Um Algoritmo Tolerante a Falhas e de Baixa Latência para Redes de Sensores sem Fio

Richard Werner Nelem Pazzi[1], Regina B. Araujo[1] e Azzedine Boukerche[2]

***Resumo*—Este artigo descreve um algoritmo tolerante a falhas e de baixa latência que atende os requisitos de aplicações de monitoramento de situações críticas. O algoritmo PEQ utiliza o mecanismo de publicação/ subscrição e o conceito de *entrega dirigida de eventos,* uma técnica que seleciona o caminho mais rápido para a notificação de eventos, reduzindo a latência. A tolerância a falhas é obtida, através de rápida re-configuração da rede, a qual passa do modo de caminho mais rápido para o modo multi-caminho de notificação confiável de eventos.**

***Palavras-Chave*—Redes de sensores sem fio, monitoramento de condições críticas, tolerância a falhas, baixa latência, economia de energia.**

***Abstract*—This paper presents a novel fault tolerant and low latency algorithm that meets sensors network requirements for critical conditions surveillance applications. The algorithm uses the publish/subscribe mechanism and the concept of *driven delivery of events,* a technique that selects the fastest path for the notification of event, reducing latency. Fault-tolerance is achieved through fast reconfiguration of the network, which switches from a shortest path mode to a multi-path reliable event notification mode.**

***Index Terms*—Wireless sensor networks, critical conditions surveillance, fault tolerance, low latency, energy savings.**

I. INTRODUÇÃO

Com os recentes desenvolvimentos das redes sem fio e de sensores multifuncionais com capacidades de processamento digital, fonte de energia e de comunicação, as redes de sensores sem fio estão sendo amplamente empregadas em ambientes físicos para monitoramento de granularidade fina em diferentes classes de aplicações. Uma das aplicações de maior apelo é a de supervisão de segurança e monitoramento de condições críticas. Em uma prisão, por exemplo, é importante manter um monitoramento contínuo e confiável do ambiente físico, especialmente na emergência de situações críticas, tais como uma rebelião de presos que pode levar a situações de incêndio ou explosão.

Em tais circunstâncias é importante que a informação seja "sentida" do ambiente físico enquanto o estado de emergência estiver em progresso, uma vez que informações mais precisas podem ser usadas pelas equipes de resgate e combate ao fogo para melhor gerenciamento da operação, através de tomadas de decisão mais eficientes, minimizando assim eventuais perdas de vidas e de patrimônio. Entretanto, de modo a manter o fluxo de informações provenientes dos sensores durante o estado de emergência, uma solução de rede de sensores sem fio tem que lidar com a eventual falha de nós sensores (sensores podem ser queimados, sofrer interferências que comprometem o envio de informações, como água ou presença de fumaça densa, ou ainda apresentarem defeitos). Assim, soluções de redes de sensores para tais ambientes têm que ser tolerante a falhas e confiáveis na entrega de eventos, além de oferecer baixa latência, rápida re-configuração e economia de energia para o aumento da vida útil da rede. Em termos de economia de energia, no estado de monitoramento, os sensores podem ser programados para notificar eventos de forma periódica (enviar temperatura a cada 10 minutos) ou orientada a eventos (enviar temperatura apenas quando esta for maior 60ºC). Nestes casos, o interesse pode não mudar por algum tempo. Algumas soluções existentes para economia de energia levam isso em consideração e desligam alguns nós, deixando-os em estado inativo – os nós são reativados apenas quando o interesse coincide com os eventos capturados do ambiente [1] [2]. Por outro lado, em cenários de aplicação baseados em consulta *(query-based)*, estas podem ser propagadas para os nós sensores arbitrariamente, de acordo com a aplicação e/ou vontade do usuário e, assim, algumas soluções que tratam da economia de energia podem não ser adequadas uma vez que a transição do estado inativo para o estado de transferência de dados (ativo) pode ter alto custo em termos de energia quando várias transições arbitrárias forem necessárias [2]. Mais ainda, economia de energia e tolerância a falhas podem apresentar interesses conflitantes quando novos caminhos, envolvendo nós inativos, tiverem que ser rapidamente ativados devido a falhas de nós em caminhos estabelecidos anteriormente. Uma solução que possa atender simultaneamente os três tipos de cenário descritos acima, oferecendo notificação de eventos com baixa latência, rápida re-configuração na presença de caminhos de entrega de evento quebrados, múltiplas rotas alternativas e economia de energia, é desconhecida dos autores. Este artigo descreve o algoritmo PEQ (*Periodic Event-driven and Query-based*) para monitoramento de condições críticas em ambientes físicos. A baixa latência é obtida através de um caminho mais curto para a entrega de eventos. Subscrições rápidas de novos interesses (para cenários baseados em consultas) são atendidas através do conceito de entrega dirigida de eventos, na qual novas subscrições para os sensores são agilizadas

