CIENCIA DE LA COMPUTACIÓN

REDES Y COMUNICACIÓN

QUANTUM NETWORK

Alumnos: José David Mamani Vilca

Semestre: VIII

Mayo del 2019

"Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

CIENCIA DE LA COMPUTACIÓN

REDES Y COMUNICACIÓN

PROTOCOLO PEER TO PEER

Alumnos: José David Mamani Vilca

Percy Maldonado Quispe

Semestre: VIII

Mayo del 2019

"Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo"

Protocolo Peer to Peer

- Basado en la implementación descrita por BiTorrent

Estructura de los mensajes:

Para nuestro protocolo los mensajes tendrán la siguiente estructura:

Segmentos de 256 Kbs.

C 0 012 message
[C] [S] [Si] [M]

- **C (Char):** Representa el comando enviado. Indica el comportamiento y las acciones que tendrá que realizar el receptor del mensaje
- **S (Bool):** Relacionado al tamaño del mensaje. Indica la culminación del envío de datos. Por ejemplo: Un mensaje muy grande será segmentado en muchos trozos cada uno con el valor de **S** dispuesto como verdadero pues no indican la finalización del mensaje. En caso se llegué al último bloque de la división, el valor de **S** será *false*, por lo que el protocolo detectará el fin del mensaje enviado.
- **Si (Int):** Compuesto por los siguientes tres datos a partir de **S.** Indican el tamaño restante del mensaje, considerando únicamente su contenido. Es muy útil, especialmente durante el último bloque enviado.
- **M (String):** Representa el contenido del mensaje.

No existe diferencia entre los mensajes del tipo *request* y los del tipo *response*. Básicamente todos los mensajes poseen la misma estructura por lo que no existe diferencia alguna entre ellos.

Lógica de los mensajes:

En nuestro protocolo se destacan dos agentes activos en todo momento: El Tracker y los Peers.

Tracker: Cumple un rol similar al de un servidor con la diferencia de que no participa en los envíos de datos. Se encarga de administrar a los Peers activos en la Red de tal manera que estos sepan de la existencia de los otros.

- **Register_peer():** Útil cuando un Peer es totalmente nuevo en el sistema. Se guarda su dirección IP, se le asigna una sesión y un estado de activo.
- **Login():** Similar a la funcionalidad previa. Si un peer ya es conocido dentro de la red, solo es necesario darle el estado de activo.
- **Get_list_of_peers():** Retorna la totalidad de peers registrados en la red. Tanto si estos están activos como inactivos.

- **Get_list_of_random_peers():** Cada vez que un peer se registre o inicie sesión, se debe de asignarle un conjunto de peers vecinos con los cuáles intercambiará información. Si bien dicha asignación debe realizarse de manera tal que la comunicación entre peers sea óptima, para el caso de nuestro protocolo se hará de forma aleatoria y claro, seleccionando solo a los Peers activos hasta ese instante.
- **Peer_join():** Función imbuida en Register y Login. Activa el estado del peer para que pueda asociarse con otros peers.
- **Peer_left():** El *Tracker* mantiene una lista de todos los peers activos recibiendo confirmaciones periódicas de estos. Si durante un tiempo determinado un peer deja de enviar sus confirmaciones entonces será automáticamente deshabilitado dentro de la Red hasta que este vuelva a iniciar sesión.
- **Get_list_of_files():** Retorna una lista de todos los archivos disponibles en la Red. Cada uno de estos archivos poseerá un Id único y estará compuesto por una cantidad determinada de *Chunks* o segmentos.
- **Logout():** Equivalente a Peer_left con la diferencia que el usuario voluntariamente deja de estar activo en la red. Hecho esto, ningún peer podrá contactarlo hasta que inicie sesión nuevamente.

Peer: Cumplen un rol de servidor /usuario. Son los encargados de esparcir la información pero al mismo tiempo de recepcionarla.

- **Login():** Petición para acceder a los servicios de la red. Mediante esta función y con la correspondiente respuesta del servidor el Peer puede ser asignado como un Peer activo.
- **Get_list_of_neighbords():** Petición hecha al *Tracker* con la intención de recibir un conjunto de direcciones IPs con las cuáles el Peer actual podrá interactuar e intercambiar datos.
- **Keep_alive():** Función innata, que periódicamente le reporta al *Tracker* que el Peer actual aún se encuentra activo.
- **Get_list_of_chunks():** Petición diseñada para otros Peers. Devuelve un conjunto chunks asociados a un archivo en particular que el Peer objetivo mantiene. Esta funcionalidad es muy útil, pues le permite al Peer actual seleccionar los chunks más raros y darles prioridad para descargarlos.
- **Select_neighbord():** Durante segmentos determinados de tiempo cada Peer dentro la red opta por cambiar de pareja bajo la lógica de buscar al Peer en su vecindario que presente la mejor conexión posible.
- **Download():** Descarga de información solicitada.
- **Upload():** Subida de información.

Diagrama de Interacción:

Peer Actual **Tracker** **Set of Peers in Neighborhood**

1) El Cliente inicia sesión enviando sus datos al Tracker. En caso de ser nuevo procede a registrarse

2) Se acepta la petición del usuario y se le incluye en la lista de Peers activos

3) El Peer nuevo solicita una lista de Peers con los que podrá conectarse. De esta forma se construye un vecindario.

4) El tracker acepta la petición y retorna un conjunto aleatorio de peers activos.

5) El Peer actual solicita una lista con todos los archivos disponibles en la red.

6) Se retorna una lista con los archivos disponibles junto con sus respectivos ID que faciliten su identificación.

7) El Peer actual selecciona una pareja con la cual intercambiar datos. Esta selección se rige por factores que determinan cuán buena es la conexión. Este proceso se repite tras cierto tiempo.

8) El Peer seleccionado acepta la conexión del Peer actual devolviendo todos los chunks que posee.

9) Con la lista de archivos ya en mano, el Peer actual procede a seleccionar los chunks que coincidan con Id del archivo buscado. La selección se hace considerando los chunks menos comunes.

11) Periodicamente el peer actual contacta con el *Tracker* indicando que aún está activo.

10) El peer objetivo devuelve los chunks solicitados.