PairSim: um modelo de comunicação bidirecional entre simuladores

PairSim: a bi-directional simulator coupling model

Franco Barpp Gomes (orientado) *, Daniel Fernando Pigatto (orientador)†

15 de setembro de 2021

RESUMO

Com a evolução de sistemas computacionais, cresce a demanda por melhorias no gerenciamento de recursos de redes visando disponibilidade e acessibilidade e, por isso, há interesse na simulação de cenários que envolvem redes de computadores. Porém, os cenários e objetivos de testes são extremamente diversos; em muitos casos, um simulador específico não atende a todas as variáveis a serem estudadas. Com a inspiração da área de redes, foi desenvolvido o *PairSim*, um modelo de intercomunicação entre simuladores que visa facilitar o uso simultâneo de dois simuladores quaisquer em cenários de teste. Esse modelo foi implementado como uma biblioteca C++ e, posteriormente, essa base foi aplicada na recriação da funcionalidade do *AVENS*, uma ferramenta de simulação conjunta entre o simulador discreto de redes *OMNeT++* e o simulador realista de voo *X-Plane*, em um cenário de *Flying Ad-hoc Networks* (FANETs). A modificação para a base do *PairSim* permite que o *AVENS* opere com versões mais recentes dos simuladores, o que não era possível anteriormente.

Palavras-chave: Redes de Computadores. Sistemas Distribuídos. Simulação.

ABSTRACT

The more computer systems evolve, the higher is the demand for improvements in network resources management aiming at availability and accessibility, and so is the interest in simulating computer network scenarios. However, simulation scenarios and objectives are extremely diverse; in some cases, only one specific simulator can't cover all of the topics a research needs. Inspired by this situation in the computer network field, *PairSim*, a bi-directional simulator coupling model that aims to make coupling between any two simulators easier, was developed. This model was then implemented as a C++ library and tested as the basis of the reimplementation of a tool called *AVENS*, which coupled the discrete network simulation framework *OMNeT++* and the flight simulator *X-Plane* to enable simulation of Flying Ad-hoc Networks (FANETs) scenarios. This reimplementation of *AVENS*, using *PairSim*, now allows *AVENS* to operate with the latest versions of both simulators, something that was not possible before.

Keywords: Computer Networks. Distributed Systems. Simulation.

1 INTRODUÇÃO

O uso de simulação computacional é muito importante para representação de determinadas situações que são difíceis de serem colocadas em prática. Por exemplo, o treinamento de um piloto de avião iniciante: de início, uma aula prática em um avião de fato teria não somente um custo alto, mas seria também perigosa. Treiná-lo primeiramente em um bom simulador acaba sendo uma alternativa muito mais viável.

* Curso de Engenharia de Computação; francog@alunos.utfpr.edu.br.
† Programa de Pós-graduação em Computação Aplicada; pigatto@utfpr.edu.br; https://orcid.org/0000-0001-8528-7407.

Em redes de computadores isso não é diferente: alguns cenários de teste podem ser simplesmente inviáveis de serem realizados fisicamente, e a capacidade de simulação abre muitas possibilidades. Questões como o planejamento de medidas em caso de desastres em redes computacionais, estas envolvendo rotas alternativas de comunicação e distribuição de sinal usando equipamentos como *drones*, são um ótimo exemplo do nível de complexidade que é possível atingir com apenas um simulador, mas que na prática seria extremamente difícil.

Nem sempre as soluções de simulação disponíveis suportam todas as necessidades de uma pesquisa. Por isso, em alguns casos, são feitas simulações usando não um, mas dois simuladores simultaneamente, com uma interface que possibilite a troca de mensagens ou compartilhamento de memória entre ambos. No âmbito da simulação de Redes Veiculares *Ad Hoc* (VANETs): "Um ambiente de simulação para VANETs idealmente requer simuladores de tráfego veicular e comunicação de redes interconectados" (ARELLANO; MAHGOUB, 2013, tradução nossa). Também, "a mobilidade dos nós é estritamente limitada no cenário realístico de tráfego e a alta velocidade leva a mudanças mais frequentes na topologia da rede. Então, para VANET, não é apropriado usar um simulador de tráfego ou um simulador de redes por si só" (YANG; HAO; CAI, 2016, tradução nossa).

