Adam Mooers

Luke Kledzik

2/19/2017

System and Networks 2

**Protocol Description**  
  
**Terminology:**

**Server**: Common connection point for establishing ring  
 **Client**: Nodes in the token ring, in reference to the server

**Peer**: Nodes in the token ring, in reference to token passing.

**Overview:**  
All messages are sent as a fixed-length buffer. This buffer contains a null-terminated string with whitespace-delimited arguments. Except during ring initialization, a single token is passed between the peers in a round-robin manner. Only the peer with the token can access the message database.

**Connecting to the Server:**  
Clients connect to the server by sending an empty UDP packet to the server. The server saves each client’s info including their port number in arrays. Once the expected number of peers have contacted the server, the server then contacts the clients back and notifies them of their neighbors. This is done by having client[i] send their data to client[(i+1)%numOfClients]. The modulus ensures that the last client in the array can be paired back to the first one and complete the ring. Clients are connected chronologically in the ring i.e. clients forward the token the client which connected to the server after them.

**Token Ring Initialization: (not implemented)**

To determine which peer should start passing the token, each simultaneously sends the following special init. token to its neighbor:

I g <host name of self> <port of self>

When a neighbor receives a packets, it first checks to see if the second argument is “g” or “s”. If s, they compute the hash of the received string and compare it to the hash of the string they sent to their neighbor about themselves. If the the hash of their own token is smaller, they change the second parameter from “g” to “s”, where s signifies that a smaller hash exists. The host then sends the modifed token to its neighbor. If the second argument is already “s”, the packet is immediately forwarded.

If a peer receives a packet with its own hostname and port, it checks the second argument. If it is still “g”, then the peer can conclude that no other peer in the ring had a smaller hash. It then creates a normal token for controlling file access.

**Standard Token Format**

When a token is not entering or leaving the ring, it sends the following standard token:

T

This token is easily parsed for efficiency. This token can be converted to a leave or join token, but should never be converted back to an initialization token. Whenever a peer has control over a “T” token, it has full control over the message log. If the terminal thread is waiting to update the file, the mutex lock it uses it temporarily unlocked allowing the terminal thread to update the message log.

**Exiting the Token Ring: (partially implemented)**

To exit, a peer sends the following special token

T L <desiredClient> <desiredClientPort> <newDestClient> <newDestPort>

T represents that this is a token that is getting passed around. L labels that this is a request to leave the ring. DesiredClient and DesiredClientPort contain the info about the client before the one that is leaving. NewDestClient and NewDestPort contain the info that the DesiredClient needs in order to modify the client info that is uses for its sendto() functions. When a peer receives this message, it checks if the desired client and client match its own client and port. If they do not, the message is forwarded. If they do, the peer’s neighbor is leaving the ring. The peer updates its client and port with newDestClient and newDestPort respectively. The token is changed to a standard token before it it forwarded.

If multiple peers seek to exit the ring simultaneously, the first peer to modify a standard token exits first. Exiting peers do not wait for confirmation that the exit was successful: they exit immediately.

**Joining the Token Ring: (partially implemented)**

To join an existing token ring, the following token is sent to any of the peers in the ring:

T J

The receiving peer then waits to receive the current token. Two cases can happen:

* If the token is a standard T token (not an exit and join request), the peer sends the hostname and port of its current neighbor to the new peer. It does not run any input commands from its user, opting to process them in the next iteration. The peer already in the ring updates its neighbor to be the new peer. The new peer sets its neighbor to the received value. This means the new peer is inserted after the peer it contacted. In the process, the token is implicitly passed to the new peer. The new peer then sends a standard token to its neighbor, completing the process.
* If the token is not a standard T token, it is forwarded to its current neighbor or otherwise handled normally. The peer then continues waiting for a standard token.

If multiple peers attempt to connect to the same peer simultaneously, all but the first will be ignored.