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*

Device/role constants used in *P*4 master arbitration requests
CONSTANTS *DeviceId*, *RoleId*

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 *mastership*

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*

Mastership/consensus related variables
$mastershipVars \triangleq \langle term, master, backups \rangle$

Node related variables
$nodeVars \triangleq \langle events, mastership \rangle$

Stream/communication related variables
$streamVars \triangleq \langle streams, requests, responses \rangle$

Device related variables

$deviceVars \triangleq \langle elections \rangle$

A sequence of all variables

$vars \triangleq \langle mastershipVars,\ nodeVars,\ streamVars,\ deviceVars \rangle$

Helpers

Returns a sequence with the head removed

$Pop(q) \triangleq SubSeq(q,\ 2,\ Len(q))$

Returns the set of values in f

$Range(f) \triangleq \{f[x] : x \in \text{DOMAIN } f\}$

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

$Max(s) \triangleq \text{CHOOSE } x \in s : \forall\, y \in s : x \geq y$

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'

$SendRequest(n,\ m) \triangleq requests' = [requests \text{ EXCEPT } ![n] = Append(requests[n],\ m)]$

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

$HasRequest(n,\ t) \triangleq Len(requests) > 0 \wedge requests[0].type = t$

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

$NextRequest(n) \triangleq requests[0]$

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

$DiscardRequest(n) \triangleq requests' = [requests \text{ EXCEPT } ![n] = Pop(requests[n])]$

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

$SendResponse(n,\ m) \triangleq responses' = [responses \text{ EXCEPT } ![n] = Append(responses[n],\ m)]$

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

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

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

$NextResponse(n) \triangleq responses[0]$

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

$$
\begin{array}{l}
JoinMastershipElection(n) \triangleq \\
\quad \land \lor \land master = Nil \\
\qquad \land term' = term + 1 \\
\qquad \land master' = n \\
\qquad \land backups' = \langle\rangle \\
\qquad \land events' = [i \in Nodes \mapsto Append(events[i], [\\
\qquad\qquad\qquad\qquad term \mapsto term', \\
\qquad\qquad\qquad\qquad master \mapsto master', \\
\qquad\qquad\qquad\qquad backups \mapsto backups'])] \\
\quad \land \lor \land master \neq Nil \\
\qquad \land n \notin Range(backups) \\
\qquad \land backups' = Append(backups, n) \\
\qquad \land \text{UNCHANGED } \langle events \rangle \\
\quad \land \text{UNCHANGED } \langle term, master, streamVars, deviceVars \rangle
\end{array}
$$

Removes a node from the mastership election

$$
\begin{array}{l}
LeaveMastershipElection(n) \triangleq \\
\quad \land \lor \land master = n \\
\qquad \land \lor \land Len(backups) > 0 \\
\qquad\quad \land term' = term + 1 \\
\qquad\quad \land master' = backups[0] \\
\qquad\quad \land backups' = Pop(backups) \\
\qquad\quad \land events' = [i \quad \in Nodes \mapsto Append(events[i], [\\
\qquad\qquad\qquad\qquad\qquad\qquad term \mapsto term', \\
\qquad\qquad\qquad\qquad\qquad\qquad master \mapsto master', \\
\qquad\qquad\qquad\qquad\qquad\qquad backups \mapsto backups'])] \\
\qquad\quad \lor \land Len(backups) = 0 \\
\qquad\quad\ \land master' = Nil \\
\qquad\quad\ \land \text{UNCHANGED } \langle term, backups \rangle \\
\qquad \lor \land n \in Range(backups) \\
\qquad\ \land backups' = [j \in \text{DOMAIN } backups \setminus \{\text{CHOOSE } j \in \text{DOMAIN } backups : backups[j] = n\} \mapsto backups[j]] \\
\qquad\ \land \text{UNCHANGED } \langle term, master \rangle \\
\quad \land \text{UNCHANGED } \langle streamVars, deviceVars \rangle
\end{array}
$$

