UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALEX LUIZ DOMINGUES CASSINELLI

EM BUSCA DA APLICAÇÃO DE PROTOCOLOS DE ROTEAMENTO PARA EVITAR ATAQUES DO TIPO BURACO NEGRO

BAURU
Novembro/2024

ALEX LUIZ DOMINGUES CASSINELLI

EM BUSCA DA APLICAÇÃO DE PROTOCOLOS DE ROTEAMENTO PARA EVITAR ATAQUES DO TIPO BURACO NEGRO

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.
Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

BAURU
Novembro/2024

C345b

Cassinelli, Alex Luiz Domingues
Em Busca da Aplicação de Protocolos de Roteamento para Evitar Ataques do Tipo Buraco Negro / Alex Luiz Domingues Cassinelli. -- Bauru, 2024
54 p. : il.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências, Bauru
Orientador: Kelton Augusto Pontara da Costa

1. Segurança de Sistemas. 2. Redes de Computadores. 3. Roteamento (Administração de redes de computadores). I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Dados fornecidos pelo autor(a).

Alex Luiz Domingues Cassinelli

Em Busca da Aplicação de Protocolos de Roteamento para Evitar Ataques do Tipo Buraco Negro

Trabalho de Conclusão de Curso do Curso de Ciência da Computação da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Prof. Dr. Kelton Augusto Pontara da Costa
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profa. Dra. Simone das Graças Domingues Prado
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Prof. Pedro Henrique Paiola
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, 19 de Novembro de 2024.

Dedico este trabalho a minha família, professores e amigos, que estiveram comigo e me apoiaram por toda esta jornada

Agradecimentos

Agradeço, primeiramente, a meus pais e meu irmão, por terem me apoiado, me ajudado e cuidado de mim, durante toda minha vida. Agradeço, também, ao professor Kelton Augusto Pontara, por ter me orientado neste estudo, agradeço pela paciência, motivação e conhecimento que ele me proporcionou.

Agradeço aos meus amigos, em especial o Vinicius e o Kaio, que me ajudaram e sempre estiveram comigo, durante toda a graduação e durante a preparação deste trabalho. Agradeço também ao Herminio, Eric, Ian, Yuki, Franco e Luana, por fazerem parte deste curto período de muita felicidade e muitas batalhas.

Por fim, gostaria de agradecer todos os professores da graduação, em especial a professora Tatiana Miguel Rodrigues, do departamento de Matemática, agradeço pela oportunidade de realizar uma Iniciação Científica, por me apresentar ao mundo da pesquisa e por toda a ajuda e paciência que me ofereceu.

Resumo

O Ataque Buraco Negro é um tipo de ataque cibernético que ocorre em uma Rede Móvel Ad hoc (MANET), se aproveita da forma que o principal algoritmo de roteamento se utiliza para encontrar o caminho entre dois nós. Este ataque é capaz de capturar e descartar os pacotes de dados desta rede, consequentemente ocasionando em uma negação de serviço (DoS). Além disso, o Ataque Buraco Negro é relativamente simples de ser executado, tendo em vista que as MANETs são caracterizadas por sua facilidade de conexão, maleabilidade e mobilidade. Portanto, é importante que sejam pesquisadas formas para evitar, pelo menos, as principais consequências deste tipo de ataque. Esta monografia se concentra na modificação e análise de um algoritmo de roteamento para evitar a principal consequência de um Ataque Buraco Negro: a DoS, utilizando uma abordagem de diferente lógica de roteamento e utilização de múltiplos caminhos simultaneamente. Esta lógica de roteamento é baseada na Busca em Profundidade, que mostrou resultados promissores para redes com poucos nós. A abordagem proposta, embora não sendo muito bem-sucedida, mostrou certa promessa, mostrando que a ideia possui méritos.

Palavras-chave: Ataques Buraco Negro; Redes Móveis Ad hoc; Algoritmo de roteamento; Busca em Profundidade; Múltiplos Caminhos.

Abstract

Black Hole Attacks are a type of cybernetic attack targeting the Mobile Ad hoc Networks (MANET), abusing the way the main routing algorithm uses to find the path between two nodes in the network. This kind of attack is able to catch and drop the data packets, in turn causing a Denial of Service (DoS). Moreover, the Black Hole Attack is relatively simple to execute, considering the main characteristics of MANETs are their ease of connectivity, malleability and mobility. Therefore, it's important to research ways to avoid, at the very least, the main repercussions of the Black Hole Attack. This monograph focuses on the modification and analysis of a routing algorithm to avoid the main consequence of the attack: the DoS, using a different approach with the routing logic and utilizing multiple paths simultaneously. This routing algorithm is based on Depth First Search and showed promising results to networks with a small quantity of nodes. The proposed approach, while not very successful, exhibited a certain promise, indicating that the idea has merits.

Keywords: Black Hole Attack; Mobile Ad hoc Network; Routing algorithm; Depth First Search; Multiple Paths.

Lista de figuras

Lista de quadros

Sumário

1 Introdução

Os Ataques Buraco Negro são um tipo de ataque cibernético, que afetam as redes ad hoc móveis, do inglês Mobile Ad hoc Network (MANET), que tem potencial para causar danos severos ao funcionamento da rede. Eles podem ser considerados como ataques de negação de serviços, do inglês Denial of Service (DoS) e podem interferir no funcionamento de uma MANET como um todo (RANI et al., 2020).

As MANETs são redes sem fio, que surgiram com o advento do uso de aparelhos móveis, e da necessidade de um tipo de rede que pudesse interligar esses aparelhos de maneira sem fio. Assim, as características principais das MANETs são seu dinamismo e sua flexibilidade Abdel-Fattah et al. (2019). As MANETs são formadas por uma rede de dispositivos móveis, também chamados de nós, onde todos possuem a mesma ordem de importância.

A principal forma que as MANETs usam para encontrar os caminhos pelos quais devem mandar os pacotes de um nó para o outro. Segundo Tuteja, Gujral e Thalia (2010), o principal protocolo usado é o protocolo do Vetor Ad-hoc de Distância sob Demanda, do inglês Ad hoc On-Demand Distance Vector (AODV), em decorrência de seu bom desempenho em MANETs grandes. Porém, esse tipo de algoritmo é bastante suscetível aos Ataques Buraco Negro, em decorrência de seu modelo de funcionamento.

Na área da prevenção de Ataques Buraco Negro, os esforços estão concentrados em trabalhar com o protocolo AODV, visto que este é o mais utilizado para o roteamento nas MANETs, como visto em (TSENG; CHOU; CHAO, 2011), que propõe uma proposta de múltiplos caminhos juntamente ao protocolo AODV, e que teve bons resultados. Outro exemplo é de (SHURMAN; YOO; PARK, 2004) utilizar um número de sequência para identificar a procedência do pacote, determinando, assim, se ele é válido ou não.

O algoritmo Vetor Ad-hoc Multicaminhos de Distância sob Demanda, do inglês Ad hoc On-Demand Multipath Distance Vector (AOMDV), é uma modificação do algoritmo AODV, com a principal diferença de manter caminhos alternativos, além do caminho principal, armazenados. Estes caminhos são denominados caminhos de nós disjuntos (TIAN; HOU, 2010), ou seja, é um caminho onde os únicos nós em comum são os nós inicial e final.

Uma alternativa para o problema dos Ataques do Tipo Buraco Negro, seria de usar um outro tipo de algoritmo, como a Busca em Profundidade, ou, do inglês, Depth First Search (DFS). Segundo Alamsyah et al. (2020) a DFS é um algoritmo de busca profunda, que, a partir de um nó raiz, escolhe o nó mais à esquerda no próximo nível.

O protocolo Aprendizado da Caminhada Ordenada, do inglês Ordered Walk Learning (OWL), é um algoritmo de roteamento alternativo ao AODV, que utiliza a DFS como base, sendo um algoritmo não tão eficiente quanto o outro em redes com mais nós, porém ele é mais eficiente em redes menores (ALAMSYAH et al., 2020). Este algoritmo foi escolhido para o estudo deste trabalho, sendo modificado para aceitar mais de um caminho, em busca de evitar melhor os Ataques Buraco Negro.

Porém, devido a resultados promissores em MANETs menores é interessante olhar para o protocolo OWL como uma possível alternativa para o AODV nestes casos, como mostra (ALAMSYAH et al., 2020). Entretanto não foram encontrados trabalhos nos quais o protocolo OWL fosse modificado para aceitar vários caminhos, sendo esta a proposta deste trabalho. Portanto este trabalho tem as seguintes contribuições:

- A utilização de um método alternativo para evitar os Ataques Buraco Negro;
- Fornecer mais dados sobre o funcionamento de um algoritmo baseado em DFS em situações de ataque;
- Desenvolver um código que seja capaz de realizar o objetivo proposto;

1.1 Problemática

O ataque de Buraco Negro é um tipo de ataque cibernético, que afeta as MANETs, pois ele se aproveita do protocolo de roteamento mais utilizado atualmente, o AODV, para se fazer efetivo, visto que ele se utiliza das mensagens RREP para fazer efeito (RANI et al., 2020). Além disso, ataques deste tipo causam muitos prejuízos, pois, segundo Chhabra, Gupta e Almomani (2013), eles afetam a performance da MANET em vários pontos, como interface sem fio e buffer.

Como o possibilitador principal do ataque de Buraco Negro é o protocolo AODV, este trabalho terá um foco em encontrar uma alternativa a este algoritmo, através de algoritmos do tipo DFS, para oferecer uma solução.

1.2 Justificativa

O ataque Buraco Negro é um tipo de ataque que tem potencial de causar muitos problemas, tais como: piorar a performance do buffer, piorar a performance da interface sem fio entre aparelhos e piorar a performance da MANET como um todo (CHHABRA; GUPTA; ALMOMANI, 2013). Além disso, é um ataque relativamente simples de executar, tendo em vista a natureza dinâmica da topologia, além da falta de conexão física entre os aparelhos. Por fim, vale ressaltar que o ataque Buraco Negro é possibilitado por conta de características específicas do protocolo AODV.

Tendo em vista estes pontos, se faz necessário um meio de combater, ou até mesmo evitar por completo, este tipo de ataque. Nesse sentido, um novo protocolo de roteamento é um desenvolvimento desejado, tendo em vista que ele pode diminuir consideravelmente as consequências desse tipo de ataque.

1.3 Objetivos

1.3.1 Objetivo Geral

Desenvolver um método alternativo de descobrimento de caminhos para evitar os ataques do tipo Buraco Negro.

1.3.2 Objetivos Específicos

- Modificar o algoritmo de detecção de caminhos para aceitar mais de um caminho;
- Preparar o ambiente de testes com dois ambientes diferentes: um sob condições normais e outro sob ABNs;
- Testar o algoritmo modificado nos dois ambientes, simulando o envio de pacotes em duas condições;
- Avaliar o algoritmo sob métricas relevantes, como taxa de entrega, latência, consumo de energia e taxa de perda de pacotes.

