MODULE *P4RuntimeElection*

EXTENDS *Naturals*, *FiniteSets*, *Sequences*, *TLC*

The set of all *ONOS* nodes

CONSTANTS *Nodes*

Stream states

CONSTANTS *Open*, *Closed*

Master arbitration message types

CONSTANTS *MasterArbitrationUpdate*

Write message types

CONSTANTS *WriteRequest*, *WriteResponse*

Response status constants

CONSTANTS *Ok*, *AlreadyExists*, *PermissionDenied*

Empty value

CONSTANT *Nil*

The current state of mastership elections

VARIABLES *term*, *master*, *backups*

The current mastership event queue for each node

VARIABLE *events*

The current mastership state for each node

VARIABLE *masterships*

The state of all streams and their requests and responses

VARIABLE *streams*, *requests*, *responses*

The current set of elections for the switch, the greatest of which is the current master

VARIABLE *elections*

Counting variables used to enforce state constraints

VARIABLES *mastershipChanges*, *streamChanges*, *messageCount*

A sequence of successful writes to the switch used for model checking

VARIABLE *writes*

Mastership/consensus related variables

$mastershipVars \triangleq \langle term, master, backups, mastershipChanges \rangle$

Node related variables

$nodeVars \triangleq \langle events, masterships \rangle$

Stream related variables

$$\mathit{streamVars} \triangleq \langle \mathit{streams},\ \mathit{streamChanges} \rangle$$

Message related variables

$$\mathit{messageVars} \triangleq \langle \mathit{requests},\ \mathit{responses},\ \mathit{messageCount} \rangle$$

Device related variables

$$\mathit{deviceVars} \triangleq \langle \mathit{elections},\ \mathit{writes} \rangle$$

A sequence of all variables

$$\mathit{vars} \triangleq \langle \mathit{mastershipVars},\ \mathit{nodeVars},\ \mathit{streamVars},\ \mathit{messageVars},\ \mathit{deviceVars} \rangle$$

Helpers

Returns a sequence with the head removed

$$\mathit{Pop}(q) \triangleq \mathit{SubSeq}(q,\ 2,\ \mathit{Len}(q))$$

Returns a sequences with the element at the given index removed

$$\mathit{Drop}(q,\ i) \triangleq \mathit{SubSeq}(q,\ 1,\ i-1) \circ \mathit{SubSeq}(q,\ i+1,\ \mathit{Len}(q))$$

Returns the set of values in f

$$\mathit{Range}(f) \triangleq \{f[x] : x \in \textsc{domain}\ f\}$$

Returns the maximum value from a set or undefined if the set is empty

$$\mathit{Max}(s) \triangleq \textsc{choose}\ x \in s : \forall\, y \in s : x \geq y$$

Messaging between the *Nodes* and the device are modelled on *TCP*. For each node, a request and response sequence provides ordered messaging between the two points. Requests and responses are always received from the head of the queue and are never duplicated or reordered, and request and response queues only last the lifetime of the stream. When a stream is closed, all that stream's requests and responses are lost.

Sends request 'm' on the stream for node 'n'

$$\begin{aligned}
&\mathit{SendRequest}(n,\ m) \triangleq \\
&\quad \land\ \mathit{requests}' = [\mathit{requests}\ \textsc{except}\ ![n] = \mathit{Append}(\mathit{requests}[n],\ m)] \\
&\quad \land\ \mathit{messageCount}' = \mathit{messageCount} + 1
\end{aligned}$$

Indicates whether any requests are in the queue for node 'n'

$$\mathit{HasRequest}(n,\ t) \triangleq \mathit{Len}(\mathit{requests}[n]) > 0 \land \mathit{requests}[n][1].\mathit{type} = t$$

Returns the next request in the queue for node 'n'

$$\mathit{NextRequest}(n) \triangleq \mathit{requests}[n][1]$$

Discards the request at the head of the queue for node 'n'

$$\mathit{DiscardRequest}(n) \triangleq \mathit{requests}' = [\mathit{requests}\ \textsc{except}\ ![n] = \mathit{Pop}(\mathit{requests}[n])]$$

