3
Ferramentas de Simulação e Algoritmos

Neste Capítulo serão abordados todos os temas referentes às simulações realizadas. Primeiramente na seção 3.1 serão descritas de maneira breve alguns simuladores que possibilitam a pesquisa das redes de sensores sem fio e a partir daí será justificada a escolha da ferramenta escolhida para este trabalho. Posteriormente são apresentados os algoritmos implementados e as modificações realizadas para adequá-los ao cenário da simulação. Por ultimo será discutido o protocolo de encaminhamento.

3.1
Características e comparações das ferramentas para simulação

Hoje em dia o uso de ferramentas de simulação é muito importante no projeto, análise e implementação de sistemas de comunicação, principalmente quando estes sistemas são caros e complexos. A simulação permite avaliar objetos ou atividades hipotéticas ou reais de diferentes sistemas a custos reduzidos.

Os simuladores de rede são importantes no momento de desenvolver, analisar e aperfeiçoar algoritmos e protocolos. Algumas das vantagens de trabalhar com simuladores são as seguintes:

- Podem ser realizados testes em diversas redes e ambientes;
- Algoritmos podem ser executados em múltiplos cenários;
- Os testes são feitos em ambientes controlados onde é mais fácil variar parâmetros.

Estes fatores dão flexibilidade para a implementação de nosso projeto já que trabalhar com nós reais traz maiores custos e tem um maior grau de complexidade para realizar a avaliação em distintos cenários.

O principal objetivo desta seção é justificar a escolha da ferramenta de simulação adotada no presente projeto.

Nos próximos parágrafos serão apresentadas, com suas respectivas características, alguns dos simuladores de rede, mais comumente usados nessa área de pesquisa.

- **NS-2**

É a segunda versão do *Network Simulator,* que é um simulador de eventos discretos orientado à pesquisa de redes. O NS-2 pode ser empregad para distintas tecnologias de rede. O núcleo do simulador foi escrito em linguagem C++, permitindo assim a construção de algoritmos que trabalham com grandes conjuntos de dados, e proporciona uma velocidade maior na implementação de protocolos. Ele também faz uso da linguagem OTcL (*Object oriented tool command language*), que é uma linguagem interpretada e interativa, desta forma os programas podem ser alterados e ser facilmente re-executados.

Uma observação que deve ser feita é a seguinte, o NS não fornece estatísticas de simulação de modo automático, estas devem ser obtidas através do *script* ou pela manipulação de objetos especiais.

- **NS-3**

Da mesma forma que na plataforma anterior ele também é um simulador de redes baseado em eventos discretos. É necessário ressaltar que este aqui não é uma extensão do NS-2. Ambos são escritos em linguagem C++ mas o ns-3 é um simulador totalmente novo e não pode ser empregado para alguns dos modelos da versão anterior. Apenas algumas funcionalidades do ns-2 são compatíveis com o ns-3.

No ns-3 todo o simulador é escrito em C++(essa é a maior diferença com o ns-2 onde alguns componentes são escritos em C++ e outros em OTcl) permitindo o uso opcionaldo Python (outra linguagem de programação). É possível utilizar ferramentas para a analise de pacotes, da mesma forma quee no ns-2. Para visualizar os resultados da simulação pode-se utilizar o *nam trace* que é uma ferramenta que permite ter informações como a conectividade dos nós.

- **Omnet++**

È uma estrutura orientada a eventos discretos para a simulação de redes. Em si não é um simulador de nada concreto, mas fornece infraestrutura e ferramentas para escrever simulações. O conceito fundamental do Omnet é a arquitetura dos componentes para os modelos de simulação. Os modelos são montados por uma série de componentes reutilizáveis que tem por nome módulos. Eles podem ser combinados de muitas formas ou maneiras. Os módulos são programados em C++ e modelam o comportamento de componentes discretos do sistema que se deseja simular. Estes módulos podem ser combinados, usando uma linguagem de alto nível chamada NED, formando assim módulos compostos.

As simulações em Omnet podem ser executadas em diversas interfaces de usuário.

O Omnet fornece estatísticas de simulação e também oferece a opção de uso de outro tipo de ferramentas para criar gráficos ou realizar outros tipos de analise.

- **Atarraya**

É uma ferramenta de simulação para ensinar e pesquisar algoritmos para o controle de topologia (*topology control*), desenvolvido em Java. Ele é portável para distintas plataformas. Ele tem protocolos para a mobilidade, encaminhamento, manutenção da topologia, construção da topologia. Ele tem uma interface visual muito agradável para o usuário e permite observar eventos e estatísticas durante a simulação. Os resultados fornecidos pela ferramenta são de muita utilidade para o projeto em questão já que são gerados muitos resultados que podem ser avaliados de forma quase automática e sem necessidade de fazer uso de outras ferramentas. Atarraya é uma ferramenta de grande utilidade, mas ainda está longe de atingir seu verdadeiro potencial, com muitas perspectivas de aperfeiçoamentos, mas hoje em dia ainda não é uma ferramenta solida já que conta com poucos protocolos e cenários limitados.

- ***TinyOs***

O *TinyOs* não é um simulador, em realidade ele é um sistema operacional desenvolvido especificamente para trabalhar com redes de sensores e cumprir com os requesitos específicos destas (considera todas as características e limitantes em termos de capacidade, memória, energia e trafego de dados de uma RSSF). Ele é um sistema orientado a eventos, ou seja, camadas de baixo nível tem a capacidade de enviar eventos as camadas acima. A linguagem empregada é nesC [referencia] que é uma extensão da linguagem C. O TinyOS implementa interfaces de comunicação, temporizadores, portabilidade com sensores e um controlador de energia que permite trabalhar com otimizações no tempo de vida dos nós. Os programas desenhados nele estão prontos para serem instalados em motes reais, em outras palavras o *TinyOs* realiza a integração com o *hardware*. Para não perder as vantagens de projeto inerentes aos simuladores, ele possui seu próprio simulador o TOSSIM [referencia], no entanto ele é muito simples e não conta com um modulo que nos permita simular o chip do radio CC2420 bem como as variações nos níveis de potência, fator de suma importância na nossa pesquisa.

- ***Avrora***

O *Avrora* é um simulador compatível com os programas desenhados para os microcontroladores do Atmel e o *hardware* da família Mica, ele poderia ser considerado como um emulador já que reproduz o comportamento exato de um mote (conjunto de *hardware*) incluindo o chip de radio CC2420. Ele é escrito em java pelo qual o desenho de programação orientada a objeto permite encapsular nas simulações informações, dispositivos e estados o que faz dele um software flexível e portável. *Avrora* oferece três mecanismos principais: testes, temporizadores e eventos, isto permite analisar o comportamento dos distintos desenhos em qualquer momento do processo de simulação sem mexer no código e no simulador. Possui uma série de ferramentas para a análise dos distintos desenhos, monitores de pacotes e de consumo energético são alguns deles. Ele tem um modelo de propagação para um ambiente interior(*indoor*)

Para fazer a escolha da ferramenta na qual foi desenvolvido o nosso projeto foram analisados e comparados os distintos simuladores mencionados. Dentre os critérios avaliados encomtran-se: interface gráfica, atendimento técnico, termos da licença de *software*, modelos de energia, estatísticas das simulações, documentação sobre o simulador e reconhecimento na comunidade cientifica. Cada um destes itens foi avaliado subjetivamente, buscando a melhor ferramenta para ser usada neste projeto, o que não quer dizer que a escolhida seja a melhor ferramenta para todas as aplicações.

Todos os simuladores considerados possuem licença livre, na característica de documentação o Atarraya demonstrou ser uma ferramenta em desenvolvimento com muito pouca informação sobre ele disponível., Lá todos os outros simuladores avaliados apresentam ampla quantidade de informação (publicações, manuais e foros de consulta entre outros). Em termos de interface com o usuário, tanto o Omnet, quanto o Atarraya e bem como o TinyOs oferecem uma interface de usuário muito amigável. Os simuladores da família NS (NS2 e NS3) são mundialmente conhecidos pela comunidade cientifica são aqueles que possuem a maior quantidade de pesquisadores trabalhando com eles. Por outro lado o Omnet++ esta atraindo maior quantidade de projetos com o passar do tempo, enquanto o TinyOS tem um grande reconhecimento na área de redes de sensores sem fio. No critério de estatísticas o Avrora demonstra ser uma ferramenta muito útil já que tem uma série de monitores prontos para realizar o análise das distintas implementações.

