WAVE - Um gerador de cargas múltiplas para experimentação em redes de computadores

Leandro C. de Almeida[1,2], **Jefferson L. F. da Silva**[1], **Ricardo P. Lins**[1], **Paulo Ditarso Maciel Jr.**[1], **Rafael Pasquini**[3], **Fábio L. Verdi**[2]

[1]Unidade Acadêmica de Informática – Instituto Federal da Paraíba (IFPB)
João Pessoa – PB – Brasil

[2]Departamento de Computação (Dcomp) – Universidade Federal de São Carlos (UFSCar)
Sorocaba – SP – Brasil.

[3]Faculdade de Computação (FACOM) – Universidade Federal de Uberlândia (UFU)
Uberlância, MG – Brasil

leandro.almeida@ifpb.edu.br, ferreira.jefferson@academico.ifpb.edu.br,

ricardo.lins@academico.ifpb.edu.br, paulo.maciel@ifpb.edu.br,

rafael.pasquini@ufu.br, verdi@ufscar.br

Abstract. *Experimentation is an essential step for many scientific types of research since it enables the researcher to observe the validity of his hypotheses. In the context of computer networks, during the experimentation phase, it is challenging to find load generators that can model traffic patterns for the most diverse types of applications. In this sense, this work presents WAVE - Workload Assay for Verified Experiments. Being application agnostic, WAVE has been widely performed in load tests for video-on-demand and key-value store applications. In the current version, WAVE can generate loads for two distinct patterns: sinusoid and flashcrowd.*

Resumo. *A experimentação é uma etapa imprescindível para muitos tipos de pesquisas científicas, visto que a partir dela o pesquisador pode observar a validade de suas hipóteses. No contexto de redes de computadores, durante a fase de experimentação, é desafiador encontrar geradores de carga que sejam capazes de modelar padrões de tráfego para os mais diversos tipos de aplicações. Neste sentido, este trabalho apresenta o WAVE - uma carga de trabalho para experimentos verificáveis. Tendo como característica ser agnóstico à aplicação, o WAVE foi amplamente executado em testes de carga para aplicações de vídeo sob demanda e armazenamento de chave-valor. Na versão atual, o WAVE é capaz de gerar cargas para dois padrões distintos: sinusoidal e flashcrowd.*

1. Introdução

No contexto de redes de computadores, experimentos científicos são geralmente realizados com o objetivo de avaliar uma hipótese. Tradicionalmente, estes experimentos podem ser conduzidos de diversas formas, tais como: simulação, estudo de caso, validação, prova de conceito, entre outras. Em alguns casos, faz-se necessário aplicar carga na rede para avaliar a robustez da proposta em condições de esgotamento ou saturação dos recursos.

Na literatura de redes observa-se que pesquisadores constroem ambientes de experimentação para retratar de maneira controlada o objeto de pesquisa. Uma carga alta, tipicamente caracterizada por um pico, pode afetar o estado da rede e, consequentemente, a qualidade de serviço das aplicações que estão em execução nesta infraestrutura. A partir da mudança no estado da rede, o pesquisador é capaz de fazer análises e considerações acerca da hipótese proposta (p.ex. aceitar ou refutar) [Fernandes 2017].

No entanto, encontrar uma carga que seja mais próxima da realidade do objeto de estudo, sobretudo ajustada ao ambiente de experimentação, não é uma tarefa trivial. Isso se dá ao fato de que, no mundo real, as aplicações possuem padrões de uso e, certamente, padrões de carga distintos. Adicionalmente, a localização da análise de uma certa carga (*datacenter*, borda da rede, Internet etc.) pode apresentar características diferenciadas de tráfego. Em redes *ethernet*, por exemplo, o tráfego pode ser comumente caracterizado como rajadas (*bursts*) auto-similares [Leland et al. 1993]. Já a carga gerada pelas aplicações em um *datacenter* típico possui uma característica de tráfego conhecido como liga-desliga (*ON-OFF*) [Benson et al. 2010]. Por outro lado, em uma rede móvel (*mobile broadband*), o tráfego de vídeo possui uma caracterização mista (rajadas e estrangulamento) a partir de múltiplos fatores [Horvath and Fazekas 2015].