Changes mastership for the device in the consensus layer, adding a mastership change event to each node's event queue

$$
\begin{array}{l}
ChangeMastership \triangleq \\
\quad \land term' = term + 1 \\
\quad \land master' = backups[0]
\end{array}
$$

```
    ∧ backups′ = Append(Pop(backups), master)
    ∧ events′ = [n ∈ Nodes ↦ Append(events[n], [
                                  term ↦ term′,
                                  master ↦ master′,
                                  backups ↦ backups′])]
    ∧ UNCHANGED ⟨streamVars, deviceVars⟩

 Receives a mastership change event from the consensus layer on node 'n'
LearnMastership(n) ≜
    ∧ Len(events[n]) > 0
    ∧ LET m ≜ events[n][0]
      IN
          ∧ mastership′ = [mastership EXCEPT ![n] = [
                              term      ↦ m.term,
                              master    ↦ m.master,
                              backups   ↦ m.backups,
                              sent      ↦ FALSE]]
    ∧ UNCHANGED ⟨mastershipVars, streamVars, deviceVars, streams⟩

 Notifies the device of node 'n' mastership info if it hasn't already been sent
SendMasterArbitrationUpdateRequest(n) ≜
    ∧ masterships[n].term > 0
    ∧ ¬masterships[n].sent
    ∧ streams[n] = Open
    ∧ LET m ≜ masterships[n]
      IN
          ∧ m.term > 0
          ∧ ¬m.sent
          ∧ ∨ ∧ m.master = n
              ∧ SendRequest(n, [
                    type        ↦ MasterArbitrationUpdate,
                    election_id ↦ m.term])
            ∨ ∧ m.master ≠ n
              ∧ SendRequest(n, [
                    type        ↦ MasterArbitrationUpdate,
                    election_id ↦ 0])
    ∧ UNCHANGED ⟨mastershipVars, nodeVars, deviceVars, streams⟩

 Receives a master arbitration update response on node 'n'
ReceiveMasterArbitrationUpdateResponse(n) ≜
    ∧ streams[n] = Open
    ∧ HasResponse(n, MasterArbitrationUpdate)
    ∧ LET m ≜ NextResponse(n)
      IN
          ∧ m  TODO
    ∧ DiscardResponse(n)
```

$\land$ UNCHANGED $\langle deviceVars,\ streams\rangle$

Sends a write request to the device from node 'n'

$SendWriteRequest(n) \triangleq$
$\land\ streams[n] = Open$
$\land$ LET $m \triangleq mastership s[n]$
IN
$\land\ m.term > 0$
$\land\ m.master = n$
$\land\ SendRequest(n, [$
$type \mapsto WriteRequest,$
$election\_id \mapsto m.term])$
$\land$ UNCHANGED $\langle mastershipVars,\ nodeVars,\ deviceVars,\ streams\rangle$

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
$\land\ m$ *TODO*: Handle the write response
$\land$ UNCHANGED $\langle mastershipVars,\ deviceVars,\ streams\rangle$

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 master for the given elections

$Master(e) \triangleq Max(Range(e))$

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 $![n] = \langle\rangle]$
$\land$ UNCHANGED $\langle mastershipVars,\ nodeVars,\ deviceVars\rangle$

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.

$CloseStream(n) \triangleq$

```
    ∧ streams[n] = Open
    ∧ elections' = [elections EXCEPT ![n] = 0]
    ∧ streams' = [streams EXCEPT ![n] = Closed]
    ∧ requests' = [requests EXCEPT ![n] = ⟨⟩]
    ∧ LET oldMaster ≜ Master(elections)
          newMaster ≜ Master(elections')
      IN
          ∨ ∧ oldMaster ≠ newMaster
            ∧ responses' = [i ∈ DOMAIN streams' ↦ Append(responses[i], [
                                                    type        ↦ MasterArbitrationUpdate,
                                                    status      ↦ Ok,
                                                    election_id ↦ newMaster])]
          ∨ ∧ oldMaster = newMaster
            ∧ responses' = [responses EXCEPT ![n] = ⟨⟩]
    ∧ UNCHANGED ⟨mastershipVars, nodeVars⟩
```

