Sistemas Distribuídos
Leader Election in Ad Hoc Network

Hélio Manuel Silva Puga
MIEEC@FEUP
up201305586@fe.up.pt

Pedro Miguel Flores Rodrigues
MIEEC@FEUP
up201306231@fe.up.pt

Rodrigo Gomes Borges
MIEEC@FEUP
up201304633@fe.up.pt

***Resumo*—A necessidade de consenso na escolha do líder, em sistemas distribuídos, é muitas vezes necessária para garantir o correto funcionamento destes. Neste sentido, e num contexto de uma rede *Ad Hoc* móvel, foi implementado um algoritmo de eleição de líder capaz de garantir que, ao fim de um período finito de tempo, o nó com melhores condições é escolhido como líder. O algoritmo implementado baseia-se num processo de *grow* seguido de *shrinkage*, permitindo a geração de uma *spanning tree* da rede. O algoritmo implementado lida com os problemas induzidos pela presença de uma rede dinâmica, tais como: *crashes* e *restarts* de nós; partição da rede em sub-redes devido à perda de ligações entre os nós e o *merge* dessas partições; ocorrência de múltiplas eleições concorrentes por parte dos diversos nós. A análise de resultados será baseada numa topologia com um número escalável de nós, onde será analisado o número de mensagens e a duração do processo de eleição recorrendo a quatro tipos de testes.**

Palavras-chave: *Ad Hoc Network*; Eleição de Líder; Sistemas Distribuídos; *Spanning Tree*.

I. Introdução

O presente relatório foi realizado no âmbito da unidade curricular de Sistemas Distribuídos do 5º ano do Mestrado Integrado em Engenharia Electrotécnica e de Computadores. Tem como objetivo descrever a metodologia utilizada na implementação do algoritmo de eleição de líder descrito em [1], complementado com resultados obtidos durante os testes realizados.

II. Definição do problema

Algoritmos de eleição de líder são essenciais para o controlo de aplicações envolvendo sistemas conectados por redes *wired* e *wireless*. No contexto de redes *wireless*, a eleição de líder pode desempenhar papéis tais como, distribuição de chaves, coordenação de *routing* e de sistemas de sensorização.

Em ambientes onde os diferentes nós que compõem a rede são móveis, é usual que a topologia da rede mude esporadicamente e que os nós possam entrar ou sair da rede dinamicamente. Assim, a ocorrência de eleições de líder é frequente, demonstrando-se uma funcionalidade crítica do funcionamento deste tipo de sistemas. Pretende-se que ao recorrer a algoritmos de eleição de líder, eventualmente após um período de tempo, seja eleito consensualmente um só nó como líder do sistema distribuído. Torna-se assim imprescindível um algoritmo capaz de lidar com os problemas emergentes de uma rede dinâmica. Destacam-se, entre estes, a necessidade de lidar com partições na rede, múltiplas eleições simultâneas, *crash* e *restart* de nós e a união de sub-redes fruto de partições antecedentes.

A. Premissas

Na implementaçao e teste do algoritmo, foram definidas as seguintes premissas referentes aos nós e ao comportamento da rede:

- As ligações entre os nós são bidirecionais e do tipo *FIFO - First In Frist Out*;
- Cada nó possui um ID único, e um valor de *desirability* (valor indicativo da qualidade do nó para ser líder);
- A entrega de mensagens é garantida se ambas as entidades permanecerem conectadas durante o envio destas;
- Cada nó possui um *buffer* de recepção com tamanho suficiente para evitar *overflow*.

Estas premissas permitem simplificar a implementação e teste do algoritmo, possibilitando o foco na determinação de soluções para os problemas indicados anteriormente para uma rede *Ad Hoc* móvel.

B. Objetivos

Pretende-se que o algoritmo seja capaz de garantir que ao fim de um tempo finito, todos os nós constituintes da rede definam consensualmente um mesmo nó como seu líder.

