## MODULE *SimplePathModel*

Implements a rate-limited, simple path model for packets traversing it to reach some destination. The model is able to unconditionally delay, drop, or deliver the packets to the destination as will.

*ACKs* are returned for each delivered packet (though the *ACK* itself can be dropped as well, but that behavior can be changed).

The model will trigger timeouts for dropped packets.

EXTENDS *Integers*, *TLC*, *Sequences*

*C* : The link capacity. After 't' timesteps, no more than 'C $* t'$ packets can be in the link. Any more is immediately dropped.
*MAX_ARRIVAL*: The maximum rate of packet arrivals.
*DROP_ACK* : A boolean constant, if TRUE, the model can drop *ACKs* on a whim.
*nAck* : The number of *ACKs* returned.
*inFlight* : The number of packets traversing the link.
timeout : If '$1'$, a packet has not reached it's destination and has timed out.

CONSTANT $C$, $MAX\_T$, $MAX\_ARRIVAL$, $DROP\_ACK$
VARIABLES $t$, $ticked$, $nAck$, $inFlight$, $timeout$

$$timeVars \triangleq \langle t, ticked \rangle$$
$$pathVars \triangleq \langle nAck, inFlight, timeout \rangle$$
$$vars \triangleq \langle t, ticked, nAck, inFlight, timeout \rangle$$

$$time \triangleq \text{INSTANCE } Time \text{ WITH } t \leftarrow t,\ ticked \leftarrow ticked,\ MAX\_T \leftarrow MAX\_T$$

$$\begin{aligned} TypeOK \triangleq\ & \land\ timeout \in 0 \,..\, 1 \\ & \land\ nAck \in Nat \\ & \land\ nAck \geq 0 \\ & \land\ inFlight \in Nat \\ & \land\ inFlight \geq 0 \\ & \land\ time!TypeOK \end{aligned}$$

$$\begin{aligned} Init \triangleq\ & \land\ timeout = 0 \\ & \land\ nAck = 0 \\ & \land\ inFlight = 0 \\ & \land\ time!Init \end{aligned}$$

$$Finished \triangleq time!Finished$$

When the link contains more than 't$*C'$ packets, excessive packets WILL be dropped immediately, but triggering a timeout and then dropping the number of packets to 't $* C'$.

Certain buffering strategies can be employed here to help, as well as some token bucket filters, but we will not implement that yet.

$$ExcessivePacketDropIsEnabled \triangleq\ \land\ inFlight > t * C$$

At the end of each timestep, the path model may accept new packets into the link as inflight packets.

$$PacketInjectionIsEnabled \triangleq \begin{array}{l} \land\ ticked = 1 \\ \land\ Finished = \text{FALSE} \end{array}$$

Path model is only enabled at the start of each timestep. Before it procedes to take actions per packet, it check whether or not excessive packets are present. If so, the packets will be dropped.

$$PathModelIsEnabled \triangleq \begin{array}{l} \land\ inFlight > 0 \\ \land\ ticked = 0 \\ \land\ Finished = \text{FALSE} \\ \land\ \neg ExcessivePacketDropIsEnabled \end{array}$$

Deliver packets and return *ACKs*

$$DeliverPacket \triangleq \begin{array}{l} \land\ PathModelIsEnabled \\ \land\ timeout' = 0 \\ \land\ nAck' = nAck + 1 \\ \land\ inFlight' = inFlight - 1 \\ \land\ \text{UNCHANGED}\ timeVars \end{array}$$

Deliver and return *ACK*, but trigger timeout

$$DeliverLate \triangleq \begin{array}{l} \land\ PathModelIsEnabled \\ \land\ timeout' = 1 \\ \land\ nAck' = nAck + 1 \\ \land\ inFlight' = inFlight - 1 \\ \land\ \text{UNCHANGED}\ timeVars \end{array}$$

Deliver the packet, but trigger timeout by dropping an *ACK*.
Can be disabled completely

$$DeliverAndDropAck \triangleq \begin{array}{l} \land\ PathModelIsEnabled \\ \land\ DROP\_ACK \\ \land\ timeout' = 1 \\ \land\ nAck' = nAck \\ \land\ inFlight' = inFlight - 1 \\ \land\ \text{UNCHANGED}\ timeVars \end{array}$$

