

# **Práctica de diseño de protocolos P2P**

**Redes de Comunicaciones**

Mourad Abbou Aazaz G2.1      Pablo José Rocamora Zamora G2.2

Febrero 24, 2017

# Contents

<b>Introducción.</b>	<b>3</b>
Resumen de los paquetes UDP: peer - tracker que usamos: . . . . .	3
Resumen de los paquetes TCP: peer - peer que usamos: . . . . .	3
<b>Autómatas de ambos protocolos:</b>	<b>5</b>
Peer-peer: Autómata unificado . . . . .	5
Peer-tracker: Cliente y servidor . . . . .	6
<b>Formato de representación de los mensajes</b>	<b>8</b>
Peer-peer . . . . .	8
Peer-tracker . . . . .	12
<b>Conclusiones</b>	<b>20</b>

## Introducción.

Este documento contiene el diseño del protocolo Peer - Tracker y Peer - Peer que hemos llevado a cabo durante el curso académico 2016/17.

### Resumen de los paquetes UDP: peer - tracker que usamos:

#### Formato del mensaje: CONTROL

Tipos que lo usan:

- Type = 1 (ADD\_SEED\_ACK)
- Type = 10 (REMOVE\_SEED\_ACK)

#### Formato del mensaje: FILEINFO

Tipos que lo usan:

- Type = 2 (ADD\_SEED)
- Type = 6 (QUERY\_FILES\_RESPONSE)

#### Formato del mensaje: SEEDINFO

Tipos que lo usan:

- Type = 3 (GET\_SEEDS)
- Type = 4 (SEED\_LIST)

#### Formato del mensaje: SEEDQUERY

Tipos que lo usan:

- Type = 5 (QUERY\_FILES)

#### Formato del mensaje: CHUNKINFO

Tipos que lo usan:

- Type = 7 (QUERY\_CHUNK)
- Type = 8 (QUERY\_CHUNK\_RESPONSE)

#### Formato del mensaje: REMOVE

Tipos que lo usan:

- Type = 9 (REMOVE\_SEED)

### Resumen de los paquetes TCP: peer - peer que usamos:

#### Formato del mensaje: CHUNKQUERY

Tipos que lo usan:

- Type = 1 (GET\_CHUNK)

- Type = 3 (CHUNK)

**Formato del mensaje: CHUNKQUERYRESPONSE**

Tipos que lo usan:

- Type = 2 (GET\_CHUNK\_RESPONSE)
- Type = 4 (CHUNK\_RESPONSE)

## Autómatas de ambos protocolos:

### Peer-peer: Autómata unificado

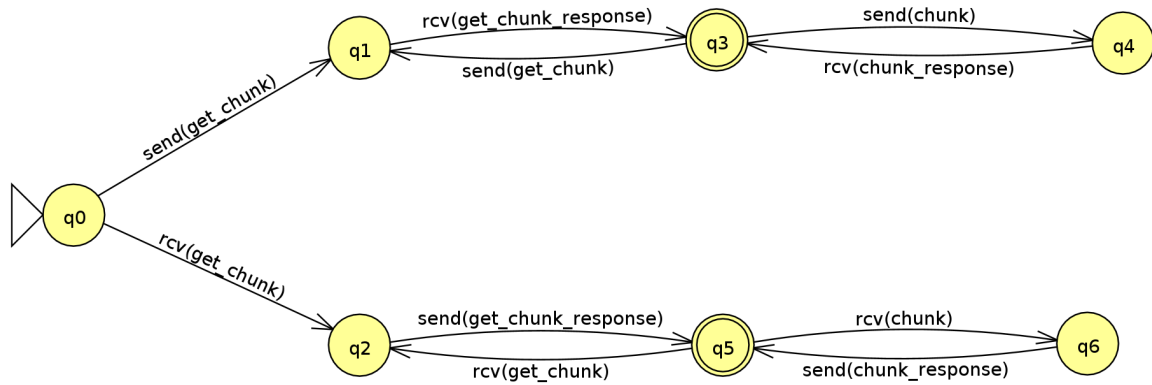


Figure 1: Autómata para el peer (unificada) con TCP

El autómata del Peer para la conexión cliente - servidor entre distintos Peer es bastante simple:

