MODULE *ZabWithFLEAndSYNC*

This is the formal specification for the *Zab* consensus algorithm, which means *Zookeeper* Atomic *Broadcast*. The differences from *ZabWithFLE* is that we implement phase RECOVERY-SYNC.

Reference: *FLE*: *FastLeaderElection.java*, *Vote.java*, *QuorumPeer.java*, *e.g.* in https://*github.com*/*apache*/zookeeper.
ZAB: *QuorumPeer.java*, *Learner.java*, *Follower.java*, *LearnerHandler.java*, *Leader.java*, *e.g.* in https://*github.com*/*apache*/zookeeper. https://*cwiki.apache.org*/confluence/display/*ZOOKEEPER*/*Zab*1.0.

EXTENDS *FastLeaderElection*

The set of requests that can go into history

CONSTANT *Value* * Replaced by *recorder.nClientRequest*

$Value \triangleq Nat$

Zab states

CONSTANTS *ELECTION*, *DISCOVERY*, *SYNCHRONIZATION*, *BROADCAST*

Sync modes & message types

CONSTANTS *DIFF*, *TRUNC*

Message types

CONSTANTS *FOLLOWERINFO*, *LEADERINFO*, *ACKEPOCH*, *NEWLEADER*, *ACKLD*, *UPTODATE*, *PROPOSAL*, *ACK*, *COMMIT*

NOTE: In production, there is no message type *ACKLD*. Server judges if counter of *ACK* is 0 to distinguish one *ACK* represents *ACKLD* or not. Here we divide *ACK* into *ACKLD* and *ACK*, to enhance readability of spec.

[*MaxTimeoutFailures*, *MaxTransactionNum*, *MaxEpoch*]

CONSTANT *Parameters*

TODO: Here we can add more constraints to decrease space, like restart, partition.

$MAXEPOCH \triangleq 10$

Variables in annotations mean variables defined in *FastLeaderElection*.

Variables that all servers use.

```
VARIABLES zabState,        Current phase of server, in
                           {ELECTION, DISCOVERY, SYNCHRONIZATION, BROADCAST}.
          acceptedEpoch,   Epoch of the last LEADERINFO packet accepted,
                           namely f.p in paper.
          lastCommitted,   Maximum index and zxid known to be committed,
                           namely 'lastCommitted' in Leader. Starts from 0,
                           and increases monotonically before restarting.
          initialHistory   history that server initially has before election.
            state,          \* State of server, in {LOOKING, FOLLOWING, LEADING}.
            currentEpoch,   \* Epoch of the last NEWLEADER packet accepted,
                           namely f.a in paper.
            lastProcessed,  \* Index and zxid of the last processed txn.
            history         \* History of servers: sequence of transactions,
```

containing: *zxid*, value, *ackSid*, epoch.
leader : [*committedRequests* + *toBeApplied*] [*outstandingProposals*]
follower: [*committedRequests*] [*pendingTxns*]

Variables only used for leader.

VARIABLES *learners*, Set of servers leader connects, namely 'learners' in Leader.

connecting, Set of learners leader has received *FOLLOWERINFO* from, namely 'connectingFollowers' in Leader.

electing, Set of learners leader has received *ACKEPOCH* from, namely 'electingFollowers' in Leader. Set of record [*sid*, *peerLastZxid*, *inQuorum*]. And $peerLastZxid = \langle -1, -1 \rangle$ means has done *syncFollower* with this *sid*. $inQuorum =$ TRUE means in code it is one element in 'electingFollowers'.

ackldRecv, Set of learners leader has received *ACK* of *NEWLEADER* from, namely 'newLeaderProposal' in Leader.

forwarding, Set of learners that are synced with leader, namely 'forwardingFollowers' in Leader.

tempMaxEpoch $(\{Maximum\ epoch\ in\ FOLLOWEINFO\} + 1)$ that leader has received from learners, namely 'epoch' in Leader.

leadingVoteSet \* Set of voters that follow leader.

Variables only used for follower.

VARIABLES *leaderAddr*, If follower has connected with leader. If follower lost connection, then null.

packetsSync packets of *PROPOSAL* and *COMMIT* from leader, namely 'packetsNotCommitted' and 'packetsCommitted' in *SyncWithLeader* in Learner.

Variables about network channel.

VARIABLE *msgs* Simulates network channel. $msgs[i][j]$ means the input buffer of server j from server i.

electionMsgs \* Network channel in *FLE* module.

Variables only used in verifying properties.

VARIABLES *epochLeader*, Set of leaders in every epoch.

proposalMsgsLog, Set of all broadcast messages.

```
                 violatedInvariants  Check whether there are conditions
                                     contrary to the facts.
Variables only used for looking.
VARIABLE  currentVote,  \* Info of current vote, namely 'currentVote'
                        \* in QuorumPeer.
          logicalClock, \* Election instance, namely 'logicalClock'
                        \* in FastLeaderElection.
          receiveVotes, \* Votes from current FLE round, namely
                        \* 'recvset' in FastLeaderElection.
          outOfElection, \* Votes from previous and current FLE round,
                        \* namely 'outofelection' in FastLeaderElection.
          recvQueue,    \* Queue of received notifications or timeout
                        \* signals.
          waitNotmsg    \* Whether waiting for new not.See line 1050
                        \* in FastLeaderElection for details.
VARIABLE  idTable \* For mapping Server to Integers,
                    to compare ids between servers.
   Update: we have transformed idTable from variable to function.

VARIABLE  clientReuqest \* Start from 0, and increases monotonically
                          when LeaderProcessRequest performed. To
                          avoid existing two requests with same value.
   Update: Remove it to recorder.nClientRequest.

Variable used for recording critical data,
to constrain state space or update values.
VARIABLE recorder  Consists: members of Parameters and pc, values.
                   Form is record:
                   [pc, nTransaction, maxEpoch, nTimeout, nClientRequest]
```

$serverVars \triangleq \langle state, currentEpoch, lastProcessed, zabState,$
$\quad acceptedEpoch, history, lastCommitted, initialHistory\rangle$

$electionVars \triangleq electionVarsL$

$leaderVars \triangleq \langle leadingVoteSet, learners, connecting, electing,$
$\quad ackldRecv, forwarding, tempMaxEpoch\rangle$

$followerVars \triangleq \langle leaderAddr, packetsSync\rangle$

$verifyVars \triangleq \langle proposalMsgsLog, epochLeader, violatedInvariants\rangle$

$msgVars \triangleq \langle msgs, electionMsgs\rangle$

$vars \triangleq \langle serverVars, electionVars, leaderVars,$
$\quad followerVars, verifyVars, msgVars, recorder\rangle$

$ServersIncNullPoint \triangleq Server \cup \{NullPoint\}$

$Zxid \triangleq$
$\quad Seq(Nat \cup \{-1\})$

$HistoryItem \triangleq$
$\quad [zxid : Zxid,$
$\quad\ value : Value,$
$\quad\ ackSid : \text{SUBSET}\ Server,$
$\quad\ epoch : Nat]$

$Proposal \triangleq$
$\quad [source : Server,$
$\quad\ epoch : Nat,$
$\quad\ zxid : Zxid,$
$\quad\ data : Value]$

$LastItem \triangleq$
$\quad [index : Nat,\ zxid : Zxid]$

$SyncPackets \triangleq$
$\quad [notCommitted : Seq(HistoryItem),$
$\quad\ committed : Seq(Zxid)]$

$Message \triangleq$
$\quad [mtype : \{FOLLOWERINFO\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{LEADERINFO\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{ACKEPOCH\},\ mzxid : Zxid,\ mepoch : Nat \cup \{-1\}] \cup$
$\quad [mtype : \{DIFF\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{TRUNC\},\ mtruncZxid : Zxid] \cup$
$\quad [mtype : \{PROPOSAL\},\ mzxid : Zxid,\ mdata : Value] \cup$
$\quad [mtype : \{COMMIT\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{NEWLEADER\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{ACKLD\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{ACK\},\ mzxid : Zxid] \cup$
$\quad [mtype : \{UPTODATE\},\ mzxid : Zxid]$

$ElectionState \triangleq \{LOOKING,\ FOLLOWING,\ LEADING\}$

$ZabState \triangleq \{ELECTION,\ DISCOVERY,\ SYNCHRONIZATION,\ BROADCAST\}$

$ViolationSet \triangleq \{$“stateInconsistent”, “proposalInconsistent”,
“commitInconsistent”, “ackInconsistent”,
“messageIllegal”$\}$

$Electing \triangleq [sid : Server,$
$\quad peerLastZxid : Zxid,$
$\quad inQuorum : \text{BOOLEAN}\ ]$

$Vote \triangleq$

$$\begin{array}{l}
\quad [\mathit{proposedLeader} : \mathit{ServersIncNullPoint}, \\
\quad \ \mathit{proposedZxid} : \mathit{Zxid}, \\
\quad \ \mathit{proposedEpoch} : \mathit{Nat}]
\end{array}$$

$$\begin{array}{l}
\mathit{ElectionVote} \triangleq \\
\quad [\mathit{vote} : \mathit{Vote}, \mathit{round} : \mathit{Nat}, \mathit{state} : \mathit{ElectionState}, \mathit{version} : \mathit{Nat}]
\end{array}$$

$$\begin{array}{l}
\mathit{ElectionMsg} \triangleq \\
\quad [\mathit{mtype} : \{\mathit{NOTIFICATION}\}, \\
\quad \ \mathit{msource} : \mathit{Server}, \\
\quad \ \mathit{mstate} \ : \mathit{ElectionState}, \\
\quad \ \mathit{mround} : \mathit{Nat}, \\
\quad \ \mathit{mvote} : \mathit{Vote}] \cup \\
\quad [\mathit{mtype} : \{\mathit{NONE}\}]
\end{array}$$

$$\begin{array}{l}
\mathit{TypeOK} \triangleq \\
\quad \land \quad \mathit{zabState} \in [\mathit{Server} \rightarrow \mathit{ZabState}] \\
\quad \land \quad \mathit{acceptedEpoch} \ \in [\mathit{Server} \rightarrow \mathit{Nat}] \\
\quad \land \quad \mathit{lastCommitted} \in [\mathit{Server} \rightarrow \mathit{LastItem}] \\
\quad \land \quad \mathit{learners} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{Server}] \\
\quad \land \quad \mathit{connecting} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{ServersIncNullPoint}] \\
\quad \land \quad \mathit{electing} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{Electing}] \\
\quad \land \quad \mathit{ackldRecv} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{ServersIncNullPoint}] \\
\quad \land \quad \mathit{forwarding} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{Server}] \\
\quad \land \quad \mathit{initialHistory} \in [\mathit{Server} \rightarrow \mathit{Seq}(\mathit{HistoryItem})] \\
\quad \land \quad \mathit{tempMaxEpoch} \in [\mathit{Server} \rightarrow \mathit{Nat}] \\
\quad \land \quad \mathit{leaderAddr} \in [\mathit{Server} \rightarrow \mathit{ServersIncNullPoint}] \\
\quad \land \quad \mathit{packetsSync} \in [\mathit{Server} \rightarrow \mathit{SyncPackets}] \\
\quad \land \quad \mathit{proposalMsgsLog} \in \textsc{subset}\ \mathit{Proposal} \\
\quad \land \quad \mathit{epochLeader} \in [1 \mathrel{..} \mathit{MAXEPOCH} \rightarrow \textsc{subset}\ \mathit{Server}] \\
\quad \land \quad \mathit{violatedInvariants} \in [\mathit{ViolationSet} \rightarrow \textsc{boolean}\ ] \\
\quad \land \quad \mathit{msgs} \in [\mathit{Server} \rightarrow [\mathit{Server} \rightarrow \mathit{Seq}(\mathit{Message})]] \\
\quad \ \text{Fast Leader Election} \\
\quad \land\ \mathit{electionMsgs} \in [\mathit{Server} \rightarrow [\mathit{Server} \rightarrow \mathit{Seq}(\mathit{ElectionMsg})]] \\
\quad \land\ \mathit{recvQueue} \in [\mathit{Server} \rightarrow \mathit{Seq}(\mathit{ElectionMsg})] \\
\quad \land\ \mathit{leadingVoteSet} \in [\mathit{Server} \rightarrow \textsc{subset}\ \mathit{Server}] \\
\quad \land\ \mathit{receiveVotes} \in [\mathit{Server} \rightarrow [\mathit{Server} \rightarrow \mathit{ElectionVote}]] \\
\quad \land\ \mathit{currentVote} \in [\mathit{Server} \rightarrow \mathit{Vote}] \\
\quad \land\ \mathit{outOfElection} \in [\mathit{Server} \rightarrow [\mathit{Server} \rightarrow \mathit{ElectionVote}]] \\
\quad \land\ \mathit{lastProcessed} \ \in [\mathit{Server} \rightarrow \mathit{LastItem}] \\
\quad \land\ \mathit{history} \in [\mathit{Server} \rightarrow \mathit{Seq}(\mathit{HistoryItem})] \\
\quad \land\ \mathit{state} \in [\mathit{Server} \rightarrow \mathit{ElectionState}] \\
\quad \land\ \mathit{waitNotmsg} \in [\mathit{Server} \rightarrow \textsc{boolean}\ ] \\
\quad \land\ \mathit{currentEpoch} \in [\mathit{Server} \rightarrow \mathit{Nat}] \\
\quad \land\ \mathit{logicalClock} \ \ \in [\mathit{Server} \rightarrow \mathit{Nat}]
\end{array}$$

Return the maximum value from the set S

```
$Maximum(S) \triangleq$ IF $S = \{\}$ THEN $-1$
                       ELSE CHOOSE $n \in S : \forall m \in S : n \geq m$
```

Return the minimum value from the set S

```
$Minimum(S) \triangleq$ IF $S = \{\}$ THEN $-1$
                       ELSE CHOOSE $n \in S : \forall m \in S : n \leq m$
```

Check server state

```
$IsLeader(s) \triangleq state[s] = LEADING$
$IsFollower(s) \triangleq state[s] = FOLLOWING$
$IsLooking(s) \triangleq state[s] = LOOKING$

$IsMyLearner(i, j) \triangleq j \in learners[i]$
$IsMyLeader(i, j) \triangleq leaderAddr[i] = j$
$HasNoLeader(i) \triangleq leaderAddr[i] = NullPoint$
$HasLeader(i) \triangleq leaderAddr[i] \neq NullPoint$
$MyVote(i) \triangleq currentVote[i].proposedLeader$
```

