Highlights

OppNDA: A Modular and Scalable Automation Framework for Streamlining DTN Research with the ONE Simulator

Hasan MA Islam, S. M. Nafis Shahriar, Pulok Akibuzzaman, Mahamudur Rahman Maharaz, Abdullah Sajid, Nusrat Rahman Aurna, Fayaza Islam, Mahdin Islam Ohi, Md. Khalid Mahbub Khan, Michael Georgiades

- A modular and interoperable network data analysis framework for delay-tolerant networks.
- A web app enabling experiment management and post-processing simulation reports for the ONE simulator.
- Integration support of machine learning algorithms on key point indicators of simulated opportunistic networks.

OppNDA: A Modular and Scalable Automation Framework for Streamlining DTN Research with the ONE Simulator⋆

Hasan MA Islam[a], S. M. Nafis Shahriar[a], Pulok Akibuzzaman[a], Mahamudur Rahman Maharaz[a], Abdullah Sajid[a], Nusrat Rahman Aurna[a], Fayaza Islam[a], Mahdin Islam Ohi[a], Md. Khalid Mahbub Khan[a,*,1] and Michael Georgiades[b,2]

[a]*Dept. of Computer Science and Engineering, East West University, Dhaka, Bangladesh*
[b]*Dept. of Computer Science, Neapolis University of Pafos, Pafos, Cyprus*

ARTICLE INFO

Keywords:
Opportunistic Network
Delay-Tolerant Network
Network Data Analysis
Internet of Vehicles

ABSTRACT

Opportunistic network simulations are essential for research on delay-tolerant networks. In these networks, nodes are intermittently connected, or traditional network infrastructure is absent. It requires meticulous efforts to produce valuable insights into simulation scenarios through comprehensive data engineering. This paper introduces Opportunistic Network Data Analyzer (OppNDA), a robust framework that complements the existing configuration and post-processing features of the ONE simulator to streamline research in DTNs. The user-friendly web application enables users to configure simulations and post-process reports through intuitive visualization. The interoperable modular architecture of OppNDA has the potential to be extended, scaled, and tailored to meet diverse research and development needs. In addition to various analyses, incorporating machine learning enhances data analytics and bolsters insights. This work consolidates the scope for future research in delay-tolerant networks through data informatics.

1. Introduction

Opportunistic network data analysis is the use of data-driven insights to understand networks that facilitate intermittent or opportunistic communication [1], accommodating mobile nodes and a constantly changing topology [2]. These networks are commonly referred to as delay-tolerant Networks (DTNs) [3], or Mobile Ad Hoc Networks (MANETs) [4]. DTNs lack a fixed infrastructure and depend on mobile or static nodes to receive and forward data. The Opportunistic Network Environment (ONE) simulator [5] has been designed and developed for opportunistic computing to evaluate the performance of various DTN routing algorithms [6] using a range of Key Performance Indicators (KPIs) [7], such as message delivery ratio, time-to-live (TTL), overhead ratio, buffer size, and buffer time. The simulator can generate voluminous reports based on simulation configurations and settings. The analysis of these networks opens exciting opportunities for research and development. Opportunistic network data analysis aims to understand the behavior of such ad hoc networks and to identify opportunities for performance optimization. However, there is no unified framework that facilitates network simulation configuration, simulation in the ONE simulator, and post-processing of simulation reports within a single platform.

To address the identified challenge, we have developed an automated, modular framework called the Opportunistic Network Data Analyzer (OppNDA) [1]. It serves the crucial purpose of streamlining and improving the efficiency of various DTN research processes, particularly by generating insights and enabling data-driven decision-making through automated repetitive tasks such as data collection, cleaning, processing, and visualization. In essence, the primary purpose of OppNDA is to equip researchers with tools and technologies to optimize their workflows, increase productivity, and deliver accurate, high-quality insights efficiently. This tool features a user-friendly, intuitive graphical

*Corresponding author
hasan.mahmood@ewubd.edu (H.M. Islam); nafisshahriar003@gmail.com (S.M.N. Shahriar); pulok519@gmail.com (P. Akibuzzaman); rahmanmehraj627@gmail.com (M.R. Maharaz); abdullah.sajid.7124@gmail.com (A. Sajid); nusrataurna07@gmail.com (N.R. Aurna); fayazaislam6@gmail.com (F. Islam); mahdinislamohi@gmail.com (M.I. Ohi); khalidmahbub.khan@yahoo.com (Md.K.M. Khan)
ORCID(s): 0009-0005-7028-7902 (H.M. Islam); 0009-0006-7414-6434 (S.M.N. Shahriar); 0009-0004-4119-2333 (P. Akibuzzaman); 0009-0006-4513-6552 (M.R. Maharaz); 0009-0006-9094-9958 (A. Sajid); 0009-0008-6873-9668 (N.R. Aurna); 0009-0001-7389-9980 (F. Islam); 0009-0005-4056-9757 (M.I. Ohi); 0000-0002-7566-4341 (Md.K.M. Khan); 0000-0002-5930-8814 (M. Georgiades)
[1]https://github.com/DhmaiNetRG/OppNDA

interface that enables users to configure simulation settings, generate and integrate custom map data, run simulations, analyze reports, and visualize data and user-defined Key Performance Indicators (KPIs) in a single operation. Moreover, evaluating the performance of opportunistic networks is complex [8, 9], as analyzing and processing large volumes of raw, unstructured data for visualization requires significant time and effort. Our lightweight Python-based framework, OppNDA, can produce statistical insights and visualizations with minimal effort, thereby reducing the risk of error. The interoperable nature of OppNDA enables integration with other scripts or APIs via Cross-Origin Resource Sharing (CORS) and with various machine learning algorithms to enhance the interpretability of results. This work leverages CORS to integrate a map data generator with OppNDA, demonstrating the framework's adaptability. OppNDA can significantly reduce the time and effort required to prototype and validate research initiatives in DTN. Consequently, the applications of opportunistic networks to real-world problems can be accelerated through OppNDA.

The key contributions of this article are as follows:

- Development of a data analysis framework for the delay-tolerant networks.
- Development of an intuitive, modular, and user-friendly web app for making it a one-stop solution for conducting experiments using the ONE simulator and post-processing results.
- Construction of an automated pipeline for data informatics of opportunistic communication that encompasses configuration, simulation, parsing, analysis, analytics, and visualization.
- Post-processing of the simulation reports involves producing statistical illustrations and predicting KPIs through various machine learning algorithms, while assessing their performance through residual analysis.

Table 1
Definitions of the notations used in this work

Symbol	Description
	System Parameters
N_{sim}	Total number of simulation scenarios in a campaign
P	Number of parallel worker processes (CPU cores)
C_{manual}	Total Time-to-Insight cost using manual effort
C_{OppNDA}	Total Time-to-Insight cost using OppNDA
Ψ	Visualization dispatch function
T_{parse}	Time Function of parsing simulation reports
T_{serial}	Estimated time of serial visualization
$T_{parallel}$	Estimated time of parallel visualization
M	Estimated memory consumption
$S_{speedUp}$	Speed Up Factor
P_{opt}	Optimal number of worker process
$\mathcal{M}(t)$	Memory footprint in multiprocessing environment
	Network Metrics
P_{del}	Delivery Probability
$\bar{L}$	Average Latency
O_{ratio}	Overhead Ratio
$\bar{H}$	Average Hop Count
$\mathcal{M}_{del}$	Set of successfully delivered messages

The system parameters and notation used in this article are listed in Table 1 and the rest of the work is organized as follows: Section 2 gives an overview of previous research relevant to this study; Section 3 describes the architecture of the Opportunistic Network Data Analyzer (OppNDA) and its core components; Section 4 outlines visual representations of post-processed data to visualize network performance; Section 5 describes the experimental setup and methodology used with OppNDA; Section 6 discusses the evaluation of generated reports and their role in network performance analysis; Section 7 states potential improvements and future directions for this research; and finally, Section 8 summarizes this work.

Table 2
Feature comparison of frameworks for automating and enhancing the workflow of opportunistic network simulator

Framework	Simulation Interface	Configuration Automation	Post-processing Automation	Data Visualization	ML Integration	Extensibility	GUI	Year
STARS	OMNeT++	Yes	Yes	Partial	No	Partial	No	2009
SAFE	ns-3	Yes	Yes	Yes	No	Yes	Yes	2012
SEM	ns-3	Yes	Yes	Yes	No	Yes	No	2019
SMO	OMNeT++	Yes	Yes	Yes	Partial	Yes	Partial	2021

2. Related Work

This section categorizes and discusses existing simulation tools, frameworks, and routing strategies for opportunistic networks by core functionality and methodological approach.

Data analysis is a critical component of research and development across disciplines, and visualizing data is necessary to improve comprehension and decision-making [10]. It is essential to understand the performance of computer networks through simulation to scale or modify them. Researchers used tools such as GloMoSim [11] to build modular environments for simulating large-scale wireless networks. Although GloMoSim included architectural features to support scalability, its performance depended on the partitioning strategy, synchronization algorithms, and inter-processor communication overhead, all of which required careful configuration for very large-scale scenarios. Furthermore, it required developing overlay layers to implement the store-carry-and-forward model, since it was designed to support continuous MANET connectivity rather than intermittent connections. Around the same time, ns-2 [12] emerged and supported both wired and wireless protocols. It was developed as an event-driven simulator. Like GloMoSim, ns-2 lacked native DTN support. It became popular very quickly, but its complexity and lack of flexibility led to the development of alternatives.

OMNeT++ [13] has become a strong, modular discrete-event simulator. Compared with ns-2, it offered a more modular and extensible architecture, making it attractive to many researchers. Many researchers preferred it because it supported hierarchical models.

DTNs have gained popularity within the research community, prompting the development of the ONE simulator. It gained popularity for its ease of use, built-in mobility models, and support for routing protocols. It was primarily designed for research on opportunistic networks and attracted a large user base. The ONE simulator provides native support for opportunistic network simulation, but does not support automated simulation configuration or advanced post-processing workflows. Consequently, researchers relied solely on the simulator and frequently utilized manual scripting and external tools. Because the ONE simulator faces optimization challenges in large-scale simulations, text-based configuration and external post-processing can impose additional overhead for researchers conducting and evaluating experiments. Although it is a popular choice among researchers for simulating DTNs, unlike OMNeT++ or ns-3 [14], the ONE simulator currently lacks an adaptive and unified framework for streamlined experimentation.

The simulation community later introduced frameworks like SAFE [15] for ns-3 and SMO [16] for OMNeT++, which significantly improved simulation workflows. These frameworks simplified setup, enabled concurrent execution, and facilitated data parsing and visualization, thereby enhancing the efficiency of older tools.

SMO was handy for OMNeT++ users interested in opportunistic networks. It supported structured testing of routing protocols and large-scale experiments. There is no one-size-fits-all solution for configuring and post-processing, although some studies have explored the use of external scripts and visualization techniques [17]. To address this gap, tools such as OppNDA have been developed to reduce effort and errors in processing network data to generate insights. OppNDA aims to provide a flexible post-processing pipeline that supports simulation setup, data parsing, and performance visualization.

Table 2 compares simulators and frameworks for opportunistic networks. The surveys [18, 19] review simulation tools like OMNeT++, the ONE, ns-3, and Adyton [20]. The work in [18] looks at core features such as scalability, extensibility, and model support. In contrast, this paper [19] focuses on evaluation methods, including mobility models, cache and traffic settings, and reproducibility. Our table adds newer factors, such as machine learning support and graphical interfaces, to reflect current research needs.

