Algoritmo de Eleição de Líder para Redes Móveis Ad Hoc

André Morais
Email: up201307834@fe.up.pt

Pedro Ferreira
Email: up201308037@fe.up.pt

***Resumo*—Neste documento será abordado o Algoritmo de Eleição de Líder para Redes Móveis Ad Hoc proposto por Sudarshan Vasudevan, Jim Kurose e Don Towsley [1]. Este trabalho decorreu no âmbito da disciplina Sistemas Distribuídos, do Mestrado Integrado em Engenharia Eletrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto. Teve como objetivo, implementar o algoritmo e analisar os resultados das diversas simulações efetuadas. Numa primeira fase será descrito o problema bem como a solução implementada e, de seguida, serão apresentados os resultados e a análise crítica.**

***Keywords*—Redes Móveis Ad Hoc, Protocolos, Topologias de Redes de Telecomunicações, Eleição de Líder.**

I. Descrição do problema

A eleição de um líder é um problema de controlo fundamental em redes sem fios, visto que, quando um coordenador falha ou sai do sistema, é necessário eleger um novo coordenador. Existe uma vasta gama de aplicações em que a eleição de um líder está presente como, por exemplo, distribuição de chaves, coordenação de roteamento ou coordenação de sensores.

Apesar de já existirem diversos algoritmos de eleição, nenhum deles suportava as características naturais de uma rede móvel: alterações frequentes de topologia. Muitos desses algoritmos funcionavam apenas em redes estáticas ou em redes dinâmicas em que se assumia que não havia alterações de topologia durante o processo de eleição. O algoritmo que aqui será descrito suporta arbitrariedade, alterações frequentes de topologia e é capaz de eleger um único líder dentro de um tempo finito. Para além disto, o líder eleito será o nó-mais-valioso de todos os nós da rede. O valor de um nó está associado às características que um nó apresenta num determinado momento. Estas podem ser a duração restante da bateria, a distância média mínima para os outros nós ou a capacidade de computação.

II. Descrição da solução

O algoritmo de eleição de um líder começa primeiro por ”crescer” e depois ”encolher” uma *spanning tree* sendo a raíz da árvore o nó que inicia a computação. Este nó é designado por *source node*. Como será visto mais à frente, depois de a árvore se ter expandido totalmente, esta irá recolher novamente ao *source node*. Nessa altura, o nó que iniciou a eleição terá informações adequadas para determinar o nó-mais-valioso e assim, poder comunicar a identidade desse nó a todos os nós da rede.

De seguida iremos ilustrar o comportamento do algoritmo com um pequeno exemplo, considerando ainda que não há mobilidade de nós, ou seja, que estamos perante uma rede estática.

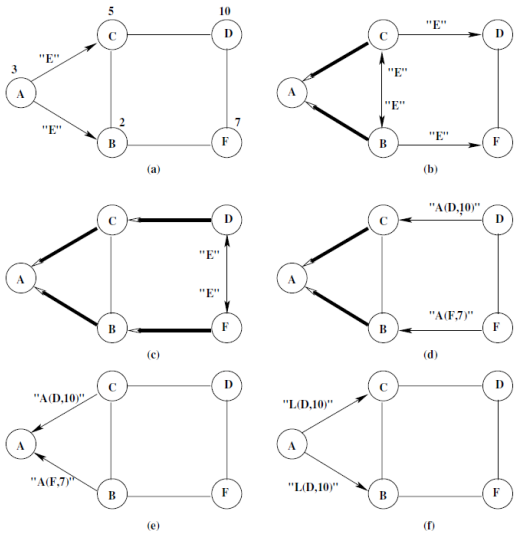

Figura 1. Exemplo de execução do algoritmo de eleição de líder. As setas finas indicam a direção do fluxo das mensagens enquanto que as setas grossas representam a construção da *spanning tree*.