Check if s is a quorum

```
$IsQuorum(s) \triangleq s \in Quorums$
```

Check $zxid$ state

```
$ToZxid(z) \triangleq [epoch \mapsto z[1], counter \mapsto z[2]]$

$TxnZxidEqual(txn, z) \triangleq txn.zxid[1] = z[1] \land txn.zxid[2] = z[2]$

$TxnEqual(txn1, txn2) \triangleq$ $\land ZxidEqual(txn1.zxid, txn2.zxid)$
                                 $\land txn1.value = txn2.value$

$EpochPrecedeInTxn(txn1, txn2) \triangleq txn1.zxid[1] < txn2.zxid[1]$
```

Actions about recorder

```
$GetParameter(p) \triangleq$ IF $p \in$ DOMAIN $Parameters$ THEN $Parameters[p]$ ELSE $0$
$GetRecorder(p) \triangleq$ IF $p \in$ DOMAIN $recorder$ THEN $recorder[p]$ ELSE $0$

$RecorderGetHelper(m) \triangleq (m :> recorder[m])$
$RecorderIncHelper(m) \triangleq (m :> recorder[m] + 1)$

$RecorderIncTimeout \triangleq RecorderIncHelper($"nTimeout"$)$
$RecorderGetTimeout \triangleq RecorderGetHelper($"nTimeout"$)$
$RecorderSetTransactionNum(pc) \triangleq ($"nTransaction" $:>$
    IF $pc[1] =$ "LeaderProcessRequest" THEN
        LET $s \triangleq$ CHOOSE $i \in Server :$
                $\forall j \in Server : Len(history'[i]) \geq Len(history'[j])$
        IN $Len(history'[s])$
    ELSE $recorder[$"nTransaction"$])$
$RecorderSetMaxEpoch(pc) \triangleq ($"maxEpoch" $:>$
                                  IF $pc[1] =$ "LeaderProcessFOLLOWERINFO" THEN
                                     LET $s \triangleq$ CHOOSE $i \in Server$ :
                                            $\forall j \in Server : acceptedEpoch'[i] \geq acceptedEpoch'[j]$
                                     IN   $acceptedEpoch'[s]$
                                   ELSE  $recorder[$"maxEpoch"$])$
$RecorderSetRequests(pc)$          $\triangleq$ ("nClientRequest" :>
                                  IF $pc[1] =$ "LeaderProcessRequest" THEN
                                     $recorder[$"nClientRequest"$] + 1$
                                   ELSE  $recorder[$"nClientRequest"$])$
$RecorderSetPc(pc)$        $\triangleq$ ("pc" :> $pc$)
$RecorderSetFailure(pc)$   $\triangleq$ CASE $pc[1] =$ "Timeout"          $\rightarrow RecorderIncTimeout$
                             $\square$    $pc[1] =$ "LeaderTimeout"    $\rightarrow RecorderIncTimeout$
                             $\square$    $pc[1] =$ "FollowerTimeout"  $\rightarrow RecorderIncTimeout$
                             $\square$    OTHER                       $\rightarrow RecorderGetTimeout$

$UpdateRecorder(pc)$ $\triangleq$ $recorder' = RecorderSetFailure(pc)$      @@ $RecorderSetTransactionNum(pc)$
                                   @@ $RecorderSetMaxEpoch(pc)$  @@ $RecorderSetPc(pc)$
                                   @@ $RecorderSetRequests(pc)$  @@ $recorder$
$UnchangeRecorder$  $\triangleq$ UNCHANGED $recorder$

$CheckParameterHelper(n, p, Comp(\_, \_))$ $\triangleq$ IF $p \in$ DOMAIN $Parameters$
                                                THEN $Comp(n, Parameters[p])$
                                                ELSE TRUE
$CheckParameterLimit(n, p)$ $\triangleq$ $CheckParameterHelper(n, p,$ LAMBDA $i, j : i < j)$

$CheckTimeout$         $\triangleq$ $CheckParameterLimit(recorder.nTimeout,$      "MaxTimeoutFailures"$)$
$CheckTransactionNum$  $\triangleq$ $CheckParameterLimit(recorder.nTransaction,$  "MaxTransactionNum"$)$
$CheckEpoch$           $\triangleq$ $CheckParameterLimit(recorder.maxEpoch,$      "MaxEpoch"$)$

$CheckStateConstraints$ $\triangleq$ $CheckTimeout \land CheckTransactionNum \land CheckEpoch$
---------------------------------------------------------------------------------------
 Actions about network
$PendingFOLLOWERINFO(i, j)$ $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = FOLLOWERINFO$
$PendingLEADERINFO(i, j)$   $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = LEADERINFO$
$PendingACKEPOCH(i, j)$     $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = ACKEPOCH$
$PendingNEWLEADER(i, j)$    $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = NEWLEADER$
$PendingACKLD(i, j)$        $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = ACKLD$
$PendingUPTODATE(i, j)$     $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = UPTODATE$
$PendingPROPOSAL(i, j)$     $\triangleq$ $\land msgs[j][i] \neq \langle\rangle$
                                $\land msgs[j][i][1].mtype = PROPOSAL$
```

$$\begin{array}{ll} \mathit{PendingACK}(i, j) & \triangleq \begin{array}{l} \land\ \mathit{msgs}[j][i] \neq \langle\rangle \\ \land\ \mathit{msgs}[j][i][1].\mathit{mtype} = \mathit{ACK} \end{array} \\ \mathit{PendingCOMMIT}(i, j) & \triangleq \begin{array}{l} \land\ \mathit{msgs}[j][i] \neq \langle\rangle \\ \land\ \mathit{msgs}[j][i][1].\mathit{mtype} = \mathit{COMMIT} \end{array} \end{array}$$

Add a message to *msgs* − add a message *m* to *msgs*.

$$\mathit{Send}(i, j, m) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } ![i][j] = \mathit{Append}(\mathit{msgs}[i][j], m)]$$

$$\mathit{SendPackets}(i, j, \mathit{ms}) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } ![i][j] = \mathit{msgs}[i][j] \circ \mathit{ms}]$$

$$\begin{array}{l} \mathit{DiscardAndSendPackets}(i, j, \mathit{ms}) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } ![j][i] = \mathit{Tail}(\mathit{msgs}[j][i]), \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad ![i][j] = \mathit{msgs}[i][j] \circ \mathit{ms}] \end{array}$$

Remove a message from *msgs* − discard head of *msgs*.

$$\mathit{Discard}(i, j) \triangleq \mathit{msgs}' = \text{IF } \mathit{msgs}[i][j] \neq \langle\rangle \begin{array}{l} \text{THEN } [\mathit{msgs} \text{ EXCEPT } ![i][j] = \mathit{Tail}(\mathit{msgs}[i][j])] \\ \text{ELSE } \mathit{msgs} \end{array}$$

Leader broadcasts a *message*(*PROPOSAL*/*COMMIT*) to all other servers in *forwardingFollowers*.

$$\mathit{Broadcast}(i, m) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } ![i] = [v \in \mathit{Server} \mapsto \begin{array}{l} \text{IF } \begin{array}{l} \land\ v \in \mathit{forwarding}[i] \\ \land\ v \neq i \end{array} \\ \text{THEN } \mathit{Append}(\mathit{msgs}[i][v], m) \\ \text{ELSE } \mathit{msgs}[i][v]]] \end{array}$$

$$\begin{array}{l} \mathit{DiscardAndBroadcast}(i, j, m) \triangleq \\ \qquad \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } \begin{array}{l} ![j][i] = \mathit{Tail}(\mathit{msgs}[j][i]), \\ ![i] = [v \in \mathit{Server} \mapsto \begin{array}{l} \text{IF } \begin{array}{l} \land\ v \in \mathit{forwarding}[i] \\ \land\ v \neq i \end{array} \\ \text{THEN } \mathit{Append}(\mathit{msgs}[i][v], m) \\ \text{ELSE } \mathit{msgs}[i][v]]] \end{array} \end{array} \end{array}$$

Leader broadcasts *LEADERINFO* to all other servers in *connectingFollowers*.

$$\begin{array}{l} \mathit{DiscardAndBroadcastLEADERINFO}(i, j, m) \triangleq \\ \qquad \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } \begin{array}{l} ![j][i] = \mathit{Tail}(\mathit{msgs}[j][i]), \\ ![i] = [v \in \mathit{Server} \mapsto \begin{array}{l} \text{IF } \begin{array}{l} \land\ v \in \mathit{connecting}'[i] \\ \land\ v \in \mathit{learners}[i] \\ \land\ v \neq i \end{array} \\ \text{THEN } \mathit{Append}(\mathit{msgs}[i][v], m) \\ \text{ELSE } \mathit{msgs}[i][v]]] \end{array} \end{array} \end{array}$$

Leader broadcasts *UPTODATE* to all other servers in *newLeaderProposal*.

$$\begin{array}{l} \mathit{DiscardAndBroadcastUPTODATE}(i, j, m) \triangleq \\ \qquad \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } \begin{array}{l} ![j][i] = \mathit{Tail}(\mathit{msgs}[j][i]), \\ ![i] = [v \in \mathit{Server} \mapsto \begin{array}{l} \text{IF } \begin{array}{l} \land\ v \in \mathit{ackldRecv}'[i] \\ \land\ v \in \mathit{learners}[i] \\ \land\ v \neq i \end{array} \\ \text{THEN } \mathit{Append}(\mathit{msgs}[i][v], m) \\ \text{ELSE } \mathit{msgs}[i][v]]] \end{array} \end{array} \end{array}$$

Combination of *Send* and *Discard* − discard head of *msgs*[*j*][*i*] and add *m* into *msgs*.

$$\mathit{Reply}(i, j, m) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } \begin{array}{l} ![j][i] = \mathit{Tail}(\mathit{msgs}[j][i]), \\ ![i][j] = \mathit{Append}(\mathit{msgs}[i][j], m)] \end{array}$$

Shuffle input buffer.

$$\mathit{Clean}(i, j) \triangleq \mathit{msgs}' = [\mathit{msgs} \text{ EXCEPT } ![j][i] = \langle\rangle, ![i][j] = \langle\rangle]$$

$$\begin{array}{l} \mathit{CleanInputBuffer}(i) \triangleq \mathit{msgs}' = [s \in \mathit{Server} \mapsto [v \in \mathit{Server} \mapsto \text{IF } v = i \text{ THEN } \langle\rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } \mathit{msgs}[s][v]]] \end{array}$$

$$\begin{array}{l} \mathit{CleanInputBufferInCluster}(S) \triangleq \mathit{msgs}' = [s \in \mathit{Server} \mapsto \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad [v \in \mathit{Server} \mapsto \text{IF } v \in S \text{ THEN } \langle\rangle \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{ELSE } \mathit{msgs}[s][v]]] \end{array}$$

Define initial values for all variables

$$\begin{array}{lll} \mathit{InitServerVars} \triangleq & \land\ \mathit{InitServerVarsL} & \\ & \land\ \mathit{zabState} & = [s \in \mathit{Server} \mapsto \mathit{ELECTION}] \\ & \land\ \mathit{acceptedEpoch} & = [s \in \mathit{Server} \mapsto 0] \\ & \land\ \mathit{lastCommitted} & = [s \in \mathit{Server} \mapsto [\mathit{index} \mapsto 0, \\ & & \qquad\qquad\qquad\quad \mathit{zxid} \ \ \mapsto \langle 0, 0\rangle]] \\ & \land\ \mathit{initialHistory} & = [s \in \mathit{Server} \mapsto \langle\rangle] \end{array}$$

$$\begin{array}{lll} \mathit{InitLeaderVars} \triangleq & \land\ \mathit{InitLeaderVarsL} & \\ & \land\ \mathit{learners} & = [s \in \mathit{Server} \mapsto \{\}] \\ & \land\ \mathit{connecting} & = [s \in \mathit{Server} \mapsto \{\}] \\ & \land\ \mathit{electing} & = [s \in \mathit{Server} \mapsto \{\}] \\ & \land\ \mathit{ackldRecv} & = [s \in \mathit{Server} \mapsto \{\}] \\ & \land\ \mathit{forwarding} & = [s \in \mathit{Server} \mapsto \{\}] \\ & \land\ \mathit{tempMaxEpoch} & = [s \in \mathit{Server} \mapsto 0] \end{array}$$

$$\mathit{InitElectionVars} \triangleq \mathit{InitElectionVarsL}$$

$$\begin{array}{lll} \mathit{InitFollowerVars} \triangleq & \land\ \mathit{leaderAddr} & = [s \in \mathit{Server} \mapsto \mathit{NullPoint}] \\ & \land\ \mathit{packetsSync} & = [s \in \mathit{Server} \mapsto \\ & & \qquad [\mathit{notCommitted} \mapsto \langle\rangle, \\ & & \qquad\ \ \mathit{committed} \quad\ \mapsto \langle\rangle]] \end{array}$$

$$\begin{array}{lll} \mathit{InitVerifyVars} \triangleq & \land\ \mathit{proposalMsgsLog} & = \{\} \\ & \land\ \mathit{epochLeader} & = [i \in 1 \,..\, \mathit{MAXEPOCH} \mapsto \{\}] \\ & \land\ \mathit{violatedInvariants} & = [\mathit{stateInconsistent} \quad\ \ \mapsto \text{FALSE}, \\ & & \quad\ \mathit{proposalInconsistent} \mapsto \text{FALSE}, \\ & & \quad\ \mathit{commitInconsistent} \quad \mapsto \text{FALSE}, \\ & & \quad\ \mathit{ackInconsistent} \qquad \mapsto \text{FALSE}, \\ & & \quad\ \mathit{messageIllegal} \qquad \mapsto \text{FALSE}] \end{array}$$

$$\begin{array}{lll} \mathit{InitMsgVars} \triangleq & \land\ \mathit{msgs} & = [s \in \mathit{Server} \mapsto [v \in \mathit{Server} \mapsto \langle\rangle]] \\ & \land\ \mathit{electionMsgs} & = [s \in \mathit{Server} \mapsto [v \in \mathit{Server} \mapsto \langle\rangle]] \end{array}$$

$$\begin{array}{l} \mathit{InitRecorder} \triangleq \mathit{recorder} = [\mathit{nTimeout} \qquad\ \mapsto 0, \\ \qquad\qquad\qquad\qquad\qquad\quad \mathit{nTransaction} \quad\ \mapsto 0, \\ \qquad\qquad\qquad\qquad\qquad\quad \mathit{maxEpoch} \qquad\ \mapsto 0, \\ \qquad\qquad\qquad\qquad\qquad\quad \mathit{pc} \qquad\qquad\quad\ \mapsto \langle \text{“Init”} \rangle, \\ \qquad\qquad\qquad\qquad\qquad\quad \mathit{nClientRequest} \mapsto 0] \end{array}$$

$$\mathit{Init} \triangleq \ \land\ \mathit{InitServerVars}$$

```
          $\land$ $InitLeaderVars$
          $\land$ $InitElectionVars$
          $\land$ $InitFollowerVars$
          $\land$ $InitVerifyVars$
          $\land$ $InitMsgVars$
          $\land$ $InitRecorder$
```

```
$ZabTurnToLeading(i) \triangleq$
        $\land$ $zabState'$       $= [zabState$    EXCEPT $![i]$  $= DISCOVERY]$
        $\land$ $learners'$       $= [learners$    EXCEPT $![i]$  $= \{i\}]$
        $\land$ $connecting'$     $= [connecting$ EXCEPT $![i] = \{i\}]$
        $\land$ $electing'$       $= [electing$    EXCEPT $![i]$  $= \{[sid$          $\mapsto i,$
                                                         $peerLastZxid \mapsto \langle -1, -1\rangle,$
                                                         $inQuorum$     $\mapsto$ TRUE$]\}]$
        $\land$ $ackldRecv'$      $= [ackldRecv$   EXCEPT $![i] = \{i\}]$
        $\land$ $forwarding'$     $= [forwarding$ EXCEPT $![i] = \{\}]$
        $\land$ $initialHistory'$ $= [initialHistory$ EXCEPT $![i]$    $= history'[i]]$
        $\land$ $tempMaxEpoch'$ $= [tempMaxEpoch$    EXCEPT $![i] = acceptedEpoch[i] + 1]$

$ZabTurnToFollowing(i) \triangleq$
        $\land$ $zabState' = [zabState$ EXCEPT $![i] = DISCOVERY]$
        $\land$ $initialHistory' = [initialHistory$ EXCEPT $![i] = history'[i]]$
        $\land$ $packetsSync' = [packetsSync$ EXCEPT $![i].notCommitted = \langle\rangle,$
                                              $![i].committed = \langle\rangle]$
```

