Network System - Integration Project Report

Danilo Toapanta
University of Twente,
P.O. Box 217, 7500AE Enschede
The Netherlands,
d.a.toapantabarahona@student.utwente.nl

1. System overview

This is a description of an ad hoc text message system that allows up to 4 users to communicate to each other. The system allows messages of up to 96 bytes which are sent as a packet of 32 bytes at a time using fragmentation. The system sends messages in 2 parts, a header which is used to identify the nodes, users' ids, and a payload part for the actual text. The system uses multi hop forwarding by giving identifications to all nodes and base on this a forwarding table is created. This ensures nodes far away from the reachable range to communicate using intermediate messages. Finally, the system handles collisions of messages sent at the same time by adding a buffer that puts messages in a queue from which messages are treated by first comes first served manner.

2. Description the system functionalities and/or protocols

2.1. Medium access control

This section takes care of possible collisions between nodes transmitting at the same time. In the sending side, a frame is first stored in a queue data structure named *waitingQueue*. From this queue, frames are transmitted to *senderQueue* when the channel is free. A channel is free when no one else is sending data. This state is signaled by a boolean variable called *state* which only becomes true when the channel is not being used. If the latter is not the case and two or more nodes are trying to send data at the same time, their respective messages are queued in *waitingQueue* and then sent in a FIFO manner.

2.2. Segmentation and reassembly

For a large message to be sent, if the size of the payload is more than 32 bytes fragmentation is used. This is done by storing the data into three different messages and then send them one at the time. Before doing so, padding is considering if needed. Furthermore, to be able to reconstruct the message at the receiver side, after each fragment a *flag* is added. The first fragments of the message the receiver does not print a new line unless the last fragment is received. This make the data stream to be continuous and guarantee a much more understandable message. In the latter case, when the last fragment message is received a dot is added by the algorithm implemented so the receiver can see this flag and know when the data stream ends. Lastly an end of line is implemented, and a new message can be sent or arrive.

2.3. Reliable broadcasting

For this section, the idea was to have an acknowledgment (*ack*) from the receiver that the header part (first message) message was received. This is done by first adding an extra space in the header file that indicates if this is a normal message or an *ack* message. The sender will send the message and enter a for loop that loops *n* number of times to achieve a certain time and only exit after receiving and ack or a timeout occurs (loop is over). If a timeout occurred, a boolean object is used to indicate that the message should be resent., and so on. Due to time constraints, this part was not implemented in the code.

2.4. Multi-hop forwarding

In order for nodes that are far away from each other to communicate, an intermediate node(s) must be used to deliver the messages. This was done by first giving a unique hash depending on the id each node. Namely, the hash takes the first letter of each username, subtract the reminder of a random number and transform this character back to char. The result is a hash for each node which can be used as a unique identifier. This *id* is then saved in each node's forwarding table and send by each node so that everyone in the chat is aware of each other. The mechanism to accomplish this requires a set up phase which consist of 2 test messages prompted by the user.

The identifier of each node is broadcasted so that each node fills its entries in the forwarding table and in this way if a sender wants to communicate with one node that is not in a reachable range it can then send its message to an intermediate node. The intermediate sender then looks in its forwarding table and forward the message to its respective destination.

All above, resembles distance routing protocol. The tuple used in this case was a *destination* variable which holds the *id* of each node and a entry for *nextHop* which hold the id of the node to be used as an intermate step for a message to reach the receiver. The forwarding table is updated by each node and new coming tuples are stored if and only if this information has not been saved it in the forwarding entries.

2.5. Application interface

To provide the user a friendly interface several actions need to be taken. Among the first and most important, the person joining Scuba Chat is required to provide a

username which is then used as an identifier for other nodes to recognize where data is coming from. A second phase is dedicated for setting up connection. Here, the sender side

insert a series of messages which are used to filling up the forwarding tables on each node. This is carried out by flooding messages to all nodes so that at the end each node knows how to reach each one. Finally, a list of available users is displayed together with the respective forwarding table that the node generated while setting up the connection. Now, the user can start chatting.

3. Testing

In order to test and verify that the long messages can be send, a message of size bigger than 32 bytes (letters) is sent and the received message is shown.

To test the functionality of the medium access control, two messages are sent at the same time and both are delivered with one arriving a delay after the first one. It guaranteed arrival but the more the messages the longer the delay, since it is first come first serve system.

To test the functionality of multi hop forwarding, a topology for the nodes is chosen in which nodes A and C are neighbors to node B but not to each other. After the 2 initialization messages of each node, node A can now send a message to node B, and a delay after it is seen by node C which confirms that the forwarding is working.