Devido a esta pluralidade nas características de cargas e tráfegos, os pesquisadores tendem a reduzir o escopo de avaliação da carga aplicada em seus experimentos. Em alguns casos, geradores de tráfego sintético (ex.: IPerf [Miranda et al. 2022], EROS-5 [Soares et al. 2020], P4STA [Kundel et al. 2020]) são utilizados para avaliar o objeto de pesquisa em condições de esgotamento de recursos [Kundel et al. 2018, Gombos et al. 2022, Feibish et al. 2022]. Entretanto, é desafiador para um gerador de carga sintética representar as características intrínsecas de uma aplicação real.

Neste sentido, este trabalho propõe o gerador de carga de trabalho para experimentos verificáveis WAVE (*Workload Assay for Verified Experiments*). Diferente de geradores sintéticos, o WAVE se posiciona no contexto de gerador de carga real, ou seja, controla a execução de instâncias de uma determinada aplicação no tempo, de acordo com um modelo matemático de carga pré-definido. Na versão atual, o WAVE possui suporte aos modelos de geração de carga *sinusoid* e *flashcrowd*. Com código[1] disponibilizado publicamente, o WAVE foi executado em vários experimentos de vídeo sob demanda e armazenamento de chave-valor [Stadler et al. 2017, Almeida et al. 2021, de Almeida et al. 2021]. Além disso, o manual do usuário[2] contendo uma descrição detalhada da instalação e uso do WAVE também está disponível.

O restante deste trabalho está organizado conforme descrito a seguir. A Seção 2 apresenta os modelos matemáticos para geração de carga suportados pelo WAVE na sua versão atual. A arquitetura, módulos e funcionalidades do WAVE estão descritos na Seção 3. Uma demonstração de execução do WAVE está documentada na Seção 4. Por fim, as considerações finais e perspectivas de trabalhos futuros são encontrados na Seção 5.

2. Modelos de carga

No escopo deste trabalho, entende-se como modelo de carga uma função matemática que mapeia o controle de instâncias de uma determinada aplicação no tempo. Cada modelo de

[1] `https://github.com/ifpb/wave`

[2] `https://github.com/ifpb/wave/blob/main/Manual_do_Usuario_do_WAVE.pdf`

carga possui um conjunto de parâmetros necessários para gerar as funções matemáticas. O WAVE recebe os parâmetros de entrada de cada modelo de carga e realiza todas as computações necessárias para o controle das instâncias (carga) durante a execução do experimento. Na versão atual, o WAVE suporta os modelos de carga *sinusoid* e *flashcrowd*.

2.1. *Sinusoid*

O *sinusoid* é um modelo matemático da função seno que resulta em um comportamento periódico de carga. Na Equação 1, pode-se observar que a função $f(y)$ deve receber os seguintes parâmetros como entrada: A (amplitude), F (frequência) e λ (lambda).

$$f(y) = A\sin(F + \lambda) \tag{1}$$

Conforme pode ser observado na Figura 1, a linha contínua em azul representa o número de instâncias de uma determinada aplicação no tempo. A linha tracejada na cor preta representa o número inicial de instâncias. A carga máxima, por sua vez, é representada pela linha tracejada na cor vermelha. Já a linha tracejada verde representa a carga mínima do experimento, ou seja, o menor número de instâncias de uma aplicação durante o experimento.

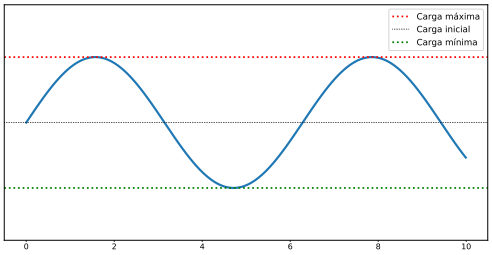

Figura 1. Representação gráfica de uma onda Sinusoid.

No WAVE, a carga gerada pelo modelo *sinusoid* produz requisições de acordo com um processo *Poisson* para aberturas de novas instâncias de aplicação.