Fast Leader Election

```
$FLEReceiveNotmsg(i, j) \triangleq$
        $\land$ $ReceiveNotmsg(i, j)$
        $\land$ UNCHANGED $\langle zabState, acceptedEpoch, lastCommitted, learners, connecting,$
                      $initialHistory, electing, ackldRecv, forwarding, tempMaxEpoch,$
                      $followerVars, verifyVars, msgs\rangle$
        $\land$ $UpdateRecorder(\langle$"FLEReceiveNotmsg"$, i, j\rangle)$

$FLENotmsgTimeout(i) \triangleq$
        $\land$ $NotmsgTimeout(i)$
        $\land$ UNCHANGED $\langle zabState, acceptedEpoch, lastCommitted, learners, connecting,$
                      $initialHistory, electing, ackldRecv, forwarding, tempMaxEpoch,$
                      $followerVars, verifyVars, msgs\rangle$
        $\land$ $UpdateRecorder(\langle$"FLENotmsgTimeout"$, i\rangle)$

$FLEHandleNotmsg(i) \triangleq$
        $\land$ $HandleNotmsg(i)$
        $\land$ LET $newState \triangleq state'[i]$
          IN
          $\lor$ $\land$ $newState = LEADING$
             $\land$ $ZabTurnToLeading(i)$
```

```
              ∧ UNCHANGED packetsSync
           ∨ ∧ newState = FOLLOWING
              ∧ ZabTurnToFollowing(i)
              ∧ UNCHANGED ⟨learners, connecting, electing, ackldRecv,
                                 forwarding, tempMaxEpoch⟩
           ∨ ∧ newState = LOOKING
              ∧ UNCHANGED ⟨zabState, learners, connecting, electing, ackldRecv,
                                 forwarding, tempMaxEpoch, packetsSync, initialHistory⟩
        ∧ UNCHANGED ⟨lastCommitted, acceptedEpoch, leaderAddr, verifyVars, msgs⟩
        ∧ UpdateRecorder(⟨"FLEHandleNotmsg", i⟩)
```

On the premise that $ReceiveVotes.HasQuorums = \text{TRUE}$,
corresponding to logic in line $1050 - 1055$ in *FastLeaderElection*.

```
FLEWaitNewNotmsg(i) ≜
        ∧ WaitNewNotmsg(i)
        ∧ UNCHANGED ⟨zabState, acceptedEpoch, lastCommitted, learners, connecting,
                          electing, ackldRecv, forwarding, tempMaxEpoch, initialHistory,
                          followerVars, verifyVars, msgs⟩
        ∧ UpdateRecorder(⟨"FLEWaitNewNotmsg", i⟩)
```

On the premise that $ReceiveVotes.HasQuorums = \text{TRUE}$,
corresponding to logic in line $1061 - 1066$ in *FastLeaderElection*.

```
FLEWaitNewNotmsgEnd(i) ≜
        ∧ WaitNewNotmsgEnd(i)
        ∧ LET newState ≜ state′[i]
           IN
           ∨ ∧ newState = LEADING
              ∧ ZabTurnToLeading(i)
              ∧ UNCHANGED packetsSync
           ∨ ∧ newState = FOLLOWING
              ∧ ZabTurnToFollowing(i)
              ∧ UNCHANGED ⟨learners, connecting, electing, ackldRecv, forwarding,
                                 tempMaxEpoch⟩
           ∨ ∧ newState = LOOKING
              ∧ PrintT("Note: New state is LOOKING in FLEWaitNewNotmsgEnd," ∘
                      " which should not happen.")
              ∧ UNCHANGED ⟨zabState, learners, connecting, electing, ackldRecv,
                                 forwarding, tempMaxEpoch, initialHistory, packetsSync⟩
        ∧ UNCHANGED ⟨lastCommitted, acceptedEpoch, leaderAddr, verifyVars, msgs⟩
        ∧ UpdateRecorder(⟨"FLEWaitNewNotmsgEnd", i⟩)
```

```
InitialVotes ≜ [vote    ↦ InitialVote,
                 round   ↦ 0,
                 state   ↦ LOOKING,
                 version ↦ 0]
```

Equals to for every server in S, performing action *ZabTimeout*.

```
$ZabTimeoutInCluster(S) \triangleq$
        $\land\ state' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } LOOKING \text{ ELSE } state[s]]$
        $\land\ lastProcessed' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } InitLastProcessed(s)$
                                                      $\text{ELSE } lastProcessed[s]]$
        $\land\ logicalClock' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } logicalClock[s] + 1$
                                                      $\text{ELSE } logicalClock[s]]$
        $\land\ currentVote' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN}$
                                                      $[proposedLeader \mapsto s,$
                                                       $proposedZxid \mapsto lastProcessed'[s].zxid,$
                                                       $proposedEpoch \mapsto currentEpoch[s]]$
                                                      $\text{ELSE } currentVote[s]]$
        $\land\ receiveVotes' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } [v \in Server \mapsto InitialVotes]$
                                                      $\text{ELSE } receiveVotes[s]]$
        $\land\ outOfElection' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } [v \in Server \mapsto InitialVotes]$
                                                      $\text{ELSE } outOfElection[s]]$
        $\land\ recvQueue' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } \langle [mtype \mapsto NONE] \rangle$
                                                      $\text{ELSE } recvQueue[s]]$
        $\land\ waitNotmsg' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN FALSE ELSE } waitNotmsg[s]]$
        $\land\ leadingVoteSet' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } \{\} \text{ ELSE } leadingVoteSet[s]]$
        $\land\ \text{UNCHANGED } \langle electionMsgs, currentEpoch, history \rangle$
        $\land\ zabState' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } ELECTION \text{ ELSE } zabState[s]]$
        $\land\ leaderAddr' = [s \in Server \mapsto \text{IF } s \in S \text{ THEN } NullPoint \text{ ELSE } leaderAddr[s]]$
        $\land\ CleanInputBufferInCluster(S)$
```

Describe how a server transitions from *LEADING*/*FOLLOWING* to *LOOKING*.

```
$FollowerShutdown(i) \triangleq$
        $\land\ ZabTimeout(i)$
        $\land\ zabState' = [zabState \text{ EXCEPT } ![i] = ELECTION]$
        $\land\ leaderAddr' = [leaderAddr \text{ EXCEPT } ![i] = NullPoint]$
        $\land\ CleanInputBuffer(i)$

$LeaderShutdown(i) \triangleq$
        $\land\ \text{LET } cluster \triangleq \{i\} \cup learners[i]$
            $\text{IN } ZabTimeoutInCluster(cluster)$
        $\land\ learners' = [learners \text{ EXCEPT } ![i] = \{\}]$
        $\land\ forwarding' = [forwarding \text{ EXCEPT } ![i] = \{\}]$

$RemoveElecting(set, sid) \triangleq$
        $\text{LET } sid\_electing \triangleq \{s.sid : s \in set\}$
        $\text{IN } \text{IF } sid \notin sid\_electing \text{ THEN } set$
            $\text{ELSE LET } info \triangleq \text{CHOOSE } s \in set : s.sid = sid$
                     $new\_info \triangleq [sid \mapsto sid,$
                                  $peerLastZxid \mapsto \langle -1, -1 \rangle,$
                                  $inQuorum \mapsto info.inQuorum]$
                 $\text{IN } (set \setminus \{info\}) \cup \{new\_info\}$
```

See *removeLearnerHandler* for details.

```
$RemoveLearner(i, j) \triangleq$
        $\land\ learners' \quad = [learners \quad \text{EXCEPT } ![i] \ = @ \setminus \{j\}]$
        $\land\ forwarding' = [forwarding\ \text{EXCEPT } ![i] = \text{IF } j \in forwarding[i]$
                                                    $\text{THEN } @ \setminus \{j\} \text{ ELSE } @]$
        $\land\ electing' \quad = [electing \quad \text{EXCEPT } ![i] \ = RemoveElecting(@, j)]$
```

Follower connecting to leader fails and truns to *LOOKING*.

```
$FollowerTimeout(i) \triangleq$
        $\land\ CheckTimeout$  test restrictions of $timeout\_1$
        $\land\ IsFollower(i)$
        $\land\ HasNoLeader(i)$
        $\land\ FollowerShutdown(i)$
        $\land\ CleanInputBuffer(i)$
        $\land\ \text{UNCHANGED } \langle acceptedEpoch, lastCommitted, learners, connecting, electing,$
                          $ackldRecv, forwarding, tempMaxEpoch, initialHistory,$
                          $verifyVars, packetsSync\rangle$
        $\land\ UpdateRecorder(\langle \text{“FollowerTimeout”}, i\rangle)$
```

Leader loses support from a quorum and turns to *LOOKING*.

```
$LeaderTimeout(i) \triangleq$
        $\land\ CheckTimeout$  test restrictions of $timeout\_2$
        $\land\ IsLeader(i)$
        $\land\ \neg IsQuorum(learners[i])$
        $\land\ LeaderShutdown(i)$
        $\land\ \text{UNCHANGED } \langle acceptedEpoch, lastCommitted, connecting, electing, ackldRecv,$
                    $tempMaxEpoch, initialHistory, verifyVars, packetsSync\rangle$
        $\land\ UpdateRecorder(\langle \text{“LeaderTimeout”}, i\rangle)$
```

Timeout between leader and follower.

```
$Timeout(i, j) \triangleq$
          $\land\ CheckTimeout$  test restrictions of $timeout\_3$
          $\land\ IsLeader(i) \quad \land\ IsMyLearner(i, j)$
          $\land\ IsFollower(j) \land\ IsMyLeader(j, i)$
          The action of leader $i$.
          $\land\ RemoveLearner(i, j)$
          The action of follower $j$.
          $\land\ FollowerShutdown(j)$
          $\land\ Clean(i, j)$
          $\land\ \text{UNCHANGED } \langle acceptedEpoch, lastCommitted, connecting, ackldRecv,$
                          $tempMaxEpoch, initialHistory, verifyVars, packetsSync\rangle$
          $\land\ UpdateRecorder(\langle \text{“Timeout”}, i, j\rangle)$
$Restart(i) \triangleq$
      $\land \lor \land\ IsLooking(i)$
          $\land$
        $\lor \land\ IsLeader(i)$
        ∨ ∧ $IsFollower(i)$
      ∧ UNCHANGED
      ∧ $UpdateRecorder(\langle$ "Restart", $i\rangle)$
```

Establish connection between leader and follower, containing actions like *addLearnerHandler*, *findLeader*, *connectToLeader*.

```
$ConnectAndFollowerSendFOLLOWERINFO(i, j) \triangleq$
        ∧ $IsLeader(i) \land \neg IsMyLearner(i, j)$
        ∧ $IsFollower(j) \land HasNoLeader(j) \land MyVote(j) = i$
        ∧ $learners' \quad = [learners \quad$ EXCEPT $![i] \quad = learners[i] \cup \{j\}]$
        ∧ $leaderAddr' = [leaderAddr$ EXCEPT $![j] = i]$
        ∧ $Send(j, leaderAddr'[j], [mtype \mapsto FOLLOWERINFO,$
                                  $mzxid \mapsto \langle acceptedEpoch[j], 0\rangle])$
        ∧ UNCHANGED $\langle serverVars, electionVars, leadingVoteSet, connecting,$
                           $electing, ackldRecv, forwarding, tempMaxEpoch,$
                           $verifyVars, electionMsgs, packetsSync\rangle$
        ∧ $UpdateRecorder(\langle$ "ConnectAndFollowerSendFOLLOWERINFO", $i, j\rangle)$
```

waitingForNewEpoch in Leader

```
$WaitingForNewEpoch(i) \triangleq (i \in connecting[i] \land IsQuorum(connecting[i])) =$ FALSE
$WaitingForNewEpochTurnToFalse(i) \triangleq$ ∧ $i \in connecting'[i]$
                                            ∧ $IsQuorum(connecting'[i])$
```

