

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO**

**DAVI HUGO MARQUES PONTES
CAIO CHAVES BEZERRA ROCHA**

**SIMULADOR DE REDES RÁDIO HETEROGÊNEAS BASEADO NO
MININET-WIFI**

**RIO DE JANEIRO
2022**

DAVI HUGO MARQUES PONTES
CAIO CHAVES BEZERRA ROCHA

SIMULADOR DE REDES RÁDIO HETEROGÊNEAS BASEADO NO
MININET-WIFI

Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador(es): Maria Cláudia Reis Cavalcanti, D.Sc.
David Fernandes Moura, D.C.
Julio Cesar Cardoso Tesolin, M.Sc

Rio de Janeiro
2022

©2022

INSTITUTO MILITAR DE ENGENHARIA

Praça General Tibúrcio, 80 – Praia Vermelha

Rio de Janeiro – RJ CEP: 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmар ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

Marques Pontes, Davi Hugo; Chaves Bezerra Rocha, Caio.

Simulador de redes rádio heterogêneas baseado no MININET-WIFI / Davi Hugo Marques Pontes e Caio Chaves Bezerra Rocha. – Rio de Janeiro, 2022.
79 f.

Orientador(es): Maria Cláudia Reis Cavalcanti, David Fernandes Moura e Julio Cesar Cardoso Tesolin.

Projeto de Final de Curso (graduação) – Instituto Militar de Engenharia, Engenharia de Computação, 2022.

1. mininet. 2. wifi. 3. modelo de mobilidade. 4. emulador. 5. movimentação militar. 6. RDS. 7. cenário Anglova. i. Cavalcanti, Maria Cláudia Reis (orient.) ii. Moura, David Fernandes (orient.) iii. Cardoso Tesolin, Julio Cesar (orient.) iv. Título

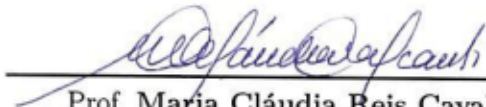
**DAVI HUGO MARQUES PONTES
CAIO CHAVES BEZERRA ROCHA**

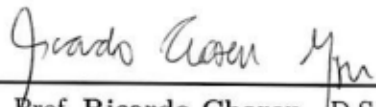
**Simulador de redes rádio heterogêneas baseado no
MININET-WIFI**


Projeto de Final de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia, como requisito parcial para a obtenção do título de Bacharel em Engenharia de Computação.

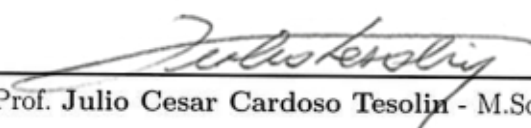
Orientador(es): Maria Cláudia Reis Cavalcanti, David Fernandes Moura e Julio Cesar Cardoso Tesolin.

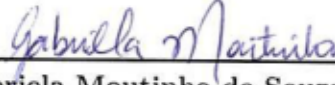
Aprovada em 19 de outubro de 2022, pela seguinte banca examinadora:


Prof. Maria Cláudia Reis Cavalcanti - D.Sc. do IME - Presidente


Prof. Ricardo Choren - D.Sc. do IME


Cel. David Fernandes Cruz Moura - D.C. do CTEX


Prof. Julio Cesar Cardoso Tesolin - M.Sc. do IME


Maj. Gabriela Moutinho de Souza Dias - D.Sc. do IME

Rio de Janeiro
2022

*Este trabalho é dedicado aos nossos pais.
Sem eles nada seria possível.*

AGRADECIMENTOS

Em primeiro lugar, a Deus, que fez com que nossos objetivos fossem alcançados, durante todos os nossos anos de estudos.

A todas as pessoas que nos incentivaram, apoiaram e possibilitaram esta oportunidade de ampliar nosso conhecimento e nossos horizontes.

A todos os amigos, familiares e mestres que sempre estiveram presentes e acreditaram na realização de nossos objetivos. Em especial a Professora Orientadora Dr. Maria Cláudia Cavalcanti, ao Cel David Moura, ao Júlio Tesolin e ao André Demori pela disponibilidade e atenção fora do comum.

*“Se você não sabe aonde quer ir,
qualquer caminho serve.
(Lewis Carroll)”*

RESUMO

Este trabalho está relacionado à iniciativas de Pesquisa e Desenvolvimento na área de redes de comunicação sem fio, destinando-se a auxiliar pesquisas existentes e futuras no estudo da movimentação dos nós pertencentes à redes sem fio voltado a cenários de manobras militares. Esse trabalho foi desenvolvido com o objetivo de estender o emulador Mininet-WiFi para prover meios de simular a movimentação de nós de forma estruturada e compatível com a realidade das operações militares. Dessa forma, foi introduzido o conceito de grupos de nós, nos quais cada grupo poderia executar um padrão de movimentação independente. Determinados métodos do Mininet-Wifi foram sobrescritos para permitir a escolha da duração dos padrões de mobilidade, com isso foi possível gerar novos padrões por meio da combinação de padrões mais simples. Por fim, foram feitos testes com movimentações previstas na doutrina do manual de cavalaria mecanizada vigente, os quais foram validados junto aos interessados no projeto, para serem disponibilizados para uso.

Palavras-chave: mininet. wifi. modelo de mobilidade. emulador. movimentação militar. RDS. cenário Anglova.

ABSTRACT

This work is related to Research and Development initiatives in the area of wireless communication networks, aiming to assist existing and future research in the study of the movement of nodes belonging to wireless networks aimed at scenarios of military maneuvers. This work was developed with the objective of extending the Mininet-WiFi emulator to provide means of simulating the movement of nodes in a structured way that is compatible with the reality of military operations. In this way, the concept of groups of nodes was introduced, in which each group could execute an independent movement pattern. Certain methods of Mininet-Wifi were overwritten to allow the choice of duration of mobility patterns, with this it was possible to generate new patterns by combining simpler ones. Finally, tests were carried out with movements showed in the current doctrine manual of the mechanized cavalry, which were validated with the projects stakeholders.

Keywords: mininet. wifi. mobility model. emulator. military movimentation. RDS. Angola scenario.

LISTA DE ILUSTRAÇÕES

Figura 1 – Representação do meios de comunicação em uma operação militar (1) .	13
Figura 2 – Movimentação das tropas Anglova (1)	14
Figura 3 – Rede WiFi modo <i>ad hoc</i> (2)	19
Figura 4 – Rede WiFi modo infraestruturado (2)	19
Figura 5 – Comparação entre plataformas experimentais para redes sem fio (2) . .	22
Figura 6 – Topologia instanciada ao se inicia o Mininet-WiFi (2)	23
Figura 7 – Diagrama de classes envolvidas na movimentação no Mininet-WiFi . .	26
Figura 8 – Diagrama de classes das entidades de movimentação do Mininet-WiFi .	26
Figura 9 – Diagrama de classes dos modelos de mobilidade randômicos	27
Figura 10 – Classificação de modelos de mobilidade (3)	29
Figura 11 – Movimentação em linha (4)	31
Figura 12 – Movimentação em coluna (4)	31
Figura 13 – Movimentação em cunha (4)	31
Figura 14 – Emprego do RC Mec na incursão (4)	32
Figura 15 – Representação simplificada da classe Node	35
Figura 16 – Representação simplificada da classe ReplayingMobility	35
Figura 17 – Fluxo de configuração da mobilidade	37
Figura 18 – Fluxo de inicialização da mobilidade	39
Figura 19 – Fluxo de mobilidade	40
Figura 20 – Diagrama de classes simplificado do Mininet-Wifi	41
Figura 21 – Diagrama de classes da solução proposta	44
Figura 22 – Novo fluxo de configuração da mobilidade	46
Figura 23 – Fluxo de mobilidade	48
Figura 24 – Gráfico de velocidade do nó	51
Figura 25 – Representação do cálculo da posição alcançável pelo nó	53
Figura 26 – Calculo da posição ponderada do nó	55
Figura 27 – Modelo de mobilidade de nós em formação em linha	58
Figura 28 – Modelo de mobilidade de nós em formação em cunha	60
Figura 29 – Modelo de mobilidade de nós em formação circular	62
Figura 30 – Posição final dos elementos do grupo 1 no teste.	70
Figura 31 – Teste de conexão entre nós do grupo 1.	71
Figura 32 – Disposição espacial dos elementos da simulação em um instante	72
Figura 33 – Representação gráfica das 7800 coordenadas que representam o movimento do nó.	73
Figura 34 – Representação gráfica das 100 coordenadas escolhidas para representar o movimento do nó.	74

LISTA DE ABREVIATURAS E SIGLAS

WiFi	<i>Wireless Fidelity</i>
IEEE	<i>Institute of Electrical and Electronics Engineers</i>
MAC	<i>Media Access Control</i>
WWAN	<i>Wireless Wide Area Network</i>
WLAN	<i>Wireless Local Area Network</i>
WPAN	<i>Wireless Personal Area Network</i>
GSM	<i>Groupe Special Mobile</i>
SDN	<i>Software Defined Network</i>
MAME	<i>Multiple Arcade Machine Emulator</i>
CLI	<i>Control Line Interface</i>
ICMP	<i>Internet Control Message Protocol</i>
IP	<i>Internet Protocol</i>
C2	Comando e Controle
VE	Verificação Especial
VC	Verificação Corrente
VF	Verificação Final

SUMÁRIO

1	INTRODUÇÃO	13
1.1	MOTIVAÇÃO	13
1.2	OBJETIVO	13
1.3	JUSTIFICATIVA	14
1.4	METODOLOGIA	15
1.5	CONTRIBUIÇÕES	15
1.6	ESTRUTURA	15
2	CONTEXTUALIZAÇÃO	17
2.1	REDES	17
2.1.1	REDES DE COMUNICAÇÃO SEM FIO	18
2.1.2	REDES DE COMUNICAÇÃO MILITARES	20
2.2	MININET-WIFI	21
2.2.1	SIMULAÇÃO E EMULAÇÃO	21
2.2.2	FUNCIONAMENTO BÁSICO	23
2.2.3	FUNCIONAMENTO MOVIMENTAÇÃO	24
2.3	MOVIMENTAÇÃO	27
2.3.1	MODELOS DE MOBILIDADE	28
2.3.2	MOVIMENTAÇÃO MILITAR	30
3	ABORDAGEM PROPOSTA	33
3.1	VISÃO GERAL	33
3.2	LEVANTAMENTO DE REQUISITOS	34
3.3	MOVIMENTAÇÃO NO MININET-WIFI	34
3.3.1	CONFIGURAÇÃO DA MOBILIDADE	36
3.3.2	INICIALIZAÇÃO DA MOBILIDADE	38
3.3.3	FLUXO DE MOBILIDADE	40
3.4	ARQUITETURA	41
3.4.1	MUDANÇAS ESTRUTURAIS	43
3.4.1.1	ADDGROUP	43
3.4.1.2	ADDSTATION	45
3.4.1.3	START_MOBILITY	45
3.4.1.4	GET_MOBILITY_PARAMS	46
3.4.1.5	PAUSE	46
3.4.2	CONFIGURAÇÃO DE MOBILIDADE DA SOLUÇÃO	46
3.4.3	FLUXO DE MOBILIDADE DA SOLUÇÃO	48

4	DESENVOLVIMENTO	50
4.1	FUNÇÕES AUXILIARES	50
4.1.1	INCREASED_SPEED	50
4.1.1.1	PARÂMETROS	50
4.1.1.2	DESCRIÇÃO	51
4.1.2	PROJECT_POSITION	51
4.1.2.1	PARÂMETROS	52
4.1.2.2	DESCRIÇÃO	52
4.1.3	WEIGHTED_POSITION	54
4.1.3.1	PARÂMETROS	54
4.1.3.2	DESCRIÇÃO	54
4.2	MODELOS DE MOBILIDADE	55
4.2.1	FORMAÇÃO EM LINHA	55
4.2.1.1	PARÂMETROS	57
4.2.1.2	DESCRIÇÃO	57
4.2.2	FORMAÇÃO EM CUNHA	58
4.2.2.1	PARÂMETROS	59
4.2.2.2	DESCRIÇÃO	59
4.2.3	FORMAÇÃO CIRCULAR	60
4.2.3.1	PARÂMETROS	62
4.2.3.2	DESCRIÇÃO	62
5	TESTES	63
5.1	FORMAÇÃO LINHA	63
5.2	FORMAÇÃO CUNHA	65
5.3	FORMAÇÃO CIRCULAR	66
5.4	MOVIMENTAÇÃO COMPOSTA	68
5.5	REPRESENTAÇÃO ANGLOVA	71
6	CONCLUSÕES E TRABALHOS FUTUROS	77
	REFERÊNCIAS	79

1 INTRODUÇÃO

1.1 Motivação

Meios de comunicações táticos resilientes, atualizados e seguros são de extrema importância para garantir o sucesso de diversas missões dentro do Exército Brasileiro. Dessa forma, surgiu a necessidade de desenvolver ferramentas para testar e analisar a performance dos meios de comunicação, bem como desenvolver pesquisas visando não só o aperfeiçoamento dos dispositivos de comunicação, mas também a lapidação da doutrina no uso desses dispositivos.

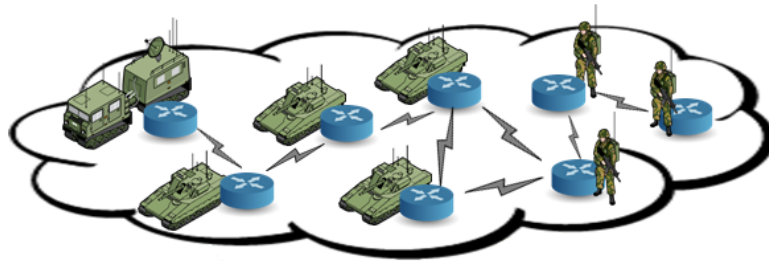


Figura 1 – Representação do meios de comunicação em uma operação militar (1)

Testar e analisar o comportamento de um rede de forma real dentro de uma operação militar é bastante complexo, necessitando de hardwares, topologias de rede e recursos humanos. Sendo assim uma boa alternativa para realizar pesquisas inclui simulações computacionais baseadas em experimentos, as quais possuem um baixo investimento e uma alta escalabilidade. Além disso, simulações são ambientes perfeitamente controlados, o que possibilita facilmente a replicabilidade de testes (5).

Para criar esse ambiente de simulação, podemos utilizar alguns *softwares*, dentre eles o Mininet-WiFi (2), o qual é um emulador de rede que permite emular um sistema de comunicação, virtualizando pontos de acessos móveis e estações *WiFi*, a partir do entendimento do funcionamento de cada componente, de forma que o resultado seja bastante semelhante a um experimento real.

1.2 Objetivo

Possibilitar estudos de movimentações de tropas de uma forma mais fidedigna à movimentação real em campo, com uma interface amigável, flexível e familiar para usuários do ecossistema da aplicação. Para isso, foram expandidas as funcionalidades básicas do Mininet-Wifi inserindo as seguintes melhorias:

- Inserção do conceito de grupo de nós, no qual cada grupo possui um padrão de movimentação independente dos demais grupos.
- Seleção do intervalo de duração dos modelos de mobilidade que atuam em cada grupo, possibilitando que os grupos possam começar a executar um padrão de mobilidade em um determinado tempo e parar de executar em outro momento.
- Criação de um padrão de movimentação a partir de padrões mais simples, visando a flexibilidade do usuário de conseguir movimentar os nós da rede de acordo com os padrões de movimentação definidos na doutrina para elaboração de manobras militares.

1.3 Justificativa

Atualmente, o emulador de redes Mininet-WiFi é largamente utilizado para realizar experimentos em redes de comunicações, além de possuir suporte com diversas características e escopo.

Entretanto, para conseguir realizar pesquisas acerca da performance das redes de comunicações dentro de operações militares, é necessário emular a movimentação segundo a doutrina militar dos pontos de acessos, os quais representarão os diversos elementos militares responsáveis pela transmissão de dados envolvidos nas operações. Atualmente, os modelos de mobilidade disponíveis para o Mininet-WiFi não contemplam os tipos de movimentação definidos na doutrina corrente. Diante disso, faz-se necessário algumas melhorias no Mininet-WiFi.

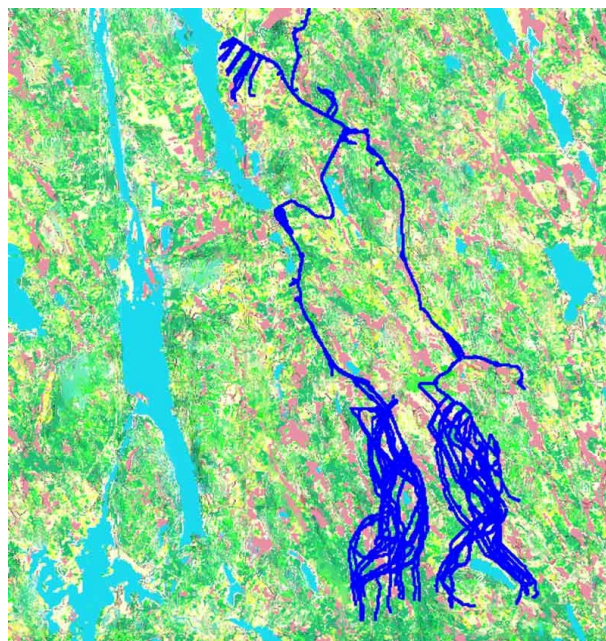


Figura 2 – Movimentação das tropas Anglova (1)

1.4 Metodologia

O projeto foi dividido em cinco etapas, as quais diferiram entre si quanto a natureza de suas atividades. A primeira etapa consistiu no aprendizado teórico relacionados a conceitos básicos de redes, redes sem fio, meios de comunicação militar, simulação de redes, formação e movimentação de tropas. Especificamente nessa etapa, na parte do estudo de simuladores de redes, vale ressaltar o estudo do emulador de redes Mininet-Wifi.

A segunda etapa tratou-se do entendimento das necessidades que o projeto devia atender e da viabilidade de executar o trabalho dentro do tempo estipulado, além da listagem de requisitos para a definição das atividades e do escopo do projeto.

A terceira etapa foi pautada na modelagem conceitual do projeto, a qual conteve o diagrama de classes utilizadas para desenvolver o módulo externo que expandiu as funcionalidades do Mininet-Wifi.

A quarta etapa se limitou à implementação do projeto em si, seguindo as boas práticas do desenvolvimento de software e o cumprimento do cronograma estipulado previamente.

Por fim, a última etapa foi destinada à elaboração e execução de testes para verificar se o sistema estava funcionando como o planejado e a validação de todos os requisitos definidos inicialmente. Além disso, foi feita a verificação da existência de algum problema de versionamento caso o Mininet-WiFi seja atualizado para uma versão mais recente.

1.5 Contribuições

Ao final do projeto, foi desenvolvido um módulo externo ao Mininet-WiFi, o qual implementou as funcionalidades de agrupamento de nós e de movimentação desses nós com dependência espacial. Com isso gerou-se um *software* capaz de auxiliar na realização de pesquisas da área de redes de comunicações militares de forma eficiente, permitindo a replicação de testes de forma fácil e com baixo custo.

1.6 Estrutura

Este documento especifica as etapas do projeto desde a concepção inicial até a implementação e validação, percorrendo todo o seu desenvolvimento. Esta especificação é feita de acordo com a estrutura descrita a seguir.

O capítulo 2 trata da fundamentação teórica que foi a base para a realização do projeto. Este capítulo abrange os conceitos de Redes de Comunicação, Mininet-WiFi e de Modelos de Mobilidade. Além disso, mais especificamente, este capítulo trata de temas importantes para a contextualização do problema como as redes de comunicação sem fio e as

redes de comunicação militares, as diferenças entre simulações e emulações, o funcionamento básico do Mininet-WiFi, conceitos de movimentação e modelos de mobilidade, chegando por fim em tópicos sobre a movimentação militar em si.

Já no capítulo 3, encontra-se a abordagem proposta do projeto, apresentando desde uma visão geral, a qual visa sumarizar os conceitos apresentados e sua convergência para a solução, até o levantamento de requisitos e a arquitetura da solução proposta.

O capítulo 4 descreve todo o desenvolvimento do projeto abrangendo tudo que foi implementado na parte de *software*, visando o cumprimento dos requisitos estipulados.

No capítulo 5 foram descritos os testes realizados a fim de validar o modelo, além de fazer a apuração do código no caso da existência de algum comportamento diferente do esperado.

Por fim, no capítulo 6, foi feita a conclusão, resumizando tudo que foi realizado nesse trabalho. Neste capítulo, também, é feito um levantamento de possíveis trabalhos futuros que corroborariam com o aperfeiçoamento e a expansão desse projeto.

2 CONTEXTUALIZAÇÃO

Para embasar o presente trabalho é importante apresentar alguns conceitos sobre redes de comunicações, mais especificamente as que envolvem comunicação sem fio, visto que este é o tipo de rede utilizada ao longo do projeto. Além disso, são citados conceitos e desafios das redes de comunicação militares, as quais possuem características únicas que as diferem das redes convencionais.

Além de entender os conceitos de redes, também se faz necessário entender um pouco sobre o funcionamento do emulador de redes Mininet-WiFi, pois ele será utilizado para o desenvolvimento do projeto. Por conta disso, neste capítulo são apresentados conceitos básicos do emulador como, por exemplo, a forma de utilização da ferramenta para criação de topologias de redes. Além disso, é demonstrado como pode ser configurado um modelo de mobilidade para os nós da rede de forma que eles possam se movimentar de forma a seguir um padrão pré-determinado.

Por fim, serão abordados conceitos de movimentação como os modelos de mobilidade amplamente utilizados no meio acadêmico e os diferentes tipos de classificação de mobilidade existentes para tais modelos. Durante a apresentação dos modelos existentes é feito um paralelo com a realidade do Mininet-WiFi para representar o que se encontra desenvolvido atualmente no mesmo. Além disso, são tratados temas de movimentação militar por meio da apresentação de termos e características que já são bem consolidadas e definidas na doutrina da Força Terrestre.

