A New WAVE: Exploring New Load Pattern Models for Experimentation in Computer Networks

Danilo C. Beuttenmuller, Matheus F. de A. Valério, Caio Luiz L. T. Silva, Icaro M. da Silva, Paulo Ditarso Maciel Jr., Leandro C. de Almeida

[1]Academic Unit of Informatics – Federal Institute of Paraíba (IFPB)
João Pessoa – PB – Brasil

{danilo.cavalcante, matheus.faelson}@academico.ifpb.edu.br,

{caio.luiz, icaro.silva}@academico.ifpb.edu.br,

{paulo.maciel,leandro.almeida}@ifpb.edu.br

Abstract. *Experimentation is a crucial step in many types of scientific research, enabling researchers to evaluate the validity of their hypotheses. In computer networks, one of the key challenges during the experimentation phase is finding load generators capable of accurately modeling diverse traffic patterns for various applications. To address this issue, our previous work introduced a load generator designed to generate load based on real application behavior. In this sense, this work improves the WAVE - Workload Assay for Verified Experiments. A new WAVE version can generate loads for three distinct patterns: sinusoid, flashcrowd, and step. Additionally, it now supports microbursts and container-based environments.*

1. Introduction

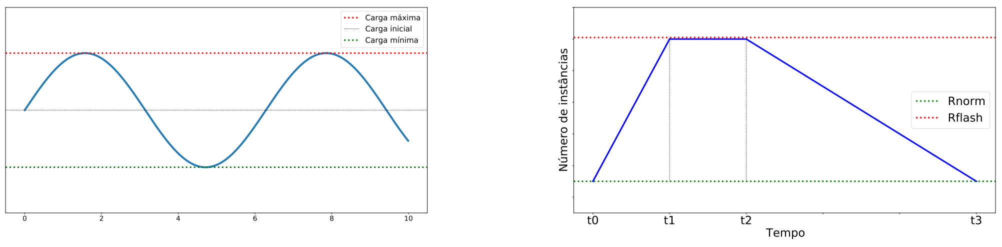
In our previous work, we introduced WAVE (Workload Assay for Verified Experiments) [Almeida et al. 2023], a load generator designed for experimentation in computer networks. Unlike conventional synthetic traffic generators, such as IPERF [Miranda et al. 2022], EROS-5 [Soares et al. 2020], and P4STA [Kundel et al. 2020], WAVE has the unique capability of generating traffic based on real applications while adhering to a load pattern defined by a mathematical equation. In other words, it can control the execution of instances of a given application over time, according to a pre-defined mathematical load model. This approach enhances the realism of network experiments by more accurately replicating real-world application behaviors.

In its initial version, WAVE had limitations that restricted its execution to virtual machine environments. For the new version, we have introduced support for container-based execution, enabling greater flexibility in generating and receiving experimental traffic. Additionally, we have expanded the load modeling capabilities by incorporating a new mathematical model, increasing the total to three supported models: sinusoidal, flashcrowd, and Heaviside step (new). Furthermore, this version includes support for microbursts, allowing for more precise emulation of short-lived traffic spikes. Finally, native support for generating video application workloads has been integrated, enhancing the realism of experiments involving multimedia traffic. The WAVE source code is publicly available in a repository[1], which also includes a user manual[2] detailing the installation steps and usage instructions.

The remainder of this paper is organized as follows. Section 2 presents the mathematical models for load generation supported by the new version of WAVE. Section 3 describes the architecture, modules, and functionalities. Use cases illustrating how WAVE can contribute to scientific research are presented in Section 4. The prerequisites for WAVE's demonstration and intended presentations are explained in Section 5.

2. Load Models

In the scope of this work, a load model is defined as a mathematical function that controls application instances over time. Each load model has a set of parameters required to generate the corresponding mathematical functions. WAVE receives the input parameters for each load model and performs all necessary computations to control the instances (load) during the experiment execution. In its previous version, WAVE supported the sinusoid (Figure 1(a)) [Stadler et al. 2017] and flashcrowd (Figure 1(b)) [Ari et al. 2003] load models; and now, we added the Heaviside step and microburst ones.



(a) Sinusoid (b) Flashcrowd

Figure 1. Load models supported since the initial release of WAVE.

2.1. Discrete Step Function

A discrete step function is a mathematical function that exhibits a piecewise constant behavior, changing its value abruptly at specific points. It is commonly used to model systems where state transitions occur at discrete intervals. In the context of a network, a discrete step function can represent variations in network load over time. For instance, in a traffic analysis scenario, the number of active network connections or the data transmission rate can be modeled using step functions, where each step corresponds to a sudden increase or decrease in traffic due to new user connections or service demands. In WAVE, the step load is represented by Equation 1 and Figure 2 illustrates an instantiated example.

$$f(x) = \sum_{i=1}^{n} a_i H(x - x_i) \tag{1}$$

[1]`https://github.com/ifpb/new_wave`
[2]`https://github.com/ifpb/new_wave/blob/main/WAVE_User_Manual.pdf`

where:

- $H(x - x_i)$ is the Heaviside step function,
- x_i are the step positions along the x-axis,
- a_i are the step heights at each x_i,
- n defines the total number of steps in the function.