1. El peer hace de cliente y pide la lista de chunks a un fichero, cuando recibe la lista de chunks puede terminar, volver a pedir la lista de chunks para actualizarla o empezar a pedir los datos de los chunks uno a uno.
2. El peer hace de servidor y recibe una solicitud de lista de chunks de un fichero, cuando envía la lista de chunks puede terminar, volver a enviar la lista de chunks para actualizarla o empezar a enviar los datos de los chunks uno a uno.

## Peer-tracker: Cliente y servidor

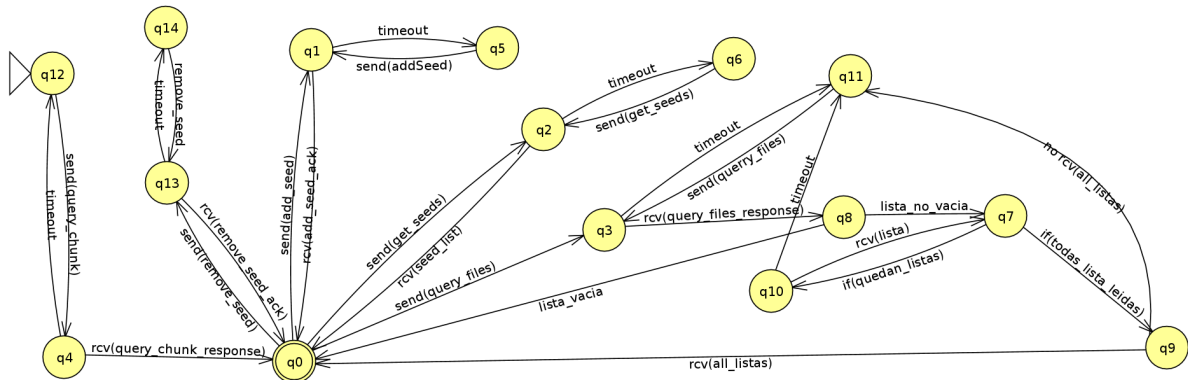


Figure 2: Autómata del cliente con UDP

El autómata del peer para la conexión del peer con el tracker es un poco mas complejo, ya que es el que lleva toda el control de los estados para así poder hacer que el autómata del servidor sea muy simple:

1. En cada comunicación con el Tracker el Peer lo primero que solicita es el tamaño de chunk, una vez que tiene ese dato puede hacer el resto de consultas.
2. Envía un *add\_seed*, si no recibe respuesta y pasa el *timeout* el peer vuelve a retransmitir el mensaje hasta que obtenga respuesta.
3. Envía un *get\_seeds*, si no recibe respuesta y pasa el *timeout* el peer vuelve a retransmitir el mensaje hasta que obtenga respuesta.
4. Envía un *query\_files*, si no recibe respuesta y pasa el *timeout* el peer vuelve a retransmitir el mensaje hasta que obtenga respuesta. Si obtiene respuesta pueden pasar 2 opciones diferentes:
  - Es una lista vacía, este caso es correcto ya que no hay ningún fichero que satisfice los criterios de búsqueda.
  - La lista contiene ficheros, se mete en un bucle en el que esperaremos recibir el mismo numero de paquetes que indica un campo del paquete, si al terminar de recibir todos los paquetes he recibido el numero de paquetes esperado, finalizo correctamente, por el contrario si no he recibido todos los paquetes o hay un *timeout* de algún paquete vuelvo a enviar el paquete *query\_files* para solicitar todos los paquetes de nuevo.
5. Envía un *remove\_seed*, si no recibe respuesta y pasa el *timeout* el peer vuelve a retransmitir el mensaje hasta que obtenga respuesta.

