Work in Progress: Packelytics - A Computer Networks Lab Tool For Enhanced Protocol Experimentation and Learning

Dr. Hamid S Timorabadi P.Eng., University of Toronto

Hamid Timorabadi received his B.Sc, M.A.Sc, and Ph.D. degrees in Electrical Engineering from the University of Toronto. He has worked as a project, design, and test engineer as well as a consultant to industry. His research interests include the applications in Engineering Education.

Mr. Ryan Seto, University of Toronto

Ryan Seto graduated from the University of Toronto with a Bachelor's degree in Computer Engineering. He has experience developing driver code, creating new machine learning models, alongside designing and manufacturing robots.

Hiu Lam Chloe Soong, University of Toronto

Chloe Soong graduated from the University of Toronto with a Bachelor's Degree in Computer Engineering. Her technical background spans full-stack user interface design to developing scan network architectures and C++ simulations for hardware verification.

Mr. Alex Yuan, University of Toronto

Alex Yuan is a fourth-year Computer Engineering student at the University of Toronto. His academic interests lie at the intersection of computer networking and hardware-based experimentation. Through his work on the Packelytics project, he focuses on developing experiential learning tools that bridge the gap between theoretical protocol logic and physical network implementation.

Andrew Bradley Cheung, University of Toronto

Andrew Bradley Cheung graduated from the University of Toronto with a degree in Computer Engineering. His primary interests are in computer vision and machine learning.

Work in Progress: Packelytics - A Computer Networks Lab Tool For Enhanced Protocol Experimentation and Learning

Abstract

Computer networking concepts are often difficult for students to connect to real system behavior when instruction relies primarily on theory or high-level simulations. This work presents Packelytics, a work-in-progress laboratory platform designed to support experiential learning in undergraduate networking courses through integrated protocol implementation, visualization, and hardware-based experimentation.

Unlike existing tools such as Wireshark or network simulators, Packelytics requires students to directly implement protocol logic by manipulating packet structures and state transitions, while receiving immediate visual feedback through animated packet exchanges. A configurable hardware testbench further enables deployment of these implementations on physical network topologies, helping connect abstract protocol behavior to real-world operation.

A pilot study involving two student cohorts evaluated the platform using both self-reported measures and a preliminary objective assessment of conceptual understanding. Results suggest improvements in student confidence, along with some measurable gains in identifying and reasoning about TCP handshake behavior. While the current findings are exploratory, they indicate that combining coding, visualization, and hardware interaction may support both perceived and demonstrated learning.

Packelytics offers a flexible framework for bridging theoretical instruction and hands-on protocol experimentation. Ongoing work will expand protocol coverage and include more structured evaluation methods to better understand its educational impact.

1.0 Introduction

Computer networks are a fundamental component of modern technology and are foundational to disciplines such as electrical and computer engineering, computer science, and information technology. Allowing communication, data transfer, and access to information on a global scale, understanding networking concepts is vital for students who plan on entering a technology-based industry. As these systems become more complex, the underlying mechanisms are often recognized by students as abstract and challenging to understand, especially when instruction relies on theoretical models and static representations.

A key challenge in networking education is reconciling the conceptual approach with practical comprehension. While students may learn protocol interactions, packet formats, and Open Systems Interconnection (OSI) layers through textbooks and lectures, they often struggle with attaching it to real-world behaviour. Existing tools, such as network simulators and visual aids, can provide high-level views of system-level interactions but simplify the inner workings of a protocol [1]. On the other hand, tools such as Wireshark expose real-time traffic but can overwhelm new learners due to the volume of data and lack of instructional support [2, 3].

To bridge this gap, Packelytics has been developed as an interactive lab tool to allow students to actively design, implement, and test networking protocols across the OSI layers, while receiving immediate visual feedback. The tool encourages students to experiment directly with protocol logic and packet construction through two integrated components: a software development environment (SDE) and a hardware testbench rig (HTR).

The software environment provides a set of guided lab instructions, a built-in code editor, and animated visualizations that illustrate packet exchanges for protocols such as the transmission control protocol (TCP) [4]. This combination allows students to write protocol logic directly within the environment, iteratively test their implementations, and view animations that can make difficult ideas concrete to learners. Complementing the software side, the hardware testbench allows students to construct physical network topologies to generate diverse test cases and test their logic on hardware. By deploying their protocol implementations onto these network topologies, students can observe how their designs behave under realistic conditions. Images of the SDE and HTR can be found in *Appendix A*.

Incorporating software experimentation and hardware validation aims to support active learning by allowing students to explore networking concepts through experimentation rather than observation. The lab activities allow students to improve their problem-solving and debugging skills which are critical for academic and professional success [5]. By linking coding, visualizations, and hands-on testing, the tool aims to support a deeper understanding of network protocol behaviour across layers.