No entanto, esse é um processo de implementação relativamente difícil e o leque de ferramentas disponíveis para tanto é limitado — normalmente vinculado somente entre dois simuladores específicos, como a *VEINS* (SOMMER et al., 2019) — e, se os simuladores almejados não têm uma ferramenta desse tipo, a necessidade de desenvolver um esquema complexo de comunicação pode ser inviável. Assim, levanta-se o questionamento: o que poderia ser desenvolvido para simplificar o processo para, efetivamente, qualquer par de simuladores?

Assim, o objetivo principal do presente trabalho é de estabelecer um padrão geral para intercomunicação de simuladores que possa, dessa forma, ser a base de módulos específicos para cada simulador. Com a implementação padronizada da comunicação, a dificuldade de implementação de um módulo seria menor e, consequentemente, seria de se esperar que a facilidade de encontrar módulos já prontos, inclusive, fosse maior. Visto que existem diversos usos que podem se beneficiar da técnica de intercomunicação de simuladores, mas que por vezes são tão específicos que as ferramentas existentes não cobrem suas necessidades, o uso de um padrão como o descrito pode vir a ser um grande facilitador de experimentos.

Como uma forma de avaliar os resultados e introduzir uma implementação como exemplo, será recriada a ferramenta *AVENS* (MARCONATO et al., 2017), que gerencia a execução conjunta da *framework* de simulação de redes *OMNeT++* (VARGA, 2010) e do simulador de voo *X-Plane* (MEYER, 2021), a qual está disponível somente para versões mais antigas dos simuladores. Com isso, seria possível introduzir uma versão funcional.

2 MÉTODO

O estudo foi iniciado com uma pesquisa bibliográfica, com o objetivo de entender como as ferramentas de integração de simuladores específicos operam. Essa pesquisa se mostrou útil principalmente no ponto da comparação entre possíveis métodos de sincronização da execução dos simuladores, o que é extremamente importante nesses casos, já que uma diferença temporal instável e desconhecida entre os simuladores pode abrir uma grande margem para erros nos resultados a serem obtidos.

Em seguida, foram desenvolvidos alguns protótipos da funcionalidade desejada, de forma que muitas decisões de *design* envolvidas puderam ser avaliadas de forma mais concreta — considerando não somente a funcionalidade em si, algo mais simples de ser discutido de forma abstrata, mas também sua facilidade de uso, uma das ideias iniciais. Esse processo levou a um *design* final muito mais bem estabelecido.

Com o modelo já formado e definido, foi feita a implementação de uma biblioteca para a linguagem C++ que

encapsula toda a lógica de sincronização e troca de dados através de um simples esquema de classes. Então, utilizando a base da biblioteca, foram feitos *plugins* para alguns simuladores: um para a *framework* de simulação *OMNeT++*, e outro para o simulador de voo *X-Plane*, e, além disso, um programa independente, que mostra a possibilidade de se sincronizar qualquer conteúdo programável com um simulador compatível.

3 RESULTADOS

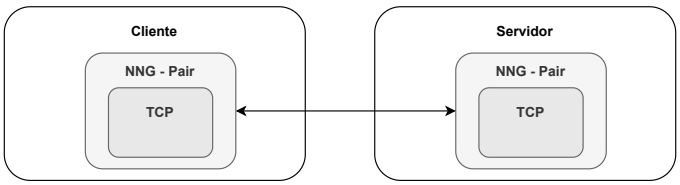
Esta seção descreve o modelo proposto em etapas.