2.2. *Flashcrowd*

O termo *flashcrowd* refere-se a descrição de um evento *flash*, no qual uma grande quantidade de tráfego é direcionada para um determinado site [Ari et al. 2003]. O comportamento de uma onda *flashcrowd* pode ser compreendido em três etapas: $i)$ subida da rampa (*ramp-up*); $ii)$ platô (*sustained*); $iii)$ descida da rampa (*ramp-down*). Neste sentido, o modelo matemático que descreve o comportamento de uma carga *flashcrowd* é composto por três equações: *ramp-up*, *sustained* e *ramp-down*.

$$rampup = \frac{1}{\log_{10}(1 + S)} \tag{2}$$

$$sustained = \log_{10}(1 + S) \tag{3}$$

$$rampdown = n \times \log_{10}(1 + S) \tag{4}$$

O parâmetro S (*Shock level*) utilizado nas equações 2, 3 e 4 representa o aumento da ordem de grandeza na taxa média de instâncias durante um evento *flash* [Ari et al. 2003]. O parâmetro n utilizado na equação 4 representa a constante de descida da rampa, ou diminuição da carga no tempo.

A Figura 2 ilustra o comportamento de uma onda *flashcrowd*, onde a linha tracejada na cor verde representa a carga média normal ($Rnorm$) do experimento. A carga máxima ($Rflash$) é representada pela linha tracejada na cor vermelha. A etapa de subida da rampa está representada pela linha contínua azul entre os tempos t_0 e t_1. O intervalo entre t_1 e t_2 representa o platô. Por fim, a descida da rampa acontece entre t_2 e t_3.

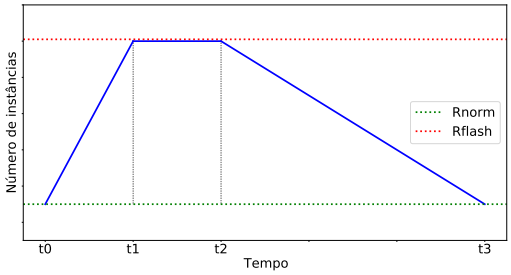

Figura 2. Representação gráfica de uma onda Flashcrowd.

3. Arquitetura do WAVE

Esta seção descreve em detalhes dos módulos funcionais e respectivos componentes tecnológicos que estão inseridos na arquitetura do WAVE. Na versão atual, o WAVE possui quatro módulos funcionais: Inicialização, WAVE Web, Provisionamento e Monitoramento, conforme pode ser visto na Figura 3. Os dois primeiros módulos, de Inicialização e WAVE Web, compõem o *frontend*, ou seja, são os componentes que possuem interação com o usuário. Já os módulos de Provisionamento e Monitoramento estão posicionados no *backend*, ou seja, não possuem interação direta com o usuário da ferramenta.

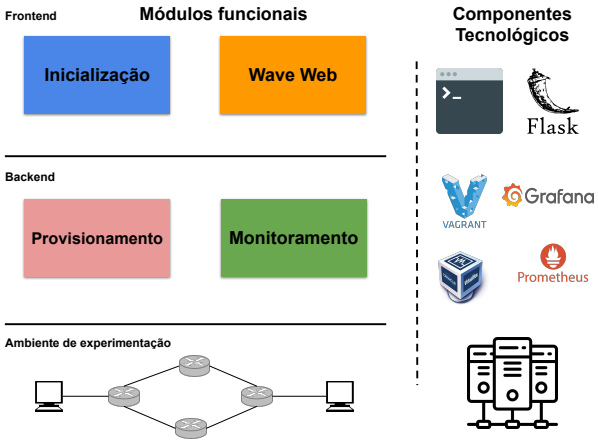

Figura 3. Componentes da arquitetura do WAVE.

O módulo de Inicialização é o componente responsável pela preparação do ambiente, ajustando os componentes necessários para a execução da ferramenta. Uma das principais responsabilidades do módulo de Inicialização é instanciar todos os outros módulos funcionais. A execução deste componente se dá através de um *script shell* (componente tecnológico) que deve ser acionado em interface de linha de comando.

O módulo WAVE Web é desenvolvido em Flask[3] (componente tecnológico) e permite a interação do usuário com a ferramenta, ou seja, definição dos parâmetros necessários para a execução do ambiente de experimentação e geração da carga de trabalho.

[3] `http://flask.pocoo.org/`

Além disso, o WAVE Web possui uma API de comunicação com os módulos de Provisionamento e Monitoramento. A API de comunicação com o módulo de Provisionamento permite que o WAVE Web envie os parâmetros informados pelo usuário para a construção e execução do ambiente de experimentação e carga de trabalho. Para o módulo de Monitoramento, a API de comunicação é utilizada para resgatar as medições coletadas durante a execução e apresentar os resultados graficamente na interface Web.