1.4 Organização do Trabalho

O trabalho possui a seguinte estrutura: O capítulo 2 oferece uma explicação sobre a base teórica do trabalho, o capítulo 3 detalha o planejamento para a realização das experimentações com as MANETs, o capítulo 4 exibe os resultados dos testes realizados para o trabalho, enquanto o capítulo 5 mostra a conclusão atingida pelo autor durante a elaboração do trabalho.

2 Fundamentação Teórica

2.1 MANETs

Ao longo de sua história as redes ad-hoc podem ser divididas em três gerações, com a primeira delas surgindo na década de 1970, nomeada Rede de Pacotes de Rádio, ou, do inglês, *Packet Radio Network* (PRNET), fruto de pesquisas militares. Esta tendência se seguiu para a segunda geração, já na década de 1980, visto que a tecnologia continuou sob controle militar. Porém, na década de 1990, com o advento dos notebooks, as aplicações comerciais das, agora nomeadas MANETs, foram ficando cada vez mais claras, iniciando-se, assim, a terceira geração (BANG; RAMTEKE, 2013).

As MANETs são redes que não precisam de infraestrutura preparada para transportar pacotes entre dois dispositivos, também chamados de nós. Em redes deste tipo, todo nó é tratado, tanto como roteador, quanto como host, além disso, a topologia de tal rede é plana, ou seja, a rede não apresenta nós com mais, ou menos, importância que outros. Por fim, as MANETs possuem topologia dinâmica, tendo em vista sua falta de infraestrutura, promovendo uma rede muito dinâmica (MIRZA; LÓPEZ BAKSHI, 2018).

Segundo Mirza e López Bakshi (2018) algumas características são importantes de serem citadas para um maior entendimento sobre as MANETs, são eles: Operações particionadas, terminais autônomos, roteamento de múltiplos saltos, topologia de rede dinâmica, capacidade de conexão flutuante e terminais leves.

O ponto principal das operações particionadas é que cada nó deve se comportar como transmissor. Sobre os terminais autônomos, cada terminal é nó da rede, sendo que ele pode operar como host, ou roteador. No roteamento de múltiplos saltos, é ditado que os pacotes de dados devem passar por um, ou mais, nós intermediários. A topologia de rede dinâmica significa que os nós criam a rede dinamicamente entre eles, formando redes próprias. A característica da capacidade de conexão flutuante é que, pela natureza dinâmica da rede, o caminho entre um par qualquer de usuários pode passar por várias conexões. Por fim, o significado dos terminais leves é que os dispositivos que compõem a rede são menos potentes que um equipamento dedicado para controlar uma rede.

Por fim, as MANETs são muito utilizadas atualmente, participando de várias áreas. Esse grande uso se dá por causa do dinamismo e falta de infraestrutura dessas redes. O primeiro setor a ser citado é o setor militar, tendo em vista que quase tudo é computadorizado, isso poderia significar criar redes de comunicação entre esses

dispositivos. Outro setor que pode ser mencionado é o setor de gerenciamento de desastres, visto que as MANETs podem ser utilizadas em caso de serviços de resposta a desastres, como incêndios e enchentes. Por fim, um último setor a ser mencionado é nas redes de sensores, visto que os mesmos poderiam formar redes de comunicação entre si e sem fio, facilitando a montagem e comunicação entre eles. (RAMPHULL et al., 2021).

2.2 Fundamentos em Grafos

Grafos são um tipo de estrutura de dados e uma linguagem universal para representar sistemas complexos (HAMILTON, 2020). De maneira geral, um grafo poderia ser descrito como uma coleção de objetos, como nós, e interações entre pares de objetos, como arestas. Porém se faz necessária uma explicação formal, portanto, segundo Hamilton (2020) um grafo $G = (V, E)$ é definido por um conjunto de vértices V, e um conjunto de arestas E, que ligam os vértices. Denota-se um nó indo de $v \in V$ para $u \in V$ como $(v, u) \in E$. O tipo de grafo de interesse deste estudo é o grafo simples, estes são grafos onde há apenas uma aresta entre vértices, não existem arestas entre um nó e ele mesmo, e o grafo é não-direcionado, o que significa dizer que $(v, u) \in E \Rightarrow (u, v) \in E$ e que $(u, v) \in E \Rightarrow (v, u) \in E$, ou seja, $(u, v) \in E \Leftrightarrow (v, u) \in E$.

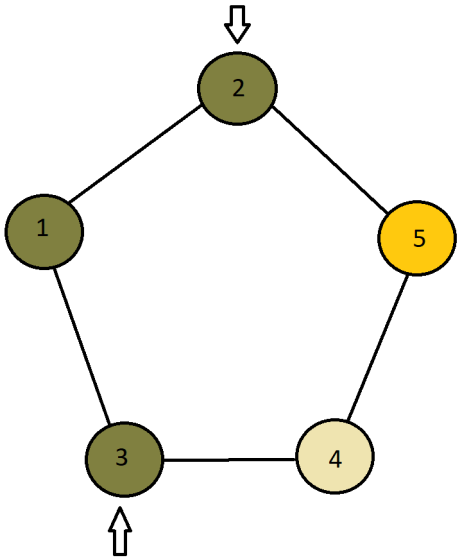
2.2.1 Buscas em Grafos

Além da representação em grafos, tem-se métodos para percorrer esses grafos, a partir daí, surgem os algoritmos de buscas. Nesta pesquisa, serão explicados dois, a busca em largura, do inglês *Breadth First Search* (BFS), e a busca em profundidade, do inglês *Depth First Search* (DFS).

A BFS é uma estratégia onde, dentre todos os nós marcados e incidentes a alguma aresta ainda não explorada, deve-se escolher aquele vértice alcançado por último na busca (ASCENSIO; ARAÚJO, 2010). Ou seja, a BFS é um algoritmo onde todos os nós adjacentes ao nó atual são explorados antes de aprofundar nos nós sucessores do nó atual.

A figura 1 apresenta o comportamento de uma busca em largura, iniciando-se a partir do nó 1 e com final no nó 5.

Figura 1 – Representação de uma BFS

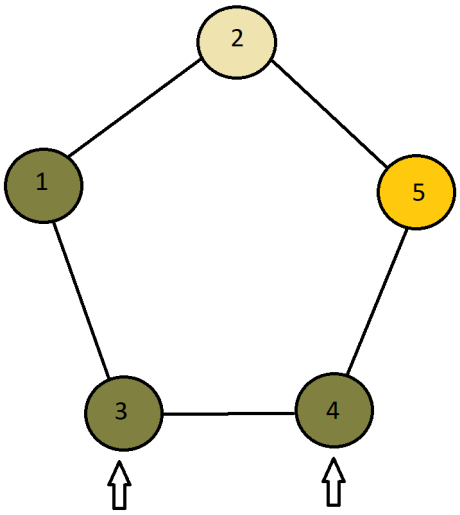


Fonte: Elaborada pelo autor.

Já a DFS é uma estratégia onde, dentre todos nós marcados e incidentes a alguma aresta ainda não explorada, deve-se escolher aquele vértice mais recentemente alcançado na busca (ASCENSIO; ARAÚJO, 2010). Ou seja, a DFS é um algoritmo onde aprofunda-se ao máximo nos sucessores do nó, antes de passar ao próximo.

A figura 2 apresenta o comportamento de uma busca em profundidade, iniciando-se a partir do nó 1 e com final no nó 5.

Figura 2 – Representação de uma DFS



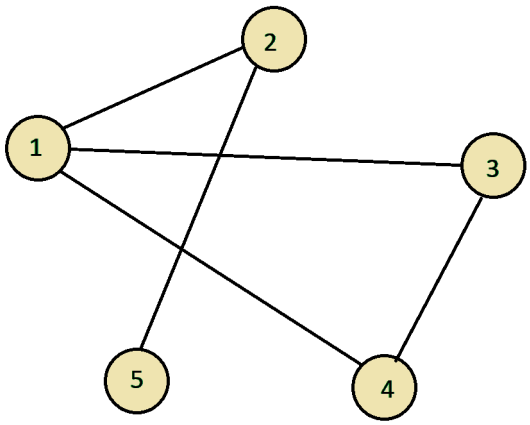
Fonte: Elaborada pelo autor.

2.2.2 Em relação às MANETs

Segundo Rajan et al. (2008), uma rede pode ser modelada matematicamente como um grafo, sendo que a Teoria de Grafos é parte importante da análise dos problemas das redes. Além disso, os problemas das MANETs podem ser elaborados de forma matemática, o que aumenta ainda mais a compatibilidade entre as duas áreas.

Por fim, a representação das MANETs em grafos permite realizar estudos sem a necessidade de implementar, ou testar, modelos e teorias em redes reais, em funcionamento, o que evita prejudicar os usuários dessas redes, enquanto mantém uma forma de testes confiável. A figura 3 é a representação em grafo de uma MANET. Note que as linhas do grafo não são conexões físicas, mas sim caminhos que os nós usam para se comunicar, de maneira sem fio, sendo completamente móveis.

Figura 3 – Representação de uma MANET utilizando teoria dos grafos



Fonte: Elaborada pelo autor.

2.3 Descobrimento de Caminhos

Considerando que as MANETs são redes dinâmicas, se faz necessário um algoritmo de descobrimento de caminhos para que a rede funcione devidamente. Nesta seção, serão abordados três algoritmos, o Vetor de Distância Ad-hoc sob Demanda, do inglês *Ad hoc On-Demand Distance Vector* (AODV), o Vetor de Distância Multicaminhos Ad-hoc sob Demanda, do inglês *Ad hoc On-Demand Multipath Distance Vector* (AOMDV), e o Aprendizado da Caminhada Ordenada, do inglês *Ordered Walk Learning* (OWL).

Ambos algoritmos, entretanto, se utilizam de ferramentas comuns, mais especificamente dois tipos de pacotes, utilizados para realizar a comunicação entre os nós, especificamente o nó de onde sairão os dados (nó origem) e o nó onde os dados

chegarão (nó destino). Esses pacotes são a Solicitação de Rota, do inglês *Route Request* (RREQ), e Resposta de Rota, do inglês *Route Reply* (RREP).

2.3.1 Protocolo AODV

O protocolo AODV possui o seguinte funcionamento: o nó origem envia uma RREQ para encontrar o menor caminho até o nó destino, esta mensagem é enviada para todos os nós adjacentes, ou nós próximos, ao nó origem, e cada nó subsequente, na próxima iteração. Ao ser encontrado, o nó destino envia uma RREP pelo caminho da mensagem que chegou nele, até que chegue à origem. Assim o menor caminho entre os dois nós, origem e destino, foi encontrado, a partir da primeira RREP que a origem recebe, ela ignora quaisquer mensagens RREP posteriores (KARTHIKEYAN; KANIMOZHI; GANESH, 2014).

Portanto, é possível perceber que o AODV possui um comportamento baseado na BFS (SWAMI; SINGH, 2015), visto que ele envia os RREQs para todos os nós adjacentes de um dado nível, assim como a BFS explora o nível inteiro de nós, antes de partir para o próximo.

Enquanto este protocolo possui pontos positivos, como boa taxa de entrega (KARTHIKEYAN; KANIMOZHI; GANESH, 2014), ele possui, também, pontos negativos, tal como o congestionamento de mensagens, caso vários nós enviem RREQs ao mesmo tempo, o que poderia provocar gargalos na rede (SWAMI; SINGH, 2015).