2.0 Lab Design: Protocol Implementation Fundamentals

The lab modules are specifically designed to address the "Theory-Practice Gap" by requiring students to move beyond passive observation to active protocol development [6]. Each lab focuses on a specific network layer protocol. For example, TCP in the transport layer or the Address Resolution Protocol (ARP) in the link layer [7]. Labs do this through four core coding tasks: *decapsulate(), validateHeader()*, *buildResponseHeader()*, and *encapsulate()*.

Across all labs, students must implement *decapsulate()* and *encapsulate()* functions to handle raw byte streams and convert them into protocol headers. This requirement forces students to master network byte order (Big-Endian) and memory alignment, making abstract header fields tangible in a programming context. This is a novel coding task introduced by Packelytics in networking courses, as our institute does not cover this operation within computer networks and instead only focuses on implementing protocol logic. Students will also have to develop the functions *buildResponseHeader()* and *validateHeader()* to programmatically handle core protocol logic like establishing a connection in TCP or the discovery process in ARP. For instance, students must logically differentiate between different TCP packets (SYN-ACK vs SYN vs ACK) based on incoming header flags. This task allows students to put the protocol logic learned in class into practice.

Additionally, by separating labs in different layers, Packelytics clarifies the specific duties of each protocol layer. Students can see how the link layer handles physical address resolution (ARP) while the transport layer ensures reliable data transmission (TCP), reinforcing the modular nature of the OSI model.

Lastly, the integration of an animation processor allows students to visualize their code immediately. If the student discovers an incorrect response in the animation that does not match their understanding of network protocol functionality, the animation provides a concrete error representation, fostering critical problem-solving skills essential for professional success [8].

3.0 Technical Design and Educational Rationale

The technical architecture of Packelytics is a three-tier system (Frontend, Middleware, and Backend). They are engineered to bridge the "Theory-Practice Gap" by transforming abstract networking concepts into tangible experimental learning [9]. The system is designed for a plug-and-play experience, ensuring that students spend their lab time implementing procedural logic rather than troubleshooting equipment maintenance.

3.1 Standardized Implementation Framework

The platform focuses on a modular software structure that mirrors professional networking practices while reinforcing core curriculum objectives.

Table 1. Lab Coding Tasks and Their Purpose

Function Names	Pedagogical Purpose & Functionality
decapsulate()	Teaches bit-level manipulation and network byte order (Big-Endian) conversion by mapping raw bytes to structured headers.
validateHeader()	Encourages critical reasoning by requiring students to implement integrity checks on incoming data.
buildResponseHeader()	Focuses on protocol state machine logic, such as differentiating between TCP SYN-ACK and ACK packets.
encapsulate()	Reinforces the serialization process, ensuring that students can translate logic back into a network-ready byte stream.

3.2 Hardware-Software Integration for Active Learning

The Hardware Testbench Rig (HTR) provides the tactile engagement often missing from purely simulated environments [10]. While simulators are effective for conceptual theory, physical hardware is more effective for imparting concrete problem-solving skills and critical thinking.

- **Tactile Engagement:** By manually wiring connections between routers and switches, students visualize how physical topology changes alter data flow.
- **Visual-Logic Feedback:** The Animation Processor provides real-time visual feedback that stabilizes the student's learning trajectory, helping them identify whether a failure is due to their logic or a physical connection error [1].

4.0 Results

The evaluation follows a pre/post intervention design combining perception-based measures with an initial objective assessment of conceptual understanding. While the current study is exploratory, it provides a basis for future controlled comparisons.

To evaluate the effectiveness of the approach, two cohorts of participants were selected to complete the first lab. Cohort 1 consisted of students with prior experience in networking, while Cohort 2 included students with little or no prior exposure. Participants completed

surveys before and after the lab to capture their perceptions of learning across key components, including the coding tasks, hardware interaction, and visualizations.

To assess overall understanding, students were also asked before and after the lab to describe their understanding of the TCP handshake using a structured set of scored responses, as shown in Figures 1 and 2.

