WELLINGTON GABRIEL VICENTE DE SOUZA

SMARTGOSSIP: UM ALGORITMO INTELIGENTE PARA DIFUSÃO PROBABILÍSTICA DE MENSAGENS EM REDES DE TOPOLOGIA ARBITRÁRIA

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Departamento de Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Elias Procópio Duarte Júnior.

CURITIBA PR

2021

RESUMO

Considere um sistema distribuído que é uma coleção de processos que se comunicam utilizando trocas de mensagens para a realização de tarefas específicas. A difusão de uma mensagem para todos os processos é necessária para a execução de diversos tipos de tarefas. Esse trabalho apresenta o algoritmo SmartGossip, para a difusão probabilística e inteligente de mensagens em redes de topologia arbitrária, inspirada pela estratégia de otimização conhecida como colônia de formigas, mais especificamente pelo conceito de comunicação por estigmergia. O algoritmo SmartGossip foi implementado utilizando o simulador OMNeT++. Foram executados experimentos medindo o número de mensagens utilizadas e o número de rodadas necessárias para completar a difusão em sistemas com tamanho de processos variando de 64 a 1024 processos. Para cada tamanho de sistema, foram geradas topologias correspondentes a grafos aleatórios com conectividades variando de 0.5 a 1.0, caso em que o grafo é completo. Para propósitos de comparação de desempenho, também foram implementadas simulações dos algoritmos baseados em inundação (Flooding) e em difusão probabilística (Gossip), duas das estratégias mais comuns na difusão de mensagens em sistemas distribuídos. Os resultados mostram que o número médio de mensagens utilizadas pelo SmartGossip é sempre menor que dos demais algoritmos, além de apresentarem baixa dispersão na medida em que o tamanho do sistema e conectividade variam.

Palavras-chave: Sistemas Distribuídos. Difusão Probabilística de Mensagens. Inteligência de Enxames.

SUMÁRIO

1 INTRODUÇÃO

Um sistema distribuído é um conjunto de processos que se comunicam e colaboram para realizar uma tarefa. Muitas destas tarefas necessitam que um processo transmita uma mensagem para todos os processos do sistema. Algoritmos de difusão (*broadcast*) permitem justamente a comunicação de um-para-todos, sendo imprescindíveis para diversos sistemas e aplicações distribuídas, inclusive da Internet (Cachin et al., 2011; Mullender, 1993). Neste contexto, especialmente para sistemas distribuídos de grande escala, é desejável que o tempo para completar a transmissão de mensagens (*i.e.* a latência) seja baixa e que o número de mensagens enviadas não seja excessivo.

A primitiva clássica de sistemas distribuídos para realizar a difusão de mensagens de forma confiável é o *reliable broadcast* (difusão confiável). Dentre as propriedades que a difusão deve garantir, a principal é o acordo, segundo o qual se uma mensagem é entregue por um processo correto, então todos os processos corretos devem entregar a mensagem. Além disso, nenhum processo entrega uma determinada mensagem mais de uma vez, não há mensagens espúrias e a difusão termina depois de um intervalo finito de tempo. Em particular, mesmo que a origem da mensagem falhe, se pelo menos um processo correto entrega a mensagem, então todos os processos devem entregar a mensagem.

Uma forma simples e direta para garantir a difusão confiável é apresentada pelo algoritmo de inundação (Flooding) (Tanenbaum e Wetherall, 2011). O algoritmo consiste em garantir que cada processo do sistema transmite a mensagem para todos seus vizinhos conhecidos ao recebê-la, garantindo que todo processo não-falho do sistema distribuído receberá a mensagem. Como ponto negativo, o algoritmo utiliza um grande número de mensagens redundantes, já que um processo não tem conhecimento do recebimento da mensagem pelos outros processos do sistema.

Para melhorar o desempenho da difusão, algoritmos de difusão probabilística surgiram como uma solução visando reduzir o número de mensagens. Apesar de manterem uma boa probabilidade de que todos os processos entreguem uma mensagem m, algoritmos probabilísticos não garantem que 100% dos processos sempre entregam m. O algoritmo de difusão probabilística mais conhecido é o Gossip (Eugster et al., 2004). Neste algoritmo, ao receber uma mensagem m, um processo transmite a mensagem apenas para um subconjunto aleatório de sua vizinhança, de tamanho definido por um parâmetro conhecido como *fanout*. Esse procedimento se repete por um número predeterminado de rodadas. Tanto o número de mensagens transmitidas como a probabilidade de que todos os processos entreguem a mensagem m são afetados diretamente pelo parâmetro *fanout* e pelo número máximo de rodadas.

Nesse trabalho, um algoritmo inteligente para a disseminação probabilística de mensagens é proposto, denominado SmartGossip. O algoritmo implementado foi inspirado pela

estratégia de otimização conhecida como colônia de formigas (Dorigo et al., 2006), utilizando o conceito de comunicação por estigmergia (Dorigo et al., 2000). No contexto de sistemas distribuídos, aplicações desta estratégia foram anteriormente propostas para a descoberta de topologia de redes dinâmicas (Nassu et al., 2007) e a disseminação de eventos de uma rede (Banzi et al., 2011).

A estratégia utilizada pelo algoritmo SmartGossip consiste em utilizar "depósitos de feromônios" nos enlaces de um sistema distribuído, armazenados localmente por cada processo, como um critério de decisão para os próximos destinos de uma mensagem. Quando um processo recebe uma mensagem por meio de um enlace, uma quantidade fixa de feromônios é depositada naquele enlace. A escolha do próximo destino da mensagem é probabilística, sendo influenciada pela concentração de feromônios em cada enlace do processo para sua vizinhança. Enlaces com uma concentração menor de feromônios possuem maior probabilidade de serem escolhidos, fazendo com que a transmissão da mensagem sempre priorize caminhos ainda não percorridos, aumentando a chance de que processos que ainda não entregaram a mensagem a recebam.

O algoritmo SmartGossip foi implementado utilizando o simulador OMNeT++(OMNeT++, 2021). Foram executados experimentos medindo o número de mensagens utilizadas e o número de rodadas necessárias para completar a difusão em sistemas com N variando de 64, 128, 256, 512 e 1024 processos. Para cada tamanho de sistema, foram geradas topologias correspondentes a grafos randômicos com conectividades variando de 0.5, 0.6, 0.7, 0.8, 0.9 a 1,0, caso em que o grafo é completo. Foram também implementados para comparação os algoritmos clássicos de difusão de mensagens Flooding e Gossip. Os resultados mostram que o número médio de mensagens utilizadas pelo SmartGossip é sempre menor que dos demais algoritmos, além de apresentarem baixa dispersão na medida em que o tamanho do sistema e conectividade variam.

O restante do trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os dois algoritmos distribuídos clássicos para difusão de mensagens: Flooding e Gossip. Estes algoritmos são utilizados como base de comparação com o algoritmo SmartGossip. O Capítulo 3 apresenta brevemente o trabalho para descoberta de topologia de redes dinâmicas na qual foi originalmente proposta a estratégia inteligente, descrevendo em seguida o algoritmo SmartGossip para difusão probabilística inteligente de mensagens. O Capítulo 4 descreve o simulador implementado com o framework OMNeT++ e apresenta os resultados obtidos nas comparações entre o Flooding, Gossip e SmartGossip. As conclusões seguem no Capítulo 5.

2 ALGORITMOS DISTRIBUÍDOS PARA DIFUSÃO DE MENSAGENS

A difusão de mensagens (*broadcast*) em um sistema distribuído permite que um processo envie uma mensagem para todos os processos do sistema. Este capítulo proporciona uma visão geral dos dois algoritmos distribuídos mais comuns para difusão de mensagens. No contexto desse trabalho, estes algoritmos servirão como base de comparação para a estratégia inteligente simulada no trabalho. O primeiro algoritmo apresentado nesta seção é conhecido como algoritmo de inundação, ou Flooding (Tanenbaum e Wetherall, 2011). A segunda estratégia apresentada é a difusão probabilística (Eugster et al., 2004), mais comumente conhecida como Gossip. Para cada uma das estratégias, o funcionamento geral é brevemente descrito e suas vantagens e desvantagens são discutidas.

