GridGooseSV: um Módulo do NS-3 para Simular Protocolos de Comunicação de *Smart Grids* definidos na IEC 61850

Lucas Seiki Oshiro[1], **Natalia Castro Fernandes**[2], **Daniel Macêdo Batista**[1]

[1]Instituto de Matemática, Estatística e Ciência da Computação (IME-USP)
Universidade de São Paulo (USP)

[2]MídiaCom - PPGEET/TET/UFF
Universidade Federal Fluminense (UFF)

{lucas.oshiro,batista}@ime.usp.br, nataliacf@id.uff.br

Abstract. *In Smart Grid communications, simulators play a vital role in evaluating performance. However, many of them lack native support for IEC 61850, adopted by digital substations, driving the need for simulator modules that support IEC 61850-based systems accurately and at scale. This paper presents* `GridGooseSV`*, an open-source* `ns-3` *module for simulating Smart Grid protocols GOOSE and SV. It uses* `libiec61850`*, an industry-adopted implementation. Experimental results demonstrate its ability to assess how network configurations affect protocol behavior, offering a valuable tool for analyzing and optimizing Smart Grid communication performance.*

Resumo. *Em comunicações de Smart Grids, simuladores são vitais para análise de desempenho. Porém, muitos carecem de suporte nativo para o padrão IEC 61850, usado em subestações digitais, o que justifica novos módulos de simulação que suportem esse padrão com precisão e em larga escala. Este artigo apresenta o* `GridGooseSV`*, um módulo de código aberto para* `ns-3` *que simula os protocolos GOOSE e SV, usados em Smart Grids. Ele utiliza a biblioteca* `libiec61850`*, uma implementação adotada pela indústria. Experimentos demonstram que o módulo permite avaliar o impacto de configurações de rede no comportamento dos protocolos, sendo uma ferramenta valiosa para analisar e otimizar Smart Grids.*

1. Introdução

A análise de desempenho em redes elétricas inteligentes (*Smart Grids*) é essencial para fins de pesquisa e planejamento de capacidade, particularmente em áreas como as subestações digitalizadas e os sistemas de tele-proteção, em que a confiabilidade e a baixa latência são críticas para operação segura e eficiente da rede. O alto custo da aquisição de equipamentos reais e os riscos de acidentes inerentes ao ambiente de transmissão de energia elétrica justificam que essa análise de desempenho seja realizada por meio de simulação.

Simular uma *smart grid* consiste em simular os sinais elétricos e a comunicação de informação entre os elementos do ambiente. O lado elétrico da simulação pode ser feito em simuladores como o `OpenDSS` ([EPRI 2025]), enquanto o lado das comunicações

pode ser feito por simuladores como o `ns-3` ([ns-3 project 2026]) ou o `OMNeT++` ([OMNeT++ 2026]). Em particular, é importante que a comunicação seja simulada seguindo algum padrão amplamente adotado, como a norma IEC 61850. Essa norma define um modelo de dados e protocolos como o SV e o GOOSE para a comunicação entre os vários elementos da *Smart Grid*. O SV é um protocolo publicação-assinatura (pub-sub) baseado em Ethernet que envia leituras de valores instantâneos de sinais analógicos, como corrente e tensão, para Dispositivos Eletrônicos Inteligentes (*Intelligent Electronic Devices* – IEDs). Esses valores são normalmente amostrados em altas taxas (por exemplo, 4800 amostras por segundo) e enviados usando transmissão multicast com alta prioridade. O GOOSE também é um protocolo pub-sub baseado em Ethernet, mas é usado para comunicações multicast entre IEDs para transportar dados críticos em termos de tempo, como disparos de proteção, alarmes e sinais de intertravamento.

Alguns trabalhos já propuseram simular protocolos da IEC 61850 em simuladores de redes existentes. [León et al. 2016] desenvolveram um módulo para o OMNeT++ que implementa os protocolos SV e GOOSE por meio do framework INET. No entanto, esse módulo não recebe manutenção atualmente e seu código-fonte está fortemente acoplado a uma versão antiga do framework INET, o que dificulta sua integração com outros cenários. [Hwang et al. 2018] propuseram emular o tráfego SV e GOOSE por meio de agendamento em tempo real no ns-3. Contudo, em experimentos preliminares, mostrou-se que essa abordagem apresenta limitações, principalmente em relação ao número máximo de arquivos abertos permitidos pelo sistema operacional subjacente e à complexidade de gerenciar múltiplas bases de código-fonte – no mínimo, o código do cenário de simulação, os geradores de tráfego e os consumidores de tráfego. Vale destacar ainda a possibilidade de outras abordagens para avaliar redes de comunicação de *Smart Grids* sem o uso de um simulador. Nesse sentido, destacamos o TITAN [Soares et al. 2021], um gerador de tráfego IEC 61850 que pode ser integrado a hardware real.