Consideremos a rede da Figura 1. As setas finas indicam a direção do fluxo das mensagens enquanto que as grossas representam a construção da *spanning tree*. O nó A ao aperceber-se que perdeu a ligação ao nó líder, irá dar início à eleição, enviando uma mensagem de *Election* (denotada como ”E” na figura) para todos os seus vizinhos (Figura 1(a)). O nó A é então designado como o *source node*. Como podemos observar na Figura 1(b), os nós B e C, ao receberem a mensagem *Election*, imediatamente definem o nó A como o seu nó pai na *spanning tree* (setas grossas), e propagam a mensagem para os seus nós vizinhos, excepto para os nós pai. Devido a esta situação, os nós B e C vão trocar mensagens ”E” entre eles, mas como ambos já se encontram na mesma eleição, vão responder com uma mensagem *Ack*. Esta troca de mensagens *Ack* não está representada na figura. Na Figura 1(c) observa-se que a *spanning tree* foi completamente construída. Os nós D e F, ao receberem *Ack* de todos os vizinhos a quem enviaram uma mensagem *Election*, vão dar início ao ”encolhimento” da *spanning tree* (Figura 1(d)). Estes enviam para os seus nós pai uma mensagem *Ack* (denotada como ”A” na figura) que contém a informação do nó-mais-valioso (e o seu valor atual) que eles conhecem. Nesta caso, esse nó-mais-valioso são eles

Tabela I. TIPOS DE MENSAGENS USADOS NO ALGORITMO

Mensagem	Propósito
Election	Iniciar a eleição e formar a *spanning tree*
Ack	Responder a uma mensagem *Election*
Leader	Anunciar um novo líder
Probe	Determinar se um nó ainda está conectado
Reply	Responder a uma mensagem *Probe*
Heartbeat	Comunicar a sua presença aos vizinhos (se líder)

Tabela II. CONSTITUIÇÃO DAS MENSAGENS USADAS NO ALGORITMO

Mensagem	Formato
Election	”Election” | srcId | srcNode | lostLeader
Ack	<”Ack” | srcId | srcNode | max | flag>
Leader	<”Leader” | srcId | srcNode | leader>
Probe	<”Probe”>
Reply	<”Reply” | srcId | srcNode>
Heartbeat	<”Heartbeat”>

próprios porque são os últimos elementos da *spanning tree*. Por fim, mais cedo ou mais tarde, o nó A (*source node*) irá receber os *Ack* dos nós B e C e terá informações para poder eleger o líder. Como o nó-mais-valioso que conhece é o nó D com valor 10, irá enviar uma mensagem *Leader* com esse nó e o seu valor para os seus vizinhos, e deixa de estar em eleição. Essa mensagem será propagada ao longo da *spanning tree* até que todos os nós da rede adotem o nó D como o nó líder.

Na Tabela I estão descritas as mensagens que são utilizadas no algoritmo. As três primeiras já foram brevemente referidas no exemplo acima apresentado.

Election. As mensagens *Election* são utilizadas para ”crescer” a *spanning tree*. Quando a eleição é iniciada por um *source node* (após a partida de um nó líder), o nó começa a difusão da computação ao enviar uma mensagem *Election* para todos os seus vizinhos. Cada nó define como pai o emissor da primeira mensagem *Election* recebida e propagam a mensagem para os seus nós vizinhos, excepto para o nó pai.

Ack. Quando um nó recebe uma mensagem *Election* de um vizinho que não é o seu pai, imediatamente responde com uma mensagem *Ack*. Contudo, um nó não retorna imediatamente um *Ack* para o seu pai. Aguarda primeiro pela receção dos *Ack* de todos os seus vizinhos (que contêm informações sobre o nó-mais-valioso) para depois enviar ao seu pai um *Ack* com a informação do nó-mais-valioso que conhece.

Leader. Quando o *source node* receber os *Ack* de todos os seus vizinhos, difunde uma mensagem *Leader*, que irá chegar a todos os nós da rede, e que contém a informação do nó eleito líder.

Probe. As mensagem *Probe* são enviadas apenas durante o processo de eleição. São enviadas pelos nós para os seus vizinhos de forma a verificar se estes se mantêm conectados ou não. Quando um nó envia uma mensagem *Probe*, este aguarda pela receção de uma mensagem *Reply* proveniente do seu vizinho.

Reply. Tal como foi acabado de referir, as mensagens *Reply* são usadas para responder a uma mensagem *Probe* e comunicar a sua presença a um vizinho. A não receção de uma mensagem *Reply* implicará a remoção desse nó da lista de vizinhos e da lista de nós pelos quais aguarda um *Ack*.

Heartbeat. As mensagens *Heartbeat* são enviadas pelo nó líder para os seus vizinhos, periodicamente, para comprovar a sua presença. Quando um nó vizinho de um líder não recebe uma mensagem *Heartbeat*, este irá proceder ao início de uma nova eleição.

Estas últimas três, são as mensagens que permitem utilizar o algoritmo em redes cuja topologia está constantemente a alterar.

