos.
- **Análisis de resultado y conclusiones.** Análisis de las condiciones en las que uno de los protocolos es mejor que el otro y evaluación de estas mejoras.

# Introducción

El objetivo el estudio general de las redes inalámbricas, la profundización sobre la pila de protocolos que implementa el proyecto OpenWSN y el desarrollo del protocolo RIP dentro de esa pila, de tal manera que sirva de opción al protocolo RPL, ya adoptado por el proyecto de software libre mencionado y la comparación de los mismos.

El proyecto OpenWSN es un proyecto cuyo objetivo es la implementación de una pila completa de protocolos basada en IoT y que pueda ejecutarse en una variedad de plataformas software y hardware. Esta implementación pretende incluir los avances que se produzcan en las distintas capas de la pila de protocolos, así incluye el estandar TSCH dentro de la capa MAC, 6LoWPAN para optimizar las cabeceras de IpV6, un protocolo de enrutamiento optimizado para estas redes como RPL o CoAP para compatibilizar la conexión con internet[14].

La pila de protocolos actualmente implementada en el proyecto es la siguiente:

| | |
|---|---|
| application | CoAP, HTTP |
| transport | UDP, TCP |
| IP/routing | IETF RPL |
| adaptation | IETF 6LoWPAN |
| medium access | IEEE802.15.4e |
| phy | IEEE802.15.4-2006 |

Un grupo de trabajo de la IETF denominado ROLL (Routing Over Low powwer and Losy networks) ha propuesto el protocolo RPL (IPv6 Routing Protocol for Low power and Lossy Networks)[10].

RPL soporta una gran variedad de capas de enlace y concretamente las que están bastante limitadas, son potencialmente ruidosas o tienen recursos limitados como las que se implementan en entornos hostiles como es el caso de las redes de sensores. Este protocolo es muy hábil en construir rutas, distribuir estas entre los nodos y adaptar la topología de red de una forma muy eficiente[11].

Tipicamente la red es MP2P, desde los nodos sensores a los nodos root en el que se define un grafico DODAG (Destination Oriented Directed Acyclic Graph) en el cual se definen los caminos disponibles y la idoneidad de los mismos, basándose en el coste del enlace, el estado del mismo y una función objetivo, que depende de cada aplicación. Esta métrica de idoneidad se denomina Rank y va decreciendo a través del gráfico hacia su destino de acuerdo con una aproximación gradiente [12,13].

Aunque el trafico MP2P es el predominante, es decir desde los nodos finales hacia el root, también es posible el tráfico P2MP, del nodo root hacia los nodos finales para propósitos de actuación sobre ellos (mantenimiento y control, e incluso actuación sobre el entorno). También es posible el tráfico P2P entre iguales.
El método de transmisión de la información de enrutamiento es vía mensajes DIO (DODAG information Option), los cuales contienen información sobre el rank, la función objetivo, id, etc.

El Protocolo RIP es un protocolo IGP cuyo algoritmo de encaminamiento está basado en el denominado vector de distancia, ya que calcula la métrica o ruta más corta posible hasta el destino a partir del número de saltos que los paquetes IP deben realizar. El protocolo tiene una limitación de saltos a 15, así que no puede alcanzar objetivos ubicados a mayor distancia. La gran ventaja frente a otros protocolos es que es un protocolo libre y por lo tanto utilizado por distintas plataformas y no solamente por las de un único fabricante[4-9].

Aunque el protocolo tiene sus desventajas en cuanto a las limitaciones, a cambio tiene la

ventaja de la facilidad de implementación y puede ser util en determinadas configuraciones y condiciones de red. Esta ventaja es la que se pretende aprovechar en este proyecto y determinar cuales son dichas condiciones.

# Objetivos Iniciales

Los **objetivos** concretos de la investigación son:

- Estudio general de las redes WSN y del proyecto OpenWSN en particular.
- Estudio de los protocolos RIP y RPL. Investigación de la estructura y funcionamiento de ambos protocolos y su comparación, enfocando su aplicación a las características particulares de las redes WSN.
- Diseño del protocolo RIP sobre redes WSN (OpenWSN) . Diseño de los componentes necesarios para la implementación del protocolo RIP (estructura de datos, paso de mensajes,...) .
- Implantación del protocolo diseñado en OpenWSN en un entorno simulado.
- Análisis y comparación de ambos protocolos sobre WSN. Estudio del comportamiento y del rendimiento de los dos protocolos de enrutamiento en redes WSN.

# Desarrollo del proyecto

## Estudio RPL

El estudio del protocolo RPL está fundamentado en los siguientes documentos:

- T. Winter, Ed.; P. Thubert, Ed. (Cisco Systems ), A. Brandt (Sigma Designs ) y otros, 2012.RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks (RFC 6550)
- Hui, J; Vasseus, JP (Cisco Systems), 2012. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams (RFC6553)
- J. Hui ; JP. Vasseur (Cisco Systems ), D. Culler (UC Berkeley ), V. Manral (Hewlett Packard Co.), 2012. An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL) (RFC6554)

Para empezar debemos definir una serie de elementos que conforman la arquitectura del protocolo:

- DAG (Direct Acyclic Graph). Grafo que dirige la información entre los nodos.
- DAG root. Nodo raíz del grafo.
- DODAG (Destination Oriented DAG). Grafo que tiene una raíz y esta configurado para

encaminar la información a dicha raíz.

- Rank. Posición relativa del nodo dentro del grafo.
- OF (Objetive function). Función objetivo y que se define según la aplicación a implementar·
- RPLInstance. Instancia que agrupa una serie de nodos ordenados en forma de grafo según una función objetivo. Un nodo puede pertenecer a más de una instancia, con una función objetivo distinta para varias aplicaciones. Y una instancia puede tener más de un DODAG.