Table 3
Feature Summary of OppNDA

Feature Category	OppNDA Capabilities	Benefit
GUI	An interactive graphical user interface for configuration and data processing	User-friendly interface to aid researchers and developers with minimal experience of the simulator environment
Configure Simulation	Produce configuration files through GUI	Facilitates simulation scenario creation with interactive graphical interface
Simulation Automation	Trigger simulation with a designed scenario	Auto-initiate simulation after configuration without any intervention
Automated Post-processing	Initiate post-processing module after simulation completion	Parse files and processes reports to feed the data processing pipeline
Data Visualization	Generate various illustrations through specified configuration	Enhance insight generation through illustrations
Data Analytics	Extends data post-processing pipeline to support evaluation through machine learning	Enables predictive performance studies
Modular Architecture	Integrate extensions through modifying framework codebase or use different tools leveraging CORS	Facilitates empowering OppNDA framework with custom-built scripts and extend its usability

We compared frameworks with respect to automation configuration, post-processing, data visualization, ML integration, extensibility, and GUI support. While SEM and SAFE support automation and extensibility using ns-3. OMNeT++ [16] provides built-in tools, including an IDE, a post-simulation analysis environment, the command-line data-parsing tool (scavetool), and opp_runall, which automates multiple simulations. However, these tools are primarily designed for small-scale simulations and struggle to scale to large-scale campaigns. The ONE simulator lacks any extensible framework, leaving a gap in automated experiments and advanced analytics. Our proposed solution, OppNDA, addresses this gap by combining automation, visualization, ML-driven insights, and a user-friendly GUI.

Researchers have also explored various routing strategies for OppNets alongside simulation tools. Epidemic routing [21] was among the first methods that performed well but consumed significant resources. Later, Spray and Wait [22] became a more efficient choice, especially in resource-constrained environments [23]. OppNets were inspired by DTNs developed for space communication [24] and have since been adapted for terrestrial applications, including mobile and sensor networks [25].

Recent studies have integrated empirical mobility data with machine learning methodologies. For example, Rashidibajgan et al. [26] used GPS data from city transportation systems to improve routing parameters such as message TTL and buffer size. Bhattacharjee et al. [27] enhanced the ONE Simulator by incorporating multimedia communication capabilities, including broadcasting, multicasting, and disaster-recovery scenarios.

At the same time, ns-3 has grown to support research on intelligent networking. Riley et al. [14] provided a foundational overview of ns-3, the successor to ns-2. Gawłowicz et al. [28] integrated ns-3 with OpenAI Gym to facilitate the development of networking applications that employ reinforcement learning. Yin et al. [29] created ns3-ai, a shared-memory interface that connects ns-3 to Python-based AI frameworks.

Even with these improvements, many simulation tasks still require manual post-processing. The advancements delineated in this paper tackle this challenge by providing a unified solution that enhances reproducibility and streamlines post-processing across diverse simulation environments.

3. Architecture of Opportunistic Network Data Analyzer

The main characteristics of the OppNDA framework are outlined in Table 3. It emphasizes its data visualization and analysis capabilities, automated simulation and post-processing, GUI-based configuration, and a flexible, modular design.

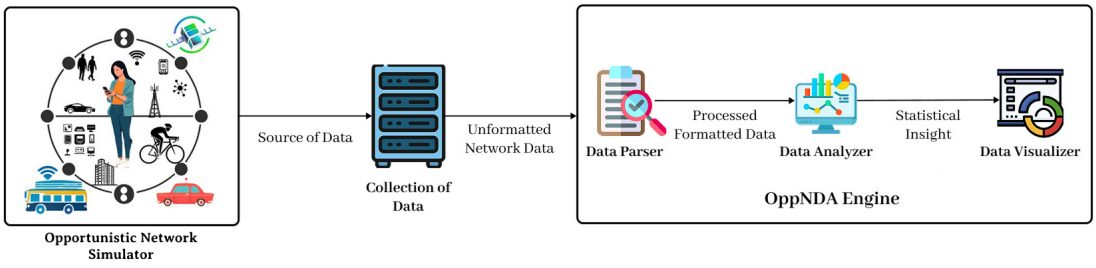


Figure 1: System overview of OppNDA as an interface of the ONE simulator

It takes time and effort to parse, process, and visualize data from voluminous reports. In addition, human involvement makes the process prone to errors. Let Ω denote a research campaign consisting of N_{sim} distinct simulation scenarios. We model cost as the total wall-clock time required to complete the campaign. The total cost of a manual workflow is defined as:

$$C_{\text{manual}}(\Omega) = \sum_{i=1}^{N_{\text{sim}}} \left(t_{\text{conf}}^{(i)} + t_{\text{exec}}^{(i)} + t_{\text{post}}^{(i)} \right) \tag{1}$$

In manual workflows, configuration and post-processing are dominated by human latency. OppNDA reduces this overhead by automating these steps and enabling parallel execution. The corresponding cost is approximated as:

$$C_{\text{OppNDA}}(\Omega) \approx t_{\text{setup}} + \frac{\sum_{i=1}^{N_{\text{sim}}} t_{\text{exec}}^{(i)}}{\kappa(P)} + \delta \tag{2}$$

where $\kappa(P)$ denotes the effective degree of parallelism achieved when executing simulations across P workers, capturing non-ideal effects such as load imbalance, scheduling overhead, and heterogeneous execution times. So, $\kappa(P) \leq P$ and δ represents system overhead. Our objective is to minimize the automation efficiency ratio, where

$$\eta = \frac{C_{\text{OppNDA}}}{C_{\text{manual}}} \tag{3}$$

Figure 1 presents an overview of the proposed framework. Here, the *OppNDA Engine* serves as the system's operational core. The simulator-generated unformatted data is collected by the *Engine*, which processes it through a series of stages. Furthermore, it produces graphical representations of complex datasets to facilitate easier analysis and decision-making.

The operation workflow of the OppNDA is illustrated in Figure 3. It supports three distinct paths, each enabling users to either initiate simulations with defined parameters or efficiently analyze previously generated results through structured processing and visualization. Users can configure and run simulations on defined scenarios and reuse existing configurations to process previously generated raw data for visualization without running simulations. These options streamline operations, enable efficient simulation execution, and facilitate the analysis of results. Building on this workflow, the architecture of the *OppNDA Engine* and complete data-processing pipeline are shown in Figure 2.

OppNDA is designed as an extensible framework for the ONE simulator. The automation pipeline involves configuring simulation settings and parsing, processing, and visualizing simulation data. It offers an intuitive GUI that enables easy access to simulation configuration and insight into the synthetic network data. The underlying technology for OppNDA uses Python to wrap the entire *Engine* and handle post-processing. It uses Flask to deploy the web app and provide user access points. Supplementary Material A contains the end-to-end workflow of the OppNDA and Supplementary Material B contains the documentation of the codebase generated with Sphinx [30].

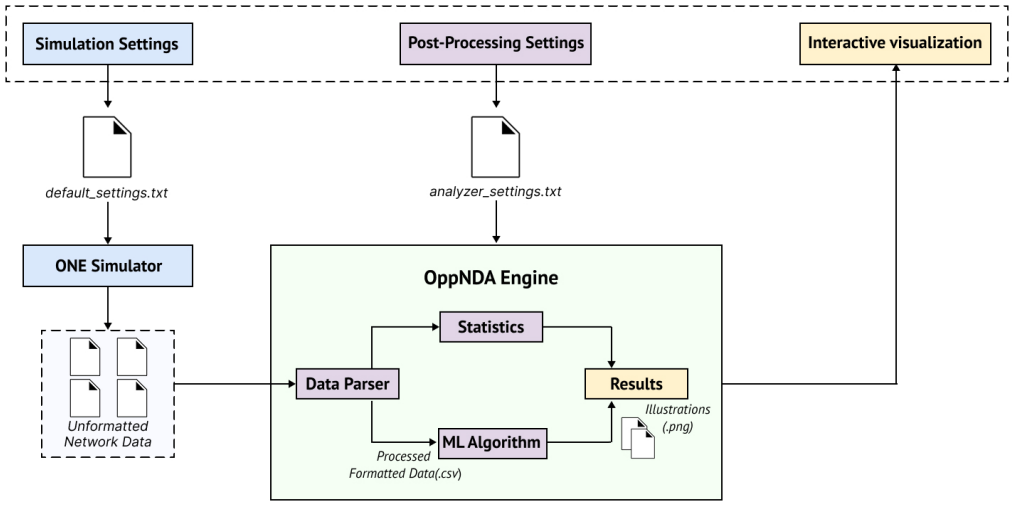


Figure 2: Architecture of the OppNDA Engine and data flow through OppNDA pipeline

Let $\mathcal{V}_{std} = \{V_{line}, V_{heat}, V_{KDE}, V_{violin}\}$ denote the standard set of univariate and bivariate projections. The visualization dispatcher Ψ extends this set based on the cardinality of varying parameters $|\mathbf{P}_{var}|$:

$$\Psi(\mathcal{D}, \theta) = \begin{cases} \mathcal{V}_{\text{std}} \cup \mathcal{V}_{\text{surf}} & \text{if } |\mathbf{P}_{\text{var}}| \geq 2 \\ \mathcal{V}_{\text{std}} & \text{if } |\mathbf{P}_{\text{var}}| = 1 \end{cases} \tag{4}$$

Ψ restricts complex 3D plotting to multivariate scenarios, and this deterministic mapping ensures that computational resources are allocated only to topologically valid projections of the simulation space.