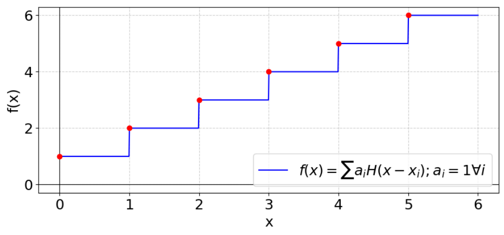
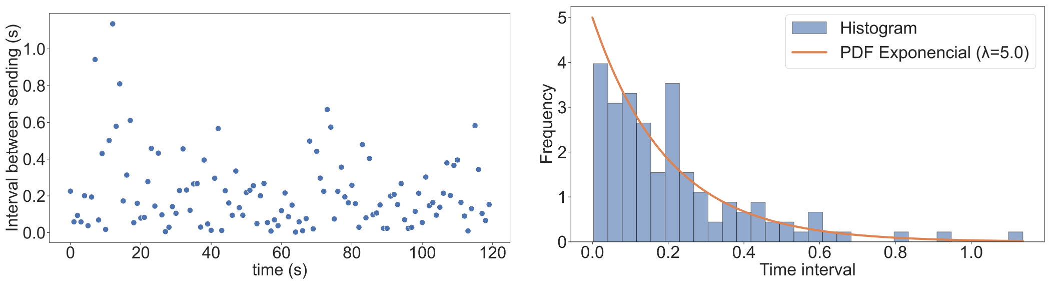


Figure 2. Discrete step function example.

2.2. Microburst

Short-lived, high-intensity transient congestion events, known as microbursts [Woodruff et al. 2020], can significantly impact the performance of various applications. However, their brief duration poses a challenge for conventional monitoring systems, which often fail to detect and characterize them effectively. Seeking to get closer to the characteristics of real traffic, the latest version of WAVE incorporates a dedicated microburst generator.

We use Python and Scapy[3] library to generate and send bursts of network packets, simulating microburst conditions. Our solution allows researchers to customize Ethernet packets with specific source and destination IP addresses and randomly generated payloads. The script sends these packets in controlled bursts with a researcher-defined interval, logging timestamps to facilitate performance analysis. Moreover, our tool is useful for network performance testing, QoS (Quality of Service) policy development, and traffic engineering research, since it emulates how network devices can handle sudden traffic spikes. Figure 3(a) illustrates the time interval between microbursts following a Poisson arrival process.



(a) Bursts' arrival times (b) Exponentially distributed inter-arrival times

Figure 3. The time interval between microbursts being sent follows a Poisson distribution.

In our microburst generator, the time interval between consecutive sends is determined using an exponentially distributed random variable, as seen in Figure 3(b). This ensures that the occurrence of microbursts is statistically consistent with real-world network traffic, where events happen randomly over time rather than at fixed or uniformly distributed intervals.

[3]`https://scapy.net/`

By using WAVE, researchers can incorporate microbursts into experiments alongside various load patterns (sinusoidal, flashcrowd, and step), introducing controlled noise into the experimental environment. These conditions aim to replicate the unpredictable nature of real-world network traffic, enhancing the realism of performance evaluations.

3. WAVE architecture

This section provides a detailed description of the modules and their technological components within the WAVE architecture. In its new version, WAVE consists of four functional modules: Initialization, Web, Provisioning, and Monitoring, as illustrated in Figure 4. The first two modules, Initialization and Web, form the frontend; meaning they are the components the researcher interacts with. In contrast, the Provisioning and Monitoring modules are positioned in the backend; meaning that they are performed without direct intervention by the researcher. They are started by the frontend Web module, based on the parameters introduced by the researcher.