Observando todos esses fatores foi determinado que o TinyOS e o Avrora são as duas ferramentas que mais se adéquam as nossas necessidades, além

dos critérios acima falados o fator determinante para a escolha foi realizar um trabalho que se ajuste as condições reais dos sistemas de redes de sensores sem fio. Para avaliar algoritmos com o TinyOS são necessários nós reais, pelo qual desenvolver um algoritmo sem a ajuda de um simulador termina sendo uma tarefa lenta, pouco flexível e com custos econômicos elevados. Por isso o Avrora se mostrou ser o complemento perfeito para este sistema operacional já que ele usa o executável gerado pelo TinyOS, emula o funcionamento do chip real, neste caso o Micaz, e fornece uma série de ferramentas prontas para o analise das implementações. Assim pode se aproveitar todas as vantagens de projeto que os simuladores oferecem.

Na Figura 3.1 pode se observar o esquema utilizado para implementar e validar nossos algoritmos. Ele foi desenvolvido o código em nesC no TinyOS, o simulador TOSSIM serviu para testar a lógica dos distintos processos do algoritmo e o Avrora permitiu avaliar os algoritmos em questão e também permitiu corrigir erros em partes do código que faziam uso da interface de radio CC2420.

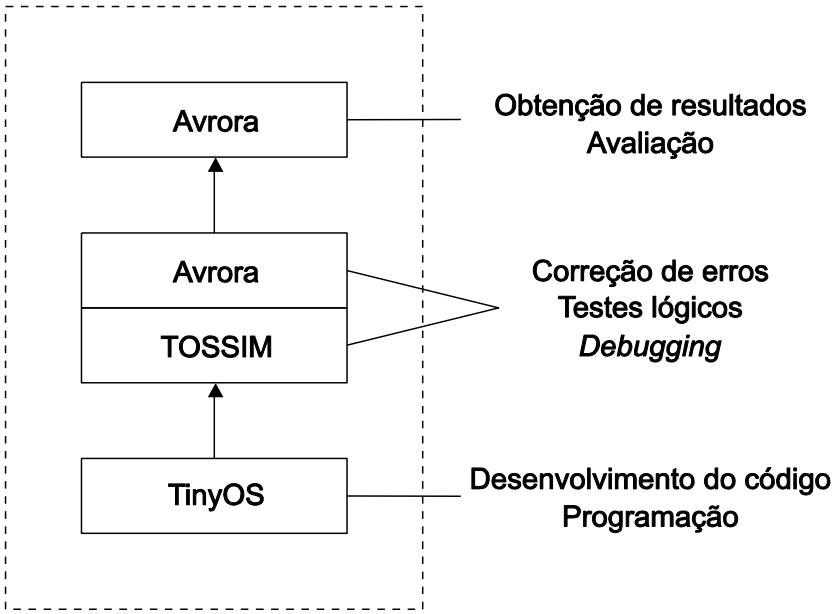

Figura 3.1: Diagrama em blocos da metodologia de simulação

3.2 Algoritmos Propostos

Nesta seção serão descritos os algoritmos de eficiência energética (um de controle de potência e outro de controle de topologia) nos quais nos baseamos para realizar o trabalho. Cabe lembrar que eles foram apresentados em [19], as modificações que foram realizadas como parte do trabalho para adequa-los ao

cenário de simulação também são detalhadas. Por ultimo é descrito o protocolo de roteamento projetado para avaliá-los.

3.2.1 DTNBOR

O Algoritmo DTNBOR (*Determine the minimum Transmission power to reach each NeighBOR*) proposto por Xiao Chen e Neil C. Rowe em [19], como seu próprio nome indica, permite determinar a potência minimima para conseguir a comunicação de um nó com cada um dos seus vizinhos, em outras palavras nos permite montar a tabela de vizinhança de cada um dos nós. O primeiro passo deste algoritmo é um nó enviar uma mensagem com a sua potência máxima (mensagem *hello*) para achar os seus vizinhos, se ele recebe uma mensagem resposta de outro sensor então este sensor passa a ser seu vizinho. O nó que iniciou a comunicação diminuirá então a potência de transmissão por um fator δ e repetira esse processo até não receber mais respostas. Assim a potência de transmissão para enxergar o nó em questão será a ultima com a qual uma mensagem resposta foi recebida. Pode se deduzir que quanto menor seja o fator de decrescimento da potência δ mais precisa será a potência mínima achada, em contrapartida o algoritmo demorara uma maior quantidade de tempo.

O algoritmo é descrito em detalhe a seguir:

Algoritmo DTNBOR

1. Cada um dos nós precisa construir sua tabela de vizinhança, a mesma deve ter os IDs (identificadores) dos vizinhos e as potências para chegar a eles. No estado inicial a tabela se encontra vazia.
2. Cada sensor *u* inicia um temporizador e envia uma mensagem (mensagem *Hello*) com o ID e sua tabela de vizinhança dele usando a potência máxima.
3. **Repete**
4. Se o nó *v* recebe a mensagem do sensor u, ele ira adicionar *u* na tabela de vizinhança dele e salvará a potência necessária para enxergar u. Imediatamente depois ira a enviar uma mensagem de resposta contendo o ID e a tabela de vizinhança dele.
5. Se o sensor *u* recebe uma mensagem de resposta de um sensor *v*, ele vai adicionar *v* na sua tabela e vai atualizar a potência de transmissão para alcançá-lo. Em seguida *u* vai reduzir o nível da potência de

transmissão por um fator δ, inicia um temporizador e envia uma mensagem com o ID e a tabela dele usando a potência já diminuída.

6. Se o sensor *u* não consegue ouvir *v* antes de o temporizador caducar, ele fará uso da potência atual salvada na sua tabela como a mínima potência de transmissão para se comunicar com *v*.
7. **Até que** não se tenham mais mudanças nas tabelas de vizinhança dos sensores.

A Figura 3.2 apresenta o diagrama de fluxo do algoritmo onde f representa o fator de decrescimento da potência δ.