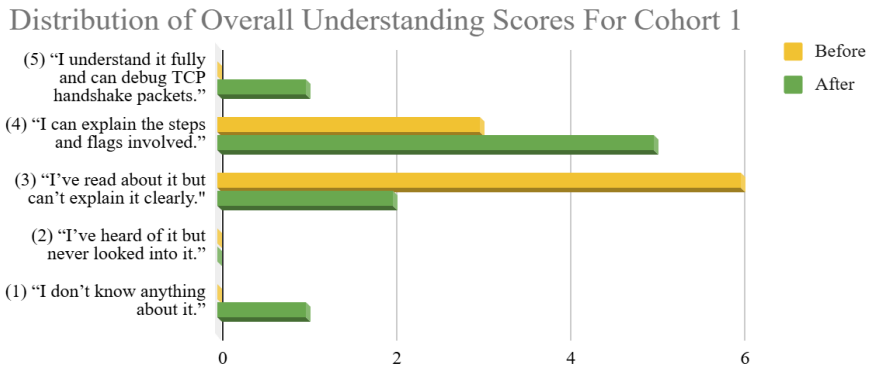

Figure 1. Distribution of overall understanding scores for cohort 1

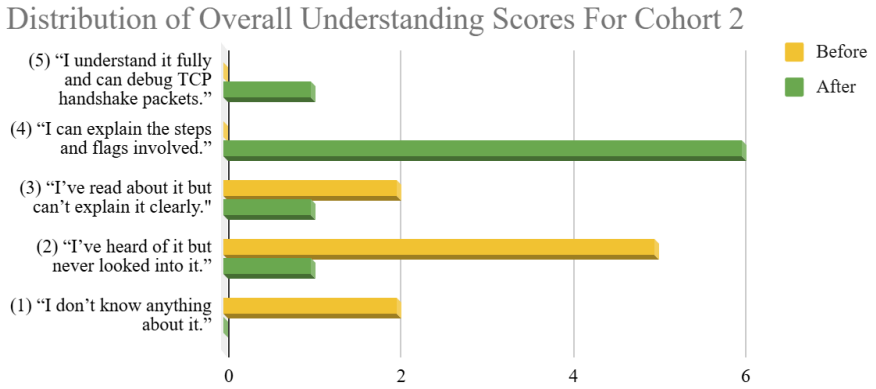

Figure 2. Distribution of overall understanding scores for cohort 2

As indicated by the survey results, the TCP handshake lab was associated with increased self-reported understanding of the content in both cohorts, with a greater apparent impact in the second cohort. In particular, there appear to be relationships between the perceived impact of the coding task, hardware, and animations. The impacts of these three features are discussed below.

4.1 Coding Task Impact

Much like the overall understanding above, the impact of the coding task was measured through students' selection from one of five score options for two questions. The first question asked whether students believed the coding task enhanced their understanding of the TCP handshake. The second question focused specifically on the learning impact of the *encapsulate()* and *decapsulate()* functions, as this represents a novel coding task introduced in the lab. The scoring system, distributions, and average scores for both cohorts and both questions are shown below.

