Thiago de Souza Lamenza

MAVSIMNET

Um framework para simulação e controle de Redes Móveis e Sensores

RELATÓRIO DE PROJETO FINAL

DEPARTAMENTO DE INFORMÁTICA

Programa de graduação em Ciência da Computação

Rio de Janeiro
Junho de 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Thiago de Souza Lamenza

MAVSIMNET

Um framework para simulação e controle de Redes Móveis e Sensores

Relatório de Projeto Final

Relatório de Projeto Final, apresentado ao programa de Ciência da Computaçãoda PUC-Rio como requisito parcial para a obtenção do titulo de Bacharel em Ciência da Computação.

Orientador: Prof. Markus Endler

Rio de Janeiro
Junho de 2022

Resumo

Lamenza, Thiago de Souza; Endler, Markus. **MAVSIMNET**. Rio de Janeiro, 2022. 32p. Projeto de Graduação – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

O framework MAVSIMNET é um projeto destinado a provisionar para o simulador de redes OMNeT++ a capacidade de simular movimentação de veículos autônomos de maneira fiel. O projeto implementa uma conexão entre esse simulador de rede e um simulador de veículos provido pelo ambiente Ardupilot, o Software In The Loop (SITL). A comunicação entre os dois simuladores é feita por TCP e as mensagens trocadas seguem um protocolo de mensagem chamado MAVLink, que é o protocolo padrão usado no ambiente Ardupilot para comunicação entre estações de controle e veículos. O projeto é distribuído como uma biblioteca de módulos no ambiente OMNeT++, onde pode ser utilizada como uma parte funcional de uma simulação ou estendida para implementar comportamentos novos.

Palavras-chave

simulador, rede, OMNeT++, INET, Ardupilot, SITL

Sumário

1
Introdução

A evolução das tecnologias de controle automatizado de veículos móveis faz crescer o número de aplicações viáveis para esse tipo de tecnologia no mundo real (1) (2). Uma dessas aplicações é a população de redes ad hoc, redes descentralizadas e temporárias que não dependem de infraestrutura previamente instalada. Nessas redes, esses veículos se comportam como nós móveis e podem atuar como coletores, transmissores ou provedores de dados.

Apesar da sua versatilidade e crescente número de aplicações reais, a implantação dessas redes traz consigo diversos desafios como de roteamento, economia geral de energia em seus nós, resiliência a falhas, dificuldades regulatórias, entre outros (3). Os desafios apresentados tornam necessária uma intensa validação das estratégias e componentes empregados no decorrer do desenvolvimento desses sistemas conectados visto que é essencial a garantia de que sua atuação atenda a todas suas expectativas. Outro fator a se considerar é que esse é um campo novo e em constante desenvolvimento. Portanto, não conta com décadas de testes em campo que validem a tecnologia, gerando uma grande demanda por ambientes que tornem esses testes possíveis.

Mais um fator é a pluralidade de condições variáveis às quais a rede pode ser exposta durante seu funcionamento. Como a maioria das aplicações se situa em ambientes não controlados e com pouca intervenção humana, os nós da rede podem estar sujeitos a condições meteorológicas e ambientais altamente mutáveis, fato que pode acarretar interferências de comunicação assim como falhas nos seus diversos componentes físicos. Fica claro que as redes devem ser resistentes a falhas que podem ser causadas pelos mencionados fatores.

Todas as questões mencionadas trazem a necessidade da criação de um ambiente controlado onde diversas condições de rede possam ser simuladas a fim de testar o funcionamento da rede em situações distintas. Esse ambiente deve permitir o teste rápido tanto de componentes físicos, tais como drones e sensores, como de protocolos de comunicação e controle, às redes móveis. O ambiente de simulação e execução também deve ter um bom custo-benefício, permitindo a geração de resultados sem grandes custos monetários ou de tempo.

Uma solução clara para atender a esses requisitos é o uso de ambientes

de simulação capazes de reproduzir de maneira fiel às condições físicas que se apresentam no campo. Para essa aplicação a simulação deve ser capaz de representar tanto características de rede, como a transmissão de mensagens seguindo o modelo multicamada usado em aplicações reais, como características físicas, tendo um modelo de mobilidade de nós móveis, capaz de representar como esses nós se comportariam no mundo real (1) (2) (3).

Para simulação de redes, uma solução possível são os simuladores de eventos discretos, como o simulador OMNeT++ (4), capazes de modelar a interação entre nós por meio da simulação de todas as camadas de comunicação em rede e para vários padrões e protocolos de rede. Já para a simulação do comportamento físico dos nós, simuladores como o Software In The Loop (6), provido pelo ambiente de controle de veículos autônomos Ardupilot (5), providenciam modelos físicos realistas para a simulação da movimentação seja ela terrestre, aérea ou submarina, e de condições físicas encontradas por nós móveis.

Existe uma carência de frameworks de simulação capazes de proporcionar um ambiente que simule as áreas de transmissão de rede e de mobilidade bem e de maneira conjunta (7). A proposta desse projeto é fazer uma conexão com os simuladores OMNeT++ e o simulador Software In The Loop, do Ardupilot, implementada em C++. Essa conexão irá permitir a criação de um ambiente de simulação capaz de atender tanto a eventos de rede como aos aspectos físicos dos nós móveis.

2
Situação Atual

A motivação inicial por trás desse projeto foi o projeto GrADyS (8), que busca investigar e desenvolver redes de veículos aéreos autônomos para monitorar grandes áreas populadas por sensores com objetivo de aplicar essa tecnologia em usos reais como monitoramento de áreas perigosas ou de difícil acesso, por exemplo.

Uma das vertentes desse projeto é a de simulação e tem o objetivo de desenvolver uma arquitetura capaz de representar a realidade com uma fidelidade suficientemente grande para servir como base para o teste de protocolos de comunicação e controle em redes móveis (MANET – Mobile Ad hoc Networks), de modo a otimizar a coleta de dados em campo. O software usado para esse fim é o simulador de eventos discretos OMNeT++, simulador focado na modelagem de redes, munido também de diversos modelos fornecidos pela biblioteca de modelos INET para auxiliar na implementação da rede.

No projeto GrADyS, para representar a movimentação dos nós móveis, foi adaptada uma solução, fornecida pelo framework INET, que é um modelo de mobilidade simples de veículos terrestres. Modelos de mobilidade são usados no INET para fornecer características de movimentação a um nó de rede. Existem diferentes modelos para diferentes situações, como modelos de movimentação em pontos aleatórios, outros que seguem padrões fixos de movimentação e diversos outros. A essa solução foram adicionados comportamentos aéreos, efetivamente permitindo a modelagem de movimentação usando três dimensões, incluindo altitude.

Também foi criada uma estratégia de modularização que separa os componentes da simulação em três conjuntos de módulos: comunicação, protocolo e mobilidade. Essas três vertentes trabalham em conjunto para modelar o comportamento dos diversos nós da rede, principalmente sensores e drones.

Essa implementação não é capaz de suprir a carência de ferramentas aptas a representar característica de rede e mobilidade de forma adequada. O modelo de mobilidade adaptado foi implementado para seguir um arquivo de pontos similar ao utilizado no ambiente Ardupilot, porém não faz uma movimentação realista considerando características físicas de um veículo real e apenas segue esses pontos como se o veículo estivesse em trilhos, sempre se movendo em

linha reta para seu destino.