2.3.2 Protocolo AOMDV

O protocolo AOMDV é uma versão modificada do protocolo AODV. Ainda baseado na BFS, ele foi alterado para manter, juntamente com a melhor rota, rotas auxiliares, que, em caso de indisponibilidade da rota principal, podem ser utilizadas para enviar os pacotes para o destino. Todas as rotas utilizadas e armazenadas pelo AOMDV têm o mínimo de nós em comum (YUAN; CHEN; JIA, 2005), para que, caso algum dos nós fique indisponível, um número mínimo de caminhos, idealmente zero, seja afetado. Cabe dizer que o AOMDV não se utiliza dessas rotas de maneira simultânea, mas as mantém como *backup*, caso precise.

Este algoritmo foi escolhido para ser modificado tendo em vista sua estrutura para manter várias rotas em uma tabela. Isto facilitará o desenvolvimento da faceta da utilização de várias rotas simultaneamente. Além disso, o fato de este algoritmo já estar implementado no NS 2 foi um fator determinante na escolha do algoritmo.

2.3.3 Protocolo OWL

O protocolo OWL tem um funcionamento semelhante ao protocolo AODV, no quesito que ele também se utiliza dos pacotes RREQ e RREP para realizar a comunicação entre dispositivos, porém, o método para distribuição desses pacotes é o que diferencia os algoritmos de roteamento. Segundo Swami e Singh (2015), o protocolo OWL envia os pacotes para apenas alguns nós por vez, ao contrário do AODV, que distribui pacotes para todos os nós adjacentes.

Assim, é possível perceber que o OWL possui um comportamento baseado na DFS (DABIDEEN; GARCIA-LUNA-ACEVES, 2009), visto que os nós que recebem os pacotes possuem nível maior que o nó que enviou esse pacote.

Este protocolo possui pontos positivos, como a resolução do problema de congestionamento de mensagens, ele também possui pontos negativos, como uma performance ruim do algoritmo, em comparação com o AODV, caso a rede tenha grande quantidade de nós (DABIDEEN; GARCIA-LUNA-ACEVES, 2009).

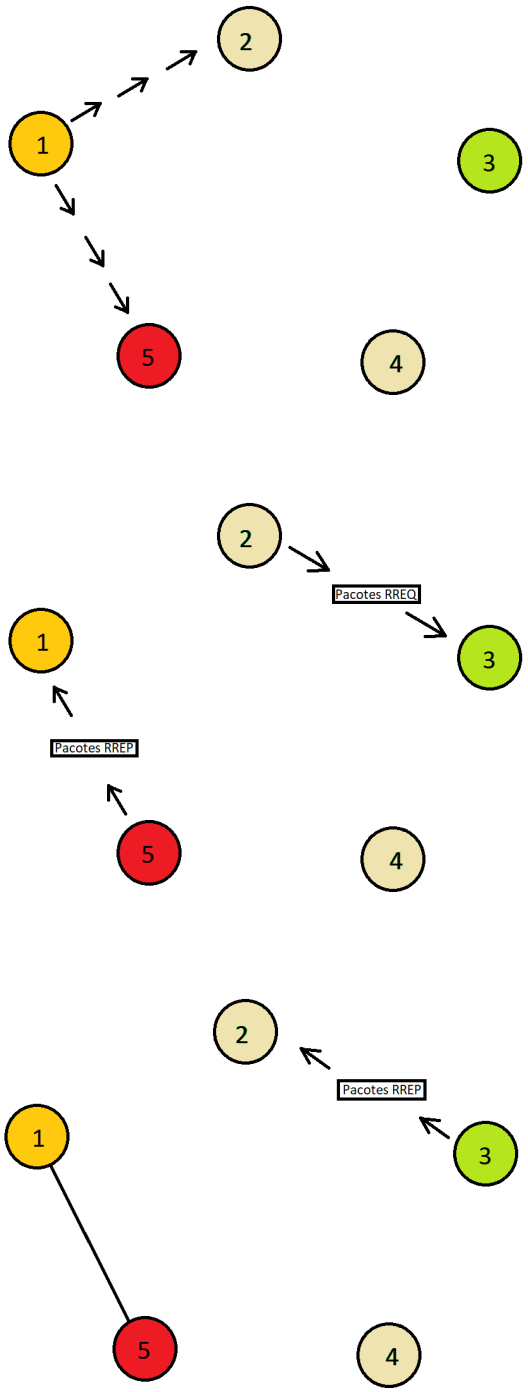
2.4 Ataques Buraco Negro

Os Ataques Buraco Negro são um tipo de ataque que afeta as MANETs, onde um nó malicioso “engana” o sistema de descobrimento de caminhos (normalmente o AODV, pois é o mais utilizado), se passando pelo nó destino, para receber todos os pacotes do nó fonte, descartando os pacotes.

Segundo Tseng, Chou e Chao (2011), o nó malicioso consegue realizar esse feito devido ao método que o AODV utiliza para descobrir os caminhos: o protocolo utiliza o caminho que responder primeiro ao pacote RREQ. Caso o nó malicioso responda com o RREP antes que o nó-origem legítimo, então o nó-fonte passará todos os pacotes para o malicioso. Além disso, os Ataques Buraco Negro podem ser feitos de maneira distribuída, ou seja, através de múltiplos nós-maliciosos, sendo que apenas um deles é necessário que responda primeiro. Por fim, os pacotes recebidos pelo nó malicioso não precisam ser, necessariamente descartados, eles podem ter suas informações enviadas para outros locais, onde os atacantes podem ter acesso a esses dados.

A figura 4 representa o funcionamento de um Ataque Buraco Negro, com o nó 1 sendo o nó origem, o nó 3 é o nó destino e o nó 5 é o nó malicioso. Note que, a partir do momento que o pacote RREP do nó 5 atinge o nó 1, não importa se o nó 3 envia seu próprio RREP, pois o caminho já está fechado.

Figura 4 – Representação visual de um Ataque Buraco Negro



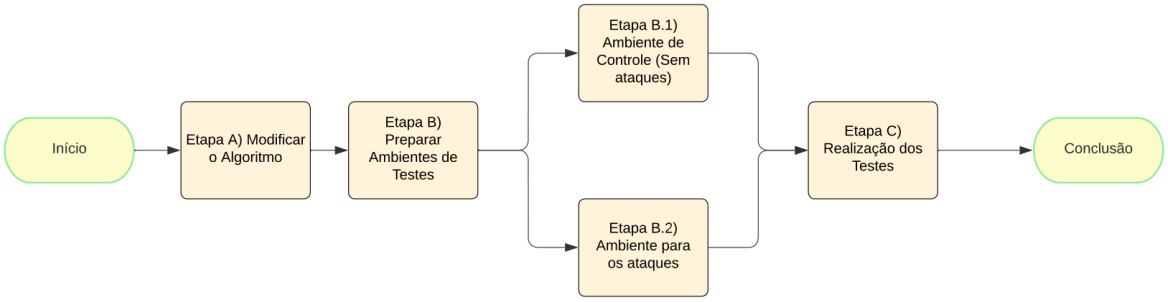
Fonte: Imagens elaboradas pelo autor

Segundo Shurman, Yoo e Park (2004), os Ataques Buraco Negro podem ser considerados como um tipo de ataque de Negação de Serviço, do inglês *Denial of Service* (DoS), ou podem ser utilizados como o primeiro passo num ataque tipo Homem no meio, do inglês, *Man in the Middle* (MitM).

3 Metodologia

Nesta seção será detalhado o planejamento para a realização dos testes para este trabalho.

Figura 5 – Fluxograma do Planejamento



Fonte: Elaborada pelo autor

3.1 Modificação do Algoritmo

O algoritmo escolhido, o AOMDV é baseado em BFS, além de não aceitar mais de um caminho entre nós, portanto, faz-se necessário realizar algumas modificações para que ele use a DFS para encontrar os caminhos e, também, é preciso que ele utilize dois caminhos. A etapa A da figura 5 representa a etapa da "Modificação do Algoritmo".

O algoritmo 1 ilustra o funcionamento do AOMDV:

Algoritmo 1: AOMDV Original Simplificado

Variáveis: pacote, rreq, rrep, origem, destino, caminho_reverso(vetor n), tabela_rotas(matriz nxn), nó_atual

Objetivo: Envio de múltiplos caminhos para pacotes RREQ e resposta RREP

Inicializar tabela de rotas e vizinhos

Função: EnviaRequest($rreq$, $destino$, $caminho\_reverso$):

$rreq.salto \leftarrow ip\_broadcast$

$caminho\_reverso.valor \leftarrow nó\_atual$

envia o pacote

Fim da função

Função: EnviaReply($rrep$, $origem$, $caminho\_reverso$):

$rrep.salto \leftarrow caminho\_reverso[1].valor$

envia pacote

Fim da função

Função: Reverte($caminho\_reverso$):

Inverte a ordem dos elementos do vetor $caminho\_reverso$

Fim da função

Função: Processar RREQ($origem$, $destino$)

if $n = destino$ **then**

| EnviaReply(RREP, nó)

else

| EnviaRequest(RREQ, destino)

end

Fim da função

Função: Processar RREP($origem$, $destino$, $tabela$):

if *rota recebida é viável* **then**

| $tabela\_rotas[1][] \leftarrow inverte(caminho\_reverso)$

end

Fim da função

Função: EnviaPacote($pacote$, $tabela\_rotas$):

$pacote.salto \leftarrow tabela\_rotas[1][].prximo$

envia o pacote

Fim da função

O algoritmo 2 revela o funcionamento modificado do AOMDV:

Algoritmo 2: AOMDV Modificado Simplificado

Variáveis: pacote, rreq, rrep, hello, origem, destino, cópia_pacote, nó_atual, caminho_reverso(vetor n), pilha_vizinhos(vetor n), tabela_rotas(matriz nxn)

Objetivo: Envio de múltiplos caminhos para pacotes RREQ e resposta RREP

Inicializar tabela de rotas e vizinhos

Função: EnviaRequest($rreq$, $destino$, $caminho\_reverso$, $pilha\_vizinhos[]$):

$rreq.salto \leftarrow pilha\_vizinhos[]$

$caminho\_reverso.valor \leftarrow nó\_atual$

envia o pacote

Fim da função

Função: EnviaReply($rrep$, $origem$, $caminho\_reverso$):

$rrep.salto \leftarrow caminho\_reverso[1].valor$

envia pacote

Fim da função

Função: Reverte($caminho\_reverso$):

Inverte a ordem dos elementos do vetor $caminho\_reverso$

Fim da função

Função: Processar RREQ($origem$, $destino$)

if $n = destino$ **then**

| EnviaReply(RREP, nó)

else

| EnviaRequest(RREQ, destino)

end

Fim da função

Função: Processar RREP($origem$, $destino$, $tabela$):

if *rota recebida é viável* **then**

| $tabela\_rotas[1][] \leftarrow inverte(caminho\_reverso)$

end

if $nó = origem$ **then**

| EnviaPacote(pacote, tabela_rotas)

end

Fim da função

Função: Processar Hello($hello$, $pilha\_vizinhos$)

$pilha\_vizinhos[] \leftarrow hello.id\_origem$

Fim da função

Função: EnviaPacote($pacote$, $tabela\_rotas$):

$cópia\_pacote \leftarrow pacote$

$pacote.salto \leftarrow tabela\_rotas[1][].próximo$

$cópia\_pacote.salto \leftarrow tabela\_rotas[2][].próximo$

envia os pacotes

Fim da função

Para olhar o código mais a fundo, as implementações das modificações estão no repositório do Github[1]. Foram colocados ali, apenas os códigos que sofreram mudanças, tendo em vista que os outros se manteram originais e podem ser adquiridos com uma instalação limpa do NS 2.