Este artigo apresenta um novo módulo de simulação, chamado `GridGooseSV`, para o simulador de redes `ns-3` que implementa os protocolos SV e GOOSE. A vantagem deste módulo é que ele se baseia na biblioteca de código aberto `libiec61850` [MZ Automation 2025], uma implementação do IEC 61850 já utilizada na indústria. O `GridGooseSV` difere dos trabalhos da literatura por ter sido desenvolvido seguindo práticas consolidadas de programação, como desenvolvimento orientado a testes (TDD) utilizando a biblioteca de testes do `ns-3`, integração contínua por meio do GitHub Actions e um controle de versão semântico utilizando *commits* convencionais. Além disso, este módulo possibilita a escalabilidade de experimentos sem as limitações do sistema operacional, como o número de arquivos abertos, e aproveita todos os recursos já existentes no `ns-3`. Em uma série de experimentos para avaliar o desempenho do `GridGooseSV`, foi possível demonstrar a sua capacidade de simular simultaneamente múltiplos fluxos GOOSE e SV, atendendo consistentemente às restrições de temporização necessárias para a geração de mensagens em um ambiente realista de *smart grid*.

O restante deste artigo está organizado da seguinte forma: A Seção 2 descreve a arquitetura e as principais funcionalidades do `GridGooseSV`. A Seção 3 apresenta experimentos que atestam a eficácia do módulo. A Seção 4 lista as URLs onde ele pode ser obtido bem como onde se encontram a sua documentação e requisitos para demonstrar o seu funcionamento. A Seção 5 apresenta ideias para extensão do `GridGooseSV`.

2. Arquitetura e Funcionalidades do `GridGooseSV`

Visando a utilização em pesquisas futuras e por outros grupos de pesquisa, durante todo o desenvolvimento buscou-se garantir que o `GridGooseSV` seguisse os princípios FAIR4RS [Chue Hong et al. 2022]: *Findable*, *Accessible*, *Interoperable* e *Reusable*.

Um dos requisitos que foi considerado para o `GridGooseSV` foi ser uma única base de código C++ na forma de um módulo do `ns-3`. Isso permite que ele seja reutilizado por outras pessoas sem a necessidade de lidar diretamente com seu código-fonte, apenas usando os recursos por ele oferecidos.

O `GridGooseSV` também teve como requisito conter implementações de *publishers* e *subscribers* SV e GOOSE que funcionassem como aplicações nativas do `ns-3`, isto é, usando seus recursos ao invés dos recursos do sistema operacional. Isso permite que o `GridGooseSV` seja autossuficiente no que se refere à criação, envio e recebimento de mensagens SV e GOOSE, dispensando o uso de dispositivos de redes virtuais no sistema operacional como `TapBridge` e não precisando seguir as restrições de tempo real. Outra consequência é ser agnóstico em relação ao sistema operacional sobre o qual o `ns-3` é executado. A implementação de referência dos protocolos SV e GOOSE é a `libiec61850`, escolhida por sua maturidade, confiabilidade, sustentabilidade e documentação abrangente. Ainda que o `GridGooseSV` seja baseado em implementações existentes dos protocolos SV e GOOSE, seu desenvolvimento não foi trivial, uma vez que foi necessário portar partes da `libiec61850` para serem usadas no `ns-3`. Os desafios apresentados têm duas causas principais: i) As linguagens de programação serem diferentes, sendo a `libiec61850` escrita em C e o `ns-3` em C++; ii) A finalidade original da `libiec61850` de ser utilizada em *hardware* real, em tempo real e sobre um sistema operacional.