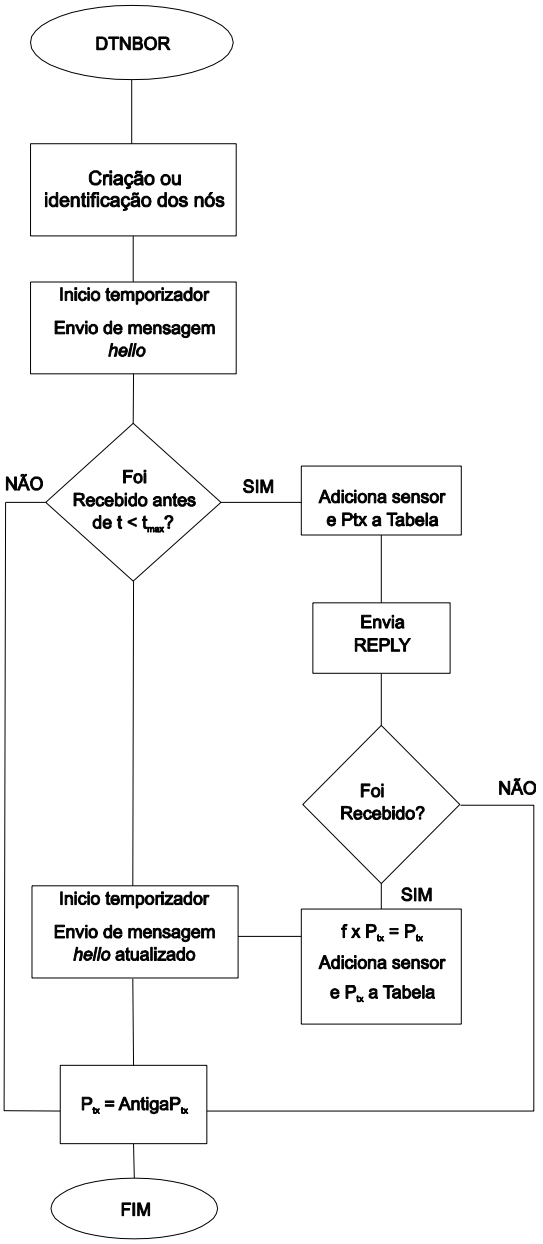

Figura 3.2: Diagrama de fluxo do algoritmo DTNBOR

Um dos primeiros detalhes a serem definidos para o desenvolvimento do código é o fator δ, como foi mostrado na seção 2.5.1 o chip de radio possui na sua folha de características [37] oito diferentes níveis de potência. No entanto como se vê na Figura 3.3 o simulador Avrora realiza uma interpolação desses valores para assim poder obter a relação entre os níveis e a potência em dBm para aqueles valores não especificados. Para este trabalho decidiu-se que o fator de decrescimento da potência será reduzir quatro níveis de potência cada vez que o nó receba uma mensagem RESPOSTA, ou seja cada vez que o código faça a atualização da potência de transmissão sendo que a potência máxima será o nível trinta e um. Esta decisão foi feita para trabalhar com os dados incluídos na folha de características e para que não se precise mexer no código quando ele for testado junto ao *hardware.*

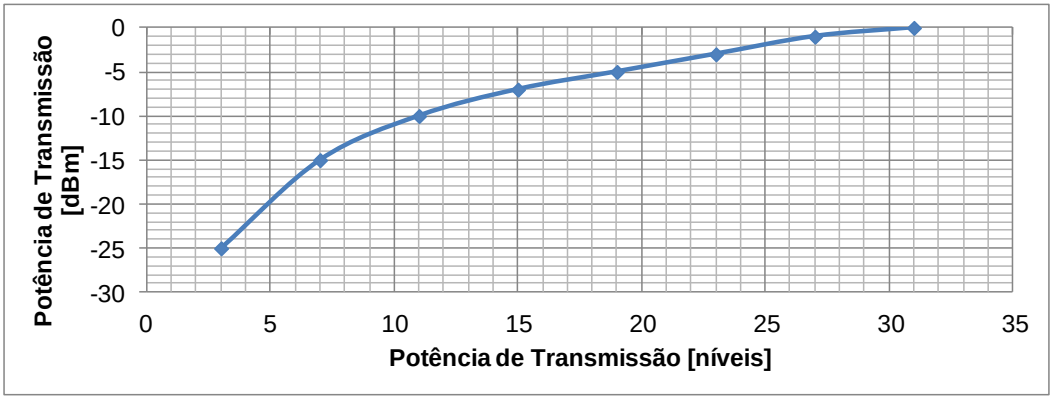

Figura 3.3: Interpolação do Avrora para níveis de potência. Fuente: AvroraZ

Na hora da implementação do código foram seguidos fielmente os passos descritos no algoritmo, o resultado foi um código que não realizava as funções propostas no documento [19], a descrição do algoritmo é bastante geral, não menciona alguns processos importantes para o correto funcionamento do mesmo. Portanto foi necessário introduzir modificações no algoritmo.

A primeira observação feita foi que além de iniciar um temporizador na terminal que envia a mensagem HELLO (o mesmo faz controle do tempo que o terminal aguarda uma resposta) era preciso iniciar um temporizador que faça o controle das primeiras respostas de todos aqueles que receberam a mensagem, este temporizador será descrito no decorrer desta seção.

A segunda modificação ataca um tema que não é considerado em nenhum momento na descrição do algoritmo, qual seja o tema da colisão de pacotes na hora do envio das mensagens denominadas RESPOSTA. Note o processo começa quando um nó envia uma mensagem *broadcast* para achar os vizinhos dele, depois todos aqueles que receberam uma mensagem *HELLO* tem que dar

enviar RESPOSTA para o nó que iniciou o procedimento. Neste momento múltiplos nós podem enviar a mensagem simultaneamente, ocorrendo o fenômeno conhecido como colisão, uma colisão significa perda da informação, esta perda fará que a tabela de vizinhança seja preenchida de maneira errada. Para solucionar esse problema se fez uso de um temporizador que permite inserir um atraso (*delay*) nas respostas dos nós.

A estrutura da mensagem enviada é diferente, e é composta de três campos; um número identificador do nó, a potência de transmissão para enxergar esse nó e uma variável de estado. Esta ultima pode tomar três valores, zero que é o valor de inicialização, um quando para indicar que adicionou aquele nó a sua tabela, dois para indicar que o algoritmo já achou a potência mínima para se comunicar com seu vizinho. A Figura 3.4 exibe o formato da mensagem e a quantidade de bytes que cada campo utiliza. Uma das diferenças desta versão do algoritmo respeito ao original é que a tabela de vizinhanças não é enviada em todas as mensagens como descreve o algoritmo. Esta decisão foi tomada por que os nós não precisam da tabela dos outros nós para formar a sua própria tabela de vizinhança e porque o armazenamento e envio da tabela com frequência consome recursos.

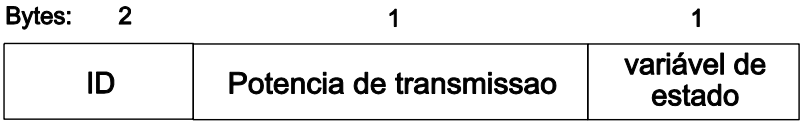

Figura 3.4: Trama da Mensagem *Hello* e Resposta

O algoritmo foi projetado para que cada nó tenha no máximo dez vizinhos. O armazenamento da tabela consome recursos na memória, sendo este um limitante do chip, dez é o número máximo de dados que a memória consegue armazenar e, portanto enviar, lembrando que a tabela de vizinhanças irá armazenar até dez vezes cada uma das variáveis da trama.

Foi necessário fazer um calculo para determinar o tempo que demora um nó em montar sua tabela com o número máximo de vizinhos, foi determinado ponto de partida em termos de tempo para que o seguinte nó na lista preencha a sua tabela. Deste modo foi possível automatizar o processo para todos os nós na rede.

O ponto de partida para este cálculo é saber quanto tempo demora o chip de radio CC2420 para enviar uma mensagem e para isto terá que ser abordado um conceito do sistema operacional. Como foi descrito anteriormente o TinyOS trabalha com muitas interfaces para as distintas tarefas a serem realizadas, a

saber: a interface *AMPacket* (o acrônimo AM se refere a *ActiveMessage)* tem como função outorgar acesso ao *buffer* do TinyOs, o *message_t,* ele possui os dados ou o pacote, na descrição técnica desta interface [38] se conseguiu achar o tamanho em bytes da trama, a carga útil (*payload*) ocupa 28 bytes, as cabeçalhos (headers) para o chip CC2420 são de 11 bytes, as informações de endereço e de controle ocupam cerca 11 bytes. Assim uma mensagem no TinyOS num cenário normal teria ao redor de 50 bytes. Com este dato e o conhecimento de que a taxa de transferência do Micaz é de 250kbps, conseguiu se obter um valor estimado para o tempo que uma mensagem enviada por uma terminal A leva para chegar a uma terminal B. Calculou se que um bit é enviado em quarenta microssegundos e portanto uma mensagem de 50 bytes é enviada em 16 milissegundos. Por questões de segurança escolheu se trabalhar com um valor de cinquenta milissegundos para ter uma margem contra qualquer imprevisto principalmente pelas características do simulador e tempos de processamento diversos. No desenvolvimento do algoritmo foram utilizados quatro temporizadores, o primeiro controla o tempo que o nó que envia uma mensagem *HELLO* (nó fonte) aguarda pela primeira RESPOSTA dos seus vizinhos, lembrando que o nosso desenvolvimento foi feito para que cada nó tenha como máximo dez vizinhos, a equação deste temporizador seria a seguinte:

$$T_{\mathrm{P}resposta} = MAX\_NBORS \times T_{resposta} \quad (1)$$

Onde *sendTime* são os cinquenta milissegundos acima descritos, $T_{resposta}$ é o tempo que o nó espera uma mensagem seja uma mensagem *HELLO* ou uma RESPOSTA e MAX_NBORS é o número máximo de vizinhos. A justificativa do valor do temporizador da primeira resposta é que mesmo que o nó fonte consiga achar apenas cinco vizinhos com a sua potência máxima, ele deve esperar o tempo de dez respostas já que por tratar se de um processo de descobrimento o nó não sabe quantos vizinhos vai ter por isso deve ficar aguardando pela máxima quantidade de respostas possíveis.

