
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*

Mastership/consensus related variables

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

Node related variables

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

Stream related variables

$streamVars \triangleq \langle streams, streamChanges \rangle$

Message related variables
 $messageVars \triangleq \langle requests, responses, messageCount \rangle$

Device related variables
 $deviceVars \triangleq \langle elections \rangle$

A sequence of all variables
 $vars \triangleq \langle mastershipVars, nodeVars, streamVars, messageVars, deviceVars \rangle$

Helpers

Returns a sequence with the head removed
 $Pop(q) \triangleq SubSeq(q, 2, Len(q))$

Returns a sequences with the element at the given index removed
 $Drop(q, i) \triangleq SubSeq(q, 1, i - 1) \circ SubSeq(q, i + 1, Len(q))$

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'
 $SendRequest(n, m) \triangleq$
 $\wedge requests' = [requests \text{ EXCEPT } ![n] = Append(requests[n], m)]$
 $\wedge messageCount' = messageCount + 1$

Indicates whether any requests are in the queue for node 'n'
 $HasRequest(n, t) \triangleq Len(requests[n]) > 0 \wedge requests[n][1].type = t$

Returns the next request in the queue for node 'n'
 $NextRequest(n) \triangleq requests[n][1]$

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

Indicates whether any responses are in the queue for node 'n'
 $HasResponse(n, t) \triangleq Len(responses[n]) > 0 \wedge responses[n][1].type = t$

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 distinct 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
 $JoinMastershipElection(n) \triangleq$
 $\wedge \vee \wedge master = Nil$
 $\wedge term' = term + 1$
 $\wedge master' = n$
 $\wedge backups' = \langle \rangle$
 $\wedge events' = [i \in Nodes \mapsto Append(events[i], [$
 $term \mapsto term',$
 $master \mapsto master',$
 $backups \mapsto backups'])]$
 $\vee \wedge master \neq Nil$
 $\wedge n \notin Range(backups)$
 $\wedge backups' = Append(backups, n)$
 $\wedge UNCHANGED \langle term, master, events \rangle$
 $\wedge mastershipChanges' = mastershipChanges + 1$
 $\wedge UNCHANGED \langle masterships, streamVars, messageVars, deviceVars \rangle$

Removes a node from the mastership election
 $LeaveMastershipElection(n) \triangleq$
 $\wedge \vee \wedge master = n$
 $\wedge \vee \wedge Len(backups) > 0$
 $\wedge term' = term + 1$
 $\wedge master' = backups[1]$
 $\wedge backups' = Pop(backups)$
 $\wedge events' = [i \in Nodes \mapsto Append(events[i], [$
 $term \mapsto term',$
 $master \mapsto master',$
 $backups \mapsto backups'])]$
 $\vee \wedge Len(backups) = 0$
 $\wedge master' = Nil$

$$\begin{aligned}
& \wedge \text{UNCHANGED } \langle term, backups, events \rangle \\
& \vee \wedge n \in \text{Range}(backups) \\
& \quad \wedge backups' = \text{Drop}(backups, \text{CHOOSE } j \in \text{DOMAIN } backups : backups[j] = n) \\
& \quad \wedge \text{UNCHANGED } \langle term, master, events \rangle \\
& \wedge mastershipChanges' = mastershipChanges + 1 \\
& \wedge \text{UNCHANGED } \langle masterships, streamVars, messageVars, deviceVars \rangle
\end{aligned}$$

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

$$\begin{aligned}
\text{SetMastership}(n) & \triangleq \\
& \wedge \vee \wedge master = n \\
& \quad \wedge \text{UNCHANGED } \langle mastershipVars \rangle \\
& \vee \wedge master \neq n \\
& \quad \wedge term' = term + 1 \\
& \quad \wedge master' = n \\
& \wedge \vee \wedge n \in \text{Range}(backups) \\
& \quad \wedge backups' = \text{Drop}(backups, \text{CHOOSE } j \in \text{DOMAIN } backups : backups[j] = n) \\
& \vee \wedge n \notin \text{Range}(backups) \\
& \quad \wedge \text{UNCHANGED } \langle backups \rangle \\
& \wedge mastershipChanges' = mastershipChanges + 1
\end{aligned}$$