Leader waits for receiving *FOLLOWERINFO* from a quorum including itself, and chooses a new epoch e' as its own epoch and broadcasts *LEADERINFO*. See *getEpochToPropose* in Leader for details.

```
$LeaderProcessFOLLOWERINFO(i, j) \triangleq$
        ∧ $CheckEpoch$   test restrictions of max epoch
        ∧ $IsLeader(i)$
        ∧ $PendingFOLLOWERINFO(i, j)$
        ∧ LET $msg \triangleq msgs[j][i][1]$
              $infoOk \triangleq IsMyLearner(i, j)$
              $lastAcceptedEpoch \triangleq msg.mzxid[1]$
          IN
          ∧ $infoOk$
          ∧ ∨   1. has not broadcast LEADERINFO
               ∧ $WaitingForNewEpoch(i)$
               ∧ ∨ ∧ $zabState[i] = DISCOVERY$
                     ∧ UNCHANGED $violatedInvariants$
                 ∨ ∧ $zabState[i] \neq DISCOVERY$
                     ∧ $PrintT($"Exception: waitingFotNewEpoch true," ∘
                        " while zabState not DISCOVERY."$)$
                     ∧ $violatedInvariants' = [violatedInvariants$ EXCEPT $!.stateInconsistent =$ TRUE$]$
               ∧ $tempMaxEpoch' = [tempMaxEpoch$ EXCEPT $![i] =$ IF $lastAcceptedEpoch \geq tempMaxEpoch[i]$
                                                                THEN $lastAcceptedEpoch + 1$
                                                                ELSE $@]$
        $\land$ $connecting' = [connecting$ EXCEPT $![i] = @ \cup \{j\}]$
        $\land$ $\lor$ $\land$ $WaitingForNewEpochTurnToFalse(i)$
               $\land$ $acceptedEpoch' = [acceptedEpoch$ EXCEPT $![i] = tempMaxEpoch'[i]]$
               $\land$ LET $newLeaderZxid \triangleq \langle acceptedEpoch'[i], 0\rangle$
                       $m \triangleq [mtype \mapsto LEADERINFO,$
                                $mzxid \mapsto newLeaderZxid]$
                   IN  $DiscardAndBroadcastLEADERINFO(i, j, m)$
            $\lor$ $\land$ $\neg WaitingForNewEpochTurnToFalse(i)$
               $\land$ $Discard(j, i)$
               $\land$ UNCHANGED $acceptedEpoch$
     $\lor$   2. has broadcast $LEADERINFO$
        $\land$ $\neg WaitingForNewEpoch(i)$
        $\land$ $Reply(i, j, [mtype \mapsto LEADERINFO,$
                       $mzxid \mapsto \langle acceptedEpoch[i], 0\rangle])$
        $\land$ UNCHANGED $\langle tempMaxEpoch, connecting, acceptedEpoch, violatedInvariants\rangle$
  $\land$ UNCHANGED $\langle state, currentEpoch, lastProcessed, zabState, history, lastCommitted,$
                  $followerVars, electionVars, initialHistory, leadingVoteSet, learners,$
                  $electing, ackldRecv, forwarding, proposalMsgsLog, epochLeader,$
                  $electionMsgs\rangle$
  $\land$ $UpdateRecorder(\langle$"LeaderProcessFOLLOWERINFO"$, i, j\rangle)$
```

Follower receives $LEADERINFO$. If $newEpoch \geq acceptedEpoch$, then follower updates $acceptedEpoch$ and sends $ACKEPOCH$ back, containing $currentEpoch$ and $lastProcessedZxid$. After this, $zabState$ turns to $SYNC$. See $registerWithLeader$ in Learner for details.

```
$FollowerProcessLEADERINFO(i, j) \triangleq$
  $\land$ $IsFollower(i)$
  $\land$ $PendingLEADERINFO(i, j)$
  $\land$ LET $msg \triangleq msgs[j][i][1]$
          $newEpoch \triangleq msg.mzxid[1]$
          $infoOk \triangleq IsMyLeader(i, j)$
          $epochOk \triangleq newEpoch \geq acceptedEpoch[i]$
          $stateOk \triangleq zabState[i] = DISCOVERY$
     IN   $\land$ $infoOk$
          $\land$ $\lor$  1. Normal case
                $\land$ $epochOk$
                $\land$ $\lor$ $\land$ $stateOk$
                       $\land$ $\lor$ $\land$ $newEpoch > acceptedEpoch[i]$
                              $\land$ $acceptedEpoch' = [acceptedEpoch$ EXCEPT $![i] = newEpoch]$
                              $\land$ LET $epochBytes \triangleq currentEpoch[i]$
                                      $m \triangleq [mtype \mapsto ACKEPOCH,$
                                               $mzxid \mapsto lastProcessed[i].zxid,$
                                               $mepoch \mapsto epochBytes]$
                                  IN  $Reply(i, j, m)$
                          $\lor$ $\land$ $newEpoch = acceptedEpoch[i]$
                              $\land$ LET $m \triangleq [mtype \mapsto ACKEPOCH,$
```

```
                                        $mzxid \mapsto lastProcessed[i].zxid,$
                                        $mepoch \mapsto -1]$
                                 IN  $Reply(i, j, m)$
                              $\land$ UNCHANGED $acceptedEpoch$
                           $\land zabState' = [zabState$ EXCEPT $![i] = SYNCHRONIZATION]$
                           $\land$ UNCHANGED $violatedInvariants$
                        $\lor \land \neg stateOk$
                           $\land PrintT($"Exception: Follower receives LEADERINFO," $\circ$
                              " whileZabState not DISCOVERY."$)$
                           $\land violatedInvariants' = [violatedInvariants$ EXCEPT $!.stateInconsistent =$ TRUE$]$
                           $\land Discard(j, i)$
                           $\land$ UNCHANGED $\langle acceptedEpoch, zabState\rangle$
                     $\land$ UNCHANGED $\langle varsL, leaderAddr, learners, forwarding, electing\rangle$
                  $\lor$ 2. Abnormal case - go back to election
                     $\land \neg epochOk$
                     $\land FollowerShutdown(i)$
                     $\land Clean(i, leaderAddr[i])$
                     $\land RemoveLearner(leaderAddr[i], i)$
                     $\land$ UNCHANGED $\langle acceptedEpoch, violatedInvariants\rangle$
           $\land$ UNCHANGED $\langle history, lastCommitted, connecting, ackldRecv, tempMaxEpoch,$
                         $initialHistory, proposalMsgsLog, epochLeader, packetsSync\rangle$
           $\land UpdateRecorder(\langle$"FollowerProcessLEADERINFO"$, i, j\rangle)$
```

```
RECURSIVE $UpdateAckSidHelper(\_, \_, \_, \_)$
$UpdateAckSidHelper(his, cur, end, target) \triangleq$
     IF $cur > end$ THEN $his$
      ELSE LET $curTxn \triangleq [zxid \mapsto his[1].zxid,$
                               $value \mapsto his[1].value,$
                               $ackSid \mapsto$ IF $target \in his[1].ackSid$ THEN $his[1].ackSid$
                                                 ELSE $his[1].ackSid \cup \{target\},$
                               $epoch \mapsto his[1].epoch]$
           IN  $\langle curTxn\rangle \circ UpdateAckSidHelper(Tail(his), cur + 1, end, target)$
```

There originally existed one bug in *LeaderProcessACK* when *monotonicallyInc* = FALSE, and it is we did not add *ackSid* of history in *SYNC*. So we update *ackSid* in *syncFollower*.

```
$UpdateAckSid(his, lastSeenIndex, target) \triangleq$
     IF $Len(his) = 0 \lor lastSeenIndex = 0$ THEN $his$
      ELSE $UpdateAckSidHelper(his, 1, Minimum(\{Len(his), lastSeenIndex\}), target)$
```

return -1: this *zxid* appears at least twice; $Len(his) + 1$: does not exist; 1 $\neg Len(his)$: exists and appears just once.

```
RECURSIVE $ZxidToIndexHepler(\_, \_, \_, \_)$
$ZxidToIndexHepler(his, zxid, cur, appeared) \triangleq$
     IF $cur > Len(his)$ THEN $cur$
```

```
        ELSE IF $TxnZxidEqual(his[cur], zxid)$
                THEN CASE $appeared = \text{TRUE} \rightarrow -1$
                        □    OTHER                 $\rightarrow Minimum(\{cur,$
                                 $ZxidToIndexHepler(his, zxid, cur + 1, \text{TRUE})\})$
                ELSE $ZxidToIndexHepler(his, zxid, cur + 1, appeared)$

$ZxidToIndex(his, zxid) \triangleq$ IF $ZxidEqual(zxid, \langle 0, 0 \rangle)$ THEN 0
                              ELSE IF $Len(his) = 0$ THEN 1
                                     ELSE LET $len \triangleq Len(his)$ IN
                                            IF $\exists\, idx \in 1 \,..\, len : TxnZxidEqual(his[idx], zxid)$
                                             THEN $ZxidToIndexHepler(his, zxid, 1, \text{FALSE})$
                                             ELSE $len + 1$
```

Find index *idx* which meets:
$history[idx].zxid \leq zxid < history[idx + 1].zxid$

```
RECURSIVE $IndexOfZxidHelper(\_, \_, \_, \_)$
$IndexOfZxidHelper(his, zxid, cur, end) \triangleq$
        IF $cur > end$ THEN $end$
        ELSE IF $ZxidCompare(his[cur].zxid, zxid)$ THEN $cur - 1$
                ELSE $IndexOfZxidHelper(his, zxid, cur + 1, end)$

$IndexOfZxid(his, zxid) \triangleq$ IF $Len(his) = 0$ THEN 0
                             ELSE LET $idx \triangleq ZxidToIndex(his, zxid)$
                                        $len \triangleq Len(his)$
                                  IN
                                  IF $idx \leq len$ THEN $idx$
                                   ELSE $IndexOfZxidHelper(his, zxid, 1, len)$

RECURSIVE $queuePackets(\_, \_, \_, \_, \_)$
$queuePackets(queue, his, cur, committed, end) \triangleq$
        IF $cur > end$ THEN $queue$
        ELSE CASE $cur > committed \rightarrow$
                   LET $m\_proposal \triangleq [mtype \mapsto PROPOSAL,$
                                          $mzxid \mapsto his[cur].zxid,$
                                          $mdata \mapsto his[cur].value]$
                   IN  $queuePackets(Append(queue, m\_proposal), his, cur + 1, committed, end)$
                □  $cur \leq committed \rightarrow$
                   LET $m\_proposal \triangleq [mtype \mapsto PROPOSAL,$
                                          $mzxid \mapsto his[cur].zxid,$
                                          $mdata \mapsto his[cur].value]$
                       $m\_commit \triangleq [mtype \mapsto COMMIT,$
                                          $mzxid \mapsto his[cur].zxid]$
                       $newQueue \triangleq queue \circ \langle m\_proposal, m\_commit \rangle$
                   IN  $queuePackets(newQueue, his, cur + 1, committed, end)$

RECURSIVE $setPacketsForChecking(\_, \_, \_, \_, \_, \_)$
```

```
$setPacketsForChecking(set, src, ep, his, cur, end) \triangleq$
    IF $cur > end$ THEN $set$
     ELSE LET $m\_proposal \triangleq [source \mapsto src,$
                                   $epoch \mapsto ep,$
                                   $zxid \mapsto his[cur].zxid,$
                                   $data \mapsto his[cur].value]$
          IN  $setPacketsForChecking((set \cup \{m\_proposal\}), src, ep, his, cur + 1, end)$
```

See *queueCommittedProposals* in *LearnerHandler* and *startForwarding* in Leader for details. For proposals in *committedLog* and *toBeApplied*, send $< PROPOSAL, COMMIT >$. For proposals in *outstandingProposals*, send *PROPOSAL* only.

```
$StartForwarding(i, j, lastSeenZxid, lastSeenIndex, mode, needRemoveHead) \triangleq$
    $\land$ LET $lastCommittedIndex \triangleq$ IF $zabState[i] = BROADCAST$
                                              THEN $lastCommitted[i].index$
                                              ELSE $Len(initialHistory[i])$
            $lastProposedIndex \triangleq Len(history[i])$
            $queue\_origin \triangleq$ IF $lastSeenIndex \geq lastProposedIndex$
                                    THEN $\langle\rangle$
                                    ELSE $queuePackets(\langle\rangle, history[i],$
                                           $lastSeenIndex + 1, lastCommittedIndex,$
                                           $lastProposedIndex)$
            $set\_forChecking \triangleq$ IF $lastSeenIndex \geq lastProposedIndex$
                                       THEN $\{\}$
                                       ELSE $setPacketsForChecking(\{\}, i,$
                                              $acceptedEpoch[i], history[i],$
                                              $lastSeenIndex + 1, lastProposedIndex)$
            $m\_trunc \triangleq [mtype \mapsto TRUNC, mtruncZxid \mapsto lastSeenZxid]$
            $m\_diff \triangleq [mtype \mapsto DIFF, mzxid \mapsto lastSeenZxid]$
            $newLeaderZxid \triangleq \langle acceptedEpoch[i], 0\rangle$
            $m\_newleader \triangleq [mtype \mapsto NEWLEADER,$
                                  $mzxid \mapsto newLeaderZxid]$
            $queue\_toSend \triangleq$ CASE $mode = TRUNC \rightarrow (\langle m\_trunc\rangle \circ queue\_origin) \circ \langle m\_newleader\rangle$
                                    $\Box$ OTHER $\rightarrow (\langle m\_diff\rangle \circ queue\_origin) \circ \langle m\_newleader\rangle$
       IN  $\land$ $\lor$ $\land$ $needRemoveHead$
                    $\land$ $DiscardAndSendPackets(i, j, queue\_toSend)$
                 $\lor$ $\land$ $\neg needRemoveHead$
                    $\land$ $SendPackets(i, j, queue\_toSend)$
             $\land$ $proposalMsgsLog' = proposalMsgsLog \cup set\_forChecking$
    $\land$ $forwarding' = [forwarding$ EXCEPT $![i] = @ \cup \{j\}]$
    $\land$ $history' = [history$ EXCEPT $![i] = UpdateAckSid(@, lastSeenIndex, j)]$
```