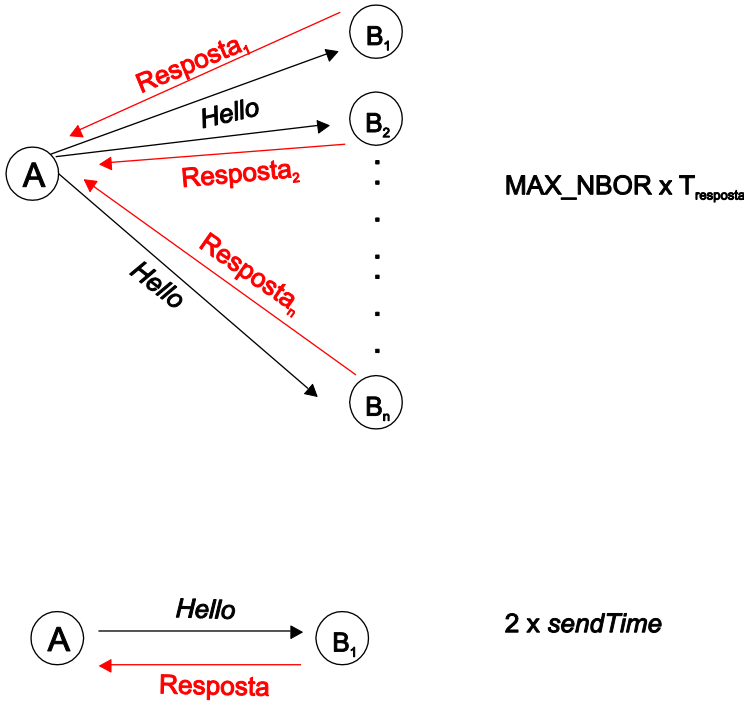

Figura 3.5: Descrição dos temporizadores $T_{Presposta}$ e $T_{resposta}$

O temporizador $T_{resposta}$ tem o valor de duas vezes o valor de um envio:

$$T_{resposta} = 2 \times sendTime \quad (2)$$

Na Figura 3.5 é esclarecido o funcionamento dos dois temporizadores descritos até o momento.

O terceiro temporizador é utilizado para evitar colisões, e pode ser visto como a introdução de uma defasagem entre as respostas e às mensagens de HELLO, sendo escolhido um valor maior ao tempo do envio de uma mensagem, pelo qual:

$$T_d = 2 \times sendTime \quad (3)$$

A partir dos valores desses três temporizadores pode ser feito o calculo do ultimo temporizador, que controla o tempo no qual um nó consegue preencher sua tabela de vizinhança, e que pode ser expresso por:

$$T_{ciclo} = T_{Presposta} + \left(8 \times T_{resposta}\right) 2 + T_d \quad (4)$$

Este cálculo considera o pior caso no qual o nó enxergara seu vizinho com a potência mínima, e portanto serão trocadas dezesseis mensagens (dois para cada nível de potência).

O algoritmo foi reescrito com as modificações feitas para adequá-lo ao TinyOS, o algoritmo modificado foi batizado como TinyDTNBOR

Algoritmo TinyDTNBOR (DTNBOR adequado a TinyOS)

1. Cada um dos nós precisa construir sua tabela de vizinhança, a mesma deve ter os IDs (identificadores) dos vizinhos, as potências para chegar a eles e uma variável de estado que permita identificar a etapa na qual se encontra o algoritmo. No estado inicial a tabela se encontra vazia.
2. É iniciado um temporizador T_{ciclo} que indica o tempo que um nó demora em preencher sua tabela com todos os vizinhos possíveis.
3. Cada sensor *u* inicia um temporizador $T_{Presposta}$ e envia uma mensagem (mensagem *Hello*) com o ID dele usando a potência máxima e a variável de estado zerada.
4. Se o nó *v* recebe a mensagem do sensor *u* imediatamente ira a enviar uma mensagem de resposta contendo o ID, a potência que é a mesma com a que o nó *u* enxergou *v* e a variável de estado ainda zerada.
5. Se o sensor *u* recebe uma mensagem de resposta de um sensor *v*, ele vai adicionar *v* na tabela dele, vai atualizar a potência de transmissão para alcançá-lo e outorgar um valor de um a variável de estado para indicar que ele já é considerado vizinho.
6. Caso haja mais de um sensor vizinho as respostas deles serão desfadas por um intervalo T_d.
7. O sensor *u* ficara aguardando pela primeira resposta dos seus vizinhos v_n até o $T_{Presposta}$ esgotar seu tempo.
8. **Repete para cada um dos vizinhos**
9. Em seguida *u* vai reduzir a potência de transmissão em quatro níveis para se comunicar com *v*, inicia um temporizador $T_{resposta}$ e envia uma mensagem usando a potência já diminuída.
10. Se o nó *v* recebe a mensagem do sensor *u* imediatamente ira a enviar uma mensagem de resposta contendo o ID, a potência que é a mesma com a que o nó *u* enxergou *v* e a variável de estado igual a um.
11. Se o sensor *u* não consegue ouvir *v* antes do temporizador $T_{resposta}$ caducar, ele fará uso da potência atual salva na sua tabela como a mínima potência de transmissão para se comunicar com *v* e fará o valor da variável de estado igual a 2, indicando assim que a potência mínima foi achada para o vizinho em questão.
12. **Continua até que** não se tenham mais mudanças nas tabelas de vizinhança dos sensores.

A Figura 3.6 apresenta o diagrama de fluxo do algoritmo TinyDTNBOR. Na Figura estão se ressaltam com a cor vermelho as caixas de novos

processos ou que tiveram mudanças em relação ao algoritmo DTNBOR original.

