MODULE *AIMD*

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

The spec is parametrized with the following: 1. *SSTHRESH_START*: Initial value of the Slow-Start threshold
2. *MAX_WINDOW*: Maximum possible value of the congestion window
3. *NUM_PACKETS*: Number of packets to send

CONSTANTS *SSTHRESH_START*, *MAX_WINDOW*, *NUM_PACKETS*

For variables, we introduce the following:
1. *cwnd*: The current *TCP* congestion window
2. timeout: Whether or not a timeout is triggered
3. *nAck*: Number of collected *ACKs*
4. *inFlight*: Number of outstanding packets
5. *nPacket*: Number of packets to send
6. *ssthresh*: Slow-Start threshold

VARIABLES *cwnd*, *timeout*, *nAck*, *inFlight*, *nPacket*, *ssthresh*

$vars \triangleq \langle cwnd, timeout, nAck, inFlight, nPacket, ssthresh \rangle$

These assumptions are merely for safety

ASSUME $\land SSTHRESH\_START > 1$
$\land MAX\_WINDOW > SSTHRESH\_START$
$\land NUM\_PACKETS > 0$

The usual *TypeOK* invariant

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

$$
\begin{aligned}
Init \triangleq\ & \land cwnd = 1 \\
& \land timeout = 0 \\
& \land nAck = 0 \\
& \land inFlight = 0 \\
& \land nPacket = NUM\_PACKETS \\
& \land ssthresh = SSTHRESH\_START
\end{aligned}
$$

The window can be increased if and only if either:

1. We are in Slow-Start and have received an ACK
2. We are in Congestion-Avoidance and have received at least a whole window-worth of $ACKs$

All of these are subjected to no timeout happenning

$$
\begin{aligned}
CanIncreaseWindow \triangleq\ & \land\ timeout = 0 \\
& \land\ cwnd < MAX\_WINDOW \\
& \land\ \text{IF}\ cwnd < ssthresh \\
& \qquad \text{THEN}\ nAck > 0 \\
& \qquad \text{ELSE}\ nAck \geq cwnd
\end{aligned}
$$

$$
\begin{aligned}
IncreaseWindow \triangleq\ & \land\ CanIncreaseWindow \\
& \land\ \text{IF}\ cwnd < ssthresh \\
& \qquad \text{THEN}\ \land\ cwnd' = 2 * cwnd \\
& \qquad\qquad\ \land\ nAck' = nAck - 1 \\
& \qquad \text{ELSE}\ \land\ nAck' = nAck - cwnd \\
& \qquad\qquad\ \land\ cwnd' = cwnd + 1 \\
& \land\ \text{UNCHANGED}\ \langle ssthresh,\ timeout,\ inFlight,\ nPacket \rangle
\end{aligned}
$$

The window can be decreased if and only if a timeout has happened (in *Reno*, there is also duplicate $ACKs$, we'll add it later

$$
ShouldDecreaseWindow \triangleq\ timeout = 1
$$

$$
\begin{aligned}
DecreaseWindow \triangleq\ & \land\ ShouldDecreaseWindow \\
& \land\ \text{IF}\ cwnd \geq 4 \\
& \qquad \text{THEN}\ ssthresh' = cwnd \div 2 \\
& \qquad \text{ELSE}\ ssthresh' = 2 \\
& \land\ \text{IF}\ cwnd < ssthresh \\
& \qquad \text{THEN}\ \land\ cwnd' = 1 \\
& \qquad \text{ELSE}\ \land\ cwnd' = cwnd \div 2 \\
& \land\ timeout' = 0 \\
& \land\ nAck' = 0 \\
& \land\ \text{UNCHANGED}\ \langle inFlight,\ nPacket \rangle
\end{aligned}
$$

New packets can be sent if and only if:

1. A timeout has not occurred
2. There is actually something to send
3. The window has space

$$
\begin{aligned}
CanSendNewPacket \triangleq\ & \land\ timeout\ = 0 \\
& \land\ nPacket > 0 \\
& \land\ inFlight < cwnd
\end{aligned}
$$

$$
\begin{aligned}
SendNewPacket \triangleq\ & \land\ CanSendNewPacket \\
& \land\ nPacket' = nPacket - 1 \\
& \land\ inFlight' = inFlight + 1
\end{aligned}
$$

$\land$ UNCHANGED $\langle nAck,\ timeout,\ cwnd,\ ssthresh\rangle$

The path model can arbitrarily decide for each packet whether or not it gets delayed, dropped or delivered. The *ACKs* can also exhibit the same behaviors.

