Persistent and Asynchronous Multicast System

In this project, you will implement a system for supporting persistent asynchronous multicast on top of TCP. This will be based on a coordinator-participant paradigm. You will implement both the multicast coordinator and the participants.

Persistent and Asynchronous Multicast Model

In a multicast communication paradigm, the messages sent to a multicast group are received by all members of the group (including the sender) at the time the message was sent. As we have discussed in the class, asynchronous implies that a sender is not blocked until everyone in the multicast group receives the message -- it is unblocked as soon as it receives an acknowledgement from the coordinator. Persistence guarantee implies that an offline participant, upon reconnection, will receive messages that were sent when it was offline. If a participant is online when a message was multicast, it should get the message immediately (i.e., messages should not be bundled for online participants).

You will implement a slightly modified version of persistence called "temporally-bound persistence". In this model, if a participant is offline more than some specified time threshold (say t_d), upon reconnecting, it will receive messages that were sent in the past t_d seconds. In other words, the messages sent before t_d will be permanently lost to the participant. If a participant is disconnected less than or equal to t_d seconds, it should not lose any messages.

The participants send and receive multicast messages to and from the coordinator (i.e., participants do not directly communicate with each other). The coordinator manages the multicast group, handles all communication and stores messages for persistence.

Participant Details

It is assumed that each participant has a unique ID provided in the configuration file (see below). For simplicity, it is assumed that a participant does not migrate from one host to another. However, it may change its ports upon disconnection and reconnection.

Each participant has at least two threads (you can have more threads depending on your design). One thread (thread-A) is for receiving user commands. The second thread (thread-B) waits on a specified port for receiving multicast messages from the coordinator (when the participant is online). All the multicast messages received by the thread must be logged in the message log file (see below). When the participant is offline, thread-B should relinquish the port (the thread may be killed or it just may be dormant as per your design). All the messages received by a participant should be logged in a file in the order they were received.

The participant program will have one command-line parameter, namely, the configuration file name. The configuration file will have the following format. The first line indicates the ID of the participant. The second line indicates the message log file name. The third line indicates the IP

address and port number of the coordinator. An example participant configuration file can be found at here (http://www.cs.uga.edu/~laks/DCS-2016-Sp/pp3/PP3-participant-conf.txt).

The participant should support the following user commands (format of the command is provided in parenthesis).

Register (`register [portnumber]`): Participant has to register with the coordinator specifying its ID, IP address and port number where its thread-B will receive multicast messages (thread-B has to be operational before sending the message to the coordinator). Upon successful registration, the participant is a member of the multicast group and will begin receiving messages.

Deregister (`deregister`): Participant indicates to the coordinator that it is no longer belongs to the multicast group. Please note that this is different than being disconnected. A participant that deregisters, may register again. But it will not get any messages that were sent since its deregistration (i.e., it will be treated as a new entrant). Thread-B will relinquish the port and may become dormant or die.

Disconnect (`disconnect`): Participant indicates to the coordinator that it is temporarily going offline. The coordinator will have to send it messages sent during disconnection (subject to temporal constraint). Thread-B will relinquish the port and may become dormant or die.

Reconnect (`reconnect [portnumber]`): Participant indicates to the coordinator that it is online and it will specify the IP address and port number where its thread-B will receive multicast messages (thread-B has to be operational before sending the message to the coordinator).

Multicast Send (`msend [message]`): Multicast [message] to all current members. Note that [message] is an alpha-numeric string (e.g., UGACSRocks). The participant sends the message to the coordinator and unblocks after an acknowledgement is received.

Coordinator Details

As mentioned before, the coordinator manages the multicast group, handles all communication and stores messages for persistence. For simplicity, it is assumed that there is only one multicast group and all registered participants belong to the same group.

The coordinator program will have one command-line parameter, namely, the configuration file name. The configuration file will have the following format. The first line indicates the port number where the coordinator will wait for incoming messages from participants. The second line indicates the persistence time threshold (t_d) in seconds. An example coordinator configuration file can be found at here (http://www.cs.uga.edu/~laks/DCS-2016-Sp/pp3/PP3-coordinator-conf.txt).

When the coordinator starts up, the multicast group is assumed to be empty. Participants join the group through register messages. Theoretically, you can have as many threads as you want for the coordinator. But remember that creating large number of threads will create a

performance bottleneck. Therefore limiting threads to a small number (< 10) is a good idea. When the coordinator gets a message from a participant, it first parses the message, acknowledges the receipt of the message, closes the connection and then performs the requested action. For example, if it receives a "register" message, it will add the participant to the member list. If it receives a multicast message, it will send the message to the members that are currently online. If any members are offline, it will store the message for providing persistence.

Points to Note

1. Participants should not miss any messages that satisfy the temporal persistence condition.
2. Participants should not receive messages that violate the temporal restrictions (i.e., messages that were sent before t_d seconds).
3. Participants should not receive duplicate messages.
4. Participants should not receive messages sent during the period when they had deregistered.
5. You may assume that the coordinator does not fail.
6. Many design details (message formats, data structures, etc.) have been deliberately omitted. Coming up with a good design is a very important part of the project.

What to Submit:

A Zipf folder containing:

1. Your code for the coordinator and the participants
2. A readme file containing (a) names of the students in the project group; (b) any special compilation or execution instructions; and (c) the following statement – if it is true – "This project was done in its entirety by <Project group member names>. We hereby state that we have not received unauthorized help of any form". If this statement is not true, you should talk to Dr. Ramaswamy (not TA) before submission.
3. All submissions will be done on ELC.