Como é possível observar, para o funcionamento da DFS, foi implementado na função de envio, um atraso, que espera por 0,05 segundos antes de enviar os pacotes, possibilitando que o caminho seja mais explorado antes do nó começar a explorar outro. Além disso, para o envio dos pacotes para mais de um caminho explorado, os pacotes foram, simplesmente, copiados, para poderem ser enviados em outros caminhos.

3.2 Preparação do Ambiente de Experimentação

Para realizar os testes, são necessários ambientes que simulem as condições de uma MANET. Estes ambientes foram construídos no programa Network Simulator 2 (NS 2). A etapa B da figura 5 refere-se à tarefa específica da "Preparação do Ambiente de Experimentação".

Os ambientes foram construídos com 10, 50 e 100 nós, além disso, o espaço delimitado para os nós aumenta conforme a quantidade de nós: para 10 nós, o espaço é de 300 x 300 metros, para 50 nós, o espaço é de 700 x 700 metros e para 100 nós, o espaço é de 1000 x 1000 metros.

Para que os testes tenham validade, serão feitos de duas formas diferentes: sem ataques e com ataques. Em todos os ambientes, há apenas um par de nós que deseja se comunicar. Todos os nós são posicionados aleatoriamente no espaço delimitado, para simular uma situação realista, portanto, nota-se que, em casos que hajam mais nós, ou que os nós estejam muito distantes, é possível que o algoritmo não encontre um caminho, dado que um pacote RREQ pode expirar antes de encontrar o destino desejado.

3.2.1 Ambiente de Controle

O ambiente de controle tem a intenção de simular uma MANET em situações ideais, sem nós maliciosos e, portanto, sem ataques ocorrendo. Este ambiente será construído desta forma, porque, desta forma, é possível ter dados dos algoritmos funcionando em condições ideais, possibilitando uma medição mais precisa, quando eles forem submetidos a ambientes com ataques. A etapa B.1 da figura 5 ilustra a tarefa geral da "Preparação do Ambiente de Controle".

[1] https://github.com/C4ssinelli/Faculdade/tree/main/TCC

O código para a realização das simulações do grupo de controle é dado no apêndice A. De maneira simples, o funcionamento de código é baseado no seguinte modelo:

- Configuração dos parâmetros do cenário, como quantidade de nós, canal, protocolo de roteamento, etc;
- Criar o objeto do simulador;
- Criar os arquivos trace a serem utilizados;
- Realizar a configuração dos nós, como modelo, mobilidade, etc. Além disso, criar os nós, posicionando-os no cenário;
- Escolher dois nós aleatórios e conectá-los.
- Inserir o último nó, este passo torna mais fácil a inserção do nó malicioso no ambiente de testes;
- Configurar a função de parada;
- Iniciar o simulador.

3.2.2 Ambiente de Testes

O ambiente de testes tem a intenção de simular uma MANET sob ataque de um, ou mais, hackers, com a intenção de colocar os algoritmos à prova. A etapa B.2 da figura 5 indica a tarefa específica da "Preparação do Ambiente de Testes".

Para este ambiente, o último nó inserido é o nó malicioso, sendo colocado na média entre as distâncias dos nós origem e destino. Isto foi feito desta forma para dar a melhor chance do agente malicioso ser bem sucedido em seu ataque. O nó atacante, neste caso, sempre descarta os pacotes.

O código para a realização das simulações do grupo de controle é dado no apêndice B. De maneira simples, o funcionamento de código é baseado no seguinte modelo:

- Configuração dos parâmetros do cenário, como quantidade de nós, canal, protocolo de roteamento, etc;
- Criar o objeto do simulador;
- Criar os arquivos trace a serem utilizados;
- Realizar a configuração dos nós, como modelo, mobilidade, etc. Além disso, criar os nós, posicionando-os no cenário;

- Escolher dois nós aleatórios e conectá-los.
- Inserir o último nó, que, nos ambientes deste grupo, é o nó malicioso;
- Configurar a função de parada;
- Iniciar o simulador.

3.2.3 Os Ambientes

Nesta subseção, estarão os códigos para a realização das simulações. Note que, caso seja de interesse, basta alterar as variáveis num_nodes, para modificar a quantidade de nós, e $val(x)$ e $val(y)$, para mudar os valores das áreas. Note, também, que a única diferença entre os dois códigos é a presença das linhas que dizem respeito ao nó malicioso ($malicious\_node$, no código de testes).

3.3 Realização dos Testes

Os testes serão realizados em duas etapas: primeiramente, serão testadas as MANETs no grupo de controle, seguidas pelo grupo de testes. Como dito anteriormente, haverá apenas um par de nós origem e destino, visto que é mais simples acompanhar a troca de pacotes. Além disso, nos experimentos do grupo de testes, haverá apenas um nó malicioso, visto que, também, é mais simples de se manter os resultados do teste verdadeiros. Todos os nós, exceto pelo nó malicioso, serão inseridos na rede em posições aleatórias, desde que estejam dentro da área delimitada para a simulação. A etapa C da figura 5 faz referência à tarefa da "Realização dos Testes".

A intenção do algoritmo modificado é que, mesmo que ele encontre um caminho que esteja sob ataque, ele tenha um caminho secundário para enviar os pacotes para o nó destino legítimo. Nota-se que, mesmo não havendo interrupção do serviço, este algoritmo não impede que algum agente malicioso consiga dados da rede, visto que o código não impede o envio dos dados para potenciais nós maliciosos.

3.4 Parâmetros de Medição

Para que a performance dos algoritmos possa ser mensurada, faz-se necessária a medição de certos parâmetros. No caso deste estudo, as medidas escolhidas foram: a taxa de entrega dos pacotes de dados, a taxa de entrega dos pacotes de roteamento, a latência na entrega dos pacotes de dados, o consumo de energia médio entre os nós e o tempo de execução de cada teste.

A ideia por trás das taxas de entrega é saber a porcentagem de pacotes que chegam a seu destino final, pois, com isso, é possível ter uma boa noção do quão corretamente os algoritmos estão funcionando. A justificativa para a medição da latência é entender quanto tempo leva para os pacotes chegarem a seu destino, entendendo, por exemplo, quanto tempo um usuário esperaria, em média, para receber os dados que ele requisitou. A intenção em medir o consumo de energia é entender o quão intensivo é o gasto energético dos algoritmos, isto é uma métrica importante, pois é possível determinar se o algoritmo é válido para dispositivos que se utilizam de baterias, como celulares e computadores móveis, discernindo se o algoritmo é viável nestes dispositivos. Por fim, o motivo de medir o tempo de execução total é para saber o quão demorado o algoritmo é para realizar todas as operações pedidas a ele, com este dado, é possível compreender se o algoritmo é válido em determinada situação.

4 Experimentação e Resultados

Neste capítulo, serão apresentados e analisados os resultados dos testes realizados, de acordo com as especificações do capítulo anterior.

4.1 Experimentação

Nesta subseção, serão comentados os experimentos realizados neste estudo. De maneira geral, estes testes se manteram relativamente semelhantes, exceto no ambiente com 100 nós, onde houve uma queda drástica de performance do algoritmo AOMDV modificado. Um ponto interessante é que, mesmo com o maior número de pacotes enviados, o consumo de energia dos nós se manteve praticamente igual entre todos os testes.

Cada experimento foi repetido 50 vezes, para obter-se um bom volume de dados. O quadro 1, são apresentados as métricas de medição dos algoritmos, juntamente com suas unidades de medida.

Quadro 1 – Apresentação dos experimentos e suas respectivas unidades de medida

Latência		
Grupos	**Testes**	**Medidas**
Grupo de Controle	PDR dos pacotes RREQ	Decimal
	PDR dos pacotes de Dados	Decimal
	Gasto médio de energia	(J)
	Latência na entrega dos pacotes	(s)
	Tempo total de execução	(s)
Grupo de Testes	PDR dos pacotes RREQ	Decimal
	PDR dos pacotes de Dados	Decimal
	Gasto médio de energia	(J)
	Latência na entrega dos pacotes	(s)
	Tempo total de execução	(s)
	Quantidade de pacotes descartados por nó malicioso	Pacotes / execução

Fonte: Elaborado pelo autor

A seguir, serão discutidos cada um dos testes, em agrupamentos por tipo de teste, para que seja feita uma comparação mais clara, serão utilizados gráficos.

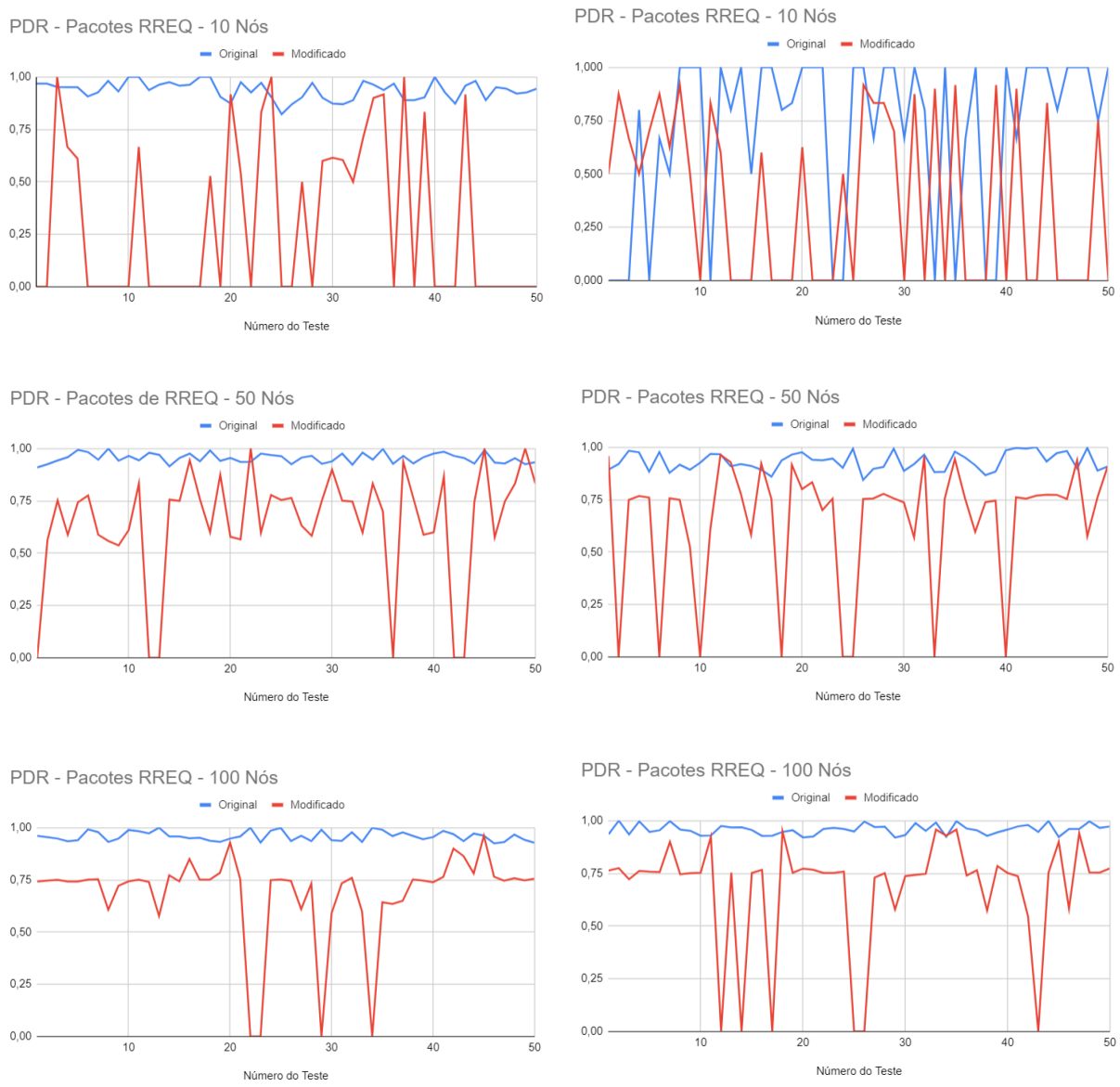
4.1.1 Taxa de Entrega de Pacotes (PDR) RREQ