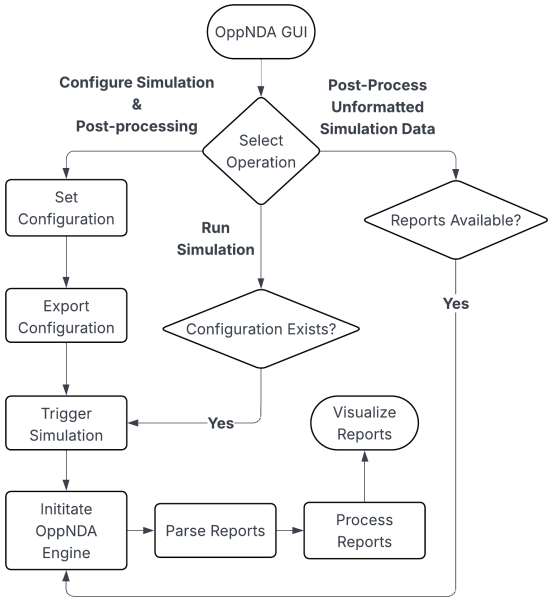


Figure 3: Operation workflow of OppNDA through GUI

Because the OppNDA system design supports modularity and interoperability, custom components can be integrated into the framework alongside the solution's built-in features. Keeping flexibility and adaptability in mind,

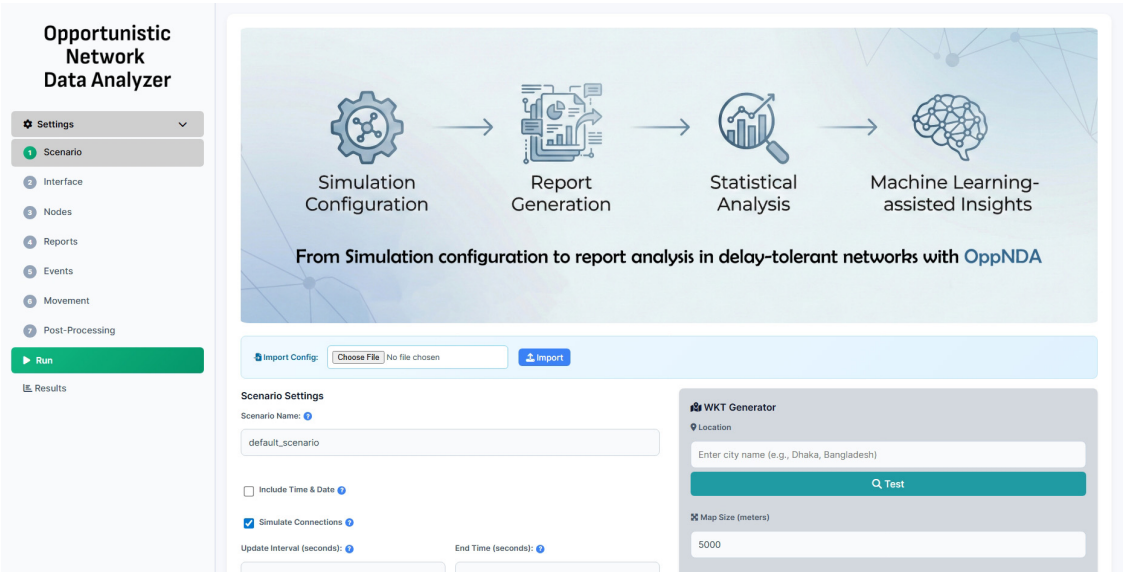


Figure 4: Graphical user interface of OppNDA. Users can modify the simulation and post-processing settings through the web app

we suggest using CORS to promote resource sharing with other web applications. We have included a tool for generating and integrating custom map data into the ONE simulator. This module uses the OpenStreetMap [31] public API to fetch map data as WKT files from the internet, which can then be further modified using tools such as OpenJump [32] to achieve the desired geospatial scenario.

3.1. Configuring simulations:

The ONE simulator parses the configuration file to construct structured data that defines the parameters of simulation scenarios. OppNDA includes a GUI that simplifies configuration, making it a more convenient and intuitive approach than navigating through lines of a text file to create a suitable simulation profile.

The OppNDA GUI comprises three segments: configuration, execution, and visualization. Figure 4 shows, within *Settings*, there are sections for configuring both the simulator and post-processing. The configurations of all the components are stored in JSON files to promote reproducibility and modularity. OppNDA exports the settings with every change and triggers the ONE simulator using the simulation setting.

3.2. Triggering Simulation:

Once the simulation scenario is configured, it can be saved as a text file in the local directory, enabling the ONE simulator to run with the specified settings. The configuration file is structured to support parsing and to enable the ONE simulator to initiate simulation via a batch or shell script invoked within the *Engine*.

The OppNDA pipeline allows users to configure, simulate, and process reports with a single button. OppNDA not only accelerates the entire workflow but also enables users to execute individual post-processing tasks using existing reports, without conducting a new simulation. Additionally, users can monitor the simulation status via the console using an OppNDA API endpoint that uses server-sent events (SSE).

3.3. Data Parsing:

The ONE simulator can produce large volumes of reports across different combinations of simulation environments. Parsing a large volume of reports is time-consuming and error-prone. Moreover, developing scripts to extract data from reports and process them requires expertise. Since OppNDA allows post-processing and evaluation of different parameter variations, it adapts filenames to prevent the simulator from overwriting reports. Upon completion of the simulation, the *Engine* triggers the post-processing segment. Data are extracted from reports and passed into the

processing pipeline. The filenames contain the simulation name and the KPI values (i.e., Delivery probability, Overhead ratio, Latency, Buffer time), which vary. The user can define the parsing delimiters to split filenames into substrings.

Let N_{sim} be the number of simulation reports and L be the average lines per report. By utilizing a multiprocessing pool with P worker processes, the approximated time function T_{parse} is reduced to:

$$T_{parse}(N_{sim}) \approx \frac{N_{sim} \cdot L \cdot \tau_{line}}{P} + \delta_{IO} \tag{5}$$

Where τ_{line} is the average processing time per line and δ_{IO} is I/O latency.

This native parser enables OppNDA to seamlessly transfer context from the simulation settings to the post-processing segment, producing dynamically meaningful visualizations of the reports.

3.4. Post-processing Reports:

The ONE simulator can generate a variety of reports at scale across different combinations of simulation environment settings. However, non-native solutions such as GraphViz [33] and gnuplot [34] are required to visualize and process data to gain insights into the simulation scenarios, or custom scripts can be automated to conduct data analysis. In contrast, OppNDA can parse a large volume of generated reports to perform the necessary computation.

Algorithm 1: Adaptive Parallel Batch Averaging

Data: Report directory $\mathcal{R}$, Report types $\mathcal{T}$, User-specified grouping variables $\mathcal{U}$, Mode $m \in \{\text{manual}, \text{auto}\}$, Workers P
Result: Set of averaged reports $\mathcal{A}$
$\mathcal{A} \leftarrow \emptyset$;
// Extract parameter tokens from structured filenames
$\mathcal{P} \leftarrow \text{ParseParameters}(\mathcal{R})$;
if $m = \textit{manual}$ **then**
  $\mathcal{V} \leftarrow \mathcal{U}$; // Use user-selected independent variables
else
  $\mathcal{V} \leftarrow \text{DetectVariations}(\mathcal{P})$; // Automatically identify varying parameters
end
foreach $t \in \mathcal{T}$ **do**
  $\mathcal{F}_t \leftarrow \{f \in \mathcal{R} \mid \text{MatchType}(f, t)\}$;
  $\mathcal{G} \leftarrow \text{Partition}(\mathcal{F}_t, \text{keys} = \mathcal{V})$;
  foreach $g \in \mathcal{G}$ ***in parallel*** *with* P *workers* **do**
    if $|g| \geq 2$ **then**
      $\bar{D} \leftarrow \text{Aggregate}(g, \text{op} = \text{Mean})$;
      $f_{\text{out}} \leftarrow \text{FormatName}(t, g.\text{key})$;
      $\text{WriteDisk}(\bar{D}, f_{\text{out}})$;
      $\mathcal{A} \leftarrow \mathcal{A} \cup \{f_{\text{out}}\}$;
    end
  end
end
return $\mathcal{A}$

OppNDA generates illustrations in parallel via multiprocessing, enabling it to produce visualizations at a large scale efficiently. After parsing the data, reports are averaged according to the user-defined variations or definitions, as specified in Algorithm 1. The visual dispatcher ensures the *Engine* meets the requirements of generating the selected types of visualizations. Algorithm 2 shows the workflow for how OppNDA decides and generates the plots. The generated plots are stored as images in the defined directory, along with the structured data in CSV files. Furthermore, the results are available with interactive features in the OppNDA dashboard.

OppNDA prioritizes parallel execution over serial processing to improve scalability and reduce end-to-end analysis latency. Let N denote the total number of visualization tasks generated during a simulation campaign, and let T_{gen}

Algorithm 2: Adaptive Visualization Dispatcher

Data: Aggregated dataset $\mathcal{D}_{agg}$, Raw dataset $\mathcal{D}_{raw}$, Configuration θ, Workers P
Result: Set of generated visualizations $\mathcal{I}$
$\mathcal{I} \leftarrow \emptyset$;
$\mathcal{V}_{std} \leftarrow \{\text{Line}, \text{Heatmap}, \text{KDE}, \text{Violin}\}$;
$\mathcal{V}_{surf} \leftarrow \{\text{Surface3D}\}$;
$\mathcal{M} \leftarrow \theta$.target_metrics;
$\tau \leftarrow \theta$.min_distinct;
Step 1: Identify Varying Parameters;
$\mathbf{P}_{var} \leftarrow \{p : |\text{unique}(\mathcal{D}_{agg}[p])| \geq \tau\}$;
Step 2: Visualization Selection;
if $|\mathbf{P}_{var}| \geq 2$ **then**
  $\mathcal{V} \leftarrow \mathcal{V}_{std} \cup \mathcal{V}_{surf}$;
else
  $\mathcal{V} \leftarrow \mathcal{V}_{std}$;
end
Step 3: Task Generation;
$\mathcal{Q} \leftarrow \emptyset$;
foreach $v \in \mathcal{V}$ **do**
  foreach $m \in \mathcal{M}$ **do**
    if $v = Surface3D$ **then**
      foreach $(p_i, p_j) \in \binom{\mathbf{P}_{var}}{2}$ **do**
        $\mathcal{Q}$.enqueue$(v, \mathcal{D}_{agg}, p_i, p_j, m)$;
      end
    else
      foreach $p \in \mathbf{P}_{var}$ **do**
        $\mathcal{Q}$.enqueue$(v, \mathcal{D}_{agg}, p, m)$;
      end
    end
  end
end
Step 4: Parallel Execution;
$\mathcal{I} \leftarrow$ ParallelExecute$(\mathcal{Q}, P)$;
return $\mathcal{I}$

represent the average execution time required to generate a single visualization, dominated by disk I/O and rendering operations.

Under a serial execution model, the total processing time is given by:

$$T_{serial} = \sum_{i=1}^{N} T_{gen}^{(i)} \approx N \cdot T_{gen} \tag{6}$$

To utilize multicore architectures, OppNDA employs a parallel execution model based on a worker pool of size P. According to Amdahl's Law [35], the theoretical speedup $S(P)$ achievable through parallelization is bounded by:

$$S(P) \leq \frac{1}{s + \frac{1-s}{P}} \tag{7}$$

where s denotes the fraction of the workload that is inherently sequential, including task scheduling and shared I/O coordination.

Assuming approximately uniform task execution times, the parallel execution time can be approximated as:

$$T_{parallel} \approx \left\lceil \frac{N}{P} \right\rceil \cdot T_{gen} + T_{overhead} \tag{8}$$

where $T_{overhead}$ captures multiprocessing overheads such as context switching, inter-process communication, and synchronization costs.

This execution model enables OppNDA to scale efficiently with increasing numbers of simulation configurations while maintaining practical analysis turnaround times.

OppNDA offers a variety of illustrations for post-processing and visualizing reports that provide information on the simulation scenario and KPIs. OppNDA also provides an extension to fit data into machine learning algorithms and evaluate their performance on generated datasets via residual analysis. It advances the capabilities of post-processing simulation reports with minimal effort and enhances the integration of machine learning in data analysis.

(i) **Statistics:** OppNDA provides access to different types of statistical illustrations to evaluate scenario performance. Various types of plots visualize the overall performance of the routers, while violin plots show the distribution of KPI central tendencies for each router. The Pearson correlation heatmap and kernel density estimation pair plot offer insights into the intricate relationships of the KPIs and a holistic overview of the entire dataset. For line plots, bar plots, and surface plots, simulation reports are averaged and grouped by KPIs to present them as a single unified data point. The statistical analysis enables the uncovering of complex performance dynamics in each simulation scenario in the ONE simulator.