Além da adaptação do código da `libiec61850`, também foi necessário desenvolver os demais componentes que constituem um módulo do `ns-3`. O desenvolvimento do `GridGooseSV` se deu, então, em três etapas:

1. Conversão do código-fonte referente ao SV e ao GOOSE da `libiec61850` para C++;
2. Desenvolvimento de uma nova camada para a HAL (*Hardware Abstraction Layer*) da `libiec61850` sobre `ns-3`[1];
3. Escrita de aplicações, *helpers* e testes, seguindo a convenção do `ns-3`.

O fato do `GridGooseSV` não requerer o uso de `TapBridges`, torna possível executá-lo sem permissão de superusuário. Além disso, ser um módulo do `ns-3` faz com que seus recursos sejam disponibilizados diretamente para o cenário de simulação escrito pelos usuários em C++ na forma de uma biblioteca, dispensando o uso de *scripts* auxiliares e de aplicações externas.

Para que pudesse ser reutilizado para pesquisas futuras, o `GridGooseSV` seguiu boas práticas de desenvolvimento de *software*, como conter testes automatizados, ter um bom histórico de *commits* e seguir as práticas recomendadas para a sua linguagem de programação (no caso, C++ no estilo usado pelo `ns-3`).

[1] A HAL permite a compatibilidade da biblioteca com Linux, Windows, macOS e BSD usando os mesmos cabeçalhos. Foi necessário desenvolver suporte ao `ns-3` levando em conta o fato de que ele é um simulador e não um sistema operacional.

Seguindo o layout de módulos do `ns-3` ([ns-3 project 2024]), o `GridGooseSV` é composto por aplicações, *helpers* e testes, com alguns exemplos simples demonstrando como usar seus recursos. A Figura 1 mostra uma visão geral da arquitetura do `GridGooseSV`. Nela, estão representados em vermelho os componentes embutidos no `ns-3`, com seus modelos de rede, aplicações, *helpers* e demais recursos que servem como base para a criação de cenários de simulação e para a criação de novos módulos. Em verde está representado o código-fonte que teve que ser escrito para fornecer recursos do `ns-3` para a `libiec61850` e vice-versa. Isto é, o desenvolvimento de uma nova HAL para a `libiec61850` usando os recursos fornecidos pelo `ns-3` e o desenvolvimento de aplicações e *helpers* para o `ns-3` usando os recursos fornecidos pela `libiec61850`. Em amarelo está representado o código-fonte e as interfaces da `libiec61850` que precisaram ser adaptados para poderem interoperar com o `ns-3`. Em azul está representado o código a ser escrito pelo usuário, ou seja, aplicações do `ns-3` personalizadas e cenários de simulação que usam essas aplicações customizadas, aplicações e *helpers* fornecidas pelo `GridGooseSV` e `ns-3` e demais componentes também fornecidos pelo `ns-3`.

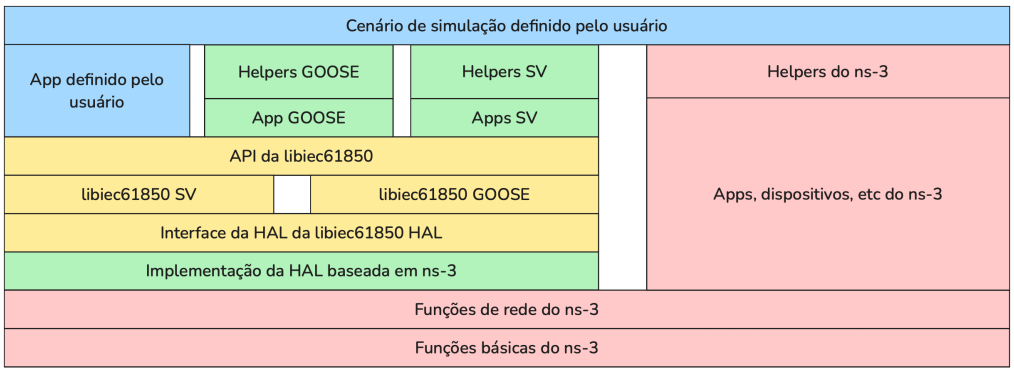

Figura 1. Arquitetura do `GridGooseSV`.

Sendo assim, no `GridGooseSV` a `libiec61850` usa os serviços do `ns-3` e provê: uma API de alto nível, fornecendo aplicações do `ns-3` prontas para uso para *publishers* e *subscribers* nos protocolos SV e GOOSE, além de classes auxiliares para facilitar sua instalação em nós do `ns-3`; e uma API de baixo nível, fornecendo acesso direto às funções, estruturas de dados e protocolos da `libiec61850`, permitindo que usuários escrevam suas próprias aplicações.

