GMSHealthMonitor Message Sequence Diagram

GMSHealthMonitor makes sure that each member in the distributed system is alive and communicating to this member. To make sure that, we create the ring of members based on current view. On this ring, each member makes sure that the next member in the ring (its neighbor) is communicating with it. For that we record last message timestamp from its neighbor. And if it sees its neighbor has not communicated in last period(member-timeout) then we check whether its neighbor is still alive or not. Based on that we informed probable coordinators to remove its neighbor from the view, if its neighbor is not alive.

This failure detector is effective at detecting situations where Java threads, in general, are not receiving adequate time slices. As the number of CPU-intensive threads increases, at some threshold, the thread scheduler (on Linux, for instance, this will be the operating system's thread scheduler) doesn't have enough CPUs to spread around to all the threads. This results in Thread.sleep() returning later and later. It also results in ready-to-run threads receiving shorter, or less frequent time slices.

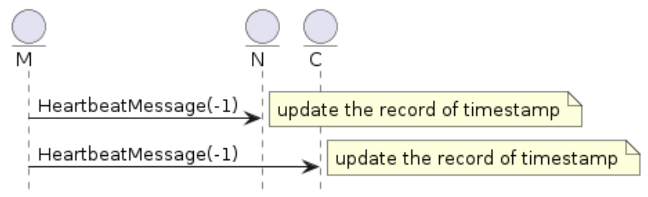
A CPU can fail in other ways, for instance, a CPU can fail to give correct answers for e.g. integer addition. The broad category of problems that can lead a process to give wrong answers is known as byzantine failure. The Geode health monitor does not address byzantine failures at all. There is no attempt to monitor the correct functioning of the CPU for instance. If the health monitor threads function then we assume other threads in the same process function correctly. Byzantine failures are currently out of scope for this discussion.

Often, when the health monitor does its job (and forces a member out of the distributed system), the question arises: did the health monitor make a mistake? See Troubleshooting CPU for information that will help you determine whether CPU was actually over-taxed (saturated) on your system, or alternately, if you might be encountering a bug in the health monitor.

HeartbeatMessage

Each Member periodically sends HeartbeatMesage (UDP) to the coordinator and two other members in the distributed system. The two members selected are most likely monitoring the sender of the HeartbeatMessage. Upon receiving the HearbeatMessage, the receiving member updates its record of receiving timestamp associated with the sender. The requestID of HeartbeatMessage is set to -1 so that the receiver does not look for a thread waiting for the message. HeartbeatMessages are themselves used as replies to HeartbeatRequestMessages(see next section).

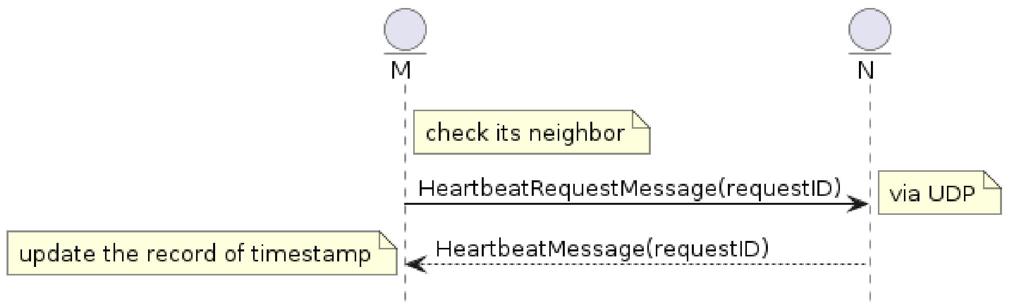
This diagram shows a member (M) sending HeartbeatMessage to all the other members (N, C) in the distributed system


M
N
C
HeartbeatMessage(-1)
update the record of timestamp
HeartbeatMessage(-1)
update the record of timestamp