(ii) **Machine Learning Algorithms:** Apart from report analysis, the datasets produced by the parser are structured in CSV files and can be fitted into different machine learning algorithms for conducting regression analysis to make predictions of the KPIs in any scenario. OppNDA offers access to a selection of machine learning algorithms (e.g., linear regression [36], K-Nearest Neighbors (KNN) [37], decision trees [38], and random forests [39]). In addition, its modular architecture enables the addition of new algorithms. It opens the door to opportunities for data analytics on synthesized structured network data.

3.5. Interactive Visualization:

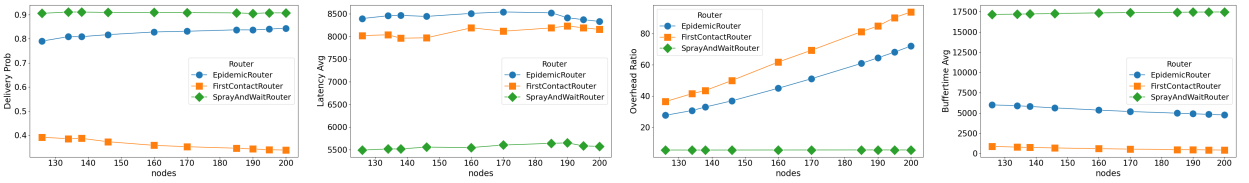
The persistent illustrations are available on the dashboard, with interactive features that facilitate navigation of the results generated by the simulation reports. It is available in the *Results* section of the GUI after post-processing. This segment can only present the illustrations. Additional controls are available in this feature. However, users can modify the source code to interact with the visualizations through the library features of Python packages.

4. Illustrations of Post-Processed Data

The ONE simulator [40] offers a handful of reports, and they focus on the evaluation of different metrics. Despite such support for data-intensive features, the ONE simulator does not provide a native solution for visualization and customization. It relies on external solutions to visualize data from the reports. OppNDA complements processing reports generated with the ONE simulator, enhancing data retrieval and visualization with intuitive features that are easy to implement and customize. It significantly reduces the workload required to process data, produce statistical insights, and gain a deeper understanding of simulation scenarios. Since OppNDA offers a modular and extensible architecture, it can be easily modified and customized to serve specific purposes, such as implementing various machine learning algorithms to predict KPIs and assess their performance through residual analysis.

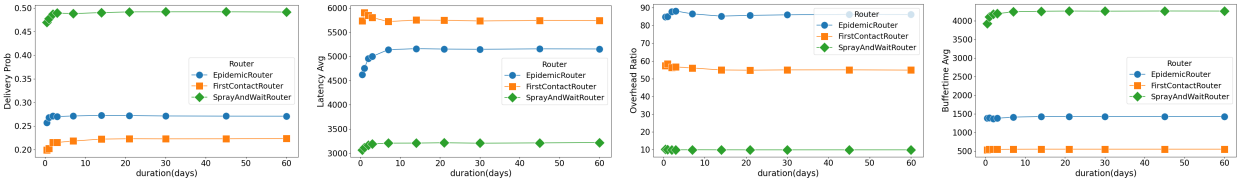
OppNDA provides various illustration types to generate insights from simulation reports with minimal effort, using Python packages such as Seaborn [41] and Matplotlib [42]. Through parsing, cleaning, and processing, the visualizations are prepared in accordance with the GUI settings.

1. **Line Plot:** Line plots are crucial for comparing the performance of multiple routers in terms of KPIs. It projects the trends and provides a holistic view of each router over the gradual progression of a metric. It can effectively illustrate the relationship between the independent and dependent variables.



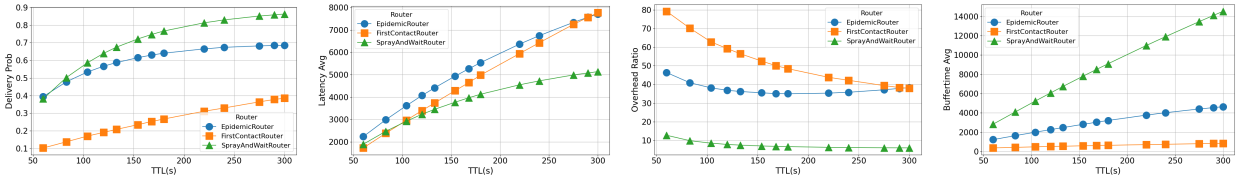
(a) Delivery Prob vs Nodes (b) Latency vs Nodes (c) Overhead Ratio vs Nodes (d) Buf Time vs Nodes

Figure 5: Performance comparison of KPI vs. number of nodes



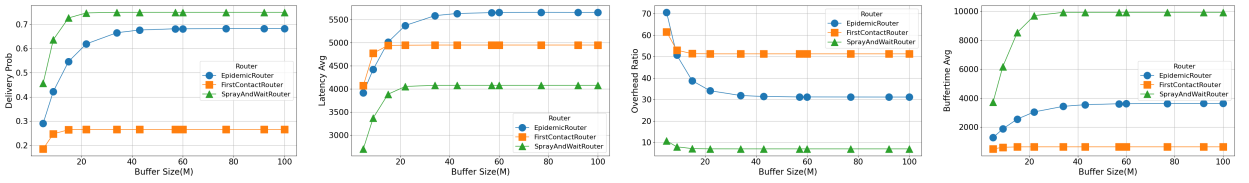
(a) Delivery Prob vs Duration (b) Latency vs Duration (c) Overhead Ratio vs Duration (d) Buf Time vs Duration

Figure 6: Performance comparison of KPI vs. simulation duration



(a) Delivery Prob vs TTL (b) Latency vs TTL (c) Overhead Ratio vs TTL (d) Buf Time vs TTL

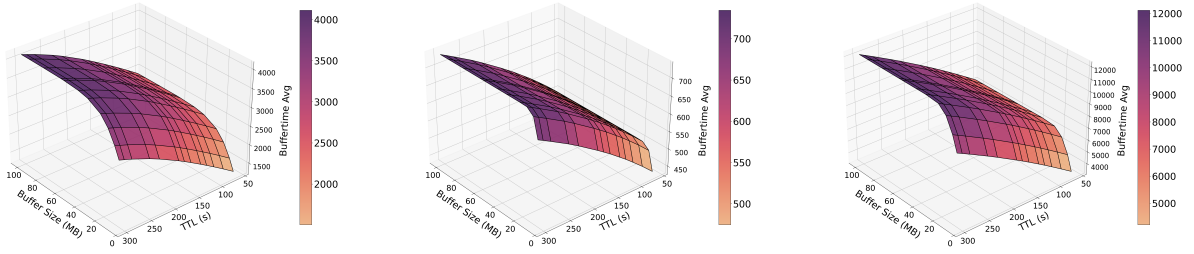
Figure 7: Performance comparison of key metrics vs. TTL for 7 days



(a) Delivery Prob vs Buf Size (b) Latency vs Buf Size (c) Overhead Ratio vs Buf Size (d) Buf Time vs Buf Size

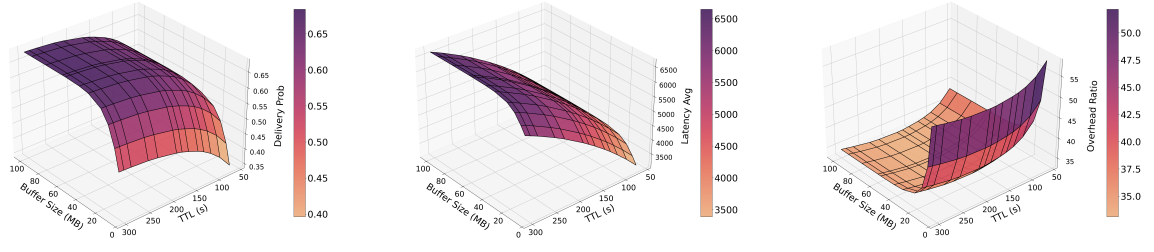
Figure 8: Performance comparison of KPI vs. buffer size for 7 days

2. **Surface Plot:** Unlike line plots, 3D surface plots can accommodate a bivariate study for any router. It elaborates on the findings of line plots by combining the changes in two metrics into a single illustration with an additional dimension.

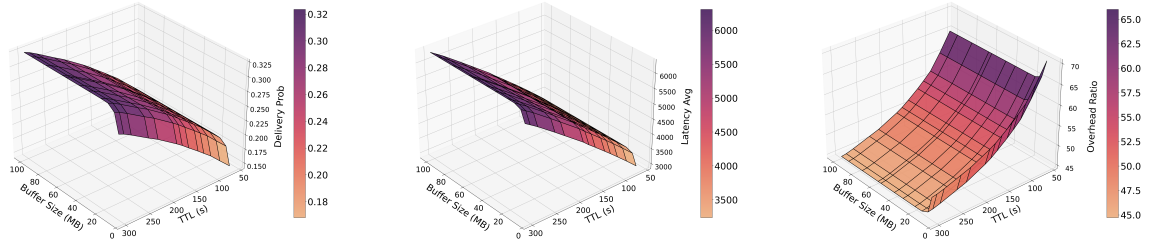


(a) Buf Time (Epidemic) (b) Buf Time (First Contact) (c) Buf Time (Spray and Wait)

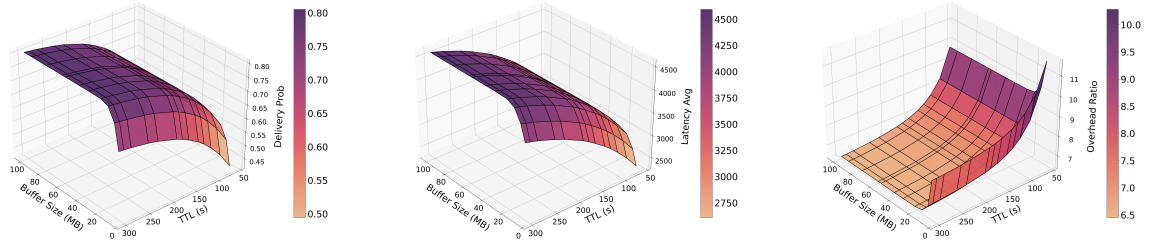
Figure 9: Comparison of buffer time for routers through 3D surface plots



(a) Delivery Prob (Epidemic) (b) Latency (Epidemic) (c) Overhead Ratio (Epidemic)



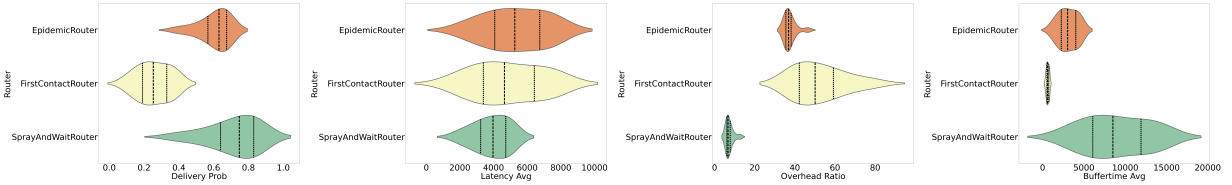
(d) Delivery Prob (First Contact) (e) Latency (First Contact) (f) Overhead Ratio (First Contact)



(g) Delivery Prob (Spray and Wait) (h) Latency (Spray and Wait) (i) Overhead Ratio (Spray and Wait)

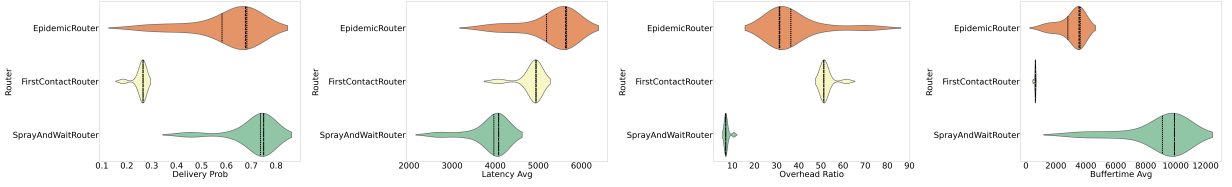
Figure 10: Network performance comparison through KPIs of the routers using 3D surface plot

3. **Violin Plot:** Central tendencies are significant for evaluating performances, as they illustrate average, minimum, and maximum values of a dataframe. Violin plots are similar to box plots, but they also show the distribution of the data, which can help in understanding the concentration of data points.



