LiveOppServation – An Open-Source Interactive Evaluation and Simulation Frontend for OMNeT++

Jannusch Bigge and Christoph Sommer
TU Dresden, Faculty of Computer Science, Germany
https://www.cms-labs.org/people/ { bigge , sommer }

***Abstract*—Modern simulation environments require advanced tools for real-time data visualization and interactive analysis. This paper presents LiveOppServation, an open-source Dash-based frontend for OMNeT++ simulations that enables continuous runtime monitoring and visualization of simulation data. Our system addresses limitations in traditional post-processing approaches by providing three key capabilities: real-time evaluation of vector data with timestamps, automatic generation of live visualizations, and conditional data-based breakpoints for targeted event analysis. Built on gRPC and Protocol Buffers, the architecture supports scalable many-to-many mapping between simulations and frontends, allowing simultaneous monitoring of multiple simulations or collaborative analysis of a single simulation from different perspectives. The design emphasizes flexibility through modular components and extensibility for integration with other simulation platforms.**

I. Introduction

Live dashboards for data visualization are commonly sought after not just in simulation [1], but in many domains, such as health [2], machine learning [3] or, even more broadly, stock trading. This is because research has repeatedly and consistently demonstrated that dashboards are an effective way to present real-time data to users [4].

In the context of OMNeT++ simulations [5], though, traditional approaches often restrict users to post-processing results, or they require manual intervention and adaptation of the model to inspect dynamic behavior.

Our work addresses these limitations by presenting LiveOppServation[1] – an open-source interactive Dash-based frontend that enables real-time monitoring and visualization of simulation data during execution. This system provides several key capabilities:

- **Runtime Evaluation**: Continuous observation of vector data with timestamps, allowing users to inspect values as they evolve during simulation.
- **Live Plotting**: Automatic generation of visualizations from collected data, supporting both real-time analysis and post-hoc inspection.
- **Conditional Breakpoints**: The ability to pause simulations based on specific criteria met by the observed data, enabling targeted investigation of critical events.

By leveraging widely adopted technologies like gRPC and Protocol Buffers (protobuf), our solution offers a flexible framework that can be extended to support additional simulation platforms and visualization paradigms.

[1]https://www.cms-labs.org/research/software/liveoppservation/

II. Related Work

The widespread use of dashboards has led to the development of numerous tools designed to implement them. Grafana is a popular open-source tool that allows the aggregation of data from multiple sources. Shiny for R and Streamlit for Python3 are tools used to create web-based dynamic dashboards. Dash, an open-source framework for building dashboard web applications in Python3, is based on the well-established Flask web framework. It offers a wide range of existing components and is also easy to extend. IEEE 1516-2025 [6] is a standard for the development of distributed simulations, and IEEE 1278.2-2015 [7] defines how to develop interactive distributed simulations, primarily focused on military applications.

Recent work has explored the use of web-based dashboards for simulation-data visualization. For instance, Jung et al. [1] developed a data visualization tool for VISSIM to investigate traffic simulations, with data stored in the cloud and utilizing elastic search for querying and visualization. Kourzanov [8] created a live interaction system for ns-3 that integrates directly with Julia notebooks and offers real-time analysis. Šljivo et al. [9] implemented a nodeJS-based web interface for ns-3 to study IEEE 802.11ah, enabling dynamic data exploration and visualization. Köstler and Kauer [10] developed a remote interface for OMNeT++ using AutobahnJS, jQuery, and ChartJS. Their implementation requires a client-side connection to the simulation, to store simulation results and a solid understanding of JavaScript for modifications of the frontend.

These solutions often focus on specific simulation platforms or lack the scalability and flexibility needed for broader applications.

III. System Design

We designed the system to support a many-to-many mapping between simulations and frontends. This allows multiple simulations to be monitored by a single frontend, enabling parallel evaluation of multiple repetitions or configurations. Conversely, multiple frontends can inspect a single simulation, allowing different variables and formulas to be analyzed from the same simulation independently.

The connection between the simulation and frontend is established using gRPC. The data is encoded on the wire using protobuf. This provides a robust and flexible data exchange mechanism that could also be used flexibly in combination with other communication mechanism – like ØMQ – in the future.

DS-RT 2025 – Posters

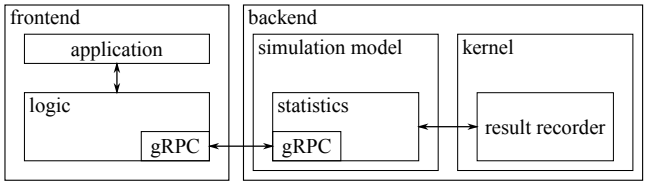

Figure 1. System architecture with frontend (containing the dashboard as an application and the data in the logic component) and the backend (containing the simulation kernel with the new vector result recorder and the statistics module in the simulation model) connected via gRPC. The results from a module are stored using the kernel's result recorder and can be accessed from the frontend through the statistics module. The frontend also stores and analyzes the data within its logic.