2.1 MODELO DE SISTEMA

O sistema distribuído aqui considerado possui topologia arbitrária (*multi-hop*), podendo ser definido como um grafo $G = (V, E)$. Cada vértice $v \in V$ representa um processo, e cada aresta $e \in E$ representa um enlace. Sabendo que $i \in V$ e $j \in V$ são dois processos distintos, e que $e_{ij} \in E$ é um enlace que os conecta, assumimos também para o sistema definido aqui que:

1. Os enlaces são perfeitos, garantindo que uma dada mensagem não será entregue mais que uma vez e que as mensagens serão recebidas em algum momento. Adicionalmente, não são direcionados, tal que $e_{ij} = e_{ji}$.

2. Um processo $i \in V$ tem conhecimento de sua vizinhança $N_i \subseteq V$. A vizinhança N_i consiste dos processos j tal que $e_{ij} \in E$, isto é, os processos adjacentes a i em G.

3. O modelo temporal é o parcialmente síncrono GST (*Global Stabilization Time*) (Larrea et al., 2004). Nesse modelo, o sistema começa instável e, após um certo instante de tempo desconhecido, passa a se comportar como síncrono para sempre.

4. O modelo de falhas utilizado é por parada (*crash*). Um processo falho interrompe sua execução completamente, não produzindo nenhuma saída e perdendo o estado interno.

   Além disso, são definidas as seguintes primitivas:

- ***send(dst, src, msg):*** utilizada por um processo *src* para transmitir uma mensagem *msg* para o processo *dst*.

- ***broadcast(msg):*** dispara a difusão da mensagem *msg* para todos os processos do sistema.

- ***receive(src, dst, msg):*** utilizada por um processo *dst* para receber uma mensagem *msg* vinda do processo *src*.

- ***deliver(msg):*** utilizada por um processo para entregar uma mensagem *msg*.

Observação importante: em alguns dos algoritmos abaixo, as primitivas são utilizadas com diferentes parâmetros, definidos caso a caso.

2.2 O ALGORITMO DE INUNDAÇÃO (FLOODING)

O algoritmo de inundação, conhecido também como Flooding, baseia-se na estratégia de "inundar" um sistema com mensagens para garantir que todos os processos deste sistema recebam uma determinada mensagem com (Tanenbaum e Wetherall, 2011). Considerando que o sistema tem topologia arbitrária (*multi-hop*) e que um único processo não conhece o estado completo da rede, um processo envia suas mensagens apenas para o subconjunto de processos conhecidos.

Seja $N_i \subseteq V$ a vizinhança de um processo i. No algoritmo baseado na estratégia de Flooding, o processo i executando a primitiva *broadcast(msg)* transmitirá a mensagem a todo processo $j \in N_i$. Por sua vez, ao receber uma nova mensagem, um processo $j \in N_i$ entrega a mensagem caso seja a primeira vez que a está recebendo. Além disso, o processo j retransmite a mensagem a todo processo $k \in N_j - \{i\}$. O procedimento se repete até que todos os processos do sistema tenham recebido a mensagem. O pseudocódigo do algoritmo é visto no Algoritmo 1.

Algoritmo 1 Difusão de Mensagens por Inundação (executado pelo processo i)

```
1:  Init:
2:      delivered ← ∅

3:  Upon broadcast(msg):
4:      for all j ∈ N_i do
5:          send(j, i, msg)
6:      end for

7:  Upon receive(src, i, msg):
8:      if msg ∉ delivered then
9:          delivered ← delivered ∪ {msg}
10:         deliver(msg)
11:         for all j ∈ N_i − {src} do
12:             send(j, i, msg)
13:         end for
14:     end if
```

Todo processo i do sistema distribuído executa o algoritmo Algoritmo 1. Ao iniciar a execução, o processo i seta o conjunto de mensagens entregues *delivered* como vazio. O conjunto *delivered* é utilizado para armazenar as mensagens que já foram entregues. Ao executar a primitiva *broadcast(msg)*, o processo i transmite a mensagem *msg* para todo processo $j \in N_i$, transmitindo a mensagem que se assume incluir o identificador do processo de origem. Ao executar a primitiva *receive(src, dst, msg)*, o processo i verifica se a mensagem já foi entregue. Em caso positivo, a mensagem *msg* é descartada. Em caso negativo, a mensagem é armazenada no conjunto *delivered* e é entregue pelo processo e, em seguida, é retransmitida para todo processo na vizinhança de i exceto a origem da mensagem, ou seja, para todo processo $j \in N_i - \{src\}$.

O algoritmo de inundação não é um algoritmo eficiente em relação ao número de mensagens utilizadas. Como todos processos enviam cada mensagem para todos seus vizinhos, em geral um grande número de mensagens repetidas serão transmitidas. No pior caso, o número de mensagens transmitidas é $|E|$. Entretanto, se um caminho existe entre dois processos, o algoritmo de inundação irá garantir que uma mensagem *msg* enviada utilizando *broadcast(msg)* será sempre entregue.

2.3 O ALGORITMO DE DIFUSÃO PROBABILÍSTICA (GOSSIP)

O algoritmo de difusão probabilística (Eugster et al., 2004), conhecido também como Gossip, se propõe a solucionar o problema de desperdício de mensagens presente no algoritmo de inundação, limitando o número de mensagens enviadas. O Gossip tem sua estratégia se baseando na forma como epidemias se alastram, com cada processo que recebe a mensagem transmitindo esta para um outro grupo aleatório de processos vizinhos, até que todos os processos do sistema recebam a mensagem.

Para definir a taxa na qual as mensagens são disseminadas de forma epidêmica, o algoritmo baseado na estratégia Gossip possui um parâmetro conhecido como *fanout*. Seja f o valor do *fanout* para uma determinada execução de *broadcast(msg)* e seja N_i um subconjunto de V dado como a vizinhança de um processo i, o processo i a executar a primitiva enviará sua mensagem para um subconjunto aleatório $S \subseteq N_i$ de no máximo f processos em sua vizinhança, tal que $|S| = min\{f, |N_i|\}$. Ao receber uma nova mensagem, um processo j a entregará para outros $min\{f, |N_j - \{i\}|\}$ vizinhos, até que todos os processos do sistema tenham recebido a mensagem disseminada ou até que o número máximo de rodadas - definido pelo parâmetro *maxRounds* - seja atingido. O pseudocódigo do Gossip pode ser visto no Algoritmo 2.

Nesse algoritmo, as primitivas são utilizadas com os parâmetros descritos a seguir:

- ***send(dst, src, msg, round):*** utilizada por um processo *src* para transmitir uma mensagem *msg* para o processo *dst* na rodada *round*.
- ***receive(src, dst, msg, round):*** utilizada por um processo *dst* para receber uma mensagem *msg* vinda do processo *src* na rodada *round*.

Algoritmo 2 Difusão Probabilística de Mensagens (executado pelo processo i)

```
1: Init:
2:     delivered ← ∅

3: Upon broadcast(msg):
4:     S ← random subset of f processes of N_i
5:     for all j ∈ S do
6:         send(j, i, msg, maxRounds − 1)
7:     end for

8: Upon receive(src, i, msg, round):
9:     if msg ∉ delivered then
10:        delivered ← delivered ∪ {msg}
11:        deliver(msg)
12:    end if
13:    if round > 0 then
14:        S ← random subset of f processes of N_i
15:        for all j ∈ S do
16:            send(j, i, msg, round − 1)
17:        end for
18:    end if
```

Todo processo i do sistema distribuído executa o Algoritmo 2. Assim como no Algoritmo 1, ao iniciar a execução, o processo i inicializa o conjunto de mensagens entregues *delivered*. Ao executar a primitiva *broadcast(msg)*, o processo i escolhe um subconjunto aleatório S de $min\{f, |N_i|\}$ processos de sua vizinhança N_i e transmite a mensagem msg para todo $j \in S$, incluindo a mensagem, o identificador de origem e o identificador da rodada. A rodada inicial é dada por $maxRounds - 1$, onde $maxRounds > 0$ é um parâmetro definido antes da execução do algoritmo para limitar o número de mensagens enviadas. Ao executar a primitiva *receive(src, dst, msg, round)*, o processo i verifica se a mensagem recebida de *src* já foi entregue e a armazena caso ainda não tenha sido. Em seguida, se o número máximo de rodadas ainda não foi atingido, o processo i continua a execução, escolhendo um subconjunto S de $min\{f, |N_i - \{src\}|\}$ processos de $N_i - \{src\}$ e retransmitindo a mensagem msg para todo $j \in S$.