Determinadas aplicações beneficiam com a eleição de um líder que apresente as melhores características relativas à rede, em vez de uma escolha aleatória de líder. Por este motivo, o algoritmo implementado possui a capacidade de eleger o nó que apresente as melhores condições, como por exemplo maior percentagem de bateria, menor distância média aos restantes nós ou melhor histórico de desempenho.

Tendo em conta o contexto em que o algoritmo se insere, pretende-se que, após a sua implementação, este consiga lidar com as seguintes possíveis ocorrências:

- Múltiplas eleições concorrentes;
- Partições de nós e do líder;
- *Merges* entre partições;
- *Crashes* e *restarts* de nós.

III. Algoritmo de Eleição de Líder

A. Algoritmo para uma rede estática

Seguidamente, descreve-se o algoritmo implementado numa situação em que se considera uma rede estática. Assim,

assume-se que os nós e as ligações entre estes nunca falham, impossibilitando a ocorrência dos problemas mencionados anteriormente relativos a uma rede dinâmica. O algoritmo baseia-se num processo de construção da *spanning tree* referente à rede, sendo este procedimento faseado. Numa primeira fase, o algoritmo procura o *growing* da *spanning tree*, sendo que numa segunda fase procede-se ao *shrinking* da mesma.

Para o processo de construção da *spanning tree* e eleição do líder mais adequado, são usados três tipos de mensagens: *Election*; *Ack* e *Leader*. Todas as mensagens utilizadas serão detalhadas no ponto III-A1.

1) Mensagens: Relativamente ao algoritmo para uma rede estática, são utilizados três tipos de mensagens partilhadas entre os nós durante a sua execução.

Para além destas três mensagens, para responder aos problemas relativos à existência de uma rede dinâmica, é necessária a utilização de mais três tipos de mensagens: *Probe*; *Reply* e *Heartbeat*. Estas não são necessárias para a eleição numa rede estática, sendo explicadas aquando da ilustração do algoritmo para uma rede dinâmica na secção III-B.

A estrutura de cada mensagem pode ser observada de seguida, sendo que os campos <type> <data> podem ser observados na tabela I.

<msg_Id> | <sender_Id> | <dest_Id> | <type> | <data>

O campo <msg_Id> corresponde ao ID da mensagem, valor que permite a identificação única de cada mensagem. O campo <sender_Id> corresponde ao ID do nó emissor, sendo que o campo <dest_Id> corresponde ao destino pretendido para a mensagem. Relativamente ao campo <data>, este contém a informação que se pretende enviar com cada tipo de mensagem. Na tabela I, é possivel verificar como varia o valor do campo <data> para cada tipo de mensagens utilizadas:

Tabela I: Mensagens utilizadas durante a execução.

<type>	**<data>**
Election	<src_Num, src_Id>
Acknowledge	<Most_Valued_Node>
Leader	<Leader_Id>
Probe	—
Reply	—
Heartbeat	<Leader_Id>

2) Variáveis: Para o controlo e progressão do algoritmo, é necessária a atualização de diversas variáveis que permitem ao algoritmo conhecer o estado atual do processo.

As variáveis presentes em cada nó apresentam-se na tabela II.

Tabela II: Lista de variáveis mantidas por cada nó durante a eleição.

Variáveis	**Significado**
δ_i	Identifica se o nó esta numa eleição ou não
P_i	Pai do nó i na spanning tree
lid_i	Lider do nó i
N_i	Mapa com os vizinhos atuais do nó i
S_i	Set de nós dos quais falta receber Ack
src_i	*Computation-index*

Cada nó mantém uma variável booleana δ_i, cujo valor é 0, se o nó não estiver numa eleição, e 1, se estiver. A variável P_i contém o ID do nó considerado pai, enquanto que a variável lid_i possuí o ID do líder. Cada nó possuí um mapa, N_i, contendo todos os nós vizinhos, e um *set*, S_i, onde estão contidos todos os nós vizinhos dos quais está pendente a recepção das mensagens de *Ack*. A variável src_i corresponde ao *computation-index* e permite distinguir entre diferentes computações, sendo utilizada para a resolução de casos de múltiplas ocorrências de eleições simultâneas.

