Práctica de diseño de protocolos P2P

Redes de Comunicaciones

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

Febrero 24, 2017

Contents

Introducción.

Este documento contiene el diseño del protocolo Peer - Tracker y Peer - Peer que hemos llevado acabo 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

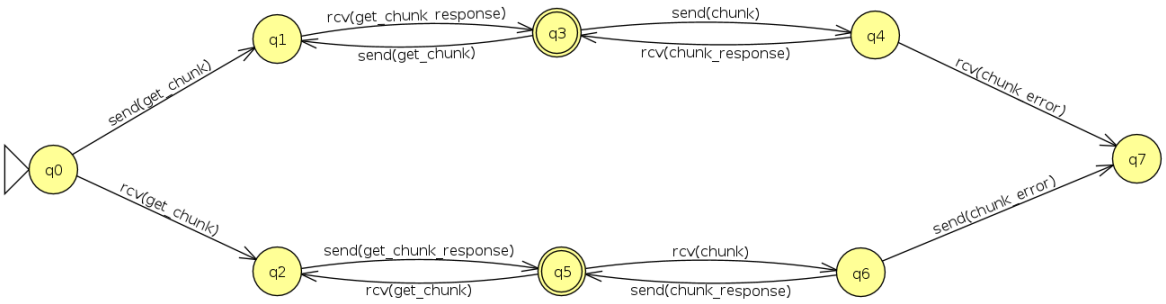

Figure 1: Autómata para el peer (unificada) con TCP

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

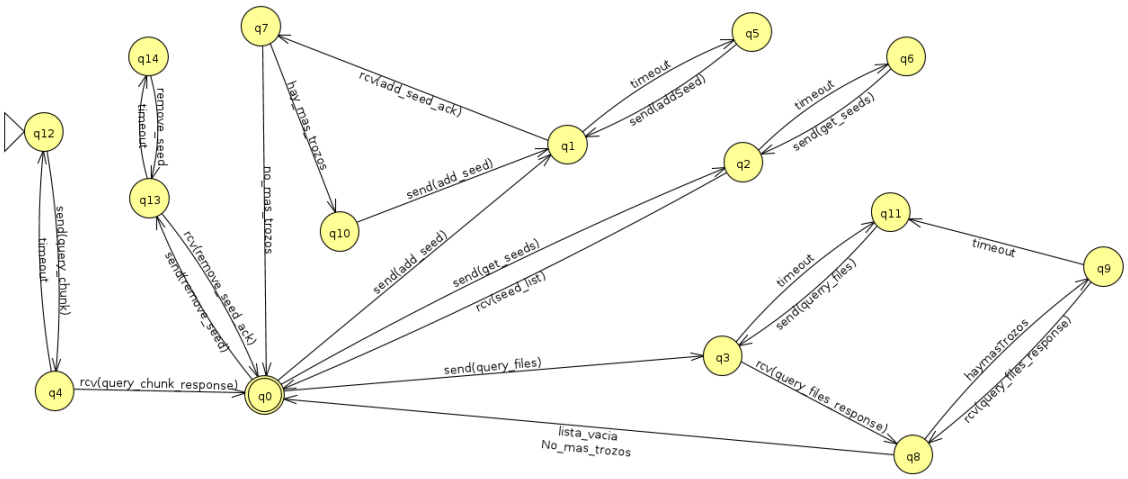

Figure 2: Autómata del cliente con UDP

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 satisface 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.