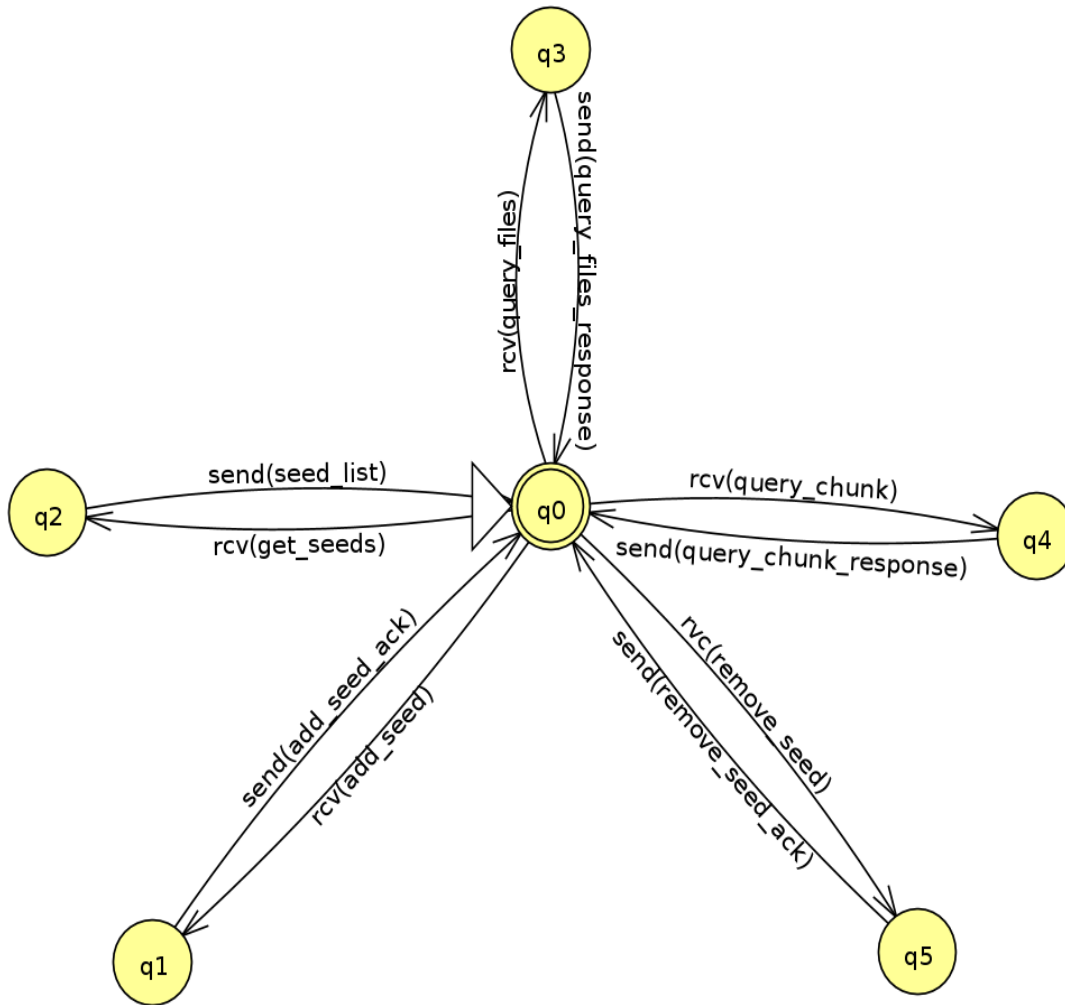


Figure 3: Autómata del servidor con UDP

El autómata de conexión del tracker es muy simple ya que solo se encarga de responder las solicitudes de los clientes, si su paquete no llega será el Peer el que se encargue de volver a pedir la información:

1. Cuando recibe un *add\_seed* responde con un *add\_seed\_ack*.
2. Cuando recibe un *get\_seeds* responde con un *seed\_list*.
3. Cuando recibe un *query\_files* responde con un *query\_files\_response*.
4. Cuando recibe un *query\_chunk* responde con un *query\_chunk\_response*.
5. Cuando recibe un *remove\_seed* responde con un *remove\_seed\_ack*.