Na Tabela II estão apresentadas as constituições de cada mensagem do algoritmo. O primeiro parâmetro indica o tipo de mensagem. Nos casos das mensagens *Election*, *Ack*, *Leader* e *Reply* é enviado também o id da computação e o id do nó que a iniciou. Mais à frente será explicado o que se entende por computação. Na mensagem *Election* é também enviado o id do nó líder que foi perdido. As mensagens *Ack* contêm uma flag que caso seja 0, indica que é uma mensagem *Ack* a indicar que o nó já se encontra em eleição, caso seja 1, indica que o nó-mais-valioso que esse nó conhece está no parâmetro max. A mensagem *Leader*, tal como esperado, contém um parâmetro em que indica o id do líder eleito.

Este algoritmo permite que haja múltiplas eleições e até mesmo eleições concorrentes. Isto porque as eleições são identificadas por índice de computação que é constituído por um id (que identifica a computação) e o id do nó que a inicializou. Este par é designado por (num_1, id_1). Uma computação tem maior prioridade do que a outra se apresentar um id de computação maior, ou se forem iguais, o nó que a inicializou ter um id superior. Esta definição é apresentada de seguida.

$$(num_1, id_1) \succ (num_2, id_2) \iff ((num_1 > num_2) \vee ((num_1 = num_2) \wedge (id_1 > id_2)))$$

Isto permite que quando um nó recebe um *Election* com um índice de computação diferente daquele em que se encontra, optar por continuar na computação em que está inserido, ou mudar para a computação cujo índice é maior.

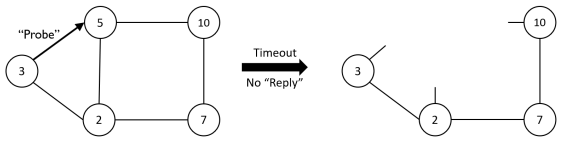

Figura 2. Exemplo de perda de ligação de um nó.

Tal como já foi referido na descrição das mensagens, este algoritmo permite lidar com a falha de nós através do envio das mensagem *Probe* e *Reply* (Figura 2). Para além dessa característica, esta implementação permite a junção de diversas partições de rede. Como podemos observar na Figura 3, quando um nó deteta um novo vizinho, este, caso não esteja em eleição, envia uma mensagem *Leader* a indicar o seu líder atual. Podemos observar na Figura 3(a) que o nó 3 envia ”L(5)” e o nó 7 envia ”L(10)”. O nó 7 como recebe uma mensagem *Leader* com um líder menor que o seu, ignora esta mensagem, mas no caso do nó 7, este ao receber a mensagem com um líder maior, adota esse nó como seu líder e procede ao envio da mensagem *Leader* para os seus vizinhos de forma a que estes atualizem o seu líder (Figura 3(b)).

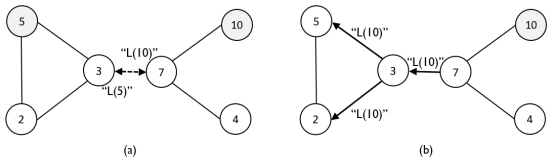

Figura 3. Exemplo do comportamento do algoritmo numa junção de duas partições de rede.

Tabela III. PRINCIPAIS VARIÁVEIS DE CADA NÓ

Variáveis	Significado
inElection	Se o nó está em eleição
parent	O pai do nó
sendAck	Se o nó enviou *Ack* para o pai
leader	O líder atual do nó
neighbors	Lista dos vizinhos do nó
waitingForAck	Lista de vizinhos de quem espera receber um *Ack*
src	O índice da computação em que está inserido

Como foi possível observar na Figura 3, passou-se a designar o nó pelo seu valor, isto porque, por uma questão de simplificação, definiu-se como valor do nó, o seu respetivo id. Para a implementação deste algoritmo foi necessário definir um conjunto de variáveis que cada nó deveria conter. As principais variáveis estão apresentadas na Tabela III juntamente com o respetivo significado.

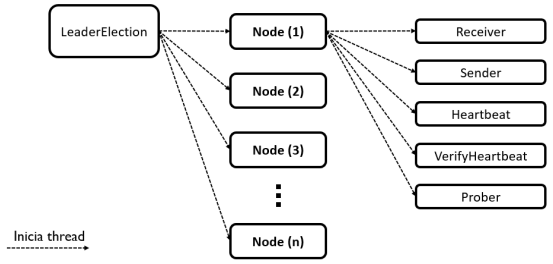

Figura 4. Estrutura da implementação.