Projeto	MAVSIMNET	FlyNetSim	AVENS
Controle integrado com a simulação	Sim	Não	Não
Número máximo de nós	25*	70**	20***
Simulador de mobilidade	SITL (Ardupilot)	SITL (Ardupilot)	X-Plane
Simulador de rede	OMNeT++	ns-3	OMNeT++
Vários tipos de veículos	Sim	Não	Não
Licenciamento	Open Source	Open Source	Open Source
Última atualização	Jun/2022	Jun/2020	Dez/2016
Linguagem de programação	C++	Python	C++

Tabela 2.1: Comparação de projetos semelhantes
* Não se tratam de limitações de arquitetura e sim limitações no hardware em que os testes foram feitos.
** Limite informado pelos autores, também são limitações de hardware.
*** Limite estrito de arquitetura.

Como mencionado, isso não é uma demanda exclusiva do projeto GrADyS e sim uma carência geral de simuladores desse tipo, sendo a conexão de dois simuladores uma ideia já testada para supri-la (9) (10). A tabela 2.1 compara algumas ferramentas que seguem esse princípio.

O simulador AVENS (9), por exemplo, busca supri-la pela conexão do OMNeT++ com o simulador de aeronaves X-Plane (11). A implementação do AVENS separa completamente as responsabilidades do simulador de redes e do simulador de voo. O controle dos veículos é responsabilidade do simulador de voo enquanto o simulador de redes é apenas informado da sua posição no espaço. O problema dessa implementação é que ela introduz uma limitação, as mensagens trocadas entre os nós móveis não podem influenciar a sua movimentação, já que a movimentação é exclusivamente controlada pelo simulador de voo. Além disso, a escolha do simulador X-Plane restringe essa solução para veículos aéreos e por seu enfoque em uma simulação visual mais intensa em recursos também limita o número de nós móveis.

O FlyNetSim (10), que utiliza o ns-3 como simulador de rede e o próprio ArduPilot como simulador de voo, escolhe uma estratégia semelhante a esse projeto, porém com um enfoque diferente. Esse simulador tem foco na comunicação entre a *Ground Control Station* (GCS), entidade controladora dos veículos, e UAVs. A comunicação entre essas duas entidades ocorre no

simulador de rede. Esse projeto implementa uma interface gráfica simples que serve de GCS, os comandos por ela gerados são enviados ao simulador NS-3, responsável por simular a arquitetura de rede pelo qual esse comando vai trafegar até um nó simulado e em sequência será transmitido ao simulador Ardupilot. O caminho contrário é tomado por telemetria saindo dos UAVs em direção à GCS. Essa implementação permite que o FlyNetSim simule o efeito que diferentes situações de rede e arquiteturas tem na performance da comunicação GCS-UAV. Os nós simulados, que representam os UAVs, são capazes de se comportar como qualquer outro nó em um simulador de rede, permitindo também simulá-los como parte de uma rede maior.

O funcionamento do FlyNetSim difere do MAVSIMNET em alguns pontos. Enquanto no primeiro ela é uma entidade externa à simulação de rede, responsável por enviar comandos e receber telemetria, no segundo ela reside dentro do simulador OMNeT++. O modo pelo qual essa entidade se comporta não é uma restrição da arquitetura do MAVSIMNET, como o enfoque desse trabalho foi o estudo do comportamento autônomo dos veículos, as funcionalidades geralmente empenhadas por uma GCS foram implementadas internamente em cada nó móvel, de lá são gerados comandos para o simulador de mobilidade segundo a situação desses nós na simulação de rede. A vantagem da GCS ser um artefato que reside na simulação de rede é a flexibilidade que essa arquitetura permite. Qualquer arquitetura de simulação que o OMNeT++ suporta pode ser utilizada. A GCS poderia ser inclusive um nó único que se conecta aos veículos móveis utilizando alguma infraestrutura de rede, como Wi-Fi, e transmite comandos para eles, exemplo que se assemelha bem à proposta do FlyNetSim. Cabe ao desenvolvedor da simulação utilizar a arquitetura disponibilizada pelo projeto para implementar esse novo comportamento.

3
Objetivos

O objetivo do projeto MAVSIMNET é permitir uma integração de simuladores capazes de reproduzir, de forma realista, os aspectos de comunicação sem fio e de movimentação física envolvendo diversos nós móveis e sensores. Esse objetivo será atingido com a criação de uma conexão entre o simulador OMNeT++, especializado na simulação de redes, e o sistema de controle remoto de veículos móveis Ardupilot, equipado com o ambiente de simulação de veículos móveis Software In The Loop. Os simuladores, ambos Open Source, serão utilizados de maneira a aproveitar o melhor de suas capacidades. O OMNeT++ simulará a comunicação entre os nós da rede e implementará os protocolos de interação dos nós e o simulador Software In The Loop ficará encarregado de suprir essa simulação com modelos de mobilidade realistas.

O simulador Software In The Loop permite a simulação de veículos móveis autônomos dos mais variados, incluindo drones, aviões, veículos terrestres e submarinos. Ele segue uma arquitetura que permite a sua conexão com outros softwares através de conexões TCP ou UDP.

Um software chamado MAVProxy (12), utilizando essa conexão, fornece uma interface de linha de comando para controle de veículos autônomos, simulados ou não. O MAVProxy pode ser configurado para mandar informações de telemetria e receber comandos em tempo real por mensagens TCP ou UDP. Inicialmente essa ferramenta ia ser utilizada para oferecer a ponte entre os dois simuladores, permitindo ao OMNeT++ estar ciente da localização e estado do veículo, assim como enviar comandos baseados nas situações de redes que ele mesmo simula. Essa estratégia foi reavaliada por motivos que serão explicados a seguir, e ela foi utilizada somente no protótipo do projeto.

Do lado do OMNeT++, deverá ser criada uma estrutura que permita gerir a recepção de telemetria e o envio dos comandos para o SITL. Essa estrutura deverá suportar o protocolo MAVLink (13), protocolo de comunicação suportado pelo SITL, sendo capaz de ler e gerar mensagens nesse formato. A natureza modular do OMNeT++ permite que essa interação aconteça de forma encapsulada e que não influencie a implementação do resto da estrutura de redes.

A interação entre essa arquitetura de mobilidade e o resto da simulação

deverá então acontecer de maneira análoga a como já ocorre na arquitetura atual usada no framework INET. Todos os módulos de mobilidade desse framework por extensão seguem uma interface base comum que os distingue dos outros módulos que compõe um nó simulado. O resultado é uma arquitetura onde cada nó simulado é construído pela composição de módulos com funções bem definidas. A estrutura modular do OMNeT++ facilita esse tipo de encapsulamento, limitando a interação entre os módulos para mensagens e evitando a criação de dependências de natureza de implementação, se a interface de mobilidade reagir e produzir as mesmas mensagens esperadas dos módulos de mobilidade do INET, essa integração já estará garantida.

A interface de comunicação com o simulador Software In The Loop deve ser implementada de maneira que seja genérica o bastante para ser utilizado em qualquer projeto. Um exemplo funcional dessa característica de sua utilização seria o acoplamento com o atual modelo de mobilidade utilizado no projeto GrADyS, que segue uma sequência de pontos no ar sem se prestar a representar um movimento realístico. Esse módulo poderia recorrer à interface oferecida com funções que permitem a interação com o simulador SITL e recebimento de telemetria (localização do veículo, velocidade, etc.). Dessa forma o comportamento do módulo de seguir um arquivo de pontos seria mantido, porém, agora munido dessas funções auxiliares e uma conexão com o SITL, responsável por simular a movimentação realística do veículo.