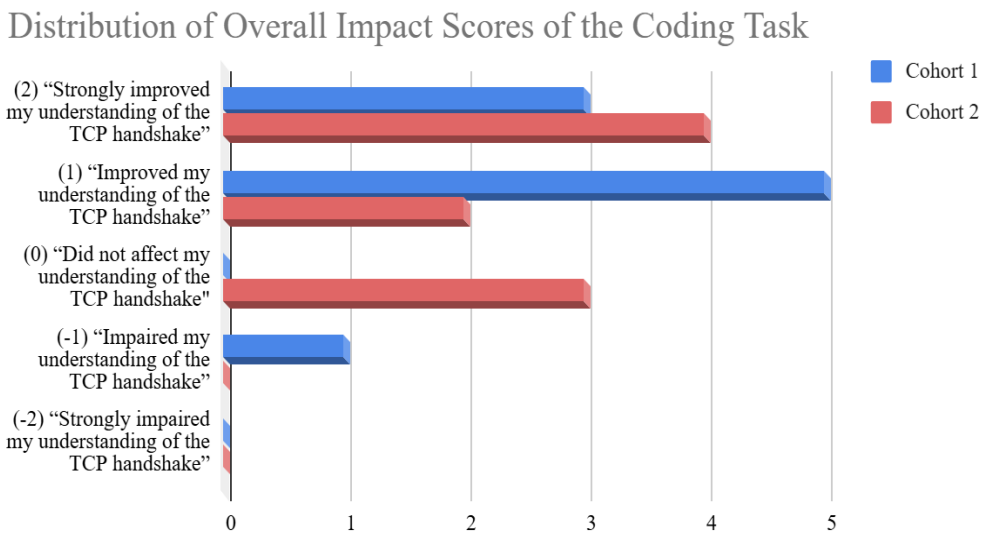

Figure 3. Distribution of overall impact scores of the coding task

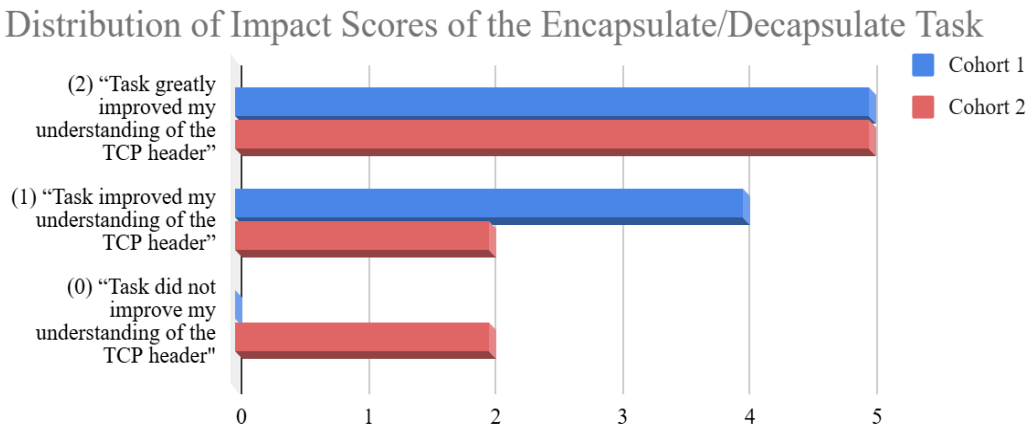

Figure 4. Distribution of impact score of the encapsulate/decapsulate task

As shown in Figures 3 and 4, the coding task appears to support improvements in students' understanding of the TCP handshake, as well as their ability to interpret packet headers through the encapsulate and decapsulate tasks.

4.2 Hardware Module Impact

Since only one set of hardware modules had been developed at the time of the study, a smaller subset of participants was able to test the module directly. In total, three participants used the hardware module, with one from Cohort 1 and two from Cohort 2. All three participants indicated that the hardware contributed positively to their understanding.

The remaining participants were asked whether they believed using hardware would improve their understanding of computer networks. The distribution of responses per cohort is shown in Figure 5.

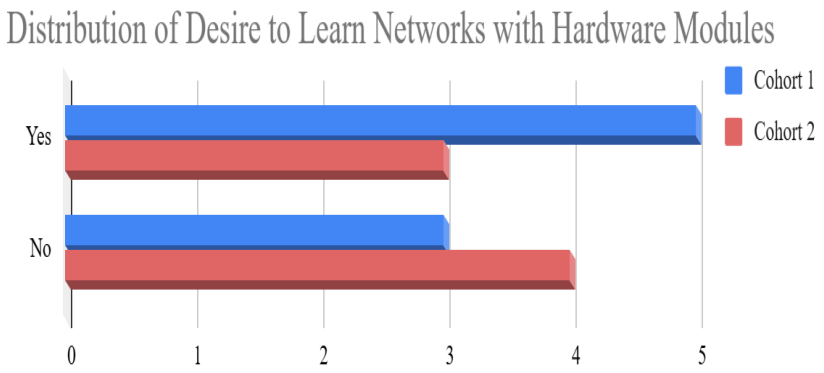

Figure 5. Distribution of desire to learn networks with hardware modules

These results suggest that a majority of students who did not use the hardware modules would have preferred to, while those who used the modules found them helpful. Given the small sample size, further testing with a larger group is needed to better understand the impact of the hardware component.

4.3 Animation Impact

Three questions were used to evaluate the impact of the animations. The first asked whether students felt the animations supported their learning. As shown in Figure 6, responses indicate a generally positive perception of the animations.

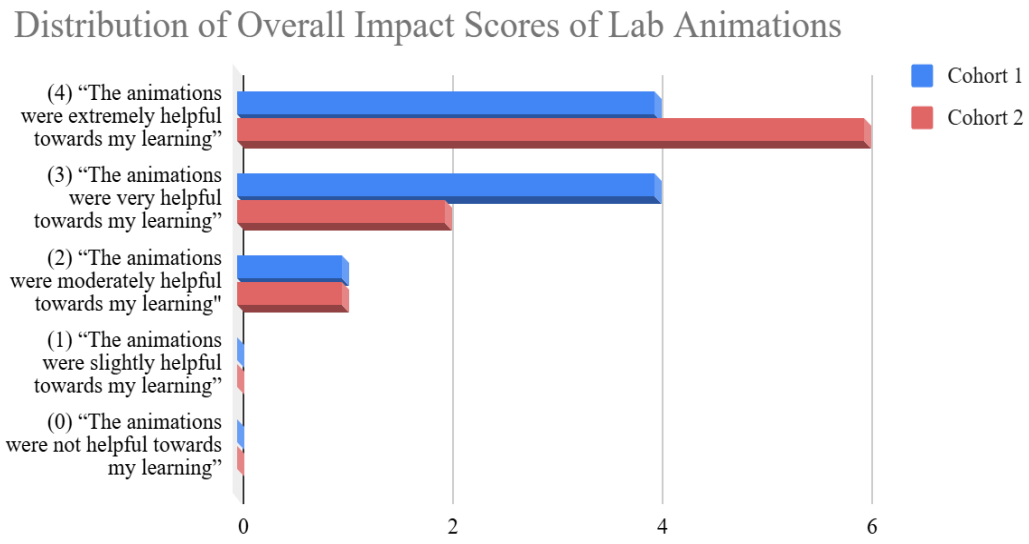

Figure 6. Distribution of overall impact scores of lab animations

The second question asked whether the lab would have been more difficult to complete without animations. The results, shown in Figure 7, suggest that animations make the lab somewhat easier, although the effect appears modest. One possible explanation is that the TCP handshake represents a relatively simple concept, and the impact of animations may be more pronounced for more complex topics such as broadcast or multicast communication.