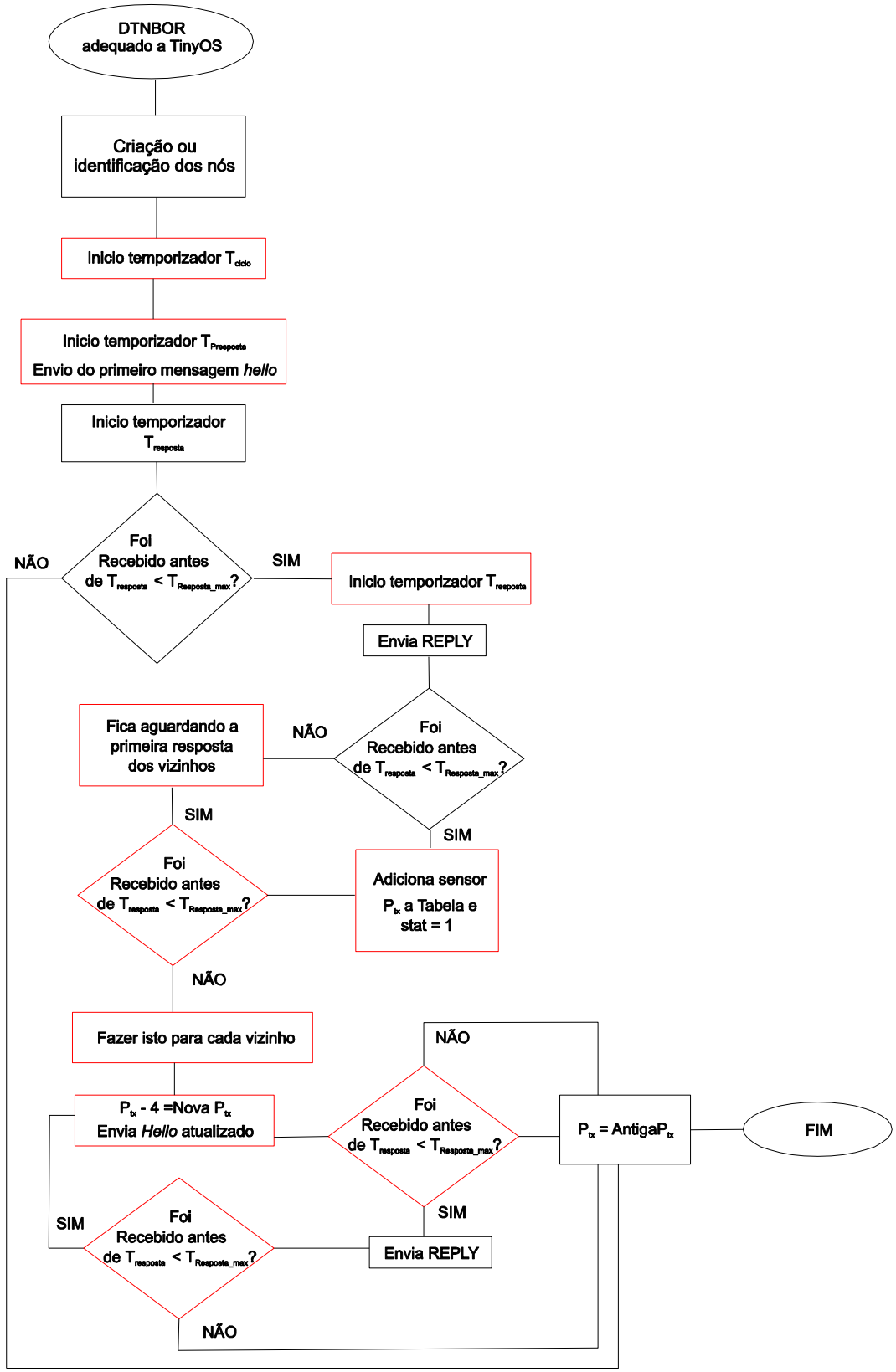

Figura 3.6: Diagrama de fluxo do algoritmo DTNBOR adequado a TinyOS

3.2.2
DTRNG

Baseado no RNG (*Relative Neighborhood Graph*), o DTRNG (*Determine transmission power using RNG*) busca determinar a potência de transmissão fazendo uso desta técnica, como foi mencionado na Seção 2.3.1 o RNG visa eliminar um caso especial dos ciclos que se formam quando uma rede é modelada como um grafo de comunicação para assim obter uma topologia que tenha maior simplicidade em termos de conexões(arestas). Esta técnica tem relação direta com as distancias entre os nós (representadas por arestas no grafo), já que busca eliminar a maior distancia entre os vértices ou nós que formam um triângulo, por tanto aquela aresta no grafo que una os dois pontos mais distantes.

Chen e Rowe [19] propõem outro enfoque para o RNG baseando se na relação existente entre potências e distâncias, ou seja, quanto maior for a potência de transmissão maior será o alcance do nó. Portanto, eliminando as arestas de maior comprimento estarão sendo eliminadas comunicações que fazem uso de potências altas. Para isto eles propõem a seguinte equação:

$$p_{|uw|} \leq p_{|uv|} \,\&\, p_{|vw|} \leq p_{|uv|} \tag{5}$$

Onde *u, v* e *w* são três nós e *p* representa a potência de transmissão, portanto p_{uw} representa a potência de transmissão que o nó *u* precisa para enxergar o nó w e assim respectivamente.

Este enfoque é apresentado graficamente na Figura 3.7. Em 3.7.a se tem três nós representados num grafo, neste também pode se observar as respectivas potências de transmissão que cada nó precisa para estabelecer comunicação com seu vizinho. Reemprazando as potências na equação 6 temos que:

$$19 \leq 27 \,\&\, 15 \leq 27$$

A condição é cumprida pelo qual *u* eliminara *v* da sua tabela de vizinhança, isto é evidenciado Na Figura 3.7.b

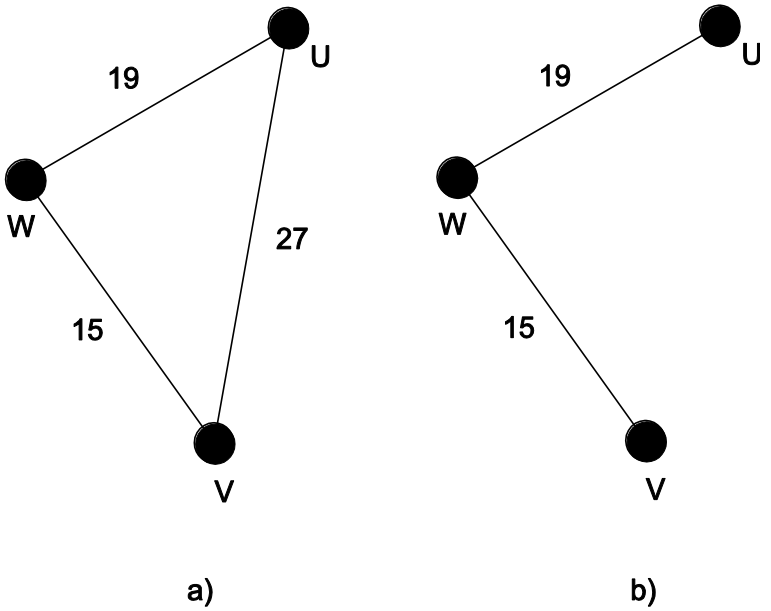

Figura 3.7: Exemplo da equação 5

Com a finalidade de ter uma solução que trabalhe com potências mínimas este algoritmo faz uso do algoritmo DTNBOR.

A seguir será apresentado o algoritmo descrito DTRNG:

Algoritmo DTRNG

1. Cada um dos nós chama o algoritmo DTNBOR para determinar a mínima potência de transmissão para enxergar seus vizinhos
2. **Repete**
3. Cada sensor *u* faz a validação da equação (5):

$$p_{|uw|} \leq p_{|uv|} \,\&\, p_{|vw|} \leq p_{|uv|}$$

4. Se for certo o sensor *u* vai tirar o id de *v* e a potência para enxergar ele da sua tabela de vizinhança.
5. **Ate que** não existam maiores remoções.
6. Cada um dos sensores vai usar a maior potência de transmissão da sua tabela como a sua potência de transmissão.

W representa o nó intermediário entre *u* e *v,* a Figura 3.8 apresenta o esquema do presente algoritmo.

Fazendo um analise da condição apresentada no ponto três do algoritmo podemos ver que precisamos fazer comparações entre três potências, neste caso a potência de *u* a *w* e de *u* a *v*, podem ser achadas na tabela de *u*, mas para ter conhecimento da potência entre os nós *v* e *w* é necessário ter acesso a tabela dos seus vizinhos, em outras palavras cada nó tem que ter informações locais da rede.

