```
┌──────────────────── MODULE FIFOBroadcast ────────────────────┐

  This module extends the Best-Effort Broadcast spec to also implement FIFO
  ordered delivery.
  The Spec takes some inspiration from the Broadcast spec as provided in the
  DARE 2024 summer school.

EXTENDS
    Naturals,
    FiniteSets,
    Bags,
    TLAPS

CONSTANTS
    Procs,
    Messages,
    Correct

ASSUME
    ∧ Procs ≠ {}
    ∧ Messages ≠ {}
    ∧ Correct ∈ SUBSET Procs

  The message type for a broadcast message, as will be transported by the
  perfect point-to-point links.
BC_Message ≜ [sdr : Procs, msg : Messages, id : Nat]

├──────────────────────────────────────────────────────────────┤

  Let's import the perfect point-to-point links spec
  See the PerfectPointToPointLink module for more details

  > "I have observed that many new users want to write TLA+ specs so they
  > can be reused. I have one word of advice for those users: Don't."
  > https://groups.google.com/g/tlaplus/c/BHBNTkJ2QFE/m/meTQs4pHBwAJ

VARIABLES
    pl_sent,
    pl_delivered

pl_vars ≜ ⟨pl_sent, pl_delivered⟩

  Internal representation of messages that are transported by the perfect
  point-to-point links.
PL_Rich_Message ≜ [sdr : Procs, rcv : Procs, msg : BC_Message]

  This may seem a bit strange at first. However, it is fine, trust me. The
  broadcast spec needs to be able to send the same message to multiple
  receivers. This could be done by doing so in a loop, however, it is
```

unnecessary to represent a loop in TLA+, it would just lead to a redundant state explosion. Instead, have an action that can (asynchronously) send the same message to multiple receivers.

```
$pl\_bcast\_send(p, qs, m) \triangleq$
    $\land p \in Procs$
    $\land qs \subseteq Procs$
    $\land$ LET $rms \triangleq \{[sdr \mapsto p, rcv \mapsto q, msg \mapsto m] : q \in qs\}$
          IN
           $\land \forall rm \in rms : rm \notin pl\_sent$
           $\land pl\_sent' = pl\_sent \cup rms$
           $\land$ UNCHANGED $pl\_delivered$
```

* !Not used

```
$pl\_send(p, q, m) \triangleq$
    $\land p \in Procs$
    $\land q \in Procs$
    $\land$ LET $rm \triangleq [sdr \mapsto p, rcv \mapsto q, msg \mapsto m]$
       IN
        $\land rm \notin pl\_sent$
        $\land pl\_sent' = pl\_sent \cup \{rm\}$
        $\land$ UNCHANGED $pl\_delivered$

$pl\_deliver(p, q, m) \triangleq$
    $\land p \in Procs$
    $\land q \in Procs$
    $\land$ LET $rm \triangleq [sdr \mapsto p, rcv \mapsto q, msg \mapsto m]$
       IN
        $\land rm \in pl\_sent$
        $\land rm \notin pl\_delivered$
        $\land pl\_delivered' = pl\_delivered \cup \{rm\}$
        $\land$ UNCHANGED $pl\_sent$

$PL\_Init \triangleq$
    $\land pl\_sent = \{\}$
    $\land pl\_delivered = \{\}$
```