As medições da PDR foi realizada a partir da divisão da quantidade total de pacotes RREQ recebida pelos nós, pela quantidade total de pacotes RREQ enviada pelos nós, assim retornando a taxa total.

Os gráficos da figura 6 mostram que a PDR de pacotes RREQ foi bastante flutuante, mantendo-se em uma média relativamente baixa para o teste de 10 nós de

ambos os grupos. Entretanto, os testes de 50 e 100 nós mostraram uma melhora na PDR deste tipo de pacote.

Figura 6 – Gráficos representando a PDR dos pacotes RREQ



Fonte: Imagens elaboradas pelo autor

Na figura, os gráficos da esquerda representam os testes do grupo de controle, enquanto que os gráficos à direita representam os experimentos do grupo de testes. Nota-se que nas tentativas com menos nós, houve menor uso dos pacotes RREQ por parte do algoritmo modificado, aumentando nos ambientes com mais nós. Para efeito de maior consolidação, o quadro 2 fornece as médias das PDRs associadas aos gráficos.

Quadro 2 – Média dos resultados dos experimentos para a PDR de Pacotes RREQ

PDR de Pacotes RREQ		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original - Controle	0,937
	Modificado - Controle	0,297
	Original - Testes	0,713
	Modificado - Testes	0,364
50 Nós	Original - Controle	0,954
	Modificado - Controle	0,645
	Original - Testes	0,932
	Modificado - Testes	0,648
100 Nós	Original - Controle	0,959
	Modificado - Controle	0,684
	Original - Testes	0,958
	Modificado - Testes	0,678

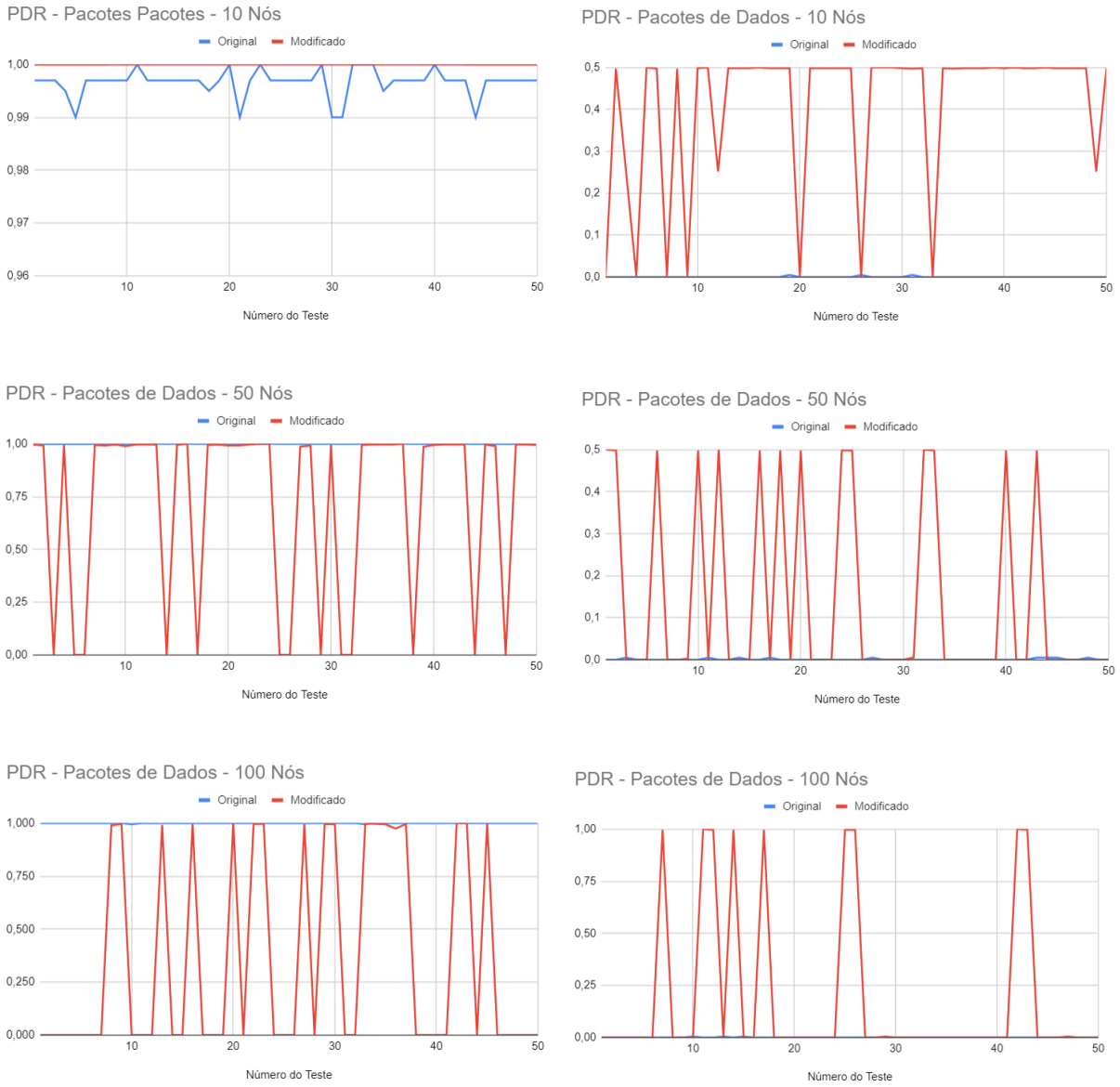
Fonte: Elaborado pelo autor

4.1.2 Taxa de Entrega de Pacotes (PDR) de Dados

As medições da PDR foi realizada a partir da divisão da quantidade total de pacotes de dados recebida pelos nós, pela quantidade total de pacotes dados enviada pelos nós, assim retornando a taxa total. No caso do grupo de testes, o algoritmo de detecção conseguia distinguir entre um pacote recebido e um pacote descartado pelo nó malicioso. Entretanto, todas as taxas do ambiente de testes, foram reduzidas proporcionalmente à quantidade de pacotes descartados pelo atacante, visto que as ações do nó malicioso eram computadas de forma diferente dos pacotes normais.

Como indicado nos gráficos apresentados na figura 7, a PDR de pacotes de dados foi bastante flutuante, especialmente nos ambientes de controle, mantendo-se em uma média relativamente baixa para todos os testes do grupo. Entretanto, os experimentos do ambiente de testes foi melhor que o algoritmo original, entretanto, o algoritmo modificado ainda deixou a desejar quando a quantidade de nós aumentou.

Figura 7 – Gráficos representando a PDR dos pacotes de Dados



Fonte: Imagens elaboradas pelo autor

Na figura, os gráficos da esquerda representam os testes do grupo de controle, enquanto que os gráficos à direita representam os experimentos do grupo de testes. Nota-se que a melhor dos ambientes foi o ambiente sob ataque, com poucos nós, entretanto, ainda é válido dizer que o algoritmo conseguiu enviar alguns pacotes para o destino.

O quadro 3 contém as médias das taxas, para uma comparação numérica entre diferentes algoritmos e ambientes.

Quadro 3 – Média dos resultados dos experimentos para a PDR de Pacotes de Dados

PDR de Pacotes de Dados		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original - Controle	1,000
	Modificado - Controle	0,997
	Original - Testes	0,0004
	Modificado - Testes	0,414
50 Nós	Original - Controle	1,000
	Modificado - Controle	0,737
	Original - Testes	0,001
	Modificado - Testes	0,140
100 Nós	Original - Controle	1,000
	Modificado - Controle	0,358
	Original - Testes	0,000
	Modificado - Testes	0,180

Fonte: Elaborado pelo autor

4.1.3 Gasto Médio de Energia

Para realizar a medição dos gastos energéticos do algoritmo, cada nó foi criado com um valor inicial de energia, que era consumido ao realizar diferentes tarefas, como enviar pacotes, receber pacotes e, simplesmente, manter-se ligado.

A figura 8, com seus gráficos, indicam que o gasto médio de energia se manteve, relativamente, constante, oscilando pouco entre as diferentes quantidades de nós e a presença, ou ausência, do nó malicioso.

Figura 8 – Gráficos representando o Gasto Energético Médio do sistema

Fonte: Imagens elaboradas pelo autor

Na figura, os gráficos da esquerda representam os testes do grupo de controle, enquanto que os gráficos à direita representam os experimentos do grupo de testes. Todas as medições de energia foram feitas em Joules (J). Como dito anteriormente, nota-se que não há grande variação entre os gastos energéticos.

O quadro 4 apresenta as médias dos gastos energéticos para cada um dos testes.