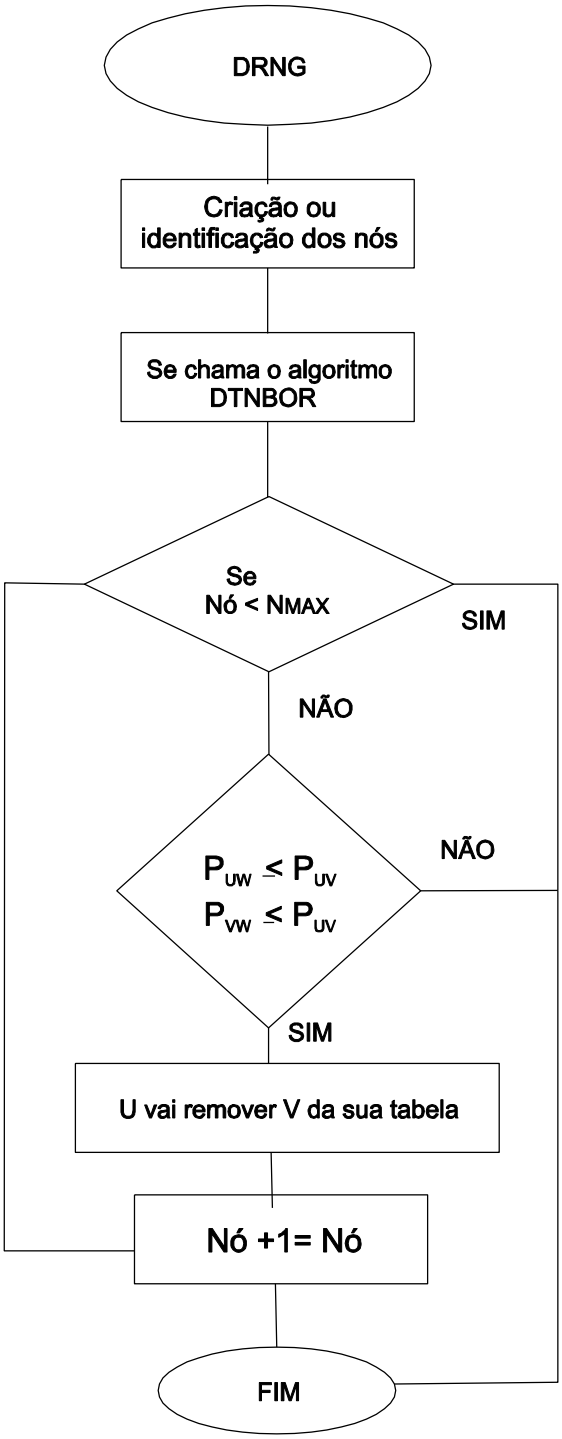

Figura 3.8: Diagrama de fluxo do algoritmo DTRNG

Pela razão descrita o primeiro passo é fazer que cada nó envie sua tabela de vizinhança a todos seus vizinhos, isto é possível devido a quantidade de nós na rede já que se temos redes com um número elevado de dispositivos, por exemplo mil, a complexidade do sistema será elevada. Escolheu-se criar temporizadores para fazer o controle deste envio de tabelas e assim evitar colisões e colocar uma ordenação na recepção das tabelas. O primeiro temporizador evita os envios simultâneos fazendo que cada nó envie um por vez

a sua tabela, para que o segundo nó inicie o seu envio terá que aguardar o tempo de dez envios (máximo de tabelas possíveis), e assim sucessivamente. No final deste processo cada nó tem uma matriz na qual encontra as tabelas de todos seus vizinhos, na Figura 3.9 é ilustrado graficamente este processo. Nela temos três nós, os mesmos já têm as suas tabelas de vizinhança preenchidas com o identificador do nó e a respectiva potência de transmissão, A e B já fizeram o envio das suas tabelas um a cada vez. Pode se observar no nó C da Figura 3.9 a estrutura da tabela de informações locais, a mesma indica na primeira coluna de quem são os dados armazenados, é assim que C sabe que A para se comunicar com B utiliza o nível de potência quinze.

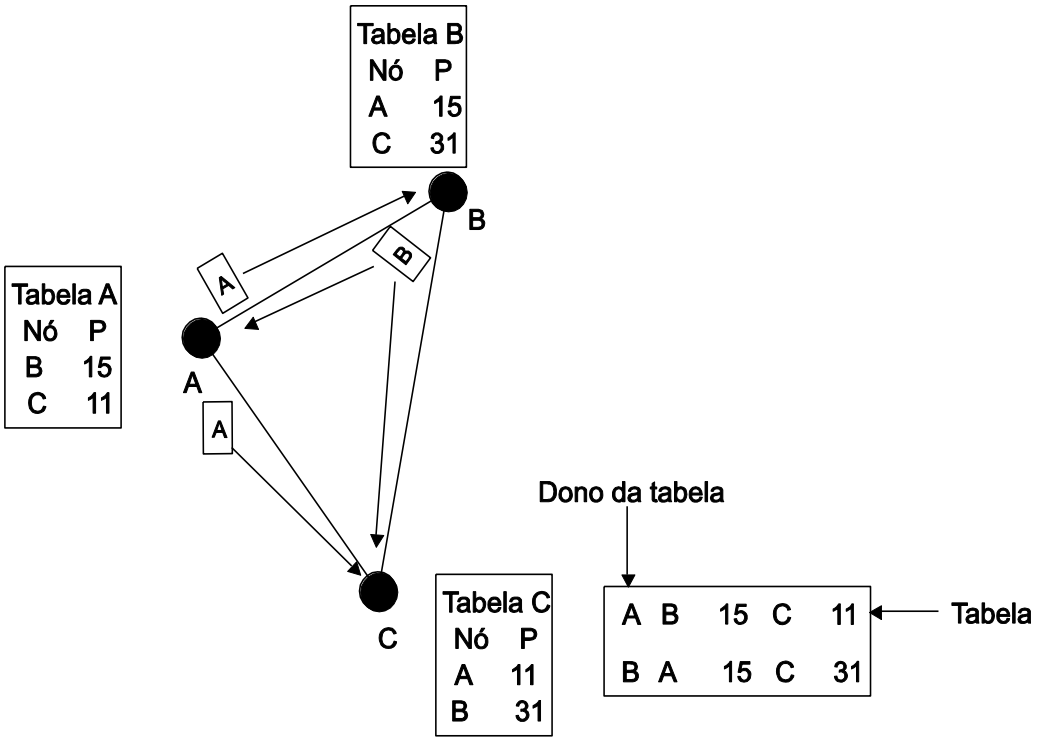

Figura 3.9: Montagem da tabela de informações locais

O segundo temporizador informa o fim deste processo para a seguir fazer a comparação entre as potências.

Na hora de fazer a comparação entre as potências foi necessário achar um critério para a escolha das potências a ser comparadas. O processo deve começar pelo primeiro nó, ele será denominado *u,* o primeiro vizinho dele passara a ser *v* e o segundo será *w*, faltando agora a potência necessária entre *v* e *w* e que deve ser achada na tabela de informações locais, na coluna que indica quem é o dono da tabela tem que se achar o número identificador de *w*, uma vez que ele for encontrado, nessa linha deve se achar o número identificador de *v,* obtendo a potência procurada. Todos os procedimentos e conceitos descritos nesta Seção estão ilustrados na Figura 3.10.

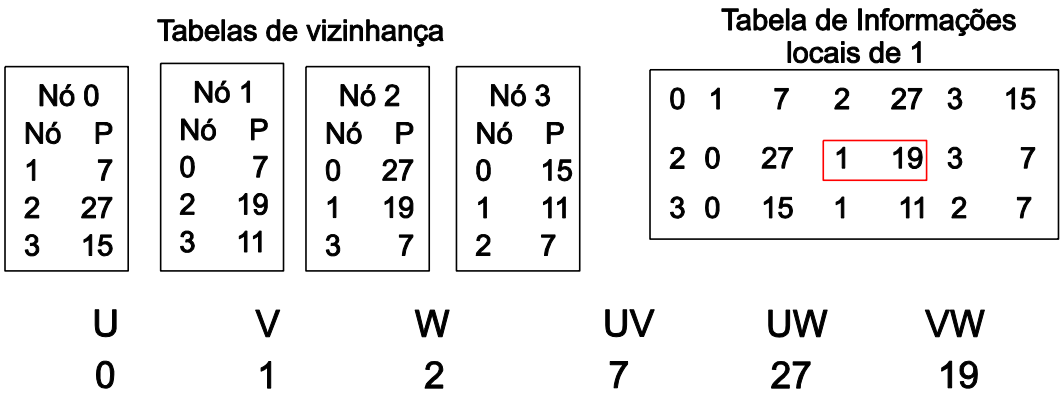

Figura 3.10: Exemplo de escolha de potências para condição do DTRNG

Na Figura 3.10 se tem um exemplo de como é feita a procura de potências para achar e eliminar todos os ciclos da rede, na figura também são apresentadas as tabelas de quatro nós e na parte de baixo da Figura são mostradas as potências de acordo com os critérios já descritos.