[1]DC - Universidade Federal de São Carlos, CP 676, 13565-905 São Carlos, SP, Brazil, {richard,regina@dc.ufscar.br}
[2]SITE – University of Ottawa, Ottawa, Canada, boukerch@site.uottawa.ca

usando-se o caminho inverso utilizado para notificações de eventos. A tolerância a falhas é controlada pelo nó destino (um ou mais *sinks*) - quando o *sink* nota que eventos anormais estão sendo notificados e que alguns nós podem ter sido destruídos, ele ativa a entrega multi-caminhos, uma re-configuração da rede que altera os nós sensores para um modo que pode tratar múltiplos caminhos. No algoritmo PEQ aqui descrito, o paradigma da publicação/subscrição é utilizado para promover a interação entre os sensores e o *sink.* A rede de sensores é configurada através de uma árvore de saltos *(hop tree)*, que é construída no momento da configuração da rede. Subscrições para os nós são propagadas para os sensores, através da árvore de saltos criada. De modo a melhor descrever o algoritmo, um modelo de grade é usado. Entretanto, a solução pode ser aplicada para redes em malha *(mesh)* e redes densas, em que os sensores são colocados arbitrariamente.

II. Dinâmica de Roteamento

O algoritmo de roteamento aqui descrito é realizado em três etapas. A primeira etapa compreende a construção da árvore de hops. O sink inicia o processo de construção da árvore de hops, que será utilizada como um mecanismo de propagação de mensagens de configuração e de subscrição na rede. A segunda etapa envolve a propagação de subscrições na rede. O sink se subscreve na rede para receber informações (eventos) de seus nós. Finalmente, a última etapa é responsável pela entrega de eventos dos nós ao sink, utilizando-se do caminho mais curto e de menor custo, em termos de economia de energia. As etapas de roteamento são descritas em detalhes nas próximas seções. É assumido que os nós estão dispostos em um grid de tal forma que a cobertura de transmissão de um nó é capaz de alcançar os oito nós vizinhos. Porém, a solução pode também ser aplicada em redes de sensores onde os nós foram lançados randomicamente no ambiente, como mostrado ao longo do texto.

A. *Construção da Árvore de Hops*

Na rede de sensores sem fio considerada aqui, um nó não possui uma compreensão global da rede, ou seja, o nó possui apenas alguma informação sobre seus vizinhos mais próximos (aqueles que estão dentro de seu raio de cobertura). Em um primeiro instante, cada nó conhecerá apenas em que nível de hop ele está na árvore. A árvore é iniciada por um sink, o qual transmite para seu(s) vizinho(s) um par atributo-valor denominado *hop*. O algoritmo para construção da árvore baseia-se na inundação da rede, iniciando um valor para hop no sink, que incrementa seu valor e o transmite para seus nós vizinhos, que armazenam esse atributo, incrementam-no e retransmitem para seus vizinhos, e assim sucessivamente, até que toda a rede esteja configurada com diferentes níveis de hops. Como a comunicação entre os nós da rede é realizada através de sinais de rádio (RF), todos os nós vizinhos recebem a transmissão. Dessa forma, um nó que acabou de transmitir uma mensagem, pode receber a mesma mensagem de seu vizinho, gerando um loop. Para evitar essas transmissões inúteis e que geram gasto de energia, um conjunto de regras foi estabelecido como parte do algoritmo de difusão de hop. Uma das regras locais estabelece que cada nó, ao receber uma transmissão com o hop, compara o hop recebido com o seu hop local. Se o valor do hop local for maior que o valor recebido, o nó atualiza seu hop, incrementa seu valor e o retransmite para seus vizinhos. No caso do hop armazenado no nó ser menor ou igual ao hop recebido, o nó não atualiza seu hop e não transmite. Dessa forma, a rede é configurada de tal forma que cada nó sabe apenas o nível de hop no qual ele está situado.