A estrutura da implementação deste algoritmo encontra-se na Figura 4. Uma classe principal designada por *LeaderElection* é responsável pela iniciação dos diversos nós que constituem a rede. Cada nó, para além da máquina de estados que executa (Figura 5), inicia diversas *Threads* que realizam tarefas auxiliares que os nós necessitam. A *Thread Receiver* é responsável por receber, do *Socket UDP*, as mensagens que serão guardadas num buffer, para mais tarde serem processadas. A *Thread Sender* terá a funcionalidade contrária, isto é, terá de enviar pacotes *UDP* com mensagens que terão de ser enviadas para outros nós e que se encontram também num buffer. A *Thread Heartbeat* é responsável por, periodicamente, enviar mensagens *Heartbeat* para os vizinhos caso o nó seja o nó líder. A *Thread VerifyHeartbeat*, caso o nó seja vizinho do nó líder, verifica se este enviou a mensagem *Heartbeat*. Por fim, a *Thread Prober* tem a função de enviar mensagens *Probe*, periodicamente, para os nós vizinhos, caso o nó esteja em eleição.

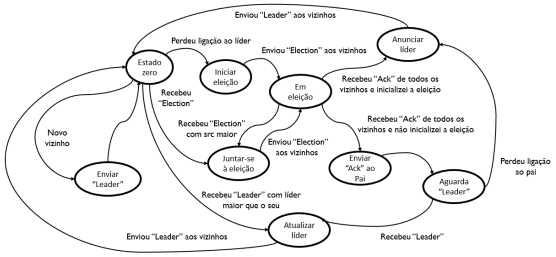

Figura 5. Máquina de Estados de cada nó.

III. RESULTADOS

Para motivos de teste, foram analisadas duas de quatro possíveis métricas de desempenho. São elas: *Tempo médio de eleição* (T), que define o intervalo de tempo entre o momento em que um nó entra em eleição até ao momento em que lhe é atribuído um líder; *Overhead de mensagens* (M), que define o número de médio de mensagens por eleição. Como métricas não analisadas neste projeto temos: *Fração de tempo sem líder* (F), indica a fração de tempo de simulação em que um nó se encontra em eleição; *Rácio de eleição* (R), define o número médio de eleições em que um nó participa por unidade de tempo.

Optámos somente pela análise de duas das características especificadas, porque no ambiente do nosso projeto todas as falhas são introduzidas manualmente. Com isso em mente, não faria sentido analisar fatores que são controlados por nós.

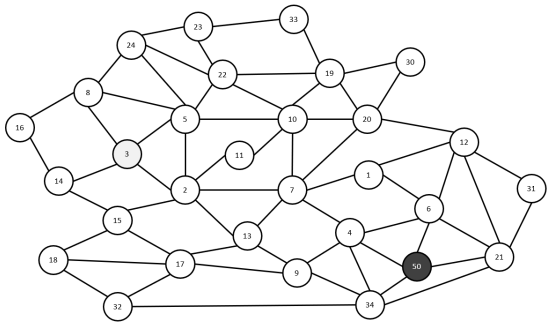

Figura 6. Topologia implementada com 30 nós.

Em todas as simulações, as métricas são estudadas e interpretadas em função do número de nós (N). Inicialmente, foram usados 5 nós, na disposição apresentada na Figura 1. Nas seguintes simulações, houve um acrescento gradual de 5 nós por simulação até um limite de 30 nós (topologia apresentada na Figura 6), dispostos numa topologia *mesh*.

A. Tempo médio de eleição

O tempo de eleição sofreu um aumento espectável com a subida do número de nós. Essa tendência de crescimento é visível no gráfico apresentado (Figura 7). O fenómeno é causado pela demora das mensagens que têm de chegar aos nós que se conectaram mais recentemente e que ainda não têm líder. Este tempo pode ser influenciado pelas mensagens *Probe* que têm a função de detetar nós que já não estão conectados, e a sua importância é fundamental para que um nó

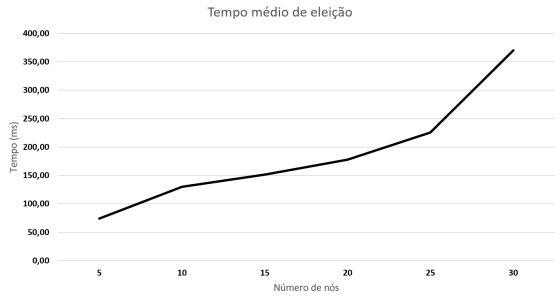

Figura 7. Tempo médio de eleição.

não fique demasiado tempo à espera de *Ack* de filhos que estão *downstream* na *spanning tree*. Nesse sentido, um baixo número de *Probe* trocados implica um aumento no tempo de eleição, caso um nó se tenha desconectado. Também a velocidade da rede segundo a qual é feita a troca de mensagens vai influenciar este parâmetro. Contudo, não tivemos possibilidade de realizar testes dessa natureza.