Substituindo os valores encontrados na equação do algoritmo teremos que:

$$27 \leq 7 \,\&\, 19 \leq 7$$

A condição não é satisfeita e, portanto não são feitas mudanças na tabela. O algoritmo continua até este processo ser repetido para todos os nós e para todos os vizinhos, por exemplo, na seguinte rodada *u* continuara sendo nó zero e *v* será o nó três.

A seguir o algoritmo é reescrito incorporando as modificações realizadas:

Algoritmo TinyDTRNG (DTRNG)adequado a TinyOS

1. Cada um dos nós chama o algoritmo DTNBOR para determinar a mínima potência de transmissão para enxergar seus vizinhos
2. **Repete**
3. Cada nó envia a sua tabela de vizinhança a todos os seus vizinhos.
4. È iniciado um temporizador de controle para evitar envios simultâneos
5. **Até que** todos os nós tenham recebido a tabela dos seus vizinhos.
6. **Repete**
7. Cada sensor determina que nós serão *u,v* e w.
8. Cada sensor *u* deve encontrar os valores de p_{uw}, p_{uv} na sua tabela.
9. O sensor *u* deve procurar a potência p_{vw} na tabela de informações locais.
10. Cada sensor *u* faz a validação da equação (5):

$$p_{|uw|} \leq p_{|uv|} \text{ e } p_{|vw|} \leq p_{|uv|}$$

11. Se for certo o sensor *u* vai tirar o número identificador de *v* e a potência para enxergar ele da sua tabela de vizinhança.

12. Até que tenham sido testados todos os nós e todos seus respectivos vizinhos.

O algoritmo continua até ter feito este processo para todos os nós e para todos os vizinhos, por exemplo, na seguinte rodada *u* continuara sendo nó zero e *v* será o nó três.

A seguir, na Figura 3.11, é apresentado o diagrama de fluxo do DTRNG adequado ao sistema operacional, sendo que os blocos que possuem modificações respeito ao algoritmo original foram marcados em vermelho.

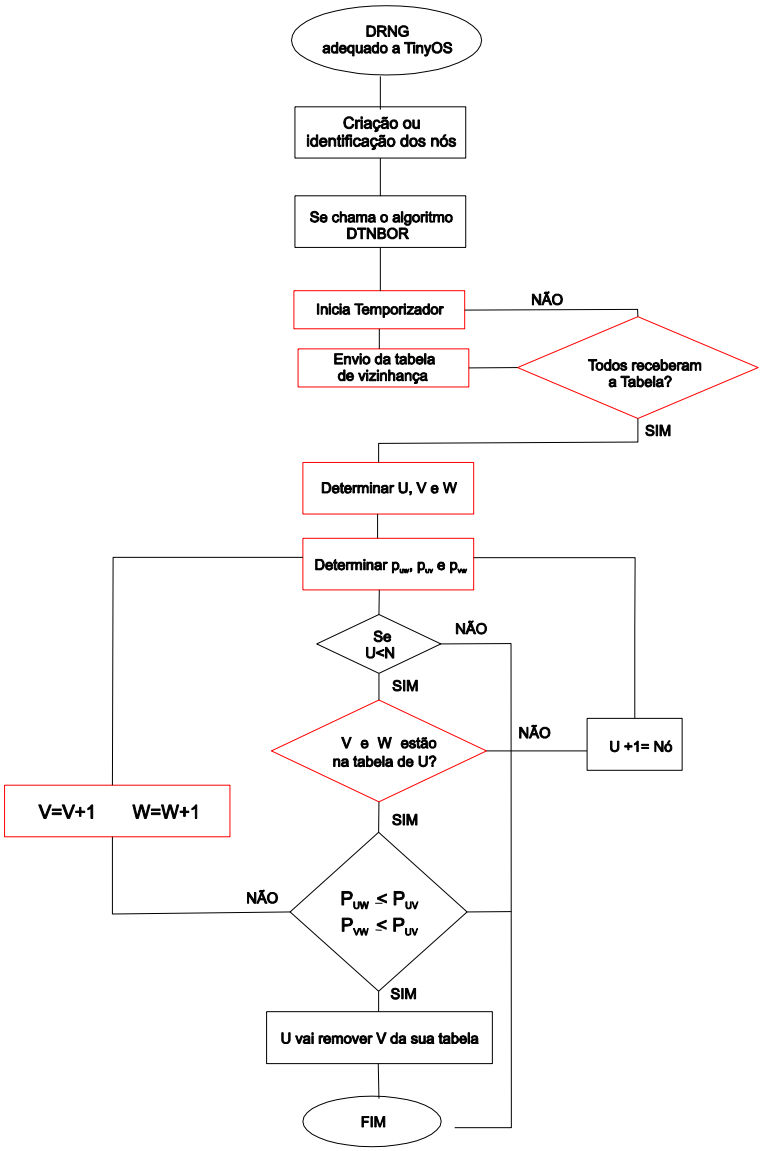

Figura 3.11: Diagrama de fluxo do algoritmo DTRNG adequado a TinyOS

3.2.3
Protocolo de encaminhamento

Devido às características dos algoritmos trabalhados, eles só oferecem como resultado uma tabela de vizinhança (DTNBOR) e uma tabela depurada (DTRNG) foi necessário desenvolver um algoritmo que permita avaliar o funcionamento destes algoritmos em distintas condições sempre levando em consideração a natureza das redes de sensores sem fio. A concepção deste aplicativo foi feita levando em consideração condições mais próximas possíveis de situações reais. A maioria das aplicações das RSSF consiste em monitorar fenômenos e posteriormente encaminhar certo tipo de informação para um nó coordenador ou para uma estação base. É esta a configuração com a qual se decidiu testar os algoritmos. Foi desenvolvido um protocolo que permitiu fazer uma agregação de dados que embora seja uma aplicação simples é muito útil na pratica.

O protocolo consta de duas fases, a primeira foi definida como descoberta de rota e a segunda é a fase de coleta de dados. Na primeira fase será definido o caminho no qual serão coletados os dados, para isto cada nó é responsável de enviar uma mensagem do tipo *broadcast* tentando se comunicar com seus vizinhos, onde os nós armazenam e utilizam a informação da primeira mensagem recebida e descartam todas as subsequentes. O nó que recebeu essa primeira mensagem em primeira instância deve salvar o número identificador do emissor da mensagem e imediatamente deve propagar a mesma, o processo deve se repetir até que não se propaguem mais mensagens pela rede. O nó que recebeu uma mensagem designará de “pai” ao emissor da mesma, assim para qualquer tipo de configuração todo nó terá um “pai” conseguindo assim garantir que a rede sempre fique conectada.

A segunda fase do protocolo consta em que todos os nós enviem a informação requerida ao nó de maior hierarquia (aquele que enviou a primeira mensagem) assim cada nó enviara sua mensagem e qualquer outra que receba a aquele que classificou de “pai”, conseguindo assim fazer o encaminhamento até o nó fonte ou raiz.

As aplicações de coleta de dados por ser parte de processos de monitoramento geralmente coletam informações em certos intervalos de tempo, ou seja um nó envia informações periodicamente. O protocolo desenvolvido realiza essa função através de um temporizador periódico podendo se configurar os intervalos de tempo e a quantidade de vezes que um nó fará um envio.

No projeto desta segunda fase também foram considerados dois critérios para evitar perdas de mensagens. Primeiramente foi programado um sistema de filas, o mesmo pode ser definido como pacotes chegando a um nó que atua como roteador e ficam aguardando seu turno para ser encaminhados. Isto é apresentado na Figura 3.12, nesta se tem uma disposição de nós querendo encaminhar suas informações para o nó A que é "pai" do nó B que a sua vez é "pai" de C_1, C_2 até C_n nós. Para C_1, C_2 até C_n encaminharem suas informações até A precisam enviar suas respostas a seu "pai", neste caso o nó B, que vai possuir uma fila na qual serão inseridas todas as mensagens que o nó receber, e ira fazendo o envio destas mensagens a seu "pai" nos seus respectivos turnos. Todos os nós tem um sistema de fila.