O módulo de Provisionamento recebe do WAVE Web os parâmetros necessários para a criação do ambiente de experimentação, bem como para a execução das instâncias que devem obedecer ao modelo de carga definido pelo usuário. O Vagrant[4] e o Virtualbox[5] são os componentes tecnológicos suportados atualmente pelo WAVE.

As medições no ambiente de experimentação são realizadas pelo módulo de Monitoramento. O número de instâncias da aplicação em uso inicializadas no tempo e a taxa de bytes recebidos na interface de rede da máquina virtual são, a princípio, as métricas previamente configuradas e exibidas através do WAVE Web. Contudo, devido ao ambiente facilmente ajustável através de seus componentes de software, a coleta de outras métricas desejadas é uma questão de configuração. A materialização do módulo de Monitoramento se dá através dos componentes tecnológicos, Prometheus[6] e Grafana[7].

4. Demonstração

A demonstração das funcionalidades do WAVE compreende a cópia dos arquivos do repositório público e a execução dos *scprits* responsáveis por instalar e provisionar o ambiente necessário à execução dos experimentos. Para tanto, os requisitos necessários na máquina (com SO Linux) onde será executado o ambiente são: o comando *Git*, o *Virtualbox* (versão 6 ou superior), o *Vagrant* (versão 2.2 ou superior), o *Python3* (incluindo *venv*), o *Docker* (versão 23) e *Docker Compose* (versão 2.16). De maneira geral, o fluxo de interação com o WAVE corresponde a preencher as configurações desejadas em um formulário Web (implementado com *Flask* em um ambiente virtual *Python*) e acionar a API de provisionamento do ambiente (acionar o *Vagrant* por meio de uma API REST). A Figura 4 apresenta o fluxo de interação entre os módulos WAVE Web e Provisionamento, para a criação das VMs especificadas via navegador do usuário. O acesso à interface Web pode ser feito local ou remotamente, caso a rede esteja devidamente configurada.

Inicialmente, os arquivos disponibilizados no repositório público são copiados com um comando `git clone`. Uma vez copiados os arquivos, utiliza-se a ferramenta `docker-compose` para criar e executar o ambiente de contêineres necessário, que esta definido no arquivos `docker-compose.yml` e `Dockerfile`. Estas configurações são consideradas parte do módulo de Inicialização do WAVE, acionado a partir do *script* `app-compose.sh`. Este último é responsável por configurar uma variável de ambiente com endereço IP da máquina, além de inicializar ou destruir o ambiente, respectivamente com os parâmetros passados `--start` ou `--destroy`. A sequência de comandos abaixo ilustra a inicialização do WAVE, que resulta no levantamento dos módulos responsáveis pela interface Web e API de provisionamento.

[4] https://www.vagrantup.com/
[5] https://www.virtualbox.org/
[6] https://prometheus.io/
[7] https://grafana.com/

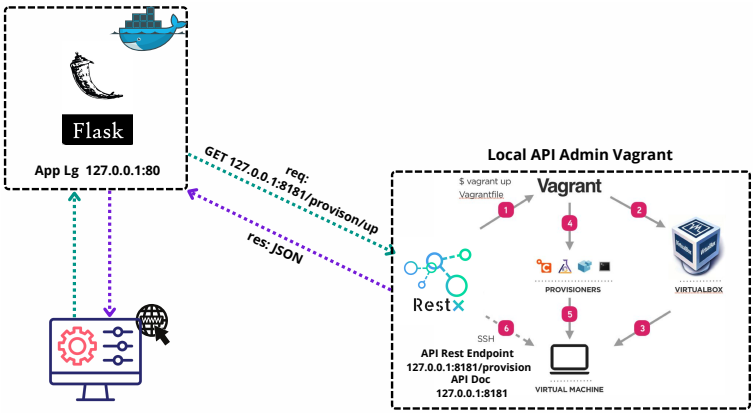

Figura 4. Fluxo de interação entre os módulos do WAVE para criação das VMs.

```
1 $ git clone https://github.com/ifpb/wave.git
2 $ cd wave
3 $ ./app-compose.sh --start
```

