Specyfikacje:

- **Best-effort Broadcast (BEB)**
  - BEB1: Ważność (Jeżeli procesy Pi i Pj są poprawne, to każda wiadomość m rozgłaszana przez Pi jest ostatecznie dostarczona do Pj)
  - BEB2: Brak powielenia (m jest dostarczona co najwyżej raz)
  - BEB3: Brak samogeneracji (m nie jest dostarczona, jeśli nie była wysłana)

- **Reliable Broadcast (RB)**
  - Własności BEB
  - RB4: Zgodność (jeśli wiadomość m została odebrana przez pewien poprawny proces, to ostatecznie wszystkie poprawne procesy odbiorą m)
  -

- **Uniform Reliable Broadcast (URB)**
  - Własności BEB
  - URB4: Jednolita zgodność (jeśli wiadomość m została odebrana przez pewien proces (poprawny lub nie), to ostatecznie wszystkie poprawne procesy odbiorą m)

- **Reliable FIFO Broadcast (RFB)**
  - RB1, RB2, RB3, RB4
  - FIFO Order: Jeżeli proces Pi rozgłosił wiadomość m1 przed m2, to każdy proces Pj nie odbierze m2 przed m1

- **Causal Order**
  - m1 $\rightarrow$ m2, gdy:
  - obie wiadomości zostały rozgłoszone przez ten sam proces, m1 przed m2
  - m1 została odebrana przez pewien proces Pi, a m2 została rozgłoszona przez Pi po odebraniu m1
  - Istnieje wiadomość m3 taka, że dla m1 i m3, oraz m3 i m2 zachodzi a) lub b)

- **Reliable Causal Broadcast (RCB)**
  - RB1, RB2, RB3, RB4
  - Causal order: Proces Pi nie odbierze wiadomości m2, dopóki nie odebrał wszystkich wiadomości m1, takich, że m1 $\rightarrow$ m2

- **Total Order Broadcast (TO)**
  - RB1, RB2, RB3, RB4
  - Globalne uporządkowanie wiadomości (total order):
    Wszystkie procesy odbierają wiadomości w tym samym porządku, tj. jeśli Pi i Pj są poprawnymi procesami, które odbierają wiadomość m i jeśli Pi odbiera m' przed m, to także Pj odbiera m' przed m.

- **Uniform Total Order Broadcast (UTO)**
  - URB1, URB2, URB3, URB4
  - Jednolite globalne uporządkowanie wiadomości (uniform total order):
    jeśli Pi i Pj odbierają wiadomość m i jeśli Pi odbiera m' przed m, to także Pj odbiera m' przed m.

Alg. 1. Lazy reliable broadcast (uses BEB and perfect failure detektor)

```
1   upon event ⟨Init⟩ do
2     delivered := {}
3     correct := {p_1, ..., p_n}
4     forall p_i ∈ {p_1, ..., p_n} do
5       from[pi] := {}
6   upon event ⟨rbBroadcast, m⟩ do
7     trigger ⟨bebBroadcast, [Data, self, m]⟩
8   upon event ⟨crash, p_i⟩ do
9     correct := correct \ {p_i}
10    forall [p_j, m] ∈ from[p_i] do
11      trigger ⟨bebBroadcast, [Data, p_j, m]⟩
12  upon event ⟨bebDeliver, p_i, [Data, p_j, m]⟩ do
13    if m ∉ delivered then
14      delivered := delivered ∪ {m}
15      trigger ⟨rbDeliver, p_j, m⟩
16      if p_i ∉ correct then
17        trigger ⟨bebBroadcast, [Data, p_j, m]⟩
18      else
19        from[p_i] := from[p_i] ∪ {[p_j, m]}
```