Un ejemplo de instancia RPL lo encontramos en el siguiente ejemplo:

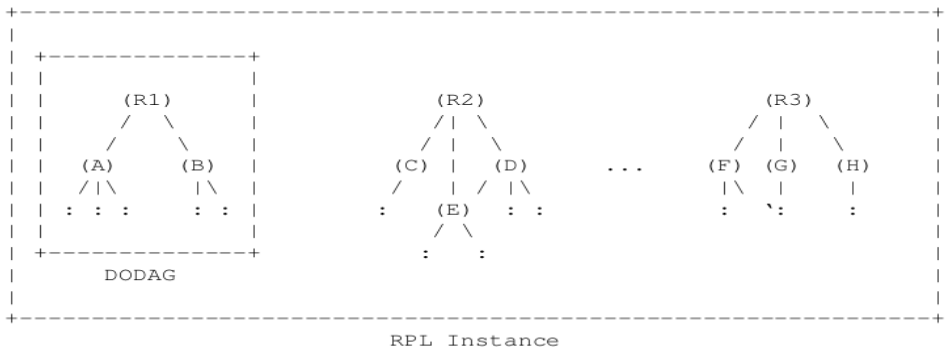



Figure 1: RPL Instance

También tenemos identificadores para cada uno de los niveles de instanciación de los objetos: RPLInstance ID, Dodag ID, Dodag Version Number, Rank. Estos identificadores agrupan los objetos de una forma jerárquica. Así podemos disponer de, además de lo indicado en el gráfico anterior, distintas versiones de un DODAG, según va evolucionado la topología con el tiempo.

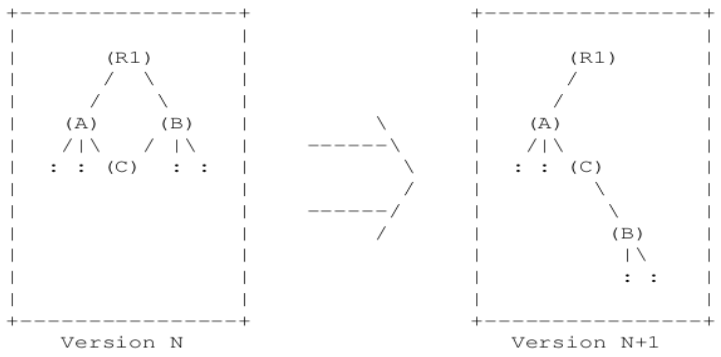



Figure 2: DODAG Version

También disponemos de distintos tipos de paquetes para transmitir la información de enrutamiento necesaria:

- DIO (DODAG Information Object). Este paquete es el que suministra información del DODAG a los nodos que quieran adherirse al mismo.

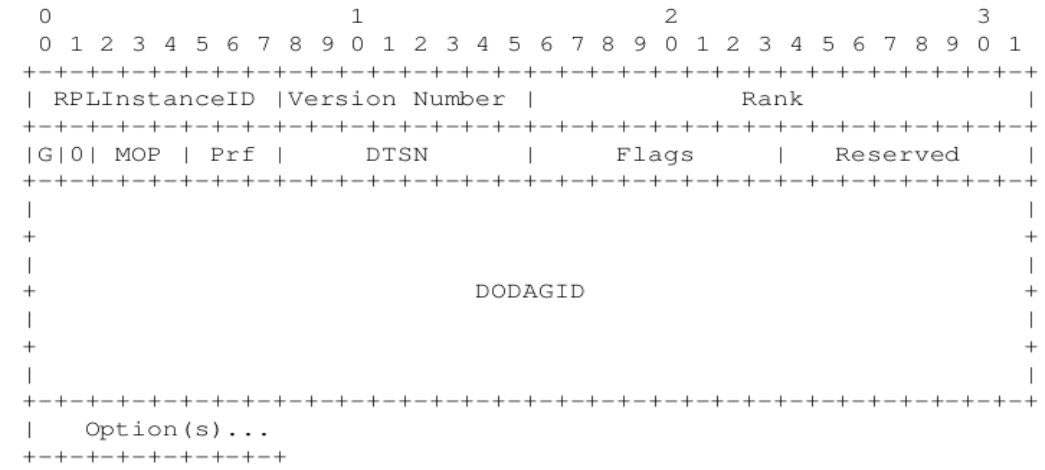
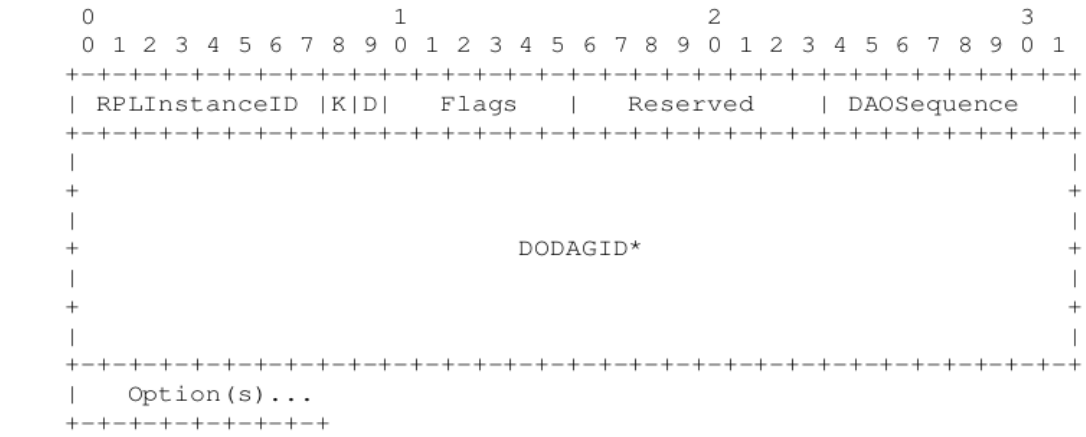



Figure 14: The DIO Base Object

- DAO (DODAG Advertisement Object). Se utiliza para solicitar información a los nodos.




The '*' denotes that the DODAGID is not always present, as described below.

Figure 16: The DAO Base Object

- DIS (DODAG Information Solicitation). Cuando un nodo necesita información solicita que se le suministgre con este tipo de paquete.

- CC (Consistency check). Se utiliza para chequear nodos y comunicaciones.