O `GridGooseSV` provê quatro classes de aplicação do `ns-3`, uma para o *publisher* e outra para o *subscriber* tanto do protocolo SV quanto do protocolo GOOSE. Elas são baseadas nos exemplos fornecidos pela `libiec61850` e também podem ser configuradas através do sistema de configurações do `ns-3`. Alternativamente, as funções da `libiec61850` relacionadas ao GOOSE e ao SV são expostas em cabeçalhos públicos no `GridGooseSV`. Dessa forma, quem precisar desenvolver uma aplicação poderá usar essas funções e seguir a documentação existente da `libiec61850`, configurando uma simulação como a mostrada na Figura 2, por exemplo. Isso pode ser especialmente útil para escrever a simulação de comunicações quando se está usando plataformas de co-simulação, como o HELICS ([Palmintier et al. 2017]) e o Mosaik ([Schütte et al. 2012]).

Além disso, dado que o `ns-3` tem capacidades de simulação em tempo real que permitem a troca de mensagens com dispositivos físicos e emulados, é possível usar o

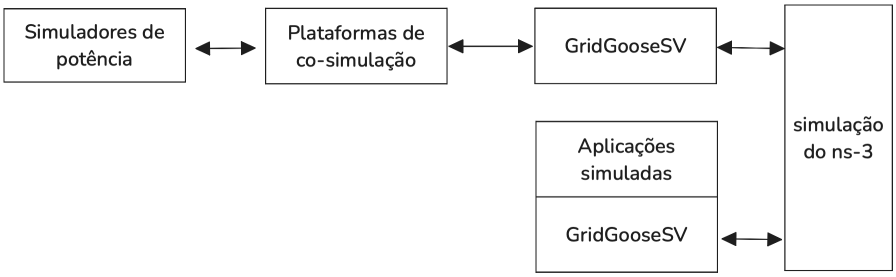

Figura 2. Integração com plataformas de co-simulação

`ns-3` para fornecer uma infraestrutura de rede simulada na qual IEDs físicos e *software* externo baseado em GOOSE ou SV possam se conectar, junto com as já mencionadas plataformas de aplicação. Uma vantagem dessa configuração é que ela permite o uso do *framework* de coleta de dados do ns-3, permitindo posteriores análises de dados da rede.

Este módulo é simples o suficiente para a configuração de um cenário de simulação sem precisar lidar diretamente com os detalhes de implementação dos protocolos da IEC 61850, além de ser flexível o suficiente para a escrita de aplicações personalizadas que atendam melhor as necessidades de cada usuário.

3. Eficácia do `GridGooseSV`

Para avaliar o `GridGooseSV`, foram executados experimentos em um hardware com a seguinte configuração: Processador i7-12700H, com 20 núcleos, 32GiB de RAM e armazenamento em SSD NVMe de 200GB. Em termos de software, a configuração foi: `ns-3`: versão 3.43, SO Manjaro Linux 25 e sistema de arquivos BtrFS.

3.1. Protocolo SV

Os dois principais casos de uso para o envio de mensagens através do protocolo SV são a proteção e a medição e, dentre eles, os cenários de proteção são os que enviam mensagens em maior frequência. Em redes elétricas que operam a 60Hz, um *publisher* SV envia 4800 mensagens por segundo no cenário de proteção, enquanto que em redes elétricas que operam a 50Hz essa taxa de envio é de 4000 mensagens por segundo. Portanto, os casos de uso de proteção a 60Hz são os que demandam maiores recursos computacionais para serem simulados. Sendo assim, eles foram usados para avaliar o funcionamento do SV no `GridGooseSV`.

3.1.1. Envio de dados

Um primeiro experimento foi simular o envio de leituras de corrente de uma rede trifásica de um *publisher* SV para um *subscriber* para assegurar que o protocolo SV estava enviando os dados corretamente. No *publisher*, definimos uma função para gerar valores de corrente fictícios para três fases de uma corrente alternada a 60Hz, com amplitude de 18000A (valor baseado em [Soares et al. 2021]) e defasagem de 120° entre elas. Em outras palavras, são três funções senoidais que no instante de tempo t devolvem os seguintes valores: **Fase A**: $\sin(t * 2\pi * 60) * 18000$, **Fase B**: $\sin(\frac{2\pi}{3} + t * 2\pi * 60) * 18000$ e **Fase C**: $\sin(\frac{4\pi}{3} + t * 2\pi * 60) * 18000$.

