MODULE *EHS_ChaosPrevention*

EHS_ChaosPrevention is a spec for automatic failover protection mechanism in *Ehs EHS* (*EdgeHealthService*) is a monitoring service running in each of Azure *Frontdoor*'s Edge which determines whether that Edge can serve user traffic or not. When an Edge is determined not fit to serve traffic then it is turned off. In order to prevent global outage of all *Edges*, we need this protection mechanism to prevent all *EHS* instances from turning off all *Edges* at the same time.

EXTENDS *FiniteSets*, *Integers*

CONSTANT
Edges, Set of *Edges* serving user traffic
NumEdgesAllowedToTurnOff Number of *Edges* allowed to turn off automatically.

ASSUME $NumEdgesAllowedToTurnOff \leq Cardinality(Edges)$

VARIABLES
edgeState, State of *Edges* indicating indicating whether it can serve traffic or not.
ehsDecision, *EHS* decision for the respective edge
messagingState, Messaging state of an edge indicating whether it has sent its message to other edges or not.
msgs Messages sent to other *Edges*

State Filters

$EdgesInOffState \triangleq \{e \in Edges : edgeState[e] = \text{“Off”}\}$

$EdgesInTurningOffState \triangleq \{e \in Edges : edgeState[e] = \text{“TurningOff”}\}$

Model *Messages*

$Messages \triangleq$
$[type : \{\text{“StatusRequest”}\}, sender : EdgesInTurningOffState, receiver : Edges]$
$\cup$
$[type : \{\text{“StatusResponse”}\}, sender : Edges, receiver : EdgesInTurningOffState, val : \{\text{“On”}, \text{“TurningOff”}, \text{“O}$

Model *Init*

$EhsInit \triangleq$
$\land\ edgeState = [edge \in Edges \mapsto \text{“On”}]$
$\land\ ehsDecision = [ehs \in Edges \mapsto \text{“Healthy”}]$
$\land\ messagingState = [ehs \in Edges \mapsto \text{“Reset”}]$
$\land\ msgs = \{\}$

Message Actions

$SendMessage(m) \triangleq msgs' = msgs \cup \{m\}$

$SendMessages(m) \triangleq msgs' = msgs \cup m$

Message Filtering

$HaveGotAllStatusResponseMessages(e) \triangleq$

Determines whether an Edge has got responses from all the *Edges* to which it sent the request to.

LET $allSentMessages \triangleq \{m \in msgs : \wedge m.type = \text{“StatusRequest”} \wedge m.sender = e\}$

$receivedMessagesForLatestSeq \triangleq \{m \in msgs : \wedge m.type = \text{“StatusResponse”} \wedge m.receiver = e\}$

IN $\wedge Cardinality(allSentMessages) = Cardinality(receivedMessagesForLatestSeq)$

$CompleteStatusRequestMessage(e) \triangleq$

Complete status request message seq once by removing the request & response messages for a sequence

LET $messagesBelongingToEdge \triangleq \{m \in msgs : \vee (m.type = \text{“StatusRequest”} \wedge m.sender = e) \vee (m.type = \text{“StatusResponse”} \wedge m.receiver = e)\}$

IN $\wedge msgs' = msgs \setminus messagesBelongingToEdge$

$EdgesThatRespondWithGivenStateForLatestRequest(state,\ e) \triangleq$

Returns that responded with the given state (On, *Off*, Turning off or *Timeout*) for the latest Status Request message

$\{m1.sender : m1 \in \{m \in msgs : \wedge m.type = \text{“StatusResponse”} \wedge m.receiver = e \wedge m.val = state\}\}$

$StateOfTargetEdgeAsPerEdge(targetEdge,\ e) \triangleq$

Returns what is the state of the edge 'TargetEdge' in the view of edge 'e' based on the response of 'TargetEdge' to the latest *StatusRequest* messsage from edge 'e'

$\{m1.val : m1 \in \{m \in msgs : \wedge m.type = \text{“StatusResponse”} \wedge m.receiver = e \wedge m.sender = targetEdge\}\}$

$EdgesNotInOnStateInTheViewOfEdge(e) \triangleq$

Get the list of *Edges* which are not in the *On* state. That is any edge which responsed with *Off*, Turning *Off* or *Timeout* as status. Also, include the edge itself.