3) Inicialização do algoritmo: Cada nó, ao iniciar a sua execução, irá iniciar as variáveis de eleição de líder apresentadas anteriormente, e iniciar um processo de eleição. Isto deve-se ao valor presente na variável lid_i do nó, que visto que iniciou a sua execução, se encontrará com o valor -1, indicando que não possuí líder.

4) Explicação do algoritmo: Numa topologia de rede estática, após a inicialização de um determinado nó, este inicia uma nova eleição. Assim, este coloca a sua variável δ_i a 1, indicando que se encontra numa eleição e envia para todos os nós presentes no mapa N_i uma mensagem de *Election*, adicionando-os ao *set* S_i. Cada vizinho deste, ao receber esta mensagem, se verificar que é a primeira mensagem de *Election* recebida referente a esta eleição, designa este como seu pai, alterando a variável P_i. Para além disto, estes nós reenviam a mensagem de *Election* para os seus vizinhos, exceptuando o seu pai. No acontecimento de um nó receber uma mensagem de *Election* com o mesmo valor de *computation index* de um nó que não é o seu pai, este responde imeadiatamente com uma mensagem *Ack*.

Após a propagação desta mensagem de eleição, a *spanning tree* encontra-se completamente *grown*, iniciando-se o processo de *shrinking* de volta para o nó que iniciou a eleição. O processo de *shrinkage* inicia-se nas folhas, onde nenhum nó possuí filhos. Eventualmente todas as folhas enviam *Ack* para os seus pais, e estes por sua vez, após receberem *Ack* de todos os nós presentes no set S_i, enviam a mensagem de *Ack* pendente para os seus pais. Nestas mensagens de *Ack* encontra-se, tal como indicado na tabela I, o valor do nó mais indicado para ser eleito líder que cada nó recebeu até à data. Após a recepção dos *Ack* de todos os seus filhos, o nó que iniciou a eleição possui toda a informação necessária para determinar o nó que deve ser considerado o líder.

Assim que o nó que iniciou a eleição tenha decidido qual o nó que deve ser escolhido como líder, este envia uma mensagem de *Leader*, contendo o ID do líder escolhido, para os seus vizinhos. Estes, por sua vez, propragam esta mensagem pelo resto da rede, verificando-se concesso na escolha de um nó como líder.

B. Algoritmo para uma rede dinâmica

Numa topologia dinâmica, a presença de mobilidade nos nós, a possibilidade de *crash* e *restart* destes, bem como as potenciais falhas de comunicação, tornam o algoritmo apresentado anteriormente inadequado. Descreve-se de seguida, o algoritmo de eleição de líder para o caso de uma topologia

dinâmica numa rede *Ad Hoc*. Neste algoritmo, ao contrário do exposto anteriormente, são considerados os casos onde várias eleições podem ocorrer concorrentemente bem como a presença de falhas de nós e ligações.

1) Múltiplas eleições concorrentes: Numa topologia de rede dinâmica, o *crash* ou perda de comunicação com líder por parte de um nó pode ser identificado por mais do que um nó, o que pode resultar em eleições concorrentes quando ambos detetam essa falha. Para além disto, é possível que vários nós iniciem a sua execução simultaneamente, gerando também eleições concorrentes.

Para resolver o problema descrito, é utilizado um *computation index*, que mantém o número da eleição <*num*> e o ID do nó que iniciou essa eleição <*id*>. Caso um nó que esteja numa eleição receba uma eleição diferente da que atualmente se encontra, este compara os *computation indexes* das duas eleições através da equação 1.

$$\begin{aligned} \langle num_1, id_1 \rangle \succ \langle num_2, id_2 \rangle \iff \\ ((num_1 > num_2) \vee ((num_1 = num_2) \wedge (id_1 > id_2))) \end{aligned} \quad (1)$$

Caso o *computation index* da eleição recebida seja superior ao da sua eleição atual, o nó desiste da sua eleição, aceitando a eleição recebida e retransmitindo-a para os restantes nós.