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

$$\begin{aligned}
\text{LearnMastership}(n) & \triangleq \\
& \wedge \text{Len}(events[n]) > 0 \\
& \wedge \text{LET } m \triangleq events[n][1] \\
& \text{IN} \\
& \quad \wedge masterships' = [masterships \text{ EXCEPT } ![n] = [\\
& \quad \quad \quad term \mapsto m.term, \\
& \quad \quad \quad master \mapsto m.master, \\
& \quad \quad \quad backups \mapsto m.backups, \\
& \quad \quad \quad sent \mapsto \text{FALSE}]] \\
& \wedge \text{UNCHANGED } \langle mastershipVars, events, streamVars, messageVars, deviceVars \rangle
\end{aligned}$$

Notifies the device of node 'n' mastership info if it hasn't already been sent

$$\begin{aligned}
\text{SendMasterArbitrationUpdateRequest}(n) & \triangleq \\
& \wedge masterships[n].term > 0 \\
& \wedge \neg masterships[n].sent \\
& \wedge streams[n] = \text{Open} \\
& \wedge \text{LET } m \triangleq masterships[n] \\
& \text{IN} \\
& \quad \wedge m.term > 0 \\
& \quad \wedge \neg m.sent \\
& \quad \wedge \vee \wedge m.master = n \\
& \quad \quad \wedge \text{SendRequest}(n, [\\
& \quad \quad \quad type \mapsto \text{MasterArbitrationUpdate}, \\
& \quad \quad \quad election_id \mapsto m.term]) \\
& \quad \vee \wedge m.master \neq n
\end{aligned}$$

$$\wedge \text{SendRequest}(n, [\\ \text{type} \quad \mapsto \text{MasterArbitrationUpdate}, \\ \text{election_id} \mapsto 0]) \\ \wedge \text{UNCHANGED } \langle \text{mastershipVars}, \text{nodeVars}, \text{deviceVars}, \text{streamVars}, \text{responses} \rangle$$

Receives a master arbitration update response on node 'n'
 $\text{ReceiveMasterArbitrationUpdateResponse}(n) \triangleq$

$$\begin{aligned} & \wedge \text{streams}[n] = \text{Open} \\ & \wedge \text{HasResponse}(n, \text{MasterArbitrationUpdate}) \\ & \wedge \text{LET } m \triangleq \text{NextResponse}(n) \\ & \text{IN} \\ & \quad \vee \wedge m.\text{status} = \text{Ok} \\ & \quad \quad \wedge \text{SetMastership}(n) \\ & \quad \vee \wedge m.\text{status} = \text{AlreadyExists} \\ & \quad \quad \wedge \text{UNCHANGED } \langle \text{mastershipVars} \rangle \\ & \wedge \text{DiscardResponse}(n) \\ & \wedge \text{UNCHANGED } \langle \text{nodeVars}, \text{deviceVars}, \text{streamVars}, \text{requests}, \text{messageCount} \rangle \end{aligned}$$

Sends a write request to the device from node 'n'
 $\text{SendWriteRequest}(n) \triangleq$

$$\begin{aligned} & \wedge \text{streams}[n] = \text{Open} \\ & \wedge \text{LET } m \triangleq \text{masterships}[n] \\ & \text{IN} \\ & \quad \wedge m.\text{term} > 0 \\ & \quad \wedge m.\text{master} = n \\ & \quad \wedge \text{SendRequest}(n, [\\ & \quad \quad \text{type} \quad \mapsto \text{WriteRequest}, \\ & \quad \quad \text{election_id} \mapsto m.\text{term}]) \\ & \wedge \text{UNCHANGED } \langle \text{mastershipVars}, \text{nodeVars}, \text{deviceVars}, \text{streamVars}, \text{responses} \rangle \end{aligned}$$