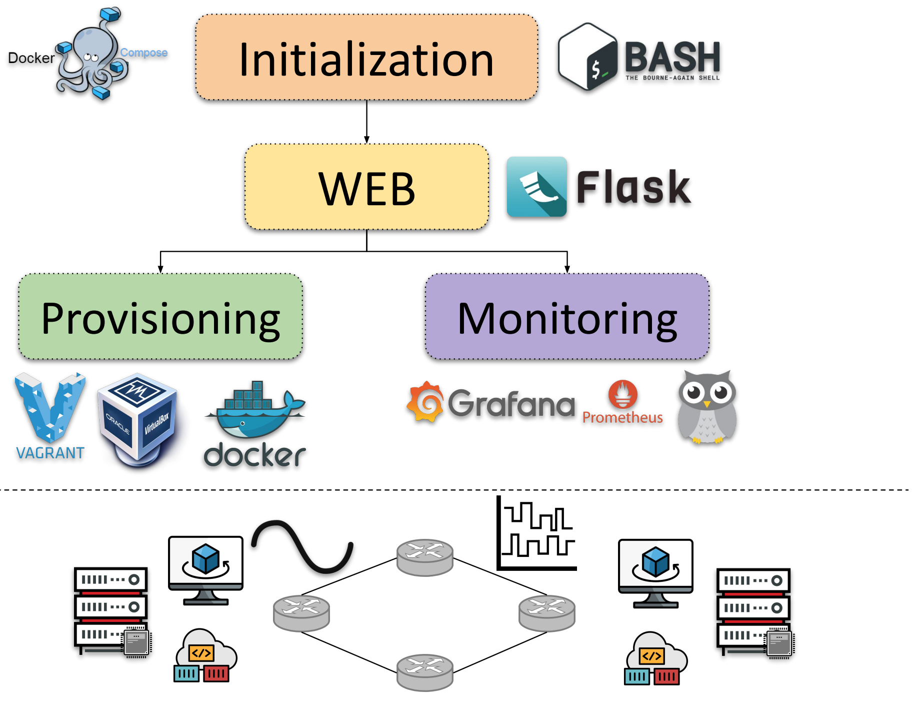


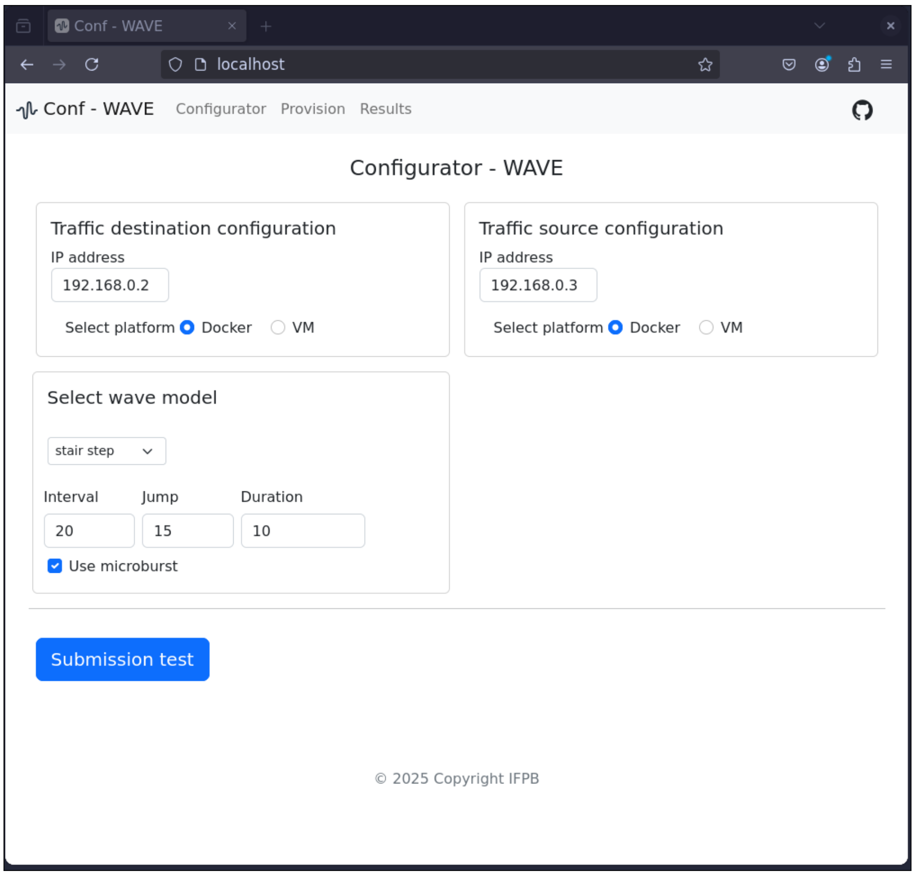
Figure 4. The WAVE architecture and its technological components.

The Initialization module sets up the environment by configuring the necessary components for execution. Its primary responsibility is to instantiate all other functional modules. This module is executed via a Docker Compose[4] which must be triggered from the command-line interface (Bash script).

As a technological component, the Web module is developed using the Flask[5] framework. It enables researcher interaction and allows the definition of necessary parameters for the execution of the experimentation environment and workload generation, as can be seen in Figure 5. Additionally, WAVE Web includes a communication API that

[4]`https://docs.docker.com/compose/`
[5]`https://flask.palletsprojects.com/`

interacts with the Provisioning and Monitoring modules. The communication API with the Provisioning module allows WAVE Web to transmit researcher-defined parameters for the setup and execution of the experimentation environment and workload. For the Monitoring module, the communication API is used to retrieve collected measurements during execution and display the results graphically on the Web interface.



Configurator - WAVE

Traffic destination configuration

IP address: 192.168.0.2

Select platform [✓ Docker] VM

Traffic source configuration

IP address: 192.168.0.3

Select platform [✓ Docker] VM

Select wave model

stair step

Interval	Jump	Duration
20	15	10

[✓ Use microburst]

© 2025 Copyright IFPB

Figure 5. WAVE Web module receiving input parameters.