A utilização deste procedimento por parte de todos os nós numa rede, garante não só que apenas uma eleição ocorre por nó, mas também que todos os nós chegam a um consenso sobre qual o nó que deve realizar a eleição.

2) Partições da rede: Uma partição corresponde ao *split* de uma rede em sub-redes, devido à perda de comunicação entre estas. Na sua ocorrência, os nós que ficam na sub-rede que integra o líder atual, mantêm-se com o mesmo líder, enquanto que os nós da restante sub-rede, por perderem comunicação com o líder, iniciam uma nova eleição entre os nós dessa sub-rede para se apurar um novo líder.

3) Merges de partições: No caso de existirem duas sub-redes com líderes distintos, na eventualidade da sua reconexão, verifica-se a partilha de mensagens de *Heartbeat* entre as sub-redes. Os nós que estabelecem esta comunicação, ao receberem a mensagem de *Heartbeat* comparam o valor do líder dessa mensagem com o ID do seu próprio líder. Caso o valor presente no *Heartbeat* seja superior ao valor do ID do seu líder, estes nós alteram o valor presente na variável Lid_i para o ID do líder recebido e propagam esta mensagem para os seus vizinhos, contendo o novo líder. Desta forma, todos os nós da nova rede formada irão chegar a um consenso quanto ao líder escolhido.

4) Crash, Restarts e Recovery de nós: Quando um nó sofre um *crash*, este problema é lidado como se tratasse de uma partição de rede, como descrito na secção III-B2, mas neste caso de um só nó.

No caso em que há um *restart*, se o nó reiniciar sem líder, este iniciará uma nova eleição. Caso o nó recupere o seu estado anterior, ou seja, uma *recovery*, o procedimento será o mesmo que um *merge* como descrito na secção III-B3.

C. Implementação

Para a implementação do algoritmo descrito na secção III recorremos à linguagem de programação Java, definindo uma arquitetura de programação orientada a objetos.

1) Arquitetura: Foram definidas 4 classes principais para o funcionamento do nosso algoritmo e ainda 3 *threads*, tornando possível a paralelização da computação do algoritmo de eleição com a comunicação e controlo temporal das mensagens.

- LeaderElectionAdHocNet: nesta classe está implementada a máquina de estados descrita na secção III-C2, sendo responsável pela execução de todo o algoritmo assim como da inicialização do mesmo;
- MainNode: esta classe contém todas as variáveis necessárias para a eleição do líder. Para além disto, esta implementa o método que permite obter as mensagens que são enviadas pelos nós vizinhos;
- Node: esta classe implementa os nós vizinhos do "MainNode", ou seja, o mapa N_i do "MainNode"vai ser composto por objetos da classe Node. Esta classe implementa os métodos que permitem enviar, para o nó vizinho, os diferentes tipos de mensagens;
- Message: esta classe implementa a estrutura das mensagens descritas na secção III-A1;
- Threads: as threads implementadas permitem fazer o envio das mensagens do tipo *Probe* em paralelo com a execução do algoritmo. Para além disto, garantem que não ocorrem perdas de mensagens através da implementação de uma fila do tipo FIFO. Por último, permite a contagem de tempo de modo a controlar as mensagens de *Heartbeat*.

2) Máquina de estados: Para implementar o algoritmo descrito acima foi implementada a máquina de estados representada na figura 1. Cada nó executa esta máquina de estados num *loop* infinito sendo que quando está no estado *Standby* o nó não está a participar em nenhuma eleição.

