# The Protocol Overlords

Jon Crowcroft, Andrew Moore, Richard Mortier
University of Cambridge

## Abstract

LLM-assisted coding separates the *what* from the *how* of software development. For networking education, this is an opportunity, not a threat: it allows students to engage with the higher-level concerns of protocol design, verification, and interoperability, rather than spending their effort on language quirks and bit-twiddling. We argue for a reimagined transport-protocol implementation course that retains the original pedagogic goals while adopting AI tooling at each stage – and we are explicit about the limits and open questions that accompany this shift.

**ACM Reference Format:**
Jon Crowcroft, Andrew Moore, Richard Mortier. 2026. The Protocol Overlords . In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 Introduction

Mechanical engineers are not expected to spend their working days welding or operating a lathe. Some hands-on experience of fabrication is required, but the job of mechanical engineering is to focus on design, measurement, and analysis. Civil engineers do not lay bricks for a living. Yet *cutting code* has, until very recently, been considered a core daily activity for computer engineers. With the advent of LLM coding assistance, the "what" and the "how" of software construction are, for the first time, separable [1].

In the context of networking education, this separation is significant. The question of designing and building a network protocol shifts dramatically away from the "laying of bricks" – writing C or Rust byte by byte – and raises to higher concerns: What should the wire format look like? How should state be modelled and verified? When is interoperability testing appropriate? What formal guarantees can we make about a system we may never see the source code of? Much as the introduction of graphical calculators into mathematics syllabi did not end the teaching of calculus, LLMs offer networking educators an opportunity to lift students' attention upward from inscrutable compiler error messages toward more general and durable principles of computer science and engineering.

This paper argues for a concrete realisation of that opportunity. We re-imagine a 1980s Masters-level protocol implementation course, updated for the era of generative AI. We use the four stages of the original course as a scaffold and show how AI tooling changes the nature – but not the purpose – of the learning at each stage. We close with what we consider the most important open question: what, precisely, should the primary learning objective of such a course be?

## 2 The Original Course

In the 1980s, one of us ran a Masters programme at UCL in Data Communications, Networks and Distributed Systems. One module – universally hated by students at the time, though differently remembered afterwards – spent an entire 12-week term on the implementation of a complex transport-layer protocol, modelled on ISO TP4 (chosen precisely because no public repositories existed that could serve as shortcuts). Students worked in C, in user space on UNIX machines, using UDP as the network substrate.

The module was structured in four sequential stages, each requiring working code and evidence of testing: (1) packet format codecs, (2) the core connection state machine, (3) the data transfer dynamics (reliability, flow control, congestion control), and (4) an interoperation arena in which each student tested their implementation against three peers'. Students who could not finish a working stage received hands-on assistance rather than a zero; the goal was understanding, not attrition.

The pedagogic intent was to give students genuine insight into protocol design and implementation. Many of those same students, at the end of the full Masters programme, reported that this was the module where they felt they had truly *understood* networking. The question we now ask is: can the same insight be achieved if AI tools do the heavy lifting on the implementation?

[1] Thanks to Matt Grosvenor for a useful analogy and insights from the *coal-face* – awm.


*Conference'17, Washington, DC, USA*

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 3 Reimagining the Four Stages

Inspired by TCP/IP Illustrated [1] and TCP-ex-machina [2], we now consider what this course looks like if it were re-run today, with LLM-based coding assistants and agentic tooling available to students. At each stage, the learning objective shifts upward in abstraction.

*Packet format codecs.* The first stage required implementing software to encode and decode binary packets according to a protocol specification – byte ordering, field alignment, and so on. Given a machine-readable protocol specification, a student today can simply ask an LLM to emit C or Rust codec code. This is not a loss. The interesting questions were never about endianness; they were about the design of the wire format itself: what fields are present, how they are sized, what invariants they encode. An LLM trained on packet traces and augmented with RAG over relevant RFCs can serve as a coding substrate, freeing the student to focus on the design decisions rather than their implementation.

*State machines.* The second stage required implementing the connection state machine (for TCP-like protocols, primarily the connection `open()`/`close()` logic). Given a finite state machine specification in any reasonable machine-readable format, an LLM can emit an implementation. Again, the more interesting part of the work is the specification itself: what are the states, what are the valid transitions, what invariants must hold? This opens the door to *verification tooling* – increasingly used alongside vibe-coding to reduce the risk of automatically generated code containing obscure bugs or vulnerabilities. Cambridge colleagues have published SIGCOMM work on exactly this topic [3], applying HOL specification and symbolic evaluation to TCP implementations; such tools become a natural part of the module.

*Dynamics.* The third stage addressed the data transfer mechanics: reliability, flow control, and congestion control. Here, there is an opportunity for agentic exploration. Agents can vary parameters, choose between algorithmic variants, and explore the performance space in ways that no single student implementation could. The multi-agent nature of congestion and flow control – where behaviour emerges from interaction between implementations – maps naturally onto the kind of simulation and emulation environments already used in swarm robotics research, and represents a genuine intellectual lesson in its own right. Pre-trained agents can serve as the "environment" against which student implementations are tested. Experimental arenas that measure performance and feed results back to the student make this stage closer in spirit to a control-systems lab than a programming exercise.

*Interoperation.* The fourth stage was an interoperation arena: each student chose three peers' implementations and attempted to demonstrate interworking, or produced a diagnosis of what failed. This is the stage most obviously enriched by AI tooling: agents can vary parameters, test edge cases, and probe failure modes far more systematically than any student pair. The same infrastructure is what is needed to scale artifact evaluation at venues such as SIGCOMM – to test more scenarios, more precisely, and report on the generality of systems associated with submitted papers.