The Provisioning module receives the necessary parameters from WAVE Web to create the experimentation environment and execute instances that adhere to the researcher-defined workload model. Currently, WAVE supports Vagrant[6], VirtualBox[7], and Docker[8] as the underlying technological components for provisioning. For the latter, Docker Compose is employed to launch the containers automatically, serving as an orchestration tool and enabling the management of multi-container applications. In this new version, the client and server components can be provisioned within isolated containers, in addition to virtual machines. This approach not only simplifies the orchestration of these services, but also ensures rapid deployment and scalability. Below is the snippet of the docker-compose.yaml file responsible for provisioning the environment.

[6]https://www.vagrantup.com/
[7]https://www.virtualbox.org/
[8]https://www.docker.com/

```
services:
  apache:
    image: ghcr.io/ifpb/new_wave/wave-apache
    container_name: server
    #ports:
    #- "80:80"

  client_container:
    image: ghcr.io/ifpb/new_wave/wave-vlc
    container_name: client
    privileged: true
    #environment:
      #- DISPLAY=${DISPLAY}
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - ./logs:/home/vlc/logs
      - /tmp/.X11-unix:/tmp/.X11-unix
    depends_on:
      - apache
```

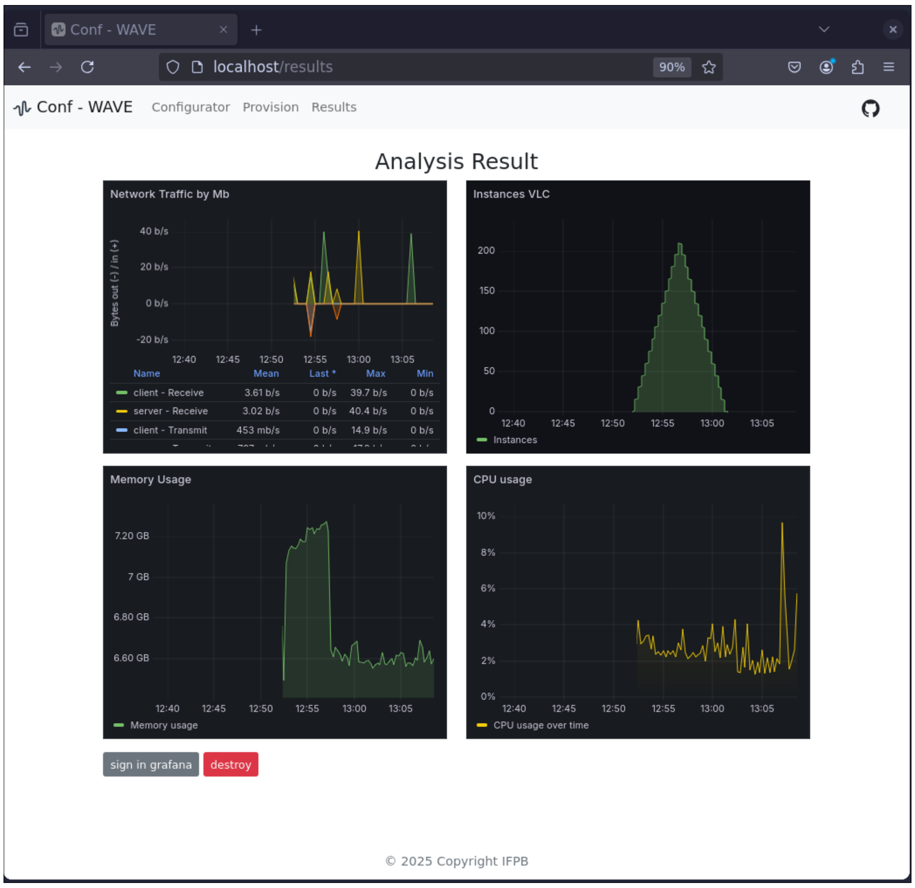
The Monitoring module is responsible for performing measurements within the experimentation environment. By default, the number of active application instances over time and the rate of bytes received on the network interface of virtual components are the preconfigured metrics displayed through WAVE Web, as can be seen in Figure 6. Moreover, additional metrics can be collected as needed with minimal configuration adjustments. The Monitoring module is implemented using the following technological components: Prometheus[9], Grafana[10] and cAdvisor[11].

In this section, we provided a detailed overview of WAVE's architecture, outlining its core modules and the key technological components that support its functionality. Each module was described in terms of its role within the system, highlighting how they interact to facilitate workload generation and network performance analysis. Additionally, the technological stack was carefully selected to ensure scalability, efficiency, and ease of integration with existing network infrastructures. With this structured design, WAVE offers a robust and flexible solution for researchers and practitioners seeking to simulate and analyze diverse networking scenarios.