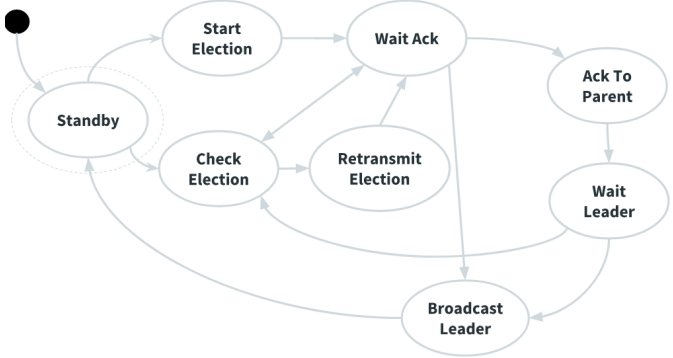

Figura 1: Máquina de estados executada por cada nó.

3) Comunicação: A comunicação foi implementada usando o protocolo UDP em modo *unicast*. Apesar deste protocolo não garantir a entrega de mensagens, a falha de entrega de mensagens não causa problemas visto que o algoritmo consegue lidar com este tipo de falhas. O UDP traz vantagens

relativamente ao TCP pois implica menos *overhead* no envio de mensagens uma vez que não é necessário estabelecer uma conexão com o outro nó, fazendo com que este seja o protocolo mais indicado.

IV. Testes e resultados obtidos

A. Métricas de desempenho

Foram definidas duas métricas de desempenho que permitem avaliar o comportamento do algoritmo implementado:

- *Message overhead*: número médio de mensagens enviadas por eleição;
- *Mean Election time*: duração média do processo de eleição.

Através destas métricas, é possível analisar o comportamento do algoritmo através das variações que acontecem nas mesmas para as diferentes topologias de testes utilizadas.

B. Topologias usadas para testes

Nos testes realizados, foram distribuídos os diversos nós por 3 computadores distintos, ligados através de uma rede *Wi-Fi*. A topologia utilizada encontra-se apresentada na figura 2. Para o aumento do número de nós em cada teste, é adicionada uma nova camada de nós a cada rede local de cada computador. Isto permite obter a evolução dos valores das métricas utilizadas, consoante o número de nós da rede testada.

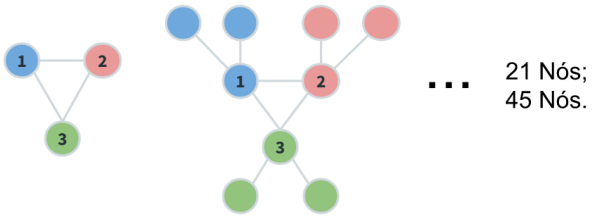

Figura 2: Topologia da rede para teste.

C. Testes realizados

1) Iniciação simultânea de todos os nós: Neste teste, todos os nós foram iniciados ao mesmo tempo e foi avaliada a evolução do *Message-overhead* e do *Mean Election-time* em função da evolução do número de nós, como referido na secção IV-B.

Podemos observar que no gráfico 3 e 4 existe um crescimento exponencial do *Mean Election Time* e do *Message Overhead*, de notar que o gráfico tem uma escala logarítmica. Isto deve-se a este caso ser o pior caso de execução do algoritmo implementado, uma vez que todos os nós estão a ser iniciados ao mesmo tempo e consequentemente a iniciar uma eleição, levando a um elevado número de computações paralelas que implicam uma elevada troca de mensagens para decidir qual a computação e executar.

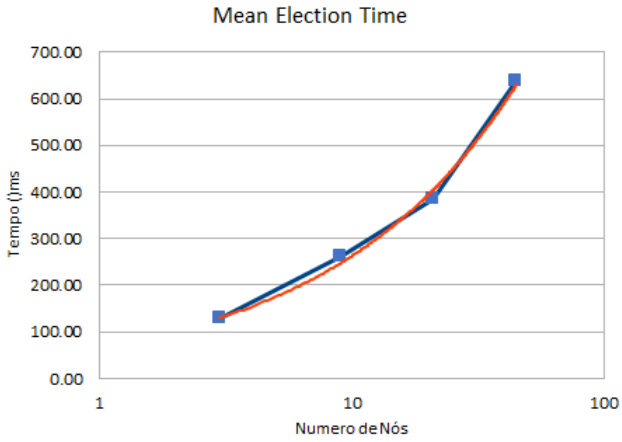

Figura 3: *Mean election time* para quando todos os nós iniciam uma eleição ao mesmo tempo.