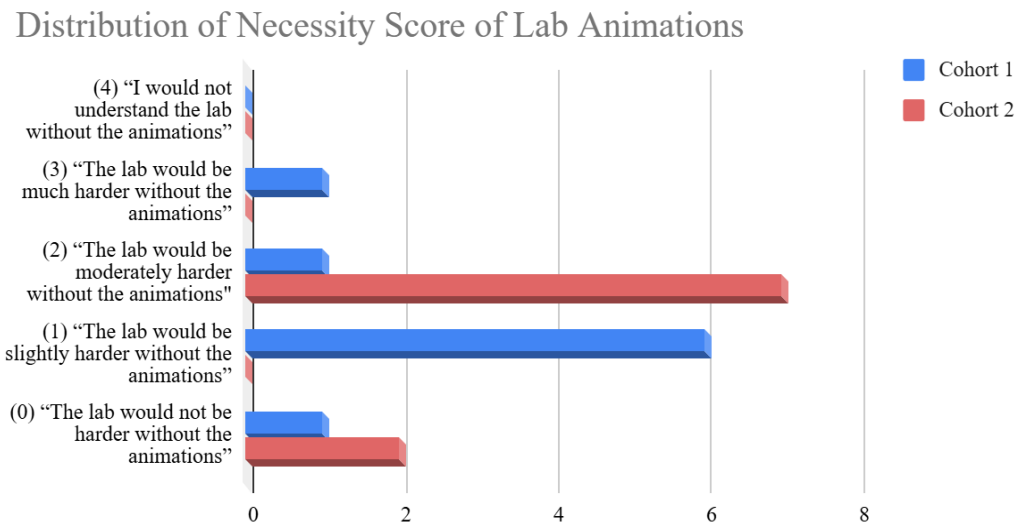

Figure 7. Distribution of necessity score of lab animations

The third question asked whether students would prefer to learn networking concepts using animations in the future. The results, shown in Figure 8, indicate a strong preference for incorporating animations, even though their impact on perceived difficulty was limited in this case.

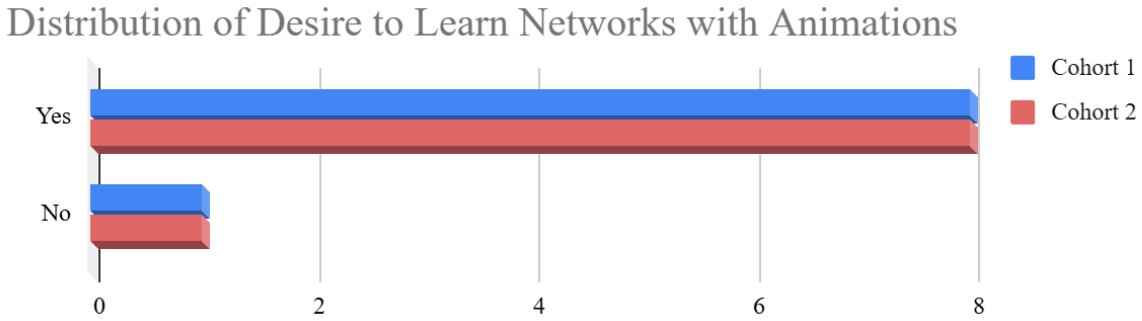

Figure 8. Distribution of desire to learn networks with animations

5.0 Limitations and Future Work

The current findings are based on a limited sample size and rely primarily on student self-reported confidence and perceived learning gains. While student perception is a useful indicator of engagement and usability, it is acknowledged that learners may overestimate their level of understanding.

Future work will include the implementation of objective performance measures for both pre- and post-intervention assessments. This may involve the use of concept inventories, as well as comparisons against a control group using traditional simulation tools.

In addition, input from instructors and domain experts in computer networking will be considered to help assess the alignment of the platform with current educational and professional expectations.

Alongside these evaluation efforts, ongoing development will focus on integrating additional protocols to further explore the scalability of the platform and expand its use across different areas of networking instruction.

6.0 Conclusion

Packelytics is a work-in-progress computer networking lab tool intended to support student understanding of networking concepts across the OSI layers. By allowing students to directly work with protocol logic and packet construction, Packelytics aims to connect theoretical instruction with practical application in an educational setting.

A key aspect of Packelytics is the combination of a software development environment (SDE) with a configurable hardware testbench (HTR). The SDE provides structured support through guided lab activities and a coding environment, along with visualizations that help make abstract protocol behavior easier to interpret. In parallel, the HTR enables students to deploy and test their implementations on physical network topologies, helping reinforce theoretical concepts through hands-on experimentation.