B. *Overhead de mensagens*

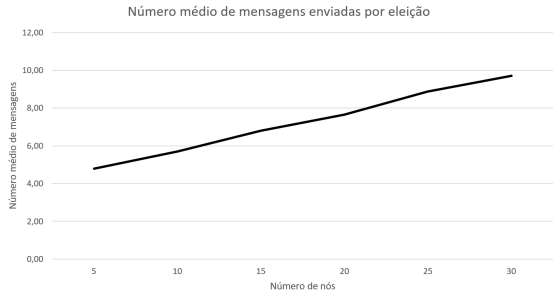

Figura 8. Número médio de mensagens enviadas por eleição.

No que toca ao número médio de mensagens por eleição, podemos observar na Figura 8, que há, da mesma forma, um aumento suave à medida que a rede cresce. Apesar desse aumento, ao analisar os valores propriamente ditos, é percetível uma grande variação de mensagens trocadas entre os nós que possuem muitos vizinhos e os que têm apenas uns poucos.

Como fatores interferentes nos resultados da avaliação deste parâmetro temos a frequência de envio de mensagens *Probe* e ainda a topologia de rede utilizado. O aumento da frequência de envio de mensagens *Probe* vai implicar um aumento do número de mensagens trocadas, por outro lado, pode ser benéfico noutros fatores de avaliação, como já tinha sido referido. Podemos também inferir uma influência da topologia da rede, no sentido em que quanto maior for a lista de vizinhos de um nó, maior será o número de mensagens trocadas com os mesmos.

IV. Análise Crítica

Uma das conclusões que retiramos após a finalização deste projeto e respetivos resultados é que, o tipo de comunicação usado tem influência nas diferentes métricas analisadas. No nosso caso, optámos por utilizar comunicação *UDP* ponto-a-ponto, o que implica uma maior troca de mensagens comparativamente ao caso de se utilizar *broadcast*. Outro fator que pode ser relevante referir é que, na nossa implementação, a mensagem *Leader* enviada pelo *source node* não é enviada por *broadcast*, mas sim por comunicação ponto-a-ponto entre o *source node* e os nós vizinhos. Isto implica um acréscimo de tempo até que todos os nós assumam o mesmo nó como líder. Para além disso, um nó líder só envia mensagens *Heartbeat* para os nós vizinhos, e apenas estes podem inicializar a eleição. Este facto pode levar a situações críticas como por exemplo, o nó líder estar ligado apenas a um nó, que entretanto é desconectado da rede. Como os restantes nós da rede perdem o acesso a esse nó, que por sua vez já não está ligado ao nó líder, esses nós não terão forma de saber se o nó líder se perdeu. A solução para este caso, era os nós vizinhos do líder terem que propagar as mensagens de *Heartbeat* para os restantes nós da rede. Isto implicaria que qualquer nó, perante a ausência dessas mensagens após um determinado período de tempo, poderiam iniciar a computação.

Por último, faltou tempo para uma melhor definição nos testes a realizar e no tratamento dos resultados obtidos: ver o que influenciava a solução, fazer alterações necessárias com intenções de obter resultados mais conclusivos e expandir o leque de testes executados (em múltiplos computadores, em diferentes redes). Contudo, o facto da nossa equipa contar apenas com dois membros, fez com que a carga de trabalho fosse elevada para ambos e limitou uma realização de testes abrangente. No que se refere a contribuições, ambos os membros contribuíram significativamente. O protocolo foi desenvolvido em conjunto, juntamente com a realização dos testes e do relatório.

Referências

[1] S. Vasudevan, J. Kurose and D. Towsley. Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. In *Proc. of the 12th IEEE ICNP'04*, 2004.

[2] S. Vasudevan, J. Kurose and D. Towsley. Design and Analysis of a Leader Election Algorithm for Mobile, Ad Hoc Networks. UMass CMPSCI Technical Report 03-20.