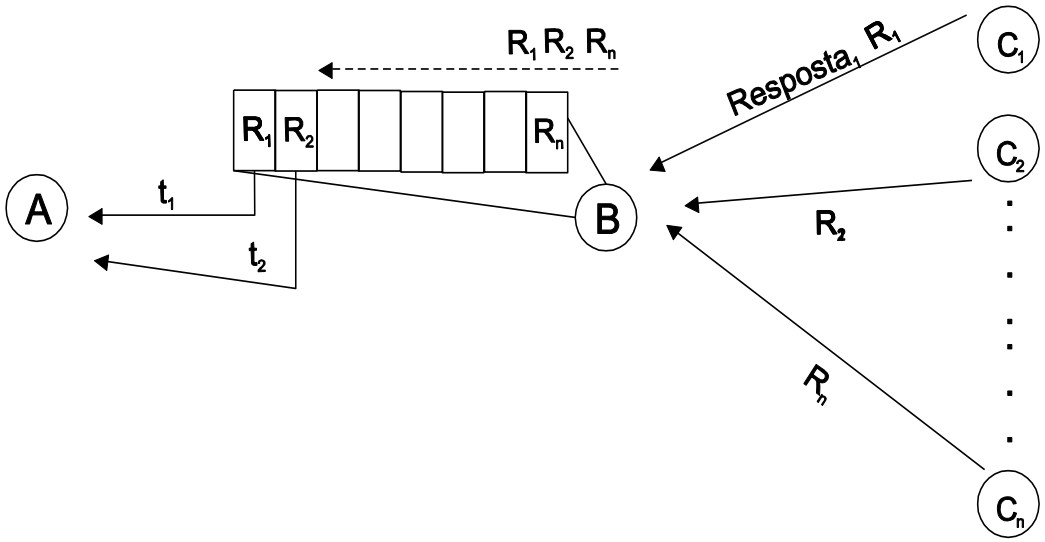

Figura 3.12: Descrição da fila implementada

O segundo critério consta em que cada nó depois de receber uma mensagem envie uma mensagem de controle (*acknowledge*) ao emissor da mesma, caso o nó emissor não receba esta mensagem de controle acontecerá o reenvio da mensagem original. A fila pode armazenar até vinte mensagens e o número máximo de reenvios possíveis é três.

A seguir será mostrado o pseudocódigo do protocolo:

Protocolo de encaminhamento

Fase de descoberta de rota

1. É escolhido o nó fonte, ele procede a enviar uma mensagem com sua potência máxima para ver quem enxerga ele.
2. **Repete**

3. O nó v_n recebe uma mensagem, se for a primeira mensagem recebida armazena o número identificador do emissor da mensagem recebida, e reenvia a mesma caso contrario a mensagem é rejeitada.
4. **Até que** não se propaguem mais mensagens pela rede.

Fase de coleta de dados

1. São iniciados dois temporizadores, um para controlar a quantidade de envios e o outro é um temporizador periódico que controla o tempo no qual se fará o envio cíclico das mensagens.
2. **Repete**
3. O nó v_n enviara uma mensagem ao seu "pai" e ficara aguardando uma resposta que confirme o recebimento da mesma
4. Se o nó v_n não recebe a confirmação de recebimento do "pai", ele procede a fazer o reenvio.
5. Se o nó v_{n+1} recebe a mensagem ele introduz a mesma em sua fila, se não houver outra mensagem na frente ela será retransmitida imediatamente, caso contrario devera aguardar seu turno para ser retirada da fila.
6. **Até que** o nó fonte tenha recebido as mensagens de todos os nós da rede.

Por ultimo será apresentado na Figura 3.13 o diagrama de fluxo representando para o funcionamento do protocolo de encaminhamento. No lado esquerdo pode se observar a fase de descoberta de rota e no direito a fase de coleta de dados.

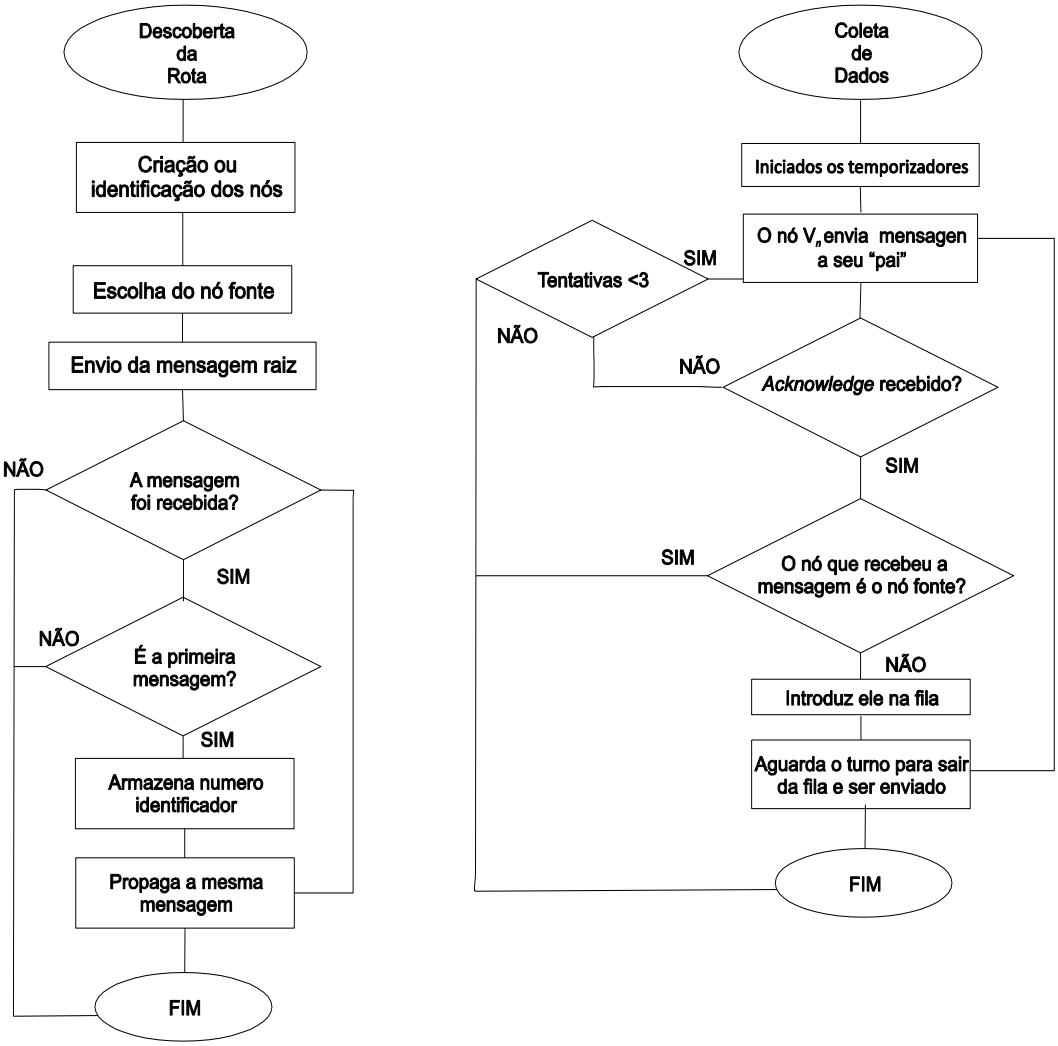

Figura 3.13: Diagrama do protocolo de encaminhamento

Neste Capítulo foram apresentadas as características e o funcionamento dos algoritmos de eficiência energética implementados no trabalho, também foram detalhadas as modificações feitas neles para adequar eles ao entorno de simulação. No Capítulo 4 são conduzidos uma série de testes para avaliar o consumo energético na transmissão dos mesmos.