While this paper focuses on the initial transport-layer implementation of Packelytics, the platform was designed to be expandable to additional protocols and networking concepts. As part of this ongoing development, an ARP module has recently been implemented, extending the platform to support instruction at the network and link layers. This module has also been integrated with the hardware testbench, allowing students to construct physical topologies and observe packet-level behavior, including the broadcasting nature of ARP communication. These developments provide additional opportunities for more advanced and interactive lab experiences.

At the same time, the current study is exploratory and limited in scope. Future work will focus on broader classroom deployment and more structured evaluation of learning outcomes, including the use of objective performance measures alongside student feedback. These efforts will help provide a clearer understanding of the platform’s impact and guide further refinement.

Overall, by integrating coding experimentation, visualization, and physical testing, Packelytics offers a practical approach to teaching computer networking. Continued development and evaluation will help determine its effectiveness in supporting both conceptual understanding and applied skills in engineering education.

7.0 References

[1] K. J. Turner and I. A. Robin, “An interactive visual protocol simulator,” *Computer Standards & Interfaces*, vol. 23, no. 4, pp. 279–310, Sep. 2001, doi: https://doi.org/10.1016/s0920-5489(01)00081-2.

[2] Muhammad Azizur Rahman and Algirdas Pakstas, “Tools and Techniques for Teaching and Research in Network Design and Simulation,” *SN computer science*, vol. 4, no. 3, Mar. 2023, doi: https://doi.org/10.1007/s42979-023-01684-6.

[3] “Mastering Wireshark: A Comprehensive Tutorial for Network Analysis,” *networks2008.org*. https://networks2008.org/wireshark/wireshark-tutorial/#steep-learning-curve-for-beginners-due-to-the-complexity-of-network-protocols-and-packet-analysis

[4] J. Postel, "Transmission Control Protocol," RFC 793, Sept. 1981.

[5] J. Glassey, S. K. Gupta, and J. M. Escola, "Artificial intelligence and machine learning in chemical engineering education: A review," *Education for Chemical Engineers*, vol. 45, pp. 28–42, Oct. 2023, doi: 10.1016/j.ece.2023.07.001.

[6] E. Arteaga, R. Biesbroek, J. Nalau, and M. Howes, “Across the Great Divide: A Systematic Literature Review to Address the Gap Between Theory and Practice,” *SAGE Open*, vol. 14, no. 1, Jan.–Mar. 2024, Art. no. 21582440241228019, doi: 10.1177/21582440241228019.

[7] D. C. Plummer, "An Ethernet Address Resolution Protocol," RFC 826, Nov. 1982.

[8] Q. Hao *et al.*, “Towards understanding the effective design of automated formative feedback for programming assignments,” *Computer Science Education*, vol. 32, no. 1, pp. 105–127, Jan. 2022, doi: 10.1080/08993408.2020.1860408.

[9] M. A. Rahman and A. Pakstas, "Tools and Techniques for Teaching and Research in Network Design and Simulation," *SN Comput. Sci.*, vol. 4, no. 3, p. 269, Mar. 2023, doi: 10.1007/s42979-023-01684-6.

[10] J. Ma and J. V. Nickerson, "Hands-on, simulated, and remote laboratories: A comparative literature review," *ACM Comput. Surv.*, vol. 38, no. 3, p. 7, Sep. 2006, doi: 10.1145/1132960.1132961.

[11] M. Hanita, D. Ansel, and K. Shakman, Matched-Comparison Group Design: An Evaluation Brief for Educational Stakeholders, Education Development Center, Boston, MA, USA, White Paper, Jan. 2017. Available: https://www.edc.org/sites/default/files/uploads/matched_comparison_group_design.pdf

[12] J. Zeng and R. Wang, "A Survey of Causal Inference Frameworks," Sep. 2022, arXiv:2209.00869. [Online]. Available: https://arxiv.org/abs/2209.00869

Appendix A

This appendix includes sample images of the Software Development Environment (SDE) and the Hardware Testbench Rig (HTR), illustrating key components of the platform used in the lab activities.