Receives a write response on node 'n'
 $\text{ReceiveWriteResponse}(n) \triangleq$

$$\begin{aligned} & \wedge \text{streams}[n] = \text{Open} \\ & \wedge \text{HasResponse}(n, \text{WriteResponse}) \\ & \wedge \text{LET } m \triangleq \text{NextResponse}(n) \\ & \text{IN} \\ & \quad \text{TODO: This should be used to determine whether writes from old masters are allowed} \\ & \quad \vee m.\text{status} = \text{Ok} \\ & \quad \vee m.\text{status} = \text{PermissionDenied} \\ & \wedge \text{DiscardResponse}(n) \\ & \wedge \text{UNCHANGED } \langle \text{mastershipVars}, \text{nodeVars}, \text{deviceVars}, \text{streamVars}, \text{requests}, \text{messageCount} \rangle \end{aligned}$$

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

$$Election(e) \triangleq \text{Max}(\text{Range}(e))$$

Returns the master for the given elections

$$Master(e) \triangleq \text{CHOOSE } n \in \text{DOMAIN } e : e[n] = Election(e)$$

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*'.

$$\begin{aligned} ConnectStream(n) &\triangleq \\ &\wedge \text{streams}[n] = \text{Closed} \\ &\wedge \text{streams}' = [\text{streams} \text{ EXCEPT } ![n] = \text{Open}] \\ &\wedge \text{requests}' = [\text{requests} \text{ EXCEPT } ![n] = \langle \rangle] \\ &\wedge \text{responses}' = [\text{responses} \text{ EXCEPT } ![n] = \langle \rangle] \\ &\wedge \text{streamChanges}' = \text{streamChanges} + 1 \\ &\wedge \text{UNCHANGED } \langle \text{mastershipVars}, \text{nodeVars}, \text{deviceVars}, \text{messageCount} \rangle \end{aligned}$$

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.

$$\begin{aligned} CloseStream(n) &\triangleq \\ &\wedge \text{streams}[n] = \text{Open} \\ &\wedge \text{elections}' = [\text{elections} \text{ EXCEPT } ![n] = 0] \\ &\wedge \text{streams}' = [\text{streams} \text{ EXCEPT } ![n] = \text{Closed}] \\ &\wedge \text{requests}' = [\text{requests} \text{ EXCEPT } ![n] = \langle \rangle] \\ &\wedge \text{LET } \text{oldMaster} \triangleq Master(\text{elections}) \\ &\quad \text{newMaster} \triangleq Master(\text{elections}') \\ &\text{IN} \\ &\quad \vee \wedge \text{oldMaster} \neq \text{newMaster} \\ &\quad \quad \wedge \text{responses}' = [i \in \text{DOMAIN } \text{streams}' \mapsto \\ &\quad \quad \quad \text{IF } i = \text{newMaster} \text{ THEN} \\ &\quad \quad \quad \quad \text{Append}(\text{responses}[i], [\\ &\quad \quad \quad \quad \quad \text{type} \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\ &\quad \quad \quad \quad \quad \text{status} \quad \mapsto \text{Ok}, \\ &\quad \quad \quad \quad \quad \text{election_id} \mapsto \text{newMaster}]) \\ &\quad \quad \quad \text{ELSE} \\ &\quad \quad \quad \quad \text{Append}(\text{responses}[i], [\\ &\quad \quad \quad \quad \quad \text{type} \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\ &\quad \quad \quad \quad \quad \text{status} \quad \mapsto \text{AlreadyExists}, \\ &\quad \quad \quad \quad \quad \text{election_id} \mapsto \text{newMaster}])] \\ &\quad \vee \wedge \text{oldMaster} = \text{newMaster} \end{aligned}$$

$$\begin{aligned}
& \wedge responses' = [responses \text{ EXCEPT } ![n] = \langle \rangle] \\
& \wedge streamChanges' = streamChanges + 1 \\
& \wedge \text{UNCHANGED } \langle mastershipVars, nodeVars, messageCount \rangle
\end{aligned}$$