Da mesma forma que tal acoplamento poderia ser utilizado para adicionar um modelo de voo realista ao módulo do projeto GrADyS ele também poderia ser utilizado para adicionar um modelo de mobilidade para qualquer outro projeto. Cabe ao desenvolvedor manejar a ferramenta e implementar módulos capazes de gerar comandos ao módulo que implementa a interface, podendo se aproveitar das atualizações de telemetria para simular e reagir a situações de rede. A parte do modelo de voo, para qual existem poucas soluções satisfatórias no ambiente OMNeT++, seria simulada pela conexão proposta nesse projeto.

Além da garantia de generalidade também devem existir garantias técnicas para permitir que o simulador seja utilizado amplamente em simulações no ambiente OMNeT++. A garantia de um bom comportamento como interface, ou seja, deve garantir resultados consistentes para as diferentes interações disponíveis e providenciar um número adequado de funções para permitir a implementação de comportamentos complexos de maneira cômoda. A comunicação definida por um conjunto de mensagens de uso genérico, mínimo para evitar complexidade excessiva na interação, mas grande o bastante para evitar que o desenvolvimento usando a interface seja oneroso, deixa o software acessível em um nível de implementação, mas existem empecilhos de execução que podem

tornar seu uso inviável.

A primeira grande garantia é a garantia de sincronia. Para ser utilizada, a aplicação deve ser capaz de não somente simular o veículo em tempo real como deve também conseguir responder a comandos e manter a simulação atualizada do estado desse veículo, através de mensagens de telemetria. O impacto da falta dessa garantia pode ser constatado no já comentado simulador AVENS que ao não garantir a qualidade de sincronização bi-direcional entre os simuladores limita seus papeis na simulação, não permitindo o nível de interação em tempo real que é buscado pelo MAVSIMNET. Tal desafio deve ser superado com um foco especial para a performance do software produzido e o uso dos recursos oferecidos pelos frameworks para facilitar a comunicação em tempo real.

Outra capacidade que a aplicação deve ter é a capacidade de simular um grande número de veículos móveis. É uma proposta do projeto a simulação de redes populadas por nós móveis e isso não pode acontecer se o número de nós simulados for excessivamente restrito. Isso impõe desafios tanto de performance quanto de arquitetura. O SITL e o OMNeT++ são capazes de simular vários nós simultaneamente. O que parece ser o ponto limitante é a performance que o OMNeT++ terá ao simular tantos nós enquanto se preocupa em manter suas posições atualizadas conforme telemetria chega do simulador SITL, assim como a demanda de processamento gerada pela execução de diversas instâncias do SITL. Ele consegue simular diversos nós de rede simultaneamente, permitindo inclusive simulações em larga escala usando multi-threading e também possui um ambiente de simulação em linha de comando que diminui o custo computacional de representar a simulação visualmente. Essas capacidades deverão ser usadas para contornar qualquer problema de performance.

4
Atividades Preparatórias

4.1
Estudos Conceituais

O estudo das ferramentas a serem utilizadas na aplicação assim como a análise de aplicações já implementadas as utilizando foi importante para fundamentar as decisões a serem tomadas no desenvolvimento. As principais ferramentas estudadas foram o software MAVProxy, o protocolo de comunicação MAVLink, o framework de simulações OMNeT++ e sua biblioteca de componentes INET.

O estudo começou com a instalação de todas as ferramentas. O MAVProxy segue um processo de instalação em um ambiente que permite a execução de código voltado para sistemas Unix no sistema operacional Windows. O processo é simples e a ferramenta MAVProxy instalada disponibiliza uma interface de linha de comando para a comunicação com os veículos simulados. Essa interface foi aproveitada para exercitar as mensagens usadas e o comportamento dos veículos em relação a elas.

O protocolo MAVLink possui uma ferramenta de geração de código capaz de gerar bibliotecas que implementem as mensagens em diversas linguagens. Foi seguido o processo de geração para a linguagem C++, gerando um conjunto de arquivos header que foram analisados e experimentados. O uso do protocolo consiste no envio de mensagens codificadas que podem ser criadas e lidas por essa biblioteca gerada.

O ambiente OMNeT++ e o pacote de componentes INET (13) já é conhecido pelo seu uso no projeto GrADyS. Diversos componentes para várias finalidades são fornecidos pelo INET. Uma dessas funcionalidades estudadas foi a emulação de rede. Essa funcionalidade permite a inclusão de nós reais em uma rede simulada ou de nós simulados em uma rede real. Essa descrição parecia útil para o que o projeto se propunha a fazer, destacando a conexão entre os dois simuladores e a inclusão de um veículo externo, simulado pelo SITL, na simulação de rede. Entretanto, o foco dessa funcionalidade é a implementação de interações de rede e não de mobilidade, razão pela qual os módulos oferecidos para implementá-la são feitos para externalizar ou internalizar, na

simulação, conexões relacionadas às diversas camadas de comunicação de rede. Sendo assim seria um desafio adaptar esses módulos para uma comunicação relacionada a simulação de mobilidade.

Fazendo-se um estudo maior de como esses módulos foram implementados foi revelada uma questão. A inclusão do mundo real na simulação gera um requisito a mais nela, a fidelidade com o tempo real. O tempo de simulação do OMNeT++ não necessariamente tem relação direta com a passagem natural do tempo. O framework funciona baseado no agendamento de mensagens que disparam eventos nos diferentes módulos da simulação. Esse agendamento pode ser acelerado ou desacelerado dependendo das demandas de performance da simulação e também das configurações da velocidade da simulação.

A superação desse desafio de transformar esse agendamento em algo fiel à passagem natural do tempo foi feito pelo pacote INET com a criação de um agendador de tempo real, criado justamente para ser usado em conjunto com a funcionalidade de emulação de rede. Em conclusão, apesar de a funcionalidade de emulação de rede não ser ter sido criada para esse fim, o agendador de tempo real se apresenta como uma solução forte para o problema de sincronização com o mundo real na simulação.

Muito do estudo foi voltado também para aplicações já existentes. O problema anteriormente mencionado da sincronização é também mencionado no artigo que descreve o projeto FlyNetSim. Nesse projeto o simulador de redes utilizado foi o ns-3, que apresenta a mesma característica do OMNeT++ de possuir um agendador de eventos sem garantia de fidelidade em tempo real. A solução empregada por eles também foi o uso de um agendador em tempo real. O fato de o problema já ter sido estudado e solucionado por outro grupo dá confiança que as escolhas e planejamentos sendo feitos para a solução dos mesmos problemas nesse projeto vão dar bons resultados.

Para auxiliar no entendimento do protocolo MAVLink foram estudados softwares que implementavam o controle de veículos usando esse protocolo (15) (16). Esse estudo foi extremamente valioso, pois permitiu a familiarização com as categorias de mensagens trocadas com os veículos simulados. O protocolo MAVLink possui diversos dialetos e cada dialeto possui um grande número de especificações de mensagens. Apesar de existir uma documentação desses dialetos não fica claro que mensagem deve ser usada em que momento, para esse tipo de dúvida a análise de códigos e ferramentas já existentes foi essencial.

4.2 Protótipos

A fim de experimentar os métodos de conexão com o MAVProxy e exercitar o conhecimento adquirido sobre essa ferramenta e sobre o protocolo MAVLink, foi criado um protótipo funcional (17).

Esse protótipo foi feito na linguagem C++, linguagem que será utilizada na aplicação final no framework OMNeT++, implementa a comunicação com o OMNeT++ utilizando um socket. A fim de facilitar e acelerar a implementação também foi usada a biblioteca C++ Boost Asio. O Boost (18) é um conjunto de bibliotecas que busca estender as funcionalidades da biblioteca padrão, esse pacote de bibliotecas é altamente revisado com utilização ampla no ambiente C++ e a biblioteca Asio é focada em fornecer soluções para I/O. Ela, apesar de não ser um requisito para aplicação final, foi essencial na simplificação da implementação desse pequeno protótipo. Além disso, a ferramenta de geração de código do protocolo MAVLink foi utilizada para gerar uma biblioteca C++ contendo a implementação das mensagens.