Leader syncs with follower using *DIFF/TRUNC/PROPOSAL/COMMIT*... See *syncFollower* in *LearnerHandler* for details.

```
$SyncFollower(i, j, peerLastZxid, needRemoveHead) \triangleq$
    LET $IsPeerNewEpochZxid \triangleq peerLastZxid[2] = 0$
        $lastProcessedZxid \triangleq lastProcessed[i].zxid$
```

```
$maxCommittedLog \triangleq$ IF $zabState[i] = BROADCAST$
                             THEN $lastCommitted[i].zxid$
                             ELSE LET $totalLen \triangleq Len(initialHistory[i])$
                                  IN   IF $totalLen = 0$ THEN $\langle 0, 0\rangle$
                                           ELSE $history[i][totalLen].zxid$

    Hypothesis: 1. $minCommittedLog$ : $zxid$ of head of history, so no SNAP.
                2. $maxCommittedLog = lastCommitted$, to compress state space.
                3. merge $queueCommittedProposals$,$startForwarding$ and
                   sending $NEWLEADER$ into $StartForwarding$.

IN  $\lor$ $case1.\ peerLastZxid = lastProcessedZxid$
         $DIFF + StartForwarding(lastProcessedZxid)$
      $\land ZxidEqual(peerLastZxid, lastProcessedZxid)$
      $\land StartForwarding(i, j, peerLastZxid, lastProcessed[i].index,$
                              $DIFF, needRemoveHead)$
    $\lor$ $\land \neg ZxidEqual(peerLastZxid, lastProcessedZxid)$
      $\land$ $\lor$ $case2.\ peerLastZxid > maxCommittedLog$
               $TRUNC + StartForwarding(maxCommittedLog)$
           $\land ZxidCompare(peerLastZxid, maxCommittedLog)$
           $\land$ LET $maxCommittedIndex \triangleq$ IF $zabState[i] = BROADCAST$
                                                THEN $lastCommitted[i].index$
                                                ELSE $Len(initialHistory[i])$
              IN   $StartForwarding(i, j, maxCommittedLog, maxCommittedIndex,$
                                    $TRUNC, needRemoveHead)$
         $\lor$ $case3.\ minCommittedLog \leq peerLastZxid \leq maxCommittedLog$
           $\land \neg ZxidCompare(peerLastZxid, maxCommittedLog)$
           $\land$ LET $lastSeenIndex \triangleq ZxidToIndex(history[i], peerLastZxid)$
                  $exist \triangleq$ $\land lastSeenIndex \geq 0$
                           $\land lastSeenIndex \leq Len(history[i])$
                  $lastIndex \triangleq$ IF $exist$ THEN $lastSeenIndex$
                                  ELSE $IndexOfZxid(history[i], peerLastZxid)$
                   Maximum $zxid$ that $< peerLastZxid$
                  $lastZxid \triangleq$ IF $exist$ THEN $peerLastZxid$
                                  ELSE IF $lastIndex = 0$ THEN $\langle 0, 0\rangle$
                                           ELSE $history[i][lastIndex].zxid$
              IN
               $\lor$ case 3.1. $peerLastZxid$ exists in history
                        $DIFF + StartForwarding$
                  $\land exist$
                  $\land StartForwarding(i, j, peerLastZxid, lastSeenIndex,$
                                          $DIFF, needRemoveHead)$
               $\lor$ case 3.2. $peerLastZxid$ does not exist in history
                        $TRUNC + StartForwarding$
                  $\land \neg exist$
```

```
                    $\land StartForwarding(i, j, lastZxid, lastIndex,$
                                          $TRUNC, needRemoveHead)$
                we will not have case 4 where $peerLastZxid < minCommittedLog$, because
                $minCommittedLog$ default value is 1 in our spec.
```

compare state summary of two servers

```
$IsMoreRecentThan(ss1, ss2) \triangleq \lor ss1.currentEpoch > ss2.currentEpoch$
                              $\lor \land ss1.currentEpoch = ss2.currentEpoch$
                                 $\land ZxidCompare(ss1.lastZxid, ss2.lastZxid)$
```

electionFinished in Leader

```
$ElectionFinished(i, set) \triangleq \land i \in set$
                            $\land IsQuorum(set)$
```

There may exist some follower shuts down and connects again, while it has sent *ACKEPOCH* or updated *currentEpoch* last time. This means *sid* of this follower has existed in *elecingFollower* but its *info* is old. So we need to make sure each *sid* in *electingFollower* is unique and *latest*(*newest*).

```
$UpdateElecting(oldSet, sid, peerLastZxid, inQuorum) \triangleq$
    LET $sid\_electing \triangleq \{s.sid : s \in oldSet\}$
    IN  IF $sid \in sid\_electing$
          THEN LET $old\_info \triangleq$ CHOOSE $info \in oldSet : info.sid = sid$
                   $follower\_info \triangleq$
                        $[sid \mapsto sid,$
                         $peerLastZxid \mapsto peerLastZxid,$
                         $inQuorum \mapsto (inQuorum \lor old\_info.inQuorum)]$
               IN  $(oldSet \setminus \{old\_info\}) \cup \{follower\_info\}$
          ELSE LET $follower\_info \triangleq$
                        $[sid \mapsto sid,$
                         $peerLastZxid \mapsto peerLastZxid,$
                         $inQuorum \mapsto inQuorum]$
               IN  $oldSet \cup \{follower\_info\}$

$LeaderTurnToSynchronization(i) \triangleq$
    $\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = acceptedEpoch[i]]$
    $\land zabState'     = [zabState$     EXCEPT $![i] = SYNCHRONIZATION]$
```

Leader waits for receiving *ACKEPOPCH* from a quorum, and check whether it has most recent state summary from them. After this, leader's *zabState* turns to *SYNCHRONIZATION*. See *waitForEpochAck* in Leader for details.

```
$LeaderProcessACKEPOCH(i, j) \triangleq$
    $\land IsLeader(i)$
    $\land PendingACKEPOCH(i, j)$
    $\land$ LET $msg \triangleq msgs[j][i][1]$
            $infoOk \triangleq IsMyLearner(i, j)$
            $leaderStateSummary \triangleq [currentEpoch \mapsto currentEpoch[i],$
```

```
                                    lastZxid        ↦ lastProcessed[i].zxid]
    followerStateSummary ≜ [currentEpoch ↦ msg.mepoch,
                                    lastZxid        ↦ msg.mzxid]
    logOk ≜  whether follower is no more up-to-date than leader
              ¬IsMoreRecentThan(followerStateSummary, leaderStateSummary)
    electing_quorum ≜ {e ∈ electing[i] : e.inQuorum = TRUE}
    sid_electing ≜ {s.sid : s ∈ electing_quorum}
IN  ∧ infoOk
    ∧ ∨  electionFinished = true, jump ouf of waitForEpochAck.
         Different from code, here we still need to record info
         into electing, to help us perform syncFollower afterwards.
         Since electing already meets quorum, it does not break
         consistency between code and spec.
       ∧ ElectionFinished(i, sid_electing)
       ∧ electing' = [electing EXCEPT ![i] = UpdateElecting(@, j, msg.mzxid, FALSE)]
       ∧ Discard(j, i)
       ∧ UNCHANGED ⟨varsL, zabState, forwarding, leaderAddr,
                         learners, epochLeader, violatedInvariants⟩
      ∨ ∧ ¬ElectionFinished(i, sid_electing)
         ∧ ∨ ∧ zabState[i] = DISCOVERY
               ∧ UNCHANGED violatedInvariants
            ∨ ∧ zabState[i] ≠ DISCOVERY
               ∧ PrintT("Exception: electionFinished false," ∘
                  " while zabState not DISCOVERY.")
               ∧ violatedInvariants' = [violatedInvariants EXCEPT
                                             !.stateInconsistent = TRUE]
         ∧ ∨ ∧ followerStateSummary.currentEpoch = −1
               ∧ electing' = [electing EXCEPT ![i]       = UpdateElecting(@, j,
                                                            msg.mzxid, FALSE)]
               ∧ Discard(j, i)
               ∧ UNCHANGED ⟨varsL, zabState, forwarding, leaderAddr,
                                 learners, epochLeader⟩
            ∨ ∧ followerStateSummary.currentEpoch > −1
               ∧ ∨  normal follower
                     ∧ logOk
                     ∧ electing' = [electing EXCEPT ![i] =
                                    UpdateElecting(@, j, msg.mzxid, TRUE)]
                     ∧ LET new_electing_quorum ≜ {e ∈ electing'[i] : e.inQuorum = TRUE}
                            new_sid_electing ≜ {s.sid : s ∈ new_electing_quorum}
                        IN
                        ∨  electionFinished = true, jump out of waitForEpochAck,
                           update currentEpoch and zabState.
                           ∧ ElectionFinished(i, new_sid_electing)
                           ∧ LeaderTurnToSynchronization(i)
                           ∧ LET newLeaderEpoch ≜ acceptedEpoch[i]
```

```
                                    IN   $epochLeader' = [epochLeader$ EXCEPT $![newLeaderEpoch]$
                                              $= @ \cup \{i\}]$  for checking invariants
                                 $\vee$  there still exists $electionFinished =$ false.
                                   $\wedge \neg ElectionFinished(i, new\_sid\_electing)$
                                   $\wedge$ UNCHANGED $\langle currentEpoch, zabState, epochLeader \rangle$
                               $\wedge Discard(j, i)$
                               $\wedge$ UNCHANGED $\langle state, lastProcessed, electionVars, leadingVoteSet,$
                                                $electionMsgs, leaderAddr, learners, history, forwarding \rangle$
                            $\vee$  Exists follower more recent than leader
                               $\wedge \neg logOk$
                               $\wedge LeaderShutdown(i)$
                               $\wedge$ UNCHANGED $\langle electing, epochLeader \rangle$
    $\wedge$ UNCHANGED $\langle acceptedEpoch, lastCommitted, connecting, ackldRecv,$
                 $tempMaxEpoch, initialHistory, packetsSync, proposalMsgsLog \rangle$
    $\wedge UpdateRecorder(\langle$ "LeaderProcessACKEPOCH", $i, j \rangle)$

Strip $syncFollower$ from $LeaderProcessACKEPOCH$.
Only when $electionFinished =$ true and there exists some
$learnerHandler$ has not perform $syncFollower$, this
action will be called.
$LeaderSyncFollower(i) \triangleq$
    $\wedge IsLeader(i)$
    $\wedge$ LET $electing\_quorum \triangleq \{e \in electing[i] : e.inQuorum = \text{TRUE}\}$
             $electionFinished \triangleq ElectionFinished(i, \{s.sid : s \in electing\_quorum\})$
             $toSync \triangleq \{s \in electing[i] : \wedge \neg ZxidEqual(s.peerLastZxid, \langle -1, -1 \rangle)$
                                          $\wedge s.sid \in learners[i]\}$
             $canSync \triangleq toSync \neq \{\}$
       IN
       $\wedge electionFinished$
       $\wedge canSync$
       $\wedge$ LET $chosen \triangleq$ CHOOSE $s \in toSync :$ TRUE
                $newChosen \triangleq [sid \mapsto chosen.sid,$
                                $peerLastZxid \mapsto \langle -1, -1 \rangle,$   $\langle -1, -1 \rangle$ means has handled.
                                $inQuorum \mapsto chosen.inQuorum]$
          IN   $\wedge SyncFollower(i, chosen.sid, chosen.peerLastZxid,$ FALSE$)$
               $\wedge electing' = [electing$ EXCEPT $![i] = (@ \setminus \{chosen\}) \cup \{newChosen\}]$
    $\wedge$ UNCHANGED $\langle state, currentEpoch, lastProcessed, zabState, acceptedEpoch,$
               $lastCommitted, initialHistory, electionVars, leadingVoteSet,$
               $learners, connecting, ackldRecv, tempMaxEpoch, followerVars,$
               $epochLeader, violatedInvariants, electionMsgs \rangle$
    $\wedge UpdateRecorder(\langle$ "LeaderSyncFollower", $i \rangle)$

$TruncateLog(his, index) \triangleq$ IF $index \leq 0$ THEN $\langle \rangle$
                                ELSE $SubSeq(his, 1, index)$
```