2.1 Redes

Redes de comunicação podem ser definidas como uma forma de conexão entre diversos dispositivos eletrônicos de forma que eles possam trocar recursos e dados. Tais dispositivos dentro de uma rede são denominados “nós” e estes se comunicam entre si através de um conjunto de processos e regras chamados de protocolos de rede, os quais são definidos por empresas e grupos de estudos, e acabam sendo adotados pelos fabricantes de dispositivos que desejam se conectar nas redes.

Nesta definição, os dispositivos da rede podem ser separados em dois grandes grupos: *hosts* e dispositivos de conexão. São denominados *hosts* dispositivos como grandes computadores, *desktops*, *laptops*, estações de trabalho, telefones celulares, sistemas de segurança ou servidores em geral. Tais *hosts* têm em comum a capacidade de oferecerem recursos, serviços e aplicações para usuários e outros *hosts* na rede. Já os dispositivos de conexão são os responsáveis por fazerem a interligação entre os *hosts*, podem ser citados

dispositivos como roteadores, que ligam uma rede a outras redes, *switches* que ligam dispositivos entre si, *modems* responsáveis por alterar a forma dos dados, dentre muitos outros dispositivos responsáveis por fazerem as redes funcionarem corretamente.

Tais dispositivos possuem um meio de conexão entre si, os quais são denominados meios de transmissão. Estes podem utilizar fios, como é o caso das redes cabeadas *Ethernet*, redes de cabos coaxiais e as redes de fibra óptica, ou podem transmitir dados sem a utilização de fios, como é o caso de enlaces rádio, redes WiFi e comunicação via satélite.

2.1.1 Redes de comunicação sem fio

As comunicações sem fio, de forma geral, têm sido estudadas há bastante tempo, os primeiros estudos de propagação de ondas eletromagnéticas datam do final do século XIX, mais especificamente em 1887 através de Henrich Rudolph Hertz, o qual foi responsável por elaborar um experimento que gerava ondas eletromagnéticas, as quais são amplamente utilizadas nos dias atuais para a comunicação sem fio. Apesar de tais ondas serem conhecidas há bastante tempo, os campos de estudos atrelados a elas são vários e continuam aumentando atualmente, as tecnologias desenvolvidas a partir delas vão desde comunicações de radiofonia até comunicações satelitais.

A demanda por comunicações sem fio é crescente desde sua criação, visto a grande demanda por comunicação existente na sociedade (6). Por conta disso, diversos tipos de tecnologia para comunicação sem fio foram desenvolvidas como o rádio *broadcast*, comunicações telefônicas, infravermelho, *Bluetooth*, comunicação via micro-ondas, comunicação satelital, *WiFi* (IEEE 802.11), dentre muitas outras que surgiram para atender demandas específicas de alguma aplicação ou de um grupo de usuários.

Dentre as tecnologias supracitadas vale a pena ressaltar o *WiFi*, uma das formas de comunicação sem fio mais utilizadas na atualidade e presente em diversos dispositivos como *laptops* e *smartphones*. A tecnologia *WiFi* é uma tecnologia classificada como WLAN (*Wireless Local Area Network*) baseada no padrão IEEE 802.11, sendo este o padrão de comunicação para redes sem fio mais aceito mundialmente (2). Isso se dá principalmente pela boa relação custo-benefício do protocolo. A arquitetura de uma rede no padrão 802.11 pode ser de dois tipos: modo infraestruturado ou modo *ad hoc*, o primeiro é composto de um ponto de acesso e estações sem fio denominadas clientes, já o segundo é composto apenas por clientes. A representação gráfica destas duas arquiteturas está ilustrada nas Figuras 3 e 4. Os dispositivos que utilizam o padrão 802.11 possuem um endereço MAC de 6 bytes na placa de interface de rede, o que é similar ao utilizado no padrão cabeado *Ethernet* e possibilita uma interoperabilidade mais fácil e fluida.

Apesar de o WiFi ser uma tecnologia bastante aceita e utilizada no mundo, ele ainda é restrito a um escopo de atuação pequeno se comparado ao universo das comunicações

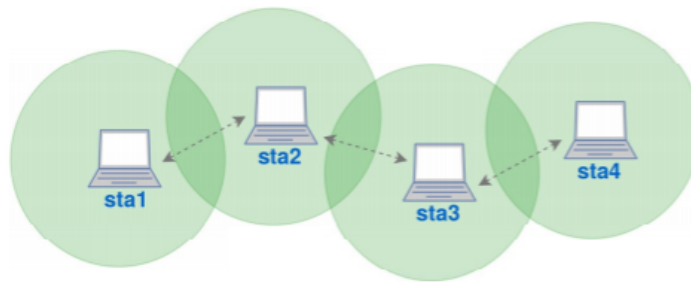
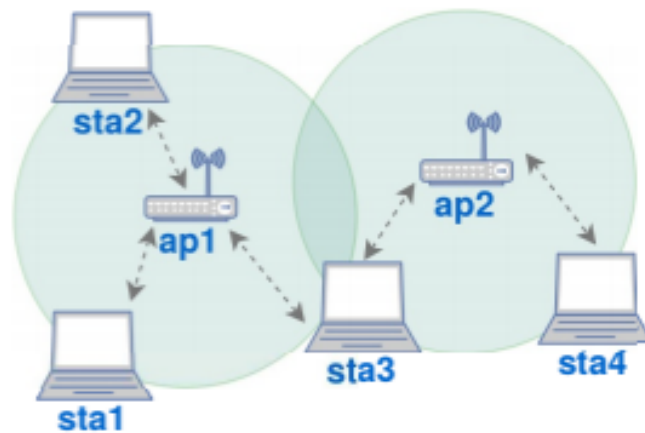
Figura 3 – Rede WiFi modo *ad hoc* (2)

Figura 4 – Rede WiFi modo infraestrutura (2)

sem fio. A utilização de tal padrão se dá em redes de curta distância e atende a problemas específicos, mas bastante comuns, de usuários e sistemas.

Para se ter um melhor entendimento do universo de comunicação sem fio supracitado é importante definir uma categorização de redes baseada na sua área de atuação física. As redes sem fio podem ser classificadas em três grandes grupos quanto às suas áreas de cobertura: rede sem fio de longa distância (WWAN), rede de área local sem fio (WLAN) e rede de área pessoal sem fio (WPAN) (7). As WWAN incluem tecnologias que possuem grande cobertura de área como é o caso de tecnologias utilizadas em redes telefônicas como GSM, 4G, 5G, Mobitex dentre outras. As WLAN englobam o caso do WiFi, HiperLan e várias outras. Por fim, as WPAN são as redes chamadas de redes pessoais e que possuem uma área de cobertura muito pequena, como é o caso das tecnologias de Bluetooth e infravermelho.

Atualmente, porém, os desafios enfrentados nas comunicações sem fio são demasiadamente complexos e por conta disso acabam dependendo de redes heterogêneas que são resultados da combinação de diferentes tecnologias de redes que podem estar inclusive em classificações distintas no que tange à área de cobertura. Para tornar tais redes mais fluidas

e para fazer com que seu funcionamento seja transparente para o usuário estão sendo implementados conceitos de redes definidas por software de forma a facilitar o controle na rede e o *handover* nas conexões (8).

Tal conceito de redes definidas por software consiste de uma abordagem que permite o controle centralizado da rede através de aplicações que não estão necessariamente localizadas nos pontos de acesso (2). A ideia de tal abordagem é dividir as responsabilidades na rede e separar o plano de controle do plano de dados, tal ação acaba por tornar a rede mais flexível e por consequência o controle se torna mais fácil (2, 8). Dessa forma, os administradores desse tipo de rede conseguem especificar o comportamento da rede de forma lógica e simplificada, através de aplicações fornecidas pelas plataformas de controle que implementam o que é chamado de *southbound* interface para dispositivos de rede. Nesse contexto, o OpenFlow é a *southbound* interface mais popular e por conta disso é utilizada no software Mininet-Wifi empregado neste trabalho.

Uma das características únicas das redes de comunicação sem fio é a utilização da propagação de ondas eletromagnéticas para difusão do sinal, isso é uma grande vantagem no quesito flexibilidade das redes porém é um grande desafio técnico dada a dificuldade de controle do meio de transmissão. Um sinal de rádio transmitido entre dois pontos sofre três tipos de fenômenos: atenuação, desvanecimento de longo prazo e desvanecimento de curto prazo (2). Tais fenômenos dificultam a correta comunicação de redes sem fio por degradar o sinal enviado da fonte para o receptor, isso apresenta um dos maiores desafios da utilização de comunicações sem fio juntamente com o problema da interferência de outras ondas eletromagnéticas.

Tais conceitos e dificuldades citados acima ilustram o vasto universo das comunicações sem fio, por conta de tudo isso a quantidade de trabalhos na área é crescente e o estudo desse tipo de comunicação se torna cada vez mais relevante.

2.1.2 Redes de comunicação militares

O ambiente de operações militares se distingue bastante do que é encontrado no dia a dia das pessoas e empresas da sociedade civil. Isso ocorre principalmente pela imprevisibilidade dos cenários e a quase certa hostilidade dos ambientes onde as Forças Armadas operam. Para operar em tais cenários é necessário haver um meio de comunicação estável e confiável durante todo o período da ação, isso é preciso pois a capacidade de comunicação nos diferentes níveis de comando é fator crucial para o melhor desempenho na conclusão das tarefas. Tal capacidade de comunicação propicia uma consciência situacional altamente desejada pelo escalão superior nas atividades militares.

Por conta dessa necessidade de comunicação, as Forças Armadas, assim como entidades do meio civil, se utilizam do advento das redes de computadores e de meios de

comunicação dos mais variados tipos para criar um modo de troca de dados e informações rápido e eficiente. Para as Forças Armadas, porém, esse desafio é maior dado o ambiente de atuação variável de tais redes e por conta da necessidade de atender 3 requisitos básicos para redes de comunicações militares: alcance, capacidade e mobilidade (9).

Desta forma, para atingir os requisitos mínimos para a tarefa, um sistema de comunicação militar deve ter uma única estrutura lógica integrada por diferentes tecnologias de enlace de redes (9). Tal ideia vai de encontro à aplicação dos conceitos de SDN, onde o controle da rede é desacoplado do fluxo de dados gerando uma maior flexibilidade na rede e possibilitando a programação do funcionamento da mesma por meio de um software centralizado, o qual aplica as políticas de roteamento desejadas e pode fazer considerações em tempo real do estado da rede (10). A aplicação desses conceitos é interessante para prover os requisitos básicos para as comunicações militares e é especialmente relevante para garantir a mobilidade das tropas enquanto é mantida a comunicação, isso se dá pela possibilidade de efetuar o *handover* entre as redes dependendo do estado atual do sistema e de ser factível realizar o roteamento dos pacotes na rede de forma mais eficiente dada a localização dos integrantes da rede (8).

2.2 Mininet-WiFi

O Mininet-WiFi é uma extensão do Mininet, o qual possui suporte nativo para WiFi, entretanto ainda é possível simular outras tecnologias de redes sem fio nos experimentos. Com ele é possível a virtualização de pontos de acesso sem perder as funcionalidades presentes no Mininet, como *hosts*, *switches* e controladores OpenFlow.

2.2.1 Simulação e emulação

Simulação computacional é um conceito no qual um sistema criado computacionalmente se comporta semelhante a um experimento real, mas é implementado de uma maneira completamente diferente. Ele fornecerá o comportamento básico do experimento, mas não necessariamente implementará todos os elementos e regras do sistema que está sendo simulado. Os simuladores fornecem ao pesquisador uma alta flexibilidade aliada a um grande nível de controle e repetibilidade, além disso possui um baixo custo se comparado com outras alternativas utilizadas para testes (11).

A emulação se difere da simulação no que tange a implementação de todos os componentes que irão interferir naquele experimento, ou seja o sistema se comportará exatamente de acordo com a situação real. Por exemplo, um simulador de voo dá a impressão que o usuário está pilotando um avião, mas está completamente desconectado da realidade de pilotar um avião. A simulação gera um ambiente similar, mas não é exatamente a mesma coisa, enquanto que a emulação seria como o sistema MAME que

replica com perfeição os *hardwares* e *firmwares* presentes nos jogos antigos do sistema de *arcade*, replicando até os erros e falhas originais que ocorreriam durante os jogos. O uso da emulação pode ser resumida através da criação de um ambiente parcialmente sintético onde são utilizadas implementações reais do que se deseja estudar, com isso a emulação tenta se aproximar mais do mundo real do que a simulação ao mesmo tempo que mantém as características de repetibilidade que os simuladores possuem (11).

Diante disso, enquadrámos o Mininet-WiFi como um emulador de redes para avaliação de desempenho, teste e depuração de protocolos de redes e demais assuntos relacionados à arquitetura de redes de computadores. Ressaltando duas importantes características do Mininet-WiFi como emulador, temos a possibilidade de utilizar ferramentas de terceiros sem modificações no código fonte e protocolos de redes reais.

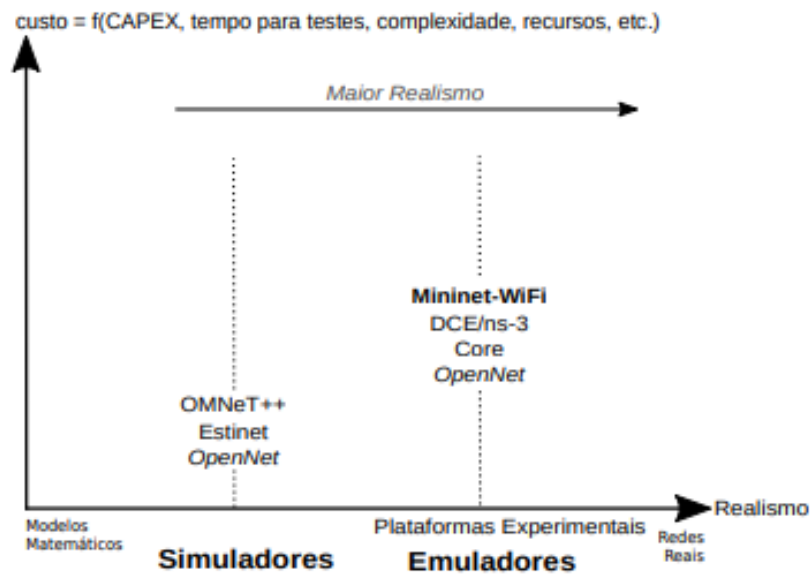


Figura 5 – Comparação entre plataformas experimentais para redes sem fio (2)

O Mininet-WiFi baseia-se no Mininet para fazer a virtualização dos dispositivos, representando de forma lógica uma cópia da pilha de rede do sistema operacional Linux, que inclui suas próprias rotas, regras de *firewall* e dispositivos de rede, atuando assim como se fossem verdadeiros computadores com as mesmas propriedades de rede que um computador físico pode ter.

As interfaces de rede sem fio dentro do Mininet-WiFi operam nos modos *managed* ou *master*, definindo assim as funções de cada nó. Por padrão cada estação possui apenas uma interface, podendo ser adicionadas mais caso haja necessidade. Uma vez que uma estação está conectada a um ponto de acesso, elas podem se comunicar com os *hosts* do Mininet, enquanto que os pontos de acesso são responsáveis pela gestão das estações associadas a ele.

2.2.2 Funcionamento básico

Para instalação do Mininet-WiFi, primeiro devemos baixar o repositório com o seguinte comando:

```
1 ~$ git clone https://github.com/intrig-unicamp/mininet-wifi
```

Após isso podemos fazer a instalação executando os seguintes comandos:

```
1 ~$ cd mininet-wifi
2 ~/mininet-wifi$ sudo util/install.sh -Wlnfv6
```

A partir desse ponto já é possível usar o Mininet-WiFi, ao se iniciar o programa com o comando `sudo mn -wifi`, iremos instanciar uma topologia de rede com duas estações conectadas a um ponto de acesso através de um meio sem fio. Além disso, há a presença de um controlador conectado ao ponto de acesso.

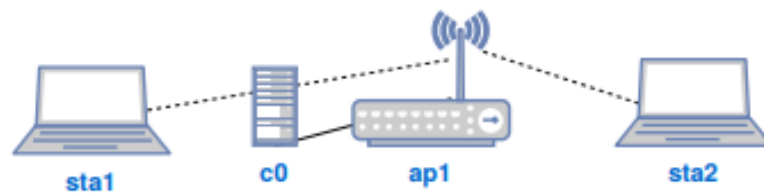


Figura 6 – Topologia instanciada ao se inicia o Mininet-WiFi (2)

Concomitantemente à inicialização do Mininet-WiFi, abre-se a interface de linha de comando (*Command Line Interface* - CLI), onde é possível utilizar basicamente qualquer comando de rede. Dentre estes, podemos listar os seguintes comandos:

- Um comando primordial é o `help`, o qual irá listar todas as instruções que são possíveis fazer no Mininet-WiFi.
- Temos o comando `nodes`, o qual irá listar todos os nós criados no ambiente de simulação.
- Com o comando `<station> ping -c1 <station>` é possível verificar a conectividade entre as duas estações. Note que o comando `-c1` definirá que apenas um pacote ICMP será enviado e a estação pode ser passada tanto utilizando seu nome quanto seu endereço IP.
- Para podermos abrir diferentes terminais para cada nó e emitir instruções como se eles estivessem sendo emitidos diretamente em um computador, análogo ao que acontece no mundo real, basta digitar `xterm <station>`.

- Uma operação exclusiva do meio sem fio é a operação de desconectar uma estação de um ponto de acesso, a qual pode ser feita com o comando:

```
15     ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1',
16                             failMode="standalone", position='50,50,0')
17
18     info("*** Configuring wifi nodes\n")
19     net.configureWifiNodes()
20
21     if '-p' not in args:
22         net.plotGraph()
23
24     net.setMobilityModel(time=0, model='RandomDirection',
25                         max_x=100, max_y=100, seed=20)
26
27     info("*** Starting network\n")
28     net.build()
29     ap1.start([])
30
31     info("*** Running CLI\n")
32     CLI(net)
33
34     info("*** Stopping network\n")
35     net.stop()
```

Código 2.1 – Implementação básica de movimentação no Mininet-WiFi (2)

O código acima exemplifica a criação de uma topologia de rede definindo um modelo de mobilidade para todos os nós da rede. Ressaltando os pontos mais importantes do código, temos a linha que inicia nossa rede com `net = Mininet_wifi()`, após isso adicionamos os nós com a função `addStation`, onde os principais parâmetros são o nome, o endereço físico, o IP, e parâmetros relativo a posição. Com isso, podemos analisar a linha 24 do Código 2.1, onde definimos o padrão de mobilidade “*Random Direction*”, para os nós da rede.

Para se ter uma compreensão mais aprofundada de como funciona a movimentação no Mininet-WiFi, faz-se necessário o estudo do fluxo interno do programa referente à mobilidade, o qual está presente na classe `Mininet_wifi`. O gatilho desse fluxo é a chamada `net.build()`, o qual atua sobre a movimentação caso o padrão de mobilidade desejado tenha sido configurado previamente, tal configuração é ilustrada na linha 24 do Código 2.1. Caso exista um padrão de movimentação configurado, o programa chamará a função `start_mobility`, a qual irá instanciar a classe `model`.

A classe `model` herda o comportamento da classe `Mobility`, a qual contém todas as funções básicas relativas à movimentação dos nós, como por exemplo as funções

responsáveis por definir a posição dos nós, retornar a velocidade, retornar à distância entre nós, dentre outras funcionalidades. Além disso, a classe `model` tem como principal função iterar sobre os padrões de movimentação, atualizando a posição dos nós e fazendo com que estes executem os comportamentos previstos por cada tipo de movimentação diferente. O diagrama que representa os relacionamentos supracitados pode ser observado na Figura 7. Já na Figura 8 é possível visualizar o diagrama de classes que representa os elementos envolvidos na movimentação dos nós no Mininet-WiFi, é possível verificar a supracitada classe `model` e seu relacionamento com a classe base responsável pelas funções básicas de movimentação.