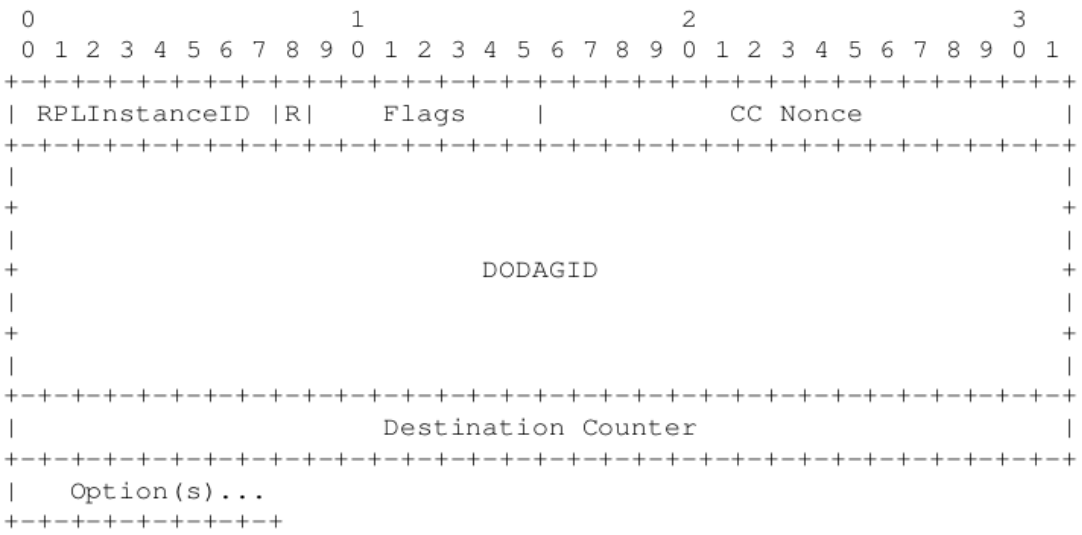



Figure 18: The CC Base Object

Todos los paquetes disponen de diversas opciones de configuración y de securización lo que permite el tráfico seguro de la información.

El modo de operación básico es el siguiente:

1. Se establece un nodo como root (DODAG root).
2. Este envía paquetes DIO suministrando información
3. Los nodo escuchan los DIOs y si procede se unen al DODAG
4. Los nodos van adquiriendo información para aprovisionar la tablas de enrutamiento

Disponemos de tres tipos de flujo de información:

- Multipoint to Point. Es el tráfico más habitual, los nodos van pasando información desde las hojas a la raiz. Este tráfico también se denomina “upward”.
- Point to Multipoint. Es el tráfico desde el nodo raiz o un subnodo de este hacia las hojas, por motivos de configuración. También se denomina “downward”. Para este tráfico se utilizan los paquetes DAO.
- Point to Point. Es el tráfico entre nodos intermedios y se utiliza por motivos de chequeo.

# Estudio RIP

El estudio del protocolo está basado en la información obtenida por los documentos oficiales de estandarización y que son los siguientes:

- Hedrick, C (Rutgers University), 1998. *Routing Information Protocol*. Network Working Group, RFC 1058
- Braden, R & Postel, J (ISI), 1987. *Requirements for Internet Gateway*s. Network Working Group, RFC 1009
- Malkin, G. (Bay Networks), 1998. *RIP Version 2*. RIP Version 2. Network Working Group, RFC 2453
- Malkin, G (Xylogics, Inc); Backer, F (Advanced Computer Communications), 1993. *RIP Version 2 MIB Extension*. Network Working Group, RFC 1389
- Baker, F & Atkinson, R (Cisco Systems), 1997. *RIP Version 2 MD5 Authentication*. Network Working Group, RFC 2082
- Malkin, G. (Xylogic, Inc); Minnear, R. (Ipsilon Networks), 1997. RIPng for IPv6. Network Working Group, RFC 2080

En base a estos documentos se pueden sacar las caractéristicas generales del protocolo:

- El protocolo está basado en el algoritmo de vector discancia de Bellman-Ford
- Es un protocolo IGP, adecuado para Sistemas Autónomos, es decir, sistemas con un solo propietario, debido a que la información de enrutamiento y la arquitectura de la red debe de ser conocida por todos los routers de la misma.
- Debido a la limitación de 15 saltos en los paquetes de enrutamiento este protocolo solamente es aplicable a redes con tamaño moderado
- Debe de implementarse con tecnología homogénea, ya que las métricas que aplica son fijas.
- En el ajuste de métrica de las distintas rutas debido al cambio de topología de red puede surgir el problema denominado “conteo hasta el infinito”, el cual se soluciona mediante el algoritmo “split horizon”.

El protocolo, como ya se ha comentado, está basado en un algoritmo de “vector distancia” y se intercambia poca cantidad de información. Para conseguir esto las información de red debe de aparecer sumarizada, es decir agrupada en la menor cantidad de rutas. Esto se ha facilitado mucho con la implementación de IPv6 y la jerarquía implícita que incorpora. El protocolo de Bellman-Ford se basa en la siguiente formula:

$$D(i,j)=\begin{cases} 0 & i=j \\ \min\limits_{k}[d(i,k)+D(k,j)] & i\neq j \end{cases}$$

donde d(i,j) indica conexión directa entre el nodo i y el j y D(i,j) indica distancia total entre i y j.

La métrica que incorpora es fija y en cada ruta se mide el total de la distancia al destino, es decir, el número de saltos para alcanzarlo. Para el cálculo de la ruta óptima solamente se intercambia información entre vecinos.

Para el mantenimiento de la información de enrutamiento se mantiene una tabla con una entrada para cada una de las rutas posibles y almacenando la siguiente información:

- Address. Dirección de la red o host de destino.
- Gateway. Primer router en la ruta hacie el destino.
- Interface. Interface física de host por donde se llega al destino.
- Metric. Distancia al destino.
- Timer. Tiempo desde la última actualización.

