**Notes on Global Snapshot protocol**

* Assumptions:
  + On remote node, execution of snapshot is (a) same for all remote nodes and (b) is deterministic in time. This avoids issues wrt when taken vs. when sent.
  + At most one snapshot request is pending at a time.
  + Only fault is link error.
  + Using CSMA with time sync features.
  + All nodes running at same clock speed, and clock skew is small for small time spans. This means the CSMA message received time diff is accurate.
* Base
  + Variables
    - SnapshotTime: the time (Base clock) when a snapshot is to take place.
    - SnapshotId: a unique identifier for each snapshot
    - SnapshotPending: true iff a snapshot is pending
  + Initially:
    - Set SnapshotId to zero.
  + Periodically:
    - Broadcast Hello message
    - After broadcast, set SnapshotTime and SnapshotId to zero
      * Message includes broadcast period, SnapshotTime and SnapshotId
      * SnapshotTime value of zero means no snapshot requested; hello only
      * Used by remote nodes to (a) sync (b) make skew estimates (c) schedule snapshot
  + On demand:
    - Set SnapshotTime to some time after next Snapshot broadcast scheduled time
    - Increment SnapshotId
  + On receipt of unicast Snapshot from node N: for each message in packet,
    - Enqueue the message (circular queue)
    - Signal message queue manager thread
    - Reply with SnapshotAck
  + On queue manager signal:
    - Get next message in queue; if none, block pending new signal (note possibility of race condition wrt enqueue/dequeue)
    - Process message
      * Adjust to local time
      * Save data to flash (NOR or SD NAND)
* Remote
  + States: Connected, Disconnected
  + Initially:
    - Set state Disconnected
    - Turn radio on
    - Sleep forever
  + On receipt of broadcast Snapshot: (note possibility of race condition with WaitNextSnapshot timer tick)
    - Turn radio off.
    - Start oneshot RequestSnapshotTimer w/deadline of Snapshot period, minus a Margin that takes Skew into account
    - If WaitReceiveSnapshot timer running then stop it
    - If state Disconnected
      * Set state Connected
      * Store base ID and initial time offset (from CSMA) as Offset
    - If state Connected
      * Update Skew estimate using time offset
      * If TakeSnapshotTimer active, update deadline with new Skew value
    - If SnapshotTime > 0 and TakeSnapshotTimer is stopped
      * Start oneshot TakeSnapshotTimer w/deadline based on message SnapshotTime along with Offset and Skew
      * Save SnapshotId
  + On tick of TakeSnapshotTimer:
    - Sense data
    - Unicast Snapshot message to base, including SnapshotId
  + On tick of WaitNextSnapshotTimer:
    - Turn radio on
    - Start oneshot WaitReceiveSnapshot timer w/deadline of twice Margin
  + On tick of WaitReceiveSnapshot timer:
    - Set state Disconnected