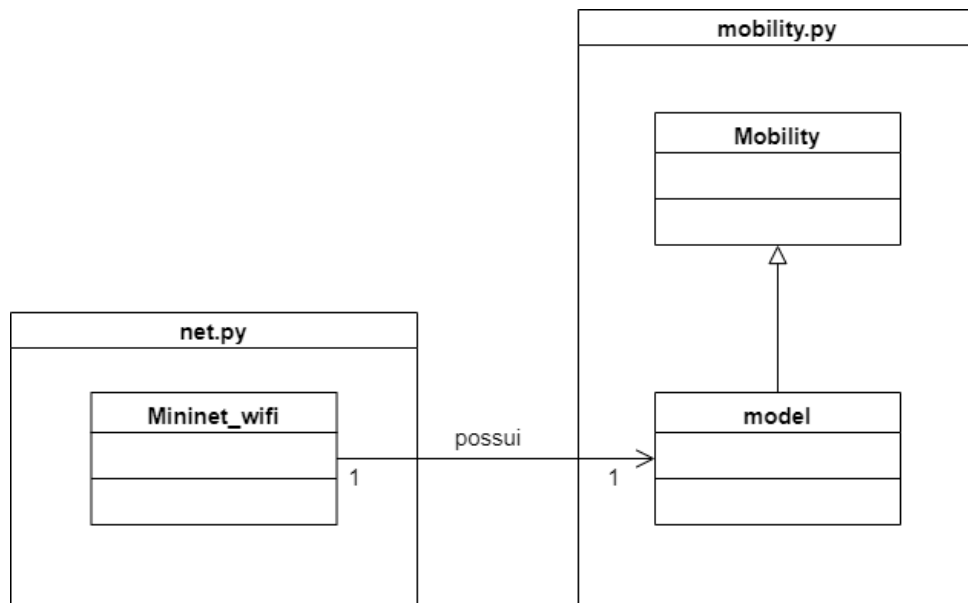


Figura 7 – Diagrama de classes envolvidas na movimentação no Mininet-WiFi

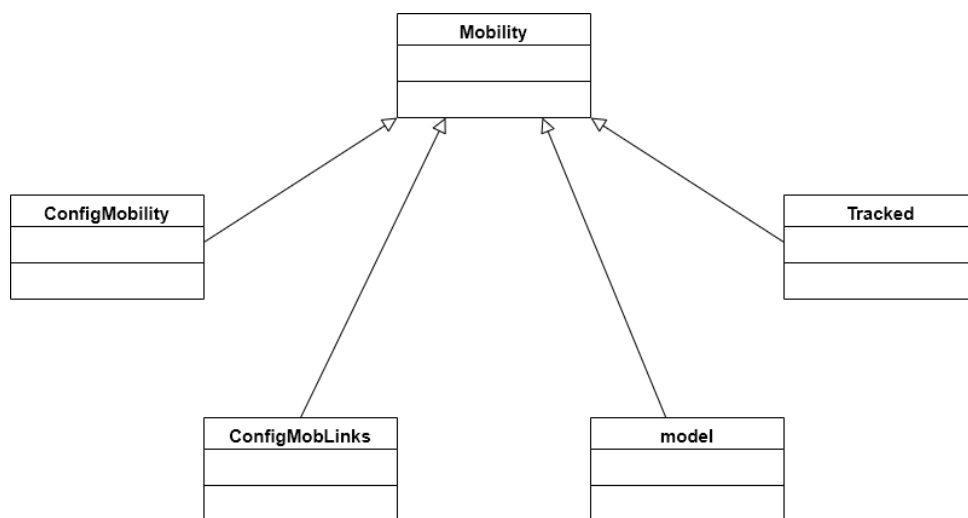


Figura 8 – Diagrama de classes das entidades de movimentação do Mininet-WiFi

Os padrões de movimentação presentes Mininet-WiFi são definidos por meio de classes e funções as quais representam os seguintes modelos de mobilidade:

- Random Walk
- Truncated Levy Walk
- Random Direction
- Random WayPoint
- Gauss-Markov
- Heterogeneous Levy Walk

Na Figura 9 é possível observar a estrutura de classes utilizada para representar os modelos de mobilidade randômicos, no caso dos modelos com dependência temporal e espacial não existem classes que representam a movimentação e sim funções responsáveis por definir o movimento dos nós. Tais classes e funções distribuem as posições do nós seguindo a lógica específica de cada padrão de mobilidade, isto é feito através dos cálculos de posicionamento dos nós realizados em cada iteração da simulação. Os pontos calculados são enviados para a classe model, que por fim atualiza a posição dos nós dentro da simulação.

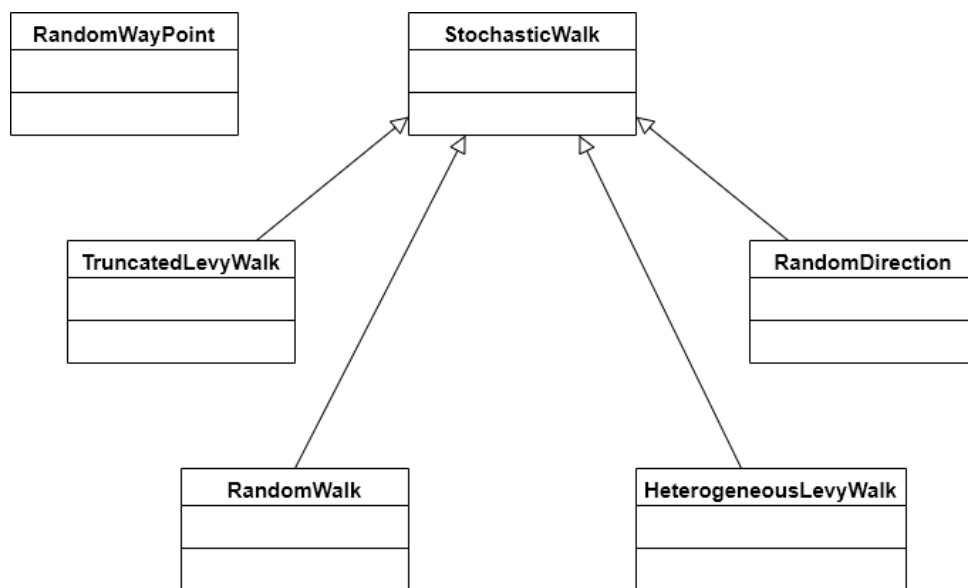


Figura 9 – Diagrama de classes dos modelos de mobilidade randômicos

2.3 Movimentação

O estudo da movimentação dos elementos que compõem as redes é extremamente importante para ambientes de simulação e emulação, principalmente quando tais redes se utilizam de tecnologias sem fio. Isso se dá pois a qualidade das conexões, a confiabilidade da rede e a forma como os pacotes serão distribuídos entre os nós estão fortemente relacionados

com a topologia da rede, caso exista a movimentação de um dos componentes da rede então tal topologia pode se alterar drasticamente.

Além disso, com o avanço das tecnologias de comunicação sem fio, o barateamento dos equipamentos e a facilidade de acesso, as redes sem fio estão se tornando cada vez maiores e mais complexas, isso faz com que o estudo de padrões de movimentação dos nós nas redes se mostrem cada vez mais importantes. Esses estudos visam facilitar o entendimento das dinâmicas dentro das redes e com isso gerar melhores *benchmarks*, os quais podem levar ao desenvolvimento de algoritmos de roteamento, *handover*, dentre outros, cada vez mais eficientes e completos.

2.3.1 Modelos de mobilidade

Para estudar a movimentação de uma forma mais geral são utilizados modelos de mobilidade, os quais são utilizados para descrever o padrão de movimentação de nós, e como sua posição, velocidade e aceleração mudam com o tempo (3). Como os padrões de mobilidade em uma rede têm um papel importante na determinação da performance de um protocolo de comunicação, é interessante que o modelo de mobilidade emule da forma mais realista possível os padrões de movimentação encontrados no problema estudado da vida real. Caso tais padrões não sejam modelados de forma satisfatória, os resultados das pesquisas podem acabar levando a escolhas ou decisões equivocadas.

A criação de modelos de mobilidade que representam os padrões de movimentação reais é um problema difícil de ser resolvido, uma das abordagens possíveis para resolver tal problema é construir os modelos a partir de informações de posição reais dos nós a cada instante durante um tempo determinado, essa parece uma boa solução a ser seguida e estudos como o do Anglova (1) providenciam os dados para isso, porém os dados ainda são muito pontuais e específicos sendo necessária a produção de mais estudos nesse sentido para a criação de um modelo viável e minimamente completo. Por conta disso, muitos pesquisadores vêm tentando desenvolver os chamados modelos de mobilidade sintéticos de forma mais realista possível (3), tais modelos não seguem a ideia supracitada da utilização de pontos angariados previamente.

O estudo da mobilidade nas redes sem fio vem mudando nos últimos anos e se alterou de um estudo a nível macroscópico, o qual estudava as células da telefonia móvel e características inerentes a elas, para um nível microscópico onde a posição de cada componente da rede é analisada e a interação entre tais nós é também levada em consideração. O resultado do desenvolvimento de modelos de mobilidade sintéticos alinhados com a ideia da análise microscópica das movimentações geraram alguns modelos conhecidos pela sociedade acadêmica, os quais podem ser classificados como ilustrado na Figura 10.

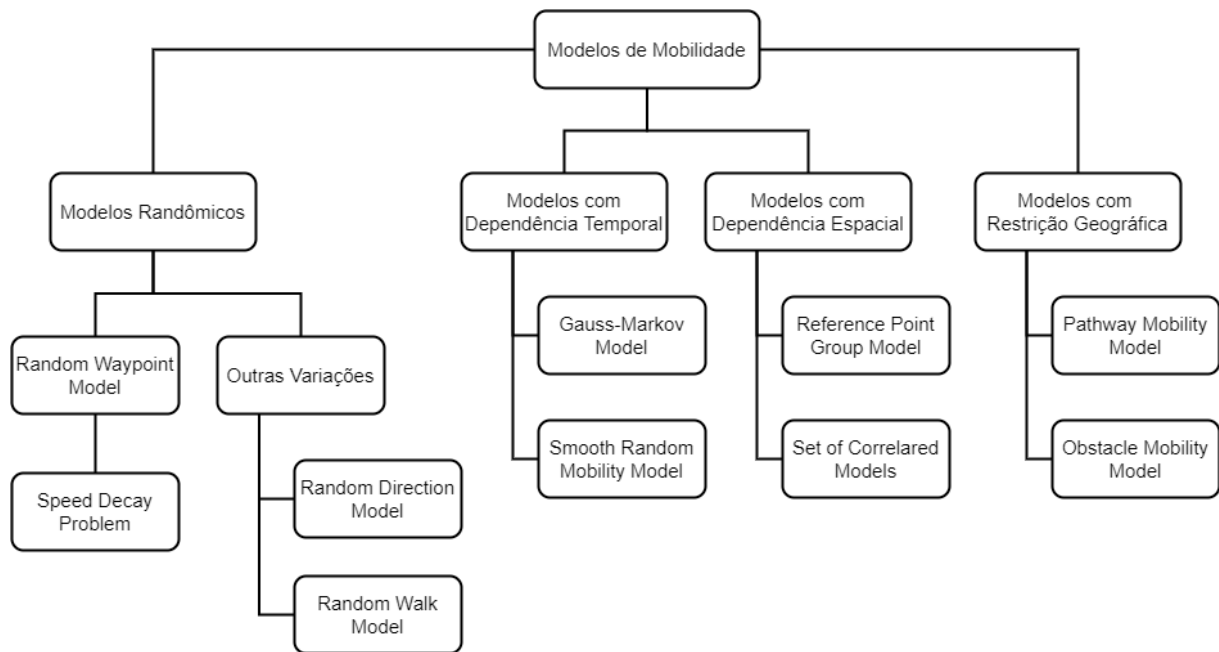


Figura 10 – Classificação de modelos de mobilidade (3)

Os modelos classificados como randômicos geram uma movimentação independente para cada nó de forma aleatória em torno de pontos especificados, tal classificação engloba vários modelos bastante utilizados em pesquisas como o *Random Waypoint*, o *Random Direction* e o *Random Walk*. Porém, para algumas situações tais modelos não são suficientes pois os nós se movem de uma forma mais complexa e estruturada. Por conta dessa limitação alguns pesquisadores focaram em desenvolver novos modelos de mobilidade que englobam diferentes características da movimentação, com isso surgiram modelos em que a movimentação dos nós é mais ou menos restrita pelo seu histórico de movimento, por outros nós na vizinhança ou mesmo pelo ambiente onde encontram-se inseridos. Tais restrições originaram as classificações vistas na Figura 10, modelos que são afetados pelo histórico de movimentação dos nós são classificados como Modelos com Dependência Temporal, já os modelos que movimentam os nós baseado em nós da vizinhança são chamados de Modelos com Dependência Espacial, por fim os modelos que estão restritos por características geográficas como ruas, avenidas e obstáculos são chamados de Modelos com Restrição Geográfica.

Alguns dos modelos supracitados são bem conhecidos e consolidados no meio acadêmico, por conta disso muitos simuladores e emuladores os implementam e possibilitam que pesquisas possam ser realizadas. O Mininet-WiFi é um desses emuladores e implementa padrões como o *Random Waypoint*, *Random Direction*, *Random Walk*, *Gauss-Markov* e *Reference Point Group*. Apesar de suportar vários dos padrões de mobilidade existente o *software* ainda falha em representar movimentações mais complexas e estruturadas muito comuns em ambientes de operações militares e que se encaixam na categoria de Modelos com Dependência Espacial.

2.3.2 Movimentação militar

A movimentação de tropas militares é um tópico da doutrina militar bastante importante para o contexto operacional, pois com uma tropa bem adestrada e uma doutrina de movimentação bem definida a capacidade da Força Armada de sobrepujar seus inimigos é acentuada. O manual que versa sobre o tema (12) define movimento e manobra da seguinte maneira: "Caracteriza-se pela capacidade de deslocar ou dispor forças de forma a colocar o inimigo em desvantagem relativa e, assim, atingir os resultados que, de outra forma, seriam mais custosos em pessoal e material."

Tal movimentação depende de muitos fatores para lograr êxito no seu objetivo, dentre tais fatores um bastante importante é o comando e controle (12). O comando e controle, também chamado de C2, possui uma grande importância no teatro de operações por ser capaz de prover consciência situacional para os comandantes nos mais diversos níveis, isso caracteriza uma grande vantagem estratégica em cenários de combate.

No que tange à movimentação dos componentes do teatro de operações, todas as manobras, e os deslocamentos associados às mesmas, são bem definidos em manuais e fazem parte da supracitada doutrina. Porém, apesar de os movimentos serem bem definidos, eles podem ser muito complexos dependendo da situação em que os elementos se encontrem. Tal complexidade na movimentação é um dos problemas que torna a realização do C2 tão difícil, fazer com que todos os elementos estejam conectados durante toda a manobra pode se mostrar um trabalho muito complicado e que demanda um bom planejamento da manobra em si.

No caso de um regimento de cavalaria mecanizada a doutrina que versa sobre a disposição e movimentação das tropas é o manual de campanha EB70-MC-10.354 (4). Em tal documento é possível visualizar desde movimentos mais simples como os deslocamentos em linha, coluna e cunha, ilustrados nas Figuras 11, 12 e 13 respectivamente, até movimentos mais complexos como o movimento do regimento no seu emprego durante uma incursão, o qual é ilustrado na Figura 14.

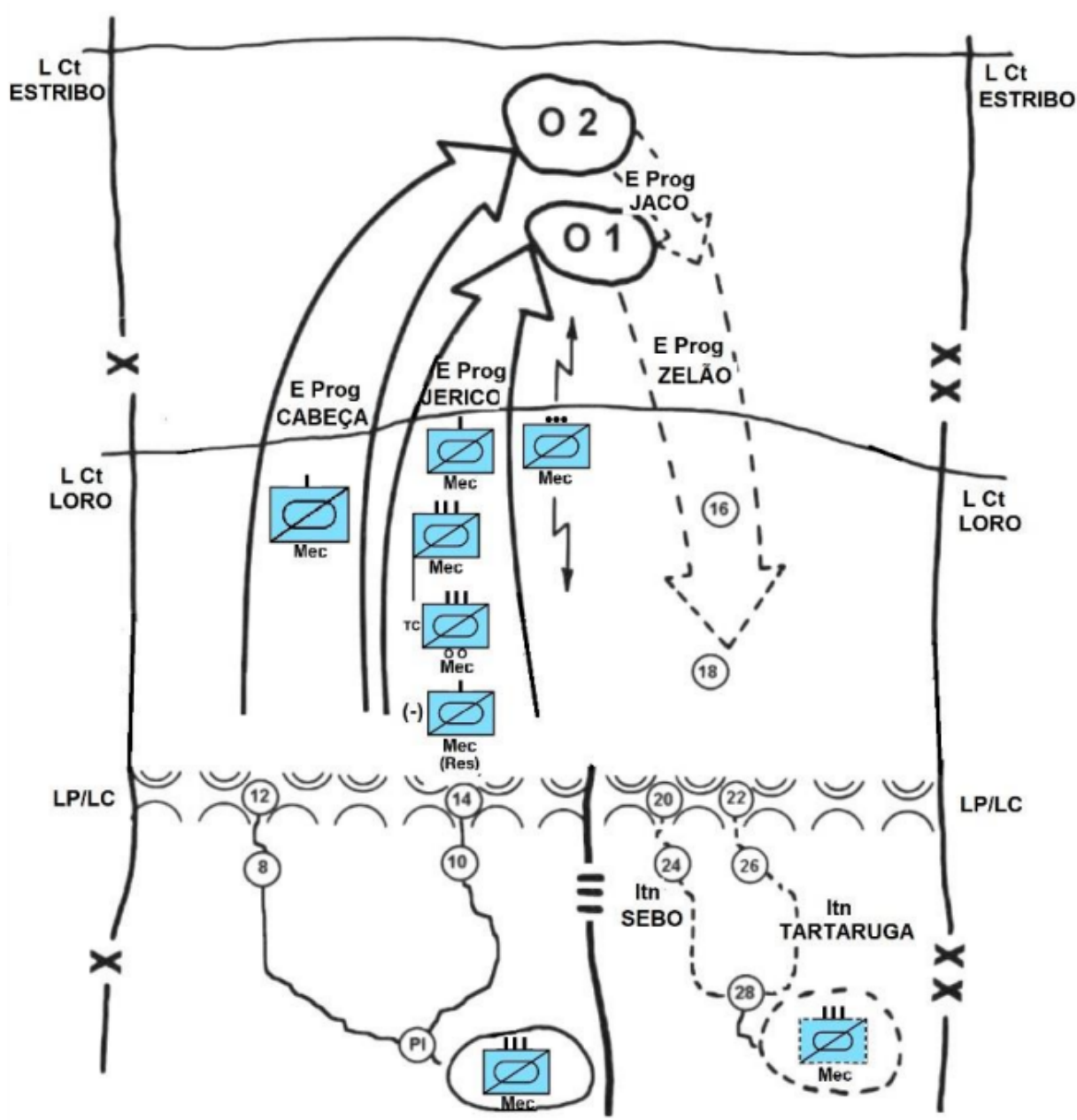


Figura 14 – Emprego do RC Mec na incursão (4)

3 ABORDAGEM PROPOSTA

3.1 Visão Geral

Inicialmente foram realizados estudos sobre redes, mais especificamente redes de comunicação sem fio, redes de comunicação militares e sobre simulações e emulações visando a contextualização do projeto, como visto na seção 2.1. Em seguida, na seção 2.2, foi feito um estudo teórico sobre o emulador de redes Mininet-WiFi embasado no livro de referência do *software* (2) seguido de um estudo prático com o uso do emulador para a criação de algumas redes, isso focado na ambientação dos envolvidos no projeto com o ambiente provido pelo sistema. Por fim, foi feito um estudo na estrutura interna do emulador para verificar como é feita a movimentação dos nós em uma rede.

Essa atividade foi seguida por um estudo de viabilidade quanto a expansão das capacidades de movimentação do Mininet-WiFi. Neste momento, foram elaborados diagramas de representação básicas da solução de movimentação do Mininet-WiFi (Seção 3.3) e da solução proposta pelo grupo (Seção 3.4).

Na etapa seguinte, foi realizado um estudo sobre os requisitos para a criação de emulação de redes de comunicação militares focando no tema da mobilidade, a qual é a área de interesse deste trabalho. Tal levantamento de requisitos foi realizado a partir da análise de documentos que definem a doutrina de movimentação militar e através de entrevistas com pesquisadores interessados no projeto, tais requisitos foram especificados na seção 3.2.

Após a fase de contextualização e de levantamento de requisitos do projeto, foram feitas as modificações estruturais necessárias para suportar grupos de mobilidade no Mininet-Wifi. Tais modificações foram modeladas em diagramas de classes e os fluxos seguidos pelo *software* foram discriminados em alguns diagramas de processos com o objetivo de facilitar o entendimento da solução proposta no trabalho.

Finalizada as modificações estruturais no projeto inicia-se a modelagem e a implementação dos modelos de mobilidade que devem fazer parte do projeto. Foram escolhidos três modelos de mobilidade diferentes responsáveis por movimentar os nós dentro de padrões espaciais. Tais modelos funcionam de forma que a transição entre diferentes padrões de movimentação são realizadas de forma fluida e contínua, o que possibilita a representação de movimentações de tropas reais.

Por fim, para verificar que as modificações feitas no projeto atendiam um ambiente de operações reais, foi montado um teste que gera passos de movimentação baseados em dados de latitude e longitude do Anglova. A transformação dos dados de satélite

para movimentações no simulador foram gerados por um *script* auxiliar criado para esse propósito.

3.2 Levantamento de requisitos

O levantamento de requisitos para o projeto foi feito a partir de dois métodos. Inicialmente, levantou-se os manuais de campanha do Exército Brasileiro que falam sobre o tema de movimentação de tropas. Após a leitura de tais manuais foram feitas algumas conversas com especialistas para definir o escopo do projeto e por fim levantar os seguintes requisitos:

- Possibilitar a alocação de nós em grupos, para que os nós dentro desse grupo consigam executar uma ação conjunta seguindo um determinado padrão de movimentação.
- Possibilitar a aplicação de um padrão de movimentação em um determinado grupo, para que nós em grupos diferentes possam executar padrões de movimentação distintos.
- Possibilitar a criação de padrões de movimentação a partir da composição de padrões mais simples, facilitando o desenvolvimento e a testagem de padrões mais complexos.
- Manter todas as características e funcionalidades originais do Mininet-WiFi, expandindo as funcionalidades através de ajustes de algumas funções do Mininet, para possibilitar a alteração no fluxo de funcionamento permitindo as mudanças previstas.

3.3 Movimentação no Mininet-Wifi

Uma etapa importante para o desenvolvimento da solução de movimentação proposta neste trabalho é entender como o software original lida com a movimentação dos elementos da rede. Essa movimentação possui um papel fundamental para o estudo das interações e comunicações entre os nós ao longo do tempo. Por conta disso, apesar de o Mininet-Wifi ser um emulador de redes, ele possui uma gama de modelos de movimentações e opções customizadas para melhor representar o cenário estudado.

A forma mais simples e fácil de movimentar os nós, é utilizar o método existente no nó para mudar sua posição atual. Tal método é chamado de `setPosition` e pertence a classe `Node` (ilustrado na Figura15), ele é responsável por mudar a posição do nó e renderizar o elemento com a posição correta na interface gráfica.

Uma segunda abordagem disponível no software para emular uma movimentação é utilizar a classe `ReplayingMobility` (ilustrado na Figura16). Tal classe é responsável por movimentar os nós seguindo uma sequência de posições pré-definidas. As posições

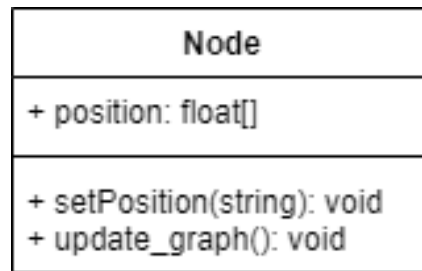


Figura 15 – Representação simplificada da classe Node

de cada nó devem estar definidas para cada passo de execução temporal do programa. Logo, durante a execução de uma emulação se conhece o movimento realizado por cada nó individual.