$EdgesThatRespondWithGivenStateForLatestRequest(\text{“Off”},\ e)$

$\cup$

$EdgesThatRespondWithGivenStateForLatestRequest(\text{“TurningOff”},\ e)$

$\cup$

$EdgesThatRespondWithGivenStateForLatestRequest(\text{“Timeout”},\ e)$

$\cup$

IF $edgeState[e] \neq \text{“On”}$ THEN $\{e\}$ ELSE $\{\}$

$CanTurnOff(e) \triangleq$

An Edge can turn off only if (total number of edges which responded as *Off*, *TurnedOff* or *Timeout*) $\leq$ Number *Edges* Allowed to Turn *Off*

$\wedge Cardinality(EdgesNotInOnStateInTheViewOfEdge(e)) \leq NumEdgesAllowedToTurnOff$

State Changes

$EhsHealthyToUnHealthy(e) \triangleq$

EHS can at anytime determine that edge goes from *Healthy* to *UnHealthy* state

$\land\ ehsDecision[e] = \text{“Healthy”}$
$\land\ ehsDecision' = [ehsDecision \text{ EXCEPT } ![e] = \text{“Unhealthy”}]$
$\land\ \text{UNCHANGED } \langle edgeState,\ msgs,\ messagingState \rangle$

$EhsUnHealthyToHealthy(e) \triangleq$

EHS can at anytime determine that edge goes from *UnHealthy* to *Healthy* state

$\land\ ehsDecision[e] = \text{“Unhealthy”}$
$\land\ ehsDecision' = [ehsDecision \text{ EXCEPT } ![e] = \text{“Healthy”}]$
$\land\ \text{UNCHANGED } \langle edgeState,\ msgs,\ messagingState \rangle$

$EdgeOnToTurningOff(e) \triangleq$

When *Ehs′* decision is *Unhealthy* then the Edge goes to *TurningOff* state if it is in *On* state.

$\land\ ehsDecision[e] = \text{“Unhealthy”}$
$\land\ edgeState[e] = \text{“On”}$
$\land\ edgeState' = [edgeState \text{ EXCEPT } ![e] = \text{“TurningOff”}]$
$\land\ \text{UNCHANGED } \langle ehsDecision,\ msgs,\ messagingState \rangle$

$EdgeTurningOffToOn(e) \triangleq$

When *Ehs′* decision is *Healthy* then the Edge goes to *On* state from Turning *Off* state, provided it has completed it messaging process which it started when it went to Turning *Off* state

$\land\ ehsDecision[e] = \text{“Healthy”}$
$\land\ edgeState[e] = \text{“TurningOff”}$
$\land\ messagingState[e] = \text{“Completed”}$
$\land\ edgeState' = [edgeState \text{ EXCEPT } ![e] = \text{“On”}]$
$\land\ messagingState' = [messagingState \text{ EXCEPT } ![e] = \text{“Reset”}]$
$\land\ CompleteStatusRequestMessage(e)$
$\land\ \text{UNCHANGED } \langle ehsDecision \rangle$

$EdgeTurningOffToTurningOff(e) \triangleq$

This is a case where the Edge can't move away from the Turning *Off* state because there are too many edges in turning off or off state or have not responded to the status request message

$\land\ ehsDecision[e] = \text{“Unhealthy”}$
$\land\ edgeState[e] = \text{“TurningOff”}$
$\land\ messagingState[e] = \text{“Completed”}$
$\land\ \neg CanTurnOff(e)$
$\land\ messagingState' = [messagingState \text{ EXCEPT } ![e] = \text{“Reset”}]$
$\land\ CompleteStatusRequestMessage(e)$
$\land\ \text{UNCHANGED } \langle edgeState,\ ehsDecision \rangle$

$EdgeTurningOffToOff(e) \triangleq$

Edge has gotten responses from other *Edges* and it determines that the total number of edges not in *On* state are below the allowed threshold. Hence, it can go to *Off* State from Turning *Off* state as it is still unhealthy.

$\land\ ehsDecision[e] = \text{“Unhealthy”}$

$\land$ $edgeState[e]$ = "TurningOff"
$\land$ $messagingState[e]$ = "Completed"
$\land$ $CanTurnOff(e)$
$\land$ $edgeState' = [edgeState$ EXCEPT $![e]$ = "Off"$]$
$\land$ $messagingState' = [messagingState$ EXCEPT $![e]$ = "Reset"$]$
$\land$ $CompleteStatusRequestMessage(e)$
$\land$ UNCHANGED $\langle ehsDecision \rangle$

$EdgeOffToOn(e) \triangleq$

When *Ehs′* decision is *Healthy* then the Edge goes to *On* state if it is in *Off* state.

$\land$ $ehsDecision[e]$ = "Healthy"
$\land$ $edgeState[e]$ = "Off"
$\land$ $edgeState' = [edgeState$ EXCEPT $![e]$ = "On"$]$
$\land$ UNCHANGED $\langle ehsDecision, msgs, messagingState \rangle$

Messaging Actions

$NotRespondedToStatusRequestMessage(e, statusRequestMessage) \triangleq$
$\land \neg(\exists\, m \in msgs :$ $\land$ $m.type$ = "StatusResponse"
$\land$ $m.sender = e$
$\land$ $m.receiver = statusRequestMessage.sender)$

$EdgeProcessResponses(e) \triangleq$

Once an Edge has received all the responses for its requests then the message has completed.

$\land$ $edgeState[e]$ = "TurningOff"
$\land$ $ehsDecision[e]$ = "Unhealthy"
$\land$ $messagingState[e]$ = "Sent"
$\land$ $HaveGotAllStatusResponseMessages(e)$
$\land$ $messagingState' = [messagingState$ EXCEPT $![e]$ = "Completed"$]$
$\land$ UNCHANGED $\langle ehsDecision, edgeState, msgs \rangle$

$EdgeSendsRequest(e) \triangleq$

When an edge comes to Turning *Off*, then it will send a request to all other edges asking their state.