O número de mensagens enviadas é diretamente influenciado pelo parâmetro f (*fanout*). Quando temos $f = |V| - 1$, o número de mensagens enviadas será igual ao do algoritmo de inundação apresentado na seção anterior. Ao escolhermos um número menor para o parâmetro f, os processos escolhidos para o conjunto S serão aleatórios, e portanto, para garantir que a mensagem será entregue para todos os processos, um valor adequado de número máximo de rodadas (*maxRounds*) é necessário. É importante notar que devido ao conjunto aleatório de processos a receberem a mensagem, o algoritmo é probabilístico.

3 UMA ESTRATÉGIA INTELIGENTE PARA DIFUSÃO PROBABILÍSTICA

A estratégia inteligente aqui apresentada para a difusão probabilística de mensagens é inspirada em algoritmos de inteligência de enxame, mais especificamente de colônias de formigas (Dorigo et al., 2006). A abordagem do algoritmo simulado baseia-se no conceito de estigmergia (Dorigo et al., 2000) como estratégia de comunicação para otimizar o desempenho da difusão de informações entre nodos de um sistema distribuído. Tal método foi originalmente proposto no contexto de descoberta de topologias de redes (Nassu et al., 2007). Adicionalmente, a mesma estratégia foi utilizada para a implementação de um algoritmo de disseminação de eventos em uma rede (Banzi et al., 2011). Este capítulo inicia descrevendo a estratégia original para descoberta de topologia de redes dinâmicas e, em seguida, descreve o algoritmo distribuído inteligente de difusão probabilística. Observação sobre nomenclatura: na Seção 3.1, as unidades do sistema são *nodos* de uma rede dinâmica; na Seção 3.2, são *processos* de um sistema distribuído.

3.1 A ESTRATÉGIA INTELIGENTE INSPIRADA EM COLÔNIA DE FORMIGAS

A estratégia inteligente inspirada por colônias de formigas (Dorigo et al., 2000, 2006) aqui apresentada foi originalmente proposta no contexto de descoberta de topologia de redes dinâmicas (Nassu et al., 2007). Em redes de topologia dinâmica, nodos podem deixar o sistema ou juntar-se a ele a qualquer momento. Considere que cada nodo mantém uma representação da topologia da rede armazenada localmente. Dada a topologia dinâmica, em um dado momento, o conhecimento de um nodo sobre a topologia atual da rede pode não mais refletir o estado atual da mesma.

O funcionamento básico do algoritmo de descoberta de topologia de redes dinâmicas baseada em inteligência de enxame consiste na utilização de "agentes inteligentes" (ou formigas) que percorrem o sistema constantemente, propagando as informações mais recentes descobertas por cada nodo sobre a topologia atual da rede. Ao visitar um nodo, o agente atualiza este com suas informações mais recentes sobre o estado da topologia da rede. Em seguida, o agente atualiza seu conhecimento sobre a topologia com dados do nodo atual e migra para outro nodo do sistema, repetindo o processo. O modelo do sistema é apresentado a seguir, com a estratégia para seleção de um próximo destino durante a migração dos agentes sendo apresentada à frente.

No modelo de sistema assumido, a topologia é definida como um grafo $G(t) = (V_t, E_t)$ no qual a cada instante de tempo t, um vértice $v_{it} \in V_t$ representa um nodo i, e cada aresta $e_{ijt} \in E_t$ representa um enlace entre os nodos i e j. A topologia é definida como uma função do tempo e, portanto, dados os instantes de tempo t_1 e t_2 tal que $t_1 \neq t_2$, $G(t_1)$ possivelmente é distinto de $G(t_2)$. Também assume-se que:

- A todo instante, um nodo conhece sua vizinhança.

- Um agente pode migrar entre dois nodos em um tempo não nulo mas limitado, com o limite sendo conhecido.

- Os enlaces não são direcionados.

A estratégia para seleção de um próximo destino consiste em um método baseado em estigmergia, com os agentes escolhendo o próximo nodo com base no nível de "feromônios" depositados em cada enlace. Inicialmente, a concentração de feromônios de cada enlace $e_{ijt} \in E_t$ é nula. Ao migrar de um nodo i para um nodo j, um agente atualiza em i a concentração de feromônios $C_{ij}(t)$ associada ao enlace e_{ijt}, como demonstrado pela Equação 3.1.

$$C_{ij}(t) \leftarrow C_{ij}(t) + 1 \tag{3.1}$$

Os feromônios depositados também "evaporam" com o passar do tempo, definido por uma taxa de evaporação $0 \leq \rho < 1$ para cada instante de tempo. A concentração de feromônios em um instante t é dada em função da concentração em um tempo anterior $t_0 < t$, conforme a Equação 3.2.

$$C_{ij}(t) = C_{ij}(t_0) \cdot (1 - \rho)^{t - t_0} \tag{3.2}$$

O próximo destino de um agente é escolhido probabilisticamente, com a concentração de cada depósito de feromônios influenciando a escolha de um enlace. Seja $\alpha > 0$ a intensidade dos feromônios depositados, $|E_{it}|$ o número de vizinhos de i no instante t, e $e_{it}(k)$ o k-ésimo vizinho de i no instante t, a probabilidade $P_{ij}(t)$ do enlace e_{ijt} ser escolhido é dado pela Equação 3.3.

$$P_{ij}(t) = \frac{(C_{ij}(t) + 1)^{-\alpha}}{\sum_{k=1}^{|E_{it}|} (C_{ie_{it}(k)}(t) + 1)^{-\alpha}} \tag{3.3}$$

Para garantir que o número de agentes se adapte ao tamanho do sistema dinâmico conforme este se altera, também faz-se necessário o controle populacional de agentes. As concentrações de feromônio também são utilizadas para este propósito, com um limite inferior $L_{min}(t)$ e um limite superior $L_{max}(t)$, respectivamente, levando à criação ou destruição de um agente quando ultrapassados. Cada nodo usa sua concentração local total de feromônios, consistindo na soma dos depósitos em cada um de seus enlaces, como dado pela Equação 3.4.

$$C_i(t) = \sum_{k=1}^{|E_{it}|} C_{ie_{it}(k)}(t) \tag{3.4}$$

Como um número maior de agentes tenderá a passar por processos com maior número de enlaces, os limites inferior $L_{min}(t)$ e superior $L_{max}(t)$ variam por processo. Dadas as constantes de peso da vizinhança $\delta \geq 0$, limite inferior mínimo $\gamma_{min} > 0$ e limite superior máximo $\gamma_{max} > \gamma_{min}$, a concentração $C_i(t)$ de feromônios é comparada com os valores de limite ajustados

para o processo i com base no tamanho de sua vizinhança no instante de tempo t, definidos respectivamente pelas Equações 3.5 e 3.6.

$$L_{min}(t) = \gamma_{min} \cdot |E_{it}|^{\delta} \tag{3.5}$$

$$L_{max}(t) = \gamma_{max} \cdot |E_{it}|^{\delta} \tag{3.6}$$

3.2 O ALGORITMO INTELIGENTE DE DIFUSÃO PROBABILÍSTICA

A abordagem baseada em feromônios utilizada na estratégia apresentada na seção anterior foi adaptada para o caso de uso de difusão probabilística de mensagens em um sistema distribuído. A decisão do caminho a ser percorrido na difusão de mensagens também é baseada em estigmergia. No contexto de difusão de mensagens, o objetivo ao adotar tal estratégia é melhorar o desempenho da difusão, tanto do número de mensagens transmitidas como da latência da difusão.

Para decidir o próximo destinatário da mensagem, o processo i calcula a probabilidade $P_{ij}(t)$ para cada enlace e_{ij} conectando i a todo vizinho $j \in N_i$, conforme descrito na Equação 3.3. Assim, os destinos são escolhidos levando em consideração as probabilidades calculadas. É importante observar que no contexto do algoritmo de difusão probabilística inteligente, não utilizamos o conceito de agentes. O processo de decisão de próximo destino que era executado pelo agente é agora executado pelo processo i com base nos feromônios e probabilidades calculadas segundo as equações apresentadas. Conforme as mensagens são enviadas, o depósito de feromônios para cada enlace é atualizado, aplicando também a evaporação de feromônios.

