

KADCY An AI-Assisted Real-Time Network Protocol Simulation and Visualization System

Ms. Khushi Vinod Bhandari¹, Ms. Kavana D N², Ms. Chandana C³

^{1,2,3}*Department of Information Science and Engineering, Shree Devi Institute of Technology*

Abstract—This paper describes KADCY, an AI-assisted web-based network protocol simulation and visualization framework that enables easier observation, inspection, and explanation of packet-level behavior in real time. The system uses a React-based interactive topology workspace and a Python server-authoritative simulation engine, plus persistent experiment records, live statistics, and an embedded networking assistant. Experiments can be constructed using a custom topology, application and transport protocols, packet count, speed, and packet ordering, with animated events including DNS resolution, connection setup, packet transmission, failures, retransmissions, and response delivery. The simulator handles TCP, UDP, HTTP, HTTPS, FTP, DNS, ICMP, and SMTP flows, including per-protocol control packets, MTU-aware fragmentation, shortest-path routing over the user topology, stochastic failure events and retransmission behavior for reliable protocols. KADCY provides deterministic, state-aware explanations of packet status, path, drops, queueing, statistics and networking concepts, plus an optional language-model bridge for richer, more natural responses. The work was inspired by prior work on network simulation, multi-view visualization, intelligent simulation, edge-AI and explainable AI, and represents an original contribution for an educational and experimental platform that unifies simulation, visualization, fault injection, analytics, and contextual assistance in one browser-based workflow.

Index Terms—Artificial intelligence, KADCY, network simulation, packet visualization, protocol analysis, real-time systems, topology, troubleshooting.

I. INTRODUCTION

Computer network simulation has emerged as a controlled way to study protocol behavior, routing, failures, and performance without the need for physical infrastructure. Past efforts have demonstrated the value of using simulation for evaluating wired and wireless network configurations and comparing protocol characteristics such as throughput and packet delivery ratio [6]. Distributed edge-AI research

similarly illustrates using event-driven network simulation to evaluate communication quality and real-time behavior in constrained environments [3]. More recent platform-oriented research highlights the need for modular simulation systems that integrate modeling, execution, analysis, and visualization rather than viewing simulation output as an isolated trace [11].

Visualization is of critical importance because event traces can grow unwieldy and require human inspection to reveal insights. Janacik, Lessmann, and Karch illustrate a synchronized multi-view approach that combined textual events with graphical network state and communication patterns, explicitly for faster protocol analysis [4]. Related work on web-based vehicular-network simulation similarly emphasizes user-friendly interfaces and processed visual outputs to make complex simulation workflows more accessible [8]. Virtual networking environments have demonstrated the potential of graphical interfaces to lower hardware and access barriers and enable interactive experimentation and multidimensional visualization [9].

The literature also highlights the potential for more intelligent and explainable approaches. Hybrid edge-cloud architectures place adaptive computation closer to the data source, while relying on centralized resources for broader analysis [1]. Explainable AI research highlights the need for intelligent systems to expose interpretable evidence for decisions, especially when the system operates in a dynamic and requires users to debug or validate behavior [2]. A custom MANET simulator highlights the benefits of combining modular simulation with AI-oriented routing experimentation and real-time visualization [7]. These directions highlight the need for an integrated environment in which network events are

not only simulated and visualized, but also interpreted in the context of the current run.

The proposed KADCY system addresses this need, through a browser-based architecture that combines a server-authoritative simulation engine, interactive topology editing, packet animation, live performance statistics, fault injection, persistent reports, and a state-aware networking assistant. The contribution is intentionally system-oriented: the system does not seek to claim a new routing algorithm, but rather packages established networking concepts and simulation mechanisms into an interactive workflow that is intended for learning, debugging, experimentation, and observing protocol behavior.

II. RELATED WORK

A. Network Simulation and Protocol Evaluation

Studies based on the NS2 simulator establish a conventional experimental pattern: a topology is defined, packets are sent through simulated nodes, and performance indicators are calculated from simulation traces [6]. The edge-AI WSN study similarly uses simulation to evaluate event-driven communication through metrics such as throughput, packet delivery ratio, lifetime, and power consumption [3]. These studies support the value of controlled, repeatable network simulation but are not primarily designed around a modern browser interface with live explanatory assistance.

B. Visualization and Interactive Simulation

Multi-view visualization research demonstrated that synchronized representations can reduce the cognitive cost of inspecting network traces [4]. The virtual networking teaching literature further emphasizes visual device layouts, animated connections, charts, and interactive operations as mechanisms for practical learning [9]. The SDVN testbed work integrates simulation tools through a web application and explicitly produces processed, visualized data [8]. KADCY follows this interaction-centered direction but focuses on packet-level protocol phases and live state inspection across several common Internet protocols.