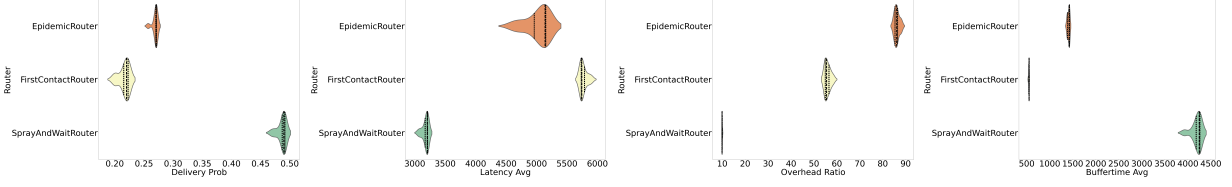
(a) Delivery Probability (b) Latency (c) Overhead Ratio (d) Buffer Time

Figure 11: Violin plot of KPIs (varying TTL over 7 days duration)



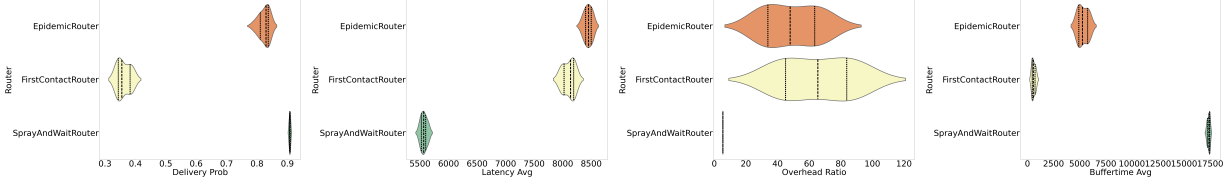
(a) Delivery Probability (b) Latency (c) Overhead Ratio (d) Buffer Time

Figure 12: Violin plot of KPIs (varying buffer size over 7 days duration)



(a) Delivery Probablity (b) Latency (c) Overhead Ratio (d) Buffer Time

Figure 13: Violin plot of KPIs (varying number of nodes)



(a) Delivery Probablity (b) Latency (c) Overhead Ratio (d) Buffer Time

Figure 14: Violin plot of KPIs (varying duration of simulation)

5. **KDE Pair Plot:** Kernel density estimation (KDE) pair plot displays the pairwise relationships among multiple variables in a dataset through univariate and bivariate distributions. The diagonal plots illustrate the distribution of a single variable using a continuous probability density function. In contrast, the off-diagonal plots show the relationship between two variables via contour lines representing the joint probability density, rather than a scatter plot.

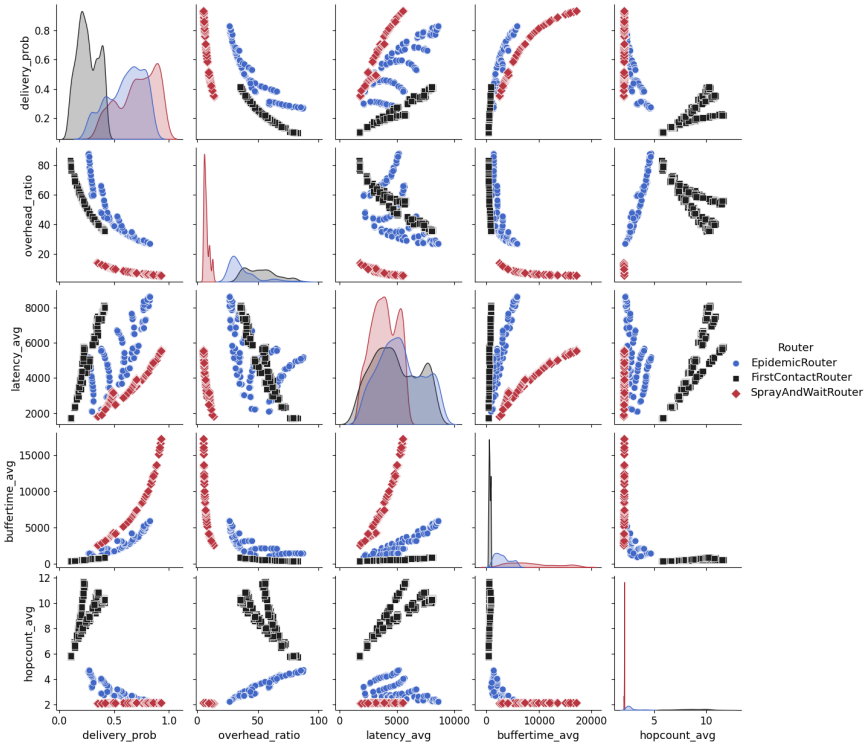
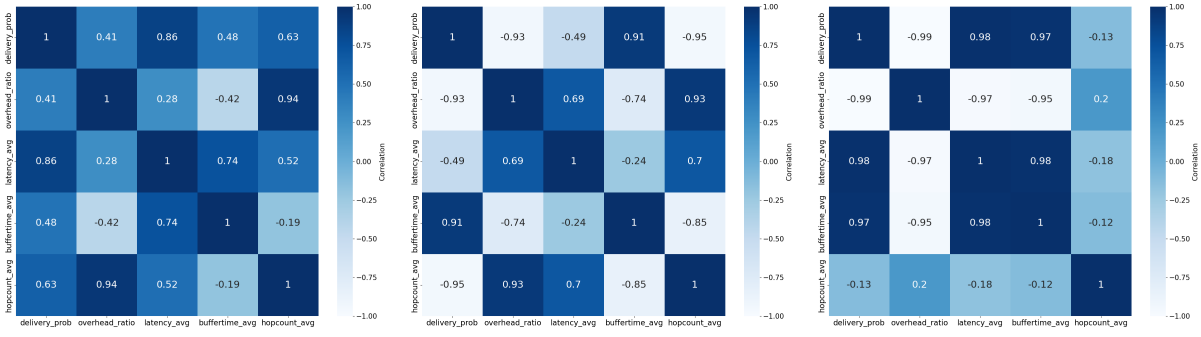


Figure 15: Pairplot of KPIs

4. **Correlation Heat Map:** The Pearson correlation heat map is a matrix that helps to identify correlations among various router metrics. It can be a vital tool for adjusting router functionality.



(a) Epidemic Router (b) First Contact Router (c) Spray and Wait Router

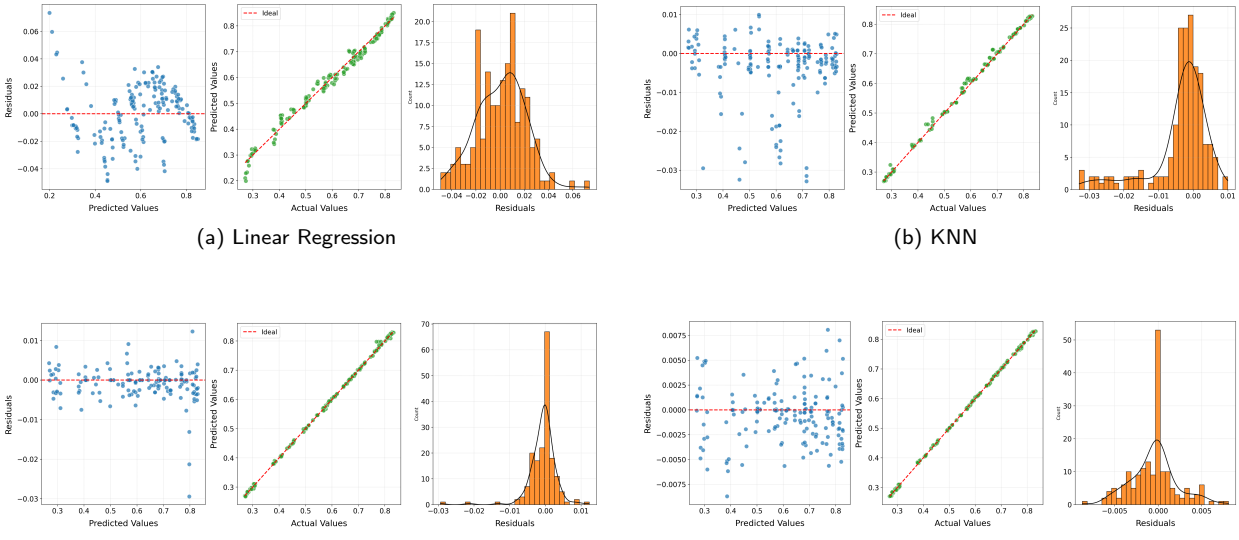
Figure 16: Correlation heatmaps of KPI vs TTL (varying duration of simulation)

For KPIs X and Y, Pearson correlation coefficient $\rho(X, Y)$:

$$\rho(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{9}$$

where the mean of X, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$; σ is the standard deviation, and $\rho(X, Y) \in [-1, 1]$

$|\rho|$ indicates strong correlation when $|\rho| > 0.7$, moderate correlation when $0.3 < |\rho| \leq 0.7$ and weak correlation when $|\rho| \leq 0.3$



(a) Linear Regression (b) KNN

(c) Decision Tree (d) Random Forest

Figure 17: Residual analysis of predicting delivery probability of spray and wait router for different ML models

6. **Residual Plot:** Machine learning algorithms can help predict KPIs, but their performance can be inconsistent. So, to ensure accurate predictions, regression analysis plays a crucial role in assessing residual errors, bias, and overall model accuracy. Figure 17 shows that different ML algorithms can be integrated and validated on simulation datasets.

Table 4
Regression models' performance across DTN routing protocols

Router	Model	RMSE (%)	MAE (%)	R^2 (%)
Epidemic	Linear Regression	2.04	1.64	98.43
	Decision Tree	0.40	0.22	99.94
	KNN	0.91	0.55	99.68
	Random Forest	**0.27**	**0.19**	**99.97**
First Contact	Linear Regression	0.31	0.24	99.89
	Decision Tree	0.13	0.06	99.98
	KNN	0.66	0.36	99.49
	Random Forest	**0.09**	**0.04**	**99.99**
Spray and Wait	Linear Regression	0.85	0.64	99.76
	Decision Tree	0.19	0.06	99.99
	KNN	0.37	0.25	99.96
	Random Forest	**0.11**	**0.05**	**100.00**

We can perform residual analysis to determine which machine learning algorithm is a better fit for making accurate predictions. We predicted the delivery probability of different routing protocols using features such as TTL, buffer size, overhead ratio, latency, buffer time, and hop count. However, users can modify it to use other parameters available in the reports. To ensure that the high R^2 values were not due to overfitting, the model was validated using k-fold cross-validation and a data split, thereby ensuring that the framework can predict KPIs for unseen configuration vectors using defined algortihms. And the resulting performance metrics for all models are reported in Table 4.