HeartbeatRequestMessage

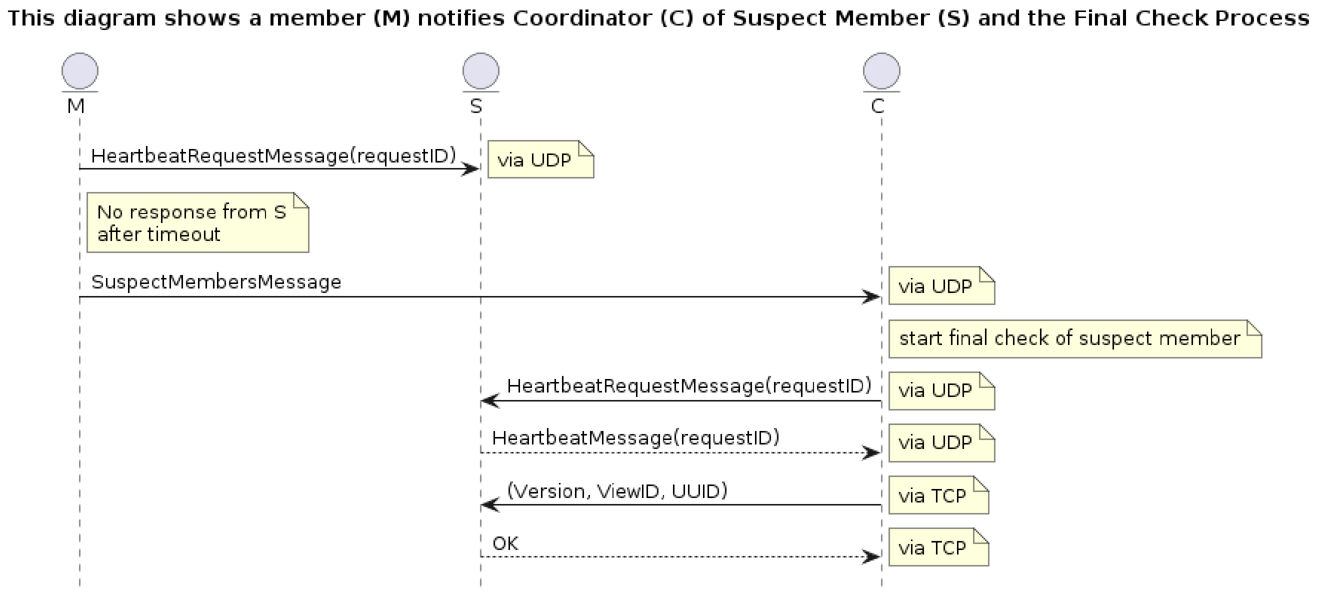
For each member, another monitoring thread is checking the timestamp of its neighbor's HeartbeatMessage or any other messages received from its neighbor. Note that any message received from another member counts as a heartbeat. See GMSHealthMonitor.contactedBy(). If the member has not received the HeartbeatMessage from its neighbor for more than (the membership timeout / 2) which would usually be 2.5s. It will start sending HeartbeatRequestMessage to its neighbor to check whether its neighbor is still alive. If its neighbor is still alive, it will respond with HeartbeatMessage with requestID set to the same as HeartbeatRequestMessage. Upon receiving the response from its neighbor, the member will update its record of timestamp accordingly. If no response for HeartbeatRequestMessage is received from its neighbor, it will check its timestamp record again. It is possible that the neighbor has sent another HeartbeatMessage during the waiting period.

This diagram shows a member (M) using a HeartbeatRequestMessage to check its Neighbor (N)

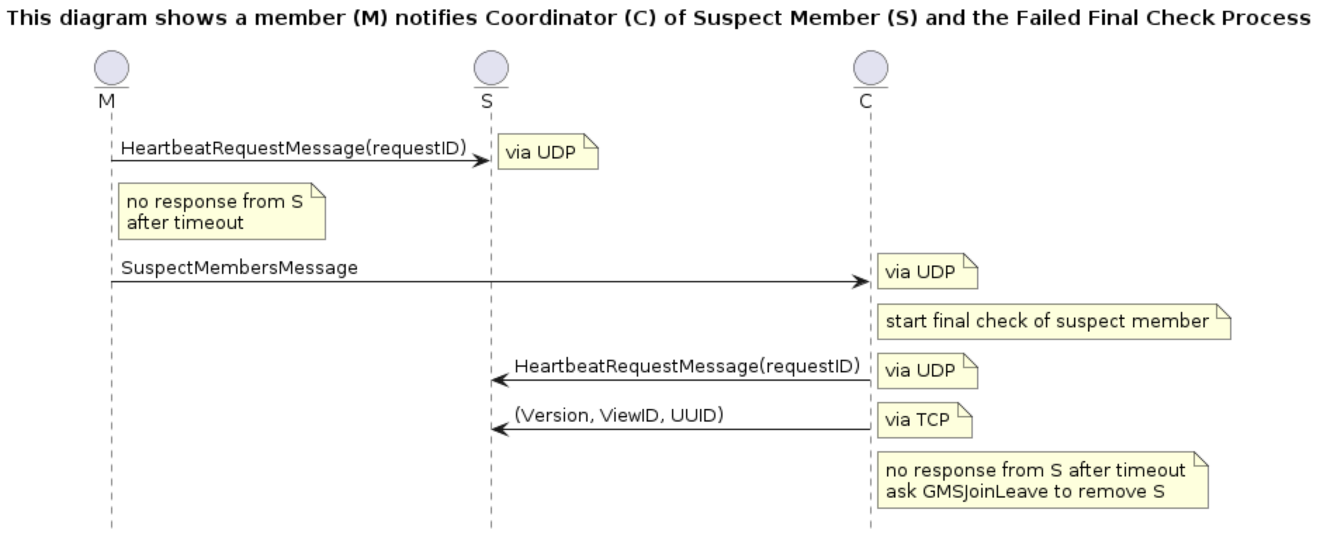

M
N
check its neighbor
HeartbeatRequestMessage(requestID)
via UDP
update the record of timestamp
HeartbeatMessage(requestID)