C. Intelligent and Explainable Network Platforms

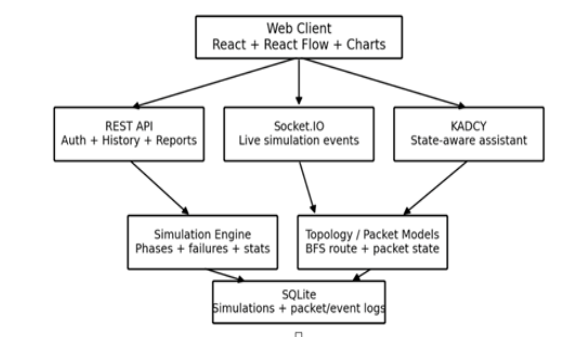
Research on intelligent simulation platforms combines modular front-end, back-end, and database

components with simulation execution, analysis, and visualization [11]. The custom MANET simulator explores the integration of AI-based routing within a modular and visual simulation environment [7]. Hybrid edge-cloud research highlights adaptive decision-making, scalable communication, and the value of combining localized and centralized intelligence [1]. Explainable AI research adds the requirement that an intelligent system provide interpretable information that can support debugging and human verification [2]. KADCY adopts these ideas at the application layer: it uses deterministic intent matching and live simulator state to explain what is happening, and optionally forwards live context to a language-model bridge.

D. Gap Addressed by the Proposed System

The referenced studies collectively cover simulation, visualization, interactive web access, intelligent computing, and explainability, but they address different application domains and toolchains. The opportunity explored in this paper is to combine these strands in a single educational network simulator: a user-defined topology, protocol-phase simulation, packet animation, failure injection, live statistics, persistent experiment history, and conversational explanation. The resulting system is therefore best understood as an integrative platform contribution rather than a replacement for established discrete-event or wireless-network simulators.

III. PROPOSED SYSTEM



A. Design Objectives

Fig. 1. High-level architecture of the KADCY network simulation system.

The proposed system was designed around the five objectives: (1) topology creation ease via drag-and-

drop interaction; (2) visualization of protocol behavior in terms of packet events; (3) expose failure and recovery behaviors rather than hide them; (4) computation and display of interpretable run statistics; and (5) contextual explanations referring to the current simulation state. The design also decouples the user interface from the authoritative simulation state such that movement, failure, statistics, and persistence can be computed on the server.

B. System Architecture

The frontend is implemented as a React 18 single-page application. React Flow provides the topology canvas, device nodes, connections and in-viewport packet animation; Chart.js renders live latency and throughput trends; Three.js/React Three Fiber provides the visual background and assistant presentation. The backend is implemented with Flask, Flask-Socket IO and SQL Alchemy, with SQLite providing persistent simulation, packet and event records. Report Lab is used for PDF report generation. REST endpoints handle auth, history and report retrieval, while Socket.IO provides the live bidirectional simulation channel.

The server-authoritative model is central to the design. On simulation start, the browser sends the URL, configuration and current topology graph to the backend. The Simulation Engine then performs packet movement, phase transitions, failure decisions, statistics sampling and event narration. The browser receives compact tick frames and renders them without having to make decisions about the simulation outcome independently. This separation is in line with the literature emphasis on modular simulation platforms and realistic interaction, while reducing inconsistency between visualization and simulation state [10], [11].

C. Interactive Topology Model

The topology editor supports 8 device categories: PC, Laptop, Switch, Router, Firewall, DNS Server, Server and Cloud. The backend converts the React Flow node and edge payload into an undirected graph and computes a shortest path with breadth-first search (BFS). Because the route is calculated from the user-provided graph, changing the topology changes the simulated path without changing the simulation engine implementation. Per-device next-hop information is emitted as a routing-table view to the frontend.

D. Packet Model and Lifecycle

Each packet is represented by a structured state object that contains an identifier, label, size, protocol, source, destination, path, priority, TTL, hop count, current segment, link progress, status, queue delay, latency,

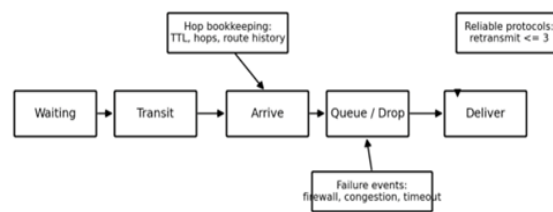


Fig. 2. Packet lifecycle used by the server-authoritative simulation engine.

attempt count, drop reason, checksum and route history. The lifecycle goes through waiting, transit, queued, delivered or dropped states. Hop arrival updates the route and TTL and router/cloud events can introduce queueing or drops. Reliable protocols create a new attempt when a data packet is dropped, up to three total attempts.