OppNDA handles illustration using multiprocessing, enabling efficient execution of large-scale visualizations. After post-processing, the results are stored in the local directory, including the CSV files, which can be used to generate additional insights. Additionally, the results are available with interactive features in the *Results* section.

5. Experiment Through OppNDA

OppNDA exports the configuration file into the defined directory to initiate the simulation. We designed three variant scenarios to assess the contributions of OppNDA to evaluating network performance and uncovering insights.

The experiments were structured into three simulation configurations to evaluate the effects of KPIs on network performance, which are as follows:

- In the first configuration, the simulation ran for 7 days, with message TTL values ranging from 60 to 300 minutes, buffer sizes between 5 MB and 100 MB, and five rngSeed between 10 and 83.
- In the second configuration, the duration was varied from 0.5 to 60 days while the rest of the parameters remained unchanged.
- In the third configuration, the number of nodes was varied over a single day duration, ranging from 126 to 200.

The last two configurations had two rngSeed while all the other parameters of the configurations remained indifferent to the default settings available in the GitHub repository of the ONE simulator[2]

5.1. Definition of Key Performance Indicators

To ensure rigorous evaluation, we formally define the KPIs used in this study. Let $\mathcal{M}$ be the set of all unique messages generated during the simulation, and let $\mathcal{M}_{del} \subseteq \mathcal{M}$ be the subset of messages successfully delivered to their destinations.

5.1.1. Average Hop Count ($\bar{H}$)

Let $\mathcal{P}_m = (v_0, v_1, \ldots, v_k)$ denote the ordered sequence of nodes (path) traversed by a message $m \in \mathcal{M}_{del}$, where v_0 is the source node and v_k is the destination node. The hop count $h(m)$ is defined as the number of edges in this path (i.e., $h(m) = k$).

The average hop count $\bar{H}$ for the simulation is the mean path length of all successfully delivered messages:

$$\bar{H} = \frac{1}{|\mathcal{M}_{del}|} \sum_{m \in \mathcal{M}_{del}} h(m) \tag{10}$$

5.1.2. Delivery Probability (P_{del})

The delivery probability is defined as the ratio of successfully delivered messages to total unique messages created:

$$P_{del} = \frac{|\mathcal{M}_{del}|}{|\mathcal{M}|} \tag{11}$$

5.1.3. Average Latency ($\bar{L}$)

For each message $m \in \mathcal{M}_{del}$, let $t_{gen}(m)$ be the generation time and $t_{recv}(m)$ be the reception time. The average latency is calculated over the set of delivered messages:

$$\bar{L} = \frac{1}{|\mathcal{M}_{del}|} \sum_{m \in \mathcal{M}_{del}} \left(t_{recv}(m) - t_{gen}(m) \right) \tag{12}$$

5.1.4. Overhead Ratio (O_{ratio})

Let N_{rel} be the total number of messages forwarded by all nodes. The overhead ratio quantifies the cost of redundant transmissions required for each successful delivery:

$$O_{ratio} = \frac{N_{rel} - |\mathcal{M}_{del}|}{|\mathcal{M}_{del}|} \tag{13}$$

[2]https://github.com/akeranen/the-one

5.1.5. Average Buffer Time ($\bar{B}_t$)

Let $b_t(m)$ denote the total time a message m resides in the buffers of intermediate nodes during its transit. The average buffer time is:

$$\bar{B}_t = \frac{1}{|\mathcal{M}_{del}|} \sum_{m \in \mathcal{M}_{del}} b_t(m) \tag{14}$$

6. Report Evaluation with OppNDA

One of the significant contributions of OppNDA is its ability to visualize datasets generated from simulations at any scale to assess the performance metrics of configured networks with minimal effort, from pipeline initiation to final results. This work enhances the post-processing support provided by the ONE simulator for DTN protocol evaluation. Users can configure illustration settings using JSON files, which can also be edited via the GUI. OppNDA offers a variety of methods for obtaining statistical insights, and users can select the configuration that best suits their needs.

Factors such as TTL and buffer size can be pivotal to the performance of DTN protocols under different network configurations where communication is delayed. A line plot is a suitable choice for plotting points to understand the holistic nature of a router, in which simulation reports are grouped and averaged to produce individual data points.

Let,

- $\mathcal{T} = \{t_1, t_2, ..., t_m\}$: Set of TTL values, $t_i \in \mathbb{R}^+$
- $\mathcal{B} = \{b_1, b_2, ..., b_n\}$: Set of buffer sizes, $b_j \in \mathbb{R}^+$
- $\mathcal{N}_{nodes} = \{n_1, n_2, ..., n_k\}$: Set of node counts, $n_l \in \mathbb{Z}^+$
- $\mathcal{D} = \{d_1, d_2, ..., d_p\}$: Set of durations, $d_q \in \mathbb{R}^+$
- $\mathcal{S} = \{s_1, s_2, ..., s_q\}$: Set of rngSeed, $s_h \in \mathbb{Z}^+$
- $\mathcal{R} = \{r_1, r_2, ..., r_w\}$: Set of routing protocols

Configuration of the first experiment:

- $|\mathcal{T}| = 13$ [TTL(Minute): 60, 83, 104, 120, 133, 154, 168, 180, 220, 240, 275, 290, 300]
- $|\mathcal{B}| = 11$ [buffer size(MB): 5, 9, 15, 22, 34, 43, 57, 60, 77, 92, 100]
- $|\mathcal{S}| = 5$ [rngSeed: 10, 21, 32, 44, 83]
- $|\mathcal{R}| = 3$ [routers: Epidemic, First Contact, Spray and Wait]

Total simulation: $13 \times 11 \times 3 \times 5 = 2145$ reports

Configuration of the second experiment:

- $|\mathcal{D}| = 10$ [duration(days): 0.5, 1, 2, 3, 7, 14, 21, 30, 45, 60]
- $|\mathcal{S}| = 2$ [rngSeed: 10, 83]
- $|\mathcal{R}| = 3$ [routers: Epidemic, First Contact, Spray and Wait]

Total simulation: $3 \times 2 \times 10 = 60$ reports

Configuration of the third experiment:

- $|\mathcal{N}_{nodes}| = 10$ [nodes: 126, 134, 138, 146, 160, 170, 185, 190, 195, 200]
- $|\mathcal{S}| = 2$ [rngSeed: 10, 83]
- $|\mathcal{R}| = 3$ [routers: Epidemic, First Contact, Spray and Wait]

Total simulation: $3 \times 2 \times 10 = 60$ reports

OppNDA utilizes multiprocessing for post-processing reports. Parsing, averaging, and visualizing data are performed by distributing the workload among an optimal number of workers to improve resource utilization and reduce execution time as described in Equation 6.4. This work validates the claim by running the post-processing module for the first configuration discussed above on a Windows machine with 32 GB of RAM and a Core i7-1185G7 processor, processing 2,145 message stat reports. The drastic improvement of average execution time, CPU usage, and memory usage comparison as shown in Figure 19, along with memory and CPU usage over execution time as shown in Figure 18, shows that multiprocessing effectively reduces the time and effort required for the research community to parse and visualize large-scale reports.

6.1. Exploratory Data Analysis

Figure 7 provides a detailed overview of how the three routers performed under variations in TTL, whereas Figure 8 focuses on the changes under variation in buffer size. We also varied the number of pedestrian nodes and the simulation duration to observe how OppNDA illustrates these insights. Figure 5 shows the impact of increasing pedestrian traffic in the simulation across different routers. Figure 6 visualizes the performance of routers over durations of 60 days with the default settings of the ONE. It also illustrates KPIs under default settings, and the geospatial data for Helsinki saturates after 7 days. Such saturated behavior can also be observed in the variation of buffer size. It was observed that a 40 MB buffer size caused changes in the KPIs of the three routers to stall.

Although changes in KPIs with respect to independent variables such as TTL and buffer can be illustrated individually, combining these variables in a single plot enables identification of their impact on a KPI. The Figure 9 and Figure 10 show comparisons through a 3D surface plot. The routers exhibit similar trends in delivery probability, latency, and buffer time. Still, the overhead ratio varies across routers, particularly for the epidemic router, which has the highest overhead ratio when the buffer size is smallest, and TTL has little to no effect. Increasing TTL extends flooding duration, but the overhead ratio rises as a trade-off for achieving a higher delivery ratio.

The central tendency represents the distribution of router data points for a particular KPI. The Figures 11, 12, 13, and 14 show the central tendency of routers for the KPIs in each simulation scenario. In contrast to box plots, violin plots also show multimodal distributions and density peaks, which can help assess the consistency of performance. The spray and wait router exhibits significant variation in buffer time across scenarios. Although the epidemic router exhibits considerable variability in data distribution through different scenarios, it is somewhat inconsistent and prone to changes in simulation dynamics.

The correlation among KPIs is crucial for performance assessment, as it uncovers intricate relationships among the metrics. This work visualized the correlation between simulation KPIs and simulation durations (0.5-60 days) to observe router behavior, as shown in Figure 16. It shows that the first contact router has the highest delivery probability when it has the lowest hop count, thereby achieving lower latency. On the other hand, the spray and wait router shows that, compared to the other two routers, the overhead ratio decreases as latency increases, meaning the longer a message waits to be delivered, the higher the chance it will reach its destination and therefore reduce message overhead. Unlike line plots, heatmaps are generated from disaggregated data points to preserve KPI correlations and avoid Simpson's paradox [43].

The diagonal and off-diagonal plots of the KDE can provide a comprehensive overview of the dataset. Figure 15 presents a KDE pairplot illustrating the joint and marginal distributions of KPIs. The KDE pairplot highlights clear trade-offs between reliability and efficiency. Epidemic routing maximizes delivery probability but incurs excessive overhead and buffer occupancy. Spray and wait achieves a balanced operating region with bounded resource consumption, whereas the first contact router prioritizes minimal overhead at the expense of delivery performance. These trends are consistent across marginal and joint KPI distributions. It reaffirms the fundamental design characteristics of each routing strategy. Also, the distinct clustering patterns across routers indicate that the KPI relationships are router-specific rather than parameter-driven.

Through exploratory data analysis, insights into network performance can be gained through careful observation.

6.2. Predictive Modeling and Evaluation

OppNDA integrates machine learning algorithms to predict network KPIs from configuration parameters.