These behaviors are enabled only if there is at least one outstanding packet.

$PathModelIsEnabled \triangleq inFlight > 0$

In brief:
1. *DeliverPacket*: Delivers the packet in time, so no timeout happens and returns the an *ACK* to the *TCP*.
2. *DeliverLate*: The packet and the *ACK* are delivered and thus the *TCP* timeouts and the window is decreased.
3. *DeliverAndDropAck*: Packet is delivered but the *ACK* is dropped. *TCP* is forced to retransmit. Retransmits are simulated by magically adding a new packet for *TCP* to send.
4. *DropCompletely*: This is like above, it will be once we implement duplicate *ACK* detection.

$$\begin{aligned} DeliverPacket \triangleq\ & \land PathModelIsEnabled \\ & \land timeout' = 0 \\ & \land nAck' = nAck + 1 \\ & \land inFlight' = inFlight \\ & \land \text{UNCHANGED } \langle nPacket,\ cwnd,\ ssthresh\rangle \end{aligned}$$

$$\begin{aligned} DeliverLate \triangleq\ & \land PathModelIsEnabled \\ & \land timeout' = 1 \\ & \land nAck' = nAck + 1 \\ & \land inFlight' = inFlight - 1 \\ & \land \text{UNCHANGED } \langle nPacket,\ cwnd,\ ssthresh\rangle \end{aligned}$$

$$\begin{aligned} DeliverAndDropAck \triangleq\ & \land PathModelIsEnabled \\ & \land timeout' = 1 \\ & \land inFlight' = inFlight - 1 \\ & \land nPacket' = nPacket + 1 \\ & \land \text{UNCHANGED } \langle cwnd,\ ssthresh,\ nAck\rangle \end{aligned}$$

$$\begin{aligned} DropCompletely \triangleq\ & \land PathModelIsEnabled \\ & \land timeout' = 1 \\ & \land nPacket' = nPacket + 1 \\ & \land inFlight' = inFlight - 1 \\ & \land \text{UNCHANGED } \langle cwnd,\ ssthresh,\ nAck\rangle \end{aligned}$$

$$\begin{aligned} Next \triangleq\ & \lor SendNewPacket \\ & \lor IncreaseWindow \\ & \lor DecreaseWindow \\ & \lor DeliverPacket \\ & \lor DeliverLate \\ & \lor DeliverAndDropAck \end{aligned}$$

$\lor$ *DropCompletely*

Some fairness specification is needed to prevent the the path model from creating useless behaviors.

1. Weak fairness must hold for *SendNewPacket*. This prevents behaviors where *TCP* just stands there and refuses to do anything.
2. Strong fairness is needed for *DeliverPacket*, if not, we end up with extremely adversarial behaviors from the path that just force *TCP* into a retransmission loop.
3. Weak fairness must hold for *DecreaseWindow*. This is a consequence of how I wrote this code. Without this, *TCP* can once again just stand there and refuse to send anything once a timeout happenes. This prevents that.

$$\begin{aligned} Spec \triangleq\ & Init \land \Box[Next]_{vars} \\ & \land \mathrm{WF}_{vars}(SendNewPacket) \\ & \land \mathrm{SF}_{vars}(DeliverPacket) \\ & \land \mathrm{WF}_{vars}(DecreaseWindow) \end{aligned}$$

The least that *TCP* should do is to actually manage to send things!

$$\begin{aligned} FinishedSending \triangleq\ & \land nPacket = 0 \\ & \land inFlight = 0 \end{aligned}$$

$$Liveness \triangleq \Diamond(FinishedSending)$$

\ * Modification History
\ * Last modified *Thu Oct* 20 16:43:31 *IRST* 2022 by *Arvin*
\ * Created *Thu Sep* 22 01:24:28 *IRST* 2022 by *Arvin*