Follower receives *DIFF*/*TRUNC*, and then may receives *PROPOSAL*,*COMMIT*,*NEWLEADER*, and *UPTODATE*. See *syncWithLeader* in Learner for details.

```
$FollowerProcessSyncMessage(i, j) \triangleq$
        $\land IsFollower(i)$
        $\land msgs[j][i] \neq \langle\rangle$
        $\land msgs[j][i][1].mtype = DIFF \lor msgs[j][i][1].mtype = TRUNC$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
              $infoOk \triangleq IsMyLeader(i, j)$
              $stateOk \triangleq zabState[i] = SYNCHRONIZATION$
          IN  $\land infoOk$
              $\land \lor$  Follower should receive packets in $SYNC$.
                   $\land \neg stateOk$
                   $\land PrintT($"Exception: Follower receives DIFF/TRUNC," $\circ$
                            " whileZabState not SYNCHRONIZATION."$)$
                   $\land violatedInvariants' = [violatedInvariants$ EXCEPT $!.stateInconsistent =$ TRUE$]$
                   $\land$ UNCHANGED $\langle history, initialHistory, lastProcessed, lastCommitted\rangle$
                 $\lor \land stateOk$
                   $\land \lor \land msg.mtype = DIFF$
                           $\land$ UNCHANGED $\langle history, initialHistory, lastProcessed, lastCommitted,$
                                     $violatedInvariants\rangle$
                      $\lor \land msg.mtype = TRUNC$
                           $\land$ LET $truncZxid \triangleq msg.mtruncZxid$
                                 $truncIndex \triangleq ZxidToIndex(history[i], truncZxid)$
                             IN
                             $\lor \land truncIndex > Len(history[i])$
                                $\land PrintT($"Exception: TRUNC error."$)$
                                $\land violatedInvariants' = [violatedInvariants$ EXCEPT
                                                $!.proposalInconsistent =$ TRUE$]$
                                $\land$ UNCHANGED $\langle history, initialHistory, lastProcessed, lastCommitted\rangle$
                             $\lor \land truncIndex \leq Len(history[i])$
                                $\land history' = [history$ EXCEPT
                                                  $![i] = TruncateLog(history[i], truncIndex)]$
                                $\land initialHistory' = [initialHistory$ EXCEPT $![i] = history'[i]]$
                                $\land lastProcessed'\ \ = [lastProcessed$ EXCEPT
                                                    $![i] = [index \mapsto truncIndex,$
                                                             $zxid\ \ \mapsto truncZxid]]$
                                $\land lastCommitted' = [lastCommitted$ EXCEPT
                                                    $![i] = [index \mapsto truncIndex,$
                                                             $zxid\ \ \mapsto truncZxid]]$
                                $\land$ UNCHANGED $violatedInvariants$
        $\land Discard(j, i)$
        $\land$ UNCHANGED $\langle state, currentEpoch, zabState, acceptedEpoch, electionVars,$
                        $leaderVars, tempMaxEpoch, followerVars,$
                        $proposalMsgsLog, epochLeader, electionMsgs\rangle$
        $\land$ $UpdateRecorder(\langle$"FollowerProcessSyncMessage", $i, j\rangle)$

See $lastProposed$ in Leader for details.
$LastProposed(i) \triangleq$ IF $Len(history[i]) = 0$ THEN $[index \mapsto 0,$
                                                      $zxid \mapsto \langle 0, 0\rangle]$
                          ELSE
                          LET $lastIndex \triangleq Len(history[i])$
                              $entry \triangleq history[i][lastIndex]$
                          IN  $[index \mapsto lastIndex,$
                               $zxid \mapsto entry.zxid]$

See $lastQueued$ in Learner for details.
$LastQueued(i) \triangleq$ IF $\neg IsFollower(i) \lor zabState[i] \neq SYNCHRONIZATION$
                        THEN $LastProposed(i)$
                        ELSE  condition: $IsFollower(i) \land zabState = SYNCHRONIZATION$
                              LET $packetsInSync \triangleq packetsSync[i].notCommitted$
                                  $lenSync \triangleq Len(packetsInSync)$
                                  $totalLen \triangleq Len(history[i]) + lenSync$
                              IN  IF $lenSync = 0$ THEN $LastProposed(i)$
                                   ELSE $[index \mapsto totalLen,$
                                         $zxid \mapsto packetsInSync[lenSync].zxid]$

$IsNextZxid(curZxid, nextZxid) \triangleq$
            $\lor$ first $PROPOSAL$ in this epoch
              $\land$ $nextZxid[2] = 1$
              $\land$ $curZxid[1] < nextZxid[1]$
            $\lor$ not first $PROPOSAL$ in this epoch
              $\land$ $nextZxid[2] > 1$
              $\land$ $curZxid[1] = nextZxid[1]$
              $\land$ $curZxid[2] + 1 = nextZxid[2]$

$FollowerProcessPROPOSALInSync(i, j) \triangleq$
        $\land$ $IsFollower(i)$
        $\land$ $PendingPROPOSAL(i, j)$
        $\land$ $zabState[i] = SYNCHRONIZATION$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
                $infoOk \triangleq IsMyLeader(i, j)$
                $isNext \triangleq IsNextZxid(LastQueued(i).zxid, msg.mzxid)$
                $newTxn \triangleq [zxid \mapsto msg.mzxid,$
                                  $value \mapsto msg.mdata,$
                                  $ackSid \mapsto \{\},$       follower do not consider $ackSid$
                                  $epoch \mapsto acceptedEpoch[i]]$  epoch of this round
          IN  $\land$ $infoOk$
              $\land$ $\lor$ $\land$ $isNext$
                      $\land$ $packetsSync' = [packetsSync$ EXCEPT $![i].notCommitted$
                                $= Append(packetsSync[i].notCommitted, newTxn)]$
          $\lor$ $\land$ $\neg isNext$
             $\land$ $PrintT($"Warn: Follower receives PROPOSAL," $\circ$
                 " while zxid != lastQueued + 1."$)$
             $\land$ UNCHANGED $packetsSync$
        $logRequest \rightarrow SyncRequestProcessor \rightarrow SendAckRequestProcessor \rightarrow$ reply ack
        So here we do not need to send ack to leader.
        $\land$ $Discard(j, i)$
        $\land$ UNCHANGED $\langle serverVars, electionVars, leaderVars, leaderAddr,$
                        $verifyVars, electionMsgs\rangle$
        $\land$ $UpdateRecorder(\langle$"FollowerProcessPROPOSALInSync"$, i, j\rangle)$

RECURSIVE $IndexOfFirstTxnWithEpoch(\_, \_, \_, \_)$
$IndexOfFirstTxnWithEpoch(his, epoch, cur, end) \triangleq$
        IF $cur > end$ THEN $cur$
         ELSE IF $his[cur].epoch = epoch$ THEN $cur$
              ELSE $IndexOfFirstTxnWithEpoch(his, epoch, cur + 1, end)$

$LastCommitted(i) \triangleq$ IF $zabState[i] = BROADCAST$ THEN $lastCommitted[i]$
                  ELSE CASE $IsLeader(i)$ $\rightarrow$
                            LET $lastInitialIndex \triangleq Len(initialHistory[i])$
                            IN  IF $lastInitialIndex = 0$ THEN $[index \mapsto 0,$
                                                                $zxid \mapsto \langle 0, 0\rangle]$
                                ELSE $[index \mapsto lastInitialIndex,$
                                       $zxid \mapsto history[i][lastInitialIndex].zxid]$
                       $\Box$ $IsFollower(i) \rightarrow$
                            LET $completeHis \triangleq history[i] \circ packetsSync[i].notCommitted$
                                $packetsCommitted \triangleq packetsSync[i].committed$
                                $lenCommitted \triangleq Len(packetsCommitted)$
                            IN  IF $lenCommitted = 0$ return last one in initial history
                                THEN LET $lastInitialIndex \triangleq Len(initialHistory[i])$
                                     IN  IF $lastInitialIndex = 0$
                                         THEN $[index \mapsto 0,$
                                                $zxid \mapsto \langle 0, 0\rangle]$
                                         ELSE $[index \mapsto lastInitialIndex,$
                                                $zxid \mapsto completeHis[lastInitialIndex].zxid]$
                                ELSE                return tail of $packetsCommitted$
                                   LET $committedIndex \triangleq ZxidToIndex(completeHis,$
                                                        $packetsCommitted[lenCommitted])$
                                   IN  $[index \mapsto committedIndex,$
                                        $zxid \mapsto packetsCommitted[lenCommitted]]$
                       $\Box$ OTHER $\rightarrow lastCommitted[i]$

$TxnWithIndex(i, idx) \triangleq$ IF $\neg IsFollower(i) \lor zabState[i] \neq SYNCHRONIZATION$
                       THEN $history[i][idx]$
                       ELSE LET $completeHis \triangleq history[i] \circ packetsSync[i].notCommitted$
                            IN  $completeHis[idx]$
```

To simplify specification, we assume *snapshotNeeded* = false and *writeToTxnLog* = true. So here we just call *packetsCommitted.add*.

```
$FollowerProcessCOMMITInSync(i, j) \triangleq$
        $\land\ IsFollower(i)$
        $\land\ PendingCOMMIT(i, j)$
        $\land\ zabState[i] = SYNCHRONIZATION$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
              $infoOk \triangleq IsMyLeader(i, j)$
              $committedIndex \triangleq LastCommitted(i).index + 1$
              $exist \triangleq \land\ committedIndex \leq LastQueued(i).index$
                        $\land\ IsNextZxid(LastCommitted(i).zxid,\ msg.mzxid)$
              $match \triangleq ZxidEqual(msg.mzxid,\ TxnWithIndex(i,\ committedIndex).zxid)$
          IN  $\land\ infoOk$
              $\land \lor \land\ exist$
                    $\land \lor \land\ match$
                          $\land\ packetsSync' = [packetsSync$ EXCEPT $![i].committed$
                                 $= Append(packetsSync[i].committed,\ msg.mzxid)]$
                          $\land$ UNCHANGED $violatedInvariants$
                       $\lor \land \neg match$
                          $\land\ PrintT($"Warn: Follower receives COMMIT," $\circ$
                               " but zxid not the next committed zxid in COMMIT."$)$
                          $\land\ violatedInvariants' = [violatedInvariants$ EXCEPT
                                    $!.commitInconsistent =$ TRUE$]$
                          $\land$ UNCHANGED $packetsSync$
                 $\lor \land \neg exist$
                    $\land\ PrintT($"Warn: Follower receives COMMIT," $\circ$
                         " but no packets with its zxid exists."$)$
                    $\land\ violatedInvariants' = [violatedInvariants$ EXCEPT
                              $!.commitInconsistent =$ TRUE$]$
                    $\land$ UNCHANGED $packetsSync$
        $\land\ Discard(j, i)$
        $\land$ UNCHANGED $\langle serverVars,\ electionVars,\ leaderVars,$
                    $leaderAddr,\ epochLeader,\ proposalMsgsLog,\ electionMsgs\rangle$
        $\land\ UpdateRecorder(\langle$"FollowerProcessCOMMITInSync"$, i, j\rangle)$

RECURSIVE $ACKInBatches(\_, \_)$
$ACKInBatches(queue, packets) \triangleq$
        IF $packets = \langle\rangle$ THEN $queue$
         ELSE LET $head \triangleq packets[1]$
                  $newPackets \triangleq Tail(packets)$
                  $m\_ack \triangleq [mtype \mapsto ACK,$
                                $mzxid \mapsto head.zxid]$
              IN  $ACKInBatches(Append(queue,\ m\_ack),\ newPackets)$
```

Update *currentEpoch*, and *logRequest* every packets in *packetsNotCommitted* and clear it. As *syncProcessor* will be called in *logRequest*, we have to reply acks here.

```
$FollowerProcessNEWLEADER(i, j) \triangleq$
    $\land IsFollower(i)$
    $\land PendingNEWLEADER(i, j)$
    $\land$ LET $msg \triangleq msgs[j][i][1]$
            $infoOk \triangleq IsMyLeader(i, j)$
            $packetsInSync \triangleq packetsSync[i].notCommitted$
            $m\_ackld \triangleq [mtype \mapsto ACKLD,$
                            $mzxid \mapsto msg.mzxid]$
            $ms\_ack \triangleq ACKInBatches(\langle\rangle, packetsInSync)$
            $queue\_toSend \triangleq \langle m\_ackld\rangle \circ ms\_ack$   send $ACK - NEWLEADER$ first.
       IN   $\land infoOk$
            $\land currentEpoch' = [currentEpoch$ EXCEPT $![i] = acceptedEpoch[i]]$
            $\land history' \quad = [history$ EXCEPT $![i] = @ \circ packetsInSync]$
            $\land packetsSync' = [packetsSync$ EXCEPT $![i].notCommitted = \langle\rangle]$
            $\land DiscardAndSendPackets(i, j, queue\_toSend)$
    $\land$ UNCHANGED $\langle state, lastProcessed, zabState, acceptedEpoch, lastCommitted,$
                    $electionVars, leaderVars, initialHistory, leaderAddr, verifyVars,$
                    $electionMsgs\rangle$
    $\land UpdateRecorder(\langle$"FollowerProcessNEWLEADER"$, i, j\rangle)$
```

quorumFormed in Leader

$QuorumFormed(i) \triangleq i \in ackldRecv[i] \land IsQuorum(ackldRecv[i])$
$QuorumFormedTurnToTrue(i) \triangleq i \in ackldRecv'[i] \land IsQuorum(ackldRecv'[i])$

$UpdateElectionVote(i, epoch) \triangleq UpdateProposal(i, currentVote[i].proposedLeader,$
$\qquad currentVote[i].proposedZxid, epoch)$

See *startZkServer* in Leader for details.

```
$StartZkServer(i) \triangleq$
    LET $latest \triangleq LastProposed(i)$
    IN  $\land lastCommitted' = [lastCommitted$ EXCEPT $![i] = latest]$
        $\land lastProcessed' \ = [lastProcessed$ EXCEPT $![i] = latest]$
        $\land UpdateElectionVote(i, acceptedEpoch[i])$

$LeaderTurnToBroadcast(i) \triangleq$
    $\land StartZkServer(i)$
    $\land zabState' = [zabState$ EXCEPT $![i] = BROADCAST]$
```

Leader waits for receiving quorum of *ACK* whose lower bits of *zxid* is 0, and broadcasts *UPTODATE*. See *waitForNewLeaderAck* for details.

```
$LeaderProcessACKLD(i, j) \triangleq$
    $\land IsLeader(i)$
    $\land PendingACKLD(i, j)$
    $\land$ LET $msg \quad \triangleq msgs[j][i][1]$
            $infoOk \triangleq IsMyLearner(i, j)$
            $match \ \triangleq ZxidEqual(msg.mzxid, \langle acceptedEpoch[i], 0\rangle)$
            currentZxid  ≜ ⟨acceptedEpoch[i], 0⟩
            m_uptodate   ≜ [mtype ↦ UPTODATE,
                            mzxid ↦ currentZxid]   not important
        IN  ∧ infoOk
            ∧ ∨  just reply UPTODATE.
                 ∧ QuorumFormed(i)
                 ∧ Reply(i, j, m_uptodate)
                 ∧ UNCHANGED ⟨ackldRecv, zabState, lastCommitted, lastProcessed,
                              currentVote, violatedInvariants⟩
              ∨ ∧ ¬QuorumFormed(i)
                ∧ ∨ ∧ match
                    ∧ ackldRecv' = [ackldRecv EXCEPT ![i] = @ ∪ {j}]
                    ∧ ∨  jump out of waitForNewLeaderAck, and do startZkServer,
                         setZabState, and reply UPTODATE.
                         ∧ QuorumFormedTurnToTrue(i)
                         ∧ LeaderTurnToBroadcast(i)
                         ∧ DiscardAndBroadcastUPTODATE(i, j, m_uptodate)
                      ∨  still wait in waitForNewLeaderAck.
                         ∧ ¬QuorumFormedTurnToTrue(i)
                         ∧ Discard(j, i)
                         ∧ UNCHANGED ⟨zabState, lastCommitted, lastProcessed, currentVote⟩
                    ∧ UNCHANGED violatedInvariants
                  ∨ ∧ ¬match
                    ∧ PrintT("Exception: NEWLEADER ACK is from a different epoch. ")
                    ∧ violatedInvariants' = [violatedInvariants EXCEPT
                                   !.ackInconsistent = TRUE]
                    ∧ Discard(j, i)
                    ∧ UNCHANGED ⟨ackldRecv, zabState, lastCommitted,
                                 lastProcessed, currentVote⟩
    ∧ UNCHANGED ⟨state, currentEpoch, acceptedEpoch, history, logicalClock, receiveVotes,
                outOfElection, recvQueue, waitNotmsg, leadingVoteSet, learners, connecting,
                electing, forwarding, tempMaxEpoch, initialHistory, followerVars,
                proposalMsgsLog, epochLeader, electionMsgs⟩
    ∧ UpdateRecorder(⟨"LeaderProcessACKLD", i, j⟩)

TxnsWithPreviousEpoch(i) ≜
        LET completeHis ≜ IF ¬IsFollower(i) ∨ zabState[i] ≠ SYNCHRONIZATION
                            THEN history[i]
                            ELSE history[i] ∘ packetsSync[i].notCommitted
            end     ≜ Len(completeHis)
            first   ≜ IndexOfFirstTxnWithEpoch(completeHis, acceptedEpoch[i], 1, end)
        IN  IF first > end THEN completeHis
             ELSE SubSeq(completeHis, 1, first − 1)

TxnsRcvWithCurEpoch(i) ≜
```

```
    LET $completeHis \triangleq$ IF $\neg IsFollower(i) \lor zabState[i] \neq SYNCHRONIZATION$
                                  THEN $history[i]$
                                  ELSE $history[i] \circ packetsSync[i].notCommitted$
        $end \triangleq Len(completeHis)$
        $first \triangleq IndexOfFirstTxnWithEpoch(completeHis, acceptedEpoch[i], 1, end)$
    IN  IF $first > end$ THEN $\langle\rangle$
         ELSE $SubSeq(completeHis, first, end)$  $completeHis[first : end]$
```