Future enhancements could be to use same toolchain to design new protocols, and to use LLMs trained on TCP/IP Illustrated (Vol I—III) and the set of RFCs to help with documenting the designs for human reading. Such a course would also be complimentary to ones like this[4], which address netops skills, and could be similarly AI enhanced.

## 4 The Central Question: What Is the Learning Objective?

The four-stage reimagination above is only coherent if we are clear about what we are trying to achieve. We see two distinct possibilities:

(1) **Learning to use AI to make network things.** The student acquires fluency with LLM-assisted development workflows as applied to networking. The analogy here is teaching undergraduates to program by making them expert in building websites with Django on Python: possibly immediately valuable in industry, but at risk of producing specialists unable to adapt as tools and stacks change.
(2) **Learning to make network things, with AI as one of many tools.** The student develops genuine understanding of protocol design and system behaviour; LLMs, FSM verifiers, formal models, and PID controllers are all tools in the box. This is the mechanical engineer who understands tolerances and material properties, and who uses a lathe when needed but is not defined by the ability to build or operate one.

We argue, in the context of UK HEI computer science teaching, for the second. The goal is not to accelerate the path to a working implementation but to ensure the student could recognise a bad one, debug an interoperability failure, or reason about the behaviour of a system they did not write. These are the skills that survive tool change.

It would be intellectually lazy – and worse, pedagogically counterproductive – to simply run a vibe-coding version of the original course and call it modernised. The purpose of the original course was not to produce C programmers; it was to produce people who understood protocols. That purpose remains unchanged.

## 5 Caveats and Open Questions

We close with three concerns we consider important and underexplored.

*Training data and generalisation.* LLMs are Bayesian systems: their outputs are constrained by the distribution of their training data. For mainstream languages and frameworks, that distribution is broad. For niche transport protocols or novel wire formats, it may be narrow and, as a result, the "genetic material" for the model's prior is limited. Whether the RFC corpus provides sufficient diversity to support the kind of open-ended protocol design we envision is an open empirical question. Early experience with vibe-coded hardware projects suggests that bespoke or novel design spaces expose these limits quickly.

*Equity.* A course that depends on commercial LLM API access raises a question that ought to be stated plainly: is this a well-resourced institution's education? Access to high-capability models is not free, and cost structures change. A course designed around a specific toolchain is fragile in ways that a course designed around principles is not.

*Ethics.* The elephant in the room is that AI tools must be considered in the context of whatever provision is made for teaching ethics in computer science. Discussions of new technology that begin by "putting aside the moral and ethical problems" or "ignoring the climate impact for a moment" rarely end in a good place. We do not attempt to resolve these questions here, however, we note that a course of the kind we describe provides a natural venue for engaging with them concretely rather than abstractly.

Two further questions were raised by the reviewers, and, coincidentally, by our faculty review of the use of LLMs in teaching very recently:

*Learning Outcomes.* Recent studies have shown the impact on skill formation by the use of AI tools[5] [6]. It behooves us to somehow ensure that students are not tempted to overuse the automation tools in lieu of exercising their own cognitive faculties. How to incentivise this will depend very much on the culture among the students and the goals they have in following the course. This remains an important problem to have a strong understanding

*Evaluation.* In the extreme, students might reasonable make use for AI for all coding stages through to testing and reporting. The question then arises how to evaluate what the student has contributed. Certainly god prompt engineering is a sill. But how to quantify?

One answer we tentatively have to both the questions above is that we can viva all the students to interactively assess how much skill and knowledge they have acquired. While viva exams are resource hungry, we can envisage at some point even using an agentic AI to help with the process, assuming it is independent from any agentic AI used in the students tasks, and not colluding.

## 6 Conclusion

When one of us ran the original module in the 1980s, students hated it during the term. At the end of the full Masters programme, many of them reported it was where they had really *got* it. This same student experience was common to another implementation-led graduate module for hardware and software network elements [7] with many similarly noting the journey allowed them to *really get what was going on.*

The course we describe here is not easier. The implementation burden shifts to AI tooling; the intellectual burden shifts upward, to design, verification, and interoperability reasoning. Whether students will hate it during and understand it afterwards is an empirical question. We think they will.

## References

[1] W. R. Stevens, *TCP/IP illustrated (vol. 1): the protocols.* USA: Addison-Wesley Longman Publishing Co., Inc., 1993.

[2] K. Winstein and H. Balakrishnan, "Tcp ex machina: computer-generated congestion control," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 123–134. [Online]. Available: https://doi.org/10.1145/2486001.2486020

[3] S. Bishop, M. Fairbairn, M. Norrish, P. Sewell, M. Smith, and K. Wansbrough, "Engineering with logic: Hol specification and symbolic-evaluation testing for tcp implementations," *SIGPLAN Not.*, vol. 41, no. 1, p. 55–66, Jan. 2006. [Online]. Available: https://doi.org/10.1145/1111320.1111043

[4] T. Holterbach, T. Bühler, T. Rellstab, and L. Vanbever, "An open platform to teach how the internet practically works," 2020. [Online]. Available: https://arxiv.org/abs/1912.02031

[5] J. H. Shen and A. Tamkin, "How ai impacts skill formation," 2026. [Online]. Available: https://arxiv.org/abs/2601.20245

[6] J. Hughes, B. Collier, and D. R. Thomas, "Stand-alone complex or vibercrime? exploring the adoption and innovation of genai tools, coding assistants, and agents within cybercrime ecosystems," 2026. [Online]. Available: https://arxiv.org/abs/2603.29545

[7] J. Sommers and A. Moore, "Scaling the practical education experience," in *Proceedings of the ACM SIGCOMM Education Workshop*, 2011.