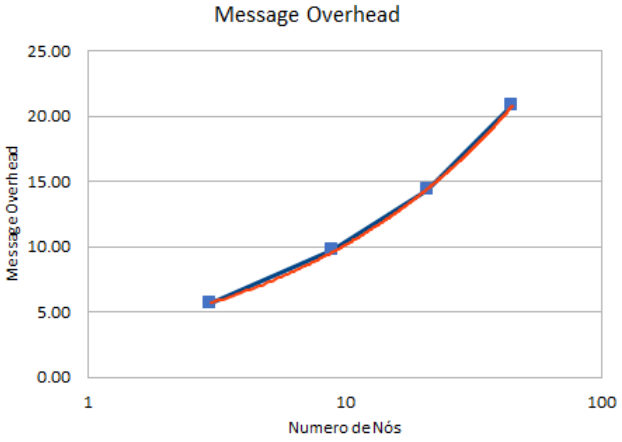

Figura 4: *Message overhead* para quando todos os nós iniciam uma eleição ao mesmo tempo.

2) Perda do líder: Para além do teste enunciado anteriormente, avaliou-se o comportamento do algoritmo através das mesmas métricas aquando da perda do líder para os diferentes números de nós, como referido na secção IV-B.

Pela análise dos gráficos 5 e 6, conclui-se que os valores obtidos são significativamente inferiores aos obtidos no teste anterior. Isto deve-se ao facto de, neste caso, alguns nós detetarem mais cedo a ausência do líder, iniciando um processo de eleição. Assim, verifica-se um menor número de eleições concorrentes e, consequentemente, um menor valor para o número médio de mensagens e para o tempo necessário para concluir a eleição.

É possível observar no gráfico 6 que o *Message Overhead* diminui com o aumento do número de nós. Isto deve-se ao facto que o aumento do número de nós na razão de 2^n, sendo n o número de andares, não corresponde a um aumento do número médio de mensagens na mesma razão, visto que os nós no último andar da árvore apenas enviam uma mensagem,

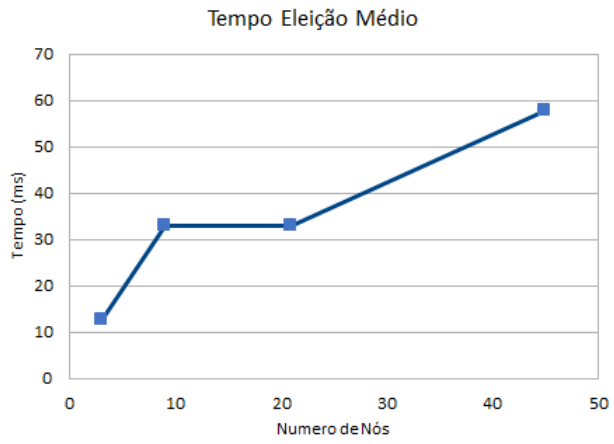

Figura 5: *Mean election time* para uma eleição quando perde o líder.