Alg. 2. Uniform reliable broadcast

```
1   upon event ⟨Init⟩ do
2     correct := {p_1, ..., p_n}
3     delivered := {}
4     pending := {}
5     for all messages m : ack[m] := {}
6   upon event ⟨crash, p_i⟩ do
7     correct := correct \ {p_i}
8   upon event ⟨urbBroadcast, m⟩ do
9     pending := pending ∪ {[self, m]}
10    trigger ⟨bebBroadcast, [Data, self, m]⟩
11  upon event ⟨bebDeliver, p_i, [Data, p_j, m]⟩ do
12    ack[m] := ack[m] ∪ {p_i}
13    if [p_j, m] ∉ pending then
14      pending := pending ∪ {[p_j, m]}
15      trigger ⟨bebBroadcast, [Data, p_j, m]⟩
16  upon event ⟨for any [p_j, m] in pending: correct ⊆ ack[m]⟩ and ⟨m ∉ delivered⟩ do
17    delivered := delivered ∪ {m}
18    trigger ⟨urbDeliver, p_j, m⟩
```

Alg. 3. xyzBroadcast using FIFOBroadcast

```
1   upon event ⟨Init⟩ do
2     delivered := {}
3     rcntDlvrs := ⟨⟩        // empty sequence
4   upon event ⟨xyzBroadcast, m⟩ do
5     trigger ⟨rfbBroadcast, rcntDlvrs · m⟩        // concatenation operation
6     rcntDlvrs := ⟨⟩
7   upon event ⟨rfbDeliver, p_i, ⟨m_1, m_2, ..., m_l⟩ for some l⟩ do
8     for i := 1, ..., l do
9       if m_i ∉ delivered then
10        trigger ⟨xyzDeliver, p_i, m_i⟩
11        delivered := delivered ∪ {m_i}
12        rcntDlvrs := rcntDlvrs · m_i        // concatenation operation
```

Alg. 4. Reliable Causal Broadcast (uses RB)

- **upon event** < Init > **do**
  - delivered := { }
  - past := { }

- **upon event** < rcoBroadcast, m> **do**
  - **trigger** < rbBroadcast, [Data,past,m]>;
  - past := past U { [self,m] };

- **upon event** <rbDeliver,pi,[Data,$past_m$,m]> **do**
  - **if** m not in delivered **then**
    - **forall** [sn, m'] in $past_m$ **do** // in causal order
      - **if** m' not in delivered **then**
        - **trigger** < rcoDeliver, sn, m'>;
        - delivered := delivered U { m' };
        - past := past U { [sn, m'] };
    - **trigger** <rcoDeliver,pi,m>;
    - delivered := delivered U { m };
    - past := past U { [pi,m] };

Alg. 5. Reliable Causal Broadcast with garbage collection (uses RB i perfect failure detector)

```
1   upon event ⟨Init⟩ do
2     delivered := {}
3     past := {}
4     correct := {p_1, ..., p_n}
5     forall m : ack[m] := {}
6   upon event ⟨crash, p_i⟩ do
7     correct := correct \ {p_i}
8   upon event ⟨rcoBroadcast, m⟩ do
9     trigger ⟨rbBroadcast, [Data, past, m]⟩
10    past := past ∪ {[self, m]}
11  upon event ⟨rbDeliver, p_i, [Data, past_m, m]⟩ do
12    if m ∉ delivered then
13      forall [sn, m] ∈ past_m do // in causal order
14        if m ∉ delivered then
15          trigger ⟨rcoDeliver, sn, m⟩
16          delivered := delivered ∪ {m}
17          past := past ∪ {[sn, m]}
18      trigger ⟨rcoDeliver, p_i, m⟩
19      delivered := delivered ∪ {m}
20      past := past ∪ {[p_i, m]}
21      forall m ∈ delivered : self ∉ ack[m] do
22        ack[m] := ack[m] ∪ {self}
23        trigger ⟨rbBroadcast, [ACK, m]⟩
24  upon event ⟨rbDeliver, p_i, [ACK, m]⟩ do
25    ack[m] := ack[m] ∪ {p_i}
26    if correct ⊆ ack[m] do
27      past := past \ {[sm, m]}
```