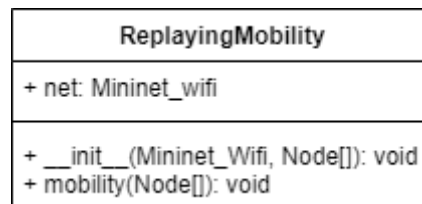


Figura 16 – Representação simplificada da classe ReplayingMobility

A terceira abordagem disponível no emulador para realizar a movimentação dos nós é utilizar os chamados modelos de movimentação, sobre os quais este trabalho fez uma breve apresentação na seção 2.3.1. Esta solução é mais complexa do que as anteriores pois é responsável por gerar as posições durante a execução do software. Além disso, ela permite a inserção de ruídos na movimentação adicionando um certo nível de aleatoriedade na movimentação dos elementos da rede. O Mininet-Wifi possui por padrão uma série de modelos de mobilidade diferentes, que podem ser utilizados nos nós da rede com certas restrições.

A última técnica utilizada pelo software para aplicar a movimentação nos nós da rede é a mais complexa. Dessa forma, foram feitos estudos adicionais sobre o comportamento de tal funcionalidade. Do modo como a solução foi projetada, o usuário consegue definir um modelo de mobilidade para todos os nós móveis da simulação. Tais nós são definidos como móveis através do atributo `position` do nó e do parâmetro `initPos` passado durante a inicialização do elemento. A definição do modelo de mobilidade na rede é feita através do método `setMobilityModel`, o qual pertence à classe `Mininet_wifi`. Esse método aceita como parâmetro diversas propriedades que posteriormente serão aplicadas na movimentação, tais como:

- **min_v**: Velocidade mínima
- **max_v**: Velocidade máxima

- **min_x**: Posição mínima no eixo X
- **max_x**: Posição máxima no eixo X
- **min_y**: Posição mínima no eixo Y
- **max_y**: Posição máxima no eixo Y
- **min_z**: Posição mínima no eixo Z
- **max_z**: Posição máxima no eixo Z

Após a definição de um modelo de mobilidade e a subsequente inicialização da emulação, uma sequência de métodos e funções são executados para de fato realizar a movimentação dos nós seguindo o modelo pré-determinado. Para facilitar o entendimento do fluxo executado, decidiu-se quebrar a execução em três grandes grupos: configuração da mobilidade, inicialização da mobilidade e fluxo da mobilidade, que serão explicados a seguir neste trabalho.

Antes de entrar nos detalhes de implementação do fluxo de modelos de mobilidade, é importante apresentar um conceito relevante que será utilizado a posteriori. Tal conceito é chamado de “*Iterator*” e é utilizado por diversas linguagens de programação. Ele consiste em um objeto que faz um apontamento para um elemento dentro dele mesmo. Exemplos de *iterators* são encontrados em diversos tipos de estrutura de programação como em ponteiros, vetores, listas, dicionários, pilhas, filas, dentre outros. Esses *iterators* são classificados em 5 diferentes tipos. Neste trabalho será utilizado o tipo mais simples de *iterator* chamado de “*input iterator*”. Com tal objeto, é possível acessar os elementos internos da classe apenas uma vez, e estes são acessados em uma ordem específica, não sendo possível alterar a ordenação da sequência ou recuperar os elementos na ordem reversa.

3.3.1 Configuração da Mobilidade

O fluxo chamado de “Configuração da Mobilidade” trata do momento onde o modelo de mobilidade e seus parâmetros são definidos. Tal processo ocorre majoritariamente dentro da classe `Mininet_wifi`, seus passos podem ser observados na Figura 17 e estão descritos abaixo.

- (P1): Primeiramente o usuário, no seu *script*, define o modelo de mobilidade que deseja utilizar juntamente com os parâmetros inerentes a ele. Tal configuração se dá pela invocação do método `setMobilityModel` da classe `Mininet_wifi`. O modelo de mobilidade para aquela rede é armazenada em uma propriedade chamada `mob_model`.

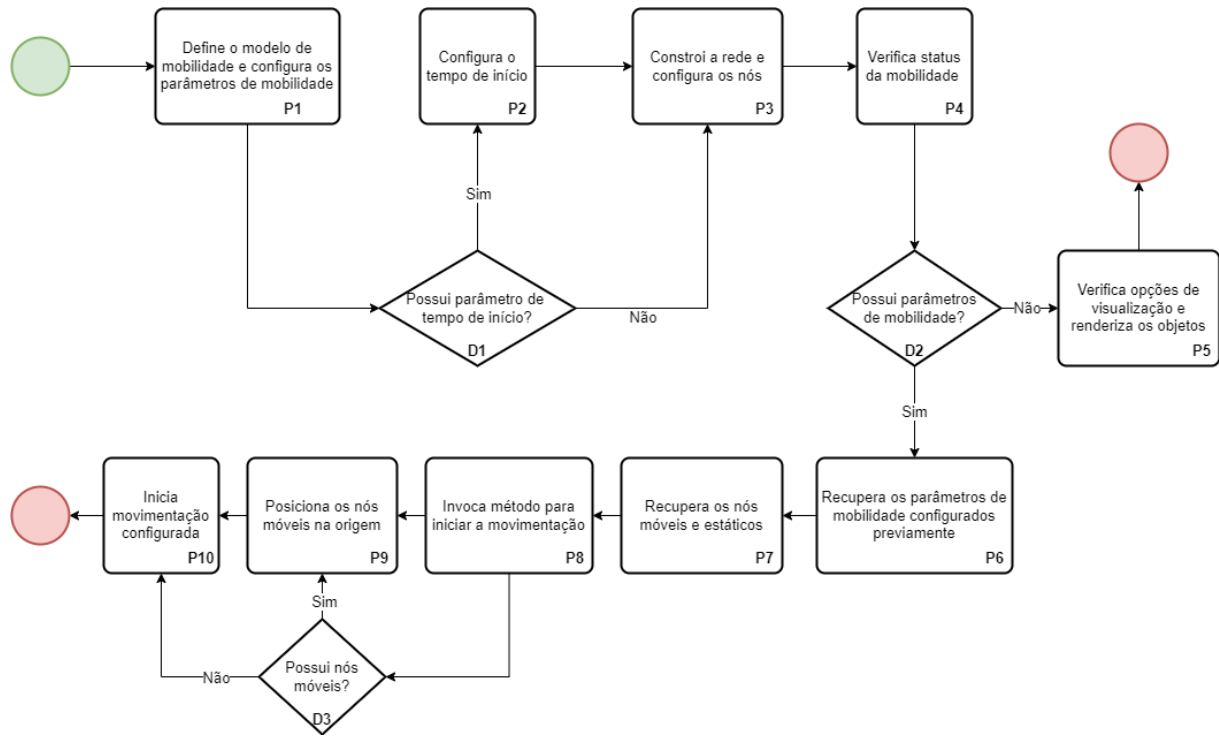


Figura 17 – Fluxo de configuração da mobilidade

- (P2): Caso o usuário tenha passado o parâmetro de tempo para iniciar a mobilidade na chamada do método então, ainda dentro de `setMobilityModel`, o valor escolhido é salvo em uma propriedade chamada de `mob_start_time`.
- (P3): Dentro de seu *script* o usuário chama o método `build` pertencente à classe `Mininet_wifi`. Este método é responsável por construir a rede a partir da topologia fornecida, iniciar alguns serviços necessários para a emulação e configurar os *hosts* pertencentes à rede.
- (P4): O método `build` citado no processo anterior faz a invocação de um outro método da classe. Este outro método é chamado de `check_if_mob`. Tal função verifica se a classe possui algum valor configurado em `mob_model` e encaminha o fluxo da aplicação para iniciar a movimentação ou não.
- (P5): Caso não exista nenhum modelo de mobilidade configurado em `mob_model`, o software apenas renderiza a janela de visualização com tamanho adequado para se ver toda a rede.
- (P6): Caso exista um modelo de mobilidade configurado, o método `get_mobility_params` é invocado para recuperar os parâmetros de mobilidade. Tais valores são salvos em uma variável local chamada de `mob_params`. Esse fluxo ocorre dentro do supracitado método `check_if_mob`.

- (P7): Em seguida, tenta-se recuperar os nós estáticos e móveis da rede. Isso é feito através do método `get_mob_stat_nodes`. Os valores retornados pela chamada da função são salvos em variáveis locais chamadas de `stat_nodes` e `mob_nodes`.
- (P8): Os parâmetros de mobilidade e os nós recuperados nos processos P6 e P7 são, por fim, passados para o método `start_mobility`, o qual é responsável pelos últimos passos de configuração.
- (P9): Caso tenham sido passados nós móveis para o método `start_mobility`, então eles são posicionados na origem das coordenadas.
- (P10): Por fim, é instanciada a classe `model`. Essa instancição é feita passando todos os parâmetros configurados até o momento no construtor da classe. Tal classe é a responsável por efetivamente realizar os passos para a movimentação dos nós na rede.

3.3.2 Inicialização da Mobilidade

O segundo fluxo pertinente para os modelos de mobilidade no Mininet Wifi é o chamado de “Inicialização da Mobilidade”. Tal fluxo parte da inicialização da classe `model` com os parâmetros inseridos pelo usuário e visa instanciar mecanismos internos tal como novas *threads*. Além disso, esse fluxo efetua a chamada das funções iterativas que possibilitam a movimentação dos nós. Os passos de tal fluxo estão ilustrados na Figura 18 e a descrição dos processos se encontra abaixo.

- (P1): O primeiro processo ocorre já no construtor da classe `model`. Nesse passo é criada uma nova *thread* chamada “mobModel”. Tal *thread* possui como “target” (objeto que deverá ser acionado ao se executar a *thread*) o método `models`. Este método recebe como parâmetros todos os elementos que foram passados para o construtor da classe. O novo objeto de *thread* criado é armazenado em uma propriedade da classe `Mobility`, a qual é classe pai de `model`. Tal propriedade é chamada de `thread_`. Por fim, a *thread* em questão é inicializada, o que ocasiona a execução do “target”.
- (P2): Já dentro do método `models` é executado um *loop* que itera sobre os nós móveis que foram passados como parâmetros para o método. Nesse processo, são configurados os parâmetros de posições mínimas e máximas nos eixos x e y juntamente com as velocidades mínimas e máximas aceitáveis na simulação. É importante ressaltar que os parâmetros “min_x”, “min_y”, “min_v” e “max_v” são configurados com valores padrão do *software*, não sendo possível alterá-los apenas os passando como parâmetros para o método.
- (P3): Nesse passo, o *Plotter* (objeto responsável pela criação da interface gráfica de interação do usuário) é inicializado através da classe `PlotGraph`, a qual recebe como

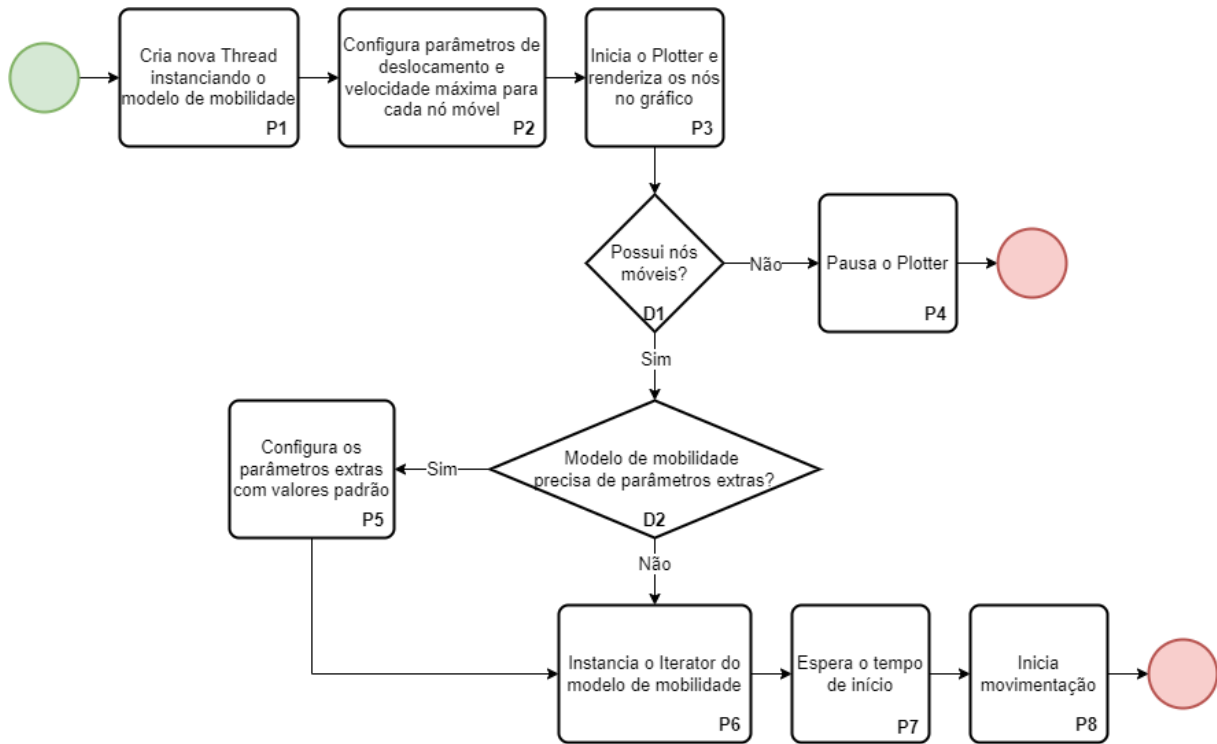


Figura 18 – Fluxo de inicialização da mobilidade

parâmetros os nós, as posições máximas em x e y, e todos os parâmetros extras que foram passados para o método `models`.

- (P4): Caso não existam nós móveis nesse ponto, o *Plotter* é pausado através do método `pause` disponibilizado pela classe `PlotGraph` e a execução da *thread* é encerrada.
- (P5): Neste ponto, é verificado qual modelo de mobilidade se encontra no parâmetro local `mob_model`. Dependendo do modelo escolhido, alguns parâmetros extras são inseridos em cada nó.
- (P6): Com os parâmetros adicionais já configurados e sabendo qual modelo de mobilidade o usuário deseja utilizar, é feita a instancição do *iterator* correspondente ao modelo. Tal objeto é instanciado recebendo os nós móveis da simulação e parâmetros adicionais que variam de acordo com o modelo implementado. Uma referência do objeto do *iterator* é salva em uma variável local chamada `mob`.
- (P7): O código é então pausado até que seja atingido o tempo de início da movimentação. Tal tempo é definido pelo parâmetro `mob_start_time`, que foi previamente configurado pelo usuário. Caso tal parâmetro não tenha sido configurado, se assume que a movimentação deve começar no tempo zero.
- (P8): Por fim, o fluxo de mobilidade é acionado através da chamada do método `start_mob_mod`, o qual deve prosseguir com a movimentação dos nós móveis. Para tal método, são

passados como parâmetros o *iterator* do modelo de mobilidade (`mob`) e os nós móveis (`mob_nodes`).

3.3.3 Fluxo de Mobilidade

No fluxo de mobilidade é onde ocorre a movimentação dos nós, os valores das posições são recuperados do *iterator* e são utilizados para atualizar o gráfico com os nós na tela. Tal fluxo está ilustrado na Figura 23 e a descrição dos processos está disposta na sequência.

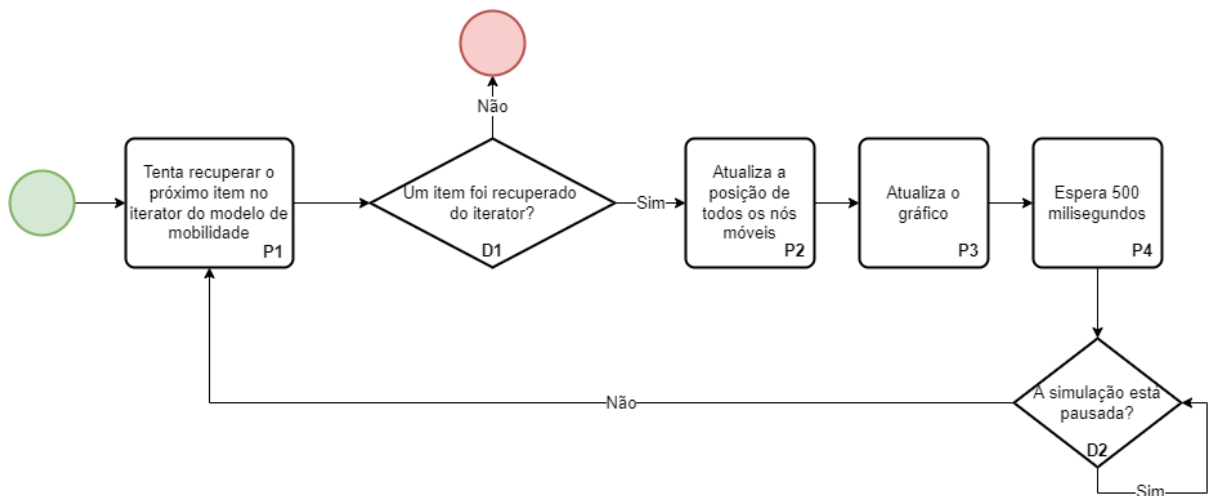


Figura 19 – Fluxo de mobilidade

- (P1): Inicialmente tenta-se recuperar os próximos valores de posição a partir do *iterator* do modelo de mobilidade (`mob`). Tais valores são salvos em uma variável local chamada de `xy`. Caso não exista nenhum valor a ser recuperado do *iterator*, a *thread* é encerrada.
- (P2): Já de posse das novas posições dos nós móveis, é feita a atualização da posição de cada nó. Para isso, é utilizado o método `set_pos` da classe pai `Mobility`. Tal método concede o novo valor para o atributo `position` da classe `Node`.
- (P3): Após atualizar a propriedade de posição de cada nó, é feita a atualização da posição do nó no gráfico. Isso é feito utilizando o método `update_2d` pertencente à classe `Node`.
- (P4): O fluxo executa uma breve pausa para representar o passo da simulação. Neste momento, existem dois possíveis cenários. No primeiro cenário, o usuário optou por não renderizar os gráficos na tela e possui acesso apenas via CLI. Nesse caso, a simulação é pausada por 500ms e então o fluxo segue normalmente. Já no segundo cenário, o usuário optou por visualizar os nós na tela. Assim, é chamado o método `pause` da classe `PlotGraph`. Isso é muito importante para o correto funcionamento

do código visto que, através deste método, as atualizações feitas no processo P3 surtem efeito em tela. Além disso, tal método executa uma breve pausa na simulação que, neste caso, é utilizada para representar o passo da simulação. Após a breve pausa, caso a simulação não esteja pausada, o fluxo volta ao processo P1.

Para facilitar o entendimento de todas as classes e métodos, suas entidades e relacionamentos, citados nos processos aqui descritos, foi desenvolvido um diagrama de classes seguindo os padrões da UML que pode ser observado na Figura 20. É importante ressaltar que tal diagrama foi simplificado visando mostrar apenas as entidades envolvidas no fluxo de interesse deste trabalho.