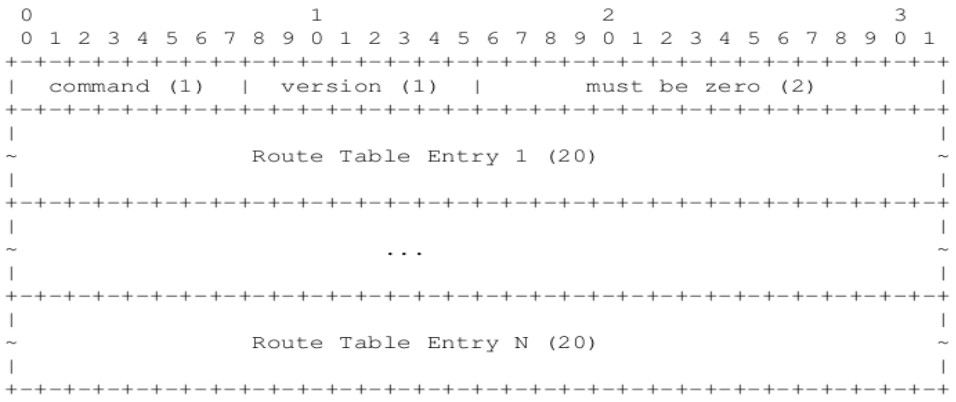
Periódicamente, cada 30 segundos, se envía información de actualización a cada vecino y cuando llega información desde un vecino se actualiza la tabla, esto nos permite mantener la topología de la red correctamente.

Cuando hay un cambio de topología, como la pérdida de conexión a un router, se invalida la ruta correspondiente por un periodo de 180 segundos. Si después de este tiempo sigue sin conexión se recalcula la ruta y se comunica al resto de routers vecinos.

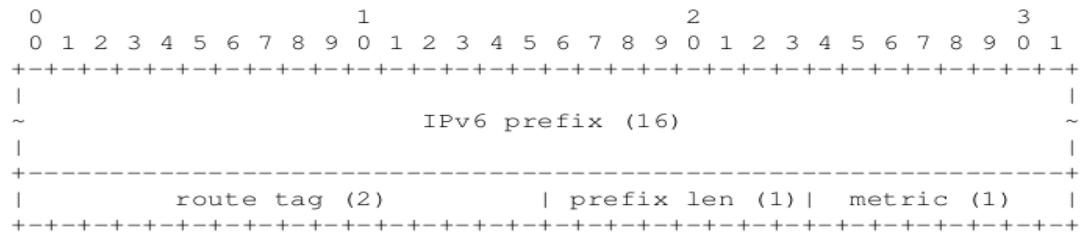
Para prevenir el “conteo hasta el infinito” indicado anteriormente se utiliza la técnida denominada “split horizon” que consiste en omitir las rutas aprendidas desde un router en las actualizaciones de topología que se envían el. De esta manera el destino no incrementa la métrica que este router a pasado y que piensa que está aprendiendo de nosotros. Una modificación es pasar estas rutas pero con metrica “infinito”, con lo cual el destino va a anular estas rutas, esto se denomina “split horizon with poisoned reverse”, que tiene el inconveniente de incrementar los mensajes entre router, por lo que solamente se recomienda en redes estables. El protocolo obliga a la implementación de la primera solución y recomienda la del segundo pero con posibilidad de deshabilitarlo.

Al contrario que RPL, el protocolo RIP solamente tiene un tipo de paquete que se trasmite a través de UDP y con los puertos 521, tanto en origen como en destino. Con este paquete se distribuye toda la información necesaria.

Los campos son los siguientes:




- Command. 1 (request) para solicitudes y 2 (response) para respuestas.
- Version. Versión del protocolo.
- Route Table Entry (RTE). Una para cada ruta. La estructura de este campo lo vemos en el siguiente gráfico.




Las rutas se agrupan según el campo “siguiente salto”, por lo que para cada uno de estos puede existir multiples destinos. Por lo tanto tenemos dos tipos de entradas (RTE), una para indicar siguiente salto y otra para indicar el destino de la ruta:

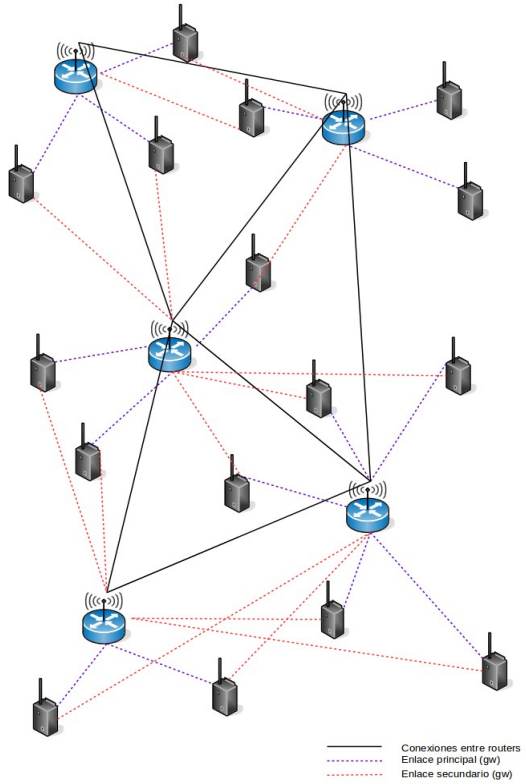
- IPv6 prefix. Para el primer caso es la dirección del siguiente salto y para el segundo el prefijo de la subred destino.
- Route tag. En la primera opción indica una etiqueta y en la segunda debe de estar rellena con ceros.
- Prefix len. En este caso es igual, el prefijo de la red en el primero y cero en el segundo.
- Metric. Indica la métrica de la red en el primer caso y debe tener “0xFF” en el segundo.