Back to the *FIFO* broadcast module

The spec consists of the following variables. The variables are not used for the core functionality of the spec; rather, they are used to keep track of state for the purpose of checking the properties.
Notably, *bc_broadcasted* and *bc_delivered* are used to keep track of which messages have broadcast and delivered.
bc_failed is used to keep track of which processes have failed.

```
 In some initial specs, we found that the time to run the model checking would
 be difficult to control. As a means to control it, we check which messages
 out of Messages have been broadcast by any process, such that each message
 is broadcast at most once. This is tracked by the variable bc_messages_used.
 Lastly, bc_state manages the internal state of the broadcast spec. It manages
 the local sequence nubers for each process, and some additional info, in
 order to implement FIFO delivery.
VARIABLES
    bc_broadcasted,   Bag of [sdr ↦ p, rcv ↦ q, msg ↦ m]
    bc_delivered,   Bag of [sdr ↦ p, rcv ↦ q, msg ↦ m]
    bc_failed,    ⊆ Procs
    bc_messages_used,    ⊆ Messages
    bc_state  [p ∈ Procs ↦  [lsn : Nat, delivered : SUBSET BC_Message]]

bc_vars ≜ ⟨bc_broadcasted, bc_delivered, bc_failed, bc_messages_used, bc_state⟩

vars ≜ ⟨pl_vars, bc_vars⟩

BC_ProcState ≜ [lsn : Nat, delivered : SUBSET BC_Message]
BC_State ≜ [p ∈ Procs ↦ BC_ProcState]

├──────────────────────────────────────────────────────────────────────┤

 broadcast message m from process p
beb_broadcast(p, m) ≜
    ∧ m ∉ bc_messages_used
    ∧ p  ∉ bc_failed
    ∧ LET qs ≜ ProcsIN
        LET bc_msg ≜ [sdr ↦ p, msg ↦ m, id ↦ bc_state[p].lsn]
            IN
            pl_bcast_send(p, qs, bc_msg)
    ∧ bc_messages_used′ = bc_messages_used ∪ {m}
    ∧ bc_state′ = [bc_state EXCEPT ![p].lsn = bc_state[p].lsn + 1]
    ∧ bc_broadcasted′ = bc_broadcasted ⊕ SetToBag({[sdr ↦ p, rcv ↦ q, msg ↦ m] : q ∈ Procs})
    ∧ UNCHANGED ⟨bc_delivered, bc_failed⟩

 deliver a broadcast message m to process p from process q
beb_deliver(p, q, m, id) ≜
     Guard against non-fifo-ordered delivery
    ∧   ∨ id = 0
        ∨ id > 0 ∧ [sdr ↦ q, id ↦ (id − 1)] ∈ {[sdr ↦ x.sdr, id ↦ x.id] : x ∈ bc_state[p].delivered}
    ∧ p ∉ bc_failed
    ∧ LET bc_msg ≜ [sdr ↦ q, msg ↦ m, id ↦ id]
            IN
             ∧ pl_deliver(q, p, bc_msg)
             ∧ bc_state′ = [bc_state EXCEPT ![p].delivered = bc_state[p].delivered ∪ {bc_msg}]
    ∧ bc_delivered′ = bc_delivered ⊕ SetToBag({[sdr ↦ q, rcv ↦ p, msg ↦ m]})
    ∧ UNCHANGED ⟨bc_broadcasted, bc_failed, bc_messages_used⟩

beb_fail(p) ≜
    ∧ p ∉ Correct
    ∧ p ∉ bc_failed
    ∧ bc_failed′ = bc_failed ∪ {p}
    ∧ UNCHANGED ⟨pl_vars, bc_broadcasted, bc_delivered, bc_messages_used, bc_state⟩

BEB_Init ≜
    ∧ bc_broadcasted = EmptyBag
    ∧ bc_delivered = EmptyBag
    ∧ bc_failed = {}
    ∧ bc_messages_used = {}
    ∧ bc_state = [p ∈ Procs ↦ [lsn ↦ 0, delivered ↦ {}]]

Init ≜
    ∧ PL_Init
    ∧ BEB_Init

Next ≜ ∃ p ∈ Procs, q ∈ Procs, m ∈ Messages, id ∈ 0 .. (Cardinality(Messages) − 1) :
    ∨ beb_broadcast(p, m)
    ∨ beb_deliver(p, q, m, id)
    ∨ beb_fail(p)

Spec ≜
    ∧ Init
    ∧ □[Next]_vars
    ∧ WF_vars(Next)

├──────────────────────────────────────────────────────────────────┤

 Let's check some properties with TLC

TypeInv ≜
    ∧ pl_sent ⊆ PL_Rich_Message
    ∧ pl_delivered ⊆ PL_Rich_Message
    ∧ bc_failed ⊆ Procs
    ∧ bc_messages_used ⊆ Messages