3.1 Comunicação

A comunicação começa a partir de um programa até outro — o módulo que inicialmente espera por uma conexão é um *Servidor*, enquanto o iniciador é um *Cliente*. Em outros pontos, seus funcionamentos são idênticos.

Como representado na Fig. 1, a comunicação entre as partes é feita através do *Transmission Control Protocol* (*TCP*), mais especificamente sobre um protocolo chamado *Nanomsg Next Generation* (*NNG*) (D'AMORE, 2021), uma das mais novas propostas do criador dos protocolos *ZeroMQ* e *Nanomsg*, ambos populares em seus nichos. O *NNG*, que pode ser considerado um exemplo do estado da arte desse tipo de ferramenta, oferece uma camada de abstração para uma comunicação *TCP socket* tradicional, com utilidades como reconexão e padronização de mensagens, mas com um *overhead* mínimo. O último ponto foi o mais atrativo para o projeto, já que é oferecida a possibilidade de uma experiência de desenvolvimento, manutenção e estabilidade melhores, mas sem sacrificar significativamente a eficiência da aplicação.

Figura 1 – Comunicação entre módulos



Fonte: Autoria própria (2021).

Nesse caso, foi utilizado o padrão *pair* do NNG, que implementa uma comunicação do tipo *one-to-one peer-to-peer* — cada *peer* pode estar conectado com outro *peer*, e ambos podem enviar dados livremente — que é exatamente o esperado no caso deste trabalho.

3.2 Sincronização

A partir do momento em que a conexão *TCP* é estabelecida, ambos os módulos devem sinalizar um ao outro, com mensagens específicas, que a execução pode ser iniciada; somente assim o módulo do servidor começará o processo de sincronização dos simuladores. Em seguida, a partir do cliente, cada módulo executa um passo pré-definido da simulação, envia seus resultados e espera a execução do outro, de forma que efetivamente somente um módulo estará sendo executado por vez. Uma boa analogia é a da corrida de bastão: cada atleta corre por uma certa distância e, então, passa o bastão para um próximo, que vai correr até o ponto seguinte.

Esse método tem vantagens e desvantagens que merecem destaque. A principal vantagem é que não é possível haver choque de comandos entre simuladores — se ambos fossem executados ao mesmo tempo, o que

não é o caso, as informações de um poderiam sinalizar a necessidade de parar um carro, por exemplo, enquanto no outro a sinalização poderia ser a de movimentá-lo — nesse caso, haveria um conflito, já que não se sabe qual orientação priorizar. Outra vantagem é que sabe-se exatamente qual é a diferença temporal entre os simuladores: um simulador estará um passo (pré-definido pelos módulos) à frente de outro. Isso também mostra a principal desvantagem: há uma diferença temporal definida e, portanto, conhecida, mas que pode ou não ser significativa.

Essa diferença temporal seria muito menor, por exemplo, se a execução fosse não de um módulo por vez, mas de ambos simultaneamente. No entanto, um 'descompasso' ocasional entre esses iria causar grandes problemas; o *tradeoff* em questão é que, embora menor, a diferença temporal é potencialmente instável, o que também não é algo desejável em muitas situações. No método utilizado no projeto, no caso de um descompasso como o descrito, não haveria problema algum — quando um simulador demorar mais que o esperado no tempo real, isso só significa que ele demoraria mais para acionar a execução seguinte.

3.3 Troca de Mensagens

A troca de mensagens ocorre através de um padrão próprio, com diferentes tipos de mensagens que incluem as notificações necessárias à sincronização e descrição de dados. Esses pacotes são escritos em *Concise Binary Object Representation* (CBOR) (BORMANN; HOFFMAN, 2020) , um formato de representação binária semelhante ao *JavaScript Object Notation* (JSON) (PEZOA et al., 2016), principalmente na sintaxe e no ponto em que não é necessário o estabelecimento de um esquema específico para um objeto.

Essa característica aumenta consideravelmente a versatilidade da ferramenta — no caso de uma simulação que envolva um carro e um avião, por exemplo, não é necessário descrever previamente um esquema de dados para cada um desses — basta descrever os dados que devem ser transmitidos no momento do envio.