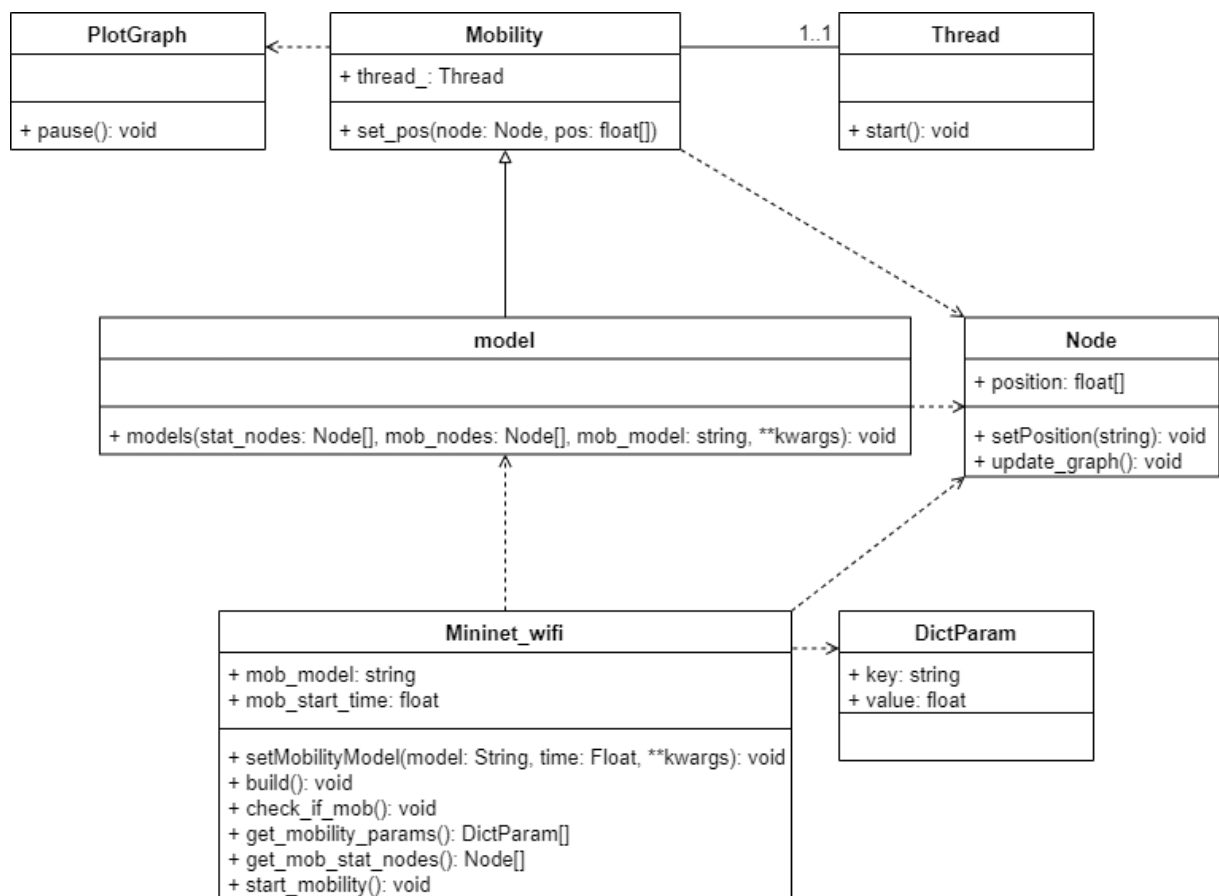


Figura 20 – Diagrama de classes simplificado do Mininet-Wifi

3.4 Arquitetura

Durante a análise de como seria estruturado o projeto, foi necessário construir um esquema que possibilitasse a interdependência espacial dos nós na rede de tal forma que o movimento de cada nó pudesse depender de outros nós. Para tal, foi necessária a criação de uma classe com o objetivo de fazer a representação de um grupo de nós, os quais podem seguir um padrão específico de movimentação. A implementação de tal classe pode ser verificada no Código 3.1.

```
1 class Group:
2     def __init__(self, groupId, mobModel, mobModelTime, **params):
3         self.groupId = groupId
4         self.mobModel = mobModel
5         self.mobModelTime = mobModelTime
6         self.nodes = []
7         self.params = params
8
9     def addStep(self, mobModel, mobModelTime, mobParams):
10        self.mobModel.append(mobModel)
11        self.mobModelTime.append(mobModelTime)
12        self.mobParams.append(mobParams)
```

Código 3.1 – Classe que representa um grupo de nós

A classe `Group` foi criada com o intuito de fornecer padrões de mobilidade diferentes para diferentes conjuntos de nós. Isso foi feito devido a necessidade de representar cenários de operações militares, os quais exigem a execução de manobras em grupos. Para permitir que os nós fossem segregados em grupos, foram necessários os seguintes atributos:

- **groupId:** Número identificador do grupo.
- **mobModel:** Lista com os modelos de mobilidade que o conjunto de nós irá seguir.
- **mobModelTime:** Lista com os intervalos de tempos que cada modelo de mobilidade estará ativo no conjunto de nós.
- **nodes:** Lista de nós que farão parte do grupo.
- **params:** Parâmetros adicionais que podem ser necessários para algum modelo de mobilidade. Tal parâmetro foi adicionado visando promover uma maior flexibilidade aos padrões de movimentação pertencentes ao grupo.

Além de criar a classe para representar os grupos, foi necessário fazer modificações na estrutura do Mininet-Wifi de forma a possibilitar uma mudança nos fluxos dispostos na seção 3.3 deste trabalho. Uma parte de tais mudanças se encontra na modificação de métodos da classe `Mininet_wifi`. Isso foi necessário para inserir o conceito de grupos de mobilidade no fluxo de configuração da mobilidade.

Um ponto importante de tal mudança era a necessidade de se manter a retrocompatibilidade com as soluções de mobilidade já fornecidas pelo Mininet-Wifi. Para fazer com que isso ocorresse, decidiu-se por fazer tais modificações na classe `Mimic`, a qual

herda o comportamento da classe base `Mininet_wifi` de tal forma que os fluxos normais do Mininet-Wifi continuassem funcionais.

Um outro ponto que sofreu modificações para possibilitar o funcionamento da mobilidade de grupos foi a classe `model`, a qual está fortemente relacionada com os fluxos de inicialização da mobilidade e com o fluxo de mobilidade propriamente dito. Com o mesmo objetivo de manter a retrocompatibilidade com os fluxos do Mininet-Wifi, foi necessário fazer a implementação de uma nova classe chamada `model_mimic`, que herda o comportamento da classe base `model` e modifica alguns de seus métodos para que o conceito de grupos funcione.

As modificações supracitadas estão explicadas nas próximas seções deste trabalho juntamente com os novos fluxos da aplicação gerados por tais modificações. Além disso, na Figura 21 pode-se observar um diagrama de classes que mostra todas as classes aqui citadas e os métodos relevantes para o fluxo de mobilidade.

3.4.1 Mudanças estruturais

As mudanças estruturais feitas para suportar o conceito de grupos de mobilidade ocorreram principalmente na classe `Mininet_wifi` e tratam os casos necessários para que a configuração da mobilidade leve em conta o conceito de grupos. Tais modificações estão apresentadas a seguir.

3.4.1.1 addGroup

```
1 def addGroup(self, groupId, mobModel, mobModelTime, **params):  
2     newGroup = Group(groupId, mobModel, mobModelTime, params)  
3     self.groups.append(newGroup)
```

Código 3.2 – Método para criação de um grupo

Este é um novo método desenvolvido para criar um objeto da classe `Group` com seus respectivos atributos como `GroupId` e `mobModel`. O objeto criado neste método é salvo em uma lista pertencente à classe `Mimic`.

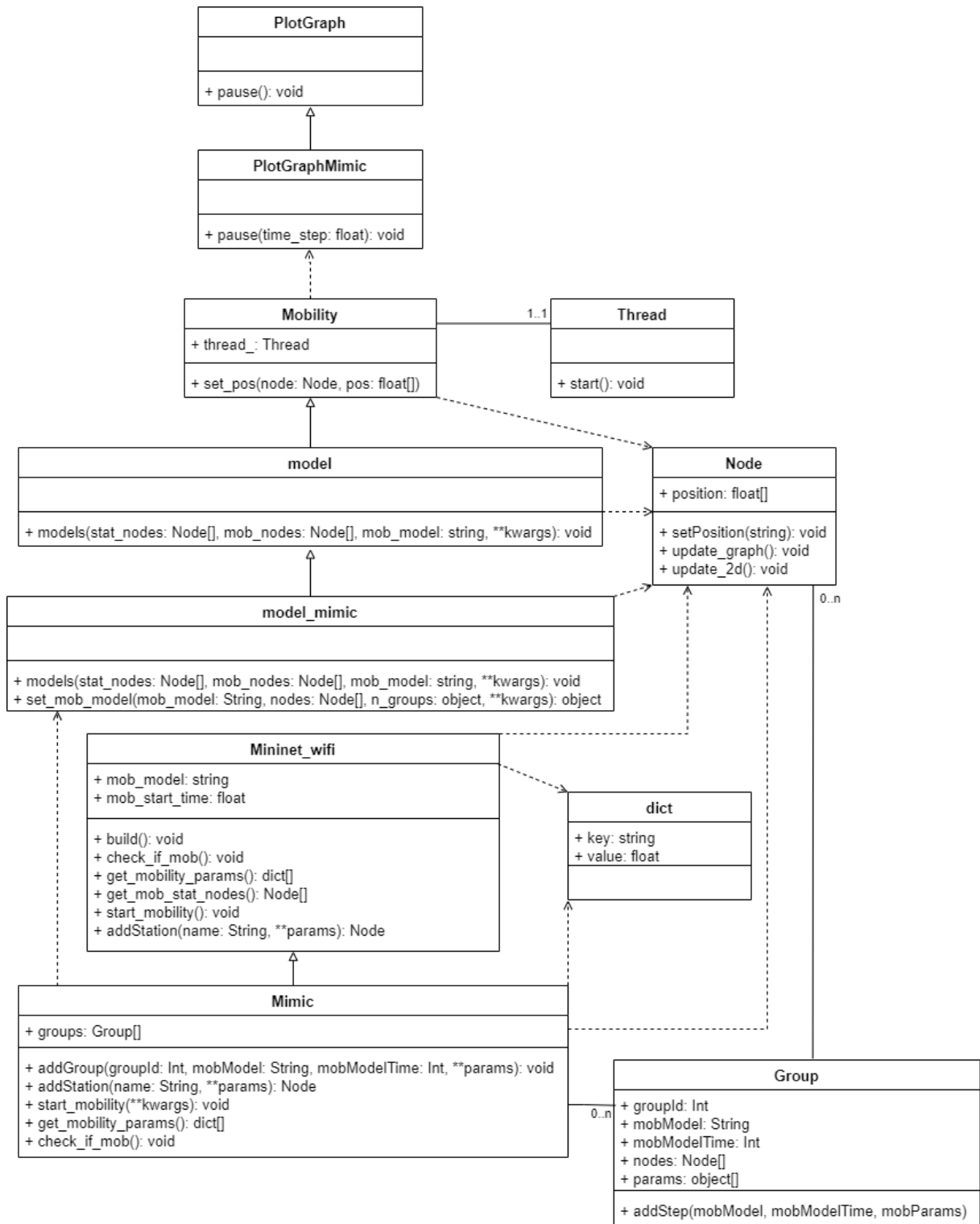


Figura 21 – Diagrama de classes da solução proposta

3.4.1.2 addStation

```
1 def addStation(self, name, cls=None, **params):
2     # Call default implementation
3     sta = super().addStation(name, cls, **params)
4
5     # Set initial position
6     if 'position' in params:
7         sta.params['initPos'] = params['position']
8
9     # Add nodes to groups
10    if 'groupId' in params:
11        for group in self.groups:
12            if group.groupId == params['groupId']:
13                group.nodes.append(sta)
14
15    return sta
```

Código 3.3 – Método para criação de uma estação

Esse método foi sobrescrito para possibilitar a inserção do identificador dos grupos dentro dos atributos dos nós, além disso nele é definida a posição inicial do nó baseada nos parâmetros passados ao método.

3.4.1.3 start_mobility

```
1 def start_mobility(self, **kwargs):
2     if self.groups:
3         for group in self.groups:
4             for node in group.nodes:
5                 if not hasattr(node, 'position'):
6                     node.position = (0, 0, 0)
7                 if not hasattr(node, 'pos'):
8                     node.pos = (0, 0, 0)
9             MobModel(**kwargs)
10    else:
11        super().start_mobility(**kwargs)
```

Código 3.4 – Método para inicialização da movimentação

Esse método foi alterado para definir a posição dos nós pertencentes aos grupos que estão armazenados na propriedade “groups” da classe `Mininet_wifi_ebx`.

3.4.1.4 get_mobility_params

```

1 def get_mobility_params(self):
2     mob_params = super().get_mobility_params()
3     mob_params['groups'] = self.groups
4     return mob_params

```

Código 3.5 – Método para recuperar os parâmetros de mobilidade

Esse método foi sobrescrito apenas para passar os grupos de mobilidade para a variável local `mob_params`.

3.4.1.5 pause

```

1 def pause(cls, time_step):
2     plt.pause(time_step)

```

Código 3.6 – Método para pausar simulação e atualizar elementos gráficos na tela.

Esse método foi criado na classe `PlotGraphMimic`, que herda de `PlotGraph`, para que o usuário consiga definir um passo de execução da simulação.

3.4.2 Configuração de Mobilidade da Solução

O fluxo chamado de “Configuração da Mobilidade” trata do momento onde o modelo de mobilidade e seus parâmetros são definidos. Tal processo ocorre majoritariamente dentro da classe `Mimic`. Seus passos podem ser observados na Figura 22 e estão descritos abaixo.

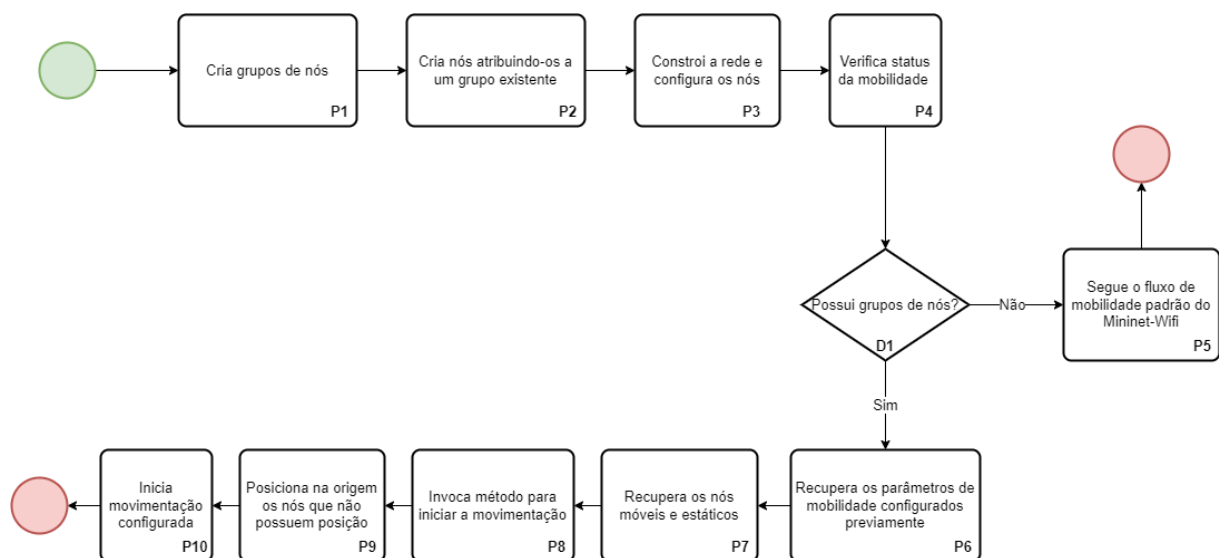


Figura 22 – Novo fluxo de configuração da mobilidade

- (P1): Primeiramente o usuário, no seu *script*, utiliza a função `addGroup` da classe `Mimic` para criar os grupos e configurar os parâmetros de interesse. Tais parâmetros consistem, primordialmente, da seleção dos modelos de mobilidade e do tempo no qual esses padrões estarão ativos.
- (P2): Após a criação dos grupos, é necessária a criação e configuração dos nós que farão partes dos grupos. É importante atentar que os modelos de mobilidade podem necessitar de parâmetros diferentes e estes podem estar definidos tanto nos grupos quanto nos nós.
- (P3): Dentro de seu *script* o usuário chama o método `build` pertencente à classe `Mimic`. Este método é responsável por construir a rede a partir da topologia fornecida, iniciar alguns serviços necessários para a emulação e configurar os *hosts* pertencentes à rede.
- (P4): O método `build` citado no processo anterior faz a invocação de um outro método da classe. Este outro método é chamado de `check_if_mob`. Tal função verifica se a classe possui algum valor configurado em `mob_model` e encaminha o fluxo da aplicação para iniciar a movimentação ou não.
- (P5): No caso de não possuir grupos, isso significará que o usuário não pretende usar a extensão criada do `Mimic`. Sendo assim, o código seguirá com a execução padrão do `Mininet_wifi`.
- (P6): Caso exista um modelo de mobilidade configurado, o método `get_mobility_params` é invocado para recuperar os parâmetros de mobilidade. Tais valores são salvos em uma variável local chamada de `mob_params`. Esse fluxo ocorre dentro do supracitado método `check_if_mob`.
- (P7): Em seguida, tenta-se recuperar os nós estáticos e móveis da rede. Isso é feito através do método `get_mob_stat_nodes`. Os valores retornados pela chamada da função são salvos em variáveis locais chamadas de `stat_nodes` e `mob_nodes`.
- (P8): Os parâmetros de mobilidade e os nós recuperados nos processos P6 e P7 são, por fim, passados para o método `start_mobility`, o qual é responsável pelos últimos passos de configuração.
- (P9): O método `start_mobility` irá passar por cada grupo e iterar por todos os seus nós, checando se estes possuem o atributo `position`. Caso não possuam eles são posicionados na origem das coordenadas, ou seja, `node.position = (0, 0, 0)`.
- (P10): Por fim, é instanciada a classe `model`. Essa instanciação é feita passando todos os parâmetros configurados até o momento no construtor da classe. Tal classe é a responsável por efetivamente realizar os passos para a movimentação dos nós na rede.

3.4.3 Fluxo de Mobilidade da Solução

O fluxo de mobilidade se refere à fase onde ocorre de fato a movimentação dos nós, a qual estará contida dentro da classe `model_mimic`. Esta é uma extensão da classe `model`, e tem como finalidade complementar o fluxo original para possibilitar o uso de grupos de nós com modelos de mobilidade diferente. Além disso, ela possibilita o uso de mais de um modelo de mobilidade por grupo de nós, utilizando-se de uma estrutura com durações bem definidas. Dessa forma, os valores das posições são recuperados do *iterator* e são utilizados para atualizar o gráfico com os nós na tela. Tal fluxo está ilustrado na Figura 23 e a descrição dos processos está disposta na sequência.

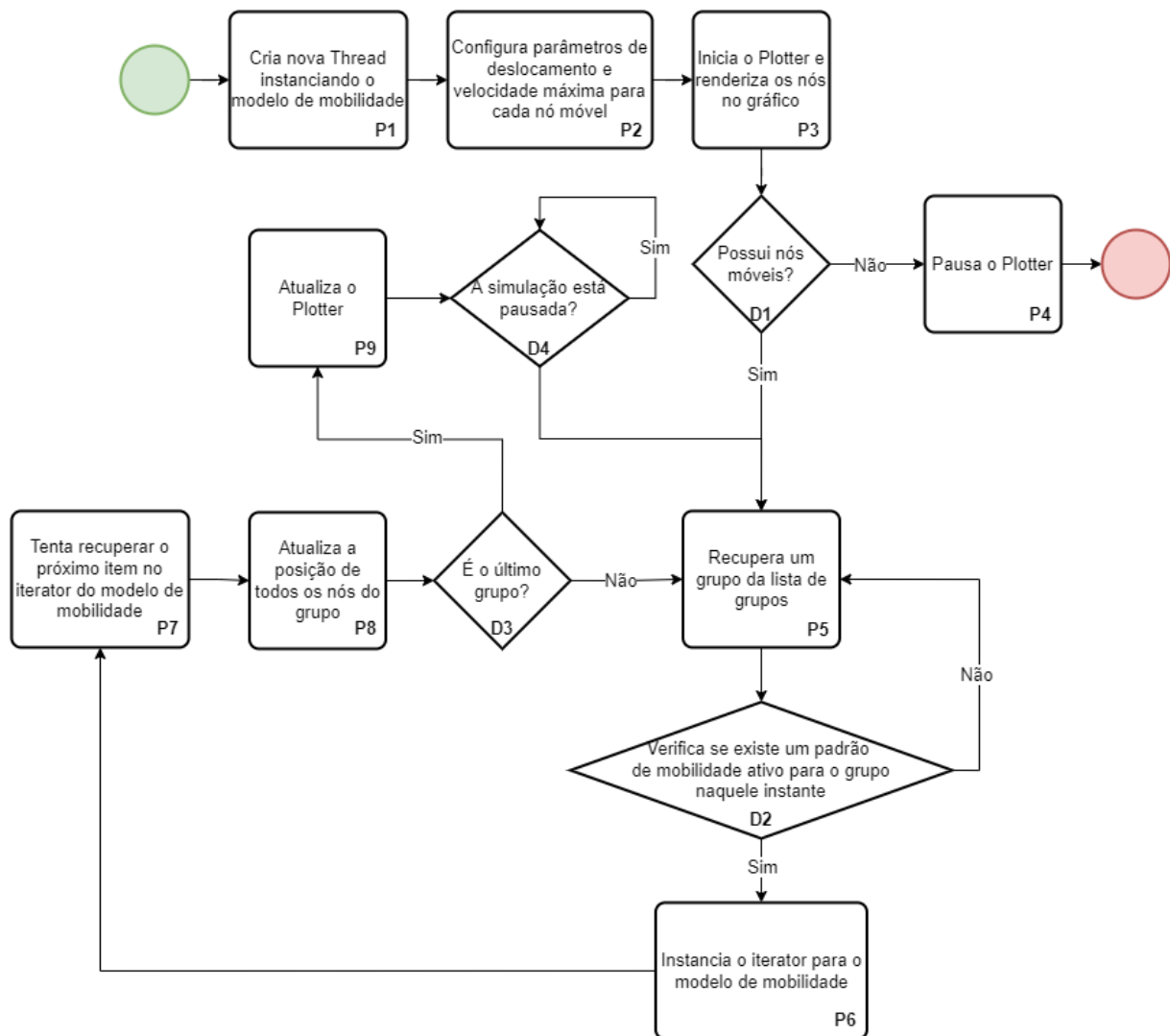


Figura 23 – Fluxo de mobilidade

(P1): O primeiro processo ocorre já no construtor da classe `model_ebx`. Nesse passo, é criada uma nova *thread* chamada “mobModel”. Tal *thread* possui como “target” (objeto que deverá ser acionado ao se executar a *thread*) o método `models`. Este método recebe como parâmetros todos os elementos que foram passados para o construtor

da classe. O novo objeto de *thread* criado é armazenado em uma propriedade da classe `Mobility`. Tal propriedade é chamada de `thread_`. Por fim, a *thread* em questão é inicializada, o que ocasiona a execução do “*target*”.