El número máximo de entradas que se pueden transmitir por paquete viene definido por la siguiente formula:

$$\#\text{RTEs} = \left[\frac{MTU \;-\; sizeof\left(IPv6\_hdrs\right) \;-\; UDP\_hdrlen \;-\; RIPng\_hdrlen}{RTE\_\text{size}}\right]$$

# Implementación de la solución

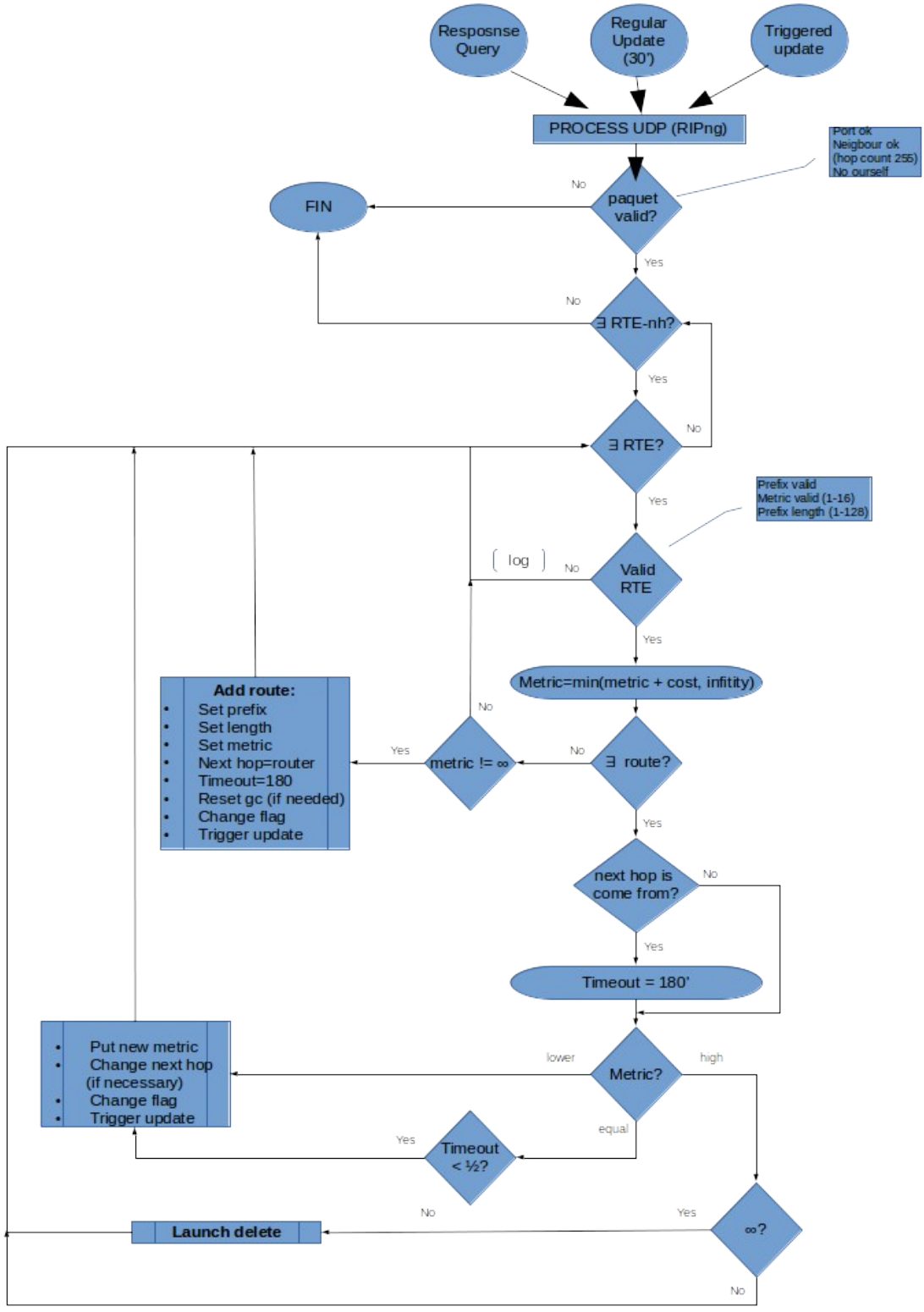
De la análisis de los requisitos y del estudio del protocolo establecemos que se debe de implementar una arquitectura como la siguiente:

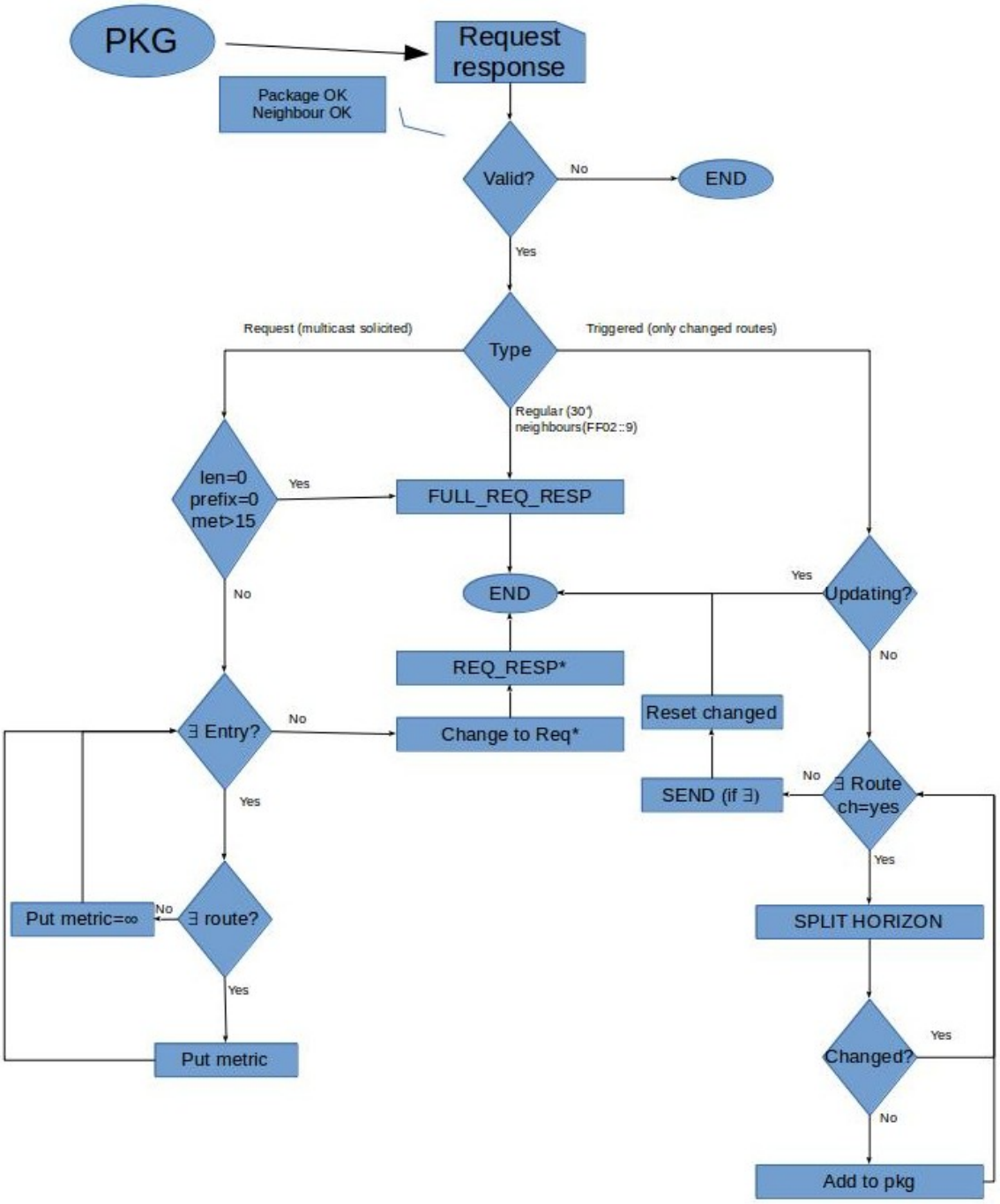



Como se puede observar tenemos dos tipos de elementos:

- Nodos router: motas con capacidad de enrutamiento. Dan servido a una subnet y están conectadas con uno o varios routers. Es en estos nodos donde se debe de implementar toda la lógica de enrutamiento de la red.
- Nodos hoja: motas sin capacidad de enrutamiento y que se conectan a un nodo router. Si este nodo cayera pueden configurarse contra otro router que estuvieran en su alcance. Estos nodos solamente implementarían la capacidad de comunicarse con otros nodos en la misma subred o contra el router y también la capacidad de autoconfiguración contra otro nodo en caso de caída.

Como resultado del análisis también procede los siguientes gráficos:


Resposnse Query
Regular Update (30')
Triggered update
PROCESS UDP (RIPng)
Port ok
Neigbour ok
(hop count 255)
No ourself
paquet valid?
No
FIN
Yes
∃ RTE-nh?
No
Yes
∃ RTE?
No
Yes
Prefix valid
Metric valid (1-16)
Prefix length (1-128)
log
Valid RTE
No
Yes
Metric=min(metric + cost, infitity)
Add route:
Set prefix
Set length
Set metric
Next hop=router
Timeout=180
Reset gc (if needed)
Change flag
Trigger update
No
metric != ∞
Yes
∃ route?
No
Yes
next hop is come from?
No
Yes
Timeout = 180'
Put new metric
Change next hop (if necessary)
Change flag
Trigger update
lower
Metric?
high
equal
Yes
Timeout < ½?
No
Yes
∞?
Launch delete
No


PKG
Request response
Package OK
Neighbour OK
Valid?
No
END
Yes
Type
Request (multicast solicited)
Triggered (only changed routes)
Regular (30')
neighbours(FF02::9)
len=0
prefix=0
met>15
Yes
FULL_REQ_RESP
No
END
Yes
Updating?
No
REQ_RESP*
∃ Entry?
No
Change to Req*
Reset changed
SEND (if ∃)
No
∃ Route
ch=yes
Yes
Yes
Put metric=∞
No
∃ route?
SPLIT HORIZON
Yes
Changed?
Yes
No
Put metric
Add to pkg

Para la implementación del protocolo se ha realizado una librería que implementa el protocolo y una serie de librerías auxiliares que implementan definiciones y funciones necesarias para el protocolo.

## Librerías auxiliares

Para ayudar a la librería principal se han implementado tres librerías, cuyo código se adjuntará como anexo. Se indican solamente la definición de datos y los prototipos:

- **LibNeighbour**. Librería para implementar la lista de vecinos del router y poder controlar si los paquetes proceden de un router válido y para enviar información a los routers adecuados.

```
//======== Neighbours
// Definilción de tipo de datos Vecino
typedef struct Neighbour_t
{
	char 				*addr;
	unsigned short 			alive;
	int 				timeout;
	struct Neighbour_t	*next;
	struct Neighbour_t	*previous;
}Neighbour_t;
// Definilción de tipo de datos Lista de vecinos
typedef struct Neighbour_l
{
	unsigned short		size;
	struct Neighbour_t	*first;
	struct Neighbour_t	*last;
}Neighbour_l;

//====================== PROTOTYPES ================================
struct Neighbour_l *create_NeighbourList(void);				//Funciones de creación y eliminación de lista
int delete_NeighbourList(struct Neighbour_l *);
struct Neighbour_t *get_firstNeighbour(struct Neighbour_l *);		//Funciones de recorrido de lista
struct Neighbour_t *get_nextNeighbour(struct Neighbour_t *);
struct Neighbour_t *get_previousNeighbour(struct Neighbour_t *);
struct Neighbour_t *get_lastNeighbour(struct Neighbour_l *);
int new_neighbour(struct Neighbour_l *,char *);				//Nuevo vecino
struct Neighbour_t *search_Neighbour(struct Neighbour_l *,char *);	//Busca vecino
int is_alive_neighbour(struct Neighbour_t *);
void change_alive_neighbour(struct Neighbour_t *);
char *get_addr_neighbour(struct Neighbour_t *);				//Funciones de entrada/salida de datos en lista
int get_timeout_neighbour(struct Neighbour_t  *);
void set_addr_neighbour(struct Neighbour_t *,char *);
void set_timeout_neighbour(struct Neighbour_t *,int);
```