## Formato de representación de los mensajes

### Peer-peer

#### Formato del mensaje: **CHUNKQUERY** para el tipo **GET\_CHUNK**

- Type = 1 (GET\_CHUNK)
  - Formato del mensaje: **CHUNKQUERY**.
  - Un Peer solicita al otro Peer la lista de Chunks que tiene de un determinado fichero, que indica a través de su Hash.

Type (1 byte)	Hash (20 bytes)
Num Chunks (longitud variable)	

Figure 4: Formato del mensaje: **CHUNKQUERY**

Información del paquete:

- Type: Siempre sera 1 para indicar que es un **GET\_CHUNK**.
- Hash: Hash del fichero del que deseamos saber los chunks disponibles.
- Num Chunks: No se usa, todo a 0.

#### Formato del mensaje: **CHUNKQUERYRESPONSE** para el tipo **GET\_CHUNK\_RESPONSE**

- Type = 2 (GET\_CHUNK\_RESPONSE)
  - Formato del mensaje: **CHUNKQUERYRESPONSE**.
  - Un Peer informa a otro Peer de la lista de chunks que tiene de un determinado fichero.

Type (1 byte)	Hash (20 bytes)
Num Chunk (long variable)	Chunk (long fijada por tracker)

Figure 5: Formato del mensaje: **CHUNKQUERYRESPONSE**

Información del paquete:

- Type: Siempre sera 2 para indicar que es un **GET\_CHUNK\_RESPONSE**.
- Hash: Hash del fichero del que nos está informando.
- Num Chunks: Numero de chunks que tiene para compartir. El tamaño del campo lo obtenemos con la fórmula: (sale abajo)<sup>1</sup>, Si se tienen todos los chunks de un fichero este campo irá todo a 1 para indicarlo.

---

<sup>1</sup> $\log_2 \frac{\text{TamañoMáximoDeUnfichero}=2^{32}}{\text{TamañoDeChunks}}$



- Chunk: Chunks que tiene el peer, se repite n veces, siendo n: Num Chunks.

### 3.1 Ejemplo.

Un peer A solicita al otro peer B la lista de chunks que tiene de un fichero ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)

1	b9153318862f0f7b5f82c913ecb2117f97c3153e
0000000...	

Figure 6: Ejemplo: GET\_CHUNK

Un peer B informa a otro peer A de la lista de chunks que tiene de un determinado fichero listo para compartir ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)

Le responde que tiene 4 chunks del fichero solicitado (1, 3, 4, 5)

2	b9153318862f0f7b5f82c913ecb2117f97c3153e
4	1
	3
	4
	5

Figure 7: Ejemplo: GET\_CHUNK\_RESPONSE

**Formato del mensaje: CHUNKQUERY para el tipo CHUNK**

- Type = 3 (CHUNK)
  - Formato del mensaje: CHUNKQUERY.
  - Un Peer solicita al otro Peer un Chunk de un fichero específico indicado a través de su Hash.

Type (1 byte)	Hash (20 bytes)
Num Chunks (longitud variable)	

Figure 8: Formato del mensaje: CHUNKQUERY

Información del paquete:

- Type: Siempre será 3 para indicar que es un CHUNK.
- Hash: Hash del fichero del que deseamos obtener un chunk.
- Num Chunks: Número del chunk del fichero que solicita.

#### Formato del mensaje: CHUNKQUERYRESPONSE para el tipo CHUNK\_RESPONSE

- Type = 4 (CHUNK\_RESPONSE)
  - Formato del mensaje: CHUNKQUERYRESPONSE.
  - Un Peer manda a otro Peer el chunk que le ha solicitado.

Type (1 byte)	Hash (20 bytes)
Num Chunk (long variable)	Chunk (long fijada por tracker)

Figure 9: Formato del mensaje: CHUNKQUERYRESPONSE

Información del paquete:

- Type: Siempre será 4 para indicar que es un CHUNK\_RESPONSE.
- Hash: Hash del fichero del que procede el chunk.
- Num Chunk: Número del chunk del que procede el dato.
- Chunk: Dato del chunk, el tamaño se establece acorde al tamaño que nos indica el Tracker.