- (P2): Já dentro do método `models`, é executado um *loop* que itera sobre os grupos que foram passados como parâmetros para o método. Os nós pertencentes aos grupos são adicionados à variável local `mob_nodes`. Na sequência, os nós que estão dentro da variável `mob_nodes` são verificados e em cada um são configurados os parâmetros de posições mínimas e máximas nos eixos x e y juntamente com as velocidades mínimas e máximas aceitáveis na simulação. É importante ressaltar que os parâmetros “*min_x*”, “*min_y*”, “*min_v*” e “*max_v*” são configurados com valores padrão do *software*, não sendo possível alterá-los apenas os passando como parâmetros para o método.
- (P3): Nesse passo, o *Plotter* (objeto responsável pela criação da interface gráfica de interação do usuário) é inicializado através da classe `PlotGraphMimic`, a qual recebe como parâmetros os nós, as posições máximas em x e y, e todos os parâmetros extras que foram passados para o método `models`.
- (P4): No caso em que não existam grupos, o *Plotter* será pausado por meio do método `pause` disponibilizado pela classe `PlotGraphMimic` e a execução da *thread* é encerrada.
- (P5): Usa a lista de grupos passadas para o método `models` para iterar sobre cada grupo verificando se existe um padrão de mobilidade ativo naquele instante.
- (P6): No caso em que esteja ativo algum modelo de mobilidade neste momento, será instanciado um objeto daquele pedrão de mobilidade, o qual irá direcionar a movimentação dos nós.
- (P7): A classe do padrão de mobilidade terá um *iterator* que gerará a posição dos nós a cada instante e atualizará a posição interna de cada nó através do método `update_2d`, definida dentro da classe `node`.
- (P8): Já de posse das novas posições dos nós móveis é feita a atualização da posição de cada nó. Para isso é utilizado o método `set_pos` da classe pai `Mobility`. Tal método atribui o novo valor para o atributo `position` da classe `Node`.
- (P9): O *Plotter* é atualizado através de uma breve pausa com o método `pause` da classe `PlotGraphMimic`. Na sequência, é verificado se a simulação está pausada. Caso esteja pausada, o código permanece esperando a liberação da simulação para avançar. Já no caso contrário, o fluxo da mobilidade continua e o processo P5 é novamente executado.

4 DESENVOLVIMENTO

A fase de desenvolvimento do projeto consistiu em implementar as modificações estruturais apresentadas na seção 3.4.1, após isso os fluxos de mobilidade foram modificados seguindo o que foi definido na seção 3.4.2 deste trabalho. Após tais modificações estarem completas, iniciou-se o desenvolvimento dos modelos de mobilidade propriamente ditos. Para desenvolver tais modelos foram criadas algumas funções auxiliares que serão apresentadas a seguir.

O objetivo desta seção do projeto foi desenvolver os modelos necessários de forma a possibilitar a representação de movimentos de tropas reais no terreno. Os modelos aqui criados são intercambiáveis e fluidos, de forma que a simulação seja o mais próxima da realidade. Para possibilitar esta movimentação fluida foram utilizados alguns métodos geométricos e algébricos que serão apresentados a seguir.

Este capítulo foi estruturado visando um melhor entendimento do projeto desenvolvido. Para tanto, a primeira seção trata de funções auxiliares que foram importantes para o desenvolvimento dos modelos. Na sequência é apresentado o desenvolvido dos modelos em si.

4.1 Funções Auxiliares

4.1.1 increased_speed

```

1 def increased_speed(pred_dist, dist, v, mv):
2     if(dist <= pred_dist):
3         return v
4     elif(dist > 2*pred_dist):
5         return mv
6     else:
7         return round(dist*(mv-v)/pred_dist + 2*v - mv, 10)

```

Código 4.1 – Função para calcular nova velocidade do nó.

4.1.1.1 Parâmetros

- **pred_dist:** Distância alcançada do nó com a velocidade do grupo.
- **dist:** Distância do nó à posição desejada.
- **v:** velocidade de movimentação do grupo em que o nó está inserido.

- **mv**: velocidade máxima de movimentação do nó.

4.1.1.2 Descrição

Esse método tem como objetivo determinar a velocidade do nó a depender da velocidade máxima que ele pode atingir, a velocidade do grupo, a distância do nó até a posição desejada e a distância alcançada com a velocidade de grupo.

Esse método irá retornar **v** se a distância que ele consegue percorrer com a velocidade de grupo for maior que a distância até a posição desejada. Caso a distância à posição desejada seja maior que duas vezes a distância que o nó consegue atingir com a velocidade do grupo será retornada a velocidade máxima do nó. Por fim, no caso de a distância percorrível se encontrar entre **pred_dist** e **2*pred_dist**, será retornado um valor que variará linearmente à depender da distância ao ponto desejado. Dessa forma, essa função pode ser representada pelo seguinte gráfico representado na Figura 24.

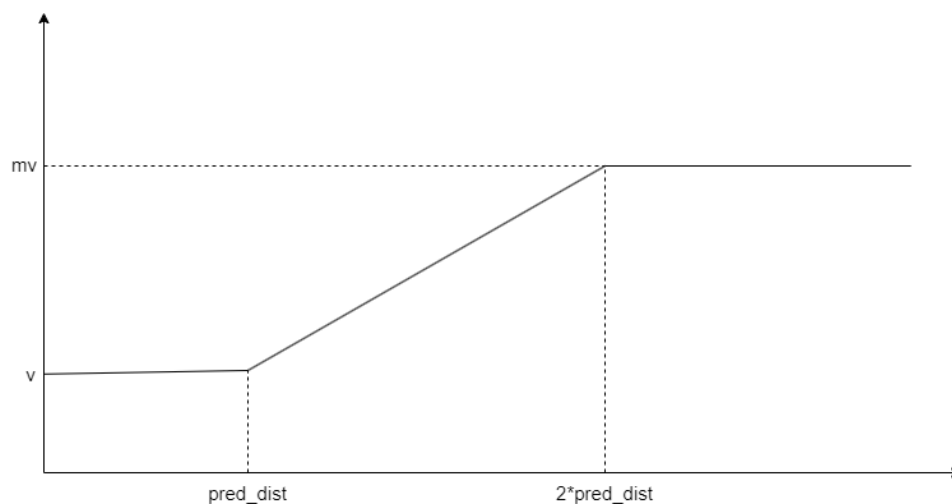


Figura 24 – Gráfico de velocidade do nó

4.1.2 project_position

```

1 def project_position(xc, yc, xf, yf, v, mv):
2     if(v > mv):
3         v = mv
4
5     xd = xf - xc
6     yd = yf - yc
7     dist = np.round(np.hypot(abs(xd), abs(yd)), 10)
8
9     ang = np.arctan2(yd, xd)
10
11     xp = xc + v*np.cos(ang)

```

```

12     yp = yc + v*np.sin(ang)
13     pred_dist = np.round(np.hypot(abs(xp - xc), abs(yp - yc)), 10)
14
15     if(dist <= pred_dist):
16         return (xf, yf)
17
18     if(v < mv):
19         v = increased_speed(pred_dist, dist, v, mv)
20
21     xp = xc + v*np.cos(ang)
22     yp = yc + v*np.sin(ang)
23     pred_dist = np.round(np.hypot(abs(xp - xc), abs(yp - yc)), 10)
24
25     if(dist <= pred_dist):
26         return (xf, yf)
27
28     return (xp, yp)

```

Código 4.2 – Função para calcular a posição do nó na próxima iteração.

4.1.2.1 Parâmetros

- **xc:** Coordenada x atual do nó.
- **yc:** Coordenada y atual do nó.
- **xf:** Coordenada x final desejada para o nó.
- **yf:** Coordenada y final desejada para o nó.
- **v:** Velocidade de movimentação do grupo em que o nó está inserido.
- **mv:** velocidade máxima de movimentação do nó.

4.1.2.2 Descrição

Esta função tem como objetivo calcular uma posição para o nó que seja realizável considerando as variáveis do problema. Para fazer tal cálculo é levado em conta a posição inicial do nó, uma posição desejada onde o nó deveria se posicionar, a velocidade de movimentação do grupo em que o nó está inserido e a velocidade máxima do nó. Uma representação visual desta função pode ser visualizada na Figura 25.

A função aceita como parâmetros a posição inicial do nó e a posição final desejada pelo usuário. A partir de tais informações é calculada a distância (`dist`) e o ângulo

(`ang`) em que o nó deverá se mover. De posse dessas informações e sabendo a velocidade de movimentação do nó, é possível achar a posição que o nó conseguirá alcançar. Após descobrir as coordenadas cartesianas de tal posição é feito o cálculo da distância entre a posição inicial e a posição final projetada (`pred_dist`). Com isso, existem dois possíveis cenários para a movimentação do nó em questão:

- **`dist ≤ pred_dist`**: Nesse caso, com sua velocidade atual, o nó conseguiria alcançar a distância desejada pelo usuário. Dessa forma, a função retorna a posição que o usuário inicialmente inseriu como sendo a desejada.
- **`dist > pred_dist`** No caso de o nó não conseguir alcançar a posição desejada com a velocidade atual, caso a velocidade atual seja menor do que a velocidade máxima do nó, é feito o incremento da velocidade utilizando-se da função `increased_speed`. Após alterar a velocidade, o cálculo da distância que o nó consegue alcançar é refeito, é calculada assim a nova distância prevista (`pred_dist`). De posse de tal distância verifica-se se o nó consegue alcançar o ponto desejado pelo usuário, caso consiga é retornado tal ponto. Já no caso de o nó não alcançar a posição desejada mesmo com a nova velocidade, é retornada a última posição realizável calculada.

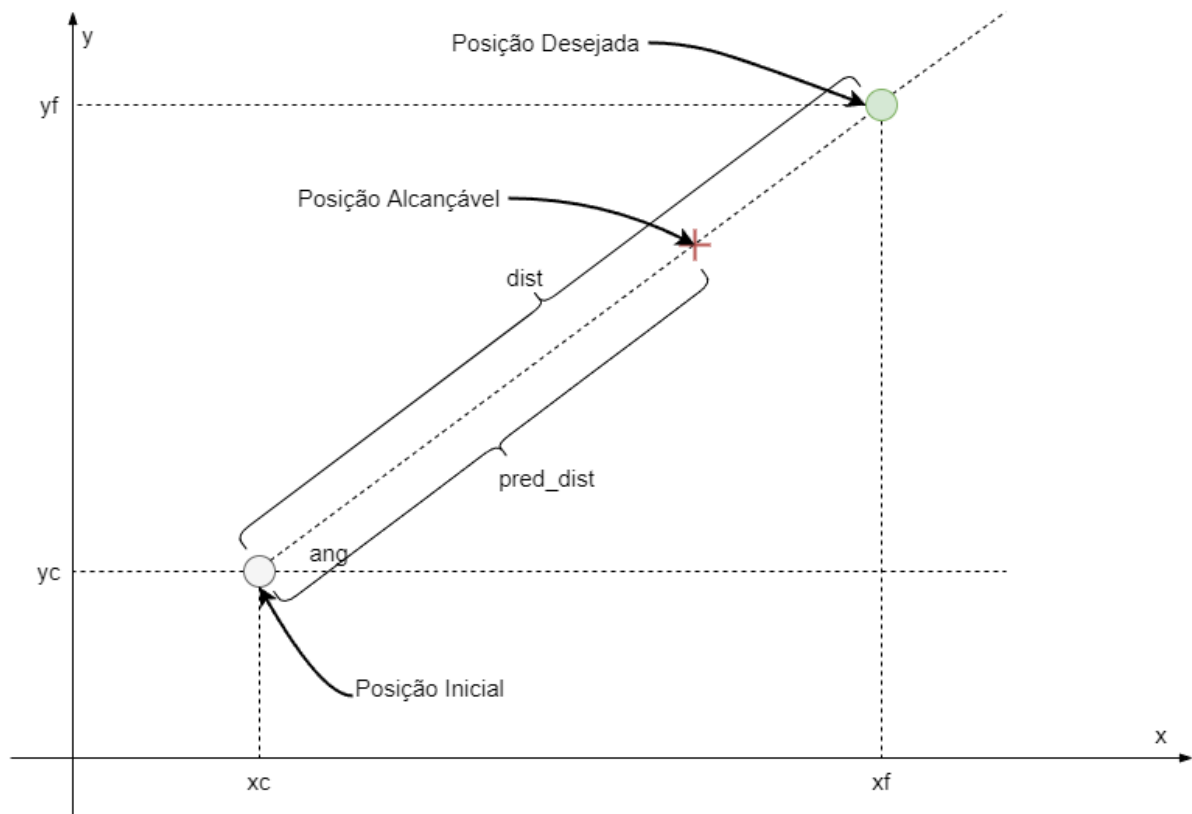


Figura 25 – Representação do cálculo da posição alcançável pelo nó

4.1.3 weighted_position

```
1 def weighted_position(base_pos, follow_pos, weight=0.5):  
2     return tuple(np.array(base_pos) * weight + np.array(follow_pos) *  
                  (1-weight))
```

Código 4.3 – para cálculo da posição ponderada do nó.

4.1.3.1 Parâmetros

- **base_pos:** Posição do nó caso seja considerado apenas a direção da movimentação.
- **follow_pos:** Posição do nó caso seja considerado apenas movimentação na direção do nó precursor.
- **weight:** Peso dado a movimentação na direção do nó precursor.

4.1.3.2 Descrição

O objetivo desta função é considerar duas possíveis posições que o nó pode alcançar e calcular um ponto que seja um meio termo entre as duas medidas. Para fazer tal cálculo é utilizado um parâmetro de peso (**weight**) que faz o cálculo da média ponderada entre as posições consideradas. Uma ilustração que representa a explicação a seguir pode ser observada na Figura 26.

A ideia de tal função vem principalmente do modelo de movimentação em linha, onde um nó deve seguir seu predecessor. Caso os nós estivessem seguindo na Direção 1 em um instante e no instante seguinte passam a se movimentar na Direção 2, existem duas possíveis de ação para a movimentação. A primeira leva em consideração apenas a posição do nó predecessor, faz-se o cálculo da nova posição do nó 1 baseado na posição do nó 0 e considerando sua velocidade. Com isso, é estimado um ponto (P1) que é a posição desejada do nó 1. Já na segunda forma de movimentação é levado em consideração a nova direção em que os nós devem se mover. Dessa forma, são utilizadas a posição do nó 0, a nova direção de movimentação (Direção 2) e a velocidade de movimentação para achar a posição desejada do nó 1 (P2).

Após serem calculados os pontos P1 e P2, alguns testes foram executados para verificar se a movimentação dos nós se aproximava da movimentação de uma tropa em linha. As trajetórias criadas a partir dos dois casos supracitados acabaram não sendo condizentes com o que se espera de uma fileira de veículos se movendo, isso gerou a necessidade de combinar ambas as trajetórias e calcular um terceiro ponto que fosse intermediário entre P1 e P2. Para achar o terceiro ponto considerou-se o seguinte algebrismo:

$$\Delta x = x2 - x1 \wedge x = x1 + w * \Delta x \implies x = x1 * (1 - w) + x2 * w$$

Analogamente:

$$y = y1 * (1 - w) + y2 * w$$

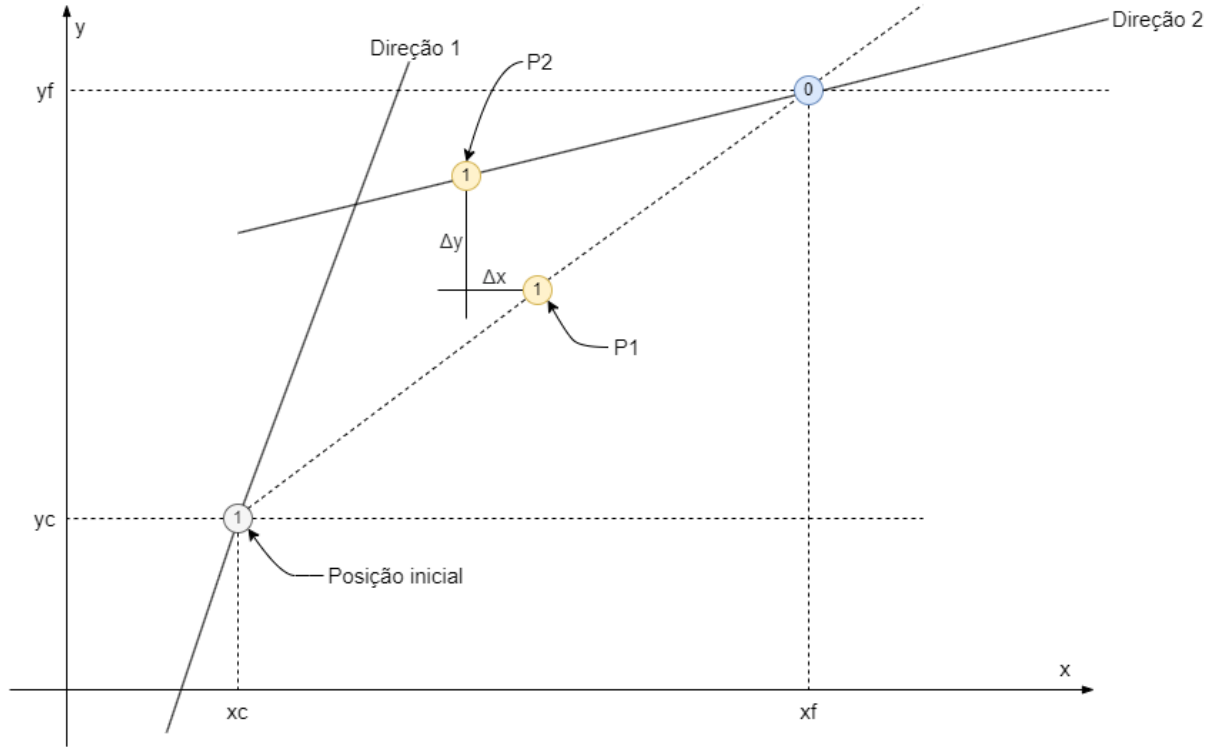


Figura 26 – Cálculo da posição ponderada do nó

4.2 Modelos de Mobilidade

4.2.1 Formação em Linha

```

1 def line_formation(nodes, angle, velocity, minDistance):
2     x = np.empty(len(nodes))
3     y = np.empty(len(nodes))
4     xn = np.empty(len(nodes))
5     yn = np.empty(len(nodes))
6
7     # Define minimum distance between nodes
8     minDistance = minDistance if minDistance else velocity
9
10    # Defining leader position
11    leader = 0
12    for i in range(len(nodes)):

```


[illegible]

```
50         (xn[i], yn[i]) = weighted_position(base_position,  
51                                           follow_position, 0.25)  
52     x[i] = xn[i]  
53     y[i] = yn[i]  
54  
55     yield np.dstack((x, y))[0]
```

Código 4.4 – Código da movimentação de nós em linha

4.2.1.1 Parâmetros

- **nodes:** Nós que seguirão esse modelo de mobilidade.
- **angle:** A direção de deslocamento dos nós.
- **velocity:** Velocidade de movimentação do grupo em que o nó está inserido.
- **minDistance:** Espaçamento mínimo entre nós.

4.2.1.2 Descrição

Esse modelo consiste na movimentação em linha dos nós, o qual terá um líder que guiará os demais nós.

Inicialmente calculamos a direção de movimentação como pode ser visto na linha 20 do Código 4.4. Após isso calculamos a cada iteração a nova posição do líder multiplicando a direção do movimento pela velocidade do grupo, como mostrado nas linhas 28 e 29.

Além disso, precisamos calcular como a posição dos demais nós irá variar. Para isso, usamos a posição do nó logo a frente quanto à numeração, à partir disso calculamos a distância ao nó da frente e verificamos se esta distância é menor que a distância mínima, caso seja o nó ficará parado, caso contrário iremos calcular duas possíveis posições desejadas. Tais posições seriam a posição base prevista, a qual se refere à uma posição sobre a direção de movimentação com uma distância de `minDistance` do nó da frente e a posição de perseguição a qual seria a posição sobre a reta que liga o nó atual ao nó da frente separados por uma distância igual a `minDistance`. Os pontos citados podem ser visualizados na Figura 26, onde os pontos amarelos são as posições desejadas.

Por fim, é usada a função `weighted_position` para ponderar a posição final à partir do peso dado as posições desejadas.



Figura 27 – Modelo de mobilidade de nós em formação em linha

4.2.2 Formação em Cunha

```

1 def wedge_formation(nodes, angle, velocity, minDistance):
2     x = np.empty(len(nodes))
3     y = np.empty(len(nodes))
4     xn = np.empty(len(nodes))
5     yn = np.empty(len(nodes))
6     positionX = np.empty(len(nodes))
7     positionY = np.empty(len(nodes))
8
9     # Defining leader position
10    leader = 0
11    for i in range(len(nodes)):
12        x[i] = nodes[i].position[0]
13        y[i] = nodes[i].position[1]
14        if nodes[i].params.get("leader", False):
15            leader = i
16
17    # Calculating movement direction
18    direction = (math.cos(angle), math.sin(angle))
19    ortDirection = (math.sin(angle), -math.cos(angle))
20
21    for num in range(len(nodes)):
22        i = (leader + num)%len(nodes)
23        positionX[i] = x[leader] + ortDirection[0]*minDistance*num
24        positionY[i] = y[leader] + ortDirection[1]*minDistance*num
25
26    while True:
27        for count in range(len(nodes)):
28            i = (leader + count)%len(nodes)
29
30            if i == leader:
31                xn[i] = x[i] + direction[0]*velocity
32                yn[i] = y[i] + direction[1]*velocity
33
34            for num in range(len(nodes)):
35                j = (leader + num)%len(nodes)

```

```

36         positionX[j] = xn[leader] + ortDirection[0]*minDistance*num
37         positionY[j] = yn[leader] + ortDirection[1]*minDistance*num
38     else:
39         prev = (i-1+len(nodes))%len(nodes)
40
41         follow_position = project_position(nodes[i].position[0],
42                                           nodes[i].position[1], positionX[i], positionY[i], velocity,
43                                           nodes[i].max_v)
44
45         (xn[i], yn[i]) = follow_position
46
47         x[i] = xn[i]
48         y[i] = yn[i]
49
50     yield np.dstack((x, y))[0]

```

Código 4.5 – Código da movimentação de nós em formato de cunha

4.2.2.1 Parâmetros

- **nodes:** Nós que seguirão esse modelo de mobilidade.
- **angle:** Coeficiente angular da reta de alinhamento.
- **velocity:** Velocidade de movimentação do grupo em que o nó está inserido.
- **minDistance:** Espaçamento mínimo entre nós.

4.2.2.2 Descrição

Esse modelo se baseia numa movimentação ortogonal ao alinhamento dos nós, os quais estão espaçados igualmente, como pode ser visto na Figura 28.