A Figura 5 demonstra a execução do ambiente de contêineres que serve de infraestrutura para os módulos WAVE Web, Provisionamento e Monitoramento. O lado esquerdo da figura ilustra a saída da execução do *script* de inicialização do WAVE (linha 3 do quadro anterior). Como consequência, a orquestração de dois contêineres é direcionado através do `docker-compose`, como pode ser visto na saída exibida ao lado direito da figura. Pelas imagens é possível confirmar a execução atual na máquina que, por sua vez, faz uso de duas imagens para levantar o ambiente. Um dos contêineres é responsável pela ferramenta *Grafana*, que faz parte do módulo de Monitoramento e exibe em tempo real as informações dos bytes recebidos na interface de rede da VM que recebe a carga de tráfego, denominada VM Servidor. Também fazem parte deste último módulo as ferramentas *Prometheus* e *node_exporter*, ambas configuradas na VM Servidor. Por outro lado, a VM Cliente é responsável por gerar carga de trabalho a partir dos parâmetros inseridos pelo usuário. Para os resultados apresentados neste documento, a aplicação utilizada para gerar tráfego foi o IPerf[8], onde múltiplas instâncias do mesmo são inicializadas e finalizadas conforme o modelo de carga selecionado (*sinusoid* ou *flashcrowd*).

A Figura 6 ilustra o funcionamento do módulo WAVE WEB. A imagem a esquerda da figura demonstra o formulário da página inicial, onde são inseridos os dados para configuração do ambiente. A imagem ao meio da figura mostra uma página seguinte, confirmando as informações inseridas no formulário. Estas informações são carregadas em um arquivo `config.yml`, que servirá de entrada para o módulo de Provisionamento configurar e subir as VMs (Cliente e Servidor). Além disso, também configurado a partir do formulário, o modelo de carga de trabalho será automaticamente iniciado entre as duas VMs, gerando o tráfego de rede como capturado pela imagem mais a direita da figura.

O vídeo[9] contendo uma descrição prévia da demonstração foi disponibilizado publicamente. Para a demonstração *in-loco*, serão necessários os seguintes itens: computa-

[8] `https://iperf.fr/`
[9] `https://youtu.be/AOsvDJgxGQ8`

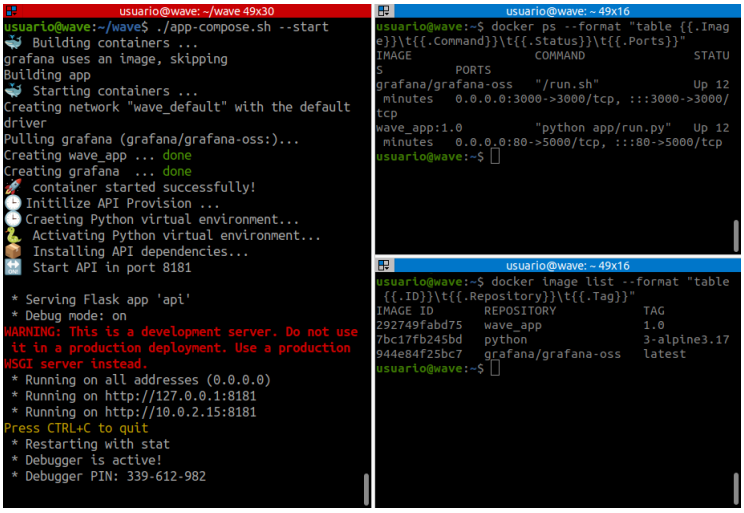

Figura 5. Ambiente de contêineres provisionado.

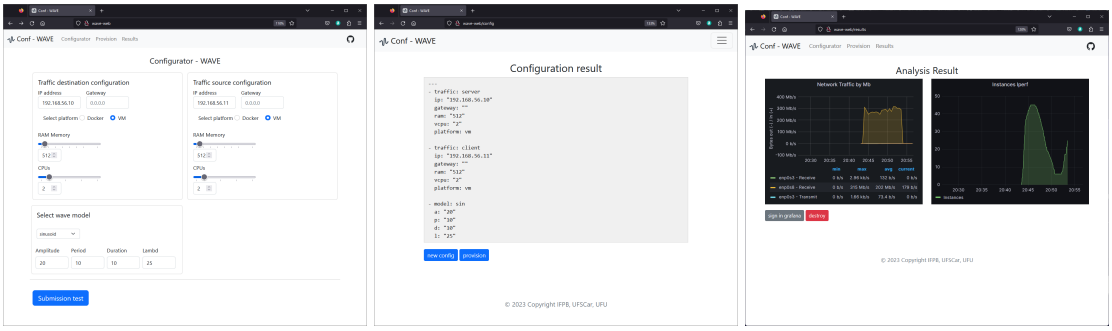

Figura 6. Páginas Web do formulário de entrada e do arquivo YAML produzido.