**Un peer A solicita al otro peer B el chunk 30 del fichero ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)**

3	b9153318862f0f7b5f82c913ecb2117f97c3153e
30	

Figure 10: Ejemplo: CHUNK

**El peer B manda el dato del chunk 30 a el peer A del fichero ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)**

4	b9153318862f0f7b5f82c913ecb2117f97c3153e
30	DATOS

Figure 11: Ejemplo: CHUNK\_RESPONSE

## Peer-tracker

### Formato del mensaje: CONTROL para el tipo ADD\_SEED\_ACK

- Type = 1 (ADD\_SEED\_ACK)
  - Formato del mensaje: CONTROL.
  - El Tracker confirma con ACK que ha recibido la solicitud ADD\_SEED de un Peer para unirse a la red P2P.

```
<message>  
  <operation>add_seed_ack</operation>  
</message>
```

Información del paquete:

- Type: Siempre sera 1 o add\_seed\_ack para indicar que es un ADD\_SEED\_ACK.

### Formato del mensaje: FILEINFO para el tipo ADD\_SEED

- Type = 2 (ADD\_SEED)
  - Formato del mensaje: FILEINFO.
  - Un Peer solicita al Tracker unirse a la red compartiendo una serie de ficheros. Si la lista de ficheros no cabe en un único paquete se partirá en los paquetes que sean necesarios, por cada paquete que se envíe se esperara un ACK y cuando llegue ese ACK se mandará el siguiente paquete. También es usado mas tarde por el Peer para añadir ficheros conforme quiera compartirlos.

```
<message>  
  <operation>add_seed</operation>  
  <port></port>  
  <paquetes></paquetes>  
  <file>  
    <name></name>  
    <size></size>  
    <hash></hash>  
  </file>  
</message>
```

Información del paquete:

- Type: Siempre sera 2 o add\_seed para indicar que es un ADD\_SEED.
- Port: Indica el puerto por el que escuchara el peer.
- Paquetes: Número de ficheros que mandamos al tracker.
- Filename: Nombre del fichero, se repetirá n veces.
- Size: Tamaño del fichero en bytes, se repetirá n veces.
- Hash: Hash del fichero, se repetirá n veces.

### Formato del mensaje: SEEDINFO para el tipo GET\_SEEDS

- Type = 3 (GET\_SEEDS)
  - Formato del mensaje: SEEDINFO.
  - Un Peer solicita al Tracker la lista de semillas para un fichero, para indicar que fichero es usara el Hash del fichero.

```
<message>
  <operation>get_seeds</operation>
  <hash></hash>
</message>
```

Información del paquete:

- Type: Siempre sera 3 o get\_seeds para indicar que es un GET\_SEEDS.
- Hash: Hash del fichero del que deseamos obtener los Seeds.

### Formato del mensaje: SEEDINFO para el tipo SEED\_LIST

- Type = 4 (SEED\_LIST)
  - Formato del mensaje: SEEDINFO.
  - Un Tracker informa al Peer de la lista de semillas del fichero solicitado a través de su Hash.

```
<message>
  <operation>seed_list</operation>
  <hash></hash>
  <size></size>
  <seeds>
    <ip></ip>
    <port></port>
  </seeds>
</message>
```

Información del paquete:

- Type: Siempre sera 4 o seed\_list para indicar que es un SEED\_LIST.
- Hash: Hash del fichero del que obtenemos los Seeds.
- Size: Tamaño del fichero en bytes.
- IP: IP del peer que tiene trozos del fichero, se repetirá n veces.
- Port: Puerto que tiene a la escucha el peer que tiene trozos del fichero, se repetirá n veces.

### 1.1. Usando mensajes multiformato un peer (155.54.2.3) se agrega como seed en el puerto 4533, y con los ficheros:

- ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)
- android-studio.zip (hash af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d , tamaño 380.943.097 bytes).