O algoritmo de configuração inicial é mostrado na figura 1. A estrutura de dados do algoritmo compreende três tabelas: configTable, routingTable e subscriptionTable. A tabela configTable possui os parâmetros de configuração associados aos sinks. A tabela routingTable é usada por um nó para encaminhar mensagens para seus nós vizinhos. Finalmente, a tabela subscriptionTable é usada para armazenar subscrições que um nó recebe.

```
config.sendConfigMsg();
// When a node receives a configuration message,
// it checks its configTable to find a match.
entry = configTable.get(config.sinkID);
if (entry)  // Entry exists?
{
    if (entry.hop > config.hop)
    {
       entry.hop = config.hop;
        config.hop=config.hop+1;
        config.sendConfigMsg();
    }
}
else  // Entry does not exist!!
{
    entry.sinkID = config.sinkID;
    entry.hop = config.hop;
    configTable.add(entry);
    config.hop = config.hop + 1;
    config.sendConfigMsg();
}
```

Fig. 1. Algoritmo de configuração inicial.

B. *Propagação da Mensagem de Subscrição*

No paradigma *publish/subscribe* [3], para um sink ser notificado sobre eventos capturados do ambiente físico pelos nós sensores, ele precisa subscrever em um ou mais nós, através da critérios (temperatura > 60°C, presença de fumaça, etc) que precisam ser comparados antes de um evento ser enviado. O envio de eventos apenas quando eles satisfazem algum critério reduz o tráfego na rede, causando um menor gasto de energia e estendendo a vida da rede. Após a configuração inicial da rede, a única informação que um nó possui é o nível de hop no qual ele está. Apenas esta informação não é suficiente para uma eficiente propagação de subscrição. Na ausência de alguma informação sobre qual nó da rede possa satisfazer o interesse do sink, uma maneira de propagar a subscrição inicial para o nó correto é inundar a rede com o interesse. Devido a este fato, uma mensagem de subscrição inicial pode ser enviada na etapa de configuração da rede.

Cada nó na rede mantém uma pequena tabela de subscrições e outra de roteamento. Cada registro na tabela de subscrições representa uma subscrição distinta. Durante a propagação da mensagem de subscrição, cada nó, ao recebe-la, compara o atributo coord da subscrição com suas próprias coordenadas. Se forem iguais, a subscrição é destinada para este nó e então é armazenada em sua tabela de subscrições. Caso contrário, o nó apenas retransmite a subscrição como parte do algoritmo descrito na figura 7.

C. *Envio da Mensagem de Notificação*

Quando a informação é capturada do ambiente físico por um nó sensor, este verifica sua tabela de subscrições para determinar se existe algum interesse registrado. Se algum critério for satisfeito, o nó verifica o senderID do nó que transmitiu a subscrição. Após isso, o nó monta uma mensagem de notificação de evento e a envia para seus vizinhos, como exemplificado na figura 2. Quando cada nó vizinho recebe a mensagem, compara destID, recebido na notificação, com seu próprio identificador (nID). Se o resultado for verdadeiro, o nó armazena coord e senderID em sua tabela de roteamento, verifica o sID da tabela de roteamento e repete o algoritmo até a notificação alcançar o sink.

```
    // Notification
sub = subscriptionTable.match(publication);
if (sub)
{  // Source node initializes a notification msg
   notif.type = sub.type;
   notif.value;
   notif.criteria = sub.criteria;
   notif.coord = sub.coord;
   notif.sinkID = sub.sinkID;
   notif.destID = sub.sID; // Sets the destination.
   notif.senderID = node.ID; //To send its ID to
                             //destination.
}
```

Fig. 2. Envio da mensagem de notificação.