Para atender às funções propostas, esse protótipo deve, primeiramente, conectar ao MAVProxy. Para esse fim a biblioteca Boost Asio fornece classes que facilitam a criação e configuração do socket. Além disso, deve receber dados de telemetria constantemente e enviar comandos para o veículo simulado. Os comandos precisam ser validados, usando os dados de telemetria recebidos, o que implica que o recebimento de telemetria deve ser ininterrupto e não pode ser afetado pelo envio de comandos. Essa última questão foi sanada fazendo o uso de corrotinas, com ajuda da biblioteca Boost Asio.

O protótipo final é composto por um curto setup usando uma classe helper, responsável por criar o socket e fazer o controle de estruturas compartilhadas por outros componentes do programa, e duas corrotinas, uma para a telemetria e outra para os comandos.

A rotina de comandos envia uma série de comandos. Primeiramente, negocia a taxa de comunicação com o MAVProxy, coloca o veículo em modo guiado, arma o veículo e o faz decolar para 25 metros. Para validar o recebimento e execução dos comandos foram criadas funções que serão registradas como requisitos a serem cumpridos antes da execução da rotina de comandos seguir. Esses requisitos são registrados ao mandar o comando e são checados pela rotina de telemetria conforme mensagens do simulador são recebidas.

O protótipo cumpriu sua tarefa de servir como plataforma para conhecer e exercitar conceitos necessários para a aplicação final, com destaque para a comunicação usando o protocolo MAVLink, com sucesso. Ela consegue se comunicar com o veículo simulado e mandar ordens para ele com sucesso,

usando as suas respostas para garantir que os comandos estão sendo seguidos corretamente.

5
Especificação do Projeto

5.1
Arquitetura

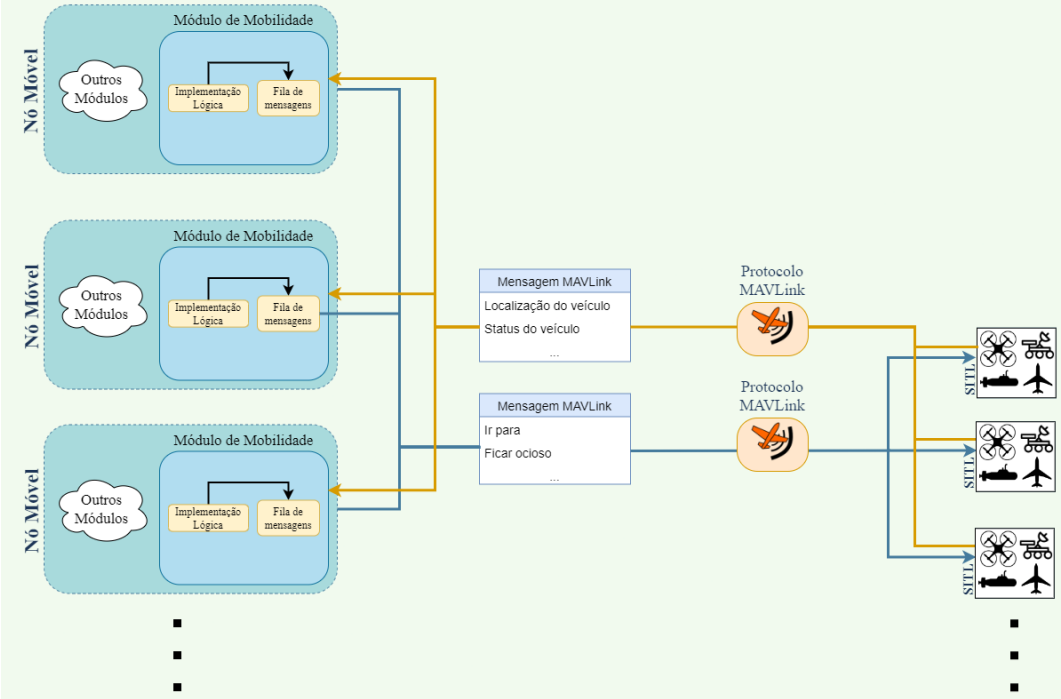

Figura 5.1: Arquitetura do projeto

Para desenvolver a arquitetura final do sistema foram analisadas e testadas todas as ferramentas descritas acima de modo a determinar que combinação delas poderia oferecer melhor performance à ferramenta. Finalmente foi utilizada uma arquitetura com módulos de mobilidade independentes, cada um gerenciando sua própria conexão com uma instância do SITL, conexões estas que são diretas e bidirecionais.

A arquitetura foi decidida na tentativa de combinar usabilidade para o usuário do framework e flexibilidade dos módulos. Para cumprir a promessa de usabilidade o módulo foi feito nos moldes da arquitetura do INET. Esse framework, além de oferecer uma grande biblioteca de módulos, estabelece uma organização que facilita a criação de simulações complexas usando esses módulos. Uma simulação INET é construída com módulos globais, responsáveis

por gerir aspectos gerais da simulação como o ambiente físico, as características do meio por onde sinais são transmitidos, entro outros. Esses módulos globais fornecem um ambiente onde residem os agentes da simulação, nós de rede.

Os nós de rede são nós simulados criados pela composição de diversos módulos filhos. Cada módulo filho representa um componente do funcionamento do seu pai. Um módulo pai contém módulos que representam as propriedades de rede dele, como o protocolo utilizado nas camadas de rede física, de enlace, transporte e todas as outras. Também possui módulos que definem características físicas como a capacidade energética, simulando uma bateria, módulos que representam aplicações de rede que utilizam a configuração de rede do módulo pai para se comunicar com outros nós e módulos que representam mobilidade, o foco desse trabalho.

Módulos de mobilidade definem como os nós de rede se movem, ou não. A movimentação ocorre em um espaço tri-dimensional e o INET é capaz de simular o efeito que ela tem na comunicação, afetando principalmente por modificar a distância entre os nós mas também pela proximidade do chão, obstrução por objetos físicos entre outros fatores. A mobilidade é um aspecto essencial em redes móveis modernas onde diversos dos nós que as populam estão em constante movimentação e auxiliar na simulação de redes populadas por veículos móveis que é o que esse trabalho visa facilitar.

O framework MAVSIMNET oferece uma série de módulos de mobilidade, no estilo do INET, que podem ser adicionados à qualquer nó móvel conferindo-lhe capacidade de mobilidade. Esses módulos não diferem em uso dos módulos de mobilidade do INET, portanto seu uso não será estranho aos seus usuários. A função desses módulos de mobilidade no framework pode ser apreciada na figura 5.1.

Outra característica da arquitetura do INET é o amplo uso de herança. É importante que os blocos usados para construir as entidades da simulação possuam interfaces consistentes dependendo da sua funcionalidade. No caso dos módulos de mobilidade, atender a essa interface significa informar a taxa de atualização da posição do veículo e modificar sua posição conforme o critério do módulo de mobilidade. Os módulos do MAVSIMNET sincronizam a taxa de atualização do módulo de mobilidade com a taxa de envio de telemetria de posição do SITL, acoplando o movimento do nó simulado no OMNeT++ à sua posição no mundo simulado do SITL.