Quadro 4 – Média dos resultados dos experimentos para o gasto de energia

Gasto de energia		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original - Controle	9,420
	Modificado - Controle	9,880
	Original - Testes	6,720
	Modificado - Testes	9,740
50 Nós	Original - Controle	10,944
	Modificado - Controle	12,060
	Original - Testes	10,525
	Modificado - Testes	11,943
100 Nós	Original - Controle	11,270
	Modificado - Controle	12,567
	Original - Testes	11,193
	Modificado - Testes	12,327

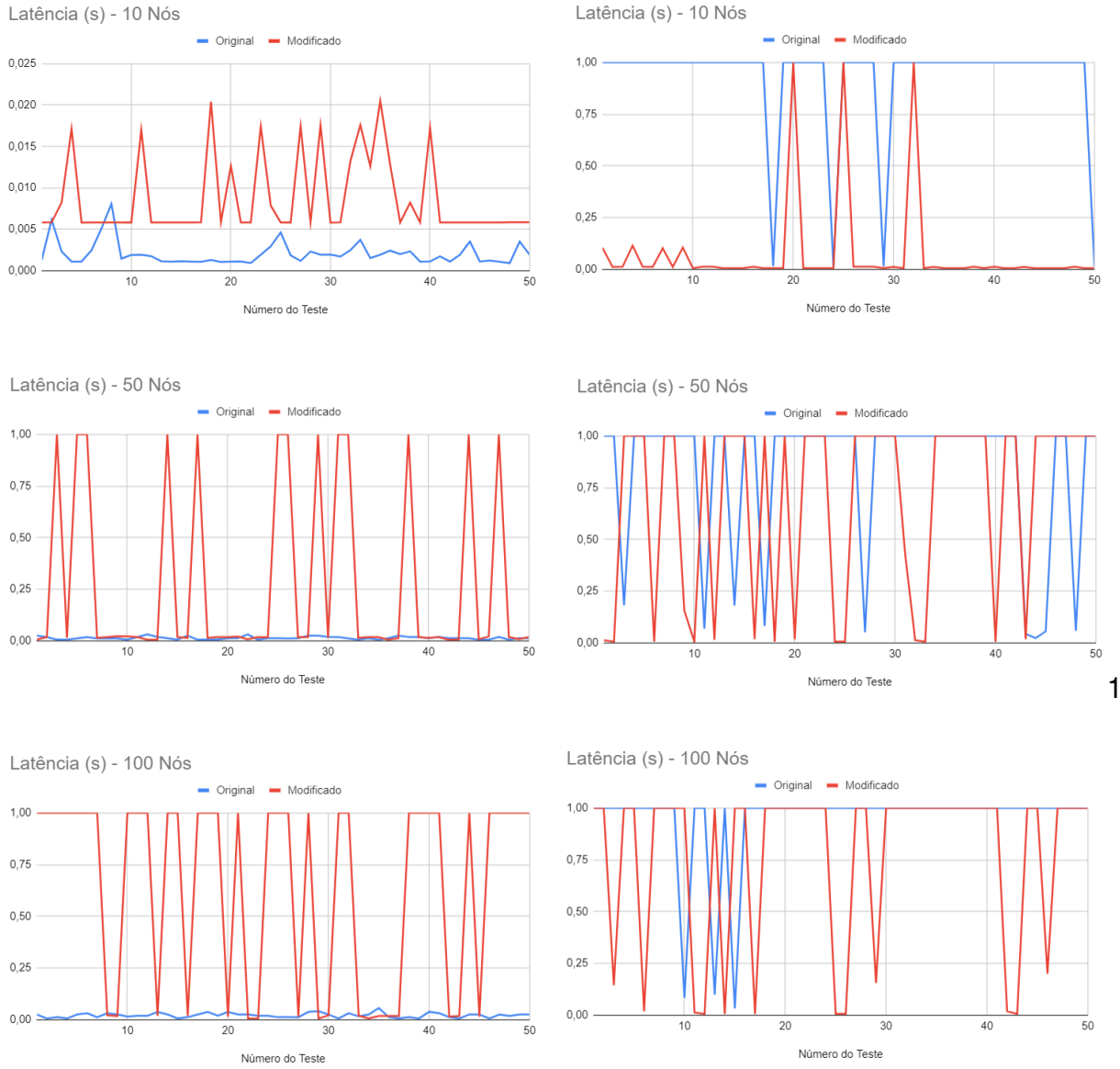
Fonte: Elaborado pelo Autor

4.1.4 Latência Média

Para realizar o cálculo da latência média, foram marcados os tempos da saída do pacote de sua origem, e da chegada do pacote ao seu destino, realizando uma simples subtração para calcular o tempo necessário para completar a viagem. A partir disso foi feita uma média entre a soma dos tempos e a quantidade de pacotes. Para essa métrica, foram contabilizados apenas os pacotes de dados.

Na figura 9, os gráficos mostram que houveram grandes diferenças na latência entre os algoritmos, principalmente nos ambientes de controle com mais nós. Deve-se notar que, quando um gráfico marcar 1s, significa que o pacote não foi recebido, sendo que o 1 representa uma quantia elevada de tempo que o usuário esperaria para tentar de novo, por isso esses erros foram contabilizados na medição desta métrica.

Figura 9 – Gráfico representando a Latência Média



1

Na figura, os gráficos da esquerda representam os testes do grupo de controle, enquanto que os gráficos à direita representam os experimentos do grupo de testes. Como pode ser observado, mesmo com o algoritmo modificado, vários pacotes não foram entregues, ou levaram tempo demais para chegar ao destino.

O quadro 5 representa as médias das medições realizadas e foi construído para auxiliar em comparações mais sólidas. Diferentemente dos gráficos, nos experimentos falhos, o valor atribuído é 10, esta mudança foi feita, pois este valor realmente altera a média muito mais que 1, tornando mais fácil de entender o nível de falha do código, entretanto, nas representações gráficas, 10 tornava o gráfico praticamente impossível de ler.

Quadro 5 – Média dos resultados dos experimentos para a latência

Latência		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original - Controle	0,002
	Modificado - Controle	0,009
	Original - Testes	9,201
	Modificado - Testes	0,616
50 Nós	Original - Controle	0,014
	Modificado - Controle	2,611
	Original - Testes	8,215
	Modificado - Testes	6,815
100 Nós	Original - Controle	0,021
	Modificado - Controle	6,405
	Original - Testes	9,404
	Modificado - Testes	7,612

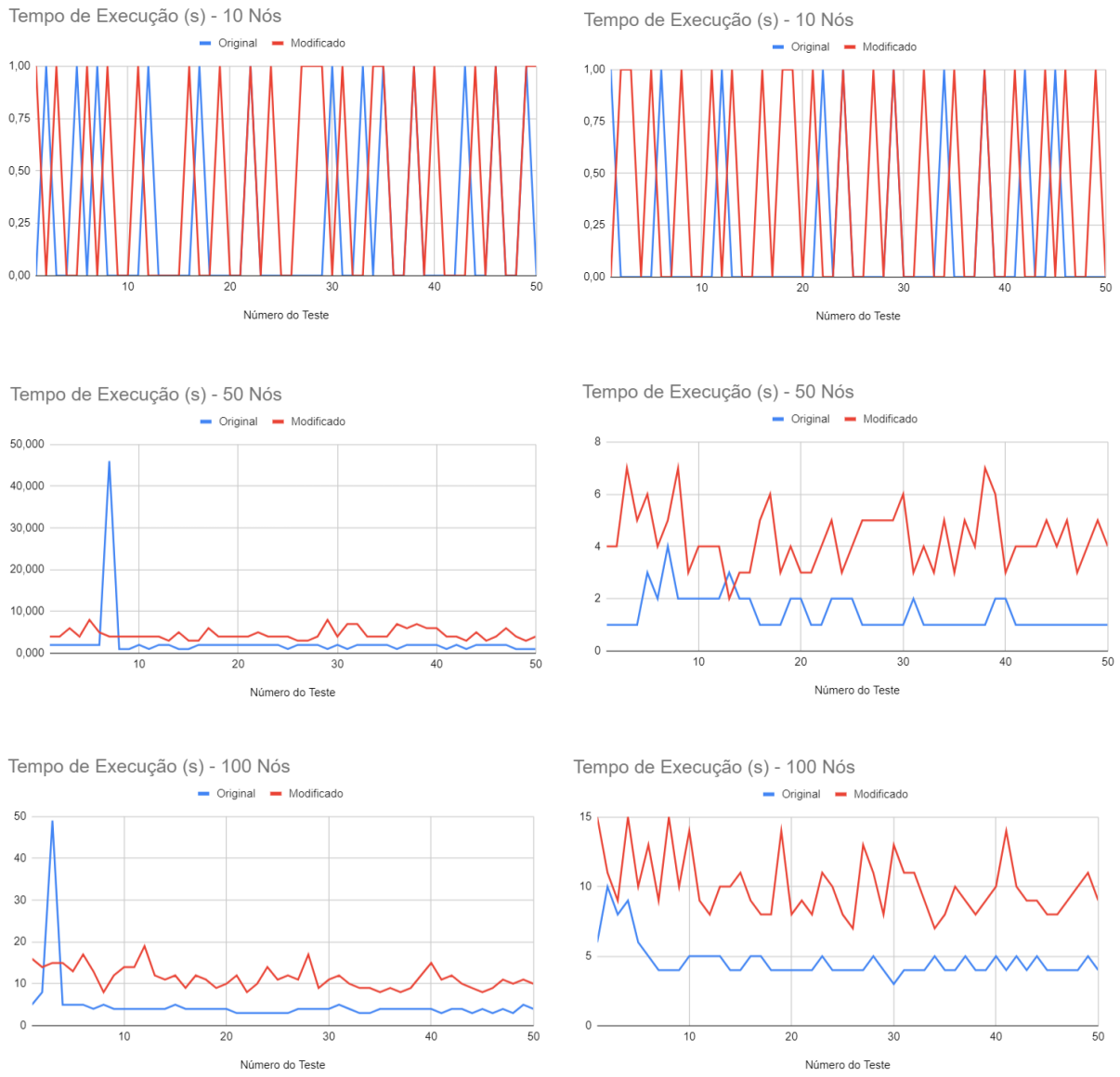
Fonte: Elaborado pelo autor

4.1.5 Tempo Total da Execução

O tempo total de execução foi calculado a partir das tomadas de tempo que o código da simulação começou e terminou, realizando uma simples subtração para receber o valor de tempo gasto.

Os gráficos da figura 10 denotam os resultados dos experimentos para obter o tempo de execução. Os tempos não variam muito entre ambientes e quantidade de nós, entretanto, uma constante é que o algoritmo modificado, consistentemente, leva mais tempo que o algoritmo original, devido à natureza da DFS, visto que ele explora apenas um caminho por vez, entretanto, nos ambientes de 10 e 50 nós, a diferença de tempo não é na mesma proporção que nos ambientes de 100 nós, ultrapassando os 10 segundos.

Figura 10 – Gráfico representando o Tempo Total de execução das simulações



Fonte: Imagens elaboradas pelo autor

Na figura, os gráficos da esquerda representam os testes do grupo de controle, enquanto que os gráficos à direita representam os experimentos do grupo de testes. Como dito anteriormente, os gráficos com 10 nós apresentam tempo bons, enquanto os de 50 nós são apenas aceitáveis.

O quadro 6 serve como ajuda em uma comparação palpável entre os diferentes experimentos, apresentando as médias entre todos os tempos medidos.