No algoritmo inteligente de difusão probabilística, o controle populacional utilizado para controlar o número de agentes na estratégia apresentada anteriormente é adaptado para o controle do número de mensagens redundantes sendo transmitidas. Apesar disso, o limite inferior L_{min} não é utilizado para decidir quando criar novas mensagens. Diferentemente da estratégia original, este algoritmo utiliza um parâmetro de *fanout*, assim como na estratégia Gossip, criando novas mensagens a cada rodada. O controle populacional do limite superior L_{max} é mantido, com as mensagens sendo destruídas quando necessário para controlar o número de mensagens redundantes. O pseudocódigo pode ser visto no Algoritmo 3. As primitivas modificadas do Algoritmo 2 também são aplicadas aqui.

Algoritmo 3 Algoritmo Inteligente de Difusão Probabilística (executado pelo processo i)

1: ***Init:***
2: $delivered \leftarrow \emptyset$
3: $N_i \leftarrow$ set of neighbors
4: **for all** $j \in N_i$ **do**
5: $pheromones[j] \leftarrow 0$
6: **end for**

7: ***Upon broadcast(msg):***
8: $S \leftarrow$ random subset of f processes of N_i
9: **for all** $j \in S$ **do**
10: $send(j, i, msg, maxRounds - 1)$
11: $pheromones[j] \leftarrow pheromones[j] + 1$
12: **end for**

13: ***Upon receive(src, i, msg, round):***
14: **for all** $j \in N_i$ **do**
15: $pheromones[j] \leftarrow pheromones[j] \cdot (1 - \rho)^{|time\ elapsed\ since\ last\ update|}$
16: **end for**
17: $pheromones[src] \leftarrow pheromones[src] + 1$
18: **if** $msg \notin delivered$ **then**
19: $delivered \leftarrow delivered \cup \{msg\}$
20: *deliver(msg)*
21: **end if**
22: $C_i(t) \leftarrow \sum_{k=1}^{|N_i|} C_{ie_i(k)}(t)$
23: **if** $round > 0$ **and** $C_i(t) < L_{max}$ **then**
24: $S \leftarrow$ random subset of f processes of N_i
25: **for all** $j \in S$ **do**
26: $send(j, i, msg, round - 1)$
27: $pheromones[j] \leftarrow pheromones[j] + 1$
28: **end for**
29: **end if**

Todo processo i do sistema distribuído executa o Algoritmo 3. Ao iniciar a execução, o processo i inicializa o conjunto de mensagens entregues (*delivered*). Em seguida, inicializa o conjunto de depósitos de feromônio (*pheromones*) de cada enlace existente entre i e j, $\forall j \in N_i$. Ao executar a primitiva *broadcast(msg)*, o processo i escolhe um subconjunto aleatório S de $min\{f, |N_i|\}$ processos de sua vizinhança N_i e transmite a mensagem msg para todo $j \in S$, incluindo a mensagem, o identificador de origem, o identificador de destino e o identificador da rodada. A rodada inicial aqui também é dada por $maxRounds - 1$, onde $maxRounds > 0$ é um parâmetro definido antes da execução do algoritmo para limitar o número de mensagens

enviadas. Após transmitir as mensagens, o processo i incrementa seu contador de feromônios local para cada um dos enlaces sobre o qual a mensagem foi transmitida.

Ao executar a primitiva *receive(src, dst, msg, round)*, o processo i atualiza o contador dos feromônios, aplicando a evaporação conforme descrita na Equação 3.2. Em seguida, atualiza também o valor de feromônio para o enlace utilizado pela mensagem recebida. O processo então verifica se a mensagem recebida de *src* já foi entregue e a armazena caso ainda não tenha sido. Para decidir se a mensagem recebida será retransmitida, o processo i verifica se o número máximo de rodadas ainda não foi atingido e se o valor da concentração total de feromônios $C_i(t)$ (Equação 3.4) não ultrapassou o limite máximo L_{max}. Caso o valor de $C_i(t)$ esteja acima do limite superior, a mensagem é destruída para garantir o controle populacional. Em caso negativo, a execução prossegue, com mais $min\{f, |N_i|\}$ processos sendo escolhidos como destino da mensagem a ser retransmitida. Após retransmitir a mensagem, o processo i incrementa o seu contador de feromônios local para cada um dos enlaces sobre o qual a mensagem foi retransmitida.

Ambas as métricas de latência e de número de mensagens enviadas são influenciadas diretamente por diversos dos parâmetros, e portanto estes devem ser ajustados apropriadamente. Primeiramente, os valores das constantes γ_{min} e γ_{max} utilizados pelas Equações 3.5 e 3.6 influenciam no controle populacional do número de mensagens enviadas por cada processo. Caso o valor de γ_{min} seja muito pequeno, o número de mensagens pode se manter estagnado, influenciando diretamente na latência. Similarmente, caso o valor de γ_{max} seja muito pequeno, as mensagens serão destruídas com maior frequência, reduzindo o número de mensagens transmitidas, também gerando impacto na latência. Inversamente, se γ_{min} e γ_{max} são valores muito elevados, o sistema pode ser inundado com mensagens desnecessárias. A constante δ também influencia diretamente no controle populacional, visto que δ é dado como o peso da vizinhança de i no cálculo dos limites locais L_{min} e L_{max}, utilizados para controle populacional. O valor da taxa de evaporação ρ também influencia no desempenho do algoritmo, tanto no controle populacional - com a concentração de feromônios ultrapassando os limites mais frequentemente - como na escolha dos próximos destinos, afetando o número de mensagens enviadas e a latência.

O algoritmo é validado e avaliado experimentalmente no próximo capítulo.

4 SIMULAÇÃO E RESULTADOS

O desempenho do algoritmo proposto foi avaliado através da execução de um simulador desenvolvido para este trabalho. Além do algoritmo SmartGossip, também foram simulados os algoritmos Flooding e Gossip, que servem como base de comparação para os resultados obtidos pelo algoritmo proposto. Este capítulo inicia descrevendo brevemente o simulador implementado e a máquina na qual as simulações foram executadas e, em seguida, apresenta os experimentos realizados e os resultados obtidos.

4.1 O SIMULADOR IMPLEMENTADO

A comparação do desempenho dos algoritmos apresentados neste trabalho foi realizada utilizando um simulador implementado através do framework de simulação OMNeT++ (OMNeT++, 2021), na linguagem C++. Foram implementadas simulações dos algoritmos clássicos de difusão de mensagens, Flooding e Gossip, e do algoritmo de difusão probabilística inteligente aqui proposto, o SmartGossip.

Inicialmente, o simulador implementado gera uma topologia aleatória de com $N > 0$ vértices, que representam os processos do sistema distribuído. O número de enlaces é influenciado por um parâmetro de conectividade $0 < C \leq 1$. O parâmetro C reflete a conectividade da topologia gerada da seguinte maneira. Quando, por exemplo, $C = 1.0$, a conectividade é 100%, ou seja há uma aresta conectando cada um dos processos a todos os demais. De forma análoga, quando $C = 0.5$ a probabilidade de haver uma aresta entre dois processos quaisquer é de 50%. Em outras palavras: C reflete a probabilidade de dois processos quaisquer serem adjacentes. A geração das topologias aleatórias parametrizadas pela conectividade é implementada da seguinte maneira pelo OMNeT++: para cada processo $i < N - 2$, o OMNeT++ gera um número aleatório entre 0 e 1 para cada processo j tal que $i < j < N - 1$ para decidir se há uma aresta entre os processos i e j. Se o resultado estiver entre 0 e C é adicionada a aresta entre i e j. O resultado desta etapa pode gerar topologias que consistem de mais de um componente conexo, assim foi implementado um passo adicional para garantir que a topologia final consiste de um único componente conexo. A geração da topologia conecta cada processo i a um processo aleatório j, tal que $j > i$. Conforme mostrado no Teorema 1 fica garantido não apenas que todo processo pertence ao mesmo componente conexo.

Teorema 1. *A estratégia proposta para criação de topologias arbitrárias produz um grafo $G = (V, E)$ que consiste de um único componente conexo.*

Demonstração. A prova é feita por indução no número de vértices $|V|$.