A execução deste modelo pode ser dividida nas seguintes etapas:

- A reta de alinhamento será definida por meio do ângulo escolhido pelo usuário e pelo nó líder, o qual será definido pelo usuário ou, no caso do usuário não definir um líder, será considerado o nó zero.
- É feita a determinação do alinhamento dos nós. Esta é feita de forma em que o líder será o primeiro nó na reta e os nós posteriores seguirão a numeração estipulada pelo usuário (ordem de criação dos nós). Por exemplo, se o líder for o nó 2 e o grupo possuir 4 elementos, os nós serão dispostos da seguinte maneira: 2, 3, 0, 1.

- Por último, é feita a atualização da posição dos nós. Esta tem como base o incremento da posição do líder de acordo com a posição atual dele e a velocidade do grupo, velocidade esta que é perpendicular à reta de alinhamento dos nós. Após isso será calculado a nova posição da reta de alinhamento e as posições esperadas dos demais nós. Finalmente o método `project_position` é utilizado para determinar a posição dos nós.

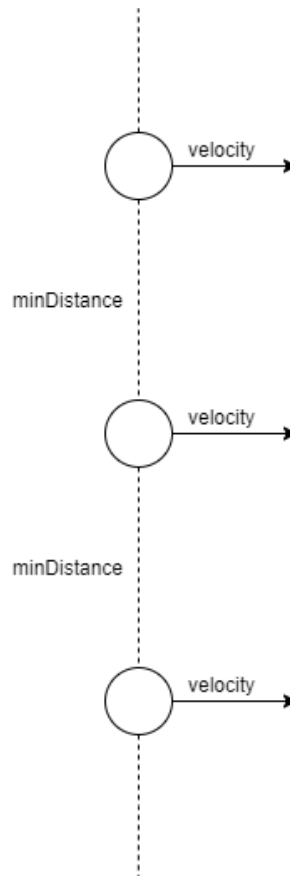


Figura 28 – Modelo de mobilidade de nós em formação em cunha

4.2.3 Formação Circular

```

1 def rotation(nodes, centro, velocity, minDistance):
2     x = np.empty(len(nodes))
3     y = np.empty(len(nodes))
4     xn = np.empty(len(nodes))
5     yn = np.empty(len(nodes))
6     angle = 0
7     raio = 0
8
9     leader = 0
10    for i in range(len(nodes)):

```

```

11     x[i] = nodes[i].position[0]
12     y[i] = nodes[i].position[1]
13     if nodes[i].params.get("leader", False):
14         leader = i
15         raio = np.hypot(abs(nodes[i].position[0]-centro[0]),
16                        abs(nodes[i].position[1]-centro[1]))
17         angle = np.arctan2((nodes[i].position[1]-centro[1]),
18                            nodes[i].position[0]-centro[0])
19
20     while True:
21         for count in range(len(nodes)):
22             i = (leader + count)%len(nodes)
23
24             if i == leader:
25                 angle = angle + velocity/raio
26                 xn[i] = centro[0] + raio*math.cos(angle)
27                 yn[i] = centro[1] + raio*math.sin(angle)
28             else:
29                 prev = (i-1+len(nodes))%len(nodes)
30
31                 xd = nodes[prev].position[0] - nodes[i].position[0]
32                 yd = nodes[prev].position[1] - nodes[i].position[1]
33                 dist = np.round(np.hypot(abs(xd), abs(yd)), 10)
34                 if(dist < minDistance):
35                     continue
36
37                 f_angle = np.arctan2(y[prev] - y[i], x[prev] - x[i])
38
39                 x_follow = (x[prev] - np.cos(f_angle)*minDistance)
40                 y_follow = (y[prev] - np.sin(f_angle)*minDistance)
41
42                 follow_position = project_position(nodes[i].position[0],
43                                                    nodes[i].position[1], x_follow, y_follow, velocity,
44                                                    nodes[i].max_v)
45
46                 (xn[i], yn[i]) = follow_position
47                 x[i] = xn[i]
48                 y[i] = yn[i]
49         yield np.dstack((x, y))[0]

```

Código 4.6 – Código da movimentação de nós em formato circular

4.2.3.1 Parâmetros

- **nodes:** Nós que seguirão esse modelo de mobilidade.
- **center:** Centro da trajetória circular que os nós irão seguir.
- **velocity:** Velocidade de movimentação do grupo em que o nó está inserido.
- **minDistance:** Espaçamento mínimo entre nós.

4.2.3.2 Descrição

Esse modelo consiste na movimentação dos nós em torno de uma trajetória circular, como pode ser visto na Figura 29.

Inicialmente são calculados o raio e o ângulo da circunferência a partir da posição do líder e da posição do centro da circunferência, isso pode ser visto nas linhas 15 e 16 do Código 4.6.

Após isso, é calculado a cada iteração a posição do líder e o novo ângulo que ele faz, como pode ser vistos nas linhas 23, 24 e 25. Já os outros nós seguirão uma abordagem semelhante ao modelo de formação em linha, utilizando-se da função `follow_position` para seguir o nó da frente.

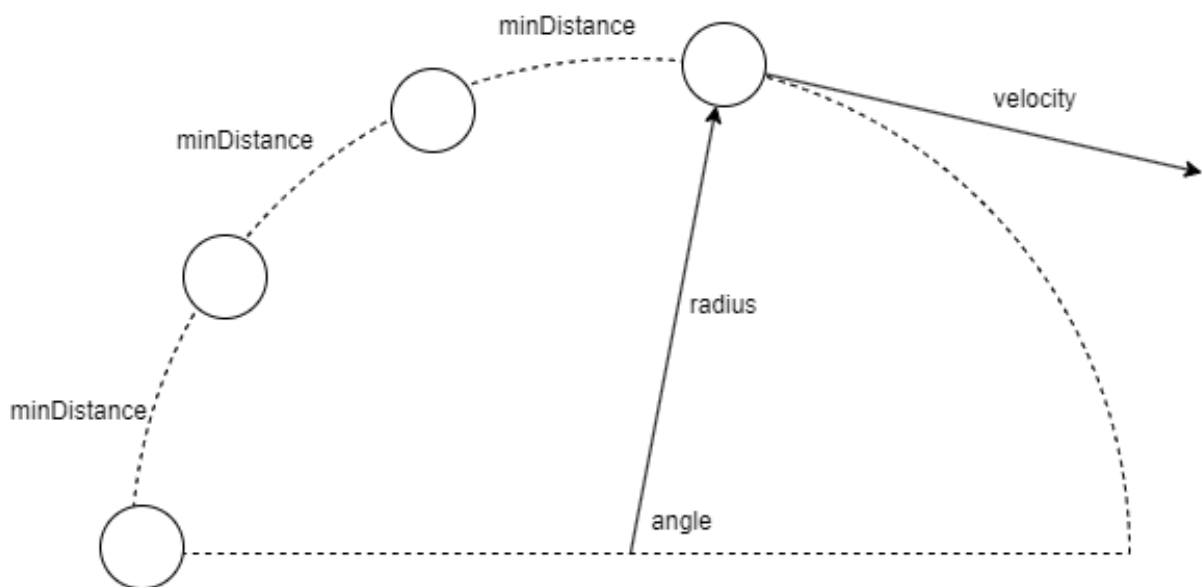


Figura 29 – Modelo de mobilidade de nós em formação circular

5 TESTES

Este capítulo abordará os testes utilizados para validar e ajustar os modelos implementados neste trabalho. Serão abordadas as especificidades de cada modelo e as consequências dos testes no aperfeiçoamento dos algoritmos utilizados, além de apresentar os resultados encontrados no projeto.

5.1 Formação Linha

O modelo de formação em linha foi o primeiro modelo abordado durante o desenvolvimento deste trabalho. Esse modelo foi bastante importante para direcionar o desenvolvimento dos demais, por conta disso seus testes foram de suma importância tanto para extrair conclusões sobre o projeto quanto para aperfeiçoar a fluidez dos modelos em geral.

Os teste utilizaram o *template* de código abaixo para validar e refinar o modelo:

```

1 import sys
2 sys.path.append('core')
3 sys.path.insert(0, '../core')
4 from net_ebx import Mininet_wifi_ebx
5 from mn_wifi.cli import CLI
6 from math import pi
7
8 # Creating network
9 net = Mininet_wifi_ebx()
10
11 # Creating groups
12 group_one = net.addGroup(1, [], [], [])
13
14 # Adding mobility steps
15 group_one.addStep('line_formation', [0, 50], {'angle':pi/2, 'velocity':1,
16         'min_distance':200})
17 group_one.addStep('line_formation', [50, 400], {'angle':0, 'velocity':2,
18         'min_distance':200})
19
20 # Creating nodes
21 net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8', max_v=10,
22         groupId=1,leader=True, position='0,0,0')
23 net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8', max_v=5,

```



```

    groupId=1, position='0,-100,0')
21 net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8', max_v=10,
    groupId=1, position='0,-200,0')
22 net.addStation('sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/8', max_v=5,
    groupId=1, position='0,-300,0')
23 ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1',
    failMode="standalone", position='1000,1000,0')
24
25 net.configureWifiNodes()
26 net.plotGraph(min_x=-100, min_y=-100, max_x=600, max_y=600)
27 net.build()
28 ap1.start([])
29 CLI(net)
30 net.stop()

```

Código 5.1 – Teste do modelo de formação em linha. (`test_line_formation.py`)

Os seguintes testes foram realizados com a finalidade de refinar o modelo:

- Teste com intervalos de tempos em que os nós ficaram ociosos. Como por exemplo, definindo nas linhas 21 e 22 do código 5.3 os intervalos de $[0, 100]$ e $[200, 300]$ nos métodos `addStep`, isso fez com que o líder ficasse parado do tempo 100 ao tempo 200. Com esse teste foi possível perceber que como os demais nós estavam seguindo o líder, todos ficaram sobrepostos no mesmo ponto quando onde o líder estava parado. Isso levou a conclusão de que seria necessário implementar uma distância mínima entre os nós.
- Testes posicionando os nós em pontos iniciais randômicos, realizados através da alteração do parâmetro `position` na função `addStation` nas linhas 26 a 30. Com isso foi possível aferir que os nós se movimentariam no sentido contrário para manter a distância ao nó da frente maior que a distância mínima, dessa forma chegamos a conclusão que o ideal seria manter os nós parados até que a distância ao nó da frente fosse maior que a distância mínima.

Por fim, foram feitos diversos testes para obter a confirmação que o modelo está funcionando como o planejado:

- Teste na transição de outros modelos para o modelo de `line_formation`, visando verificar se os nós farão a conversão corretamente, alterando suas trajetórias e velocidades dentro do limite estipulado para alcançar o alinhamento previsto.

- Testes com vários passos utilizando-se do mesmo modelo, a diferença dos passos foi apenas o azimuth de movimentação. Isso foi feito para aferir o impacto da variável `weight` no cálculo da posição ponderada do ponto.

Os testes de feitos nesse modelo mostraram que ele consegue representar de forma coerente o movimento real esperado.

5.2 Formação Cunha

O modelo de formação em cunha herda vários métodos e princípios do modelo de formação em linha. Um desses princípios é o de calcular uma posição desejada e usar a função `project_position` para determinar para onde determinado nó seguirá. Este modelo difere do anterior pelo fato de considerar apenas a distância prevista para seguir o nó à sua frente (`follow_position`).

Os teste utilizaram o *template* de código abaixo para validar o modelo:

```
1 import sys
2 sys.path.append('core')
3 sys.path.insert(0, '../core')
4 from net_ebx import Mininet_wifi_ebx
5 from mn_wifi.cli import CLI
6 from math import pi
7
8 # Creating network
9 net = Mininet_wifi_ebx()
10
11 # Creating groups
12 group_one = net.addGroup(1, [], [], [])
13
14 # Adding mobility steps
15 group_one.addStep('wedge_formation', [0, 100], {'angle':pi/4, 'velocity':1,
16         'min_distance':50})
17 group_one.addStep('line_formation', [100, 300], {'angle':0, 'velocity':3,
18         'min_distance':50})
19 group_one.addStep('wedge_formation', [300, 400], {'angle':-pi/4, 'velocity':1,
20         'min_distance':50})
21
22 # Creating nodes
23 net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8', max_v=10,
24         groupId=1,leader=True, position='0,0,0')
```

```
21 net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8', max_v=5,  
    groupId=1, position='0,-100,0')  
22 net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8', max_v=10,  
    groupId=1, position='0,-200,0')  
23 net.addStation('sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/8', max_v=5,  
    groupId=1, position='0,-300,0')  
24 ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1',  
    failMode="standalone", position='1000,1000,0')  
25  
26 net.configureWifiNodes()  
27 net.plotGraph(min_x=-300, min_y=-300, max_x=1000, max_y=1000)  
28 net.build()  
29 ap1.start([])  
30 CLI(net)  
31 net.stop()
```

Código 5.2 – Teste do modelo de formação em cunha. (`test_wedge_formation.py`)

Foram efetuados os seguintes teste para validação do modelo:

- Teste na transição de outros modelos para o modelo da formação em cunha. Esse teste teve como objetivo verificar se os nós iriam executar a conversão correta, alterando suas trajetórias e velocidade dentro do limite estipulado, para alcançar o alinhamento previsto no modelo.
- Teste com diversas transições dentro do mesmo modelo, diferindo apenas dos parâmetros utilizados em cada passo. Esse teste foi feito para verificar se os nós iriam se reorganizar corretamente, seguindo os comportamentos esperados para a alteração em cada parâmetro.

A execução dos testes ocorreu como o esperado, validando o modelo implementado.

5.3 Formação Circular

O modelo de formação circular, assim como o de formação em cunha, herda vários métodos e princípios do modelo de formação em linha. A posição do nó é calculada baseada na posição do nó a sua frente e a função `project_position` é novamente utilizada para estimar uma posição alcançável pelo nó. A diferença deste modelo reside no cálculo do posicionamento do nó líder, tal nó é posicionado em uma trajetória circular que depende dos parâmetros inseridos pelo usuário.

Os teste do modelo utilizaram o *template* de código abaixo para validar a movimentação:

```

1 import sys
2 sys.path.append('core')
3 sys.path.insert(0, '../core')
4 from net_ebx import Mininet_wifi_ebx
5 from mn_wifi.cli import CLI
6 from math import pi
7
8 # Creating network
9 net = Mininet_wifi_ebx()
10
11 # Creating groups
12 group_one = net.addGroup(1, [], [], [])
13
14 # Adding mobility steps
15 group_one.addStep('line_formation', [0, 50], {'angle':pi/2, 'velocity':1,
16         'min_distance':50})
17 group_one.addStep('rotation', [50, 400], {'center':[100, 0], 'velocity':-5,
18         'min_distance':50})
19
20 # Creating nodes
21 net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/8', max_v=10,
22         groupId=1, leader=True, position='0,0,0')
23 net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/8', max_v=5,
24         groupId=1, position='0,-100,0')
25 net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/8', max_v=10,
26         groupId=1, position='0,-200,0')
27 net.addStation('sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/8', max_v=5,
28         groupId=1, position='0,-300,0')
29 ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1',
30         failMode="standalone", position='1000,1000,0')
31
32 net.configureWifiNodes()
33 net.plotGraph(min_x=-100, min_y=-100, max_x=600, max_y=600)
34 net.build()
35 ap1.start([])
36 CLI(net)
37 net.stop()

```

Código 5.3 – Teste do modelo de formação circular. (`test_rotation_formation.py`)

Foram efetuados os seguintes teste para validação do modelo:

- Teste variando o posicionamento dos nós, com a finalidade de verificar se os nós respeitarão a distância mínima. Esse teste foi necessário pois tanto nesse modelo quanto na formação em linha, os nós devem ficar parados caso a distância entre os nós seja menor do que a distância mínima definida pelo usuário.
- Teste na transição de outros modelos para o modelo circular. Esse teste teve como finalidade verificar se os nós farão a conversão correta, alterando suas trajetórias e velocidades dentro dos limites estipulados para alcançar a formação prevista no modelo.

A execução dos testes ocorreu como o esperado, validando o modelo implementado.

5.4 Movimentação Composta

Esse teste consiste na utilização de diversos modelos de mobilidade, alternando de um modelo para o outro na tentativa de representar mais fielmente o deslocamento em operações complexas, onde existe mais de um tipo de formação durante sua execução.

Os testes desse tipo de movimentação usou o seguinte *template*, o qual está inserido na pasta de testes do projeto (13).

```

1 import sys
2 sys.path.append('core')
3 sys.path.insert(0, '../core')
4 from net_ebx import Mininet_wifi_ebx
5 from mn_wifi.cli import CLI
6 from mininet.log import info
7 from mn_wifi.link import wmediumd
8 from mn_wifi.wmediumdConnector import interference
9 from math import pi
10
11 # Creating network
12 net = Mininet_wifi_ebx(link=wmediumd, wmediumd_mode=interference)
13
14 # Creating groups
15 group_one = net.addGroup(1, [], [], [])
16 group_two = net.addGroup(2, [], [], [])
17
18 # Adding mobility steps Group 1
19 group_one.addStep('line_formation', [0, 100], {'angle':-pi/3, 'velocity':3,
20         'min_distance':25})
21 group_one.addStep('wedge_formation', [200, 400], {'angle':pi/4, 'velocity':1,
22         'min_distance':30})

```

```

21 group_one.addStep('rotation', [400, 500], {'center':[400, -100], 'velocity':5,
    'min_distance':20})
22
23 # Adding mobility steps Group 2
24 group_two.addStep('line_formation', [0, 200], {'angle':0, 'velocity':3,
    'min_distance':50})
25 group_two.addStep('wedge_formation', [200, 400], {'angle':pi/4, 'velocity':1,
    'min_distance':70})
26 group_two.addStep('rotation', [400, 500], {'center':[425, -50], 'velocity':-5,
    'min_distance':40})
27
28 # Creating nodes
29 net.addStation('sta1', mac='00:00:00:00:00:02', ip='10.0.0.2/16', max_v=10,
    groupId=1, leader=True, position='0,0,0')
30 net.addStation('sta2', mac='00:00:00:00:00:03', ip='10.0.0.3/16', max_v=5,
    groupId=1, position='0,-100,0')
31 net.addStation('sta3', mac='00:00:00:00:00:04', ip='10.0.0.4/16', max_v=10,
    groupId=1, position='0,-200,0')
32 net.addStation('sta4', mac='00:00:00:00:00:05', ip='10.0.0.5/16', max_v=5,
    groupId=1, position='0,-300,0')
33
34 net.addStation('sta5', mac='00:00:00:00:00:06', ip='10.0.0.6/16', max_v=15,
    groupId=2, leader=True, position='100,100,0')
35 net.addStation('sta6', mac='00:00:00:00:00:07', ip='10.0.0.7/16', max_v=10,
    groupId=2, position='100,200,0')
36 net.addStation('sta7', mac='00:00:00:00:00:08', ip='10.0.0.8/16', max_v=10,
    groupId=2, position='100,300,0')
37 net.addStation('sta8', mac='00:00:00:00:00:09', ip='10.0.0.9/16', max_v=15,
    groupId=2, position='100,400,0')
38
39 ap1 = net.addAccessPoint('ap1', ssid='new-ssid', mode='g', channel='1',
    failMode="standalone", position='425,-50,0')
40
41 info("*** Configuring Propagation Model\n")
42 net.setPropagationModel(model="logDistance", exp=4)
43
44 info("*** Configuring wifi nodes\n")
45 net.configureWifiNodes()
46
47 net.plotGraph(min_x=-400, min_y=-400, max_x=1000, max_y=1000)
48 net.build()
49 ap1.start([])

```

```
50 CLI(net)
51 net.stop()
```

Código 5.4 – Teste de movimentação composta. (`test_hybrid_moviment.py`)

Os testes dessa movimentação tiveram com finalidade verificar se o comportamento geral do projeto estava adequado. Buscou-se verificar se as modificações estruturais do projeto para suportar a divisão em grupo de nós, padrões de movimentação em intervalos bem definidos e a composição de mais de um padrão de movimentação estavam dentro do esperado. Além disso, este teste foi utilizado para verificar a coerência dos modelos utilizados na movimentação. Dessa forma, o teste foi idealizado com mais de um grupo, contando com variados modelos de mobilidade e alteração em parâmetros como posição, quantidade de grupos, velocidade e intervalo de atuação dos modelos de mobilidade. Ademais, nesse teste foi verificada se a conexão de rede entre os nós estava funcional, seguindo as restrições de alcance e de conectividade com o ponto de acesso.

O resultado do teste foi positivo e dentro do esperado. As transições de modelo ocorreram de forma fluida e consistente, os intervalos de tempo foram respeitados e as velocidades foram seguidas corretamente. O teste de conectividade dos pontos foi realizado após o fim da movimentação dos nós, e foi realizado com os elementos do grupo 1. Esses elementos foram escolhidos pois sua posição final ficou dentro do alcance do ponto de acesso (ap1), essa posição final pode ser observada na Figura 30. O resultado do teste de conexão está ilustrado na Figura 31.