    ∧ bc_state ∈ [Procs → BC_ProcState]

 BEB1: Validity: If a correct process broadcasts a message m, then every correct
 process eventually delivers m.
Prop_BEB1_Validity ≜
    □∀ p  ∈ Procs, q ∈ Procs, m ∈ Messages :
        (p ∈ Correct ∧ q ∈ Correct) ⇒
            (([sdr ↦ p, rcv ↦ q, msg ↦ m] ∈ DOMAIN bc_broadcasted) ⇒
                (◇([sdr ↦ p, rcv ↦ q, msg ↦ m] ∈ DOMAIN bc_delivered)))
 BEB2: No duplication: No message is delivered more than once.
Prop_BEB2_NoDuplication ≜
    □∀ m ∈ BagToSet(bc_delivered) :
        (IF BagIn(m, bc_delivered) THEN bc_delivered[m] ELSE 0) ≤ 1
         (CopiesIn(m, bc_delivered) ≤ 1) \* This doesn't work on the Toolbox, but works in VS Code

 BEB3: No creation: If a process delivers a message m with sender s, then m was
 previously broadcast by process s.
Prop_BEB3_NoCreation ≜ □(BagToSet(bc_delivered) ⊆ BagToSet(bc_broadcasted))

 FIFO delivery: If some process broadcasts message m1 before it broadcasts
 message m2, then no process delivers m2 unless it has already delivered
 m1.
PROP_FIFODelivery ≜
    □∀ p ∈ Procs : ∀ m ∈ bc_state[p].delivered :
        ∨ m.id = 0
        ∨ ∃ mp ∈ bc_state[p].delivered :
            mp.sdr = m.sdr ∧ (mp.id + 1 = m.id)

├──────────────────────────────────────────────────────────────────────┤

 Let's do one proof in TLAPS

THEOREM Spec ⇒ Prop_BEB3_NoCreation
PROOF
⟨1⟩ DEFINE NoCreation ≜ BagToSet(bc_delivered) ⊆ BagToSet(bc_broadcasted)
⟨1⟩1 Init ⇒ NoCreation
    PROOF BY DEF Init, NoCreation, BEB_Init
⟨1⟩2 Next ∧ NoCreation ⇒ NoCreation′
    ⟨2⟩ SUFFICES ASSUME Next, NoCreation
        PROVE NoCreation′
        PROOF BY DEF Next, NoCreation
    ⟨2⟩ PICK p ∈ Procs, q ∈ Procs, m ∈ Messages, id ∈ 0 . . (Cardinality(Messages) − 1) :
                ∨ beb_broadcast(p, m)
                ∨ beb_deliver(p, q, m, id)
                ∨ beb_fail(p)
        PROOF BY DEF Next
    ⟨2⟩1 beb_broadcast(p, m) ⇒ NoCreation′
        ⟨3⟩ SUFFICES ASSUME beb_broadcast(p, m) ∧ NoCreationPROVE NoCreation′
            PROOF OBVIOUS
        ⟨3⟩1 BagToSet(bc_delivered) ⊆ BagToSet(bc_broadcasted)
            PROOF BY DEF NoCreation
        ⟨3⟩2 bc_delivered = bc_delivered′
            PROOF BY beb_broadcast(p, m) DEF beb_broadcast
        ⟨3⟩3 bc_broadcasted′ = bc_broadcasted ⊕ SetToBag({[sdr ↦ p, rcv ↦ q_1, msg ↦ m] : q_1 ∈ Procs})
            PROOF BY DEF beb_broadcast
        ⟨3⟩4 $BagToSet(bc\_broadcasted) \subseteq BagToSet(bc\_broadcasted')$
            PROOF OMITTED  Follows obviously from ⟨3⟩3, but TLAPS seems to struggle with reasoning about Bags
        ⟨3⟩5 $NoCreation'$
            PROOF BY ⟨3⟩1, ⟨3⟩2, ⟨3⟩3, ⟨3⟩4
        ⟨3⟩ QED
            PROOF BY ⟨3⟩5
    ⟨2⟩2 $beb\_deliver(p, q, m, id) \Rightarrow NoCreation'$
        PROOF OMITTED  Proof omitted for time reasons
    ⟨2⟩3 $beb\_fail(p) \Rightarrow NoCreation'$
        PROOF BY  DEF beb_fail
    ⟨2⟩ QED
        PROOF BY ⟨2⟩1, ⟨2⟩2, ⟨2⟩3
⟨1⟩3 UNCHANGED $vars \land NoCreation \Rightarrow NoCreation'$
    PROOF BY  DEF vars, NoCreation, bc_vars
⟨1⟩4 $[Next]_{vars} \land NoCreation \Rightarrow NoCreation'$
    PROOF BY ⟨1⟩2, ⟨1⟩3, PTL
⟨1⟩5 $Spec \Rightarrow \Box NoCreation$
    BY ⟨1⟩1, ⟨1⟩4, PTL DEF Spec
⟨1⟩6 QED
    PROOF BY ⟨1⟩5  DEF Prop_BEB3_NoCreation, NoCreation
```

* Modification History
* Last modified *Thu Oct* 10 14:59:12 *CEST* 2024 by *jonasspenger*
* Created *Wed Oct* 09 12:56:24 *CEST* 2024 by *jonasspenger*