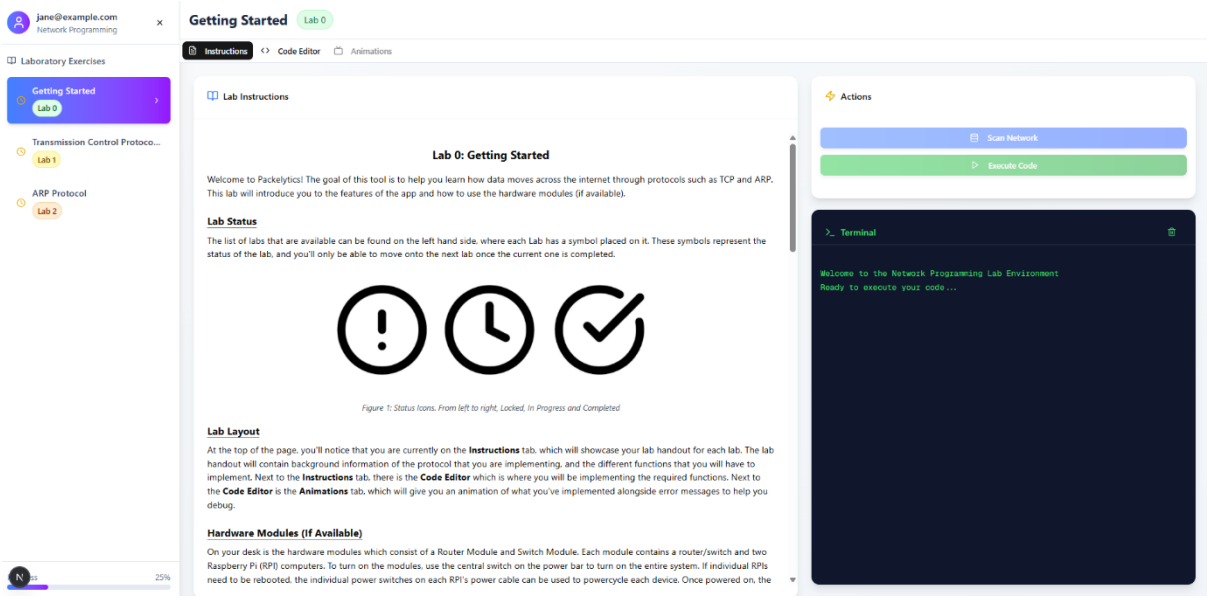

Figure A1. SDE instructions page

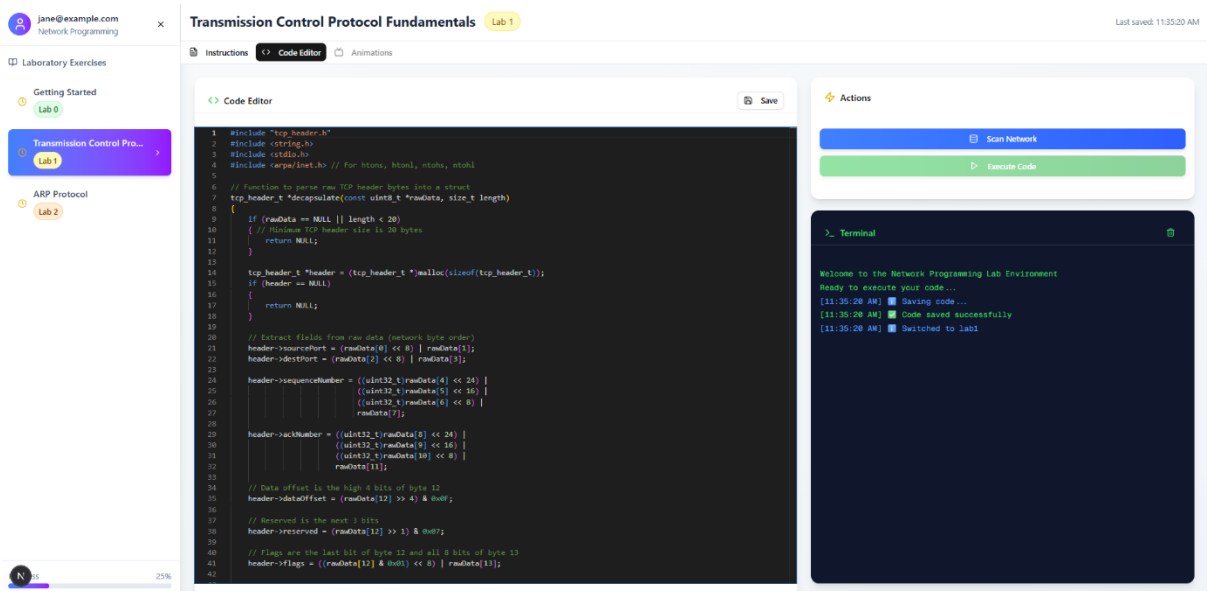

Figure A2. SDE code editor

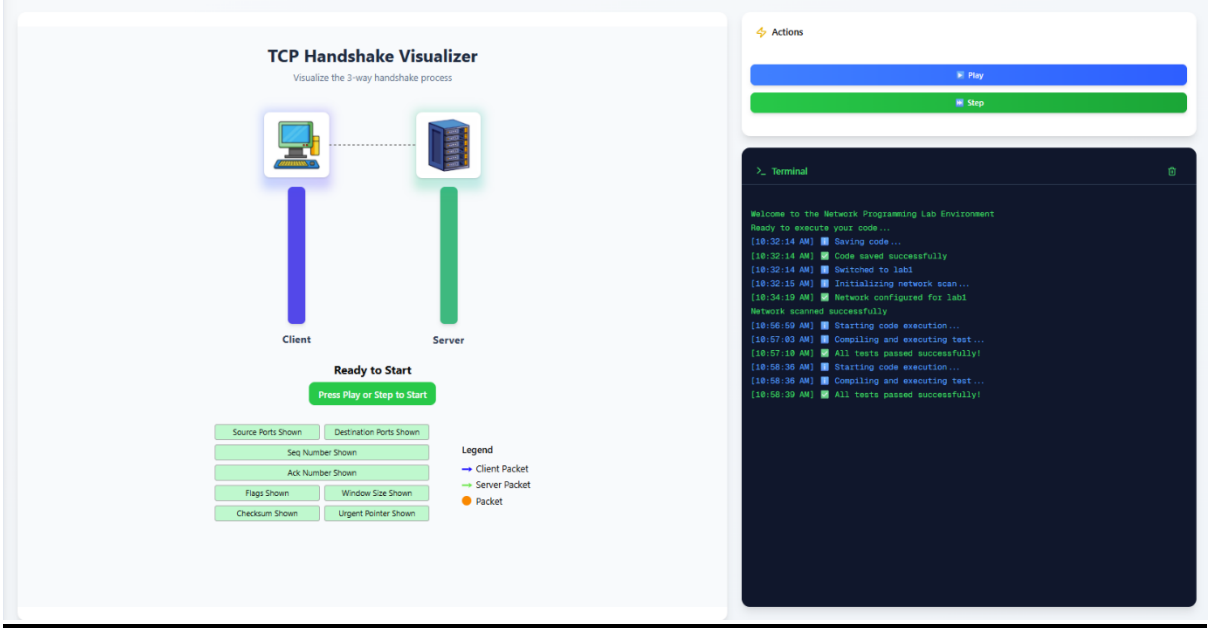