dor para execução da demonstração do WAVE, tela para projeção e conectividade com à Internet. Durante o salão de ferramentas, os participantes do simpósio poderão observar o processo de instalação e execução do WAVE. Além disso, o módulo de monitoramento exibirá as medições coletadas por meio de gráficos contendo o número de *bytes* enviados e recebidos pela interface de rede da VM Servidor, bem como o número de instâncias de aplicação executadas no experimento a partir da VM Cliente.

5. Considerações Finais

A definição da carga em um experimento científico é crucial para uma análise adequada dos resultados obtidos. No contexto das pesquisas em redes de computadores, geradores de tráfego sintético são comumente utilizados. Entretanto, estes não conseguem retratar com exatidão os comportamentos de aplicações reais. Portanto, o WAVE se posiciona como um gerador de carga de trabalho que controla a execução de instâncias de aplicações no tempo e, consequentemente, o comportamento do tráfego gerado por estas aplicações. Como propostas de extensão do WAVE, pretende-se disponibilizar: o uso de contêineres para geração e recebimento do tráfego (além de VMs); um modelo de carga de micro-rajadas; e outros tipos de aplicações reais (como p.ex., vídeo sob demanda com o VLC).

Referências

[Almeida et al. 2021] Almeida, L., Verdi, F., and Pasquini, R. (2021). Estimando métricas de serviço através de in-band network telemetry. In *Anais do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 252–265, Porto Alegre, RS, Brasil. SBC.

[Ari et al. 2003] Ari, I., Hong, B., Miller, E., Brandt, S., and Long, D. (2003). Managing flash crowds on the internet. In *MASCOTS 2003*, pages 246– 249.

[Benson et al. 2010] Benson, T. et al. (2010). Understanding data center traffic characteristics. *SIGCOMM Comput. Commun. Rev.*, 40(1):92–99.

[de Almeida et al. 2021] de Almeida, L. C., Pasquini, R., and Verdi, F. L. (2021). Using machine learning and in-band network telemetry for service metrics estimation. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 33–39.

[Feibish et al. 2022] Feibish, S. L., Liu, Z., Ivkin, N., Chen, X., Braverman, V., and Rexford, J. (2022). Flow-level loss detection with sketches. In *Proceedings of the Symposium on SDN Research*, SOSR '22, page 25–32, New York, NY, USA. ACM.

[Fernandes 2017] Fernandes, S. (2017). *Performance Evaluation for Network Services, Systems and Protocols*. Springer.

[Gombos et al. 2022] Gombos, G., Mouw, M., Laki, S., Papagianni, C., and De Schepper, K. (2022). Active queue management on the tofino programmable switch: The (dual)pi2 case. In *IEEE Int. Conference on Communications (ICC)*, pages 1685–1691.

[Horvath and Fazekas 2015] Horvath, G. and Fazekas, P. (2015). Modelling of youtube traffic in high speed mobile networks. In *Proceedings of the 21th European Wireless Conference*, pages 1–6.

[Kundel et al. 2018] Kundel, R. et al. (2018). P4-CoDel: Active Queue Management in Programmable Data Planes. In *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–4.

[Kundel et al. 2020] Kundel, R. et al. (2020). P4STA: High Performance Packet Timestamping with Programmable Packet Processors. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9.

[Leland et al. 1993] Leland, W. E. et al. (1993). On the self-similar nature of ethernet traffic. In *Conf. Proceedings on Communications Architectures, Protocols and Applications*, SIGCOMM '93, page 183–193, New York, NY, USA. ACM.

[Miranda et al. 2022] Miranda, G. et al. (2022). Evaluating time-sensitive networking features on open testbeds. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–2.

[Soares et al. 2020] Soares, R. et al. (2020). EROS-5: Gerador de Tráfego Sintético para Redes 5G. In *Anais Estendidos do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 41–48, Porto Alegre, RS, Brasil. SBC.

[Stadler et al. 2017] Stadler, R., Pasquini, R., and Fodor, V. (2017). Learning from network device statistics. *J. Netw. Syst. Manag.*, 25(4):672–698.