3.4 Biblioteca

O *design* de uso da biblioteca foi definido como o seguinte: um objeto que seja um módulo cliente ou um módulo servidor tem um comportamento padrão a ser implementado, que engloba o esquema definido de comunicação, e tem a possibilidade de incluir um *Modelo*, isto é, um objeto que contém um conjunto de funções para execução em cada fase do processo. Em linguagens de programação que suportam *Object Oriented Programming* (OOP) (STROUSTRUP, 1988), essa ideia é facilmente implementável através de classes e heranças, mas não é estritamente necessário que seja assim para outras linguagens, desde que seja seguida a mesma ideia.

Na execução do *Modelo*, é possível adicionar *Dispositivos* e *Ações*. *Dispositivos* são as entidades que estão sendo simuladas; para declarar um tipo de dispositivo, é necessário somente descrever quais dados desse serão enviados e quais serão recebidos. *Ações*, por sua vez, são definições de algo a ser executado através de uma requisição com parâmetros, semelhante à uma função.

Tanto os dados de um *Dispositivo* quanto os parâmetros de uma ação devem, na implementação, ser tratados somente como objetos genéricos codificados em CBOR. Dessa forma, mesmo em linguagens mais fortemente tipadas, se mantém simples lidar com qualquer tipo de informação necessária.

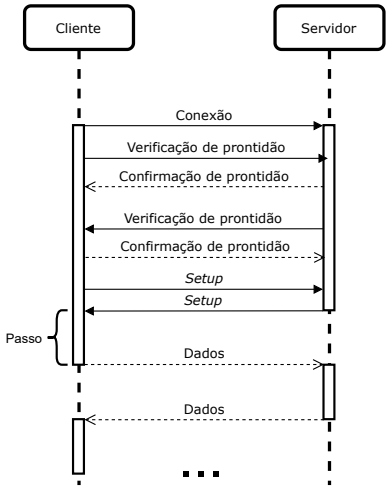
Esse *design* foi concretizado em uma biblioteca C++ de simples utilização, do tipo *header-only*, que disponibiliza um conjunto de classes para cada uma das entidades mencionadas. A escolha da linguagem dessa implementação inicial foi fortemente influenciada pelas interfaces disponíveis em simuladores: em muitos casos, a interface disponível é em C++. Essa implementação deve ser o principal exemplo de possíveis implementações

posteriores em outras linguagens, em termos de usabilidade.

3.5 Estados de Execução

A seguir, é possível descrever a sequência de execução de forma mais completa. A Fig. 2 representa a sequência: primeiramente, é necessário que a conexão se estabeleça e que ambas as partes sinalizem estarem prontas. Depois disso, é feito um processo de *setup*, compartilhando dados do cenário inicial. Assim, a partir do começo da execução, um módulo por vez executa um passo de sua simulação e outro espera por dados; o que executa o faz por um passo pré-definido de sua simulação, interrompe-a, reúne dados de *Dispositivos* e *Ações*, envia-os ao outro módulo e, então, notifica-o para que comece sua execução.

Figura 2 – Diagrama sequencial do modelo



Fonte: Autoria própria (2021).

O outro módulo, por sua vez, estando com sua execução já interrompida, processa os dados enviados pelo primeiro até que seja notificado, através de uma mensagem específica, que tudo o que deveria ser enviado já o foi; só então começa sua própria fase de execução.

3.6 Reimplementação do AVENS

A partir do modelo apresentado e da biblioteca desenvolvida em C++, a funcionalidade da ferramenta AVENS, que se baseava em um protocolo próprio entre dois *plugins*, um em cada simulador, foi reimplementada.