Quadro 6 – Média dos resultados dos experimentos para o tempo de execução

Tempo de Execução		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original - Controle	0,260
	Modificado - Controle	0,420
	Original - Testes	0,200
	Modificado - Testes	0,400
50 Nós	Original - Controle	2,600
	Modificado - Controle	4,560
	Original - Testes	1,460
	Modificado - Testes	4,320
100 Nós	Original - Controle	4,900
	Modificado - Controle	11,460
	Original - Testes	4,660
	Modificado - Testes	10,060

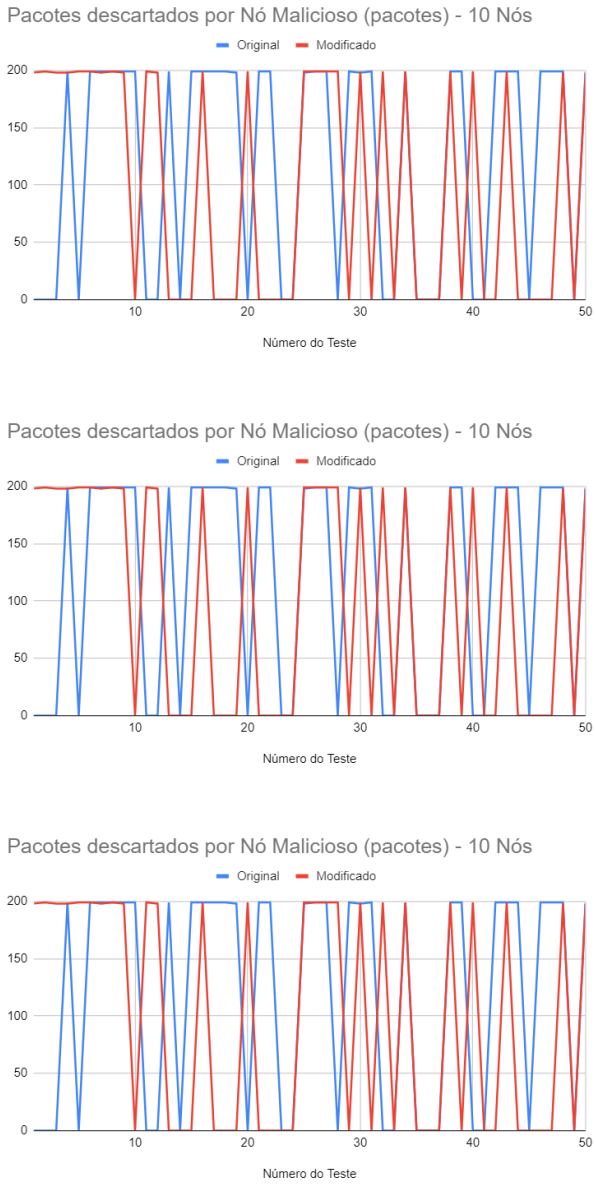
Fonte: Elaborado pelo autor

4.1.6 Efetividade do Nó Malicioso

Especificamente nos ambientes sob ataque, foi medida a quantidade de pacotes que o nó malicioso descartava durante a operação da rede. Para isso, foram contabilizados os pacotes que eram descartados com um tipo de mensagem específica.

A figura 11, e seus gráficos, mostra que o nó malicioso foi muito bem sucedido nos ambientes com o código original, descartando a grande maioria dos pacotes, especialmente a partir de 50 nós, enquanto isso, nos ambientes com o algoritmo modificado, ele ainda conseguiu relativo sucesso, embora menor.

Figura 11 – Gráficos representando o descarte de pacotes do Nó Malicioso



Fonte: Imagens elaboradas pelo autor

Os gráficos mostram que o algoritmo teve sucesso, embora pouco, em evitar o ataque, especialmente conforme a quantidade de nós aumentou.

Para uma comparação concreta entre os dois algoritmos, o quadro 7 apresenta as médias dos pacotes descartados.

Quadro 7 – Média dos resultados dos experimentos para o tempo de execução

Pacotes descartados por Nó Malicioso		
Quantidade de nós	**Testes**	**Medidas**
10 Nós	Original	119,320
	Modificado	99,340
50 Nós	Original	194,820
	Modificado	71,560
100 Nós	Original	198,940
	Modificado	43,700

Fonte: Elaborado pelo autor

4.1.7 De Maneira Geral

Como mostrado pelas figuras das subseções anteriores, o algoritmo modificado foi, de maneira geral, mais lento e menos confiável que o código original, entretanto, ele ainda conseguiu evitar alguns dos ataques, além de, também, manter-se bem conservador no gasto de energia.

4.2 Análise e Discussão dos Resultados

Primeiramente, os resultados mostram, de maneira geral, que o algoritmo modificado é pior que o algoritmo original, exceto nos casos com ataques, como visto na PDR decrescente dos pacotes de dados, no aumento do tempo de execução e da latência. A única das áreas que se mantém, relativamente, competitiva é no consumo de energia, que surpreendentemente, se manteve relativamente similar ao longo de todos os testes, mesmo tendo enviado mais pacotes, entretanto, este comportamento pode ser apenas um reflexo do aumento da quantidade de nós.

Além disso, alguns dos resultados permitem reflexões para o código desenvolvido para este estudo. Em alguns testes da versão modificada, especialmente nas redes menores, o algoritmo acabava não utilizando os pacotes RREQ, isso se deve, provavelmente, a como foi implementada a DFS no código: ela depende da fila de vizinhos, implementada pelo próprio AOMDV, entretanto, normalmente esta fila não é preenchida até que já exista um caminho válido. No algoritmo original, isto não é problema, visto que os caminhos são encontrados através de *broadcasts*, entretanto, para direcionar os pacotes de maneira direta, era necessário ter os vizinhos mapeados, o que levou à situação atual.

Outro resultado interessante a ser analisado é a questão dos pacotes RREQ e dos pacotes de dados. Em alguns testes, mesmo que haja a transmissão de pacotes

RREQ, a PDR dos pacotes de dados é baixíssima, ou é, de fato, zero. Isto indica alguns possíveis erros: erro no código, erro durante o teste, ou erro na contagem. O erro no código seria como o algoritmo interpreta a recepção de pacotes RREQ, especificamente no nó-destino. Pode ser, também, que algum problema esteja no código que simula os ambientes, devolvendo resultados errôneos, entretanto, a chance deste ser o motivo é baixa, tendo em vista que isto não acometeu o código original. Por fim, pode ser que a avaliação dos resultados do teste esteja incorreta, novamente, uma chance bastante baixa. Portanto, infere-se que a causa mais provável esteja no código modificado do AOMDV.

Por fim, com o comportamento do código na presença de um nó malicioso mostra um pouco de promessa, tendo em vista que, algumas vezes, ele realizou a ideia proposta neste estudo: entregar os pacotes para o nó destino, mesmo que haja um nó malicioso próximo, na rede. E, enquanto estas ocasiões ainda foram relativamente raras, elas, ainda sim, devem ser levadas em consideração, tendo em vista que o algoritmo original não teve nenhum caso com entrega expressiva dos pacotes de dados para o destino.

5 Conclusão

5.1 Conclusão

Os Ataques Buraco Negro são um tipo de ataque cibernético, relativamente simples de executar, tendo em vista que seu alvo, as MANETs, são bastante maleáveis e fáceis de conectar. Sendo assim, é simples se passar por um nó legítimo e, como forma de ataque, capturar e descartar todo o tráfego de pacotes de dados de determinada rede. Tudo isso é possível devido ao algoritmo de roteamento das MANETs, que busca ter maior velocidade e eficiência, do que segurança. Nesse sentido, é importante estudar formas alternativas de encontrar os caminhos através dos nós da rede, neste caso, foram utilizadas as abordagens da DFS e dos múltiplos caminhos para evitar as consequências do ataque, mas não evitar que o ataque ocorra.

Apesar de já existirem algoritmos de roteamento baseados em DFS, como Dabideen e Garcia-Luna-Aceves (2009) e Swami e Singh (2015), o viés de novidade é unir este aspecto com a utilização de múltiplos caminhos para evitar as principais consequências de um Ataque Buraco Negro, especialmente a negação de serviço.

Embora seja uma ideia promissora, a implementação do código para a realização de testes não atingiu as expectativas, sendo uma implementação pouco refinada. Assim, os testes que resultaram dos experimentos com este algoritmo modificado devem ser levados em consideração com cuidado e contexto.

Com isso, conclui-se que o objetivo final foi atingido, pois a ideia dos múltiplos caminhos, somada à DFS, foi explorada, entretanto a execução desta exploração usufruiria de um segundo olhar posteriormente, com ajustes no código e um conjunto de testes mais robustos para ser mais concreto.

5.2 Trabalhos Futuros

Para o futuro, é válido que, primeiramente, seja construído um código próprio para a realização de mais testes, isto se deve ao fato que, um algoritmo especializado nos pontos de interesse, isto é, a DFS e a utilização de múltiplos caminhos, acabaria por ser mais enxuto e teria, como consequência um melhor desempenho.

Além disso, para mais trabalhos na área, além do refinamento do código, é interessante apresentar maior robustez nos experimentos. Os realizados durante este estudo foram de valia, entretanto, mais conexões e mais nós atacantes são necessários para levar o código ao limite. Esta é uma situação relativamente realista, tendo em vista

que a característica principal das MANETs é sua maleabilidade.

Referências

ABDEL-FATTAH, F.; FARHAN, K. A.; AL-TARAWNEH, F. H.; ALTAMIMI, F. Security challenges and attacks in dynamic mobile ad hoc networks manets. IEEE, p. 28–33, 2019.

ALAMSYAH, A. n.; AMIR, A.; SUBITO, M.; FAUZI, R.; AMIRULLAH. Performance analysis of breadth-first search and depth-first search on manet for health monitoring system. IOP Publishing, 2020.

ASCENSIO, A. F. G.; ARAÚJO, G. S. de. *Estrutura de Dados: algoritmos, análise da complexidade e implementações em Java e C++*. [S.l.]: Pearson Prentice Hall, 2010.

BANG, A.; RAMTEKE, P. Manet: History, challenges and applications. 09 2013.

CHHABRA, M.; GUPTA, B.; ALMOMANI, A. A novel solution to handle ddos attack in manet. *Journal of Information Security*, SciRes, v. 4, n. 3, p. 165–179, 2013.

DABIDEEN, S.; GARCIA-LUNA-ACEVES, J. Owl: Towards scalable routing in manets using depth-first search on demand. In: *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. [S.l.: s.n.], 2009. p. 583–592.

HAMILTON, W. L. Graph representation learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan and Claypool, v. 14, n. 3, p. 1–159, 2020.

KARTHIKEYAN, B.; KANIMOZHI, N.; GANESH, S. H. Analysis of reactive aodv routing protocol for manet. In: *2014 World Congress on Computing and Communication Technologies*. [S.l.: s.n.], 2014. p. 264–267.

MIRZA, S.; LÓPEZ BAKSHI, S. Z. introduction to manet. *International Research Journal of Engineering and Technology (IRJET)*, IRJET, v. 5, n. 1, p. 17–20, 2018.

RAJAN, M. A.; CHANDRA, M. G.; REDDY, L. C.; HIREMATH, P. Concepts of graph theory relevant to ad-hoc networks. *Int. J. of Computers III*, v. 3, p. 465–469, 01 2008.

RAMPHULL, D.; MUNGUR, A.; ARMOOGUM, S.; PUDARUTH, S. A review of mobile ad hoc network (manet) protocols and their applications. *Proceedings of the Fifth International Conference on Intelligent Computing and Control Systems*, p. 204–211, 2021.