Pode ser notado que devido à característica da configuração inicial da rede, o número máximo de vizinhos que transmitem para um nó é três. Na figura 3 é possível verificar que cada nó processa mensagem apenas dos nós que estão em um nível de hop anterior. Por exemplo, um nó no nível 5 recebe mensagens apenas de nós de nível 4, e assim por diante. Esta característica facilita a escolha do vizinho que transmitiu mais rápido, e também evita loops de mensagens. Supondo que um sink S envia uma subscrição para a rede configurada como na figura 3 e considerando que o nó localizado no topo da rede e à esquerda seja o sensor que produz um evento que satisfaz a subscrição do sink S, a Figura 4 mostra o caminho formado até o sink S para o envio da mensagem de notificação. Note que as setas indicam os links que poderiam formar caminhos alternativos, dependendo apenas da escolha de cada nó pelos links mais rápidos que entregaram a subscrição.

Uma característica importante da configuração dos valores de hops pode ser observada na etapa de envio da notificação. Quando um nó recebe uma transmissão de um vizinho, ele apenas retransmite a mensagem se esta partiu de um nó de hop superior em uma unidade. Por exemplo, apenas os nós com hop = 4 repassam a informação transmitida pelo nó com hop = 5, e assim por diante. O algoritmo para a notificação é mostrado na figura 5. Pode ser observado na figura 4 que os nós que não fazem parte do caminho para o sink não possuem setas. Isto significa que esses nós não transmitem mensagens naquele ponto e, portanto, não gastam energia como os outros nós que formam o caminho.

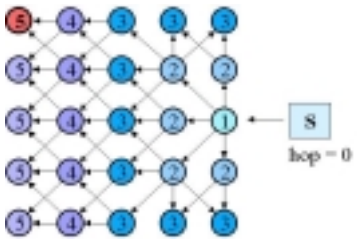

Figura 3. Configuração inicial e propagação da subscrição.

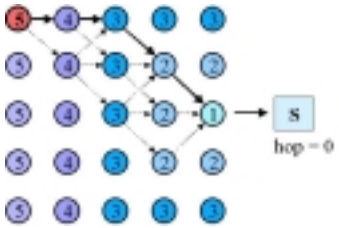

Figura 4. Caminho criado entre os nós para a entrega da notificação.

```
// When a node receives a notification message,
if (notif.destID == node.ID)
{
        // Gets the record for sinkID.
      route = routingTable.get(notif.sinkID);
        // Stores the sender ID in the routtable.
      route.nID = notif.senderID;
        // Stores the coord in the routing table.
      route.coord = notif.coord;
        // Sets the destination.
      notif.destID = route.sID;
        // Sends the message.
      notif.sendNotificationMsg();
}
```

Fig. 5. Algoritmo de notificação.

D. *Envio Dirigido de Mensagens de Subscrição*

O objetivo principal do algoritmo de propagação de mensagens de notificação é encontrar o caminho mais rápido entre o nó que produziu o evento e o sink. O caminho utilizado para entrega de notificações pode ser usado posteriormente pelo sink para enviar subscrições para a mesma região delimitada pelo atributo coord. Para isso, durante a entrega da notificação, cada nó no caminho para o sink armazena, na tabela de roteamento, os atributos coord e senderID do nó vizinho que transmitiu a mensagem de notificação. Quando um nó recebe uma mensagem de subscrição, ele compara o atributo coord da subscrição com o coord armazenado. Se o resultado for verdadeiro, o nó retransmite a mensagem especificando o atributo destID para que somente o nó vizinho que possua este destID transmita. Assim, só transmitirão os nós que serviram de rota para

notificações anteriores. Resumindo, para acelerar novas subscrições para uma região de sensores, a etapa de subscrição pode usar o caminho inverso utilizado por notificações, como mostrado na figura 6 (chamado de envio dirigido). Isto é útil quando subscrições do tipo consulta (query-driven) devem ser suportadas. Caso contrário, se o nó não possuir um valor para coord compatível com o da transmissão, o nó transmite sem especificar o senderID, assim todos vizinhos transmitirão a mensagem de subscrição. O algoritmo de envio de mensagens de subscrição é mostrado na figura 7.