**El peer envía al tracker un mensaje FILEINFO tipo ADD\_SEED**

2	4533	2
15	ubuntu14.04.iso	
1024572864	b9153318862f0f7b5f82c913ecb2117f97c3153e	
18	android-studio.zip	
380943097	af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d	

Figure 12: Ejemplo: ADD\_SEED

El tracker responde con un mensaje de CONTROL tipo ADD\_SEED\_ACK

1

Figure 13: Ejemplo: ADD\_SEED\_ACK

## 1.2. Usando un lenguaje de marcas especificar la comunicación del apartado 1.1

El peer envía al tracker un mensaje FILEINFO tipo ADD\_SEED

```
<message>
  <operation>add_seed</operation>
  <port>4533</port>
  <paquetes>1</paquetes>
  <file>
    <name>ubuntu14.04.iso</name>
    <size>1024572864</size>
    <hash>b9153318862f0f7b5f82c913ecb2117f97c3153e</hash>
  </file>
  <file>
    <name>android-studio.zip</name>
    <size>380943097</size>
    <hash>af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d</hash>
  </file>
</message>
```

El tracker responde con un mensaje de CONTROL tipo ADD\_SEED\_ACK

```
<message>
  <operation>add_seed_ack</operation>
</message>
```

### Formato del mensaje: SEEDQUERY para el tipo QUERY\_FILES

- Type = 5 (QUERY\_FILES)
  - Formato del mensaje: SEEDQUERY.
  - Un Peer solicita al Tracker la lista de ficheros que coinciden con su patrón de búsqueda. Podemos buscar por tamaño, nombre o ambas, para tamaño tenemos varios operadores aritméticos.

```
<message>
  <operation>query_files</operation>
  <op></op>
  <file>
    <size></size>
    <name></name>
  </file>
</message>
```

Información del paquete:

- Type: Siempre será 5 o query\_files para indicar que es un QUERY\_FILES.
- Op: Operador de condición para tamaño: *(podemos usar operadores bash: gt, lt, ge, etc)*
  - 0: >
  - 1: >=
  - 2: <
  - 3: <=
  - 4: Aprox (10Mb) (EXTRA)
- Size: Tamaño del fichero que estamos buscando en bytes, si no lo usamos todo a 0.
- Filename: nombre del fichero que buscamos, si no lo usamos todo a 0.

### Formato del mensaje: FILEINFO para el tipo QUERY\_FILES\_RESPONSE

- Type = 6 (QUERY\_FILES\_RESPONSE)
  - Formato del mensaje: FILEINFO.
  - El tracker lista los ficheros que coinciden con su patrón de búsqueda. Si la lista de ficheros no cabe en un único paquete se partirá en los paquetes que sean necesarios, y se enviarán todos, el Peer tendrá que comprobar si se han recibido todos los paquetes, sino se han recibido volverá a hacer la solicitud.

```
<message>
  <operation>query_files_response</operation>
  <paquetes></paquetes>
  <file>
    <name></name>
    <size></size>
    <hash></hash>
  </file>
</message>
```

Información del paquete:

- Type: Siempre será 6 o query\_files\_response para indicar que es un QUERY\_FILES\_RESPONSE.
- Paquetes: Número de paquetes que se van a enviar, normalmente será 1.
- Filename: nombre del fichero, se repetirá n veces.
- Size: Tamaño del fichero en bytes, se repetirá n veces.
- Hash: Hash del fichero, se repetirá n veces.

#### Formato del mensaje: **CHUNKINFO** para el tipo **QUERY\_CHUNK**

- Type = 7 (QUERY\_CHUNK)
  - Formato del mensaje: **CHUNKINFO**.
  - El Peer pregunta al Tracker cuál es el tamaño de los chunks. Este mensaje será el primero que se mande para iniciar la comunicación con el Tracker.

```
<message>
  <operation>query_chunk</operation>
</message>
```

Información del paquete:

- Type: Siempre será 7 para indicar que es un QUERY\_CHUNK.

