Reconnect/Disconnect/Heartbeat

Overview

Logos network is a distributed network, which is inherently prone to network, hardware, or software failures. It is therefore imperative from the quality of service and network's throughput prospective that any failure is remedied as soon as it is detected by the core software. This document addresses explicit failures of the TCP/IP connection and implicit failures inferred from a period of extended inactivity by a remote delegate.

Execution Concept

A delegate communicates with remote delegates by exchanging consensus messages over TCP/IP connections via read/write function calls executed on the connected socket. A TCP/IP connection failure will result in the socket call returning with an error indicating the nature of the failure. This type of failure can be detected by the core software which then can attempt reconnecting to the remote delegate. It is possible that other types of failures in the network or the remote delegate do not result in the TCP/IP connection failure excluding the remote delegate from participating in the consensus process. To remedy this type of failures, the core software maintains the time of the last message received from a remote delegate. If the time since the last communication exceeds a threshold value then the TCP/IP connection to the remote delegate is closed and the core software attempts reconnecting to the remote delegate.

Class Diagrams

The following classes ConsensusNetIO, ConsensusNetIOManager, NetIOAssembler, ConsensusManager are enhanced with new data members and methods to support reconnect/heartbeat functionality as shown on Figure 1. In addition a new message type - HeartBeat is added, which has is_request set to true for reqeuest and false for response. Note that only new data members and methods are provided.

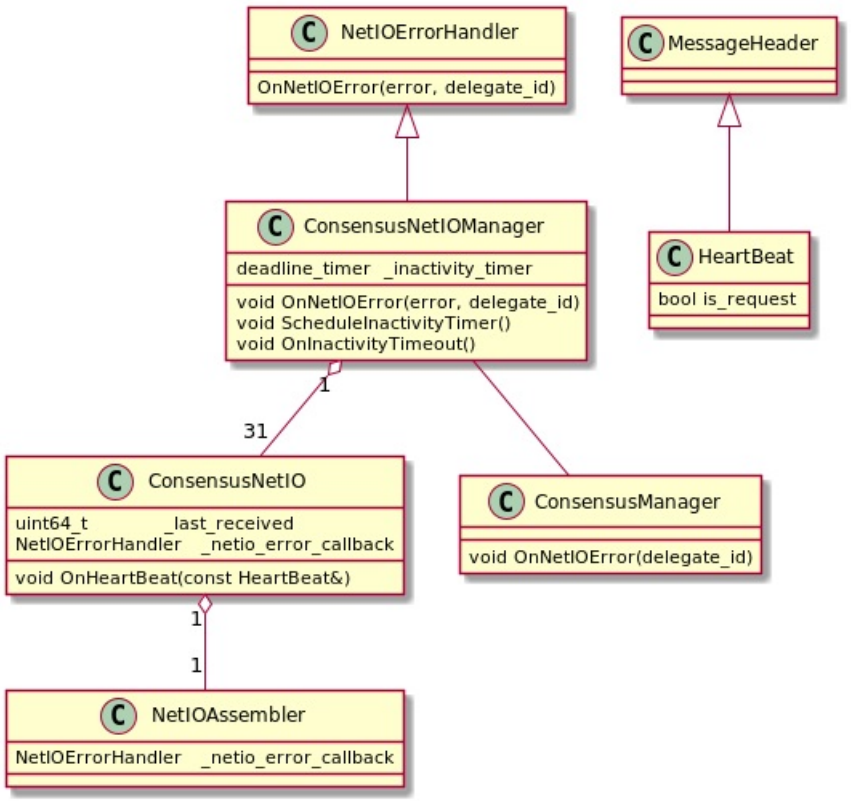

Figure 1. Enhanced Interfaces.

ConsensusNetIOManager class implements OnNetIOError() method of NetIOErrorHandler abstract class which is passed to Constructors of ConsensusNetIO and NetIOAssembler and stored as _netio_error_callback data member. These calls are used for the propagation of the socket error to ConsensusManager and ConsensusNetIOManager. ConsensusManager implements OnNetIOError() to handle destruction of ConsensusConnection.

ConsensusNetIO data member _last_received is a time-stamp of the last received message from a remote delegate. ConsensusNetIOManager schedules periodic _inactivity_timer with ScheduleInactivityTimer() callback. OnInactivityTimeout() method is called on the timer timeout. If the time since the last message exceeds 80 seconds then the reconnection logic is envoked, otherwise a HeartBeat message is sent to the remote delegate. HeartBeat message is handled by OnHeartBeat() call in the ConsensusNetIO class.