E. Protocol Phase Modeling

A finite phase machine is used to decompose the simulation into phases such as URL entry, DNS resolution, handshake/setup, request, packet transmission, server processing, response, and connection close. TCP, HTTP, HTTPS, FTP, and SMTP make use of TCP three-way handshake, whereas HTTPS also incorporates TLS-oriented control messages representing ClientHello, certificate exchange, and Finished, and UDP does not have connection establishment. ICMP makes use of four echo request/reply rounds, while DNS may be executed as a standalone simulation or as the name-resolution sub-process of another protocol.

IV. SIMULATION AND FAILURE MODEL

A. Packet Generation and Ordering

The engine implements protocol-specific profiles to generate data packet sizes. Examples include 536-1460 byte TCP data packets, 512-1400 byte UDP datagrams, and larger HTTP, HTTPS, FTP, and SMTP payloads. Data packets above the 1500-byte MTU are fragmented using a 1460-byte payload capacity, resulting in labels such as P4.1 and P4.2. In addition

to FIFO and LIFO ordering, packets can be launched in a random, small-to-large, large-to-small, or priority order. This diversity ensures that the process of queueing and packet ordering is visible to the experimenter.

B. Timing and Movement

The engine advances in 100 ms real time and uses a user-selected speed multiplier. Control packets and data packets have different movement factors so that handshake and management events complete before or with the data transfer phase. During transmission, data packets are launched at a fixed 0.6 simulation-second gap. The packets' progress is updated constantly from zero to one on each link, making the flow visible within the React Flow viewport.

C. Failure and Recovery Events

The engine implements a stochastic failure model that is intended to showcase common networking behaviors rather than implement a calibrated physical network. Data packets can be blocked by the firewall, dropped due to congestion or a buffer overflow, delayed in a router/cloud queue, or dropped due to a timeout or broken-link event. The engine also tracks the TTL expiration and supports manual failure injection for congestion and router failure. Under forced congestion, the probability of buffer/congestion drops and queueing increases. Reliable protocols retransmit a data packet that was dropped, whereas control packets are protected from stochastic data drops so that the phase machine can terminate deterministically.

V. KADCY INTELLIGENT ASSISTANT

A. Deterministic State-Aware Layer

KADCY is implemented as a networking assistant which has two layers: The default is deterministic, does not require an external model, and includes a networking knowledge base, synonym expansion for fuzzy-style intent matching, and live-state handlers. Questions such as "why was packet P3 dropped?", "show statistics", "show packet path" and "what is in transit?" are answered from the active Simulation Engine state. When a packet is dropped, KADCY can report the packet size, protocol, status, drop reason, location, attempts, latency, hops, queue delay, and route history, derived from the live object.

B. Explanation and Narration

During simulation, the assistant produces event narration for protocol phases, DNS resolution, packet departures, forwarding, congestion, drops, retransmissions, response delivery and end-of-run summaries. This design follows the usability goal of making event traces easier to interpret while reflecting the explainability theme in recent intelligent-system research [2]. Importantly, the assistant does not infer a hidden model decision - it explains explicit simulator state and deterministic rules. As a result, the explanations are closer to contextual system diagnostics than post-hoc model interpretation methods such as SHAP or LIME.

C. Optional Language-Model Layer

An optional language-model bridge can receive the current protocol, phase, resolved address, statistics, and recent event messages as context. In the case that the external call is unavailable or fails, the deterministic assistant is available. This fallback ensures that core simulator functionality is preserved without making the networked language service a single point of failure. The architecture reflects the broader literature interest in combining local and centralized intelligence while maintaining operational continuity [1], [11].

VI. IMPLEMENTATION

A. Frontend Implémentation

The frontend organizes the views for authenticating, displaying a topology, controlling and viewing a simulation, viewing statistics, performing network inspections, and using an assistant using separate React components. The shared state between components is encapsulated in a Simulation Context which holds the socket event handlers, packet positions (needed to render them in the React Flow viewport), statistics, narrative, stage, DNS data, and completion summary. The Packet Layer component renders the packet positions within the viewport, so they appear fixed relative to the topology nodes when the user is panning or zooming. The KADCY panel is rendered in a fixed position on top of the interface, so it does not cause the simulation canvas to resize when opened.