Drop the packet completely and trigger timeout

$$DropCompletely \triangleq \begin{array}{l} \land\ PathModelIsEnabled \\ \land\ timeout' = 1 \\ \land\ nAck' = nAck \\ \land\ inFlight' = inFlight - 1 \\ \land\ \text{UNCHANGED}\ timeVars \end{array}$$

Drop packets exceeding the link capacity

$$\begin{array}{lll} \mathit{DropExcess} & \triangleq & \land \mathit{ExcessivePacketDropIsEnabled} \\ & & \land \mathit{timeout}' = 1 \\ & & \land \mathit{inFlight}' = t * C \\ & & \land \textsc{unchanged}\ \langle \mathit{timeVars},\ \mathit{nAck} \rangle \end{array}$$

$$\begin{array}{lll} \mathit{Next} & \triangleq & \lor \mathit{DeliverPacket} \\ & & \lor \mathit{DeliverLate} \\ & & \lor \mathit{DeliverAndDropAck} \\ & & \lor \mathit{DropCompletely} \\ & & \lor \mathit{DropExcess} \\ & & \lor\ \land \lnot \mathit{ExcessivePacketDropIsEnabled} \\ & & \quad \land \mathit{time}!\mathit{Next} \\ & & \quad \land \textsc{unchanged}\ \mathit{pathVars} \end{array}$$

Packet deliverance must remain strongly fair to prevent useless scenarios.

$$\begin{array}{lll} \mathit{Fairness} & \triangleq & \land \mathit{time}!\mathit{Fairness} \\ & & \land \mathrm{SF}_{\mathit{vars}}(\mathit{DeliverPacket}) \end{array}$$

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

Always either:

1) The link utility is less than or equal to 1
2) We are at the start of taking actions per packet, which means that if link utility is larger than 1, it will be immediately corrected.

$$\mathit{RateLimited} \triangleq \Box(\mathit{inFlight} \leq t * C \lor \mathit{ticked} = 0)$$

To test the specification, we create another spec on top of the spec above. This specification will allow for packet injections of random values and if everything goes well, the invariants must remain true.

$$\mathit{Max}(a,\ b) \triangleq \textsc{if}\ a > b\ \textsc{then}\ a\ \textsc{else}\ b$$

$$\mathit{newPacketsAllowed}(\mathit{timePassed},\ \mathit{existingPackets}) \triangleq \mathit{Max}(\mathit{timePassed} * \mathit{MAX\_ARRIVAL} - \mathit{existingPackets} -$$

$$\begin{array}{l} \mathit{getRandomArrival}(\mathit{timePassed},\ \mathit{existingPackets}) \triangleq \\ \quad \mathit{RandomElement}(0\ ..\ \mathit{newPacketsAllowed}(\mathit{timePassed},\ \mathit{existingPackets})) \end{array}$$

$$\begin{array}{lll} \mathit{InjectPackets} & \triangleq & \land \mathit{PacketInjectionIsEnabled} \\ & & \land \mathit{inFlight}' = \mathit{inFlight} + \mathit{getRandomArrival}(t,\ \mathit{inFlight}) \\ & & \land \mathit{nAck}' = \mathit{nAck} \\ & & \land \mathit{timeout}' = \mathit{timeout} \\ & & \land \mathit{time}!\mathit{DoTick} \end{array}$$

$$\begin{array}{lll} \mathit{NextTest} & \triangleq & \lor \mathit{Next} \\ & & \lor \mathit{InjectPackets} \end{array}$$

For the same reason that delivering packets must be strongly fair, having packets to deliver in the first place must also be strongly fair!

$$\begin{aligned} \mathit{FairnessTest} \triangleq\ & \land\ \mathit{time}\,!\,\mathit{Fairness} \\ & \land\ \mathrm{SF}_{\mathit{vars}}(\mathit{DeliverPacket}) \\ & \land\ \mathrm{SF}_{\mathit{vars}}(\mathit{InjectPackets}) \end{aligned}$$

$$\mathit{SpecTest} \triangleq \mathit{Init} \land \Box[\mathit{NextTest}]_{\mathit{vars}} \land \mathit{FairnessTest}$$

Termination condition just to check we have not overriden the time module.

$$\mathit{Termination} \triangleq \mathit{time}\,!\,\mathit{Termination}$$

\ ∗ Modification History
\ ∗ Last modified *Wed Nov* 02 15:51:35 *IRST* 2022 by *Arvin*
\ ∗ Created *Thu Oct* 27 00:16:46 *IRST* 2022 by *Arvin*