Txns received in current epoch but not committed.
See *pendingTxns* in *FollowerZooKeeper* for details.

```
$PendingTxns(i) \triangleq$ IF $\neg IsFollower(i) \lor zabState[i] \neq SYNCHRONIZATION$
                      THEN $SubSeq(history[i], lastCommitted[i].index + 1, Len(history[i]))$
                      ELSE LET $packetsCommitted \triangleq packetsSync[i].committed$
                               $completeHis \triangleq history[i] \circ packetsSync[i].notCommitted$
                           IN  IF $Len(packetsCommitted) = 0$
                                THEN $SubSeq(completeHis, Len(initialHistory[i]) + 1, Len(completeHis))$
                                ELSE $SubSeq(completeHis, LastCommitted(i).index + 1, Len(completeHis))$

$CommittedTxns(i) \triangleq$ IF $\neg IsFollower(i) \lor zabState[i] \neq SYNCHRONIZATION$
                        THEN $SubSeq(history[i], 1, lastCommitted[i].index)$
                        ELSE LET $packetsCommitted \triangleq packetsSync[i].committed$
                                 $completeHis \triangleq history[i] \circ packetsSync[i].notCommitted$
                             IN  IF $Len(packetsCommitted) = 0$ THEN $initialHistory[i]$
                                  ELSE $SubSeq(completeHis, 1, LastCommitted(i).index)$
```

Each *zxid* of *packetsCommitted* equals to *zxid* of
corresponding *txn* in *txns*.

```
RECURSIVE $TxnsAndCommittedMatch(\_, \_)$
$TxnsAndCommittedMatch(txns, packetsCommitted) \triangleq$
    LET $len1 \triangleq Len(txns)$
        $len2 \triangleq Len(packetsCommitted)$
    IN  IF $len2 = 0$ THEN TRUE
         ELSE IF $len1 < len2$ THEN FALSE
               ELSE $\land ZxidEqual(txns[len1].zxid, packetsCommitted[len2])$
                    $\land TxnsAndCommittedMatch(SubSeq(txns, 1, len1 - 1),$
                                                 $SubSeq(packetsCommitted, 1, len2 - 1))$

$FollowerLogRequestInBatches(i, leader, ms\_ack, packetsNotCommitted) \triangleq$
    $\land history' = [history$ EXCEPT $![i] = @ \circ packetsNotCommitted]$
    $\land DiscardAndSendPackets(i, leader, ms\_ack)$
```

Since commit will call *commitProcessor.commit*, which will finally
update *lastProcessed*, we update it here atomically.

```
$FollowerCommitInBatches(i) \triangleq$
    LET $committedTxns \triangleq CommittedTxns(i)$
        $packetsCommitted \triangleq packetsSync[i].committed$
        $match \triangleq TxnsAndCommittedMatch(committedTxns, packetsCommitted)$
    IN
    $\lor \land match$
       $\land lastCommitted' = [lastCommitted$ EXCEPT $![i] = LastCommitted(i)]$
       $\land lastProcessed' = [lastProcessed$ EXCEPT $![i] = lastCommitted'[i]]$
       $\land$ UNCHANGED $violatedInvariants$
    $\lor \land \neg match$
       $\land PrintT($"Warn: Committing zxid withou see txn. /" $\circ$
            "Committing zxid != pending txn zxid."$)$
       $\land violatedInvariants' = [violatedInvariants$ EXCEPT
                $!.commitInconsistent =$ TRUE$]$
       $\land$ UNCHANGED $\langle lastCommitted, lastProcessed \rangle$
```

Follower jump out of *outerLoop* here, and log the stuff that came in between snapshot and uptodate, which means calling *logRequest* and commit to clear *packetsNotCommitted* and *packetsCommitted*.

```
$FollowerProcessUPTODATE(i, j) \triangleq$
    $\land IsFollower(i)$
    $\land PendingUPTODATE(i, j)$
    $\land$ LET $msg \triangleq msgs[j][i][1]$
           $infoOk \triangleq IsMyLeader(i, j)$
           $packetsNotCommitted \triangleq packetsSync[i].notCommitted$
           $ms\_ack \triangleq ACKInBatches(\langle\rangle, packetsNotCommitted)$
       IN  $\land infoOk$
            Here we ignore ack of UPTODATE.
           $\land UpdateElectionVote(i, acceptedEpoch[i])$
           $\land FollowerLogRequestInBatches(i, j, ms\_ack, packetsNotCommitted)$
           $\land FollowerCommitInBatches(i)$
           $\land packetsSync' = [packetsSync$ EXCEPT $![i].notCommitted = \langle\rangle,$
                                                    $![i].committed = \langle\rangle]$
           $\land zabState' = [zabState$ EXCEPT $![i] = BROADCAST]$
    $\land$ UNCHANGED $\langle state, currentEpoch, acceptedEpoch, logicalClock,$
           $receiveVotes, outOfElection, recvQueue, waitNotmsg, leaderVars,$
           $initialHistory, leaderAddr, epochLeader, proposalMsgsLog, electionMsgs\rangle$
    $\land UpdateRecorder(\langle$"FollowerProcessUPTODATE"$, i, j\rangle)$
```

```
$IncZxid(s, zxid) \triangleq$ IF $currentEpoch[s] = zxid[1]$ THEN $\langle zxid[1], zxid[2] + 1\rangle$
                          ELSE $\langle currentEpoch[s], 1\rangle$
```

Leader receives client propose and broadcasts *PROPOSAL*. See *processRequest* in *ProposalRequestProcessor* and propose in Leader for details. Since *prosalProcessor.processRequest* $\rightarrow$ *syncProcessor.processRequest* $\rightarrow$ *ackProcessor.processRequest* $\rightarrow$ *leader.processAck*, we initially set $txn.ackSid = \{i\}$, assuming we have done *leader.processAck*. Note: In production, any server in traffic can receive requests and
forward it to leader if necessary. We choose to let leader be the sole one who can receive write requests, to simplify spec and keep correctness at the same time.

```
$LeaderProcessRequest(i) \triangleq$
        $\land CheckTransactionNum$  test restrictions of transaction num
        $\land IsLeader(i)$
        $\land zabState[i] = BROADCAST$
        $\land$ LET $request\_value \triangleq GetRecorder($"nClientRequest"$)$  unique value
              $newTxn \triangleq [zxid \mapsto IncZxid(i, LastProposed(i).zxid),$
                                 $value \mapsto request\_value,$
                                 $ackSid \mapsto \{i\},$  assume we have done $leader.processAck$
                                 $epoch \mapsto acceptedEpoch[i]]$
              $m\_proposal \triangleq [mtype \mapsto PROPOSAL,$
                                    $mzxid \mapsto newTxn.zxid,$
                                    $mdata \mapsto request\_value]$
              $m\_proposal\_for\_checking \triangleq [source \mapsto i,$
                                                   $epoch \mapsto acceptedEpoch[i],$
                                                   $zxid \mapsto newTxn.zxid,$
                                                   $data \mapsto request\_value]$
           IN  $\land history' = [history$ EXCEPT $![i] = Append(@, newTxn)]$
               $\land Broadcast(i, m\_proposal)$
               $\land proposalMsgsLog' = proposalMsgsLog \cup \{m\_proposal\_for\_checking\}$
        $\land$ UNCHANGED $\langle state, currentEpoch, lastProcessed, zabState, acceptedEpoch,$
               $lastCommitted, electionVars, leaderVars, followerVars, initialHistory,$
               $epochLeader, violatedInvariants, electionMsgs\rangle$
        $\land UpdateRecorder(\langle$"LeaderProcessRequest"$, i\rangle)$

 Follower processes $PROPOSAL$ in $BROADCAST$. See $processPacket$ in Follower for details.
$FollowerProcessPROPOSAL(i, j) \triangleq$
        $\land IsFollower(i)$
        $\land PendingPROPOSAL(i, j)$
        $\land zabState[i] = BROADCAST$
        $\land$ LET $msg \triangleq msgs[j][i][1]$
              $infoOk \triangleq IsMyLeader(i, j)$
              $isNext \triangleq IsNextZxid(LastQueued(i).zxid, msg.mzxid)$
              $newTxn \triangleq [zxid \mapsto msg.mzxid,$
                                 $value \mapsto msg.mdata,$
                                 $ackSid \mapsto \{\},$
                                 $epoch \mapsto acceptedEpoch[i]]$
              $m\_ack \triangleq [mtype \mapsto ACK,$
                                 $mzxid \mapsto msg.mzxid]$
          IN  $\land infoOk$
              $\land \lor \land isNext$
                       $\land$ UNCHANGED $violatedInvariants$
                 $\lor \land \neg isNext$
                       $\land PrintT($"Exception: Follower receives PROPOSAL, while" $\circ$
                             " the transaction is not the next."$)$
                       $\land violatedInvariants' = [violatedInvariants$ EXCEPT
```

```
                                $!.proposalInconsistent = \text{TRUE}]$
                   $\land\ history' = [history\ \text{EXCEPT}\ ![i] = Append(@,\ newTxn)]$
                   $\land\ Reply(i,\ j,\ m\_ack)$
              $\land\ \text{UNCHANGED}\ \langle state,\ currentEpoch,\ lastProcessed,\ zabState,\ acceptedEpoch,$
                   $lastCommitted,\ electionVars,\ leaderVars,\ followerVars,\ initialHistory,$
                   $epochLeader,\ proposalMsgsLog,\ electionMsgs\rangle$
              $\land\ UpdateRecorder(\langle \text{"FollowerProcessPROPOSAL"},\ i,\ j\rangle)$

 See outstandingProposals in Leader
$OutstandingProposals(i) \triangleq \text{IF}\ zabState[i] \neq BROADCAST\ \text{THEN}\ \langle\rangle$
                                   $\text{ELSE}\ \ SubSeq(history[i],\ lastCommitted[i].index + 1,$
                                                 $Len(history[i]))$

$LastAckIndexFromFollower(i,\ j) \triangleq$
         $\text{LET}\ set\_index \triangleq \{idx \in 1\ ..\ Len(history[i]) : j \in history[i][idx].ackSid\}$
         $\text{IN}\ \ \ Maximum(set\_index)$

 See commit in Leader for details.
$LeaderCommit(s,\ follower,\ index,\ zxid) \triangleq$
         $\land\ lastCommitted' = [lastCommitted\ \text{EXCEPT}\ ![s] = [index \mapsto index,$
                                                                  $zxid\ \ \mapsto zxid]]$
         $\land\ \text{LET}\ m\_commit \triangleq [mtype \mapsto COMMIT,$
                                     $mzxid \mapsto zxid]$
           $\text{IN}\ \ \ DiscardAndBroadcast(s,\ follower,\ m\_commit)$

 Try to commit one operation, called by LeaderProcessAck.
 See tryToCommit in Leader for details.
 commitProcessor.commit → processWrite → toBeApplied.processRequest
  → finalProcessor.processRequest, finally processTxn will be implemented
 and lastProcessed will be updated. So we update it here.
$LeaderTryToCommit(s,\ index,\ zxid,\ newTxn,\ follower) \triangleq$
         $\text{LET}\ allTxnsBeforeCommitted \triangleq lastCommitted[s].index \geq index - 1$
                     Only when all proposals before zxid has been committed,
                     this proposal can be permitted to be committed.
             $hasAllQuorums \triangleq IsQuorum(newTxn.ackSid)$
                     In order to be committed, a proposal must be accepted
                     by a quorum.
             $ordered \triangleq lastCommitted[s].index + 1 = index$
                     Commit proposals in order.
         $\text{IN}\ \ \ \lor\ \land$  Current conditions do not satisfy committing the proposal.
                     $\lor\ \neg allTxnsBeforeCommitted$
                     $\lor\ \neg hasAllQuorums$
                   $\land\ Discard(follower,\ s)$
                   $\land\ \text{UNCHANGED}\ \langle violatedInvariants,\ lastCommitted,\ lastProcessed\rangle$
                 $\lor\ \land\ allTxnsBeforeCommitted$
                   $\land\ hasAllQuorums$
```

```
        ∧ ∨ ∧ $\neg ordered$
              ∧ $PrintT($“Warn: Committing zxid ” $\circ ToString(zxid) \circ$ “ not first.”$)$
              ∧ $violatedInvariants' = [violatedInvariants$ EXCEPT
                    $!.commitInconsistent =$ TRUE$]$
          ∨ ∧ $ordered$
              ∧ UNCHANGED $violatedInvariants$
        ∧ $LeaderCommit(s, follower, index, zxid)$
        ∧ $lastProcessed' = [lastProcessed$ EXCEPT $![s] = [index \mapsto index,$
                                                          $zxid \;\; \mapsto zxid]]$
```

Leader Keeps a count of acks for a particular proposal, and try to commit the proposal. See case *Leader.ACK* in *LearnerHandler*, *processRequest* in *AckRequestProcessor*, and *processAck* in Leader for details.

```
$LeaderProcessACK(i, j) \triangleq$
      ∧ $IsLeader(i)$
      ∧ $PendingACK(i, j)$
      ∧ LET $msg \triangleq msgs[j][i][1]$
            $infoOk \triangleq IsMyLearner(i, j)$
            $outstanding \triangleq LastCommitted(i).index < LastProposed(i).index$
                   outstandingProposals not null
            $hasCommitted \triangleq \neg ZxidCompare(msg.mzxid, LastCommitted(i).zxid)$
                   namely, $lastCommitted \geq zxid$
            $index \triangleq ZxidToIndex(history[i], msg.mzxid)$
            $exist \;\; \triangleq index \geq 1 \wedge index \leq LastProposed(i).index$
                   the proposal exists in history
            $ackIndex \triangleq LastAckIndexFromFollower(i, j)$
            $monotonicallyInc \triangleq$ ∨ $ackIndex = -1$
                                 ∨ $ackIndex + 1 = index$
                   TCP makes everytime ackIndex should just increase by 1
        IN  ∧ $infoOk$
            ∧ ∨ ∧ $exist$
                ∧ $monotonicallyInc$
                ∧ LET $txn \triangleq history[i][index]$
                      $txnAfterAddAck \triangleq [zxid \;\;\; \mapsto txn.zxid,$
                                             $value \;\; \mapsto txn.value,$
                                             $ackSid \mapsto txn.ackSid \cup \{j\},$
                                             $epoch \;\; \mapsto txn.epoch]$
                  IN    $p.addAck(sid)$
                  ∧   $history' = [history$ EXCEPT $![i][index] = txnAfterAddAck]$
                  ∧     ∨ ∧  Note: outstanding is 0.
                             / proposal has already been committed.
                           ∨ $\neg outstanding$
                           ∨ $hasCommitted$
                         ∧ $Discard(j, i)$
                         ∧ UNCHANGED $\langle violatedInvariants, lastCommitted, lastProcessed \rangle$
```

```
                    ∨ ∧ outstanding
                      ∧ ¬hasCommitted
                      ∧ LeaderTryToCommit(i, index, msg.mzxid, txnAfterAddAck, j)
                ∨ ∧ ∨ ¬exist
                    ∨ ¬monotonicallyInc
                  ∧ PrintT("Exception: No such zxid. " ∘
                        " / ackIndex doesn't inc monotonically.")
                  ∧ violatedInvariants' = [violatedInvariants
                        EXCEPT !.ackInconsistent = TRUE]
                  ∧ Discard(j, i)
                  ∧ UNCHANGED ⟨history, lastCommitted, lastProcessed⟩
    ∧ UNCHANGED ⟨state, currentEpoch, zabState, acceptedEpoch, electionVars,
            leaderVars, initialHistory, followerVars, proposalMsgsLog, epochLeader,
            electionMsgs⟩
    ∧ UpdateRecorder(⟨"LeaderProcessACK", i, j⟩)
```