B. Backend Implementation

The backend exposes authentication, history/report, and question endpoints for KADCY, and uses Socket.IO to establish connections from the frontend. When a connection is made, the backend spawns a Simulation Engine instance for the user. The engine's tick loop advances the clock, launches packets, progresses links' progress, processes arrivals, advances the protocol's phase machine, samples statistics, and emits a compressed state frame. Once completed, the engine persists the summary, packet log, and assistant event log to an SQLite database.

C. Persistence and Reporting

All persistent records are scoped to the authenticated user. For each completed simulation, the URL/domain, protocol, configuration, and summary statistics are stored. Per data packet, the size, status, number of attempts, hops, queue delay, latency, drop reason, and route are persisted. Additionally, each KADCY event record stores the generated narrative with a timestamp. The system supports generating PDF, CSV, and plain-text event-log exports from a completed run, so it can be examined outside of the application.

VII. FUNCTIONAL EVALUATION AND DISCUSSION

Due to the artifact under investigation being a prototype application, and not a calibrated network emulator, the evaluation in this paper focuses on the functional coverage and observability, as opposed to the claims of physical-network accuracy. The implementation was inspected for compliance to reference-oriented design goals and the feature set encoded in the project source. Table I summarizes the main functional verification targets.

TABLE I. FUNCTIONAL VERIFICATION TARGETS

Functional area and implemented behavior
Topology - BFS path calculated from provided nodes and edges
Protocol flow - TCP-family SYN, SYN-ACK and ACK stages

HTTPS - TLS-oriented Hello, certificate and Finished events
Connectionless - UDP skips handshake; DNS may operate standalone
ICMP - Four echo request/reply rounds
Failures - Manual congestion and router-failure injection
Reliability - Up to three attempts for reliable data protocols
Observability - Packet inspector, timeline, routing table and charts
Persistence - SQLite simulation, packet and event logs
Assistant - Live packet, route, statistics and concept answers

A. Observability metrics

The engine calculates several run-level metrics. Packet loss is determined by counting the dropped packets launched. The average latency is calculated using the delivered data packets and their end-to-end delay. Throughput is computed using the delivered bytes and the simulation elapsed time. Queue length, in-transit count, utilization, and retransmission count also provide valuable insights. A custom health index is calculated using the loss, average latency, and queue length and classified as Excellent, Good, Degraded, or Poor according to the prototype thresholds.

These quantities are valuable for interactive teaching and comparative experiments because they relate visible events to summary behavior. For example, a forced congestion event would raise the queueing and potentially the loss, and a reliable protocol would exhibit retransmissions visible in both the animation and statistics. This is similar to prior work that evaluates simulation outputs through throughput, packet delivery, and related QoS metrics [3], [6], but extended to include packet-forensics and conversational explanation.

B. Discussion of Research Positioning

The primary research value of KADCY lies in integration. Traditional simulators remain more focused on detailed protocol studies and large-scale experimental benchmarking. The current system is

optimized for accessibility, visualization, and explanation. The platform therefore occupies a complementary space between a teaching-oriented virtual lab [9], a visualization tool [4], and an intelligent modular simulation environment [11]. The design also trends toward the modern tendency to combine simulation with AI-assisted interaction [1], [2], [7], but the current implementation uses AI for interpretation and assistance rather than a trained routing optimizer.

VIII. LIMITATIONS

Several limitations should be stated explicitly. First, the simulator models only abstractly the packet movement, without implementing the production TCP/IP stack, wireless channel, or physical network emulator. Second, BFS provides a shortest path for the unweighted topology graph, so no link capacity, latency, cost, or congestion are part of the path-selection objective. Third, the failure probabilities are heuristic implementation parameters; they have not been calibrated against the measured traces. Fourth, the DNS resolution uses the generated IPv4-style values for the simulation; it does not perform the authoritative DNS resolution. Fifth, the optional language-model layer depends on an external service; it should not be interpreted as the source of the simulator ground truth. Finally, the current study does not include a user study or a controlled statistical comparison against the NS2, NS3, OMNeT++, or Cooja tools.

IX. FUTURE WORK

The future work can augment the routing layer with the weighted shortest-path or adaptive routing methods and add wireless propagation and mobility models, including tracing or replaying the traces from established simulators. The assistant can be extended with retrieval over the experiment history, richer explanation provenance, and explicit confidence indicators. The evaluation can be formalized by comparing the metrics of KADCY against the reference tools for a comparable topology and workload and measuring the execution time, packet delivery ratio, latency, throughput, and usability. The student study could be controlled to measure whether the live explanation and synchronized visualization

improve the task completion or conceptual understanding, building upon the motivation found in the virtual network teaching research [9].