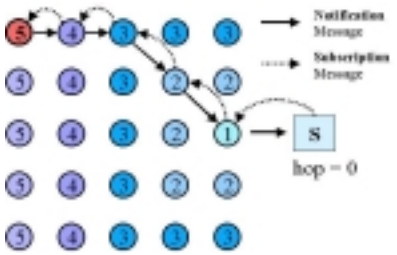

Fig. 6. Envio dirigido de subscrições.

```
// When a node receives a subscription message, it
first checks to see if its hop value matches the
subscription hop value.

// Gets the config record associated with sinkID.
config = configTable.get(sub.sinkID);
if (sub.hop == config.hop)
      // To get only the first subscription msg.
  if (config.subTimeStamp <> sub.timeStamp)
     // checks to see if this node is the
     // publisher.
     if (node.coord == sub.coord AND node.type ==
         sub.type)
     {// Does this subscription exist?
        if (subscriptionTable.match(sub))
           subscriptionTable.refresh(sub);
        else  // it stores the subscription.
           subscriptionTable.add(sub);
     }
     else
     {   // checks if it is the destination.
         if (node.ID == sub.destID)
         {  // Tries to get the record
            route.routingTable.get(sub.coord)
       if (route) // Is there a route?
              sub.destID = route.nID; //sets the
                           // destination.
           else // There is no route!
           {  // Stores the sender ID.
              route.sID = sub.senderID;
              // To send its ID to the other nodes.
               sub.senderID = node.ID;
                 // Msg destined to all neighbors.
              sub.destID = null;
                 // Only nodes with hop+1 will
                 //process the message.
              sub.hop = sub.hop+1;
              sub.sendSubMsg(); // Sends the msg.
           }
    }
    else  // Is this msg to all neighbors?
           if (sub.rID == null)
           {  // To avoid getting the same
              //subscription from other nodes.
              config.subTimeStamp =
                sub.timeStamp;
                 // Used to forward notificmsgs.
              route.sID = sub.senderID;
     route.sinkID = sub.sinkID;// Idem.
                 // Creates a route in the
                 //routing table.
     routingTable.add(route);
                 // Only nodes with hop+1 will
                 //process the msg.
              sub.hop = sub.hop+1;
              sub.sendSubMsg();// Sends the msg.
           }
}
```

Fig. 7. Algoritmo de subscrição.

Como o envio dirigido de subscrições utiliza o mesmo caminho criado para mensagens de notificação, apenas aqueles nós que formam o caminho é que gastam energia transmitindo. Os outros nós, ou apenas recebem e não transmitem (no caso dos nós vizinhos aos nós do caminho, e que não pertencem a ele), ou nem sequer recebem mensagens. A figura 8 mostra um mapa que representa o consumo de energia da rede na utilização do caminho especificado. Nos nós mais escuros há um gasto maior de energia.

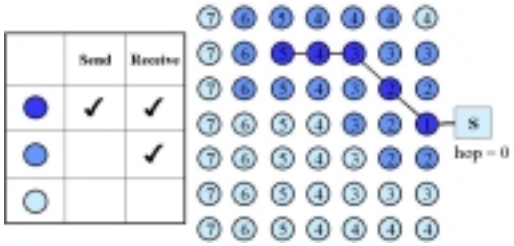

Fig. 8. Mapa de energia da rede.