$\land$ $edgeState[e]$ = "TurningOff"
$\land$ $ehsDecision[e]$ = "Unhealthy"
$\land$ $messagingState[e]$ = "Reset"
$\land$ $messagingState' = [messagingState$ EXCEPT $![e]$ = "Sent"$]$
$\land$ $SendMessages(\{[type \mapsto$ "StatusRequest", $sender \mapsto e, receiver \mapsto e1] : e1 \in Edges \setminus \{e\}\})$
$\land$ UNCHANGED $\langle ehsDecision, edgeState \rangle$

$EdgeRespondsWithStatus(e) \triangleq$

Whenever there is a pending request, asking its state, for the edge then the edge responds with it state.

$\land \exists\, m \in msgs :$
$\land$ $m.type$ = "StatusRequest"
$\land$ $m.receiver = e$
$\land$ $NotRespondedToStatusRequestMessage(e, m)$

$\land$ $SendMessage([type \mapsto$ "StatusResponse", $sender \mapsto e,\ receiver \mapsto m.sender,\ val \mapsto edgeState[e]])$
$\land$ UNCHANGED $\langle ehsDecision,\ edgeState,\ messagingState \rangle$

$EdgeRespondsWithAbort(e) \triangleq$

At times an edge can respond with a timeout message for pending state request to it.
This simulates request timeouts

$\land \exists\, m \in msgs :$
$\land\ m.type =$ "StatusRequest"
$\land\ m.receiver = e$
$\land\ NotRespondedToStatusRequestMessage(e,\ m)$
$\land\ SendMessage([type \mapsto$ "StatusResponse", $sender \mapsto e,\ receiver \mapsto m.sender,\ val \mapsto$ "Timeout"$])$
$\land$ UNCHANGED $\langle ehsDecision,\ edgeState,\ messagingState \rangle$

Model Invariants

$EhsTypeOK \triangleq$

Invariant ensuring that the different state don't take an invalid value.

$\land\ edgeState \in [Edges \to \{$"On", "TurningOff", "Off"$\}]$
$\land\ ehsDecision \in [Edges \to \{$"Healthy", "Unhealthy"$\}]$
$\land\ messagingState \in [Edges \to \{$"Reset", "Sent", "Completed"$\}]$

$EhsStateOK \triangleq$

Important Invariant which ensures that not too many *Edges* are in *Off* state thus preventing global outage.

$\land\ Cardinality(EdgesInOffState) \leq NumEdgesAllowedToTurnOff$

$EhsModelOk \triangleq$
$\land\ EhsTypeOK$
$\land\ EhsStateOK$

$NoEdgeHasLiedAboutItsState \triangleq$

Invariant which ensures that no edges gives a wrong reply to the *StatusRequest* Message.

$\neg(\exists\, m \in msgs :\ \land\ m.type =$ "StateResponse"
$\land\ (m.val \neq$ "Timeout" $\land\ edgeState[m.sender] \neq m.val))$

$NoTwoTurningOffEdgesSeeEachOtherInOnState \triangleq$

Invariant which ensures there are no race condition when two edges are turning off at the same time.
No two edges which are going to turning off state at the time should see each other in *On* state.
This ensures that the edges updates its state before requesting other edges about their respective state.

$\neg(\exists\, e1,\ e2 \in Edges :\ \land\ e1 \neq e2$
$\land\ edgeState[e1] =$ "TurningOff"
$\land\ edgeState[e2] =$ "TurningOff"
$\land\ messagingState[e1] =$ "Completed"
$\land\ messagingState[e2] =$ "Completed"
$\land\ StateOfTargetEdgeAsPerEdge(e2,\ e1) = \{$"On"$\}$
$\land\ StateOfTargetEdgeAsPerEdge(e1,\ e2) = \{$"On"$\})$

$$
\begin{array}{l}
EhsNext \triangleq \\
\quad \lor \exists\, e \in Edges : \\
\qquad EhsHealthyToUnHealthy(e) \lor EhsUnHealthyToHealthy(e) \\
\qquad\quad \lor EdgeOnToTurningOff(e) \lor EdgeTurningOffToOn(e) \lor EdgeTurningOffToOff(e) \lor EdgeTurningOff \\
\qquad\qquad \lor EdgeSendsRequest(e) \lor EdgeProcessResponses(e) \lor EdgeRespondsWithStatus(e) \lor EdgeRespond
\end{array}
$$

$$
EhsSpec \triangleq EhsInit \land \Box[EhsNext]_{\langle edgeState,\, ehsDecision,\, messagingState,\, msgs \rangle}
$$

* Modification History
* Last modified *Wed* May 04 19:02:51 *PDT* 2016 by *guhanr*
* Created *Wed* May 04 19:02:11 *PDT* 2016 by *guhanr*