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,

$$\begin{aligned}
& \text{HandleMasterArbitrationUpdate}(n) \triangleq \\
& \quad \wedge streams[n] = \text{Open} \\
& \quad \wedge \text{HasRequest}(n, \text{MasterArbitrationUpdate}) \\
& \quad \wedge \text{LET } m \triangleq \text{NextRequest}(n) \\
& \quad \text{IN} \\
& \quad \quad \vee \wedge m.election_id \in \text{Range}(elections) \\
& \quad \quad \quad \wedge \text{SendResponse}(n, [\\
& \quad \quad \quad \quad type \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\
& \quad \quad \quad \quad election_id \mapsto m.election_id, \\
& \quad \quad \quad \quad status \quad \quad \mapsto \text{AlreadyExists}]) \\
& \quad \quad \quad \wedge \text{UNCHANGED } \langle deviceVars \rangle \\
& \quad \quad \vee \wedge m.election_id \notin \text{Range}(elections) \\
& \quad \quad \quad \wedge elections' = [elections \text{ EXCEPT } ![n] = m.election_id] \\
& \quad \quad \quad \wedge \text{LET } oldMaster \triangleq \text{Master}(elections) \\
& \quad \quad \quad \quad newMaster \triangleq \text{Master}(elections') \\
& \quad \quad \quad \text{IN} \\
& \quad \quad \quad \quad \vee \wedge oldMaster \neq newMaster \\
& \quad \quad \quad \quad \quad \wedge responses' = [i \in \text{DOMAIN } streams \mapsto \\
& \quad \quad \quad \quad \quad \quad \text{IF } i = newMaster \text{ THEN} \\
& \quad \quad \quad \quad \quad \quad \quad \text{Append}(responses[i], [\\
& \quad \quad \quad \quad \quad \quad \quad \quad type \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\
& \quad \quad \quad \quad \quad \quad \quad \quad status \quad \mapsto \text{Ok}, \\
& \quad \quad \quad \quad \quad \quad \quad \quad election_id \mapsto newMaster]) \\
& \quad \quad \quad \quad \quad \quad \text{ELSE} \\
& \quad \quad \quad \quad \quad \quad \quad \text{Append}(responses[i], [\\
& \quad \quad \quad \quad \quad \quad \quad \quad type \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\
& \quad \quad \quad \quad \quad \quad \quad \quad status \quad \mapsto \text{AlreadyExists}, \\
& \quad \quad \quad \quad \quad \quad \quad \quad election_id \mapsto newMaster})] \\
& \quad \quad \quad \quad \quad \quad \wedge messageCount = messageCount + 1 \\
& \quad \quad \quad \quad \vee \wedge oldMaster = newMaster \\
& \quad \quad \quad \quad \quad \wedge \text{SendResponse}(n, [\\
& \quad \quad \quad \quad \quad \quad type \quad \quad \mapsto \text{MasterArbitrationUpdate}, \\
& \quad \quad \quad \quad \quad \quad status \quad \mapsto \text{Ok}, \\
& \quad \quad \quad \quad \quad \quad election_id \mapsto newMaster]) \\
& \quad \quad \quad \quad \wedge \text{DiscardRequest}(n) \\
& \quad \quad \quad \quad \wedge \text{UNCHANGED } \langle mastershipVars, nodeVars, streamVars \rangle
\end{aligned}$$

