**Interface:**

**Requests:**

**subscribe (**topic**)**

**unsubscribe (**topic**)**

**publish** **(**topic, m**)**

**Indications:**

**psDelivery** **(**topic, m**)**

**State:**

diameter // diameter of the overlay

radiusSubs // includes own subscriptions Map[<process>] = {(<topic>, <TTL>)}

subHops // max hops of gossip when renewing subscriptions

pubHops // max hops of gossip when publishing a message of a given topic

neighbors // partial view of the overlay

delivered // set of Ids of messages

pending // messages to be forwarded

**Upon init () do:**

diameter ← #π \* 10

radiusSubs ← {}

subHops ← diameter / 2

pubHops ← diameter / 2

neighbors ← ⊥

delivered ← {}

**Setup Periodic Timer RenewSub (T)**

**Upon subscribe (**topic**) do:**

radiusSubs[this] ← radiusSubs[this] U {(topic, TTL)}

mid ← generateUID({this, topic, getTimeOfSystem() })

delivered ← delivered U {mid}

pending ← pending U {(**SUB**, this, topic, TTL, subHops - 1, mid)}

**Trigger** **GetNeighbors ()**

**Upon unsubscribe (**topic**) do:**

radiusSubs[this] ← radiusSubs[this] \ {(topic, -)}

mid ← generateUID({**UNSUB**, this, topic, getTimeOfSystem()})

delivered ← delivered U {mid}

pending ← pending U {(**UNSUB**, this, topic, subHops - 1, mid)}

**Trigger GetNeighbors ()**

**Upon publish (**topic, m**) do:**

mid ← generateUID({**PUB**, this, topic, m})

delivered ← delivered U {mid}

pending ← pending U {(**PUB**, this, topic, pubHops, m, mid)}

**Trigger GetNeighbors()**

**Upon RenewSub () do:**

**For each** p ∈ radiusSubs **do:**

**if** {topic, ttl} ∈ radiusSubs[this] ∧ ttl=0 **then**

**Trigger subscribe (**topic**)**

**Upon Neighbors (**N**) do:**

neighbors ← N

//mandar pending todo?

**For each** (**SUB**, s, topic, TTL, subHops, mid) ∈ pending **do**:

**Trigger Gossip** **(SUB**, this, topic, TTL, subHops, mid**)**

**For each (UNSUB**, s, topic, subHops, mid**)** ∈ pending **do**:

**Trigger Gossip(UNSUB**, s, topic, subHops, mid**)**

**For each (PUB**, s, topic, m**)** ∈ pending **do**:

**Trigger Gossip(PUB**, s, topic, pubHops, m, mid**)**

pending ← {}

**Upon Receive (SUB**, s, topic, TTL, subHops, mid**) do:**

**if** mid ∉ delivered **then**

radiusSubs[s] ← radiusSubs[s] U {(topic, TTL)}

delivered ← delivered U {mid}

**if** subHops > 0 **then**

pending ← pending U {(**SUB**, s, topic, TTL, subHops - 1, mid)}

**Trigger GetNeighbors()**

**Upon Receive (UNSUB**, s, topic, subHops, mid**) do:**

**if** mid ∉ delivered **then**

radiusSubs[s] ← radiusSubs[s] \ {(topic, -)}

delivered ← delivered U {mid}

**if** subHops > 0 **then**

pending ← pending U {(**UNSUB**, s, topic, TTL, subHops - 1, mid)}

**Trigger GetNeighbors()**

**Upon Receive (PUB**, s, topic, pubHops, m, mid**) do:**

**if** mid ∉ delivered **then**

delivered ← delivered U {mid}

**if** {this, TTL} ∈ radiusSubs[topic] ∧ TTL > 0 **then**

**Trigger psDelivery(**topic, m**)**

**For Each** {p, TTL} ∈ radiusSubs[topic] **do:**

**if** TTL > 0 **then**

**Trigger Send (**topic, m, mid**)**

**if** pubHops > 0 **then**

pending ← pending U {(**PUB**, s, topic, pubHops - 1, m)}

**Trigger GetNeighbors()**

**Upon Receive (DIRECTMSG**, topic, m, mid**) do:**

if mid ∉ delivered **then**

delivered ← delivered U {mid}

**if** {topic, TTL} ∈ radiusSubs[this] ∧ TTL > 0 **then**

**Trigger psDelivery(**topic, m**)**

//é necessário otimizar o Hyparview?

// assumir que o hyparview fornece uma procedimento para gossip

//timeout para apagar radiusSubs, delivered

//hopsPub=hopsSub= d/2 ?

//como e que se escreve o factor cagança?

//ttl

//map com chave processo e outro com topic

//3 pendings por terem atributos diferentes

//ttl ← data e não temporizador, por constante no state?

//grande probabilidade