O *publisher* realizou amostras dessas funções 4800 vezes por segundo e as enviou através do protocolo SV. O *subscriber*, então, coletava os dados e preenchia um arquivo SV com essas leituras. Após a execução, os dados foram carregados para um `DataFrame` da biblioteca Pandas do Python para análise de dados, e a partir dele, foi possível traçar um gráfico representando os valores de corrente dessas três fases nos primeiros 100 milissegundos simulados. Esse gráfico pode ser visto na Figura 3.

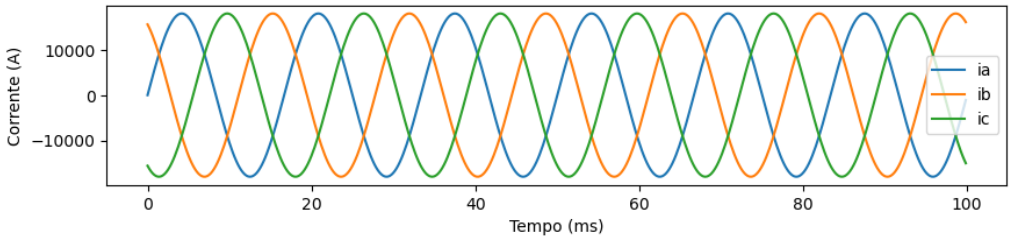

Figura 3. Gráfico de dados obtidos de mensagens SV geradas dentro do `ns-3`.

Com esse experimento foi possível certificar que o envio e recebimento de mensagens SV estava sendo feito corretamente dentro de um ambiente simulado.

3.1.2. Tempo decorrido

Para entender a viabilidade de usar o `GridGooseSV` para simular pacotes SV, o seu desempenho foi medido com diferentes quantidades de *publishers* e um único *subscriber*.

Um caso de uso intensivo do GridGooseSV seria um cenário de proteção a 60Hz, que envia 4800 mensagens por segundo e geraria arquivos `pcap` contendo todas elas. Dessa forma, foi avaliado o desempenho do `GridGooseSV` em duas situações: uma gerando os arquivos `pcap` e outra sem gerá-los. O número de *publishers* SV começava em 1 e dobrava na amostragem seguinte até chegar em 128 *publishers*. Dessa forma, há 8 quantidades diferentes de *publishers*, com e sem arquivos `pcap`, totalizando 16 amostras.

Dado que o escalonamento de eventos do `ns-3` é *single-threaded*, o uso de CPU manteve-se em 100% de um único núcleo em todas as amostras. Dessa forma, um maior número de núcleos não impacta no desempenho da simulação.

O tempo decorrido na simulação variou de acordo com o número de *publishers*. Os valores são mostrados na Figura 4, em que é possível ver que as execuções que geraram arquivos `pcap` levaram mais tempo para serem concluídas.

O consumo de memória RAM do `GridGooseSV` em todos esses cenários não aumentou significativamente: a diferença entre o uso de RAM do cenário com 128 *publishers* e do cenário com apenas um *publisher* foi de apenas 250 MiB. A memória disponível, então, não é um gargalo para este módulo.

3.1.3. Simulação em tempo real

Também foi avaliado o desempenho do `GridGooseSV` no modo de tempo real enviando pacotes SV na taxa de proteção e observando o atraso entre a geração de pacotes no