O MAVSIMNET busca garantir extensibilidade seguindo essa mesma filosofia de herança. Um desenvolvedor deve ser capaz de implementar um módulo de mobilidade que usa a conexão com o SITL para simular movimentação realista, porém usa essa habilidade para criar um padrão de movimentação

não suportado pelos módulos já oferecidos pela biblioteca. Para atender a esse requisito o módulo foi construído em volta de uma classe base, chamada *MAVLinkMobilityBase*. Essa classe é responsável por controlar uma instância do SITL. Ela cria a instância, estabelece uma conexão TCP com ela e recebe e enviam mensagens para a instância.

Para efetivar a garantia de sincronia entre os simuladores esse módulo deve ser usado em conjunto com o módulo *RealTimeScheduler* do INET. Um módulo *scheduler* é uma classe do OMNeT++ responsável por despachar os eventos que constituem a simulação nos tempos apropriados. O *scheduler real time* faz isso para manter uma taxa de um para um entre o tempo real e o tempo da simulação. Essa estratégia garante a sincronia com o SITL que também roda em tempo real, porém traz malefícios. O maior deles é que quando, por motivos de performance, os módulos da simulação não conseguem completar seus eventos em tempos apropriados, esse módulo pode gerar erros na simulação.

Como uma salvaguarda para essa situação o módulo é opcional, ou seja, o usuário do MAVSIMNET pode decidir sacrificar a sincronicidade da simulação. Essa não é uma escolha ideal, porém é muitas vezes é necessária, principalmente em simulações complexas. O usuário ainda pode decidir rodar a simulação sem um multiplicador de velocidade, isso a manterá aproximadamente sincronizada, mas demoras de eventos e outros fatores farão com que os tempos dos dois simuladores variem.

Além de aliviar o trabalho pesado de estabelecer uma instância do SITL conectada essa classe também oferece uma série de métodos que auxiliam o trabalho de administrar a comunicação com o simulador físico. Para entender melhor como esses métodos são úteis é necessário compreender o funcionamento do protocolo MAVLink e como ele é usado para controlar veículos.

Mensagens MAVLink representam uma transação de informação entre entidades que suportam o protocolo MAVLink, sejam elas veículos, ground stations ou, no caso do projeto, um módulo no simulador OMNeT++. As mensagens possuem uma origem e um destinatário e trazem dentro de si dados que compõem o corpo da mensagem. Esses dados podem representar diversas coisas, as mais comuns são comandos, geralmente passados de uma entidade controladora para um veículo, e telemetria, informações do estado do veículo geralmente passados para uma entidade controladora.

O protocolo MAVLink suporta diferentes meios de controlar um veículo. Alguns dos meios envolvem o envio de várias etapas que o veículo deve seguir como uma missão pre-definida e outros envolvem a passagem de comandos que definem a próxima ação imediata à ser tomada pelo veículo. Em ambos os

casos alguns cuidados importantes devem ser tomados.

O primeiro é estipular uma estratégia de limite de tempo e tentativas. Se tratando de mensagens que serão passadas por uma rede é esperado que algumas mensagens sejam perdidas, ou talvez cheguem a uma entidade quando ela não está pronta para recebe-lá. Seja qual for a causa, é importante garantir que essas falhas na medida do possível não sejam catastróficas. Uma boa solução para esse problema é considerar que o envio de uma mensagem falhou após um determinado limite de tempo sem resposta do destinatário, motivando o re-envio da mensagem. Esse processo deve se repetir até um número máximo de tentativas, que se atingido deve acarretar desistência de enviar essa mensagem.

Outra estratégia importante, principalmente para entidades controladoras, é garantir que o envio de um próximo comando só seja feito quando o anterior foi concluído. Isso se aplica tanto para o meio de controle por missão, cujos passos são enviados em sequência onde o próximo passo só deve ser enviado após o destinatário reconhecer o anterior, ou controle direto, quanto é essencial garantir que ações não sejam interrompidas por desejo de começar uma próxima. Para isso foi criado um sistema de fila de mensagens e condições de prosseguimento.

Novas mensagens entram em uma fila que só anda quando a mensagem que está no seu início for enviada e obtiver o resultado desejado. Esse resultado pode ser um simples reconhecimento do recebimento da mensagem ou um comportamento mais complexo. Um comando de decolagem, por exemplo, pode estar condicionado a só ser completo quando o veículo atingir uma altitude desejada. Para decidir se essas condições de conclusão das mensagens já foram concluídas deve-se observar o fluxo de mensagens vindo do destinatário. No exemplo do comando de decolagem, deve-se observar as mensagens que contém telemetria de posição, checando se a altitude chegou no valor desejado.

A classe *MAVLinkMobilityBase* oferece métodos que implementam tanto a estratégia de tempo limite e tentativas quanto a checagem de condições de conclusão de mensagens. Ela oferece uma declaração que encapsula a mensagem MAVLink e anexa a ela algumas informações importantes. Dentre essas informações estão o tempo limite dessa mensagem, o número de tentativas e uma função que define a condição de conclusão da mensagem. Oferece também métodos para inserir essa estrutura em uma fila de mensagens. Essa fila é gerida pelo módulo, andando quando a condição de conclusão da mensagem à sua frente é completa. Quando uma mensagem chega à frente da fila ela é enviada para a instância do SITL, seguindo os critérios de tempo limite e tentativas. O tempo limite também se aplica ao tempo de demora

para a condição de conclusão da mensagem ser completa, permitindo que a mensagem seja re-enviada ou cancelada caso demore demais para ser concluída.

O objetivo dessa classe base não é implementar qualquer comportamento de mobilidade e sim oferecer uma base forte sobre a qual os módulos do framework poderão ser construídos. A ideia é aliviar o esforço de desenvolvimento de implementar a conexão e interação com o SITL e permitir assim que o desenvolvedor foque na implementação dos comportamentos que o módulo deve ter.

Como mencionado, o INET padroniza o funcionamento dos seus tipos de classe para possibilitar que o seu uso como uma peça na composição de um nó seja o mais simples possível. No caso das classes de mobilidade isso é feito estabelecendo um paradigma comum à todas essas classes. A função das classes de mobilidade é fornecer à simulação a localização atual do nó móvel, a padronização dessas classes é feita estabelecendo uma rotina comum à todas que é chamada com uma frequência constante, responsável por determinar a posição atual do veículo. O trabalho de um programador de uma classe de mobilidade clássica do INET é então atualizar a posição do veículo antes dessa rotina ser chamada, garantindo que a posição do veículo na simulação esteja sempre atualizada.

Em uma implementação clássica essa atualização se dá manipulando diretamente as coordenadas atuais do veículo para fazer com que ele siga o comportamento desejado. Um exemplo simples é o cálculo da posição atual do veículo baseado em sua posição anterior, sua velocidade, a direção do seu movimento e o tempo desde a última atualização. Com essas informações é possível atualizar a posição do veículo na simulação atualizando as coordenadas para considerar o quanto o veículo se moveu desde a última atualização. Classes de mobilidade do framework MAVSIMNET não podem seguir esse padrão de atualização direta já que a movimentação não é definida por um critério matemático e sim pelos dados retornados da instância do simulador SITL.

A impossibilidade da atualização direta gera a necessidade de um nível de indireção. O programador, ao invés de atualizar a posição do nó móvel manualmente, deve dar ordens ao simulador SITL que, é responsável por determinar a posição do veículo. Essa é a maior ruptura entre o estilo de desenvolvimento de uma classe de mobilidade INET e MAVSIMNET. Para amenizar essa diferença foram criados dois módulos, *TelemetryConditions* e *VehicleRoutines*.