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) ≜
    ∧ streams[n] = Open
    ∧ HasRequest(n, MasterArbitrationUpdate)
    ∧ LET m ≜ NextRequest(n)
      IN
          ∨ ∧ m.election_id ∈ elections
            ∧ SendResponse(n, [
                  type        ↦ MasterArbitrationUpdate,
                  election_id ↦ m.election_id,
                  status      ↦ AlreadyExists])
            ∧ UNCHANGED ⟨deviceVars⟩
          ∨ ∧ LET oldMaster ≜ Master(elections)
                  newMaster ≜ Master(elections')
            IN
                ∨ ∧ oldMaster ≠ newMaster
                  ∧ responses' = [i ∈ DOMAIN streams ↦ Append(responses[i], [
                                                         type        ↦ MasterArbitrationUpdate,
                                                         status      ↦ Ok,
                                                         election_id ↦ newMaster])]
                ∨ ∧ oldMaster = newMaster
                  ∧ SendResponse(n, [
                        type        ↦ MasterArbitrationUpdate,
                        status      ↦ Ok,
                        election_id ↦ newMaster])
    ∧ DiscardRequest(n)
    ∧ UNCHANGED ⟨mastershipVars, nodeVars, streams⟩
```

Handles a write request on the device

$$
\begin{array}{l}
HandleWrite(n) \triangleq \\
\quad \land\ streams[n] = Open \\
\quad \land\ HasRequest(n,\ WriteRequest) \\
\quad \land\ \text{LET}\ m \triangleq NextRequest(n) \\
\quad\ \ \text{IN} \\
\qquad \lor\ \land\ Cardinality(\text{DOMAIN}\ elections) = 0 \\
\qquad\quad \land\ SendResponse(n,\ [\\
\qquad\qquad\quad type \mapsto WriteResponse, \\
\qquad\qquad\quad status \mapsto PermissionDenied]) \\
\qquad \lor\ \land\ Max(Range(elections)) \neq m.election\_id \\
\qquad\quad \land\ SendResponse(n,\ [\\
\qquad\qquad\quad type \mapsto WriteResponse, \\
\qquad\qquad\quad status \mapsto PermissionDenied]) \\
\qquad \lor\ \land\ m.election\_id \notin elections \\
\qquad\quad \land\ elections[n] = m.election\_id \\
\qquad\quad \land\ SendResponse(n,\ [\\
\qquad\qquad\quad type \mapsto WriteResponse, \\
\qquad\qquad\quad status \mapsto Ok]) \\
\quad \land\ \text{UNCHANGED}\ \langle mastershipVars,\ nodeVars,\ deviceVars\rangle
\end{array}
$$

$$
\begin{array}{l}
Init \triangleq \\
\quad \land\ term = 0 \\
\quad \land\ master = Nil \\
\quad \land\ backups = \langle\rangle \\
\quad \land\ masterships = [n \in \text{DOMAIN}\ Nodes \mapsto [term \mapsto 0,\ master \mapsto 0,\ backups \mapsto \langle\rangle]] \\
\quad \land\ streams = [n \in \text{DOMAIN}\ Nodes \mapsto Closed] \\
\quad \land\ requests = [n \in \text{DOMAIN}\ Nodes \mapsto \langle\rangle] \\
\quad \land\ responses = [n \in \text{DOMAIN}\ Nodes \mapsto \langle\rangle] \\
\quad \land\ elections\ = [i\ \in \text{DOMAIN}\ Nodes \mapsto 0]
\end{array}
$$

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

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

* Modification History
* Last modified *Fri Feb* 15 18:05:05 *PST* 2019 by *jordanhalterman*
* Created *Thu Feb* 14 11:33:03 *PST* 2019 by *jordanhalterman*