Sequence Diagrams

Figure 2 shows a sequence of calls to reconnect a delegate to a remote delegate when a socket error occurs on either read or write calls in NetIOAssembler or ConsensusNetIO. A socket error can be detected by examining the error value returned by the read or write calls. If the error is not null then OnNetIOError() is propagated to ConsensusManager which destroys respective instance of ConsensusConnection. Then ConsensusNetIOManager destroys respective instance of ConsensusNetIO. Finally, ConsensusNetIOManager creates a new instance of ConsensusNetIO (the sequence of calls to follow is part of the current core software), which in turn initiates connection to the remote delegate if the delegate is TCP/IP client with regards to the remote delegate. When connection is accepted, BindIOChannel() call is propagated to ConsensusManager which creates ConsensusConnection and ConsensusConnection is registered with ConsensusNetIO via AddConsensusConnection() call. If a delegate is TCP/IP server with respect to the remote delegate, then the sequence of calls is the same as described above except that instead of instantiating

ConsensusNetIO, the core software simply waits on Accept() call. OnAccept() callback is called when a connection is accepted and ultimatelly the connection event is propagated to ConsensusNetIOManager via OnConnectionAccepted() call. ConsensusNetIOManager instantiates ConsensusNetIO and the rest of the calls is the same as in the client case. Note that client identification message which is sent to the server on conneciton and public key exchange between the client and server are omitted from the diagram for simplicity.

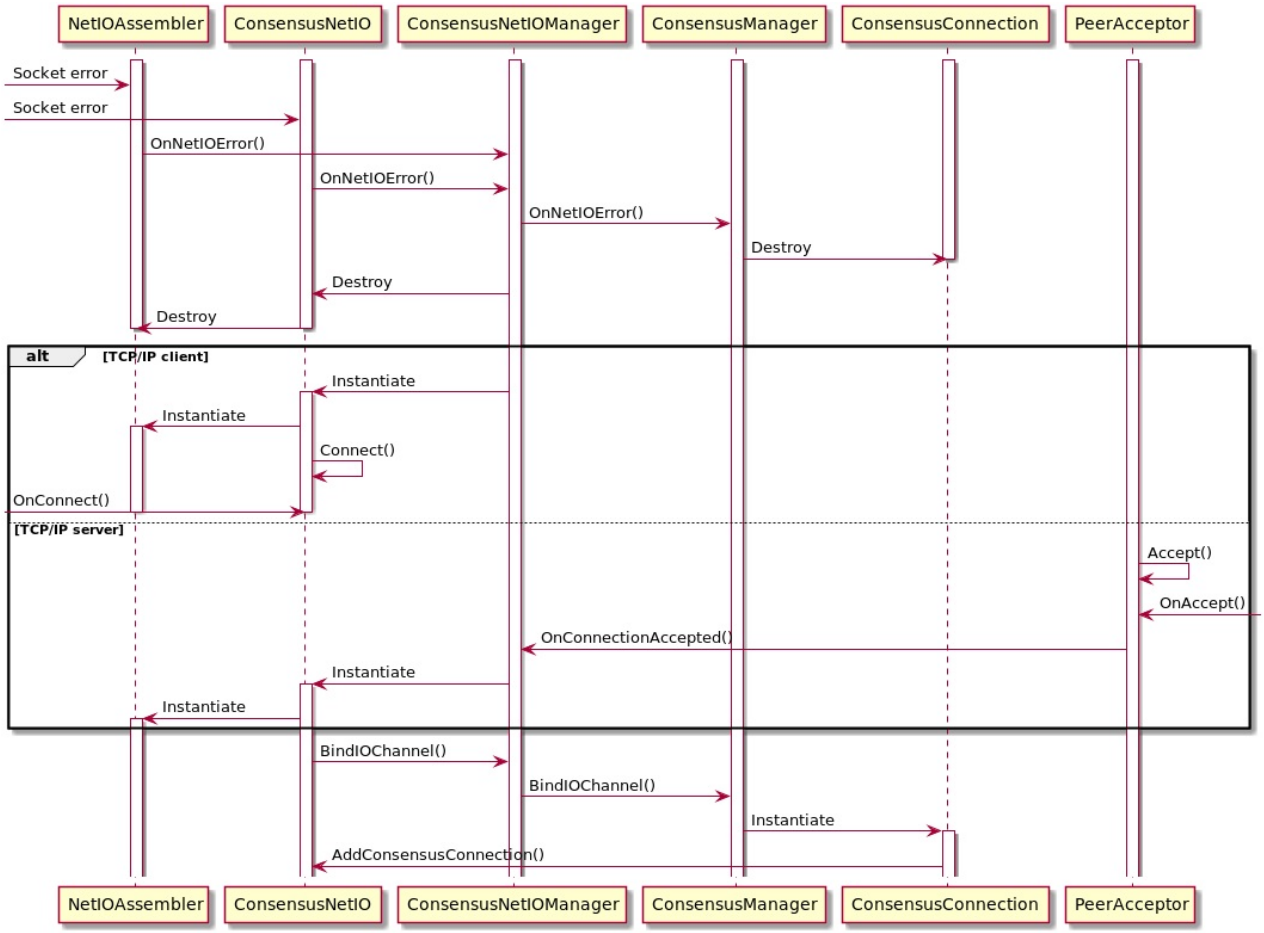

Figure 2. Sequence diagram for client reconnection on socket error.