Base: considere um grafo G que consiste de apenas 2 vértices, i e j, seja $i < j$. Na estratégia proposta i é conectado por uma aresta a um vértice $j > i$. Como neste grafo há um único vértice que atende à condição, os dois vértices são conectados por uma aresta.

Hipótese da indução: considere que para um grafo gerado com a estratégia proposta e com número de vértices $|V| = M$ o grafo gerado consiste de um único componente conexo.

Passo da indução: considere agora a criação de um topologia com $|V| = M + 1$ vértices, que é feita pela adição de um novo vértice i ao grafo gerado no passo da indução com M vértices, de forma que $\forall j \in V - i : j > i$. A estratégia proposta garante que vai haver pelo menos uma aresta do vértice i para um dos demais vértices em $V - i$, portanto o grafo gerado consiste de um único componente conexo. □

A máquina utilizada para a execução das simulações e obtenção dos resultados apresentados na Seção 4.2 é baseada em um processador Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz com 4 cores, tendo 16GB RAM e 2TB de memória secundária SSD.

4.2 RESULTADOS

As simulações foram executadas para cinco diferentes valores de N: 64, 128, 256, 512 e 1024. Para cada valor de N, foram geradas topologias de seis níveis de conectividade diferentes, $C = 0.5, 0.6, ..., 1$. Cada combinação de configurações de número de processos e nível de conectividade foi executada 30 vezes para obter os resultados aqui apresentados. Todos os algoritmos (Flooding, Gossip e SmartGossip) foram executados por tantas rodadas quantas foram necessárias para que todos os processos entreguem a mensagem.

O algoritmo Gossip foi simulado utilizando *fanout* = $log_{10}(N)$, valor que tem sido tradicionalmente apontado na literatura como adequado (Eugster et al., 2004). No trabalho também foi realizada uma avaliação experimental em que foram utilizados diversos outros valores/funções para o *fanout* e $log_{10}(N)$ efetivamente se mostrou o mais apropriado.

Os valores para os parâmetros do algoritmo SmartGossip foram definidos experimentalmente, a partir de diversas execuções das simulações, para obtenção dos valores mais adequados. Em particular, foi utilizada a potência dos feromônios (α) = 8, taxa de evaporação (ρ) = 0.1, peso da vizinhança (δ) = 0.5 e o mesmo *fanout* do Gossip, isto é *fanout* = $log_{10}(N)$. Valores diferentes para o parâmetro de limite superior γ_{max} foram encontrados para cada tamanho de sistema N. Para N = 64 e 128, $\gamma_{max} = 1.3$; para N = 256, $\gamma_{max} = 0.9$, para N = 512, $\gamma_{max} = 0.8$ e, finalmente, para N = 1024, $\gamma_{max} = 0.5$.

As métricas utilizadas foram o número de mensagens utilizadas e o número de rodadas executadas até a difusão acabar (latência). Os resultados são apresentados a seguir.

A Tabela 4.1-A mostra os resultados obtidos para a execução do algoritmo de Flooding quando o sistema consiste de 64 processos, com conectividade C variando de 0.5 a 1.0, caso em que a topologia corresponde a um grafo completo. Para $C = 0.5$ é possível observar que a média obtida nas 30 execuções foi de 1277.80 mensagens trocadas entre os processos, sendo no melhor

Flooding N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	1277.80	1027	1511	141.32	3.37	3	4	0.49
0.6	1577.47	1361	1757	123.49	2.40	2	3	0.50
0.7	2005.60	1881	2095	53.94	2.17	2	3	0.38
0.8	2400.93	2109	2615	149.65	2.03	2	3	0.18
0.9	2934.53	2651	3305	200.44	2.03	2	3	0.18
1.0	3969.00	3969	3969	0.00	1.00	1	1	0.00

Gossip N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	458.80	254	510	104.15	7.80	7	8	0.41
0.6	492.93	254	1022	133.33	7.90	7	9	0.40
0.7	424.67	254	510	122.74	7.67	7	8	0.48
0.8	424.67	254	510	122.74	7.67	7	8	0.48
0.9	450.27	254	510	110.13	7.77	7	8	0.43
1.0	450.27	254	510	110.13	7.60	7	8	0.50

SmartGossip N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	310.20	228	376	51.93	7.77	7	9	0.57
0.6	304.47	222	416	67.52	7.60	7	11	0.81
0.7	313.93	234	426	74.78	7.57	7	9	0.63
0.8	344.53	234	430	72.61	7.67	7	8	0.48
0.9	360.47	236	482	83.83	7.73	7	9	0.58
1.0	323.60	234	502	90.92	7.47	7	9	0.57

Tabela 4.1: Número de testes e rodadas para os três algoritmos com N=64.

caso 1027 mensagens e no pior caso 1511 mensagens. Na medida em que a conectividade C aumenta, podemos observar que o número de mensagens também aumenta. Assim, para $C = 0.6$ a média do número de mensagens cresce para 1577.47; para $C = 0.7$ são 2005.6 mensagens; $C = 0.8$ são 2400.93 mensagens; $C = 0.9$ são 2934 mensagens, finalmente para $C = 1$ o número de mensagens chega ao pico de 3969 – caso em que todos os processos comunicam com todos os demais, provocando a geração do maior número possível de mensagens redundantes pelo Flooding. O desvio padrão calculado para $C = 0.5$ foi de 141.32 mensagens, depois ora aumenta ora diminui, até chegar a zero quando $C = 1$, conforme esperado, pois a mesma topologia (grafo completo) é gerada em todos os casos. O número de rodadas foi diretamente influenciado pela conectividade. Para $C = 0.5$, a média foi de 3.37 rodadas, mas conforme a conectividade aumenta, este valor diminui, até que para $C = 1.0$ apenas uma rodada única é necessária, visto que todos processos conseguem comunicar-se com os demais.

A Tabela 4.1-B mostra os resultados obtidos para a execução do algoritmo de Gossip para um sistema também de 64 processos e com conectividade C variando de 0.5 a 1.0. Para $C = 0.5$, a média do número de mensagens foi de 458.80, valor que aumenta para 492.93 ($C = 0.6$) e então cai para 424.67 tanto para $C = 0.7$ como $C = 0.8$. Depois subindo um pouco para 450.27 tanto para $C = 0.9$ como $C = 1.0$. Destaca-se que para valores maiores de N não houve resultados idênticos para conectividades diferentes, como ocorreu neste caso. O valor maior para $C = 0.6$ pode ser interpretado como uma decorrência das topologias geradas: no pior caso necessitando de 1022 mensagens, o maior valor para todas as conectividades deste experimento. O número de rodadas se manteve praticamente constante entre 7 e 8 na média, com o pior caso acontecendo justamente para a instância destacada de $C = 0.6$.

Chegando ao SmartGossip, a Tabela 4.1-C mostra que em todos os casos os números de mensagens foram menores que os correspondentes para o Gossip, que por sua vez, já foram menores que os valores obtidos para o Flooding. Para $C = 0.5$, a redução do número de mensagens do SmartGossip para o Gossip é de 32.4%, enquanto que a redução do SmartGossip para o Flooding é de 75.7%. Para $C = 1.0$, a redução foi de 28.1% e 91.8%, respectivamente. Por outro lado, os números de rodadas do SmartGossip são semelhantes aos do Gossip, e maiores que os do Flooding, que completa a difusão em 1 única rodada quando $C = 1.0$.

A Tabela 4.2-A mostra os resultados obtidos para o Flooding com $N = 128$. O tamanho do sistema dobra em comparação com o dos resultados apresentados anteriormente. Em termos do número de mensagens, quando aumenta a conectividade C, aumenta o número de mensagens, com média igual a 5041.07 para $C = 0.5$, subindo para 16129 mensagens quando $C = 1.0$. O número de rodadas se mantém muito reduzido, de 1 até 3, no máximo, considerando todos os valores da conectividade.