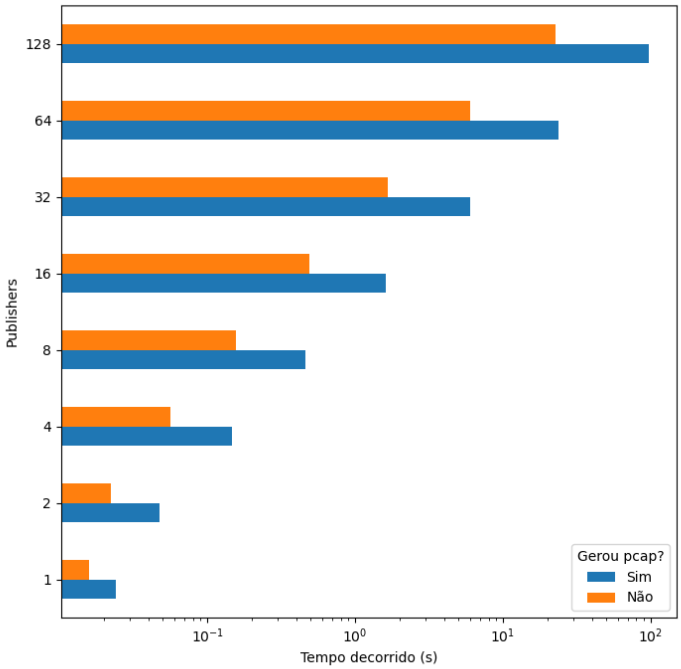

Figura 4. Tempo de simulação para diferentes quantidades de *publishers* SV, gerando arquivos `pcap` ou não.

publisher e a sua recepção no *subscriber*. Neste cenário, foi possível simular até 17 *publishers*, visto que adicionar mais publishers fez a simulação de tempo real abortar porque ao menos um evento excedeu seu tempo agendado.

No ambiente utilizado, com 17 *publishers SV*, a mediana do tempo decorrido entre o envio das mensagens e seus respectivos recebimentos foi de apenas 8 µs e o tempo decorrido máximo foi de 284 µs no pior caso (17 *publishers*), menos de um décimo do requisito de latência para a classe TT6 definida na IEC 61850 ([Barbierato et al. 2020]).

Dessa forma, o `GridGooseSV` também pode ser usado para a geração de tráfego SV em uma rede real.

3.2. Protocolo GOOSE

Para avaliar o protocolo GOOSE, foi criado um cenário de uma comunicação com um *publisher* e um *subscriber* conectados através de um canal CSMA. Após 100 milissegundos, foi simulado um evento crítico enviando mensagens com intervalos seguindo uma progressão geométrica, além de um segundo evento crítico após 100ms do primeiro evento. Foram avaliados diferentes intervalos de atraso no canal, partindo de 0ms a 9ms.

O impacto desse atraso pode ser visto na Figura 5. Nessa figura é mostrada uma linha do tempo da chegada de mensagens GOOSE com diferentes valores de atraso no canal CSMA. As linhas verticais indicam quando cada mensagem foi enviada, sendo suas cores iguais às dos pontos que representam o momento de suas chegadas. É possível notar que com 9ms de atraso no canal, algumas mensagens críticas só foram recebidas após o

envio da mensagem seguinte.

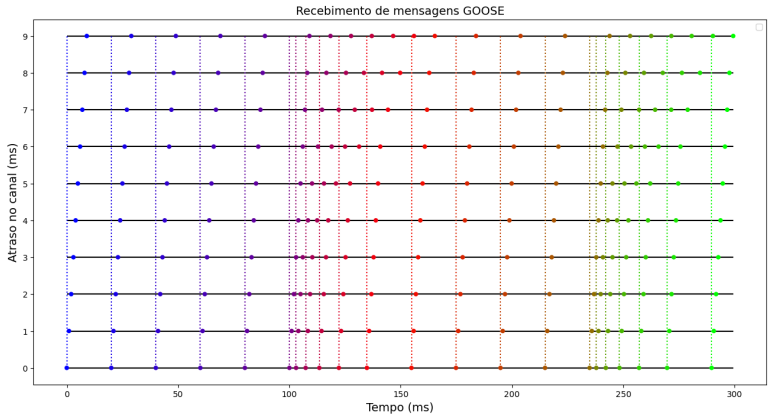

Figura 5. Linha do tempo da chegada de mensagens GOOSE.

4. Disponibilidade e Recursos Necessários

O código-fonte do `GridGooseSV` é distribuído sob a licença GPL v3 e está disponível no GitHub[2]. Seu `README` fornece a documentação para instalação do módulo em 4 passos. O código-fonte dos experimentos relatados na Seção 3 está disponível como exemplos de uso[3].

Recomenda-se reproduzir os experimentos em um computador com Linux ou macOS; no mínimo 10GB de espaço livre em disco; processador Intel i5 de 10ª geração ou superior e 8GB de RAM.