Utilizando os mesmos termos definidos anteriormente, o módulo para o simulador *X-Plane* foi desenvolvido como um servidor, simplesmente através da especialização da classe oferecida pela biblioteca. Seu modelo de execução inclui como dispositivo um avião controlado pelo usuário, definido através da especialização da classe relacionada na biblioteca, de forma que seu método de coleta de dados foi determinado pelas funções da interface de *plugins* do simulador. Além disso, é possível adicionar outros dispositivos facilmente.

O módulo para a *framework* de simulação *OMNeT++*, por sua vez, foi desenvolvido como o cliente. Nesse caso, os dispositivos presentes no cenário do *OMNeT++* não são simplesmente dispositivos *PairSim*: são também módulos *OMNeT++*. Ou seja, para simuladores cujas entidades fazem parte de seu esquema padrão de classes, não se faz necessário criar outro tipo para definir o utilizado pelo *PairSim* — basta torná-los, também,

especializações da classe oferecida pelo *PairSim*.

4 CONCLUSÕES

Este trabalho mostrou que é possível chegar a um modelo factível e próximo do realizado em outras ferramentas de simulação, mas com modificações que permitem maior versatilidade, possibilitando uma aplicação geral sem necessidade de grande trabalho. Esse ponto é essencial para que seja atingido o objetivo inicial: existir uma boa quantidade de implementações de *plugins* de simuladores compatíveis com o modelo estabelecido, para que a integração entre um par desses seja a mais descomplicada possível.

Quanto à reimplementação do *AVENS*, foi possível notar que não somente o uso do modelo resultante é simples, mas a implementação de um *plugin* através da biblioteca também o é, o que facilita o seu uso.

Finalmente, pode-se dizer que houve, com relação aos objetivos estabelecidos, um progresso para a área de simulação envolvendo mais de um simulador. Como trabalhos futuros, sugere-se efetuar mais experimentos e adequações a diferentes simuladores e cenários, buscando atingir uma validação mais completa da integração.

AGRADECIMENTOS

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio financeiro e à UTFPR pela infraestrutura, o que tornou o desenvolvimento do projeto viável.

REFERÊNCIAS

ARELLANO, Wilmer; MAHGOUB, Imad. TrafficModeler extensions: A case for rapid VANET simulation using, OMNET++, SUMO, and VEINS. In: 2013 High Capacity Optical Networks and Emerging/Enabling Technologies. [S.l.: s.n.], 2013. P. 109–115. DOI: 10.1109/HONET.2013.6729767.

BORMANN, C; HOFFMAN, P. RFC 8949 Concise Binary Object Representation (CBOR), 2020.

D'AMORE, Garrett. **NNG - nanomsg-NG**. 31 jul. 2021. Disponível em: . Acesso em: 8 set. 2021.

MARCONATO, Emerson Alberto et al. Avens-a novel flying ad hoc network simulator with automatic code generation for unmanned aircraft system, 2017.

MEYER, Austin. **X-Plane 11 Flight Simulator**. 10 set. 2021. Disponível em: . Acesso em: 10 set. 2021.

PEZOA, Felipe et al. Foundations of JSON schema. In: PROCEEDINGS of the 25th International Conference on World Wide Web. [S.l.: s.n.], 2016. P. 263–273.

SOMMER, Christoph et al. Veins: The open source vehicular network simulation framework. In: RECENT Advances in Network Simulation. [S.l.]: Springer, 2019. P. 215–252.

STROUSTRUP, Bjarne. What is object-oriented programming? **IEEE software**, IEEE, v. 5, n. 3, p. 10–20, 1988.

VARGA, Andras. OMNeT++. In: MODELING and tools for network simulation. [S.l.]: Springer, 2010. P. 35–59.

YANG, Yue; HAO, Si-Yuan; CAI, Ha-Bin. Comparison and Evaluation of Routing Protocols Based on a Collaborative Simulation Using SUMO and NS3 with TraCI. In: 2016 International Conference on Information System and Artificial Intelligence (ISAI). [S.l.: s.n.], 2016. P. 253–257. DOI: 10.1109/ISAI.2016.0061.