6.2.1. Regression Model

We model a target KPI Y (i.e., Delivery Probability) as a function of the input configuration vector $\mathbf{X} = (x_1, x_2, \ldots, x_p)$ (i.e., TTL, Buffer Size). OppNDA constructs a mapping function $f : \mathbf{x} \rightarrow y$, where $\mathbf{x}$ is the configuration vector (e.g., TTL, Buffer Size) and y is the target KPI. We employ the following regressors:

6.2.2. Linear Regression

The baseline model assumes a linear relationship between parameters and network performance. We estimate the coefficient vector $\boldsymbol{\beta}$ to minimize the residual sum of squares:

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \quad \hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{15}$$

6.2.3. K-Nearest Neighbors (KNN)

For non-linear topologies, KNN predicts the KPI for a new configuration $\mathbf{x}_0$ by averaging the outcomes of the K closest historical simulations in the feature space $\mathcal{N}_0$:

$$\hat{y}(\mathbf{x}_0) = \frac{1}{K} \sum_{\mathbf{x}_i \in \mathcal{N}_0} y_i \tag{16}$$

6.2.4. Decision Tree Regressor

The decision tree recursively partitions the parameter space into distinct regions $R_1, \ldots, R_M$. The prediction for a given region is the mean KPI of simulations falling within it:

$$\hat{f}(x) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}(x \in R_m), \; c_m = \text{avg}(y_i | x_i \in R_m) \tag{17}$$

6.2.5. Random Forest

Compared with a single decision tree, a random forest employs an ensemble of B decorrelated decision trees to achieve better performance. It can be expressed as the aggregated average:

$$\hat{y}_{rf} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\mathbf{x}) \tag{18}$$

6.2.6. Residual Analysis and Error Metrics

To validate the model's accuracy, we analyze the residuals e_i for each test sample i:

$$e_i = y_i - \hat{y}_i \tag{19}$$

The model performance is quantified using the Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and the Coefficient of Determination (R^2):

Root Mean Square Error:

$$\text{RMSE} = \sqrt{\frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} (\hat{y}_i - y_i)^2}$$

Mean Absolute Error:

$$\text{MAE} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} |\hat{y}_i - y_i|$$

Coefficient of Determination:

$$R^2 = 1 - \frac{\sum_{i=1}^{N_{\text{test}}} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N_{\text{test}}} (y_i - \bar{y})^2}$$

where $\bar{y} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} y_i$

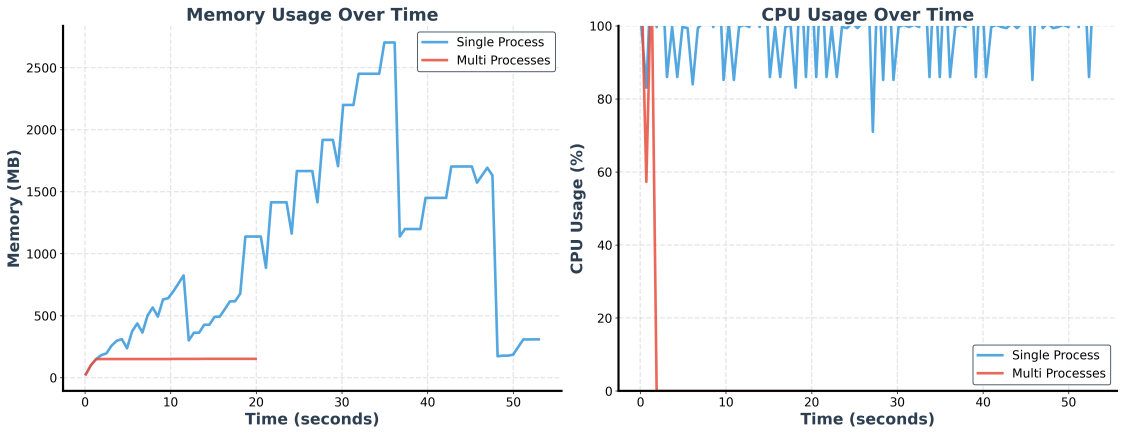


Figure 18: Performance assessment of single process and multi-processes for average execution time, CPU usage, and memory usage

6.3. Validation of Framework Integrity

To ensure that OppNDA functions as a transparent abstraction layer over the ONE simulator without introducing distortions or bias, we validate the framework along two dimensions: data consistency and baseline agreement.

6.3.1. Data Consistency

Let $\mathcal{R}$ denote the set of raw simulation reports generated by the ONE simulator, and let $\mathcal{D}$ denote the structured dataset produced by OppNDA. The parsing process $\Phi : \mathcal{R} \rightarrow \mathcal{D}$ extracts simulator-defined performance metrics using deterministic pattern-matching rules. OppNDA constructs a relation that explicitly mirrors the set of performance indicators reported by the simulator. For each key metric defined in the simulator output, a corresponding field exists in $\mathcal{D}$, ensuring that no simulator-reported information is omitted during parsing. To verify consistency, we perform direct cross-checks between selected raw reports and their parsed representations, confirming numerical equivalence of all extracted metrics. Since parsing does not involve aggregation, filtering, or transformation at this stage, the process preserves the original simulator outputs exactly. Consistency checks were performed across multiple simulation configurations by randomly sampling raw report files and verifying one-to-one correspondence between simulator-reported values and their parsed representations for all extracted metrics.

6.3.2. Baseline Agreement

We compare performance metrics computed by the framework with baseline results obtained from manual post-processing of simulator outputs to validate the analytical correctness of OppNDA. For identical simulation configurations, OppNDA-generated metrics exhibit exact agreement. This confirms that the framework does not introduce analytical artifacts and faithfully reproduces established evaluation outcomes.

These validation results demonstrate that OppNDA preserves both the quantitative integrity of simulator outputs and the qualitative performance trends reported in prior DTN studies. It is important to note that this validation focuses on analytical correctness and data preservation and does not aim to assess the fidelity of the underlying simulator.

6.4. Performance Analysis

This work employs multiprocessing for visualization generation to reduce execution time and improve resource utilization. We define the speedup factor S_{speedup} as the ratio between single-process execution time T_{single} and parallel execution time T_{parallel}:

$$S_{\text{speedup}} = \frac{T_{\text{single}}}{T_{\text{parallel}}} \tag{20}$$

In OppNDA, visualization workloads consist of a large number of independent tasks, and in typical usage scenarios, the number of reports, N_{reports}, significantly exceeds the number of available processing cores, P. Under such conditions, parallel execution enables effective workload distribution across workers. However, the achievable speedup is bounded by the overhead of sequential coordination and inter-process communication costs.

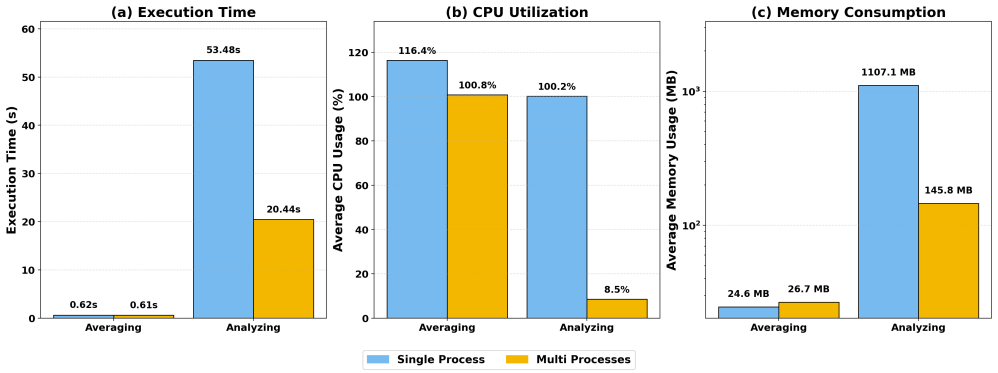


Figure 19: CPU and memory usage of single process and multiple processes over time to average and analyze for visualization.

Consequently, the observed speedup satisfies:

$$S_{\text{speedup}} \leq P \tag{21}$$

and approaches linear scaling only under ideal conditions, where task execution times are uniform and parallel overhead is negligible.

This work significantly reduces memory usage through multiprocessing. Let M_{base} be the baseline memory footprint of the *OppNDA Engine* runtime. For a batch of k concurrent worker processes, where each process handles a report of size S_{rpt}, the peak memory consumption M_{peak} is modeled as:

$$M_{peak} \approx M_{base} + k \cdot (C_{parse} \cdot S_{rpt})$$

where C_{parse} is the parsing overhead constant. OppNDA optimizes k dynamically such that $M_{peak} \leq M_{sys}$, preventing swap-thrashing.

In the multiprocessing environment with P workers, the instantaneous memory footprint $\mathcal{M}(t)$ is modeled as:

$$\mathcal{M}(t) \approx \mathcal{M}_{base} + \sum_{i=1}^{P} \left(\gamma \cdot \text{size}(r_i) + \mathcal{M}_{overhead}\right)$$

where γ denotes the data expansion factor introduced during in-memory processing, r_i is the report handled by worker i, and $M_{overhead}$ represents fixed per-process overhead.

To prevent OS-level thrashing, OppNDA implements a dynamic semaphore that caps P such that:

$$P_{opt} = \max \left\{ p \in \mathbb{Z}^+ \mid \mathcal{M}(t)\Big|_{P=p} \leq \eta \cdot \mathcal{M}_{RAM} \right\}$$

where M_{RAM} is the total available system memory and $\eta \in (0, 1)$ is a conservative safety margin for executing the *OppNDA Engine*.

The parameters η and γ are framework-level constants and do not alter the underlying simulation behavior, and are not tuned per experiment. All evaluations in this work use fixed values for these parameters to ensure consistency and reproducibility. The adaptive process of scheduling explains the observed memory stabilization in Figure 18. It demonstrates that multiprocessing improves execution efficiency without exceeding system resource limits. Figure 19 illustrates that multiprocessing outperforms execution through a single process in both averaging and analyzing the reports.

7. Future Work

Future directions for this work include making OppNDA versatile and compatible with heterogeneous network simulators, enabling it to adapt to the content-centric network (CCN) framework [44] as an application module within

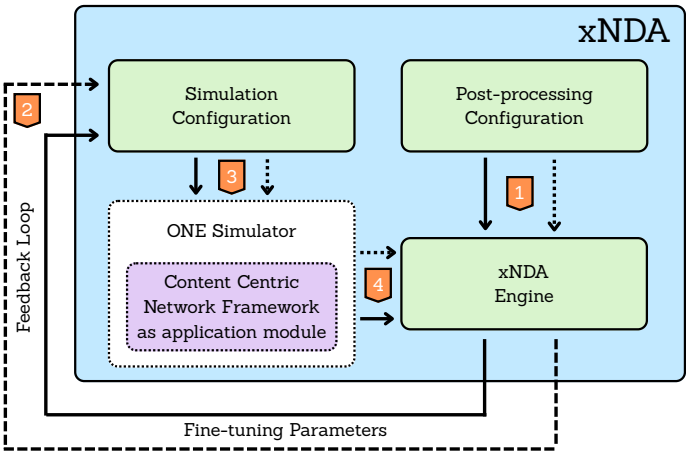


Figure 20: OppNDA transitioning into xNDA as a potential prospect of future directions to integrate the content-centric network framework for facilitating content-centric network data analysis

the ONE simulator. Other types of networks can also be encoded as overlays. It would evolve the *OppNDA Engine* into the *xNDA Engine* to support a diverse pool of network simulators and provide seamless post-processing of simulation reports. We aim to include features to replicate and modify simulators' components and to introduce distributed data analytics to enhance scalability. Another goal of this work is to use reinforcement learning to predict simulation parameters (e.g., node velocity and transmission speed) and to automate configuration within specified constraints (e.g., buffer size or TTL) across arbitrary scenarios. It will enable the xNDA to predict simulation settings based on predicted KPIs, thereby achieving suitable simulation outcomes. Steps 2, 3, and 4 form a feedback loop that supports reinforcement learning for prediction and fine-tuning of simulation settings. Once the user defines the desired simulation outcome and constraints in step 1, as shown in Figure 20, the system iterates through these steps. However, it would introduce new challenges beyond incorporating features from different simulators into the GUI, such as managing cold starts and the state space of simulation configurations.

xNDA will support the research community in advancing experiments on delay-tolerant, content-centric, and other networks through processing simulation reports and producing insights.