E. Reparo de Caminhos Quebrados e Configuração de Caminhos Múltiplos para Tolerância a Falhas

O caminho criado para o envio da mensagem de notificação é único e mais eficiente (oferece latência mais baixa e economiza energia). Ele também pode ser usado para o envio dirigido de novas subscrições (para cenários baseados em consulta, por exemplo, que requerem subscrições randômicas). Porém, por ele ser único, qualquer falha em um dos seus nós impossibilitará a entrega do evento e do envio dirigido de subscrições. Quando um sink percebe que a rede necessita de reconfiguração devido às falhas em caminhos, ele reenvia uma mensagem de configuração, associando novos valores de hops para os nós. Um caminho inutilizado é mostrado na figura 9(a). Quando o sink reenvia a mensagem de configuração, o caminho para a notificação pode ser reconstituído, como mostra a Figura 9(b). É óbvio que, se todos os vizinhos de um nó falharem ele ficará isolado e sua transmissão não alcançará nenhum nó. Uma solução seria configurar o módulo de rádio do nó para aumentar sua área de cobertura, porém gerando um maior gasto de energia. Outra solução é oferecer tolerância à falhas através do estabelecimento de múltiplos caminhos dos nós para o sink. Quando um sink recebe um evento crítico, como detecção de fogo, ele pode informar a rede de sensores que, de agora em

diante, os nós podem usar todos os seus vizinhos para enviar eventos para o sink.

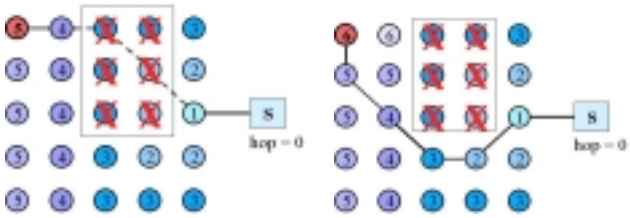

Figura 9 (a). Região com nós destruídos. 9(b). Reconstrução do caminho

Esse mecanismo é necessário devido a maior probabilidade de falhas em nós em uma situação de fogo ou em outras circunstâncias críticas. Os nós podem mudar para um modo de emergência onde cada nó transmite a informação especificando os nós em um nível menor de hop como sendo os receptores. Por exemplo, um nó no nível de hop 5 especificará os nós no nível 4 como sendo os receptores. Dessa forma, mais caminhos podem ser criados para entregar as mensagens de notificação para o sink, e uma maior confiabilidade pode ser conquistada, como pode ser visto na figura 10. Note também que um nó pode receber a mesma mensagem de vizinhos distintos. Para evitar essa repetição de mensagens, um nó receptor pode verificar suas lista de transmissões recentes para ver se já recebeu a mensagem e decidir em não transmiti-la.

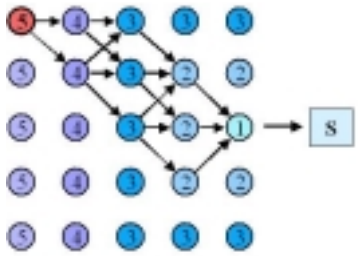

Fig. 10. Tolerância a falhas através do mecanismo de envio no modo de emergência.

A decisão de reconfiguração da rede para corrigir caminhos quebrados sempre parte do sink Um sink pode decidir em reconfigurar a rede quando eventos anormais são recebidos, para poder mudar para o modo de emergência, onde múltiplos caminhos são definidos para o envio de notificações. Situações ambíguas também são tratadas no sink, ou seja, o sink decide, através das políticas estabelecidas, se um evento anormal recebido de um sensor é um evento que reflete o que aconteceu no ambiente físico ou ele é apenas o resultado de um sensor avariado.

III. TRABALHOS RELACIONADOS

É bem conhecido que um sensor no estado ocioso também consome energia e que economizar energia significa desligar completamente a comunicação [2]. STEM [5] oferece uma boa solução de economia de energia quando os sensores têm que ser chaveados, de tempos em tempos, para o modo de transmissão de dados, i.e., quando o cenário da aplicação é basicamente orientado a eventos. Entretanto, quando o chaveamento para modo de transmissão de dados tem que ser feito com freqüência, como quando diferentes tipos de subscrições são solicitados (em cenários baseados em consulta), o custo do chaveamento para o modo de transmissão de dados pode superar os benefícios da economia de energia. No algoritmo descrito aqui, três cenários de aplicação são suportados (periódico, orientado a evento e baseado em consulta), i.e., o algoritmo suporta tolerância à falha e baixa latência, requisitos não suportados pelo STEM. Outros protocolos são muito eficientes na economia de energia, mas também não suportam tolerância à falhas e re-configuração de rede, como EAD [4] e LEACH [6], dentre outros. Em outros paradigmas, como o *Directed Difusion* - DD [1], cada nó armazena toda mensagem de interesse emitida pelo nó destino *(sink)*, mesmo que o nó não publique um evento correspondente a aquele interesse. No nosso algoritmo, cada nó intermediário possui uma tabela de roteamento para direcionar as mensagens que chegam e assim, o nó não tem que executar um algoritmo completo de emparelhamento de interesse *(matching)* sempre que recebe uma mensagem.