Figure A3. SDE Lab 1 animations page

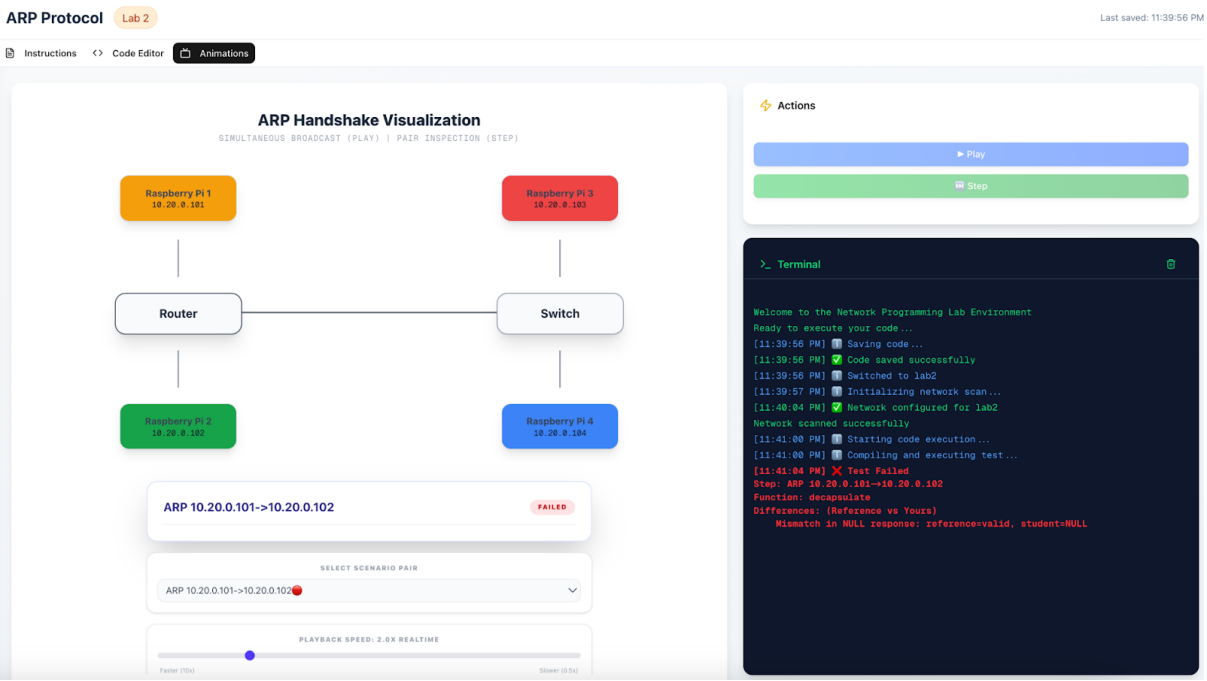

Figure A4. SDE Lab 2 animations page

Figure A5. Hardware testbench rig