4. Use Cases

In the context of workload tools for computer networks, defining clear use cases is essential to ensure the applicability and relevance of the proposed solution. By outlining

[9]https://prometheus.io/
[10]https://grafana.com/
[11]https://github.com/google/cadvisor



	Name	Mean	Last *	Max	Min
🟩	client - Receive	3.61 b/s	0 b/s	39.7 b/s	0 b/s
🟨	server - Receive	3.02 b/s	0 b/s	40.4 b/s	0 b/s
🟦	client - Transmit	453 mb/s	0 b/s	14.9 b/s	0 b/s

Figure 6. Monitoring module displaying network traffic, the number of video player instances, and resource usage.

different scenarios in which WAVE can be employed, researchers can better assess its effectiveness and limitations. Moreover, well-defined use cases facilitate reproducibility, enabling other researchers to validate findings and compare results across different network environments. This structured approach not only strengthens the scientific rigor of studies in the field but also fosters innovation by identifying potential improvements and new research directions. In this section, we will describe two use cases in which WAVE can be used to create controlled noise in an experimental environment. First, we examine bursts of incast traffic in a DC (Data Center) context, followed by a video streaming scenario within a CDN (Content Delivery Network) context.

4.1. Incast traffic bursts

Incast burst traffic is common in distributed applications, particularly those that rely on data aggregation architectures. In this sense, it has attracted the interest of researchers and industry in the context of data centers [Chen and other 2018, Canel et al. 2024].

A typical use case occurs in distributed storage systems, where multiple storage nodes simultaneously transmit data blocks to an aggregator node. This communication pattern can lead to congestion at the ToR (top-of-rack) switch, causing packet loss and increased transmission latency, negatively impacting overall system performance. Another scenario involving incast traffic arises in distributed machine learning training, where multiple GPU (Graphics Processing Unit) servers exchange gradient updates with a centralized parameter server or utilize collective communication algorithms. During these exchanges, numerous GPUs transmit data simultaneously, leading to bursts of high-intensity traffic directed toward a single destination. If the network infrastructure is unable to efficiently handle this communication pattern, synchronization delays among processing nodes can significantly increase, reducing training efficiency.

In its current version, WAVE can generate microbursts following a Poisson arrival process, which characterizes a data center's typical traffic behavior.

4.2. Customer's load in a CDN context (video streaming)

On-demand video streaming prominently emerges as a significant component, currently constituting 60-75% of the total Internet traffic [Sandvine 2023]. This subject has captivated the interest of the scientific community [Lin et al. 2020, Wei et al. 2021, Kim and Chung 2022, Hafez et al. 2023, Spang et al. 2023]. In this context, WAVE is capable of controlling the execution of video player instances, simulating thousands of viewers consuming audiovisual content through a CDN. When a large number of users request the same video content simultaneously, edge servers within the CDN must handle a surge in concurrent connections, generating substantial outbound traffic. This pattern of load generation can lead to bandwidth saturation, increased latency, and potential service degradation if the infrastructure is not adequately provisioned.

Another key aspect of load generation in video streaming within a CDN is the dynamic adaptation to fluctuations in user demand. Events such as live sports broadcasts, product launches, or viral content releases can trigger sudden spikes in traffic, requiring real-time scaling of network resources. In these scenarios, CDNs leverage intelligent traffic routing, dynamic resource allocation, and predictive analytics to optimize server workload distribution and prevent service disruptions.

5. Demonstration

For the *in-loco* demonstration, the following items will be required: a computer for running the new WAVE demonstration, a projection screen, a power strip, and Internet connectivity. A video outlining the usage flow of the new WAVE will be shown throughout the demonstration. During the tool session, symposium participants will be able to observe the process of installing and running WAVE. Additionally, the monitoring module will display the collected measurements through graphs showing the number of *bytes* sent and received by the instances' network interfaces (VMs or containers), as well as the number of application instances executed in the experiment from the Client VM.

We consider that defining the load in a scientific experiment is crucial for an adequate analysis of the obtained results. In the context of computer networks research, synthetic traffic generators are commonly used. However, they fail to accurately depict the behavior of real applications. Therefore, WAVE positions itself as a workload generator that controls the execution of application instances over time and, consequently, the traffic behavior generated by these applications. The key highlights of this WAVE extension include: the use of containers for traffic generation and reception (in addition to VMs); a microburst load model; a load model based on a step function; and the use of a real video application with VLC.

References

[Almeida et al. 2023] Almeida, L., Silva, J., Lins, R., Maciel Jr., P. D., Pasquini, R., and Verdi, F. (2023). WAVE - Um gerador de cargas múltiplas para experimentação em redes de computadores. In *Anais Estendidos do 41^o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 9–16, 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.

[Canel et al. 2024] Canel, C. et al. (2024). Understanding incast bursts in modern datacenters. In *Proceedings of the 2024 ACM on Internet Measurement Conference*, IMC '24, page 674–680, New York, NY, USA. Association for Computing Machinery.

[Chen and other 2018] Chen, X. and other (2018). Catching the microburst culprits with snappy. In *Proceedings of the Afternoon Workshop on Self-Driving Networks*, SelfDN 2018, page 22–28, New York, NY, USA. Association for Computing Machinery.

[Hafez et al. 2023] Hafez, N. A., Hassan, M. S., and Landolsi, T. (2023). Reinforcement learning-based rate adaptation in dynamic video streaming. *Telecommunication Systems*, 83(4):395–407.