- **LibRoutingTable**. Librería para implementar la tabla de rutas.

```
//======== Routing Table
// Definilción de tipo de datos Ruta
typedef struct Route_t
{
	char 			*prefix;	// Prefijo
	unsigned short		len;		// Mascara
	unsigned short		metric;	// Metrica
	char 			*nextHop;	// Gateway
	unsigned short		changed;	// Ruta Cambiada
	int 			timeout;	// Timeout
	struct Route_t *next;
```

```
        struct Route_t *previous;
}Route_t;
// Definilción de tipo de datos Tabla
typedef struct Table_t
{
        unsigned short      size;
        struct              Route_t    *first;
        struct              Route_t    *last;
}Table_t;

//====================== PROTOTYPES ==============================
struct Table_t *create_table(void);
int delete_table(struct Table_t *);
struct Route_t *create_route(char *,unsigned short,unsigned short,char *,unsigned short,int);     // Creación de estructrua de ruta
int new_route(struct Table_t *,struct Route_t *);                                                 // Inserción de ruta en tabla
int delete_route(struct Table_t *,struct Route_t *);
int modify_route(struct Table_t *,struct Route_t *,unsigned short,char *,unsigned short,int);     // Modificación de ruta
struct Route_t *search_route(struct Table_t *,char *,unsigned short);                             // Busqueda de ruta
struct Route_t *get_first(struct Table_t *);                                                      // Funciones de recorrido de lista
struct Route_t *get_next(struct Route_t *);
struct Route_t *get_previous(struct Route_t *);
struct Route_t *get_last(struct Table_t *);
int set_prefix(struct Route_t *, char *);                                                         // Funciones de entrada de datos
int set_len(struct Route_t *, unsigned short);
int set_metric(struct Route_t *, unsigned short);
int set_nexHop(struct Route_t *,char *);
int set_changed(struct Route_t *, unsigned short);
int set_timeout(struct Route_t *,int);
char *get_prefix(struct Route_t *);                                                               // Funciones de extracción de datos
unsigned short get_len(struct Route_t *);
unsigned short get_metric(struct Route_t *);
char *get_nextHop(struct Route_t *);
unsigned short get_changed(struct Route_t *);
int get_timeout(struct Route_t *);
```

- **LibRIPngPacket**. Libería para facilitar la composición de paquetes RIP, así como para extraer información de los mismos.

```
//======== RTE  -- definir tabla de RTEs

// Definición de tipo de datos ruta
typedef struct Rte_t
{
        char                *prefix;
        unsigned short      tag;
        unsigned short      len;
        unsigned short      metric;
        struct Rte_t        *next;
        struct Rte_t        *previous;
}Rte_t;
// Definición de lista de rutas
typedef struct Rte_l
{
        unsigned short      size;
        struct Rte_t        *first;
        struct Rte_t        *last;
}Rte_l;
//======== Formato de paquete RIPng
typedef struct RIPngPacket_t
{
        unsigned short      command;
        unsigned short      version;
        unsigned short      zeroed;
        struct Rte_l        *routes;
}RIPngPacket_t;

struct Rte_l *create_rtelist(void);                                   //Funciones de creación y eliminación de lista
unsigned short delete_rtelist(struct Rte_l *);
struct Rte_t *first_rte(struct Rte_l *);                              //Funciones de recorrido de lista
```

```
struct Rte_t *next_rte(struct Rte_t *);
struct Rte_t *previous_rte(struct Rte_t *);
struct Rte_t *last_rte(struct Rte_l *);
struct Rte_t *new_rte(char *,unsigned short,unsigned short,unsigned short);          //Nueva ruta
struct Rte_t *search_rtenh(struct Rte_l *,char *);                                   //Búsqueda de ruta
struct Rte_t *add_rte(struct Rte_l *,struct Rte_t *,struct Rte_t *);                 //Insercción de ruta
int delete_rte(struct Rte_l *,char *);
char *get_rtePrefix(struct Rte_t *);                                                 //Funciones de entrada/salida de datos en lista
int set_rtePrefix(struct Rte_t *,char *);
unsigned short get_rteLen(struct Rte_t *);
int set_rteLen(struct Rte_t *, unsigned short);
unsigned short get_rteMetric(struct Rte_t *);
int set_rteMetric(struct Rte_t *, unsigned short);
int is_rtenh(struct Rte_t *);

struct RIPngPacket_t *create_RIPngPacket(unsigned short,Rte_l *);                    //Funciones de creación y eliminación de paquetes
int delete_RIPngPacket(struct RIPngPacket_t *);
unsigned short get_RIPngCommand(struct RIPngPacket_t *);                             //Funciones de entrada/salida de datos en paquete
int set_RIPngCommand(struct RIPngPacket_t *,unsigned short);
unsigned short get_RIPngVersion(struct RIPngPacket_t *);
int set_RIPngVersion(struct RIPngPacket_t *,unsigned short);
struct Rte_l *get_RIPngRoutes(struct RIPngPacket_t *);
int set_RIPngRoutes(struct RIPngPacket_t *, struct Rte_l *);
```

### Librería principal

Se ha definido una librería para la implementación de las funciones de procesamiento del protocolo (**libRIPng**).

```
#include "libRIPngPacket.h"

//====================== PROTOTYPES ==============================
void init(void);                                                                     // Inicialización de estructuras y petición inicial
RIPngPacket_t *request_send();                                                       // Solicitud de información
RIPngPacket-t *request_response(RIPngPacket *,char *,int,Table_t *);         // Respuesta a petición de información
int input_proc(RIPngPacket *,int,Table_t *);                                         // Procesamiento de información recibida
```

Las fuentes se proporcionan en ficheros externos

# Conclusiones

## Objetivos conseguidos

Los objetivos conseguidos han sido los siguientes:

- Estudio general de las redes WSN y del proyecto OpenWSN en particular. El estudio del estado del arte se incluye en la PEC2, si bien en esta parte también se ha incidido con mayor profunidad en la parte de protocolos de Red.
- Estudio de los protocolos RIP y RPL. Investigación de la estructura y funcionamiento de ambos protocolos y su comparación, enfocando su aplicación a las características particulares de las redes WSN. Se ha logrado el objetivo en su totalidad.
- Diseño del protocolo RIP sobre redes WSN (OpenWSN) . Diseño de los componentes necesarios para la implementación del protocolo RIP (estructura de datos, paso de mensajes,...) . Se ha realizado el diseño del protocolo tal y como se se ha espuesto en los puntos anteriores.
- Implantación del protocolo RIP. Se han implementado las estructuras y los procedimiento de paso de mensajes parcialmente. Se disponen de las librería auxiliares para dar cobertura al

protocolo (lista de vecinos, tipo y lista de rutas y tabla de enrutamiento, y estructura de paquetes), si bien la librería principal no está terminada.

## Objetivos pendientes

Quedan por tanto pendientes los siguientes puntos:

- Implantación completa del protocolo RIP sobre plataforma OpenWSN. No se ha llegado a la integración de la librería en la pila OpenWSN y por tanto tampoco la simulación sobre ella.
- Análisis y comparación de ambos protocolos sobre WSN. No se ha llegado a abordar este punto.

## Conclusión

Debido a que no se ha terminado de implementar el protocolo y no se han podido hacer pruebas empíricas del protocolo y por lo tanto tampoco ninguna comparación cuantitativa tan solo cabe sacar conclusiones teóricas de las bondades de uno y otro protocolo.

- Por un lado el protocolo RPL es mucho más complejo que el RIP, tanto en los tipos de mensajes a transportar como en la complejidad de los mismos. La cantidad de paquetes a enviar es también mayor, sobre todo en redes estables en las que lo estático que es el protocolo RIP le hace que no sobrecargue prácticamente nada el tráfico de red.
- Por otro lado en redes inestables y ruidosas es mucho más adecuado RPL debido al propio diseño del protocolo y a la adaptabilidad del mismo a entornos cambiantes.
- El protocolo RPL es mucho más adecuado a redes multifuncionales, es decir, redes que despliegan multiples aplicaciones con requisitos distintos para cada aplicación y se pueden utilizar los mismos componentes de red para distintas funciones. En RIP esto no puede hacerse puesto que la métrica es fija y no se pueden establecer distintos enrutamiento dependiendo de la aplicación que se utilice.
- Debido a los propios requisitos del protocolo RIP, este lo hace util solamente en redes no demasiado grandes ya que es un protocolo diseñado para sistemas autónomos de pequeño y mediano tamaño. El crecimiento de la red también es un problema para este protocolo.
- Otra dificultad es la configuración estática del enrutamiento que hay que hacer con los routes de RIP, aunque esto se mitiga bastante con las nuevas funcionalidades de IPv6 para la autoconfiguración y la configuración jerárquica de direcciones, así como las facilidades de multipath que proporciona.
- Por otro lado para redes estáticas y que no cambian con facilidad con el tiempo es mucho más rápido y sobrecarga mucho menos la red el protocolo RIP.

# Referencias

[1] T. Winter, Ed.; P. Thubert, Ed. (Cisco Systems ), A. Brandt (Sigma Designs ) y otros, 2012.RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks (RFC 6550)

[2] Hui, J; Vasseus, JP (Cisco Systems), 2012. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams (RFC6553)

[3] J. Hui ; JP. Vasseur (Cisco Systems ), D. Culler (UC Berkeley ), V. Manral (Hewlett Packard Co.), 2012. An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RPL) (RFC6554)

[4] Hedrick, C (Rutgers University), 1998. *Routing Information Protocol*. Network Working Group, RFC 1058

[5] Braden, R & Postel, J (ISI), 1987. *Requirements for Internet Gateway*s. Network Working Group, RFC 1009

[6] Malkin, G. (Bay Networks), 1998. *RIP Version 2*. RIP Version 2. Network Working Group, RFC 2453

[7] Malkin, G (Xylogics, Inc); Backer, F (Advanced Computer Communications), 1993. *RIP Version 2 MIB Extension*. Network Working Group, RFC 1389

[8] Baker, F & Atkinson, R (Cisco Systems), 1997. *RIP Version 2 MD5 Authentication*. Network Working Group, RFC 2082

[9] Malkin, G. (Xylogic, Inc); Minnear, R. (Ipsilon Networks), 1997. RIPng for IPv6. Network Working Group, RFC 2080

[10] Montenegro, G; Kushalnagar, N; Hui, J; Culler, D., 2007. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*

*[11] Thubert, P (Cisco Systems); Brandt, A (Sigma Designs); Hui, J (Arch Rock Corporation); Kesley, R (Ember Corporation); Lewis, P (Stanford University), 2010. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. IETF, RFC 6550, Standards Track, ISSN 2070-1721*

[12] Hui, J; Vasseus, JP (Cisco Systems); Culler, D (UC Berckeley); Manral, V. (Hewlett P.co), 2012. *An IPv6 Routing Header for Source Routes with the Routing Protocol for Low-Power and Lossy Networks (RFC6554-RPL)*. IEEE Comunications Surveys and Tutorials, pp 1-13

*[13] Hui, J; Vasseus, JP (Cisco Systems), 2012. The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams (RFC6553). IEEE Comunications Surveys and Tutorials, pp 1-9*

*[14] Watteyne, Thomas (BSAC, University of California, Berkeley, CA, USA); Vilajosana, Xavier (Universitat Oberta de Catalunya, Barcelona, Spain); Kerkez, Branco (BSAC, University of California, Berkeley, CA, USA); Chraim, Fabien (BSAC, University of California, Berkeley, CA, USA), 2012. OpenWSN: a estandards-based low-power wireless development environment. Transactions on emerging telecommunications technologies. Vol 23, pp 480-493.*