8. Conclusion

The growing complexity of opportunistic network simulations underscores the need for analytical frameworks that can accurately and efficiently process voluminous, unstructured raw data. While several existing tools and frameworks address isolated aspects of simulation or data analysis, there remains a clear need for a system that unifies the entire workflow. This work, OppNDA, addresses this by managing configuration, executing simulation, and automating post-processing in a single platform for the research community. By combining a user-friendly GUI with the data-informatics pipeline, researchers and practitioners can analyze complex network behaviors effortlessly with OppNDA. It transforms raw data into meaningful insights through its modular and extensible architecture. Although the current OppNDA framework has been tailored for integration with the ONE simulator and for supporting opportunistic network data analysis, this work is underway to seamlessly incorporate other simulators and networks, improve the user experience, and extend the features to support the computer network research community.

Supplementary Materials

Two supplementary documents are provided to support reproducibility and practical adoption of OppNDA. Supplementary Material A presents a step-by-step workflow demonstrating the complete pipeline from simulation configuration to result generation. Supplementary Material B contains extended technical documentation of the framework.

Funding

This work did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No external datasets were used in this study. Simulation outputs are generated using the ONE simulator. Supplementary materials are provided with the submission, and the OppNDA framework will be released publicly via a version-controlled repository upon acceptance.

CRediT authorship contribution statement

Hasan MA Islam: Planning, Supervision, Software, Writing – original draft, Writing – review & editing, Validation, Methodology, Conceptualization.

S. M. Nafis Shahriar: Writing – original draft, Writing – review & editing, Software, Experimentation, Validation, Visualization, Data Analysis.

Pulok Akibuzzaman: Writing – original draft, Writing – review & editing, Software, Experimentation, Validation, Data Analysis.

Mahamudur Rahman Maharaz: Planning, Supervision, Writing – review & editing, Investigation, Conceptualization.

Abdullah Sajid: Writing – review & editing, Validation, Experimentation.

Nusrat Rahman Aurna: Writing – review & editing, Experimentation, Validation.

Fayaza Islam: Writing – review & editing, Experimentation.

Mahdin Islam Ohi: Writing – review & editing, Experimentation, Validation.

Md. Khalid Mahbub Khan: Writing – review & editing, Methodology.

Michael Georgiades: Supervision, Writing – review & editing.

References

[1] C. Boldrini, K. Lee, M. Önen, J. Ott, E. Pagani, Opportunistic networks, Computer Communications 48 (2014) 1–4, opportunistic networks. doi:https://doi.org/10.1016/j.comcom.2014.04.007.
URL https://www.sciencedirect.com/science/article/pii/S0140366414001431

[2] V. F. Mota, F. D. Cunha, D. F. Macedo, J. M. Nogueira, A. A. Loureiro, Protocols, mobility models and tools in opportunistic networks: A survey, Computer Communications 48 (2014) 5–19.

[3] K. Scott, S. C. Burleigh, Bundle Protocol Specification, RFC 5050 (Nov. 2007). doi:10.17487/RFC5050.
URL https://www.rfc-editor.org/info/rfc5050

[4] S. Kurkowski, T. Camp, M. Colagrosso, Manet simulation studies: the incredibles, ACM SIGMOBILE Mobile Computing and Communications Review 9 (4) (2005) 50–61.

[5] A. Keranen, Opportunistic network environment simulator, Special Assignment report, Helsinki University of Technology, Department of Communications and Networking (2008).

[6] N. Chakchouk, A survey on opportunistic routing in wireless communication networks, IEEE Communications Surveys & Tutorials 17 (4) (2015) 2214–2241.

[7] D. Parmenter, Key performance indicators: developing, implementing, and using winning KPIs, John Wiley & Sons, 2015.

[8] R. Sachdeva, A. Dev, Review of opportunistic network: Assessing past, present, and future, International Journal of Communication Systems 34 (11) (2021) e4860. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.4860, doi:https://doi.org/10.1002/dac.4860.
URL https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.4860

[9] S. Trifunovic, S. T. Kouyoumdjieva, B. Distl, L. Pajevic, G. Karlsson, B. Plattner, A decade of research in opportunistic networks: Challenges, relevance, and future directions, IEEE Communications Magazine 55 (1) (2017) 168–173. doi:10.1109/MCOM.2017.1500527CM.

[10] D. Srivastava, An introduction to data visualization tools and techniques in various domains, International Journal of Computer Trends and Technology 71 (4) (2023) 125–130.

[11] X. Zeng, R. Bagrodia, M. Gerla, Glomosim: a library for parallel simulation of large-scale wireless networks, in: Proceedings of the twelfth workshop on Parallel and distributed simulation, 1998, pp. 154–161.

[12] T. Issariyakul, E. Hossain, Introduction to network simulator 2 (ns2), in: Introduction to network simulator NS2, Springer, 2008, pp. 1–18.

[13] A. Varga, R. Hornig, An overview of the omnet++ simulation environment, in: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops, 2008, pp. 1–10.

[14] G. F. Riley, T. R. Henderson, The ns-3 network simulator, in: Modeling and tools for network simulation, Springer, 2010, pp. 15–34.

[15] L. F. Perrone, C. S. Main, B. C. Ward, Safe: Simulation automation framework for experiments, in: Proceedings of the 2012 Winter Simulation Conference (WSC), IEEE, 2012, pp. 1–12.

[16] P. A. B. Bautista, L. F. Urquiza-Aguiar, L. L. Cárdenas, M. A. Igartua, Large-scale simulations manager tool for omnet++: Expediting simulations and post-processing analysis, IEEE access 8 (2020) 159291–159306.

[17] R. Sachdeva, A. Dev, Review of opportunistic network: Assessing past, present, and future, International Journal of Communication Systems 34 (11) (2021) e4860.

[18] J. Dede, A. Förster, E. Hernández-Orallo, J. Herrera-Tapia, K. Kuladinithi, V. Kuppusamy, P. Manzoni, A. bin Muslim, A. Udugama, Z. Vatandas, Simulating opportunistic networks: Survey and future directions, IEEE Communications Surveys & Tutorials 20 (2) (2017) 1547–1573.

[19] V. Kuppusamy, U. M. Thanthrige, A. Udugama, A. Förster, Evaluating forwarding protocols in opportunistic networks: Trends, advances, challenges and best practices, Future Internet 11 (5) (2019). doi:10.3390/fi11050113.
URL https://www.mdpi.com/1999-5903/11/5/113

[20] N. Papanikos, D.-G. Akestoridis, E. Papapetrou, Adyton: A network simulator for opportunistic networks, ZSCC (2015).

[21] A. Vahdat, D. Becker, et al., Epidemic routing for partially connected ad hoc networks (2000).

[22] T. Spyropoulos, K. Psounis, C. S. Raghavendra, Spray and wait: an efficient routing scheme for intermittently connected mobile networks, in: Proceedings of the 2005 ACM SIGCOMM Workshop on Delay-Tolerant Networking, WDTN '05, Association for Computing Machinery, New York, NY, USA, 2005, p. 252–259. doi:10.1145/1080139.1080143.
URL https://doi.org/10.1145/1080139.1080143

[23] D. Yulianti, S. Mandala, D. Nasien, A. Ngadi, Y. Coulibaly, Performance comparison of epidemic, prophet, spray and wait, binary spray and wait, and prophetv2, Faculty of Computing, Universiti Teknologi Malaysia (2020).

[24] K. Fall, S. Farrell, Dtn: an architectural retrospective, IEEE Journal on Selected areas in communications 26 (5) (2008) 828–836.

[25] S. Wang, X. Wang, J. Huang, R. Bie, X. Cheng, Analyzing the potential of mobile opportunistic networks for big data applications, IEEE Network 29 (5) (2015) 57–63.

[26] S. Rashidibajgan, T. Hupperich, Improving the performance of opportunistic networks in real-world applications using machine learning techniques, Journal of Sensor and Actuator Networks 11 (4) (2022) 61.

[27] S. Bhattacharjee, R. Chatterjee, T. Pal, S. DasBit, Implementing multicasting and broadcasting of multimedia data in one simulator, in: Proceedings of the 10th EAI International Conference on Simulation Tools and Techniques, 2017, pp. 48–56.

[28] P. Gawłowicz, A. Zubow, Ns-3 meets openai gym: The playground for machine learning in networking research, in: Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, 2019, pp. 113–120.

[29] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, X. Hei, Ns3-ai: Fostering artificial intelligence algorithms for networking research, in: Proceedings of the 2020 Workshop on ns-3, 2020, pp. 57–64.

[30] G. Wiggins, G. Cage, R. Smith, S. Hitefield, M. McDonnell, L. Drane, J. McGaha, M. Brim, M. Abraham, R. Archibald, et al., Best practices for documenting a scientific python project, Tech. rep., Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States) (2023).

[31] M. Haklay, P. Weber, Openstreetmap: User-generated street maps, IEEE Pervasive computing 7 (4) (2008) 12–18.

[32] D. F. Marble, Geographic information systems: an overview, Introductory readings in geographic information systems 3 (4) (1990) 8.

[33] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, G. Woodhull, Graphviz and dynagraph—static and dynamic graph drawing tools, in: Graph drawing software, Springer, 2004, pp. 127–148.

[34] J. Racine, gnuplot 4.0: a portable interactive plotting utility (2006).

[35] J. L. Gustafson, Reevaluating amdahl's law, Communications of the ACM 31 (5) (1988) 532–533.

[36] G. James, D. Witten, T. Hastie, R. Tibshirani, J. Taylor, Linear regression, in: An introduction to statistical learning: With applications in python, Springer, 2023, pp. 69–134.

[37] L. E. Peterson, K-nearest neighbor, Scholarpedia 4 (2) (2009) 1883.

[38] Y.-Y. Song, Y. Lu, Decision tree methods: applications for classification and prediction, Shanghai archives of psychiatry (2015).

[39] L. Breiman, Random forests, Machine learning 45 (1) (2001) 5–32.

[40] A. Keränen, J. Ott, T. Kärkkäinen, The one simulator for dtn protocol evaluation, in: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, Simutools '09, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, 2009. doi:10.4108/ICST.SIMUTOOLS2009.5674.
URL https://doi.org/10.4108/ICST.SIMUTOOLS2009.5674

[41] M. L. Waskom, Seaborn: statistical data visualization, Journal of open source software 6 (60) (2021) 3021.

[42] P. Barrett, J. Hunter, J. T. Miller, J.-C. Hsu, P. Greenfield, matplotlib–a portable python plotting package, in: Astronomical data analysis software and systems XIV, Vol. 347, 2005, p. 91.

[43] C. H. Wagner, Simpson's paradox in real life, The American Statistician 36 (1) (1982) 46–48.

[44] V. Jacobson, M. Mosko, D. Smetters, J. Garcia-Luna-Aceves, Content-Centric Networking, Whitepaper, Palo Alto Research Center (2007) 2–4.