[Kim and Chung 2022] Kim, M. and Chung, K. (2022). Reinforcement Learning-Based adaptive streaming scheme with edge computing assistance. *Sensors (Basel)*, 22(6).

[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.

[Lin et al. 2020] Lin, H., Shen, Z., Zhou, H., Liu, X., Zhang, L., Xiao, G., and Cheng, Z. (2020). Knn-q learning algorithm of bitrate adaptation for video streaming over http. In *2020 Information Communication Technologies Conference (ICTC)*, pages 302–306.

[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.

[Sandvine 2023] Sandvine (2023). Global Internet Phenomena. Technical report, Sandvine.

[Soares et al. 2020] Soares, R., Ferreira, G., Solis, P., and Caetano, M. (2020). Eros-5: Gerador de tráfego sintético para redes 5g. In *Anais Estendidos do 38^o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 41–48, Brasil. SBC.

[Spang et al. 2023] Spang, B. et al. (2023). Sammy: Smoothing video traffic to be a friendly internet neighbor. In *Proceedings of the ACM SIGCOMM 2023 Conference*, page 754–768, New York, NY, USA. Association for Computing Machinery.

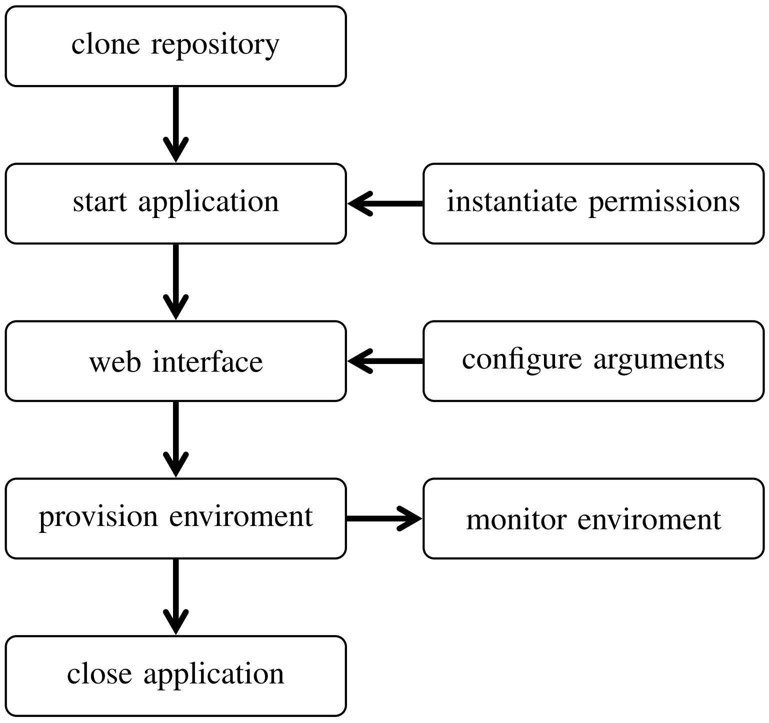
[Stadler et al. 2017] Stadler, R., Pasquini, R., and Fodor, V. (2017). Learning from Network Device Statistics. *Journal of Network and Systems Management*, 25(4):672–698.

[Wei et al. 2021] Wei, X. et al. (2021). Reinforcement learning-based QoE-oriented dynamic adaptive streaming framework. *Information Sciences*, 569:786–803.

[Woodruff et al. 2020] Woodruff, J., Moore, A. W., and Zilberman, N. (2020). Measuring burstiness in data center applications. In *Proceedings of the 2019 Workshop on Buffer Sizing*, BS '19, New York, NY, USA. Association for Computing Machinery.

Appendix: An Excerpt from the User Manual.

Quick Workflow Guide



The WAVE workflow begins with cloning the repository, ensuring that the latest version of the tool is available. Next, the user starts the application, which requires instantiating permissions to configure access control. Once the application is running, it provides a web interface, where users can configure arguments to tailor the workload and network conditions. After configuration, the system proceeds to provision the environment, setting up the necessary resources, while simultaneously enabling environment monitoring to track performance and resource usage. Finally, after the workload execution and analysis, the user closes the application, concluding the process.