Handles a write request on the device

$$\begin{aligned}
\text{HandleWrite}(n) &\triangleq \\
&\wedge \text{streams}[n] = \text{Open} \\
&\wedge \text{HasRequest}(n, \text{WriteRequest}) \\
&\wedge \text{LET } m \triangleq \text{NextRequest}(n) \\
&\text{IN} \\
&\quad \vee \wedge \text{Cardinality}(\text{DOMAIN } \text{elections}) = 0 \\
&\quad \wedge \text{SendResponse}(n, [\\
&\quad \quad \text{type} \mapsto \text{WriteResponse}, \\
&\quad \quad \text{status} \mapsto \text{PermissionDenied}]) \\
&\quad \vee \wedge \text{Election}(\text{elections}) \neq m.\text{election_id} \\
&\quad \wedge \text{SendResponse}(n, [\\
&\quad \quad \text{type} \mapsto \text{WriteResponse}, \\
&\quad \quad \text{status} \mapsto \text{PermissionDenied}]) \\
&\quad \vee \wedge m.\text{election_id} \notin \text{Range}(\text{elections}) \\
&\quad \wedge \text{elections}[n] = m.\text{election_id} \\
&\quad \wedge \text{SendResponse}(n, [\\
&\quad \quad \text{type} \mapsto \text{WriteResponse}, \\
&\quad \quad \text{status} \mapsto \text{Ok}]) \\
&\wedge \text{DiscardRequest}(n) \\
&\wedge \text{UNCHANGED } \langle \text{mastershipVars}, \text{nodeVars}, \text{deviceVars}, \text{streamVars} \rangle
\end{aligned}$$

$$\begin{aligned}
\text{Init} &\triangleq \\
&\wedge \text{term} = 0 \\
&\wedge \text{master} = \text{Nil} \\
&\wedge \text{backups} = \langle \rangle \\
&\wedge \text{events} = [n \in \text{Nodes} \mapsto \langle \rangle] \\
&\wedge \text{masterships} = [n \in \text{Nodes} \mapsto [\text{term} \mapsto 0, \text{master} \mapsto 0, \text{backups} \mapsto \langle \rangle]] \\
&\wedge \text{streams} = [n \in \text{Nodes} \mapsto \text{Closed}] \\
&\wedge \text{requests} = [n \in \text{Nodes} \mapsto \langle \rangle] \\
&\wedge \text{responses} = [n \in \text{Nodes} \mapsto \langle \rangle] \\
&\wedge \text{elections} = [n \in \text{Nodes} \mapsto 0] \\
&\wedge \text{mastershipChanges} = 0 \\
&\wedge \text{streamChanges} = 0 \\
&\wedge \text{messageCount} = 0
\end{aligned}$$

$$\begin{aligned}
\text{Next} &\triangleq \\
&\vee \exists n \in \text{Nodes} : \text{ConnectStream}(n) \\
&\vee \exists n \in \text{Nodes} : \text{CloseStream}(n) \\
&\vee \exists n \in \text{Nodes} : \text{JoinMastershipElection}(n) \\
&\vee \exists n \in \text{Nodes} : \text{LeaveMastershipElection}(n) \\
&\vee \exists n \in \text{Nodes} : \text{LearnMastership}(n) \\
&\vee \exists n \in \text{Nodes} : \text{SendMasterArbitrationUpdateRequest}(n) \\
&\vee \exists n \in \text{Nodes} : \text{HandleMasterArbitrationUpdate}(n) \\
&\vee \exists n \in \text{Nodes} : \text{ReceiveMasterArbitrationUpdateResponse}(n)
\end{aligned}$$

$\forall \exists n \in Nodes : SendWriteRequest(n)$
 $\forall \exists n \in Nodes : HandleWrite(n)$
 $\forall \exists n \in Nodes : ReceiveWriteResponse(n)$

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

\ * Modification History
 \ * Last modified Sat Feb 16 01:00:10 PST 2019 by jordanhalterman
 \ * Created Thu Feb 14 11:33:03 PST 2019 by jordanhalterman