A Tabela 4.2-B mostra os resultados obtidos para o Gossip com $N = 128$. Curiosamente, para $C = 0.5$ o número de mensagens na média (5832.90) foi maior que o correspondente do Flooding (5041.07). Por outro lado, para todos os outros valores da conectividade C o Gossip precisou de menos mensagens, com uma redução de 16129 (Flooding) para 970.8 (Gossip)

Flooding N=128								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	5041.07	4575	5485	268.09	2.60	2	3	0.50
0.6	6289.80	5909	6653	216.24	2.23	2	3	0.43
0.7	7944.87	7715	8121	96.06	2.07	2	3	0.25
0.8	9738.53	9179	10227	280.84	2.00	2	2	0.00
0.9	12143.07	11467	12795	406.26	2.00	2	2	0.00
1.0	16129.00	16129	16129	0.00	1.00	1	1	0.00

Gossip N=128								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	5832.90	1022	16382	4193.72	7.50	7	9	0.57
0.6	5595.87	2046	16382	3178.82	7.53	7	9	0.57
0.7	5595.63	2046	16382	4624.59	7.47	6	9	0.86
0.8	4332.90	1022	16382	3837.79	7.10	6	9	0.88
0.9	2626.27	1022	8190	2400.49	6.60	6	8	0.62
1.0	970.80	510	1022	156.23	6.00	6	6	0.00

SmartGossip N=128								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	817.20	484	904	129.33	8.87	8	9	0.35
0.6	836.40	492	928	151.65	8.83	8	9	0.38
0.7	872.13	500	958	147.19	8.87	8	9	0.35
0.8	951.40	506	1320	189.95	9.03	8	10	0.49
0.9	887.47	504	994	172.84	8.83	8	9	0.38
1.0	942.87	502	1418	168.86	8.93	8	10	0.37

Tabela 4.2: Número de testes e rodadas para os três algoritmos com N=128.

quando a conectividade é $C = 1.0$. Chama a atenção, entretando que o desvio padrão do Gossip aumenta significativamente em todos os casos. O número de rodadas necessárias para completar a difusão varia de 6 a 9, com baixa dispersão.

A Tabela 4.2-C apresenta o resultados do SmartGossip. O número de mensagens na média ficou expressivamente menor que o do Gossip para todos os valores da conectividade, com os desvios padrão obtidos bem pequenos no caso do SmartGossip. Para $C = 0.5$, a redução do número de mensagens do SmartGossip para o Gossip é de 86%, enquanto que a redução do SmartGossip para o Flooding é de 84%. Para $C = 1.0$, a redução foi de 4,7% e 94.3%, respectivamente. Por outro lado, os números de rodadas do SmartGossip (8 a 10) aumentam em comparação aos do Gossip (6 a 9), que já são maiores que os do Flooding, que completa a difusão em 1 única rodada quando $C = 1.0$ e em 3 rodadas no máximo para outros valores de conectividade.

Flooding N=256								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	20473.67	19339	21393	475.99	2.47	2	4	0.57
0.6	25767.73	24635	26565	396.94	2.13	2	3	0.35
0.7	32371.93	31463	32931	381.46	2.13	2	3	0.35
0.8	39882.07	38317	41093	641.56	2.13	2	3	0.35
0.9	49378.93	47949	50255	605.84	2.00	2	2	0.00
1.0	65025.00	65025	65025	0.00	1.00	1	1	0.00

Gossip N=256								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	25256.57	4094	65534	18571.30	8.93	8	11	0.74
0.6	22252.83	4094	65534	14892.44	8.90	8	10	0.55
0.7	21979.87	4094	65534	17226.84	8.87	8	10	0.63
0.8	15426.27	4094	65534	13507.31	8.20	7	10	0.76
0.9	8940.93	2046	32766	8845.37	7.67	7	9	0.71
1.0	2250.80	2046	4094	624.90	6.97	6	7	0.18

SmartGossip N=256								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	1836.60	1676	2218	202.83	10.40	10	13	0.77
0.6	1890.67	1754	2316	205.30	10.20	10	11	0.41
0.7	1910.47	1786	2456	204.09	10.13	10	11	0.35
0.8	1923.60	1832	2552	166.89	10.07	10	11	0.25
0.9	2061.47	1866	2696	308.41	10.20	10	11	0.41
1.0	2039.20	1022	2810	366.69	10.13	9	11	0.43

Tabela 4.3: Número de testes e rodadas para os três algoritmos com N=256.

A Tabela 4.3-A mostra os resultados obtidos para o Flooding com $N = 256$. Novamente, o tamanho do sistema é dobrado em comparação com o experimento anterior. Para esse tamanho de N, o mesmo padrão se mantém: conforme a conectividade aumenta, o número de mensagens redundantes aumenta. Para $C = 0.5$, temos que a média do número de mensagens é de 20473.67. Na maior conectividade $C = 1.0$, o número de mensagens aumenta para 65025. O número de rodadas se mantém no mesmo nível que para os outros tamanhos de N, sendo influenciado pela conectividade.

A Tabela 4.3-B mostra os resultados para o algoritmo Gossip com $N = 256$. Em média, nos níveis mais baixos de conectividade $C = 0.5$ e $C = 0.6$, o Gossip utiliza 25256.57 e 22252.83 mensagens respectivamente. Essas médias são maiores do que as obtidas pelo Flooding para os mesmos níveis de conectividade. Apesar disso, para níveis maiores de conectividade, o Gossip utiliza um número consideravelmente menor de mensagens: para $C = 1.0$, o número médio de mensagens utilizadas pelo Gossip é de 2250.80, enquanto o Flooding utiliza 65025 mensagens para o mesmo caso. É possível perceber também que o desvio padrão para o Gossip continua aumentando conforme o tamanho do sistema cresce, para todas as conectividades.

A Tabela 4.3-C apresenta os resultados para o algoritmo SmartGossip com $N = 256$. Novamente, o número de mensagens utilizadas e o desvio padrão são expressivamente inferiores aos outros dois algoritmos para todos os níveis de conectividade. Para $C = 0.5$, a redução do número de mensagens utilizadas quando comparado com o Gossip é de 92.7%, enquanto que a redução do SmartGossip para o Flooding é de 91.0%. Para $C = 1.0$, a redução foi de 9.4% e 96.8%, respectivamente. Novamente, o número de rodadas do SmartGossip (9 a 13) aumentou em comparação ao Gossip (6 a 11).

A Tabela 4.4-A mostra os resultados obtidos para o Flooding com $N = 512$. Manteve-se o mesmo padrão de aumento de número de mensagens influenciado pela conectividade. Para $C = 0.5$, temos que a média do número de mensagens é de 81449.40. Para $C = 1.0$, o número de mensagens foi 261121. O número de rodadas também se manteve no mesmo nível dos tamanhos de sistema anteriores.

A Tabela 4.4-B apresenta os resultados para o algoritmo Gossip com $N = 512$. Nos níveis mais baixos de conectividade, o Gossip continua utilizando em média um número maior de mensagens quando comparado ao Flooding. Para $C = 0.5$, em média 97207 mensagens foram utilizadas. Para $C = 1.0$, o número médio de mensagens foi 4640.13.

A Tabela 4.4-C apresenta os resultados para o algoritmo SmartGossip com $N = 512$. Para $C = 0.5$, a redução do número de mensagens utilizadas quando comparado com o Gossip é de 95.5%, enquanto que a redução do SmartGossip para o Flooding é de 94.5%. Para $C = 1.0$, a redução foi de 4.67% e 98.3%, respectivamente.

Finalmente, a Tabela 4.5 mostra os resultados para todos os algoritmos quando $N = 1024$. A Tabela 4.5-A mostra os resultados obtidos para o Flooding com $N = 1024$. Em termos do número de mensagens, quando aumenta a conectividade C, aumenta o número de mensagens, com média igual a 324801.93 para $C = 0.5$, subindo para 1046529 – mais de 1 milhão de

Flooding N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	81449.40	79593	83707	1182.74	2.63	2	3	0.49
0.6	102471.87	100447	104363	980.18	2.37	2	3	0.49
0.7	128620.47	127127	129947	919.44	2.10	2	3	0.31
0.8	157876.13	154517	160845	1659.29	2.13	2	3	0.35
0.9	196694.87	194111	199387	1659.83	2.00	2	2	0.00
1.0	261121.00	261121	261121	0.00	1.00	1	1	0.00

Gossip N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	97207	32766	262142	67472.91	10.27	9	12	0.64
0.6	112500.87	32766	1048561	182884.78	10.10	9	11	0.61
0.7	84648.67	16382	524286	96025.58	9.93	9	12	0.69
0.8	80279.60	16382	524286	98148.51	9.67	8	11	0.84
0.9	41231.07	8190	131070	32016.69	9.10	8	11	0.76
1.0	4640.13	4094	8190	1416.18	7.43	7	8	0.50

SmartGossip N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	4289	3688	5354	676.73	11.43	11	13	0.57
0.6	4198.73	3788	5468	683.67	11.23	11	12	0.43
0.7	4135.33	3848	5722	628.34	11.13	11	12	0.35
0.8	4560.40	3884	6066	959.38	11.30	11	12	0.47
0.9	4353.80	3940	6304	852.88	11.17	11	12	0.38
1.0	4423.07	3946	6950	960.56	11.20	11	13	0.48

Tabela 4.4: Número de testes e rodadas para os três algoritmos com N=512.