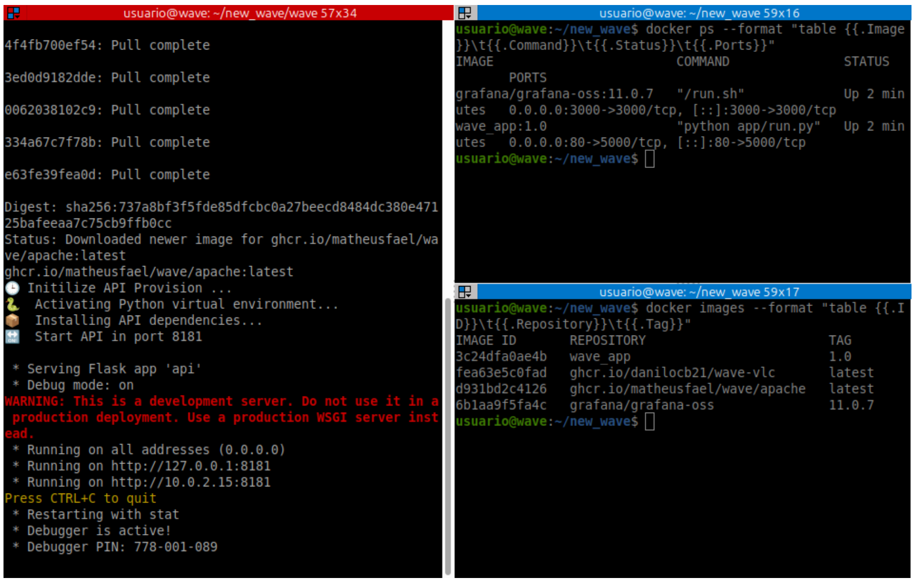
System's Requirements and Starting Environment

The WAVE's functionalities are executed by the *scripts* responsible for installing and provisioning the environment required to run the experiments. To do so, the machine (running a Linux OS) must meet the following requirements (tested versions): *Git* command, *VirtualBox* (version 7), *Vagrant* (version 2.3.4), *Python3* (including *venv*), *Docker* (version 27), and *Docker Compose* (version 2.32.4). In general, the interaction flow with WAVE consists of configuring the desired settings through a web form (implemented with *Flask* in a *Python* virtual environment) and triggering the environment provisioning API (which invokes *Vagrant* or *Docker* via a REST API). The Web interface can be accessed either locally or remotely, provided the network is properly configured.

Initially, the files available in the public repository are copied using the `git clone` command. Once the files are copied, the `docker-compose` tool is used to create and run the required containerized environment, as defined in the `docker-compose.yml` and `Dockerfile` files. These configurations are considered part of WAVE's Initialization module, which is triggered by the *script* app-compose.sh. This script is responsible for setting an environment variable with the machine's IP address and for either initializing or destroying the environment, depending on the provided parameters: --start for initialization or --destroy for termination. The commands snippet below illustrates WAVE's initialization process, which deploys the modules responsible for the Web interface and provisioning API.

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

After cloning the official repository and starting the system, it is possible to check the execution in a **Docker** environment, as illustrated in Figure 7. As can be seen in the figure, the WAVE Initialization module uses two containers for its execution: wave_app and grafana-oss. The left side of the figure shows the WAVE startup command output, which corresponds to the output of line 3 of the command snippet shown above.



```
usuario@wave: ~/new_wave/wave 57x34

4f4fb700ef54: Pull complete

3ed0d9182dde: Pull complete

0062038102c9: Pull complete

334a67c7f78b: Pull complete

e63fe39fea0d: Pull complete

Digest: sha256:737a8bf3f5fde85dfcbc0a27beecd8484dc380e471
25bafeeaa7c75cb9ffb0cc
Status: Downloaded newer image for ghcr.io/matheusfael/wa
ve/apache:latest
ghcr.io/matheusfael/wave/apache:latest
🕒 Initilize API Provision ...
🐍  Activating Python virtual environment...
📦  Installing API dependencies...
🔛  Start API in port 8181

 * Serving Flask app 'api'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a
 production deployment. Use a production WSGI server inst
ead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8181
 * Running on http://10.0.2.15:8181
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 778-001-089
```

```
usuario@wave: ~/new_wave 59x16
usuario@wave:~/new_wave$ docker ps --format "table {{.Image
}}\t{{.Command}}\t{{.Status}}\t{{.Ports}}"
IMAGE                          COMMAND                STATUS
       PORTS
grafana/grafana-oss:11.0.7     "/run.sh"              Up 2 min
utes   0.0.0.0:3000->3000/tcp, [::]:3000->3000/tcp
wave_app:1.0                   "python app/run.py"    Up 2 min
utes   0.0.0.0:80->5000/tcp, [::]:80->5000/tcp
usuario@wave:~/new_wave$
```