Sends response 'm' on the stream for node 'n'

$$\mathit{SendResponse}(n,\ m) \triangleq$$

```
    $\land responses' = [responses \text{ EXCEPT } ![n] = Append(responses[n], m)]$
    $\land messageCount' = messageCount + 1$
```

Indicates whether any responses are in the queue for node 'n'

```
$HasResponse(n, t) \triangleq Len(responses[n]) > 0 \land responses[n][1].type = t$
```

Returns the next response in the queue for node 'n'

```
$NextResponse(n) \triangleq responses[n][1]$
```

Discards the response at the head of the queue for node 'n'

```
$DiscardResponse(n) \triangleq responses' = [responses \text{ EXCEPT } ![n] = Pop(responses[n])]$
```

This section models mastership arbitration on the controller side. Mastership election occurs in two disctinct types of state changes. One state change occurs to change the mastership in the consensus layer, and the other occurs when a node actually learns of the mastership change. Nodes will always learn of mastership changes in the order in which they occur, and nodes will always learn of a mastership change. This, of course, is not representative of practice but is sufficient for modelling the mastership election algorithm.

Adds a node to the mastership election

```
$JoinMastershipElection(n) \triangleq$
    $\land \lor \land master = Nil$
            $\land term' = term + 1$
            $\land master' = n$
            $\land backups' = \langle\rangle$
            $\land events' = [i \in Nodes \mapsto Append(events[i], [$
                                                    $term \mapsto term',$
                                                    $master \mapsto master',$
                                                    $backups \mapsto backups'])]$
       $\lor \land master \neq Nil$
            $\land master \neq n$
            $\land n \notin Range(backups)$
            $\land backups' = Append(backups, n)$
            $\land events' = [i \in Nodes \mapsto Append(events[i], [$
                                                    $term \mapsto term,$
                                                    $master \mapsto master,$
                                                    $backups \mapsto backups'])]$
            $\land \text{UNCHANGED } \langle term, master\rangle$
    $\land mastershipChanges' = mastershipChanges + 1$
    $\land \text{UNCHANGED } \langle masterships, streamVars, messageVars, deviceVars\rangle$
```

Removes a node from the mastership election

```
$LeaveMastershipElection(n) \triangleq$
    $\land \lor \land master = n$
            $\land \lor \land Len(backups) > 0$
                    $\land term' = term + 1$
                    $\land master' = backups[1]$
            $\land backups' = Pop(backups)$
            $\land events' = [i \in Nodes \mapsto Append(events[i], [$
                                        $term \mapsto term',$
                                        $master \mapsto master',$
                                        $backups \mapsto backups'])]$
          $\lor \land Len(backups) = 0$
            $\land master' = Nil$
            $\land \text{UNCHANGED } \langle term, backups, events \rangle$
      $\lor \land n \in Range(backups)$
        $\land backups' = Drop(backups, \text{CHOOSE } j \in \text{DOMAIN } backups : backups[j] = n)$
        $\land \text{UNCHANGED } \langle term, master, events \rangle$
    $\land mastershipChanges' = mastershipChanges + 1$
    $\land \text{UNCHANGED } \langle masterships, streamVars, messageVars, deviceVars \rangle$
```

Sets the current master to node 'n' if it's not already set

```
$SetMastership(n) \triangleq$
    $\lor \land master = n$
      $\land \text{UNCHANGED } \langle mastershipVars \rangle$
    $\lor \land master \neq n$
      $\land term' = term + 1$
      $\land master' = n$
      $\land \lor \land n \in Range(backups)$
            $\land backups' = Drop(backups, \text{CHOOSE } j \in \text{DOMAIN } backups : backups[j] = n)$
         $\lor \land n \notin Range(backups)$
            $\land \text{UNCHANGED } \langle backups \rangle$
      $\land mastershipChanges' = mastershipChanges + 1$
```