Flooding N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	324801.93	321103	329511	2270.88	2.73	2	3	0.45
0.6	409473.40	406649	413489	1933.79	2.27	2	3	0.45
0.7	510177.07	504443	515235	2495.76	2.10	2	3	0.31
0.8	627267.33	622269	634715	3109.50	2.07	2	3	0.25
0.9	779679.00	773191	788787	3944.21	2.00	2	2	0.00
1.0	1046529.00	1046529	1046529	0.00	1.00	1	1	0.00

Gossip N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	667243.60	88572	2391438	418509.25	9.17	8	10	0.53
0.6	708583.83	88572	7174452	1299886.98	9.10	8	10	0.61
0.7	460579.87	88572	2391483	444635.09	9.17	8	10	0.46
0.8	431055.97	88572	797160	290275.40	8.70	8	10	0.53
0.9	188955.30	88572	797160	144764.56	8.60	8	10	0.56
1.0	10496.10	9840	29523	3593.61	7.00	7	7	0.00

SmartGossip N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0.5	9160.40	8667	12438	1095.49	8.10	8	9	0.31
0.6	9208.50	8790	13209	1055.97	8.07	8	9	0.25
0.7	9938.80	9126	14556	1803.76	8.13	8	9	0.35
0.8	9835.70	9213	14790	1654.08	8.10	8	9	0.31
0.9	9787.10	9297	15474	1536.23	8.07	8	9	0.25
1.0	10208.30	9450	16425	2097.82	8.10	8	9	0.31

Tabela 4.5: Número de testes e rodadas para os três algoritmos com N=1024.

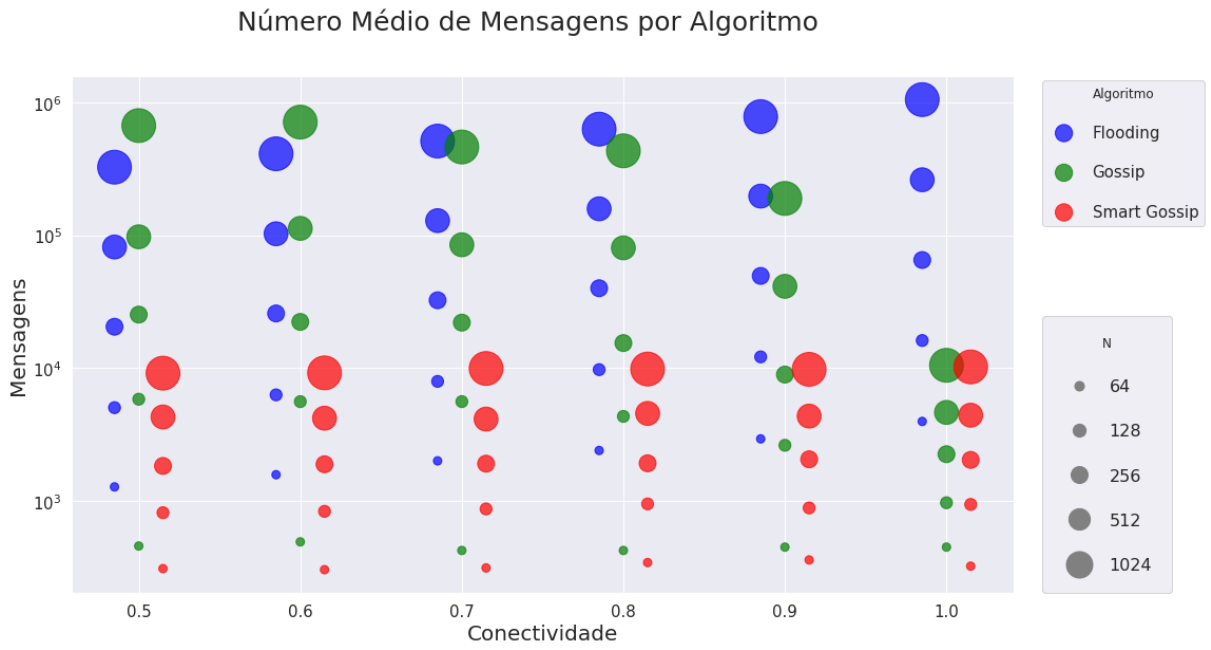

Figura 4.1: Número médio de mensagens utilizadas por cada algoritmo para cada combinação de tamanho de sistema N e conectividade C.

mensagens, portanto – quando $C = 1.0$. O número de rodadas continua muito reduzido, de 1 até 3, no máximo, considerando todos os valores da conectividade.

A Tabela 4.5-B mostra os resultados obtidos para o Gossip com $N = 1024$. Novamente, para $C = 0.5$ o número de mensagens na média (667243,6) foi maior que o correspondente do Flooding (324801,93). Para $C = 0.6$, o Gossip (708583.83) ainda precisou de mais mensagens que o Flooding (409473.40) na média. Por outro lado, para todos os outros valores da conectividade C o Gossip precisou de menos mensagens, com uma redução de 1046529 (Flooding) para 10496.1 (Gossip) quando a conectividade é $C = 1.0$. O desvio padrão do Gossip continua muito elevado neste caso. O número de rodadas necessárias para completar a difusão varia de 7 a 10, com baixa dispersão.

Finalmente, a Tabela 4.5-C apresenta o resultados do SmartGossip. O número de mensagens na média ficou expressivamente menor que o do Gossip para todos os valores da conectividade, com os desvios padrão obtidos bem pequenos no caso do SmartGossip. Para $C = 0.5$, a redução do número de mensagens do SmartGossip para o Gossip é de 98,6%, enquanto que a redução do SmartGossip para o Flooding é de 97,2%. Para $C = 1.0$, a redução foi de 2,7% e 99.2%, respectivamente. Por outro lado, os números de rodadas do SmartGossip (8 a 9) são semelhantes aos do Gossip (7 a 10), que já são maiores que os do Flooding, que completa a difusão em 1 única rodada quando $C = 1.0$ e em 2 ou 3 rodadas no máximo para outros valores de conectividade.

A Figura 4.1 apresenta uma comparação entre o número médio de mensagens utilizadas por cada algoritmo para cada tamanho de sistema e conectividade. No gráfico, cada tamanho de marcador utilizado representa um valor diferente de N, com cada cor representando um