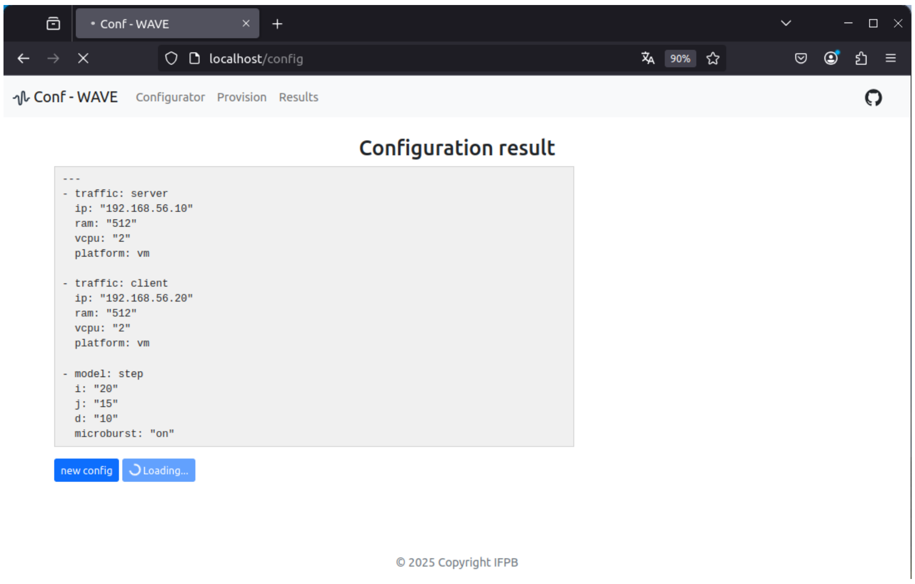
```
usuario@wave: ~/new_wave 59x17
usuario@wave:~/new_wave$ docker images --format "table {{.I
D}}\t{{.Repository}}\t{{.Tag}}"
IMAGE ID       REPOSITORY                          TAG
3c24dfa0ae4b   wave_app                            1.0
fea63e5c0fad   ghcr.io/danilocb21/wave-vlc         latest
d931bd2c4126   ghcr.io/matheusfael/wave/apache     latest
6b1aa9f5fa4c   grafana/grafana-oss                 11.0.7
usuario@wave:~/new_wave$
```

Figure 7. Checking the execution in a Docker environment.

Provisioning and Ending the WAVE's Execution

After entering the arguments in the homepage form (Figure 5), WAVE Web displays a confirmation page for the input data as illustrated in Figure 8. These arguments are edited in a YAML-formatted file called config.yaml. If the user notices any errors in the arguments, they can return to the homepage using the new configuration button or the menu in the top bar of the web page. Once the data has been verified, the user can proceed with the environment provisioning. Upon the provisioning conclusion, the WAVE Web is redirected to the monitoring results page as shown in Figure 6.

At any moment, it's possible to close the containers/virtual machines environment by clicking the **destroy** button on the traffic verification page indicated in Figure 6. The user will be redirected to the homepage as shown in Figure 9, where she can restart



Configuration result

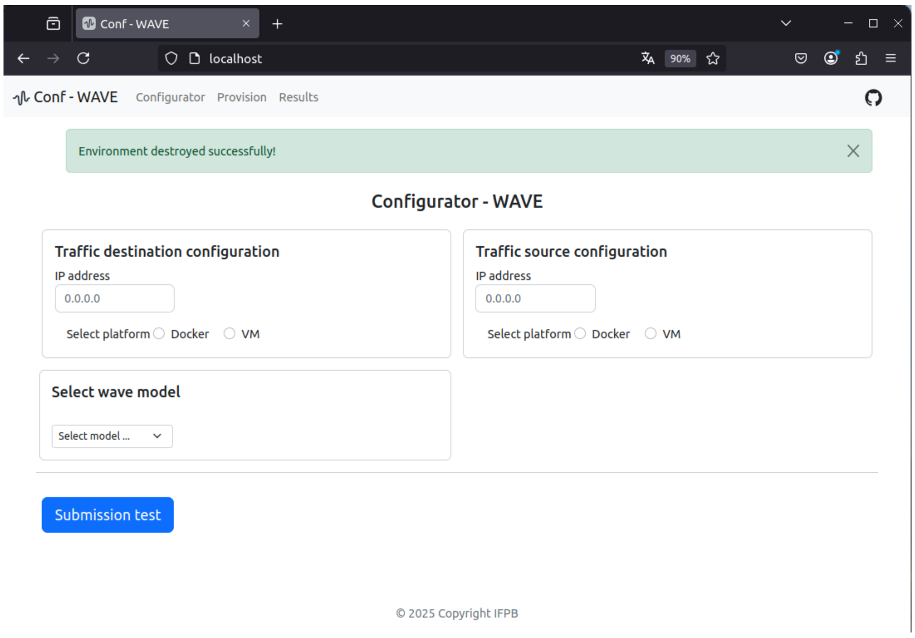
```
---
- traffic: server
  ip: "192.168.56.10"
  ram: "512"
  vcpu: "2"
  platform: vm

- traffic: client
  ip: "192.168.56.20"
  ram: "512"
  vcpu: "2"
  platform: vm

- model: step
  i: "20"
  j: "15"
  d: "10"
  microburst: "on"
```

Figure 8. Confirmation and provisioning page.

the experiment if desired. It is important, tough, to differentiate the virtual environment's decommission (as just described) from the ending of the WAVE platform as stated next.



Environment destroyed successfully!

Configurator - WAVE

Traffic destination configuration
IP address: 0.0.0.0
Select platform ○ Docker ○ VM

Traffic source configuration
IP address: 0.0.0.0
Select platform ○ Docker ○ VM

Select wave model

Figure 9. Virtual environment destroyed successfully.

By running the command below, the user terminates the WAVE Web module and removes the containers responsible for the other initiated modules. To restart the entire system, simply replace the `--destroy` argument with `--start`.

```
$ ./app-compose.sh --destroy
```