RANI, P.; KAVITA; VERMA, S.; NGUYEN, G. N. Mitigation of black hole and gray hole attack using swarm inspired algorithm with artificial neural network. *IEEE Access*, IEEE, v. 8, p. 121755–121764, 2020.

SHURMAN, M.; YOO, S.-M.; PARK, S. Black hole attack in mobile ad hoc networks. ACM, p. 96–97, 2004.

SWAMI, B.; SINGH, R. Performance analysis of dfs based ordered walk learning routing protocol in manet. IEEE, p. 195–198, 2015.

TIAN, Y.; HOU, R. An improved aomdv routing protocol for internet of things. In: *2010 International Conference on Computational Intelligence and Software Engineering*. [S.l.: s.n.], 2010. p. 1–4.

TSENG, F.-H.; CHOU, L.-D.; CHAO, H.-C. A survey of black hole attacks in wireless mobile ad hoc networks. *Human-centric Computing and Information Sciences*, SpringerOpen, v. 1, 2011.

TUTEJA, A.; GUJRAL, R.; THALIA, S. Comparative performance analysis of dsdv, aodv and dsr routing protocols in manet using ns2. IEEE, p. 330–333, 2010.

YUAN, Y.; CHEN, H.; JIA, M. An optimized ad-hoc on-demand multipath distance vector(aomdv) routing protocol. In: *Asia-Pacific Conference on Communications*. [S.l.: s.n.], 2005. p. 569–573.

APÊNDICE A – Código TCL do Grupo de Controle

```
# Definir a quantidade de nós
set num_nodes 10

# Inicia a contagem de tempo
set initial_time [clock seconds]

# Configurações da simulação
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              500      ;# Largura do cenário ajustada
set val(y)              500      ;# Altura do cenário ajustada
set val(ifqlen)         50       ;# Tamanho da fila IFQ
set val(seed)           1.0
set val(adhocRouting)   AOMDV    ;# Protocolo de roteamento
set val(nn)             $num_nodes
set val(stop)           100.0    ;# Tempo de simulação

# Energia inicial (em Joules)
set val(energy) 100.0

# Inicializar o simulador NS-2
set ns [new Simulator]

# Criaçao dos arquivos trace
set tracefd [open out.tr w]
set namtrace [open out.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# Arquivo de Energia
set energyt [open et.tr w]

# Arquivo de Tempo
set timert [open timer.tr w]

# Arquivo de nós origem e destino
set srct [open srct.tr w]
set dstt [open dstt.tr w]

# Definir o tipo de antena e protocolo de roteamento
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

# Configuração da camada de canal e mobilidade
create-god $val(nn)
$ns node-config -adhocRouting $val(adhocRouting) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
```

```
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-energyModel "EnergyModel" \
-initialEnergy $val(energy) \
-txPower 0.5 \
-rxPower 0.3 \
-idlePower 0.1 \
-sleepPower 0.05 \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace ON

# Criação dos nós com modelo de energia
set nodes {}
for {set i 0} {$i < $val(nn) - 1} {incr i} {
        set node [$ns node]
        $node random-motion 0   ;# Desabilitar movimento dos nós
        $node set X_ [expr rand() * $val(x)]
        $node set Y_ [expr rand() * $val(y)]
        $node set Z_ 0.0
        lappend nodes $node
        $ns initial_node_pos $node 20    ;# Atualizar posiçao do nó para o NAM
}

# Função para escolher dois nós aleatórios distintos
proc choose_random_pair {nodes_list} {
        set src [lindex $nodes_list [expr int(rand() * [llength $nodes_list])]]
        set dst $src
        while {$dst == $src} {
                set dst [lindex $nodes_list [expr int(rand() * [llength $nodes_list])]]
        }
        return [list $src $dst]
}

# Escolher um par de nós aleatórios e criar uma única conexao
set pair [choose_random_pair $nodes]
set src_node [lindex $pair 0]
set dst_node [lindex $pair 1]

proc create_connection {ns src dst} {
        set udp [new Agent/UDP]
        set null [new Agent/Null]
        $ns attach-agent $src $udp
        $ns attach-agent $dst $null

        set cbr [new Application/Traffic/CBR]
        $cbr set packetSize_ 512
        $cbr set interval_ 0.5
        $cbr attach-agent $udp

        $ns connect $udp $null
        $ns at 0.5 "$cbr start"
}

# Criar a única conexão entre os dois nós escolhidos
create_connection $ns $src_node $dst_node

# Calcular a média exata das posições entre src_node e dst_node
```

```
set src_x [$src_node set X_]
set src_y [$src_node set Y_]
set dst_x [$dst_node set X_]
set dst_y [$dst_node set Y_]
set midpoint_x [expr ($src_x + $dst_x) / 2.0]
set midpoint_y [expr ($src_y + $dst_y) / 2.0]

# Adicionar o novo nó na média exata entre os nós src e dst
set last_node [$ns node]
$last_node set X_ $midpoint_x
$last_node set Y_ $midpoint_y
$last_node set Z_ 0.0
$ns initial_node_pos $last_node 20

# Imprimir o número dos nós escolhidos
puts $srct "_[$src_node id]_"
puts $dstt "_[$dst_node id]_"

#puts "Posições do nó 1: [$src_node set X_] [$src_node set Y_]"
#puts "Posições do nó 2: [$dst_node set X_] [$dst_node set Y_]"
#puts "Posições do nó 3: [$last_node set X_] [$last_node set Y_]"

# Configurar a finalização da simulação
$ns at $val(stop) "stop"
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
proc stop {} {
        global ns tracefd namtrace nodes energyt initial_time timert srct dstt

        # Relatório final de energia
        foreach node $nodes {
                set energy [$node energy]
                puts $energyt "$energy Joules"
        }

        set final_time [clock seconds]

        set execution_time [expr {$final_time - $initial_time}]

        puts $timert "$execution_time"

        $ns flush-trace
        close $srct
        close $dstt
        close $tracefd
        close $namtrace
        close $energyt

        exit 0
}

# Iniciar a simulação
$ns run
```

APÊNDICE B – Código TCL do Grupo de Testes

```
# Definir a quantidade de nós
set num_nodes 10

# Starta a contagem de tempo
set initial_time [clock seconds]

# Configurações da simulação
set val(chan)           Channel/WirelessChannel
set val(prop)           Propagation/TwoRayGround
set val(netif)          Phy/WirelessPhy
set val(mac)            Mac/802_11
set val(ifq)            Queue/DropTail/PriQueue
set val(ll)             LL
set val(ant)            Antenna/OmniAntenna
set val(x)              300      ;# Largura do cenário ajustada
set val(y)              300      ;# Altura do cenário ajustada
set val(ifqlen)         50       ;# Tamanho da fila IFQ
set val(seed)           1.0
set val(adhocRouting)   AOMDV    ;# Protocolo de roteamento
set val(nn)             $num_nodes
set val(stop)           100.0    ;# Tempo de simulação

# Energia inicial (em Joules)
set val(energy) 100.0

# Inicializar o simulador NS-2
set ns [new Simulator]

# Criação de traços
set tracefd [open out.tr w]
set namtrace [open out.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

# Arquivo de Energia
set energyt [open et.tr w]

# Arquivo de Tempo
set timert [open timer.tr w]

# Definir o tipo de antena e protocolo de roteamento
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

# Configuração da camada de canal e mobilidade
create-god $val(nn)
$ns node-config -adhocRouting $val(adhocRouting) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
```

```
-channelType $val(chan) \
-topoInstance $topo \
-energyModel "EnergyModel" \
-initialEnergy $val(energy) \
-txPower 0.5 \
-rxPower 0.3 \
-idlePower 0.1 \
-sleepPower 0.05 \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace ON

# Criação dos nós com modelo de energia
set nodes {}
for {set i 0} {$i < $val(nn) - 1} {incr i} {
        set node [$ns node]
        $node random-motion 0   ;# Desabilitar movimento dos nós
        $node set X_ [expr rand() * $val(x)]
        $node set Y_ [expr rand() * $val(y)]
        $node set Z_ 0.0
        lappend nodes $node
        $ns initial_node_pos $node 20    ;# Atualizar posição do nó para o NAM
}

# Função para escolher dois nós aleatórios distintos
proc choose_random_pair {nodes_list} {
        set src [lindex $nodes_list [expr int(rand() * [llength $nodes_list])]]
        set dst $src
        while {$dst == $src} {
                set dst [lindex $nodes_list [expr int(rand() * [llength $nodes_list])]]
        }
        return [list $src $dst]
}

# Escolher um par de nós aleatórios e criar uma única conexão
set pair [choose_random_pair $nodes]
set src_node [lindex $pair 0]
set dst_node [lindex $pair 1]

proc create_connection {ns src dst} {
        set udp [new Agent/UDP]
        set null [new Agent/Null]
        $ns attach-agent $src $udp
        $ns attach-agent $dst $null

        set cbr [new Application/Traffic/CBR]
        $cbr set packetSize_ 512
        $cbr set interval_ 0.5
        $cbr attach-agent $udp

        $ns connect $udp $null
        $ns at 0.5 "$cbr start"
}

# Criar a única conexão entre os dois nós escolhidos
create_connection $ns $src_node $dst_node

# Calcular a média exata das posições entre src_node e dst_node
set src_x [$src_node set X_]
set src_y [$src_node set Y_]
set dst_x [$dst_node set X_]
set dst_y [$dst_node set Y_]
```

```
set midpoint_x [expr ($src_x + $dst_x) / 2.0]
set midpoint_y [expr ($src_y + $dst_y) / 2.0]

# Adicionar o novo nó malicioso na média exata entre os nós src e dst
set malicious_node [$ns node]
$malicious_node set X_ $midpoint_x
$malicious_node set Y_ $midpoint_y
$malicious_node set Z_ 0.0
$ns initial_node_pos $malicious_node 20

$ns at 0.0 "$malicious_node color red"
$malicious_node color "red"
$ns at 0.0 "[$malicious_node set ragent_] malicious"

# Imprimir o número dos nós escolhidos
puts "Nó de origem escolhido aleatoriamente: [$src_node id]"
puts "Nó de destino escolhido aleatoriamente: [$dst_node id]"
puts "Nó malicioso adicionado na média das posições: [$malicious_node id]"

#puts "Posicoes do no 1: [$src_node set X_] [$src_node set Y_]"
#puts "Posicoes do no 2: [$dst_node set X_] [$dst_node set Y_]"
#puts "Posicoes do no 3: [$malicious_node set X_] [$malicious_node set Y_]"

# Configurar a finalização da simulação
$ns at $val(stop) "stop"
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
proc stop {} {
        global ns tracefd namtrace nodes energyt initial_time timert

        # Relatório final de energia
        foreach node $nodes {
                set energy [$node energy]
                puts $energyt "$energy Joules"
        }

        set final_time [clock seconds]

        set execution_time [expr {$final_time - $initial_time}]

        puts $timert "$execution_time"

        $ns flush-trace
        close $tracefd
        close $namtrace
        close $energyt

        exit 0
}

# Iniciar a simulação
$ns run
```