O primeiro oferece um conjunto de funções que podem ser usadas como condições para conclusão de mensagens. Elas implementam condições comuns como aguardar que o veículo chegue à alguma altura ou em algum local

específico, aguardar o veículo estar pronto para ser armado, verificar se o comando foi reconhecido, entre outros. Esse módulo diminui a dificuldade de programar essas condições de conclusão, tarefa que requer conhecimento de diversos tipos de mensagem MAVLink enviados pela instância do SITL para o OMNeT++, já que essas mensagens têm as informações necessárias para confirmar a conclusão da mensagem.

O segundo oferece funções que geram listas de instruções para ordens comuns. Controlar um veículo utilizando o protocolo MAVLink envolve conhecer diversas categorias de mensagens e comandos, muitas vezes pouco documentados, usados para comandar o veículo a fazer ações comuns. Esse módulo serve para oferecer geradores prontos capazes de gerar os comandos necessários para o tipo de veículo de interesse para ações muito utilizadas como ativar o veículo, decolar para uma altitude específica, modificar o modo de controle e ordenar que ele se mova para um local de destino.

Além de ferramentas para desenvolvimento, o MAVSIMNET também oferece uma série de módulos para aqueles usuários que querem inserir um módulo pronto em suas simulações sem se preocupar em desenvolvê-los por conta própria. Sobre a classe base *MAVLinkMobilityBase* foram construídas duas classes, *MAVLinkRandomWaypointMobility* e *MAVLinkFileMobility*. Elas foram construídas para demonstrar as capacidades do framework assim como servir de exemplo de implementação para qualquer contribuidor ou usuário futuro que tenha interesse no framework e para servir como classes prontas para alguns casos de uso do framework.

A classe *MAVLinkRandomWaypointMobility* implementa um padrão de movimento semelhante à classe *RandomWaypointMobility* do framework INET. O padrão consiste em mover o nó para posições aleatórias, dentro de limites parametrizados, e fazê-lo aguardar por um tempo determinado antes de se mover para outro ponto aleatório.

A simplicidade da sua implementação demonstra a capacidade da classe base de mitigar o esforço gasto com os meandros da comunicação entre os dois simuladores. Providencia também uma classe pronta para um caso de uso comum numa simulação de rede populada por nós móveis, onde nem sempre é importante definir um padrão de comunicação complexo e um aleatório supre as necessidades do problema.

A outra classe, *MAVLinkFileMobility*, também oferece um caso de uso comum, especialmente para veículos reais que usam o protocolo MAVLink. É comum fornecer à esses veículos autônomos que seguem o protocolo um arquivo de missão que contém as etapas que devem ser seguidas para concluir um trajeto. Esse arquivo tem um formato definido e o módulo aceita esse mesmo

formato, lendo ele e informando ao veículo os passos para se completar uma missão definida pelo usuário. Isso é útil caso seja interessante uma mobilidade controlada e determinística, diferente do módulo aleatório.

5.2 Motivações

As decisões que levaram a criação da arquitetura do projeto foram motivadas por experimentação e estudo feitas em preparação para sua implementação assim como a experiência obtida durante ela. Essa seção irá detalhar esse processo, visando dar validade a arquitetura final do projeto.

Durante a fase de estudos e prototipagem foi imaginada uma arquitetura centralizada onde um módulo da simulação iria controlar a instanciação e destruição do simulador SITL assim como criar e manter a conexão entre os dois simuladores. O protótipo criado em C++ foi produtivo de modo que motivou o estudo do protocolo MAVLink e das ferramentas de execução do SITL, o MAVProxy sendo uma delas. Houve, porém, uma área que esse protótipo não cobriu que é a escalabilidade dessa estratégia para diversos veículos.

O modo de execução padrão do SITL, utilizado por usuários que buscam manualmente inicializar e controlar uma instância do simulador, inicializa em conjunto com o simulador o software MAVProxy e os conecta. O MAVProxy fornece uma interface de linha de comando que permite o envio de mensagens para o veículo simulado. Também permite a abertura de conexões UDP para o veículo que podem ser usadas por softwares de controle de voo, de observação e também foram usadas pelo protótipo de modo a estabelecer uma conexão de mão dupla com o veículo. O MAVProxy se encarrega de rotear essas conexões, que podem ser simultâneas e numerosas.

A ideia inicial para implementação do MAVSIMNET era então utilizar esse modo de inicialização padrão para facilitar a tarefa de conexão com o simulador. Invocar o SITL utilizando esse modo de execução envolve a chamada de um programa com uma série de parâmetros que por sua vez inicializava o MAVProxy e o SITL.

A primeira implementação do MAVSIMNET seguiu esses dois princípios, uma arquitetura centralizada usando o SITL acoplado ao MAVProxy. Um módulo central era responsável por disparar e se conectar com as instâncias do SITL conforme as demandas dos módulos de mobilidade. Esse módulo era apenas um intermediário entre as classes de mobilidade e as instâncias do MAVProxy, roteando as mensagens entre essas entidades. As primeiras limitações começaram a se mostrar quando foram introduzidos vários veículos

na simulação. O uso de recursos observado durante monitoramento das simulações excedia significativamente o esperado, limitando o número de veículos simulados simultaneamente.

Também foi observado um certo nível de redundância nessa arquitetura já que estava sendo invocada uma instância do MAVProxy para cada instância do SITL que enviam e recebem mensagens roteadas pelo módulo central até a classe de mobilidade destino. O MAVProxy é um software capaz de rotear mensagens entre diversas conexões UDP, não fazia sentido continuar utilizando a classe central para fazer esse trabalho de rotear mensagens entre diferentes instâncias do MAVProxy quando esse programa já foi feito com essa capacidade em mente.

A arquitetura foi repensada então para invocar somente uma instância do MAVProxy para diversos veículos. Analisando os parâmetros do modo de execução que estava sendo usado foi constatada a existência parâmetros para execução com múltiplos veículos. Experimentos mostraram, no entanto, que o MAVProxy, apesar de ser capaz de se conectar com diversas instâncias do SITL, não conseguia transmitir mensagens para várias simultaneamente. As funcionalidades de conexões simultâneas do MAVProxy foram feitas para suportar conexões de vários softwares de controle com uma singular instância do SITL, não o contrário.

Esse obstáculo fez com que a arquitetura fosse repensada para sua forma atual. O modo de inicialização do SITL seria trocado, já que ele não permitia a instanciação do SITL sem o MAVProxy de maneira simples. O SITL seria então inicializado a partir do seu firmware compilado e o MAVProxy não seria utilizado. Nesse ponto ficou claro também que a centralização do gerenciamento do SITL não fazia mais sentido, já que não existia mais uma entidade central vista no MAVProxy. Cada módulo de mobilidade ficou então responsável por invocar o SITL como um subprocesso a partir do firmware compilado.

Além de simplificar o uso do framework essa mudança também teve um grande impacto positivo na sua performance. Ao contrário do esperado, a melhora de performance não veio diretamente da eliminação do MAVProxy como uma dependência, já que o uso de recurso desse programa não era significativo, mesmo quando uma janela dele era aberta para cada veículo. A melhora veio da redução do número de mensagens sendo trocadas com o SITL. A forma que o SITL determina a frequência com que mensagens de telemetria serão passadas para suas conexões é por comandos que definem o intervalo de tempo entre as mensagens. Por padrão as mensagens de telemetria não são geradas, seu envio recorrente é iniciado apenas após o recebimento desses comandos.

O MAVProxy, por padrão, envia uma série de comandos de frequência para diversas categorias de telemetria gerando uma alta carga de mensagens sendo enviadas pelo SITL. Foi observado que uma parte significativa do uso de recursos de processamento do SITL é relacionado a frequência com que ele envia mensagens de telemetria. Após a eliminação do MAVProxy como dependência, esse uso caiu significativamente. Agora a classe de mobilidade ficou inteiramente responsável por controlar quais mensagens de telemetria são recebidas do SITL.