A. Frontend

The frontend is built using Dash, a Python3 framework based on Flask that allows for easy integration with Plotly and offers a wide range of HTML components.

The logic for maintaining connections and managing simulations and data is separated from the Dash application, making it possible to exchange visualization components independently.

The user interface (cf. Figure 2) consists of four main parts:

- general settings, includes connecting and selecting a simulation, and requesting time information;
- vector metadata, which provides a list of all registered vectors that can be filtered and used to select which data to send to the frontend;
- vector data, which displays the actual data from the selected vectors in both a table and a plot;
- and a debug watch list, which allows users to observe specific vectors and values and set conditional breakpoints.

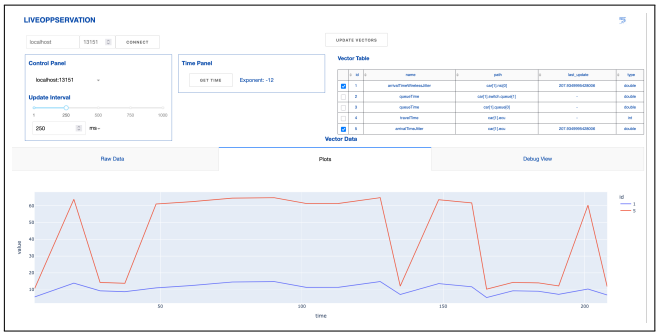

Figure 2. Screenshot of LIVEOPPSERVATION prototype.

B. Backend

The first new backend feature is a vector result recorder – which is part of the simulation kernel. This recorder is responsible for capturing and storing simulation data during runtime emitted by the modules with the corresponding timestamp. By default, OMNeT++ provides two types of result recorders: one that stores results to a file and another that stores them to an SQLite database. Furthermore, OMNeT++ provides a base vector recorder, which serves as a foundation for the creation of new ones, which we used to create a recorder that stores all results in memory using C++ structures and added functions that allow us to query and access the data during runtime.

The second functionality is a statistics module. This module is responsible for establishing a connection to the frontend, maintaining the debug watch list, and instructing the recorder to yield data when specific conditions are met. This module is implemented as a standard OMNeT++ module, which means it can be easily configured through the omnetpp.ini file. It also allows us to make the result recording independent of the connection to the server, enabling easy extensions and modifications. Creating a module within the simulation also allows efficient listening of emitted signals, which are the preferred way for the statistics recording.

These design choices ensure that the module and the result recorder can be seamlessly integrated into existing simulation setups without requiring modifications to the simulation kernel itself.

IV. Conclusion and Future Work

We were able to create an independent custom frontend for OMNeT++ simulations that can be used to watch and evaluate results of multiple simulations during runtime for evaluation or debugging. By using widely adopted technologies like gRPC and Protocol Buffers (protobuf), we were able to create a flexible and easily extensible system.

In the future we want to implement the ability to adjust parameters during runtime and the aggregation of multiple simulation results in one database.

Also, a new vector result recorder will be created that forwards the data to both the newly created in-memory recorder and to a second persistent recorder, defined by the user.

References

[1] J. Jung, T. Oh, I. Kim, and S. Park, "Open-sourced real-time visualization platform for traffic simulation," *Elsevier Procedia Computer Science*, vol. 220, pp. 243–250, 2023.

[2] U. Buchholz, A.-S. Lehfeld, A. Loenenbach, K. Prahm, U. Preuß, and W. Haas, *GrippeWeb - Daten des Wochenberichts*, Jul. 2025. [Online]. Available: https://robert-koch-institut.github.io/GrippeWeb_Daten_des_Wochenberichts/

[3] M. Abadi et al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*, White Paper, 2016.

[4] A. Perer and B. Shneiderman, "Integrating statistics and visualization: case studies of gaining clarity during exploratory data analysis," in *SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*, Florence, Italy: ACM, Apr. 2008, pp. 265–274.

[5] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *1st International ICST Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS 2008)*, Marseille, France: ICST, 2008.

[6] "Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules," IEEE, Std 1516-2025, 2025.

[7] "Standard for Distributed Interactive Simulation (DIS) – Communication Services and Profiles," IEEE, Std 1278.2-2015, 2015.

[8] P. Kourzanov, "Live network simulation in julia: design and implementation of LiveSim.jl," in *10th Workshop on ns-3 (WNS3 '18)*, Surathkal, India: ACM, 2018, pp. 30–36.

[9] A. Šljivo, D. Kerkhove, I. Moerman, E. De Poorter, and J. Hoebeke, "Interactive web visualizer for IEEE 802.11ah ns-3 module," in *10th Workshop on ns-3 (WNS3 '18)*, Surathkal, India: ACM, 2018, pp. 23–29.

[10] M. Köstler and F. Kauer, "A Remote Interface for Live Interaction with OMNeT++ Simulations," in *4th OMNeT++ Community Summit*, Bremen, Germany: arXiv, 2017.