X. CONCLUSION

This paper describes KADCY, an AI assisted real-time network protocol simulation and visualization prototype which includes features such as custom topology construction, protocol-phase modeling, packet animation, failure injection, live metrics, persistence, reporting, and contextual network assistance. The design of this system draws upon research themes in network simulation [3], [6], intelligent simulation engines [7], [11], multi-view visualization [4], web-based simulation [8], virtual networking education [9], and explainable or adaptive AI [1], [2]. This platform's contribution is in its unified browser-based workflow where network events can be inspected, examined, and explained. The current scope of this platform reflects a balance between being an educational/experimental simulator, while not being a physically accurate network emulator. The described architecture and explicitly named limitations form a basis for potential future work in calibrated evaluation, richer routing models, and formal human centered evaluation.

ACKNOWLEDGMENT

The Author acknowledges the research literature and technical inspiration described in reference papers provided for this work. The implementation described in this manuscript reflects contributions based on the KADCY/PacketScope project source which was developed for the networking simulation project associated with this work.

REFERENCES

- [1] K. K. Ratra and D. K. Seth, "AI-driven hybrid edge-cloud architecture for real-time big data analytics and scalable communication for retail supply chains," in Proc. SoutheastCon 2025, 2025, doi: 10.1109/SOUTHEASTCON56624.2025.10971468.
- [2] I. Sahoo, M. Parikh, K. K. Ganeeb, G. A. Gajwani, B. Harve, P. Nutalapati, and R.

- Keshava, "Explainable AI for autonomous systems: A framework for trustworthy decision making," in Proc. 10th Int. Conf. on computer and communication engineering (ICCCE), 2025, doi: 10.1109/ICCCE66530.2025.11473362.
- [3] A. Prajapati and B. Banerjee, "Simulating distributed wireless sensor networks for edge-AI," in Proc. 5 th Conf. on cloud and internet of things (CIoT), 2022, doi: 10.1109/CIOT53061.2022.9766520.
- [4] P. Janacik, J. Lessmann, and M. Karch, "Multi-view communication visualization for wireless network simulations," in Proc. IEEE Int. Conf. on intelligent systems, modelling and simulation (ISMS), 2010, doi: 10.1109/ISMS.2010.43.
- [5] H. Huang, C. M. Davis, and K. R. Davis, "Real-time power system simulation with hardware devices through DNP3 in cyber-physical testbed," in Proc. IEEE Texas power and energy conference (TPEC), 2021, doi: 10.1109/TPEC51183.2021.9384947.
- [6] P. More, S. Rathi, and H. B. Thatte, "Generating network simulations and comparing different routing protocols using NS2," in Proc. 3rd asian conference on innovation in technology (ASIANCON), 2023, doi: 10.1109/ASIANCON58793.2023.10270073.
- [7] J. Badar and J. Papaj, "Development of a custom MANET simulator with enhanced integration of AI-based routing protocols," in Proc. 35 th Int. conf. radioelektronika (RADIOELEKTRONIKA), 2025, doi: 0.1109/RADIOELEKTRONIKA65656.2025.11008424.
- [8] F. Afif, Fadjrianah, E. Mulyana, N. R. Syambas, and G. N. Nurkahfi, "Data processing and visualization in software defined vehicular network tested," in Proc. 18 th Int. Conf. on Telecommunication systems, services, and applications (TSSA), 2024, doi: 10.1109/TSSA63730.2024.10864278.
- [9] W. Wang and X. Liu, "Application of virtual simulation experiments for undergraduate networking teaching," in Proc. 4 th Int. Conf. on intelligent autonomous systems (ICoIAS), 2021, doi: 10.1109/ICoIAS53694.2021.00042.
- [10] W. Li, C. Wang, Y. Hu, W. Zhang, and F. Wang, "Simulation system design of unmanned aerial vehicle swarm," in Proc. IEEE Int. Conf. on unmanned systems (ICUS), 2022, doi: 10.1109/ICUS55513.2022.9986772.
- [11] C. Gao, B. Li, and Z. Dai, "Design and implementation of intelligent computing simulation platform for future network architecture," in Proc. Int. Conf. on power, electrical engineering, electronics and control (PEEEEC), 2024, doi:10.1109/PEEEEC63877.2024.00123.