Channel Class Pseudocode for Beacon Network Simulator

Beacon Network Simulator

July 15, 2025

1 Channel Class

Algorithm 1 Channel Initialization

procedure INITIALIZE(*metrics*)
    active_transmissions $\leftarrow \emptyset$ ▷ List of (beacon, start_time, end_time)
    metrics $\leftarrow$ *metrics*
    buoys $\leftarrow \emptyset$
    seen_attempts $\leftarrow \emptyset$ ▷ Set of (receiver_id, sender_id, timestamp)
end procedure
procedure SETBUOYS(*buoys*)
    this.buoys $\leftarrow$ *buoys*
end procedure

Algorithm 2 Channel Update

procedure UPDATE(*sim_time*)
    active_transmissions $\leftarrow \{(b, start, end) \in active\_transmissions \mid end > sim\_time\}$ ▷ Remove expired transmissions
end procedure

Algorithm 3 Channel Broadcast

procedure BROADCAST($beacon, sim\_time$)
  if $metrics \neq null$ **then**
    $metrics.log\_sent()$
  end if
  $transmission\_time \leftarrow beacon.size\_bits()/BIT\_RATE$
  $new\_end\_time \leftarrow sim\_time + transmission\_time$
  for all $(existing, start, end) \in active\_transmissions$ **do**
    if $beacon.sender\_id = existing.sender\_id$ **then**
      continue ▷ Skip checking same sender
    end if
    $time\_overlap \leftarrow (sim\_time \leq end) \wedge (start \leq new\_end\_time)$ ▷ Check temporal overlap
    if $time\_overlap \wedge \text{InRange}(beacon.position, existing.position)$ **then**
      log_error("Collision detected")
      if $metrics \neq null$ **then**
        $metrics.log\_collision()$
      end if
      return $false$ ▷ Collision occurred
    end if
  end for
  $active\_transmissions.append((beacon, sim\_time, new\_end\_time))$
  log_info("Broadcasting beacon")
  $receivers\_in\_range \leftarrow \{buoy \in buoys \mid buoy.id \neq beacon.sender\_id \wedge \text{InRange}(beacon.position, buoy.position)\}$
  $n\_receivers \leftarrow |receivers\_in\_range|$
  if $metrics \neq null$ **then**
    $metrics.log\_potentially\_sent(beacon.sender\_id, n\_receivers)$
  end if
  return $true$
end procedure

Algorithm 4 Channel Is Busy

procedure ISBUSY($position, sim\_time$)
  for all $(beacon, start, end) \in active\_transmissions$ **do**
    if $start \leq sim\_time \leq end \wedge \text{InRange}(position, beacon.position)$ **then**
      return $true$
    end if
  end for
  return $false$
end procedure

Algorithm 5 Channel Receive All

```
1: procedure ReceiveAll(receiver_id, receiver_position, sim_time)
2:     received ← ∅
3:     for all (beacon, start, end) ∈ active_transmissions do
4:         if beacon.sender_id = receiver_id then
5:             continue                                          ▷ Skip beacons from self
6:         end if
7:         key ← (receiver_id, beacon.sender_id, beacon.timestamp)
8:         if key ∈ seen_attempts then
9:             continue                                  ▷ Already processed this beacon
10:        end if
11:        distance ← EuclideanDistance(receiver_position, beacon.position)
12:        if IDEAL_CHANNEL then
13:            if distance > COMMUNICATION_RANGE_HIGH_PROB then
14:                continue                                              ▷ Out of range
15:            end if
16:            propagation_delay ← distance/SPEED_OF_LIGHT
17:            arrival_time ← start + propagation_delay
18:            if sim_time < arrival_time then
19:                continue                                          ▷ Not yet arrived
20:            end if
21:            seen_attempts.add(key)
22:            received.append(beacon)
23:        else
24:            if distance > COMMUNICATION_RANGE_MAX then
25:                continue                                          ▷ Out of range
26:            end if
27:            propagation_delay ← distance/SPEED_OF_LIGHT
28:            arrival_time ← start + propagation_delay
29:            if sim_time < arrival_time then
30:                continue                                          ▷ Not yet arrived
31:            end if
32:            seen_attempts.add(key)
33:            if distance ≤ COMMUNICATION_RANGE_HIGH_PROB then
34:                if random() < DELIVERY_PROB_HIGH then
35:                    received.append(beacon)
36:                else if metrics ≠ null then
37:                    metrics.log_lost()
38:                end if
39:            else if distance ≤ COMMUNICATION_RANGE_MAX then
40:                if random() < DELIVERY_PROB_LOW then
41:                    received.append(beacon)
42:                else
43:                    log_error("Packet lost")
44:                    if metrics ≠ null then
45:                        metrics.log_lost()
46:                    end if
47:                end if
48:            end if
49:        end if
50:    end for
51:    if |received| ≤ 1 then
52:        if metrics ≠ null ∧ |received| = 1 then
53:            metrics.log_received(received[0].sender_id, received[0].timestamp, sim_time, receiver_id)
54:            metrics.log_actually_received(received[0].sender_id)
55:        end if
56:        return received
57:    else
58:        log_error("Collision at receiver")
59:        if metrics ≠ null then
60:            for all beacon ∈ received do
61:                metrics.log_collision()
62:            end for
63:        end if
```

Algorithm 6 Channel In Range

procedure InRange($pos1, pos2$)
    $dx \leftarrow pos1.x - pos2.x$
    $dy \leftarrow pos1.y - pos2.y$
    $distance \leftarrow \sqrt{dx^2 + dy^2}$
    if $IDEAL\_CHANNEL$ **then**
        return $distance \leq COMMUNICATION\_RANGE\_HIGH\_PROB$
    else
        return $distance \leq COMMUNICATION\_RANGE\_MAX$
    end if
end procedure