Figure 3 shows a sequence of calls for the heartbeat logic. ConsensusNetIOManager schedules inactivity timer _inactivity_timer via ScheduleInactivityTimer() call. The timeout value is set to 20 seconds. ConsensusNetIO saves timestamp of each delegate's last received consensus/heartbeat message into _last_received (per delegate) data member. OnInactivityTimeout() callback is called when the timer times out. The call loops over each channel and if the time since last received message exceeds 80 seconds then ConsensusNetIOManager calls OnNetIOError() and the reconnection logic is envoked as described in the above paragraph. If the time since last received message exceeds 60 then ConsensusNetIOManager sends HeartBeat message with is_request set to true to the remote delegate. When a remote delegate receives HeartBeat message it 1) resets _last_received timer; 2) if HeartBeat::is_request is true then ConsensusConneciton responds with HeartBeat message with is_request set to false; otherwise ConsensusNetIO ignores HeartBeat message.

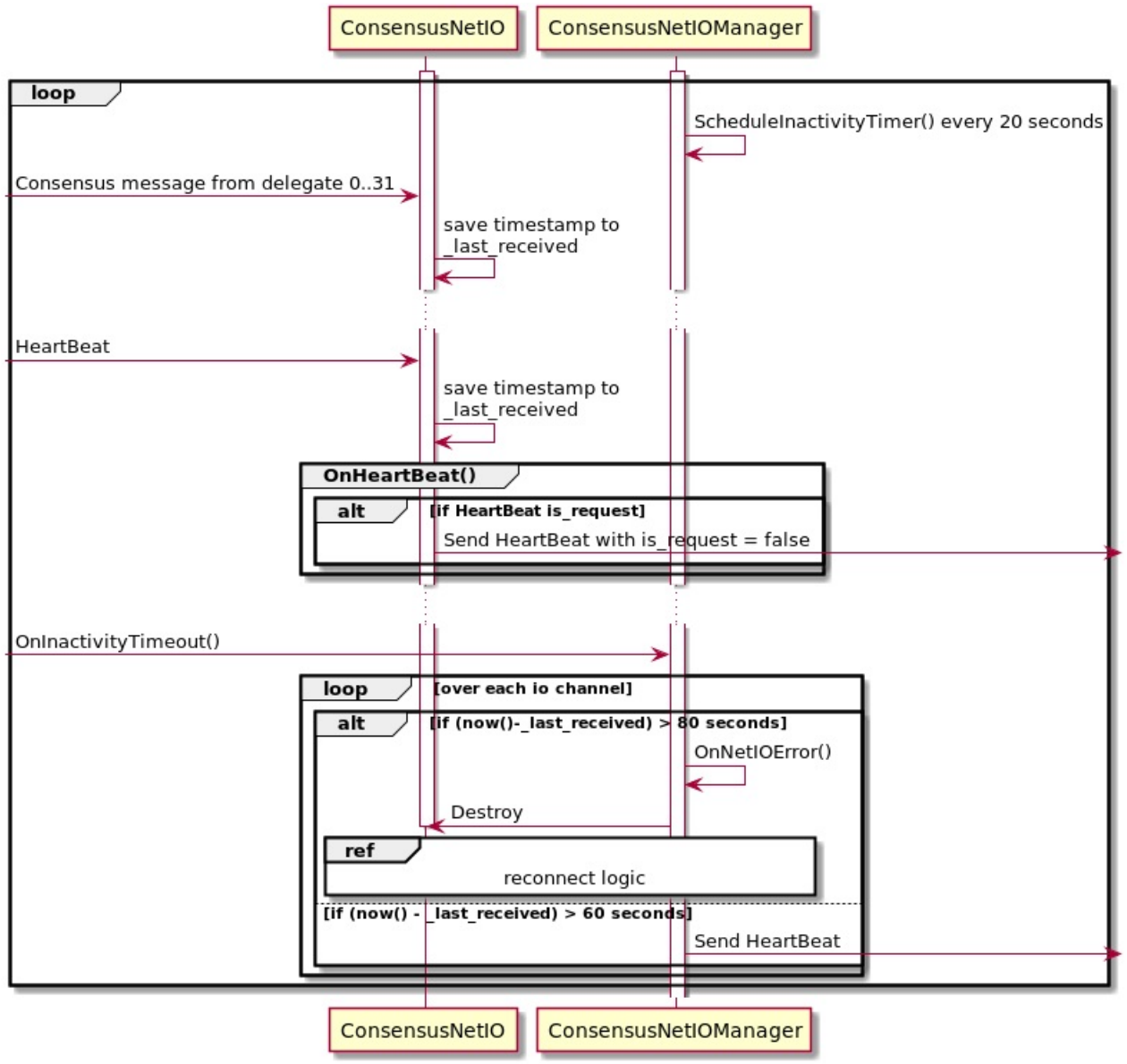

Figure 3. Sequence diagram for the heatbeat logic.