#### Formato del mensaje: **CHUNKINFO** para el tipo **QUERY\_CHUNK\_RESPONSE**

- Type = 8 (QUERY\_CHUNK\_RESPONSE)
  - Formato del mensaje: **CHUNKINFO**.
  - El Tracker responde al Peer cuál es el tamaño de los chunks.

```
<message>
  <operation>query_chunk_response</operation>
  <size></size>
</message>
```

Información del paquete:

- Type: Siempre será 8 para indicar que es un QUERY\_CHUNK\_RESPONSE.
- Tamaño Chunk: Indica el tamaño del chunk.

#### Formato del mensaje: **REMOVE** para el tipo **REMOVE\_SEED**

- Type = 9 (REMOVE\_SEED)
  - Formato del mensaje: **REMOVE**.
  - El Peer informa al Tracker de que quiere hacer una solicitud de borrado, hay 2 opciones:
    - \* Darse de baja como peer: si queremos dejar de ser un Peer activo pondremos el Hash todo a 1.
    - \* Dar de baja un fichero: Si solo queremos borrar un fichero de la lista de activos pondremos el Hash del fichero.

```
<message>
  <operation>remove_seed</operation>
  <port></port>
  <hash></hash>
</message>
```



Información del paquete:

- Type: Siempre sera 9 o remove\_seed para indicar que es un REMOVE\_SEED.
- Port: Indica el puerto del peer.
- Hash: Hash del fichero al que damos de baja o todo a 1.

#### Formato del mensaje: CONTROL para el tipo REMOVE\_SEED\_ACK

- Type = 10 (REMOVE\_SEED\_ACK)
  - Formato del mensaje: CONTROL.
  - El Tracker confirma que ha recibido la solicitud de un Peer para borrar al Peer o a un fichero.

```
<message>  
  <operation>remove_seed_ack</operation>  
</message>
```

Información del paquete:

- Type: Siempre será 10 o remove\_seed\_ack para indicar que es un REMOVE\_SEED\_ACK.

#### 2.1. Usando mensajes multiformato el peer solicita un QUERY\_FILES al tracker y éste responde con la lista de archivos correspondiente.

- ubuntu14.04.iso (hash b9153318862f0f7b5f82c913ecb2117f97c3153e, tamaño 1.024.572.864 bytes)
- android-studio.zip (hash af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d , tamaño 380.943.097 bytes).

El peer envía al track un mensaje SEED\_QUERY\_FILES tipo QUERY\_FILES para buscar los ficheros de tamaño superior o igual a 250000 bytes

5	1	250000
000000000000000000	000000000....	

Figure 14: Ejemplo: QUERY\_FILES

El tracker le responde con un mensaje FILEINFO tipo QUERY\_FILES\_RESPONSE con los ficheros de tamaño superior a 250000 bytes

6	0000000000000000	0000000000000000	2
15	ubuntu14.04.iso		
1024572864	b9153318862f0f7b5f82c913ecb2117f97c3153e		
18	android-studio.zip		
380943097	af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d		

Figure 15: Ejemplo: QUERY\_FILES\_RESPONSE

## 2.2. Usando lenguaje de marcas especificar la comunicación del apartado 2.1.

Peer informa a tracker con un mensaje SEEDQUERY:

```
<message>
  <operation>query_files</operation>
  <op>1</op> <!--podemos usar operadores bash: gt, lt, ge, etc-->
  <file>
    <size>250000</size>
    <name></name>
  </file>
</message>
```

El tracker responde con un mensaje FILEINFO:

```
<message>
  <operation>query_files_response</operation>
  <num_seq></num_seq>
  <port></port>
  <file>
    <name>ubuntu14.04.iso</name>
    <size>1024572864</size>
    <hash>b9153318862f0f7b5f82c913ecb2117f97c3153e</hash>
  </file>
  <file>
    <name>android-studio.zip</name>
    <size>380943097</size>
    <hash>af09cc0a33340d8daccdf3cbfefdc9ab45b97b5d</hash>
  </file>
</message>
```

El Peer pregunta al Tracker el tamaño de los chunks



## Conclusiones

El diseño esta completo, aunque se le podrían añadir algunas mejoras en los paquetes para aumentar la funcionalidad del tracker y de los peers.