Receives a mastership change event from the consensus layer on node 'n'

```
$LearnMastership(n) \triangleq$
    $\land Len(events[n]) > 0$
    $\land \text{LET } e \triangleq events[n][1]$
           $m \triangleq masterships[n]$
      $\text{IN}$
          $\lor \land e.term > m.term$
            $\land masterships' = [masterships \text{ EXCEPT } ![n] = [$
                                  $term \mapsto e.term,$
                                  $master \mapsto e.master,$
                                  $backups \mapsto e.backups,$
                                  $sent \mapsto \text{FALSE}]]$
          $\lor \land e.term = m.term$
            $\land masterships' = [masterships \text{ EXCEPT } ![n] = [$
                                  $term \mapsto e.term,$
                                  $master \mapsto e.master,$
                                  $backups \mapsto e.backups,$
                                  $sent \mapsto m.sent]]$
    ∧ events′ = [events EXCEPT ![n] = Pop(events[n])]
    ∧ UNCHANGED ⟨mastershipVars, streamVars, messageVars, deviceVars⟩

 Notifies the device of node 'n' mastership info if it hasn't already been sent
SendMasterArbitrationUpdateRequest(n) ≜
    ∧ streams[n] = Open
    ∧ LET m ≜ masterships[n]
      IN
          ∧ m.term > 0
          ∧ ¬m.sent
          ∧ ∨ ∧ m.master = n
                ∧ SendRequest(n, [
                       type        ↦ MasterArbitrationUpdate,
                       election_id ↦ m.term + Cardinality(Nodes),
                       term        ↦ m.term])
            ∨ ∧ m.master ≠ n
              ∧ n ∈ Range(m.backups)
              ∧ SendRequest(n, [
                       type        ↦ MasterArbitrationUpdate,
                       election_id ↦ m.term + Cardinality(Nodes) − CHOOSE i ∈ DOMAIN m.backups : m.backu
                       term        ↦ m.term])
    ∧ masterships′ = [masterships EXCEPT ![n].sent = TRUE]
    ∧ UNCHANGED ⟨mastershipVars, events, deviceVars, streamVars, responses⟩

 Receives a master arbitration update response on node 'n'
ReceiveMasterArbitrationUpdateResponse(n) ≜
    ∧ streams[n] = Open
    ∧ HasResponse(n, MasterArbitrationUpdate)
    ∧ LET m ≜ NextResponse(n)
      IN
          ∨ ∧ m.status = Ok
            ∧ SetMastership(n)
          ∨ ∧ m.status = AlreadyExists
            ∧ UNCHANGED ⟨mastershipVars⟩
    ∧ DiscardResponse(n)
    ∧ UNCHANGED ⟨nodeVars, deviceVars, streamVars, requests, messageCount⟩

 Sends a write request to the device from node 'n'
SendWriteRequest(n) ≜
    ∧ streams[n] = Open
    ∧ LET m ≜ masterships[n]
      IN
          ∧ m.term > 0
          ∧ m.master = n
          ∧ SendRequest(n, [
                 type        ↦ WriteRequest,
                  $election\_id \mapsto m.term + Cardinality(Nodes),$
                  $term \qquad\quad \mapsto m.term])$
    $\land$ UNCHANGED $\langle masteshipVars, nodeVars, deviceVars, streamVars, responses\rangle$
```

Receives a write response on node 'n'

```
$ReceiveWriteResponse(n) \triangleq$
    $\land streams[n] = Open$
    $\land HasResponse(n, WriteResponse)$
    $\land$ LET $m \triangleq NextResponse(n)$
      IN
         TODO: This should be used to determine whether writes from old masters are allowed
         $\lor m.status = Ok$
         $\lor m.status = PermissionDenied$
    $\land DiscardResponse(n)$
    $\land$ UNCHANGED $\langle mastershipVars, nodeVars, deviceVars, streamVars, requests, messageCount\rangle$
```

This section models the $P4$ switch. The switch side manages stream states between the device and the controller. Streams are opened and closed in a single state transition for the purposes of this model. Switches can handle two types of messages from the controller nodes: *MasterArbitrationUpdate* and Write.