A classe *MAVLinkMobilityBase* comanda o envio recorrente de uma série de mensagens, todas necessárias e em intervalos sincronizados com a taxa de atualização do OMNeT++. O controle direto da instância do SITL permitiu então que o número de instâncias que podem ser utilizadas na simulação subisse vertiginosamente no mesmo hardware, foi observado um aumento de três instâncias para vinte cinco.

5.3 Uso

O MAVSIMNET foi feito para dois tipos de usuários, aqueles que tem interesse em desenvolver e estender as capacidades do framework para atender às suas necessidades e aqueles que buscam integrar um módulo pronto em sua simulação com configuração mínima.

Para o primeiro grupo de usuários o framework oferece a classe base, munindo esses usuários de métodos e uma estrutura que facilita o desenvolvimento de módulos de mobilidade. Oferece também dois módulos utilitários que focam em facilitar a interação com o veículo simulado através de mensagens MAVLink. Para interações não implementadas os usuários tem a liberdade de criar qualquer mensagem suportada pelo protocolo. A biblioteca de *headers* do MAVLink está inclusa no projeto e a documentação das mensagens pode ser encontrada no site do protocolo (19).

Já para o segundo o framework oferece dois módulos de mobilidade prontos, um de movimento aleatório e outro de movimento seguindo um arquivo. Movimentação aleatória é um caso de uso comum em simulações e o módulo de arquivo implementa a leitura de arquivos de mensagens MAVLink. Esses arquivos são extremamente flexíveis e esse é inclusive o método utilizado por diversas estações de controle reais para controlar seus veículos.

Ambos os usuários usarão o MAVSIMNET como qualquer outro pacote OMNeT++, que oferece uma IDE capaz de importar projetos e adicioná-los a um espaço de trabalho local. As únicas dependências do projeto, além do próprio OMNeT++, são o INET, framework amplamente usado no desen-

volvimento de simulações com o OMNeT++, e o firmware do Ardupilot. O processo de instalação assim como o funcionamento do projeto, suas classes e instruções sobre os parâmetros e funcionalidades dos seus módulos estão explicados na documentação oficial do projeto que está publicamente disponível assim como o seu código-fonte na plataforma Github (20).

O projeto foi testado na versão 5.6.2 do OMNeT++ e programado na plataforma Windows, mas adaptado para a plataforma Linux. A adaptação envolveu utilizar o mínimo possível de funções exclusivas de sistema operacional e, para aquelas inevitáveis, construir versões para ambos os sistemas operacionais.

A documentação do projeto, também disponível no Github, foi gerada usando a biblioteca *MkDocs* (21) capaz de transformar documentos escritos em *Markdown* em uma página web estilizada. Para extrair documentação diretamente do código-fonte da ferramente foi escrito um pequeno programa de *Python* que basicamente gera texto em *Markdown* a partir dos comentários que documentam as funções da ferramenta.

6
Implementação e avaliação

6.1
Testes e validação

Para avaliar o projeto implementado foram realizados uma série de testes de casos de uso potenciais. Esses testes se resumem na criação de simulações que refletem usos potenciais da ferramenta e a implantação da ferramenta em casos de uso reais.

Observando outras ferramentas, principalmente o próprio INET, foi identificado que elas disponibilizam com a ferramenta uma série de simulações testes, chamadas *showcases*, destinadas a oferecer ao usuário da ferramenta um exemplo de simulação para ilustrar o uso dela assim como permitir que ele valide que a instalação foi feita de maneira correta. Seguindo essa estratégia foram criadas algumas situações exemplo que sirvam esse mesmo propósito no MAVSIMNET.

Essas situações testam os dois módulos de mobilidade disponibilizados pela ferramenta, o *MAVLinkFileMobility* e o *MAVLinkRandomWaypointMobility*. Para o primeiro módulo foi disponibilizada uma simulação com arquivos de missão prontos em um cenário simples, o percorrimento de um trajeto. Para o segundo módulo foram elaborados dois cenários de simulação, um que mostra todos os tipos de veículos suportados pelo framework em ação simultaneamente, totalizando nove veículos de três tipos seguindo pontos aleatórios. Um segundo cenário de simulação foi elaborado para demonstrar um número grande de veículos inclusos na simulação. Esse segundo cenário foi usado para fazer o benchmarking da ferramenta nesse quesito.

Também foi montado um terceiro *showcase*. Essa simulação não foi focada em demonstrar nenhum dos módulos e sim em oferecer um exemplo simples da integração do framework em uma simulação de rede. Ela consiste em cinco drones responsáveis por passar dados para cinco veículos terrestres, todos seguindo pontos aleatórios. Essa simulação é mais próxima de uma típica do OMNeT++, ao contrário das anteriores que não exploravam as suas capacidades de rede.

Além dessas simulações criadas como parte do framework, também foram

feitos experimentos com casos de uso reais do projeto GrADyS. O módulo de mobilidade desse projeto foi adaptado para herdar da classe base do MAVSIMNET conferindo-lhe capacidades de movimentação utilizando o SITL. Também foi adaptado para utilizar mensagens MAVLink concretizar seu padrão de movimentação, com o auxílio das classes utilitárias do MAVSIMNET.

O módulo foi integrado com sucesso, sendo usado para rodar todas as simulações prontas feitas pelo projeto GrADyS. Isso foi feito como uma prova de conceito, demonstrando a capacidade de integração do módulo em uma simulação existente. Algumas das simulações do GrADyS são complexas e não muito performáticas, requerendo a desativação do módulo *RealTimeScheduler*, perdendo a garantia de sincronicidade. Mesmo assim, as simulações não são pesadas o bastante para gerar atrasos grandes quando executadas sem um multiplicador de velocidade, o que permitiu que o uso do módulo fosse avaliado apesar desse sacrifício.

6.2 Desafios

Durante a criação dessas simulações demonstrativas foram identificadas diferentes limitações e comportamentos indesejados da ferramenta. Isso era esperado já que parte do intuito da sua criação era servir como um ambiente de validação. Muitos desses desafios foram contornados, mas algumas dificuldades permanecem.

O primeiro grande desafio está relacionado a uma garantia citada nos objetivos do projeto, a garantia de sincronia. Para efetivar essa garantia o simulador OMNeT++ deve conseguir executar a simulação em tempo real, como o SITL faz. Isso é possível usando o agendador de eventos *real time* oferecido pelo INET. Essa classe implementa uma série de tratamentos para conseguir manter a simulação rodando em tempo real. Como ela foi criada para emulação de redes, funcionalidade do INET que permite inserir nós de rede externos à simulação dentro dela.

Essa classe atendeu aos objetivos do projeto mas também gerou uma série de desafios. Devido a interações desconhecidas com o ambiente de simulação uma série de interações cujas origens foram extremamente difíceis de rastrear aconteceram. Essas interações muitas vezes geravam erros crípticos que demandavam um grande tempo de desenvolvimento para serem tratados.

A maioria desses erros foi encontrado e consertado, porém algumas dificuldades precisaram ser contornadas com sacrifícios. Foi observado, durante tentativas de testes usando simulações mais complexas, que o módulo *RealTimeScheduler* causa erros catastróficos quando os módulos da simulação não

conseguem executar seus eventos de maneira a garantir que a simulação se mantenha em tempo real.

Como mencionado na descrição da arquitetura do projeto, essa dificuldade foi contornada tornando o módulo opcional, sacrificando a garantia de sincronia em prol de permitir a simulação de uma maior variedade de situações.