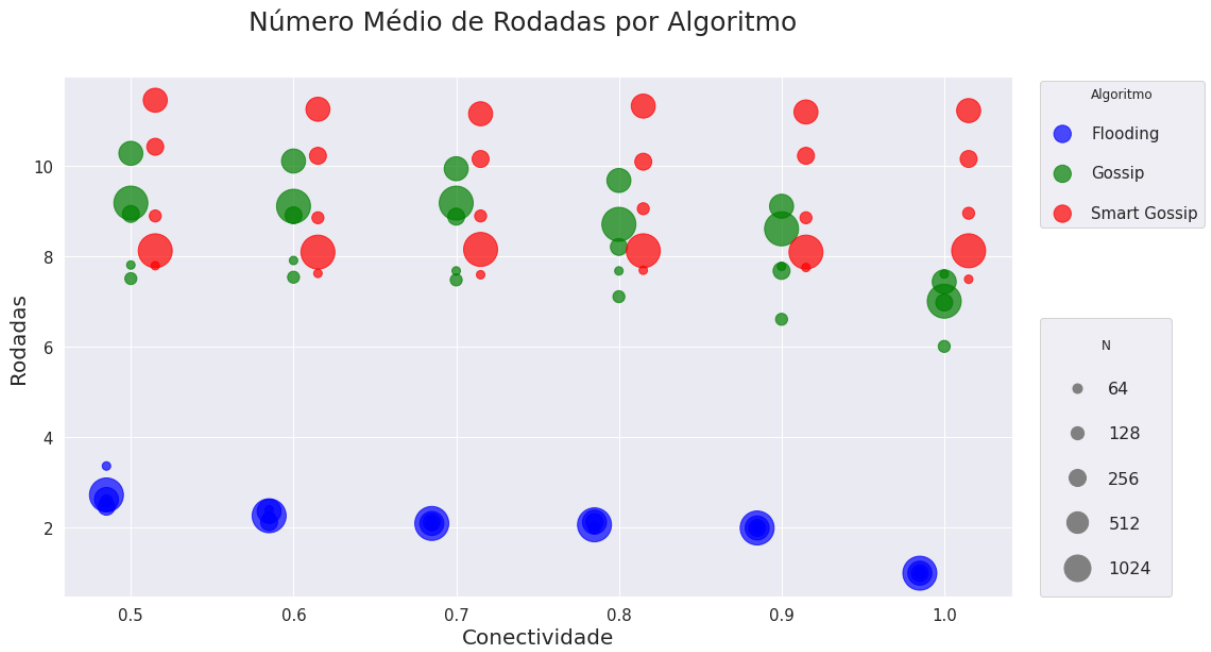

Figura 4.2: Número médio de rodadas necessárias para finalizar a difusão para cada algoritmo e para cada combinação de tamanho de sistema N e conectividade C.

algoritmo, conforme a legenda. Como é possível observar, para toda combinação de valores para N e C, o número médio de mensagens utilizadas pelo algoritmo SmartGossip foi inferior ao número médio de mensagens utilizadas pelos dois algoritmos clássicos Flooding e Gossip. Outro ponto importante a ser destacado é que o número médio de mensagens utilizadas pelo algoritmo SmartGossip para cada tamanho N se mantiveram num nível estável através das diferentes conectividades.

A Figura 4.2 apresenta a mesma comparação para o número médio de rodadas necessárias para completar a difusão. O algoritmo Flooding garante o melhor resultado no quesito de latência com sua estratégia de inundação, dado o número elevado de mensagens. Para este algoritmo na conectividade $C = 1.0$, basta apenas uma rodada. Para o algoritmo SmartGossip, é possível notar que o número médio de rodadas utilizadas é inferior aos níveis apresentados pelo algoritmo Gossip em certos casos, como para $N = 1024$ em todas conectividades $0.5 \leq C < 1.0$. Em outros casos, como para $N = 512$, o algoritmo SmartGossip necessitou de uma ou duas rodadas a mais. De modo geral, os resultados obtidos pelo SmartGossip no quesito de latência são equiparáveis aos obtidos pelo algoritmo Gossip.

4.3 DISCUSSÃO DOS RESULTADOS

Acreditamos que os resultados apresentam com clareza que o algoritmo SmartGossip representa uma contribuição, em comparação com as outras duas estratégias para difusão de mensagens: Flooding e Gossip. Para *todos* os valores de N e *todas* as conectividades C o SmartGossip resultou em número médio de mensagens utilizadas menor que o dos outros dois algoritmos. Além disso, deve ser destacado que, diferente do Gossip, o SmartGossip manteve

uma baixa dispersão em todos os casos. Por exemplo, a média do número de mensagens utilizadas pelo SmartGossip com $N = 1024$ variou de 9160.4 ($C = 0.5$) até 10208.30 ($C = 1.0$). Por outro lado, no mesmo caso, o Gossip variou de 10496.10 ($C = 1.0$) até 708583.83 ($C = 0.5$). Para todos os casos o Gossip apresentou desvio padrão do número de mensagens utilizadas muito significativo, enquanto que o SmartGossip apresentou desvio padrão pequenino.

Outro ponto a ser destacado sobre o número de mensagens utilizadas pelo SmartGossip é que o desvio padrão varia pouco entre os diferentes valores de C, com o número médio de mensagens utilizadas sofrendo pouca influência da conectividade.

No caso do Gossip, é possível inferir que a conectividade tem forte impacto no desempenho do algoritmo. Quanto menor a conectividade, maior o número de mensagens e a latência do Gossip para completar a difusão. Este fato fica em particular evidência quando a conectividade passa de 0.9 para 1.0, o que facilita para o Gossip atingir todos os processos. Destaca-se que padrões semelhantes de variação do número médio de mensagens são encontrados também para outros valores de N, com o desvio padrão aumentando conforme o tamanho do sistema aumenta.

Do outro lado, o SmartGossip necessita de 1 ou 2 rodadas a mais, em média, que o Gossip para completar a difusão. O Flooding termina a difusão com menor número de rodadas, mas empregando quantidades significativamente maiores de mensagens.

5 CONCLUSÃO

Esse trabalho propôs um algoritmo inteligente para difusão probabilística de mensagens em sistemas de topologia arbitrária denominado SmartGossip, inspirado por técnicas de colônia de formigas, mais especificamente pelo conceito de comunicação por estigmergia. A estratégia inteligente na qual o algoritmo SmartGossip é baseado foi originalmente proposta no contexto de descoberta de topologia de redes dinâmicas. Após a descrição da estratégia, o algoritmo inteligente de difusão probabilística SmartGossip foi especificado, e seu funcionamento detalhado, com destaque para a influência dos parâmetros em seu desempenho. O algoritmo foi implementado utilizando o simulador OMNeT++ e resultados de diversos experimentos realizados foram apresentados. Foram também apresentados resultados de comparação com os algoritmos distribuídos clássicos de difusão de mensagens: Flooding e Gossip. A comparação do SmartGossip com o Flooding e o Gossip utilizou como métricas o número de mensagens utilizadas e a latência (número de rodadas para completar a difusão). O algoritmo SmartGossip apresentou, em média, um número consideravelmente menor de mensagens utilizadas em todos os cenários testados quando comparado aos algoritmos clássicos, independente da conectividade do sistema, com número de rodadas equiparável ao Gossip.

Trabalhos futuros incluem executar o algoritmo sobre outros tipos de topologia (e.g. anel, hipercubo, torus, entre outras), assim como sobre topologias dinâmicas, que mudam ao longo do tempo. O objetivo é avaliar o desempenho do algoritmo em mais cenários. Adicionalmente, a validação do algoritmo em uma *testbed* sobre a Internet também está planejada como trabalho futuro. Por fim, apesar de que os parâmetros utilizados pelo SmartGossip foram determinados experimentalmente, acredita-se que é possível definir uma estratégia sistemática para geração dos melhores valores a serem utilizados em cada situação.

REFERÊNCIAS

Banzi, A. S., Jr., E. P. D. e Pozo, A. R. (2011). An approach based on swarm intelligence for event dissemination in dynamic networks. Em *Reliable Distributed Systems, IEEE Symposium on*, páginas 121–126. IEEE Computer Society.

Cachin, C., Guerraoui, R. e Rodrigues, L. (2011). *Introduction to Reliable and Secure Distributed Programming (2nd Ed)*. Springer.

Dorigo, M., Birattari, M. e Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dorigo, M., Bonabeau, E. e Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871.

Eugster, P., Guerraoui, R., Kermarrec, A.-M. e Massoulié, L. (2004). Epidemic information dissemination in distributed systems. *IEEE Computer*, 37:60–67.

Larrea, M., Fernández, A. e Arévalo, S. (2004). On the implementation of unreliable failure detectors in partially synchronous systems. *IEEE Transactions on Computers*, 53(7):815–828.

Mullender, S. (1993). *Distributed Systems (2nd Ed.)*. ACM Press/Addison-Wesley.

Nassu, B. T., Nanya, T. e Duarte, E. P. (2007). Topology discovery in dynamic and decentralized networks with mobile agents and swarm intelligence. Em *Seventh International Conference on Intelligent Systems Design and Applications (ISDA 2007)*, páginas 685–690.

OMNeT++ (2021). Omnet++: Discrete event simulator. Acessado em 10/12/2021.

Tanenbaum, A. S. e Wetherall, D. (2011). *Computer Networks (5th Ed.)*. Prentice Hall.