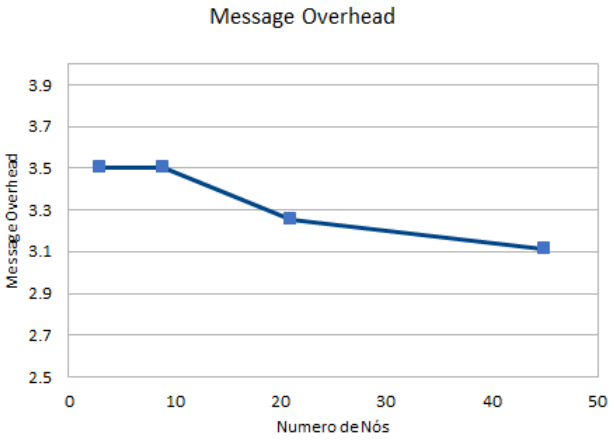

Figura 6: *Message overhead* para uma eleição quando perde o líder.

que é a mensagem de *Ack* ao pai.

3) Criação de partições: Para o teste da ocorrência de partições, recorreu-se à simulação da quebra de comunicação entre os nós que serviam de ligação entre as sub-redes da rede, levando à criação de três partições presentes em cada um dos computadores utilizados.

Neste teste, nas sub-redes sem líder, o problema será tratado como indicado na secção III-B2, levando a resultados semelhantes aos apresentados na secção IV-C2. Para a sub-rede que permaneceu com o líder inalterado, não há necessidade de uma nova eleição.

4) Ocorrência de Merge: No caso do teste da ocorrência de *merges*, as ligações entre os diferentes computadores foram restabelecidas, verificando-se que não se iniciou nenhuma eleição, existindo apenas a partilha de líder através das mensagens de *Heartbeat*, sendo que todos os nós adotaram o maior líder presente, caso não tivessem previamente esse líder definido.

Visto que neste teste não ocorre nenhuma eleição, não podemos extrair informação relevante em relação às métricas apresentadas anteriormente, no entanto, podemos concluir que o tempo que é necessário para que as duas sub-redes escolham consensualmente um mesmo líder será o tempo necessário para a propagação de uma mensagem de *Heartbeat* ao longo de toda a rede. De realçar que este tempo é, maioritariamente, dependente das características da rede em que os nós estão inseridos.

V. Conclusão e análise critica

Em suma, considera-se que o algoritmo implementado e os resultados obtidos são bastante satisfatórios, realçando o cumprimento de todos os objetivos delineados no planeamento do projeto. No que diz respeito à contribuição dos diferentes elementos da equipa, consideramos que os três elementos constituintes apresentaram um contributo semelhante para o desenvolvimento deste projeto.

As principais dificuldades encontradas surgiram sobretudo pela pouca exposição dos procedimentos de implementação encontrados no artigo relativo ao tema deste projeto. Para além disto, a impossibilidade de teste do nosso algoritmo para uma grande quantidade de nós, devido à ausência de um ambiente de simulação próprio, revelou-se um grande entrave, na medida em que reduziu a quantidade de resultados obtidos, influenciando a análise das tendências destes.

Relativamente a melhorias futuras, a utilização de um ambiente de simulação capaz de simular redes móveis compostas por mais de 50 nós, permitiria uma melhor avaliação do comportamento do algoritmo implementado e a obtenção de melhores conclusões relativas aos resultados dos testes realizados. Para além disto, destacamos a possibilidade de implementação de uma versão otimizada do algoritmo através da implementação de comunicações em modo UDP *multicast*, o que permitiria diminuir a quantidade de mensagens *Election* necessárias, pois uma só mensagem seria suficiente para chegar a todos os nós, e a diminuição do número de mensagens *Ack* necessárias, sendo que para isto o algoritmo seria adaptado de modo a permitir que apenas fosse necessário o envio de *Ack* ao nó pai.

Como suporte a este relatório, no link https://youtu.be/GQa2o_LN8AI encontra-se uma pequena demonstração do algoritmo implementado em funcionamento. Para auxiliar a visualização, foram inseridas legendas que devem ser ativadas.

Referências

[1] Sudarshan Vasudevan, Jim Kurose, and Don Towsley
Design and Analysis of a Leader Election Algorithm for Mobile Ad Hoc Networks. Unniversity of Massachusetts, Amherst.