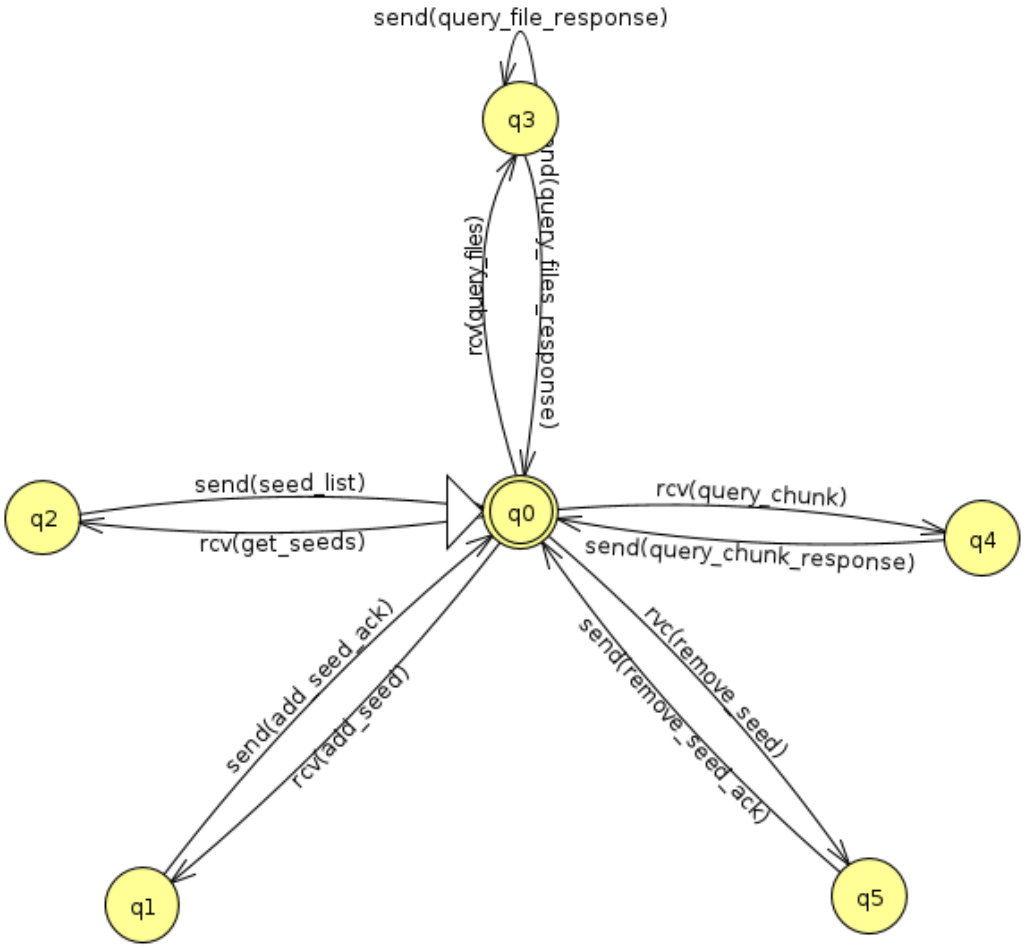

Figure 3: Autómata del servidor con UDP

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 sera 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.

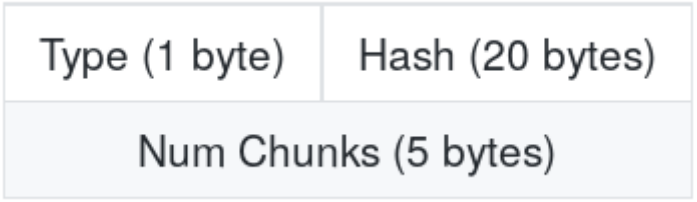

Figure 4: Formato del mensaje: CHUNKQUERY

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.

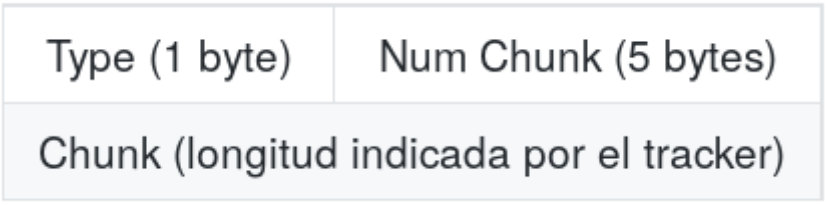

Figure 5: Formato del mensaje: CHUNKQUERYRESPONSE

Información del paquete:

- Type: Siempre sera 2 para indicar que es un GET_CHUNK_RESPONSE.
- Num Chunks: Numero de chunks que tiene para compartir. Si se tienen todos los chunks de un fichero este campo irá todo a 1 para indicarlo.
- Chunk: Chunks que tiene el peer, se repite n veces, siendo n: Num Chunks.

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.

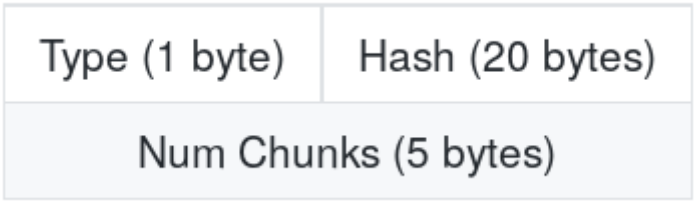

Figure 6: Formato del mensaje: CHUNKQUERY

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.

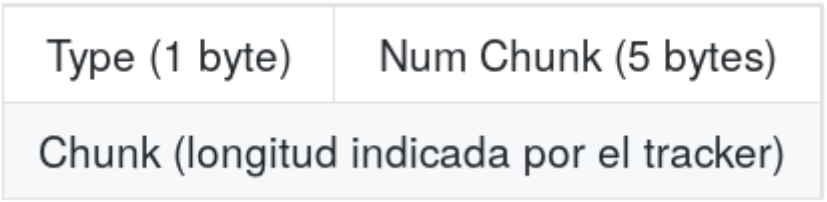

Figure 7: Formato del mensaje: CHUNKQUERYRESPONSE

Información del paquete:

- Type: Siempre sera 4 para indicar que es un CHUNK_RESPONSE.
- 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.

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>
    <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.
- 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>
    <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.
- 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.

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>
    <file>
        <op></op>
        <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 unico 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><!--si se da de baja el peer esta campo no aparecera-->
</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 no lo ponemos si nos damos de baja como peer

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.

Errores que pueden surgir durante las comunicaciones:

1. Un peer se desconecte sin avisar al tracker: en ese caso el peer que detecte al peer desconectado deberá informar al tracker para que le dé de baja.
2. Un peer deje de tener un fichero que estaba compartiendo: en este caso tendrá que mandar al tracker un *remove* del fichero que ya no tiene.

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.