IV. RESULTADOS DE SIMULAÇÃO

O algoritmo foi implementado em C e simulado no simulador de redes NS-2 [7]. Os cenários de simulação consistem de diversos campos de sensores com tamanhos diferentes, variando de 100 a 500 nós. Os nós foram espalhados aleatoriamente pelo espaço e foram usados cinco nós fontes e um nó sink.

A Tabela 1 mostra os parâmetros usados na simulação. Os valores foram baseados nos valores reportados para o DD em [1]. Cada nó na rede foi configurado de acordo com a Tabela 1 (adaptada de [1]), os nós fontes geram 10 eventos por segundo, e a cada 20 segundos um evento de reparo é enviado pelo sink. A faixa de alcance do rádio dos nós foi configurada para 20 metros para representar com maior realismo um módulo de rádio dos sensores. O PEQ e o DD foram testados nos mesmos cenários de simulação e com os mesmos parâmetros. Cada valor nos gráficos foi extraído de uma média de 20 simulações.

Tabela 1. Parâmetros da Simulação [1]

Parameters	**Values**
Tempo de simulação (s)	500
Número de nós	100-500
Tx. de dados do fonte (eventos/s)	10
Intervalo de reparo (s)	20
Alcance do rádio (m)	20
Energia para Transmissão (mW)	14.88
Energia para Recepção (mW)	12.50
Energia no modo Ocioso (mW)	12.36
Energia no modo Sleep (mW)	0.016

O algoritmo foi avaliado através de duas métricas importantes: atraso **Sink-Fonte-Sink** e **Atraso médio,** mostrados, respectivamente nas figuras 11 e 12.

O PEQ usa a mensagem de subscrição para propagar a configuração inicial que constrói o caminho do sink ao nó fonte, que será utilizado para a entrega de eventos. O Directed Diffusion, por outro lado, propaga a subscrição (interesse) e, quando o nó fonte recebe essa subscrição, ele propaga um evento exploratório ao sink usando múltiplos caminhos, e apenas um desses caminhos será "reforçado" pelo sink. A criação de caminhos no PEQ possui menos passos e é mais rápida que no DD, resultando em menores atrasos, como mostrado na Figura 11.

As falhas na rede foram simuladas desligando-se uma fração fixa de nós. Esses nós foram escolhidos e desligados aleatoriamente durante a simulação. Com o aumento do tamanho da rede, o atraso é também aumenta devido ao maior número de nós que um evento precisa atravessar do nó fonte ao sink. Isso faz sentido, já que para reparar um caminho quebrado, o algoritmo precisa achar nós que estejam funcionando. Assim que o número de nós quebrados aumenta, o novo caminho criado torna-se maior. O PEQ sempre tenta encontrar o caminho mais curto ao sink. Como mostra a Figura 12, para um tamanho fixo da rede, por exemplo, 500 nós, o atraso aumentou de 49ms para 58ms, um atraso aceitável para o cenário de aplicação considerado neste artigo. Comparando o PEQ com o DD, o PEQ resultou em atrasos menores na rede com presença de falhas, devido ao complexo mecanismo que o DD usa para encontrar caminhos alternativos.