SuspectMembersMessage and Final Check

If there is still no response from its neighbor, and no update on the timestamp, the member will initiate a suspicion on its neighbor. Basically, the member will then send SuspectMemberMessage which includes a list of SuspectRequests to a list of recipients. Depending on the size of the view, the list of recipients may contain all the members in the view, if the view size is less than or equal to 4. If the view size is larger than 4, the list of recipients may have up to 7 members, which include 5 members preferred to be coordinator, the sender itself and a random member. How the recipient reacts to the SuspectMemberMessage depends on whether it is the coordinator or not. If it is the coordinator, it will start final check on the suspect member. To do final check, the coordinator sends HeartbeatRequestMessage to the suspect member, expecting a response from suspect member. At the same time, the coordinator also starts TCP final check, which initiates a TCP connection to the suspect member and exchanges the messages if the suspect member is still alive.


This diagram shows a member (M) notifies Coordinator (C) of Suspect Member (S) and the Final Check Process
M
S
C
HeartbeatRequestMessage(requestID)
via UDP
No response from S after timeout
SuspectMembersMessage
via UDP
start final check of suspect member
HeartbeatRequestMessage(requestID)
via UDP
HeartbeatMessage(requestID)
via UDP
(Version, ViewID, UUID)
via TCP
OK
via TCP

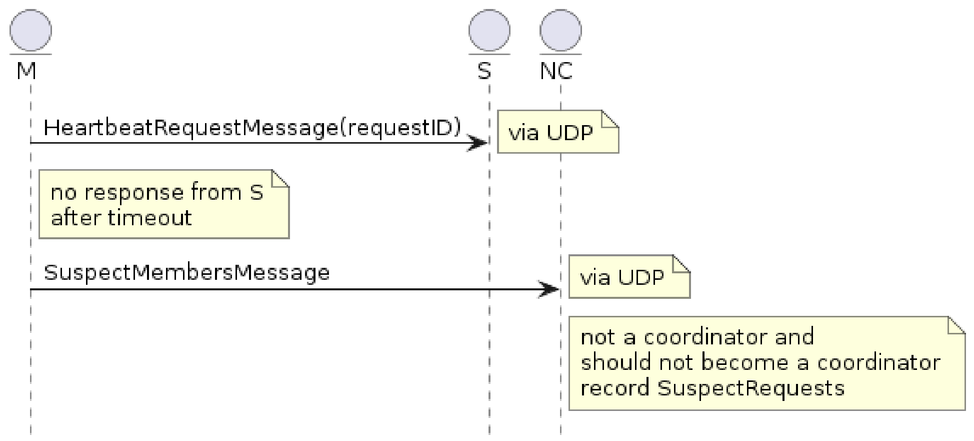
If final check failed, the coordinator then asks GMSJoinLeave to remove the suspect member from the system.


This diagram shows a member (M) notifies Coordinator (C) of Suspect Member (S) and the Failed Final Check Process
M
S
C
HeartbeatRequestMessage(requestID)
via UDP
no response from S after timeout
SuspectMembersMessage
via UDP
start final check of suspect member
HeartbeatRequestMessage(requestID)
via UDP
(Version, ViewID, UUID)
via TCP
no response from S after timeout ask GMSJoinLeave to remove S

If the recipient of SuspectMembersMessage is not the coordinator, it checks to see if it might become the coordinator and so should initiate a final check. For a recipient to become the coordinator, it first depends on a few settings: whether network partition detection is enabled, or whether authentication is enabled. If either one of these is enabled, and the current coordinator is a suspect member, and any member to the left of the recipient in current view is a suspect member, the recipient might become the coordinator and will initiate a final check.

If the recipient is not the coordinator and the above checks fail, it records the SuspectRequest for subsequent use, which helps determine whether it should become the coordinator in the future.

This diagram shows a member (M) notifies non-coordinator member (NC) of Suspect Member (S)


M
S
NC
HeartbeatRequestMessage(requestID)
via UDP
no response from S
after timeout
SuspectMembersMessage
via UDP
not a coordinator and
should not become a coordinator
record SuspectRequests