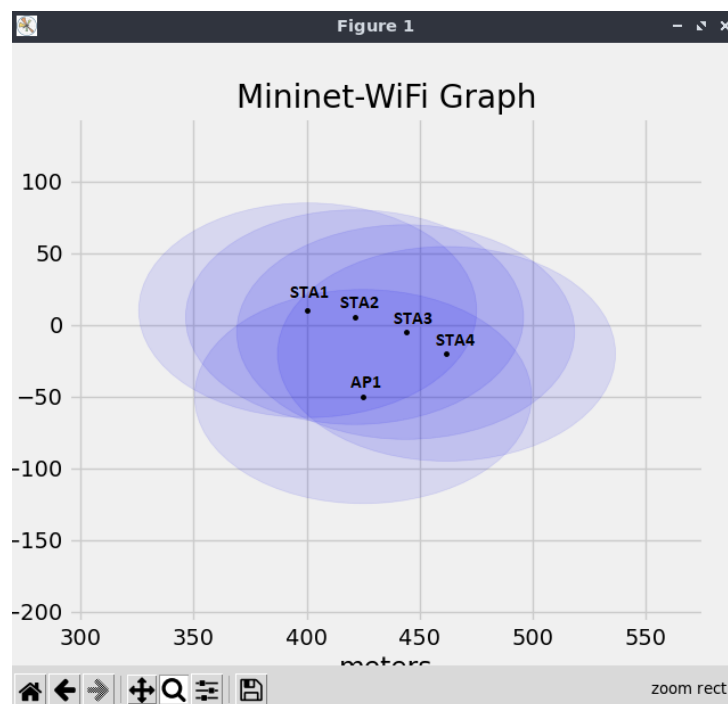


Figura 30 – Posição final dos elementos do grupo 1 no teste.

```
dpontes@pfc:~/mn-wifi-ebx/tests$ sudo python test_hybrid_moviment.py
mininet-wifi> stal ping -c4 ap1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.091 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.039 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.049 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.027 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3071ms
rtt min/avg/max/mdev = 0.027/0.051/0.091/0.024 ms
mininet-wifi> stal ping -c4 sta2
PING 10.0.0.3 (10.0.0.3) 56(84) bytes of data.
64 bytes from 10.0.0.3: icmp_seq=1 ttl=64 time=7.55 ms
64 bytes from 10.0.0.3: icmp_seq=2 ttl=64 time=1.22 ms
64 bytes from 10.0.0.3: icmp_seq=3 ttl=64 time=6.11 ms
64 bytes from 10.0.0.3: icmp_seq=4 ttl=64 time=1.19 ms

--- 10.0.0.3 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3003ms
rtt min/avg/max/mdev = 1.188/4.016/7.554/2.860 ms
mininet-wifi> stal ping -c4 sta3
PING 10.0.0.4 (10.0.0.4) 56(84) bytes of data.
64 bytes from 10.0.0.4: icmp_seq=1 ttl=64 time=4.69 ms
64 bytes from 10.0.0.4: icmp_seq=2 ttl=64 time=1.30 ms
64 bytes from 10.0.0.4: icmp_seq=3 ttl=64 time=1.78 ms
64 bytes from 10.0.0.4: icmp_seq=4 ttl=64 time=1.73 ms

--- 10.0.0.4 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 1.296/2.376/4.693/1.350 ms
mininet-wifi> stal ping -c4 sta4
PING 10.0.0.5 (10.0.0.5) 56(84) bytes of data.
64 bytes from 10.0.0.5: icmp_seq=1 ttl=64 time=10.8 ms
64 bytes from 10.0.0.5: icmp_seq=2 ttl=64 time=1.59 ms
64 bytes from 10.0.0.5: icmp_seq=3 ttl=64 time=1.81 ms
64 bytes from 10.0.0.5: icmp_seq=4 ttl=64 time=5.18 ms

--- 10.0.0.5 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 1.588/4.838/10.776/3.711 ms
mininet-wifi> █
```

Figura 31 – Teste de conexão entre nós do grupo 1.

5.5 Representação Anglova

O teste de representação do Anglova é o ponto culminante de todo este projeto, visto que ele busca representar movimentos de tropas reais a partir de dados georreferenciados obtidos a partir de operações efetuadas no terreno.

Antes de falar sobre os testes realizados nesta etapa é interessante entender o que é o cenário Anglova. O Anglova nada mais é do que um cenário de emulação desenvolvido pela OTAN, mais especificamente pelo grupo de pesquisa de redes heterogêneas (NATO IST-124 Research Task Group), para realizar experimentos em ambientes de redes táticas militares (1). Através deste cenário foram disponibilizados diversos dados, dentre eles está o posicionamento georreferenciado das tropas ao longo da operação.

Sabendo disso, o objetivo desse teste foi de representar, de forma reduzida, a movimentação das tropas presentes no cenário Anglova, dentro do emulador Mininet-Wifi. Para tal, iniciou-se por escolher uma das companhias presentes no cenário para fazer a representação. Isso foi feito através da análise de um pequeno vídeo disponível no portal do cenário Anglova (1), onde é possível verificar a disposição dos elementos das companhias ao longo da realização do exercício no terreno. A Figura 32 é um extrato do supracitado vídeo e representa a posição dos elementos da simulação, divididos por companhia, após

50 minutos do início da operação.

Após analisar tal vídeo e conversar com os interessados no projeto, optou-se por fazer a representação da companhia 3 com um número de elementos reduzido. Tal companhia foi escolhida por apresentar um padrão de movimento mais simples de ser representado.

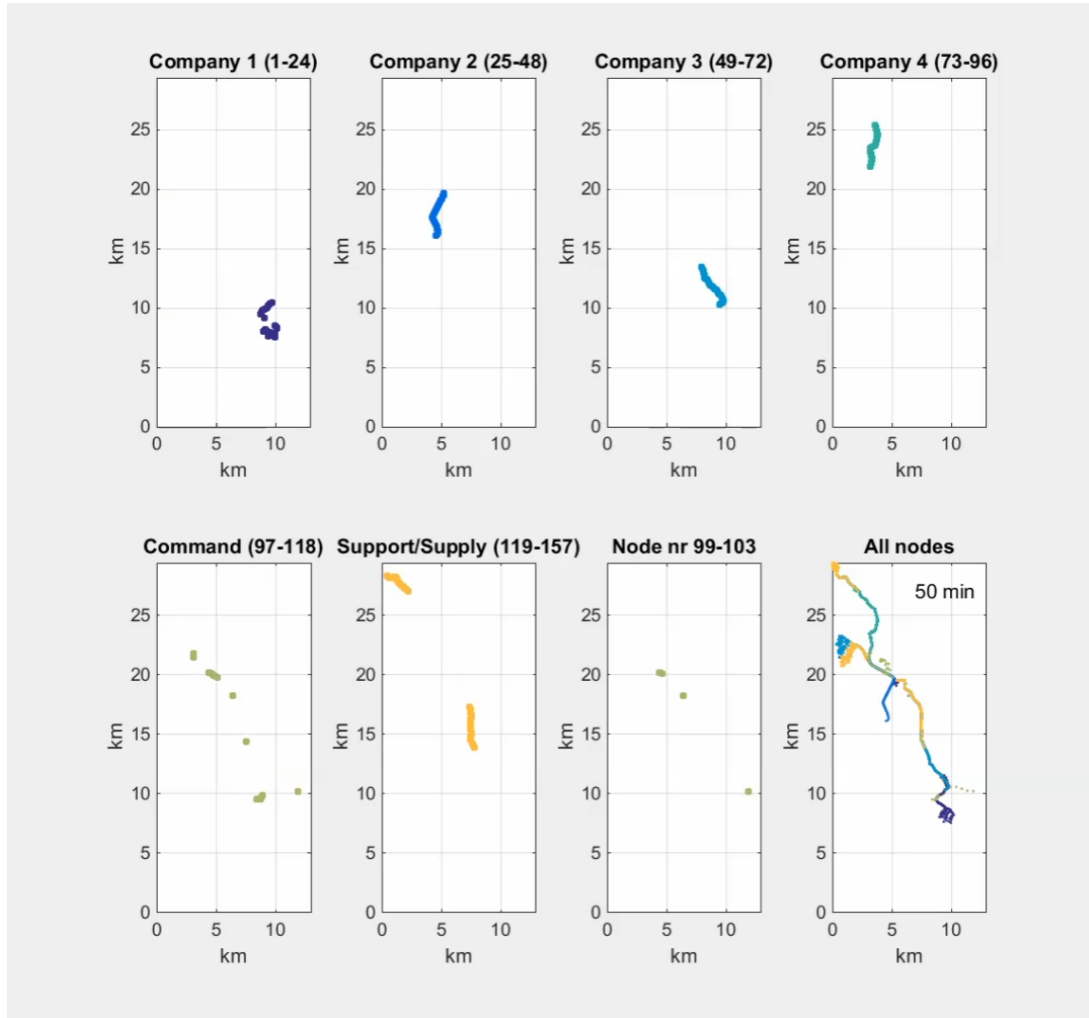


Figura 32 – Disposição espacial dos elementos da simulação em um instante

Os dados de georreferenciamento do Anglova foram disponibilizados através de um arquivo de texto estruturado, neste arquivo é possível observar dados de latitude e longitude para cada nó da simulação ao longo de toda a operação. Cada nó possui um total de 7800 pontos que representam o seu movimento ao longo de todo o desdobramento no terreno. A companhia analisada por este trabalho contém os nós identificados pelos números entre 49 e 72.

Para representar a movimentação da tropa, optou-se por utilizar a movimentação em linha que foi desenvolvida neste trabalho. A ideia foi escolher um ponto dentro do universo de opções da companhia e descobrir os dados de direção, velocidade e intervalo de tempo da movimentação. De posse de tais dados é possível representar a movimentação

do nó analisado ao longo do tempo. O nó escolhido foi definido como o líder do grupo no qual ele está inserido, desta forma, os demais nós irão segui-lo ao longo do tempo.

A análise dos dados de georreferenciamento foi feita através de um *script* que visa transformar dados de latitude e longitude em passos de movimentação do grupo. O nó escolhido para análise foi o que possui número de identificação igual a 50. Após escolhido o nó, os seus dados de latitude e longitude foram lidos do arquivo e armazenados em um vetor de tuplas para serem utilizados a posteriori.

Visando obter uma representação visual do caminho percorrido pelo nó em questão, os dados de latitude e longitude foram manipulados algebricamente para que estivessem no intervalo de $[-1, 1]$ e, por fim, criou-se o gráfico apresentado na Figura 33. O Código 5.5 apresenta a manipulação algébrica utilizada juntamente com o método de criação do gráfico.

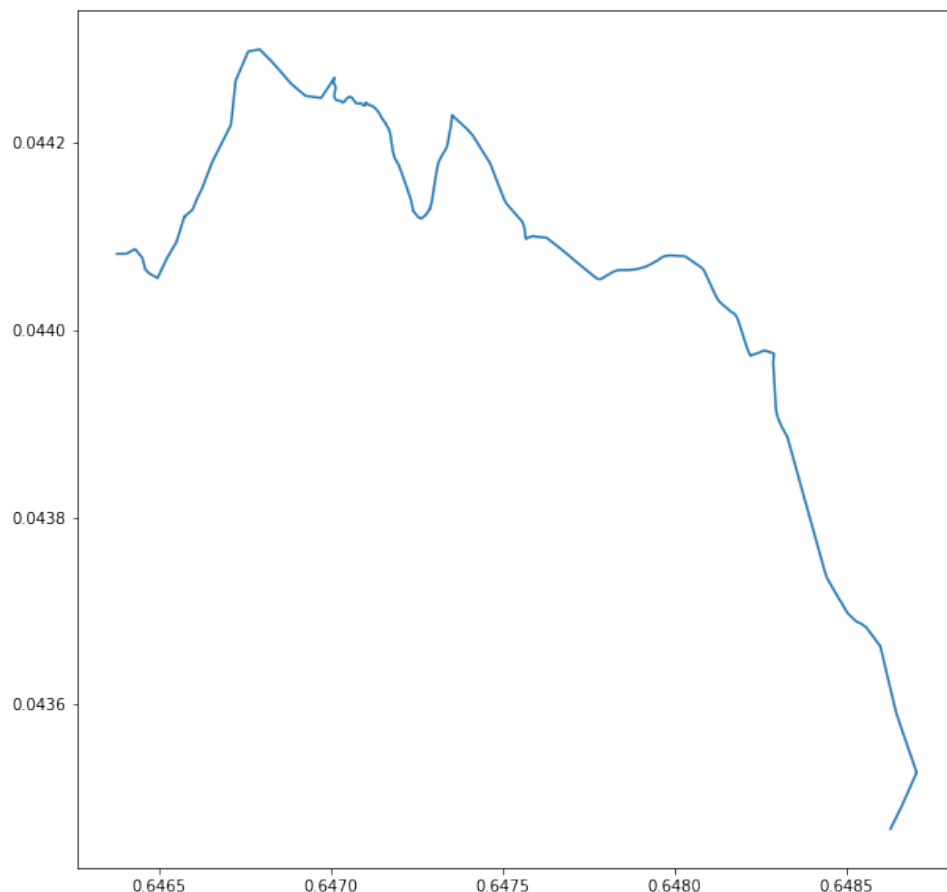


Figura 33 – Representação gráfica das 7800 coordenadas que representam o movimento do nó.

```
1 plt.figure(figsize=(10,10))
2 plt.plot(*zip(*[(lat / 90, lng / 180) for (lat, lng) in latlong]))
3 plt.show()
```

Código 5.5 – Código para tratamento e desenho de coordenadas.

$$\theta = \arctan \frac{\sin \Delta\lambda \cos \phi_2}{\cos \phi_1 \sin \phi_2 - \sin \phi_1 \cos \phi_2 \cos \Delta\lambda}$$

```

1 def getDistance(ll_ori, ll_dst, precision=4):
2     phi1 = ll_ori[0] * np.pi / 180
3     phi2 = ll_dst[0] * np.pi / 180
4
5     lambda1 = ll_ori[1] * np.pi / 180
6     lambda2 = ll_dst[1] * np.pi / 180
7
8     d_phi = phi2 - phi1
9     d_lambda = lambda2 - lambda1
10
11     alpha = np.sin(d_phi/2)**2 +
12             np.cos(phi1)*np.cos(phi2)*np.sin(d_lambda/2)**2
13
14     distance = 2 * 6371 * np.arctan2(np.sqrt(alpha), np.sqrt(1 - alpha))
15     return np.round(distance * 1000, precision)
16
17 def getAzimuth(ll_ori, ll_dst, precision=4):
18     phi1 = ll_ori[0] * np.pi / 180
19     phi2 = ll_dst[0] * np.pi / 180
20
21     lambda1 = ll_ori[1] * np.pi / 180
22     lambda2 = ll_dst[1] * np.pi / 180
23
24     d_phi = phi2 - phi1
25     d_lambda = lambda2 - lambda1
26
27     tetha = np.arctan2(np.sin(d_lambda)*np.cos(phi2),
28                        np.cos(phi1)*np.sin(phi2)-np.sin(phi1)*np.cos(phi2)*np.cos(d_lambda))
29     return np.round(tetha, precision)

```

Código 5.6 – Funções para o cálculo de distância e azimuth entre duas coordenadas.

De posse de funções para fazer o cálculo da distância e do azimuth a partir das coordenadas é possível escrever os passos de movimentação que irão representar o movimento do nó. Visando tornar a simulação um pouco mais rápida, decidiu-se utilizar um fator (`factor`) que divide os intervalos de execução de cada passo da movimentação. Tal fator acaba influenciando também na velocidade de movimentação do nó naquele passo.

Por fim, adicionou-se uma distância mínima entre nós para representar o espaçamento entre os elementos na movimentação. O código resultante de todas essas considerações pode ser observado abaixo:

```
1 factor = 5
2 resumed_dist = 78
3
4 min_dist = 50
5 last_time = 0
6
7 for i in range(len(resumed_latlong)):
8     if i == 0:
9         continue
10    d = getDistance(resumed_latlong[i-1], resumed_latlong[i])
11    tetha = getAzimuth(resumed_latlong[i-1], resumed_latlong[i])
12    print(f"group_one.addStep('line_formation', [{(i-1)*resumed_dist/factor},
        {(i)*resumed_dist/factor}], [{'angle':{tetha},
        'velocity':{d/(resumed_dist/factor)}, 'min_distance': {min_dist}}])")
```

Código 5.7 – Código para criação de passos de movimentação.

Por fim, a saída do *script* foi adicionada ao arquivo `test_anglova.py` para uma eventual execução dessa movimentação. A execução desse teste pode ser observada em um vídeo disponibilizado no repositório deste trabalho (13).

6 CONCLUSÕES E TRABALHOS FUTUROS

Esse trabalho foi idealizado com o objetivo de criar uma extensão dos padrões de mobilidade do emulador Mininet-WiFi, de forma que as emulações realizadas no *software* ficassem mais próximas da realidade encontrada em cenários de emprego militar.

A extensão proposta neste trabalho visou possibilitar, de uma forma fácil para os usuários da aplicação base, a criação de cenários que envolvam a movimentação de elementos da simulação seguindo certos padrões pré-determinados. Além disso, com o objetivo de tornar a aplicação o mais amigável e flexível possível, uma característica adicional que esta solução buscou possibilitar foi a criação de movimentos mais complexos a partir da composição de modelos mais simples. Para fazer com que isso fosse possível, foram desenvolvidos diversos modelos de mobilidade, os quais podem ser classificados como modelos com dependência espacial, dentro da classificação apresentada na Figura 10.

Ao final desse trabalho foi possível verificar o cumprimento dos requisitos definidos no início do projeto, ou seja, o Mininet-Wifi foi expandido, mantendo todas suas funcionalidades originais intactas, como pode ser visto nos esquemas apresentados na seção 3.4. Além disso, criou-se o conceito de grupos de nós dentro do emulador, o que possibilitou novas funcionalidades como por exemplo a aplicação de diferentes modelos de mobilidade a grupos distintos de nós, a execução de modelos de mobilidade em intervalos de tempo bem definidos e a composição de modelos de mobilidade.

Além das modificações estruturais supracitadas, este trabalho implementou com sucesso três novos modelos de mobilidade com dependência espacial. Os modelos implementados foram a formação em linha, a formação em cunha e por fim a formação circular. Tais modelos foram desenvolvidos de forma a possibilitar transições suaves entre eles, isso foi feito alinhado com o objetivo de compor diferentes modelos para a criação de movimentações complexas. O desenvolvimento dos modelos contemplou ainda características periféricas como a distância mínima entre nós e a aplicação do conceito de velocidade de grupo, ainda que considerando a velocidade máxima dos nós.

Após o desenvolvimento da solução, foram executados diversos testes para verificar se o projeto estava dentro do esperado. Um dos testes realizados envolveu a representação de uma operação militar real, tal simulação foi feita com base no cenário Anglova (1). A partir de dados de latitude e longitude dos elementos da operação, foi possível fazer a representação da movimentação de uma companhia utilizando-se do Mininet-Wifi. Esse teste foi importante para ilustrar como as expansões desenvolvidas neste trabalho podem ser utilizadas para estudar cenários reais.

Por fim, é importante salientar que apesar de este trabalho ter produzido algumas

melhorias na movimentação da simulação, ainda existem muitos pontos que podem ser desenvolvidos e aperfeiçoados. Uma melhoria de fácil identificação é o desenvolvimento de outros modelos de mobilidade, os quais tratem de outros cenários além do contexto da cavalaria mecanizada, o qual foi o centro de estudo neste trabalho. Além disso, é possível refinar os modelos existentes para que eles fiquem mais próximos da realidade, isso pode ser feito através de funcionalidades como a introdução de certa aleatoriedade na movimentação dos nós e a implementação de mecanismos para evitar colisões de elementos durante a movimentação. Um terceiro ponto de melhoria seria considerar o terreno onde os nós estão se movendo, levar em conta a velocidade de movimentação em diferentes terrenos e permitir que os nós se movimentem através de caminhos bem definidos na simulação.

REFERÊNCIAS

- 1 PEUHKURI, M. *Anglova Scenario*. 2022. Portal Anglova. Disponível em: <<https://anglova.net/>>. Acesso em: 08 mai 2022.
- 2 FONTES, R.; ROTHENBERG, C. *Emulando Redes sem Fio com Mininet-WiFi*. 1. ed. [S.l.: s.n.], 2019.
- 3 BAI, F.; HELMY, A. G. A survey of mobility models in wireless adhoc networks. In: . [S.l.: s.n.], 2004.
- 4 Brasil. Exército. Comando de Operações Terrestres. *Regimento de Cavalaria Mecanizado (EB70-MC-10.354)*. Brasília, Brasil, 2020.
- 5 SURI, N.; HANSSON, A.; NILSSON, J.; LUBKOWSKI, P.; MARCUS, K.; HAUGE, M.; LEE, K.; BUCHIN, B.; MİSİRHOĞLU, L.; PEUHKURI, M. A realistic military scenario and emulation environment for experimenting with tactical communications and heterogeneous networks. In: *2016 International Conference on Military Communications and Information Systems (ICMCIS)*. [S.l.: s.n.], 2016. p. 1–8.
- 6 KUROSE, J. F.; ROSS, K. W. *Computer networking*. 6. ed. Upper Saddle River, NJ: Pearson, 2012.
- 7 KARYGIANNIS, T.; OWENS, L. *Wireless Network Security 802.11, Bluetooth and Handheld Devices*. Washington, D.C., 2002.
- 8 CÓRDOVA, R. T. *SDN - DMM: Redes definidas por software para gerenciamento de mobilidade distribuído em redes IP móveis heterogêneas*. 167 p. Mestrado em Engenharia Elétrica — Universidade de Brasília, 2016. 08 mai. de 2022.
- 9 CAMILO, M. J.; MOURA, D. F. C.; SALLES, R. M. Redes de comunicações militares: desafios tecnológicos e propostas para atendimento dos requisitos operacionais do exército brasileiro. *Revista Militar de Ciência e Tecnologia*, v. 37, n. 3, p. 5–25, 2020. 08 mai. de 2022. Disponível em: <<http://www.ebrevistas.eb.mil.br/CT/article/view/6910>>.
- 10 XIA, W.; WEN, Y.; FOH, C. H.; NIYATO, D.; XIE, H. A survey on software-defined networking. *IEEE Communications Surveys Tutorials*, v. 17, n. 1, p. 27–51, 2015. 08 mai. de 2022. Disponível em: <<https://ieeexplore.ieee.org/abstract/document/6834762>>.
- 11 DEMORI, A. M.; SILVA, R. R. M. R. da; ALBUQUERQUE, P. P. L. de; MOURA, D. F. C.; TESOLIN, J. C. C.; CAVALCANTI, M. C. R. Minimanager: Uma proposta de plataforma web para emulação de redes heterogêneas de comunicação móvel. In: . [S.l.: s.n.], 2022.
- 12 Brasil. Exército. Estado-Maior. *Movimento e manobra (EB20-MC-10.203)*. Brasília, Brasil, 2015.
- 13 PONTES, D.; CHAVES, C. *Repositório de código do projeto Mininet-Wifi-EBx*. 2022. Disponível em: <<https://gitlab.com/nrg-ime/mn-wifi-ebx>>. Acesso em: 30 set 2022.