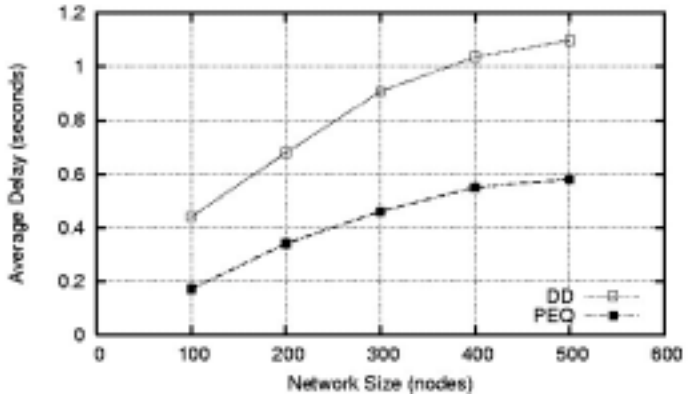

Fig. 11. Gráfico do atraso médio Sink-Fonte-Sink.

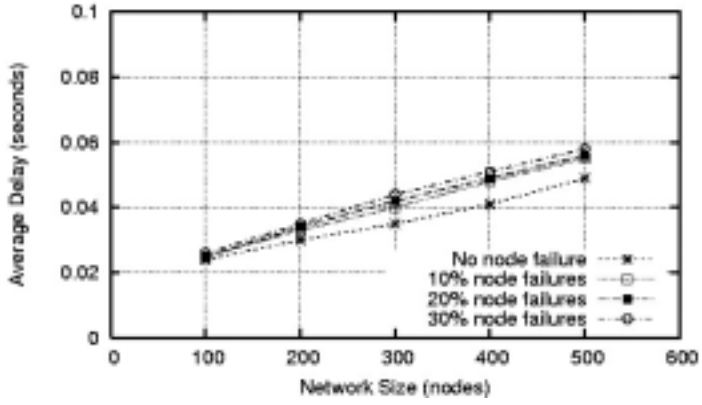

Fig. 12. Gráfico do atraso médio em uma rede com falhas.

V. CONCLUSÕES

Este artigo descreve um algoritmo para monitoramento de ambientes físicos em condições críticas em que baixa latência é suportada através de configuração do caminho mais curto para a entrega de eventos. Subscrições rápidas de novos interesses (em cenários baseados em consultas) são suportadas através do conceito de *entrega dirigida de eventos*, no qual novas subscrições para uma região de sensores são agilizadas utilizando-se o caminho inverso usado para notificações de eventos. Tolerância a falhas é controlada pelo nó receptor *(sink)* de tal forma que, quando o nó receptor "nota" que eventos anormais estão sendo notificados e que alguns nós podem ter sido destruídos, uma entrega multi-caminhos é ativada, através de re-configuração da rede.

Métricas importantes foram estimadas e comparadas em relação ao paradigma Directed Diffusion, mostrando que o PEQ apresenta praticamente a metade o atraso médio do DD, mesmo com uma alta porcentagem de falha. Valores acima de 80% foram obtidos para taxa de sucesso no envio de eventos, fazendo do PEQ um bom candidato para satisfazer os requisitos de aplicações de monitoramento em situações de emergência.

REFERÊNCIAS

1 C. Intanagonwiwat, R. Govindan and D. Estrin: Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In Proc. 6th ACM/IEEE International Conference on Mobile Computing – MOBICOM'2000.

2 I. Chatzigiannakis, S. Nikoletseas and P. Spirakis: A Comparative Study of Protocols for Efficient Data Propagation in Smart Dust Networks. In Proc. 2nd ACM Workshop on Principles of Mobile Computing – POMC'2002.

3 P. Th. Eugster, P. Felber, R. Guerraoui, A. Kermarrec: The many faces of publish/subscribe. *ACM Comput. Surv. 35(2): 114-131 (2003)*

4 A. Boukerche, X. Cheng, J. Linus, Energy-Aware Data-Centric Routing in Microsensor Networks. In MSWiM'03, September 19, 2003, San Diego, California, USA. (2003).

5 C. Schurgers, V. Tsiatsis, S. Ganeriwal and M. Srivastava: Topology Management for Sensor Networks: Exploiting Latency and Density. In Proc. MOBICOM 2002.

6. W. R. Heinzelman, A. Chandrakasan and H. Balakrishnan: Energy Efficient Communication Protocol for Wireless Microsensor Networks. In Proc. 33rd Hawaii International Conference on System Sciences – HICSS'2000.

7. The Network Simulator ns-2. www.isi.edu/nsman/ns