Returns the highest election ID for the given elections

```
$ElectionId(e) \triangleq Max(Range(e))$
```

Returns the master for the given elections

```
$Master(e) \triangleq$
    IF $Cardinality(\{i \in Range(e) : i > 0\}) > 0$ THEN
        CHOOSE $n \in$ DOMAIN $e : e[n] = ElectionId(e)$
    ELSE
        $Nil$
```

Opens a new stream between node 'n' and the device
When a new stream is opened, the 'requests' and 'responses' queues for the node are cleared and the 'streams' state is set to 'Open'.

```
$ConnectStream(n) \triangleq$
    $\land streams[n] = Closed$
    $\land streams' = [streams$ EXCEPT $![n] = Open]$
    $\land requests' = [requests$ EXCEPT $![n] = \langle\rangle]$
    $\land responses' = [responses$ EXCEPT $![n] = \langle\rangle]$
    $\land streamChanges' = streamChanges + 1$
    $\land$ UNCHANGED $\langle mastershipVars, nodeVars, deviceVars, messageCount\rangle$
```

Closes the open stream between node 'n' and the device
When the stream is closed, the 'requests' and 'responses' queues for the node are cleared and a 'MasterArbitrationUpdate' is sent to all remaining connected nodes to notify them of a mastership change if necessary.

```
$CloseStream(n) \triangleq$
    $\land\ streams[n] = Open$
    $\land\ elections' = [elections \ \text{EXCEPT}\ ![n] = 0]$
    $\land\ streams' = [streams \ \text{EXCEPT}\ ![n] = Closed]$
    $\land\ requests' = [requests \ \text{EXCEPT}\ ![n] = \langle\rangle]$
    $\land\ \text{LET}\ oldMaster \triangleq Master(elections)$
              $newMaster \triangleq Master(elections')$
       $\text{IN}$
           $\lor\ \land\ oldMaster \neq newMaster$
              $\land\ responses' = [i \in \text{DOMAIN}\ streams' \mapsto$
                                      $\text{IF}\ i = newMaster\ \text{THEN}$
                                          $Append(responses[i], [$
                                              $type \mapsto MasterArbitrationUpdate,$
                                              $status \mapsto Ok,$
                                              $election\_id \mapsto ElectionId(elections')])$
                                       $\text{ELSE}$
                                          $Append(responses[i], [$
                                              $type \mapsto MasterArbitrationUpdate,$
                                              $status \mapsto AlreadyExists,$
                                              $election\_id \mapsto ElectionId(elections')])]$
              $\land\ messageCount' = messageCount + 1$
           $\lor\ \land\ oldMaster = newMaster$
              $\land\ responses' = [responses \ \text{EXCEPT}\ ![n] = \langle\rangle]$
              $\land\ \text{UNCHANGED}\ \langle messageCount \rangle$
    $\land\ streamChanges' = streamChanges + 1$
    $\land\ \text{UNCHANGED}\ \langle mastershipVars, nodeVars, writes \rangle$
```