Um vídeo com a instalação e uso do GridGooseSV está disponível no YouTube[4]. No salão de ferramentas será demonstrada a criação de cenários simples de comunicação usando pacotes SV e GOOSE, atestando seu funcionamento através da geração de arquivos `.pcap`.

5. Trabalhos Futuros

O padrão IEC 61850 define um terceiro protocolo chamado MMS, um protocolo cliente-servidor baseado em TCP/IP voltado para monitorar e configurar parâmetros em tempo de execução de IEDs por meio de sistemas SCADA. Uma ideia de trabalho futuro consiste em implementar esse protocolo no `GridGooseSV`. Outra ideia consiste em estender os exemplos incluindo alguma integração com hardware real.

Agradecimentos

Esta pesquisa é parte do CPE SMARTNESS (FAPESP 21/00199-8) e parte do projeto de pesquisa STARLING – Segurança e Alocação de Recursos em B5G via Técnicas de Inteligência Artificial (FAPESP/MCTI/MCom/CGI.br proc. 2021/06995-0). Ela também é parte do projeto FAPESP proc. 2024/09448-9.

[2] `https://github.com/lucasoshiro/GridGooseSV`
[3] `https://github.com/lucasoshiro/GridGooseSV/tree/main/examples`
[4] `https://www.youtube.com/watch?v=lkWwmxJp6Ls`

Referências

Barbierato, L., Estebsari, A., Bottaccioli, L., Macii, E., and Patti, E. (2020). A Distributed Multimodel Cosimulation Platform to Assess General Purpose Services in Smart Grids. *IEEE Transactions on Industry Applications*, 56(5):5613–5624.

Chue Hong, N. P., Katz, D. S., Barker, M., Lamprecht, A.-L., Martinez, C., Psomopoulos, F. E., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., et al. (2022). FAIR Principles for Research Software (FAIR4RS Principles). `https://zenodo.org/records/6623556`. Último acesso em 15 de Fevereiro de 2026.

EPRI (2025). OpenDSS. `https://sourceforge.net/projects/electricdss/`. Último acesso em 15 de Fevereiro de 2026.

Hwang, S.-H., Im, Y., Song, H.-C., and Park, J. (2018). Real Time Emulation of IEC61850 SV, GOOSE and MMS using NS-3. *Journal of Engineering and Applied Sciences*, 13(3):634–638.

León, H., Montez, C., Stemmer, M., and Vasques, F. (2016). Simulation Models for IEC 61850 Communication in Electrical Substations using GOOSE and SMV Time-Critical Messages. In *Anais do IEEE World Conference on Factory Communication Systems (WFCS)*, pages 1–8.

MZ Automation (2025). IEC 61850 Protocol Library. `https://www.mz-automation.de/iec-61850-protocol-library`. Último acesso em 15 de Fevereiro de 2026.

ns-3 project (2024). ns-3 manual. `https://www.nsnam.org/docs/release/3.43/manual/html/index.html`. Último acesso em 15 de Fevereiro de 2026.

ns-3 project (2026). ns-3 — a discrete-event network simulator for internet systems. `https://www.nsnam.org/`. Último acesso em 15 de Fevereiro de 2026.

OMNeT++ (2026). OMNeT++ Discrete Event Simulator. `https://omnetpp.org/`. Último acesso em 15 de Fevereiro de 2026.

Palmintier, B., Krishnamurthy, D., Top, P., Smith, S., Daily, J., and Fuller, J. (2017). Design of the HELICS High-Performance Transmission-Distribution-Communication-Market Co-Simulation Framework. In *Anais do Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, pages 1–6.

Schütte, S., Scherfke, S., and Sonnenschein, M. (2012). Mosaik – Smart Grid Simulation API – Toward a Semantic based Standard for Interchanging Smart Grid Simulations. In *Anais do 1st International Conference on Smart Grids and Green IT Systems (SMARTGREENS)*, pages 14–24.

Soares, A. A. Z., Soares, L. F., Mattos, D. P., Pinheiro, P. H., Quincozes, S. E., Ferreira, V. C., Apostolo, G. H., Carrara, G. R., Moraes, I. M., Albuquerque, C., et al. (2021). Enabling Emulation and Evaluation of IEC 61850 Networks with TITAN. *IEEE Access*, 9:49788–49805.