Essa dificuldade gerou a ideia de aproveitar das habilidades do SITL de modificação da velocidade de simulação enquanto ele é executado para consertar esse problema. As instâncias do SITL seriam aceleradas e desaceleradas conforme fosse necessário para garantir sincronia com o OMNeT++, contornando a necessidade de ser usar o *scheduler* de tempo real e permitindo inclusive que a simulação fosse executada com multiplicadores de velocidade. Apesar de a ideia ser interessante o tempo de implementação necessário e o quão tardia foi a sua concepção são proibitivas para sua implementação imediata no projeto.

Outro grande consumidor de horas de desenvolvimento foi a interação com os veículos usando o protocolo MAVLink. A documentação das declarações de mensagens (19) muitas vezes se provou incompleta, inconsistente e até incorreta. Existe uma grande variedade de mensagens documentadas nessas páginas, mas não fica claro quando elas devem ser usadas e quais são os requisitos para elas atenderem seus objetivos.

O *Mission Planner* (16), ferramenta pública de controle de veículos usando MAVLink, possui uma funcionalidade de inspeção de fluxo dessas mensagens. Essa funcionalidade se provou ser mais confiável do que a documentação do protocolo. Frequentemente quando havia desejo de implementar uma nova operação no MAVSIMNET ela tinha que ser primeiro replicada no *Mission Planner* e a observação das mensagens trocadas pelo software seria usada como base para implementar o comportamento desejado. Essas dificuldades motivaram a criação dos módulos utilitários do MAVSIMNET que tiram esse fardo para as ações mais comuns que usuários poderiam desejar implementar.

Muitos dos desafios acima explicados estão relacionados ao fato central de que a implementação do MAVSIMNET é por natureza pouco depurável. Ela depende de dois simuladores, o OMNeT++ e o SITL que, apesar de não serem ferramentas obscuras, contando cada uma com um grupo de usuários dedicado e atenção acadêmica e de mercado, não são ferramentas de amplo uso por terem casos de uso bem específicos. A dificuldade de encontrar bons materiais de consulta ou achar relatos das experiências de outros usuários com problemas e suas soluções também foi observada no projeto GrADyS, que só trata com o OMNeT++.

O OMNeT++ é uma ferramenta que tem mais de duas décadas de desenvolvimento, é natural que uma ferramenta com tal idade possua um

grande legado tecnológico acumulado em sua base de código. Esse legado se expressa negativamente durante o desenvolvimento na dificuldade de se encontrar documentação precisa de algumas funcionalidades e ferramentas. Por ser uma ferramenta de uso principalmente acadêmico ela não atrai uso tão amplo quanto ferramentas comerciais. A falta do uso em massa acarreta uma falta de experiências compartilhadas, ferramenta extremamente importante para passagem de conhecimento e poderosa na solução de problemas. A experiência de trabalhar com o SITL se mostrou semelhante nesse sentido.

7
Conclusão

O desenvolvimento do framework MAVSIMNET espera oferecer ferramentas para desenvolvimento de simulações de rede populadas por veículos usando tecnologias de simulação de veículos. A integração entre esses dois simuladores expande o leque de situações que ambos conseguem representar, permitindo aos seus usuários melhor reproduzir os cenários que desejam.

O que foi desenvolvido é apenas um primeiro passo nesse caminho, é um ponto de partida para quem deseja elaborar essa ideia e desenvolver seus próprios módulos usando a conexão com o MAVLink para tal. Também é um conjunto de ferramentas sólido para quem deseja apenas utilizar os módulos prontos, se atendo à personalização através de parâmetros, aumentando a flexibilidade do sistema.

Os próximos passos do projeto são o desenvolvimento de mais módulos complementando o framework com ferramentas prontas para mais casos de uso. Também é de interesse do projeto habilitá-lo a se comunicar com veículos reais. Isso parece ser uma tarefa complexa, porém essa impressão é imprecisa já que o SITL se comunica com o simulador do mesmo modo que um veículo real se comunicaria, utilizando o mesmo protocolo de mensagens, o MAVLink.

Referências bibliográficas

[1] GUILLEN-PEREZ, A.; CANO, M.-D.. **Flying ad hoc networks: A new domain for network communications**. Sensors, 18(10), 2018.

[2] SÁNCHEZ-GARCÍA, J.; GARCÍA-CAMPOS, J.; ARZAMENDIA LOPEZ, M.; GUTIÉRREZ, D.; TORAL, S. ; GREGOR, D.. **A survey on unmanned aerial and aquatic vehicle multi-hop networks: Wireless communications, evaluation tools and applications**. Computer Communications, 119, 02 2018.

[3] GUPTA, L.; JAIN, R. ; VASZKUN, G.. **Survey of important issues in uav communication networks**. IEEE Communications Surveys Tutorials, 18(2):1123–1152, 2016.

[4] **Omnet++ discrete event simulator**. `https://omnetpp.org/`. Acesso em: Setembro/2021.

[5] **Ardupilot**. `https://ardupilot.org/`. Acesso em: Setembro/2021.

[6] **SITL simulator (software in the loop)**. `https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`. Acesso em: Setembro/2021.

[7] KANG, S.; ALDWAIRI, M. ; KIM, K.-I.. **A survey on network simulators in three-dimensional wireless ad hoc and sensor networks**. International Journal of Distributed Sensor Networks, 12(9):1550147716664740, 2016.

[8] OLIVIERI, B.; PAULON, M. ; ENDLER, M.. **Gradys: Exploring movement awareness for efficient routing in ground-and-air dynamic sensor networks**. 2020.

[9] MARCONATO, E.; RODRIGUES, M.; PIRES, R.; PIGATTO, D.; QUERINO FILHO, L.; PINTO, A. ; CASTELO BRANCO, K.. **Avens – a novel flying ad hoc network simulator with automatic code generation for unmanned aircraft system**. 01 2017.

[10] BAIDYA, S.; SHAIKH, Z. ; LEVORATO, M.. **Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot**. In: PROCEEDINGS OF THE 21ST ACM INTERNATIONAL CONFERENCE ON MODELING, ANALYSIS AND SIMULATION OF WIRELESS AND MOBILE SYSTEMS, MSWIM '18, p. 37–45, New York, NY, USA, 2018. Association for Computing Machinery.

[11] **X-plane 11 flight simulator**. `https://www.x-plane.com/`. Acesso em: Setembro/2021.

[12] **Mavproxy**. `https://ardupilot.org/mavproxy/`. Acesso em: Setembro/2021.

[13] **Mavlink**. `https://mavlink.io/en/`. Acesso em: Setembro/2021.

[13] **INET, an open-source omnet++ model suite for wired, wireless and mobile networks. inet evolves via feedback and contributions from the user community**. `https://mavlink.io/en/`. Acesso em: Setembro/2021.

[15] **Ardupilot/pymavlink: Python mavlink interface and utilities**. `https://github.com/ArduPilot/pymavlink`. Acesso em: Novembro/2021.

[16] **Mission planner is a full-featured ground station application for the ardupilot open source autopilot project**. `https://ardupilot.org/planner/index.html`.

[17] `https://github.com/Thlamz/MAVLinkExperiments`. Acesso em: Novembro/2021.

[18] **Boost provides free peer-reviewed portable c++ source libraries**. `https://www.boost.org/`. Acesso em: Novembro/2021.

[19] **Mavlink - common message set**. `https://mavlink.io/en/messages/common.html`.

[20] **Mavsimnet - código fonte do projeto**. `https://github.com/Thlamz/MAVSIMNET`.

[21] **Mkdocs - project documentation with markdown.** `https://www.mkdocs.org/`.