Handles a master arbitration update on the device
If the *election_id* is already present in the 'elections', send an 'AlreadyExists'
response to the node. Otherwise,

```
$HandleMasterArbitrationUpdate(n) \triangleq$
    $\land\ streams[n] = Open$
    $\land\ HasRequest(n, MasterArbitrationUpdate)$
    $\land\ \text{LET}\ m \triangleq NextRequest(n)$
       $\text{IN}$
           $\lor\ \land\ m.election\_id \in Range(elections)$
              $\land\ SendResponse(n, [$
                        $type \mapsto MasterArbitrationUpdate,$
                        $election\_id \mapsto m.election\_id,$
                        $status \mapsto AlreadyExists])$
              $\land\ \text{UNCHANGED}\ \langle deviceVars \rangle$
           $\lor\ \land\ m.election\_id \notin Range(elections)$
              $\land\ elections' = [elections \ \text{EXCEPT}\ ![n] = m.election\_id]$
              $\land\ \text{LET}\ oldMaster \triangleq Master(elections)$
                        $newMaster \triangleq Master(elections')$
```

```
                IN
                    $\lor \land oldMaster \neq newMaster$
                      $\land responses' = [i \in$ DOMAIN $streams \mapsto$
                                              IF $i = newMaster$ THEN
                                                  $Append(responses[i], [$
                                                      $type \mapsto MasterArbitrationUpdate,$
                                                      $status \mapsto Ok,$
                                                      $election\_id \mapsto ElectionId(elections')])$
                                              ELSE
                                                  $Append(responses[i], [$
                                                      $type \mapsto MasterArbitrationUpdate,$
                                                      $status \mapsto AlreadyExists,$
                                                      $election\_id \mapsto ElectionId(elections')])]$
                      $\land messageCount' = messageCount + 1$
                    $\lor \land oldMaster = newMaster$
                      $\land SendResponse(n, [$
                              $type \mapsto MasterArbitrationUpdate,$
                              $status \mapsto Ok,$
                              $election\_id \mapsto ElectionId(elections')])$
    $\land DiscardRequest(n)$
    $\land$ UNCHANGED $\langle masterhipVars, nodeVars, streamVars, writes \rangle$

 Handles a write request on the device
$HandleWrite(n) \triangleq$
    $\land streams[n] = Open$
    $\land HasRequest(n, WriteRequest)$
    $\land$ LET $m \triangleq NextRequest(n)$
      IN
        $\lor \land elections[n] = m.election\_id$
          $\land Master(elections) = n$
          $\land writes' = Append(writes, [node \mapsto n, term \mapsto m.term])$
          $\land SendResponse(n, [$
                  $type \mapsto WriteResponse,$
                  $status \mapsto Ok])$
        $\lor \land \lor elections[n] \neq m.election\_id$
             $\lor Master(elections) \neq n$
          $\land SendResponse(n, [$
                  $type \mapsto WriteResponse,$
                  $status \mapsto PermissionDenied])$
          $\land$ UNCHANGED $\langle writes \rangle$
    $\land DiscardRequest(n)$
    $\land$ UNCHANGED $\langle mastershipVars, nodeVars, elections, streamVars \rangle$
```

The invariant asserts that no master can write to the switch after the switch has been notified of a newer master

$TypeInvariant \triangleq \forall i \in \text{DOMAIN } writes : i = 1 \lor writes[i-1].term \leq writes[i].term$

$$
\begin{array}{l}
Init \triangleq \\
\quad \land\ term = 0 \\
\quad \land\ master = Nil \\
\quad \land\ backups = \langle\rangle \\
\quad \land\ events = [n \in Nodes \mapsto \langle\rangle] \\
\quad \land\ masterships = [n \in Nodes \mapsto [term \mapsto 0,\ master \mapsto 0,\ backups \mapsto \langle\rangle,\ sent \mapsto \text{FALSE}]] \\
\quad \land\ streams = [n \in Nodes \mapsto Closed] \\
\quad \land\ requests = [n \in Nodes \mapsto \langle\rangle] \\
\quad \land\ responses = [n \in Nodes \mapsto \langle\rangle] \\
\quad \land\ elections = [n \in Nodes \mapsto 0] \\
\quad \land\ mastershipChanges = 0 \\
\quad \land\ streamChanges = 0 \\
\quad \land\ messageCount = 0 \\
\quad \land\ writes = \langle\rangle
\end{array}
$$

$$
\begin{array}{l}
Next \triangleq \\
\quad \lor\ \exists\, n \in Nodes : ConnectStream(n) \\
\quad \lor\ \exists\, n \in Nodes : CloseStream(n) \\
\quad \lor\ \exists\, n \in Nodes : JoinMastershipElection(n) \\
\quad \lor\ \exists\, n \in Nodes : LeaveMastershipElection(n) \\
\quad \lor\ \exists\, n \in Nodes : LearnMastership(n) \\
\quad \lor\ \exists\, n \in Nodes : SendMasterArbitrationUpdateRequest(n) \\
\quad \lor\ \exists\, n \in Nodes : HandleMasterArbitrationUpdate(n) \\
\quad \lor\ \exists\, n \in Nodes : ReceiveMasterArbitrationUpdateResponse(n) \\
\quad \lor\ \exists\, n \in Nodes : SendWriteRequest(n) \\
\quad \lor\ \exists\, n \in Nodes : HandleWrite(n) \\
\quad \lor\ \exists\, n \in Nodes : ReceiveWriteResponse(n)
\end{array}
$$

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

* Modification History

* Last modified Sun *Feb* 17 03:10:44 *PST* 2019 by *jordanhalterman*

* Created *Thu Feb* 14 11:33:03 *PST* 2019 by *jordanhalterman*