Follower processes *COMMIT* in *BROADCAST*. See *processPacket* in Follower for details.

```
FollowerProcessCOMMIT(i, j) ≜
    ∧ IsFollower(i)
    ∧ PendingCOMMIT(i, j)
    ∧ zabState[i] = BROADCAST
    ∧ LET msg ≜ msgs[j][i][1]
          infoOk ≜ IsMyLeader(i, j)
          pendingTxns ≜ PendingTxns(i)
          noPending ≜ Len(pendingTxns) = 0
      IN
       ∧ infoOk
       ∧ ∨ ∧ noPending
             ∧ PrintT("Warn: Committing zxid without seeing txn.")
             ∧ UNCHANGED ⟨lastCommitted, lastProcessed, violatedInvariants⟩
         ∨ ∧ ¬noPending
             ∧ LET firstElementZxid ≜ pendingTxns[1].zxid
                   match ≜ ZxidEqual(firstElementZxid, msg.mzxid)
               IN
                ∨ ∧ ¬match
                  ∧ PrintT("Exception: Committing zxid not equals" ∘
                          " next pending txn zxid.")
                  ∧ violatedInvariants' = [violatedInvariants EXCEPT
                        !.commitInconsistent = TRUE]
                  ∧ UNCHANGED ⟨lastCommitted, lastProcessed⟩
                ∨ ∧ match
                  ∧ lastCommitted' = [lastCommitted EXCEPT
                          ![i] = [index ↦ lastCommitted[i].index + 1,
                                  zxid  ↦ firstElementZxid]]
                  ∧ lastProcessed' = [lastProcessed EXCEPT
```

```
                            $![i] = [index \mapsto lastCommitted[i].index + 1,$
                                     $zxid \quad \mapsto firstElementZxid]]$
                       $\land$ UNCHANGED $violatedInvariants$
            $\land Discard(j, i)$
            $\land$ UNCHANGED $\langle state, currentEpoch, zabState, acceptedEpoch, history,$
                        $electionVars, leaderVars, initialHistory, followerVars,$
                        $proposalMsgsLog, epochLeader, electionMsgs\rangle$
            $\land UpdateRecorder(\langle$"FollowerProcessCOMMIT"$, i, j\rangle)$
```

Used to discard some messages which should not exist in network channel. This action should not be triggered.

```
$FilterNonexistentMessage(i) \triangleq$
    $\land \exists j \in Server \setminus \{i\} : \land msgs[j][i] \neq \langle\rangle$
                                $\land$ LET $msg \triangleq msgs[j][i][1]$
                                  IN
                                    $\lor \land IsLeader(i)$
                                      $\land$ LET $infoOk \triangleq IsMyLearner(i, j)$
                                        IN
                                        $\lor msg.mtype = LEADERINFO$
                                        $\lor msg.mtype = NEWLEADER$
                                        $\lor msg.mtype = UPTODATE$
                                        $\lor msg.mtype = PROPOSAL$
                                        $\lor msg.mtype = COMMIT$
                                        $\lor \land \neg infoOk$
                                          $\land \lor msg.mtype = FOLLOWERINFO$
                                            $\lor msg.mtype = ACKEPOCH$
                                            $\lor msg.mtype = ACKLD$
                                            $\lor msg.mtype = ACK$
                                    $\lor \land IsFollower(i)$
                                      $\land$ LET $infoOk \triangleq IsMyLeader(i, j)$
                                        IN
                                        $\lor msg.mtype = FOLLOWERINFO$
                                        $\lor msg.mtype = ACKEPOCH$
                                        $\lor msg.mtype = ACKLD$
                                        $\lor msg.mtype = ACK$
                                        $\lor \land \neg infoOk$
                                          $\land \lor msg.mtype = LEADERINFO$
                                            $\lor msg.mtype = NEWLEADER$
                                            $\lor msg.mtype = UPTODATE$
                                            $\lor msg.mtype = PROPOSAL$
                                            $\lor msg.mtype = COMMIT$
                                    $\lor IsLooking(i)$
                                $\land Discard(j, i)$
    $\land violatedInvariants' = [violatedInvariants$ EXCEPT $!.messageIllegal =$ TRUE$]$
    $\land$ UNCHANGED $\langle serverVars, electionVars, leaderVars,$
```

$followerVars, proposalMsgsLog, epochLeader, electionMsgs\rangle$

$\land UnchangeRecorder$

Defines how the variables may transition.

$Next \triangleq$

FLE modlue

$\lor \exists i, j \in Server : FLEReceiveNotmsg(i, j)$
$\lor \exists i \in Server : FLENotmsgTimeout(i)$
$\lor \exists i \in Server : FLEHandleNotmsg(i)$
$\lor \exists i \in Server : FLEWaitNewNotmsg(i)$
$\lor \exists i \in Server : FLEWaitNewNotmsgEnd(i)$

Some conditions like failure, network delay

$\lor \exists i \in Server : FollowerTimeout(i)$
$\lor \exists i \in Server : LeaderTimeout(i)$
$\lor \exists i, j \in Server : Timeout(i, j)$

Zab module - Discovery and Synchronization part

$\lor \exists i, j \in Server : ConnectAndFollowerSendFOLLOWERINFO(i, j)$
$\lor \exists i, j \in Server : LeaderProcessFOLLOWERINFO(i, j)$
$\lor \exists i, j \in Server : FollowerProcessLEADERINFO(i, j)$
$\lor \exists i, j \in Server : LeaderProcessACKEPOCH(i, j)$
$\lor \exists i \in Server : LeaderSyncFollower(i)$
$\lor \exists i, j \in Server : FollowerProcessSyncMessage(i, j)$
$\lor \exists i, j \in Server : FollowerProcessPROPOSALInSync(i, j)$
$\lor \exists i, j \in Server : FollowerProcessCOMMITInSync(i, j)$
$\lor \exists i, j \in Server : FollowerProcessNEWLEADER(i, j)$
$\lor \exists i, j \in Server : LeaderProcessACKLD(i, j)$
$\lor \exists i, j \in Server : FollowerProcessUPTODATE(i, j)$

Zab module $-$ *Broadcast* part

$\lor \exists i \in Server : LeaderProcessRequest(i)$
$\lor \exists i, j \in Server : FollowerProcessPROPOSAL(i, j)$
$\lor \exists i, j \in Server : LeaderProcessACK(i, j)$ *Sync* + *Broadcast*
$\lor \exists i, j \in Server : FollowerProcessCOMMIT(i, j)$

An action used to judge whether there are redundant messages in network

$\lor \exists i \in Server : FilterNonexistentMessage(i)$

$Spec \triangleq Init \land \Box[Next]_{vars}$

Define safety properties of *Zab* 1.0 protocol.

$ShouldNotBeTriggered \triangleq \forall p \in \text{DOMAIN } violatedInvariants : violatedInvariants[p] = \text{FALSE}$

There is most one established leader for a certain epoch.

$Leadership1 \triangleq \forall i, j \in Server :$
$\land IsLeader(i) \land zabState[i] \in \{SYNCHRONIZATION, BROADCAST\}$
$\land IsLeader(j) \land zabState[j] \in \{SYNCHRONIZATION, BROADCAST\}$
$\land acceptedEpoch[i] = acceptedEpoch[j]$

```
                  $\Rightarrow i = j$

$Leadership2 \triangleq \forall epoch \in 1 \,..\, MAXEPOCH : Cardinality(epochLeader[epoch]) \leq 1$
```

PrefixConsistency: The prefix that have been committed
in history in any process is the same.

```
$PrefixConsistency \triangleq \forall i, j \in Server :$
                  LET $smaller \triangleq Minimum(\{lastCommitted[i].index, lastCommitted[j].index\})$
                  IN  $\lor smaller = 0$
                      $\lor \land smaller > 0$
                        $\land \forall index \in 1 \,..\, smaller :$
                             $TxnEqual(history[i][index], history[j][index])$
```

Integrity: If some follower delivers one transaction, then some primary has broadcast it.

```
$Integrity \triangleq \forall i \in Server :$
             $\land IsFollower(i)$
             $\land lastCommitted[i].index > 0$
             $\Rightarrow \forall idx \in 1 \,..\, lastCommitted[i].index : \exists proposal \in proposalMsgsLog :$
                LET $txn\_proposal \triangleq [zxid \mapsto proposal.zxid,$
                                             $value \mapsto proposal.data]$
                IN  $TxnEqual(history[i][idx], txn\_proposal)$
```

Agreement: If some follower f delivers transaction a and some follower f' delivers transaction b,
then f' delivers a or f delivers b.

```
$Agreement \triangleq \forall i, j \in Server :$
              $\land IsFollower(i) \land lastCommitted[i].index > 0$
              $\land IsFollower(j) \land lastCommitted[j].index > 0$
              $\Rightarrow$
              $\forall idx1 \in 1 \,..\, lastCommitted[i].index, idx2 \in 1 \,..\, lastCommitted[j].index :$
                 $\lor \exists idx\_j \in 1 \,..\, lastCommitted[j].index :$
                     $TxnEqual(history[j][idx\_j], history[i][idx1])$
                 $\lor \exists idx\_i \in 1 \,..\, lastCommitted[i].index :$
                     $TxnEqual(history[i][idx\_i], history[j][idx2])$
```

Total order: If some follower delivers a before b, then any process that delivers b
must also deliver a and deliver a before b.

```
$TotalOrder \triangleq \forall i, j \in Server :$
              LET $committed1 \triangleq lastCommitted[i].index$
                  $committed2 \triangleq lastCommitted[j].index$
              IN  $committed1 \geq 2 \land committed2 \geq 2$
                    $\Rightarrow \forall idx\_i1 \in 1 \,..\, (committed1 - 1) : \forall idx\_i2 \in (idx\_i1 + 1) \,..\, committed1 :$
                  LET $logOk \triangleq \exists idx \in 1 \,..\, committed2 :$
                                  $TxnEqual(history[i][idx\_i2], history[j][idx])$
                  IN  $\lor \neg logOk$
                      $\lor \land logOk$
                        $\land \exists idx\_j2 \in 1 \,..\, committed2 :$
```

```
                         $\land\ TxnEqual(history[i][idx\_i2],\ history[j][idx\_j2])$
                         $\land\ \exists\ idx\_j1 \in 1\ ..\ (idx\_j2 - 1):$
                               $TxnEqual(history[i][idx\_i1],\ history[j][idx\_j1])$
```

Local primary order: If a primary broadcasts a before it broadcasts b, then a follower that delivers b must also deliver a before b.

```
$LocalPrimaryOrder \triangleq$ LET $p\_set(i, e) \triangleq \{p \in proposalMsgsLog : \land\ p.source = i$
                                                                  $\land\ p.epoch\ = e\}$
                          $txn\_set(i, e) \triangleq \{[zxid \mapsto p.zxid,$
                                                   $value \mapsto p.data] : p \in p\_set(i, e)\}$
                      IN  $\forall\ i \in Server : \forall\ e \in 1\ ..\ currentEpoch[i]:$
                            $\lor\ Cardinality(txn\_set(i, e)) < 2$
                            $\lor\ \land\ Cardinality(txn\_set(i, e)) \geq 2$
                              $\land\ \exists\ txn1, txn2 \in txn\_set(i, e):$
                                $\lor\ TxnEqual(txn1, txn2)$
                                $\lor\ \land\ \neg\ TxnEqual(txn1, txn2)$
                                  $\land$ LET $TxnPre\ \ \triangleq$ IF $ZxidCompare(txn1.zxid, txn2.zxid)$ THEN $txn2$ ELS
                                            $TxnNext \triangleq$ IF $ZxidCompare(txn1.zxid, txn2.zxid)$ THEN $txn1$ ELS
                                     IN  $\forall\ j \in Server: \land\ lastCommitted[j].index \geq 2$
                                                            $\land\ \exists\ idx \in 1\ ..\ lastCommitted[j].index:$
                                                                  $TxnEqual(history[j][idx],\ TxnNext)$
                                           $\Rightarrow \exists\ idx2 \in 1\ ..\ lastCommitted[j].index:$
                                                 $\land\ TxnEqual(history[j][idx2],\ TxnNext)$
                                                 $\land\ idx2 > 1$
                                                 $\land\ \exists\ idx1 \in 1\ ..\ (idx2 - 1):$
                                                       $TxnEqual(history[j][idx1],\ TxnPre)$
```

Global primary order: A follower f delivers both a with epoch e and b with epoch e', and $e < e'$, then f must deliver a before b.

```
$GlobalPrimaryOrder \triangleq \forall\ i \in Server : lastCommitted[i].index \geq 2$
                       $\Rightarrow \forall\ idx1, idx2 \in 1\ ..\ lastCommitted[i].index:$
                             $\lor\ \neg EpochPrecedeInTxn(history[i][idx1],\ history[i][idx2])$
                             $\lor\ \land\ EpochPrecedeInTxn(history[i][idx1],\ history[i][idx2])$
                                 $\land\ idx1 < idx2$
```

Primary integrity: If primary p broadcasts a and some follower f delivers b such that b has epoch smaller than epoch of p, then p must deliver b before it broadcasts a.

```
$PrimaryIntegrity \triangleq \forall\ i, j \in Server: \land\ IsLeader(i)\quad \land\ IsMyLearner(i, j)$
                                         $\land\ IsFollower(j) \land\ IsMyLeader(j, i)$
                                         $\land\ zabState[i] = BROADCAST$
                                         $\land\ zabState[j] = BROADCAST$
                                         $\land\ lastCommitted[j].index \geq 1$
                  $\Rightarrow \forall\ idx\_j \in 1\ ..\ lastCommitted[j].index:$
                        $\lor\ history[j][idx\_j].zxid[1] \geq currentEpoch[i]$
                        $\lor\ \land\ history[j][idx\_j].zxid[1] < currentEpoch[i]$
                            $\land\ \exists\ idx\_i \in 1\ ..\ lastCommitted[i].index:$
```

$$TxnEqual(history[i][idx\_i],\ history[j][idx\_j])$$

* Modification History

* Last modified *Mon Nov* 22 21:49:29 *CST* 2021 by Dell

* Created Sat *Oct* 23 16:05:04 *CST* 2021 by Dell