Do Large Language Models Dream of Sockets?

Jari Arkko
jari.arkko@ericsson.com
Ericsson
Jorvas, Finland

Dag Lindbo
dag.lindbo@ericsson.com
Ericsson
Stockholm, Sweden

Martin Klitte
martin.klitte@ericsson.com
Ericsson
Lund, Sweden

ABSTRACT

This paper discusses the concept of protocol Large Language Models (LLMs). These are models capable of conversing in native protocol messages. These models could potentially be used to help understand protocols, e.g., diagnose errors from packet traces. Our ongoing research investigates the feasibility of these models, their applications, and limitations.

We present our preliminary results, including how LLMs understand the behavior of example systems such as web servers. Our contribution focuses on testing how and how well an LLM can diagnose protocol traces, receive and send protocol messages over sockets, and handle complex protocol fields and interfaces.

KEYWORDS

AI, LLMs, protocols, simulation, packet traces

ACM Reference Format:
Jari Arkko, Dag Lindbo, and Martin Klitte. 2024. Do Large Language Models Dream of Sockets?. In *Applied Networking Research Workshop (ANRW '24), July 23, 2024, Vancouver, AA, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3673422.3674900

1 INTRODUCTION

Current LLMs enable conversation in different languages and modalities, e.g., images and text. From a networking perspective an interesting new type of language would be protocols, the language of the machines. We can already today converse with LLMs about protocol specifications, as those are in human language. But could we converse using these protocols natively, e.g., have the LLM understand a Domain Name System (DNS) request, or even send a response?

ACM ISBN 979-8-4007-0723-0/24/07
https://doi.org/10.1145/3673422.3674900

Protocols are clearly a type of a language, so we should be able teach this language to LLMs. But what is the optimal way to do this? How would it be used? Potential applications include explaining how a system works, diagnosing issues in packet traces, test data generation, simulation of the systems, or even code generation to reproduce behavior, e.g., for quick prototyping. Which of these are feasible in practice?

The work is based on Generative AI, LLMs, and Generative Pre-trained Transformers (GPTs). We used GPT-4 [11], a foundation model pre-trained with a large corpus of data.

There is prior work on understanding protocol specifications, e.g. Sharma and Yegneswaran [13] extract state machines and Duclos et al. extract formal models [3] from RFCs. Such approaches also can be used for, e.g., generating code [16] or fuzzing tests [9]. Specification data sets have also appeared, e.g., Nikbakht et al. [10]. Packet traces can also be used, e.g., non-AI methods are used by Wang et al. [14] to infer state machines and by Holkovič et al. to diagnose protocol errors [4]. Jiang et al. [6] and Yin et al. [17] apply Generative AI to produce synthetic data resembling input packet traces. This can be useful for testing. LLMs focused on the digital world are also starting to appear, for instance Wu et al. describe bGPT, a general-purpose byte oriented LLM that is used, for instance, to model CPU states [15].

2 EXPERIMENTS

We investigated protocol LLM capabilities through a series of small experiments. We collected packet traces from example protocol systems in simple configurations, such as a recursive DNS resolver (Bind9) that takes requests from clients, configured to use a backend global resolver to resolve the requests. Another example was a web server (Apache) serving HTTP requests. By providing packet traces to an LLM we hope to teach the LLM about the associated behavior, such as what kind of requests result in what kind of responses. We need to understand and generate native protocol messages, but the objective is not so much in the protocol formats (traditional message syntax definitions such as ASN.1 do this better), but rather in the behavior of the system implementing the protocol. Real systems often have a complex interplay of events in different interfaces.

The training can be performed by the creation of tailored foundation models, fine tuning, etc. We have applied simple few-shot, in-context learning [1] in our initial work. We also built an experimental software package to enable feeding protocol information to LLMs, connect them to sockets, etc.

What Can LLMs Say About Packet Traces? To convert the packet traces to a form suitable for the LLM, we represent them as a set of data packet sequences related to a particular task (such as a resolving a name via DNS). Each sequence consists of one or more packets. These are presented to the LLM. We can then test if the LLM can explain behavior. When we asked what happens when a client requests an IPv6 address record from the resolver, the LLM explained the steps in detail, e.g., including the details of how the resolver in our case adds DNS extension mechanism options [2] to the requests sent to the backend server. We can also feed the LLM with another trace for comparison, to diagnose problems in that trace. We created 50 faulty protocol messages and then analyzed manually the correctness of the LLM's diagnosis of those messages. Correctness varied between 72% to 98%, depending on the used techniques (more on this later).

What About Complex Fields? LLMs were able to process protocol messages even as binary data. Hex dumps, for instance, still enabled the LLM to find patterns in the traces, such as recognizing what data should be copied from requests to responses. But diagnosis results improved when the LLM was presented decoded messages (e.g., via ASN.1).

It is also difficult for the LLMs to deal with protocol message aspects that involve mathematical operations, such as length or checksum fields, or encryption. We built a system that combines traditional message syntax definitions with symbolic representations and then fed these to the LLM. For instance, a message with a checksum field followed by an octet string could be represented as "Csum = CSUM(Payload) | Payload = 0xaabbcc". If the system supporting the LLM can parse and construct such messages, it is easy for the LLM to consume and produce such symbolic messages. With the same in-context learning, the LLM with symbolic input performed better than the LLM with only binary input, in particular with checksum and other complex fields. This is not a surprise, given that LLMs are unable to directly perform mathematical operations. AI applications that use reasoning employ similar symbolic techniques [12].

What Is the Source of Knowledge? Can the capabilities of protocol LLMs be explained due to the intrinsic knowledge that the foundation model (such as GPT-4) has, or due to the traces provided during our in-context learning? To test this, we defined a new protocol, unknown to the LLM. The only knowledge the LLM had of this new protocol was provided via the traces. The LLM performed well even with the new protocol, i.e., the traces appeared to be sufficient. Of course, any intrinsic knowledge the LLM has can help. We tested this, and diagnosis results improved when the protocol specification was a part of the training input to the LLM (if the experiment otherwise had room to improve).

Can We Simulate a System? The use of LLMs does not have to be limited to analysis. We built a system that connects data used in-context learning, the LLM, and live sockets. We can then ask the LLM to "predict" what the next event should be, e.g., as in this prompt: "Given the training input sequences, what would be the correct next input in sequence, if we received the following input?" This succeeds surprisingly well. For instance, we were able to use the LLM as a real, functioning (if slow) recursive DNS resolver based on the packet capture sequences alone. However, the LLM will not perform things that are not visible in the external behavior (such as DNSSEC validation).

Can we generate code? We experimented with the LLM producing code based on the traces of simple protocols. The generation was guided by the formal syntax of the protocol which defines the available messages and their fields. This resulted in implementations that are similar to the simulators discussed above but operate without the LLM and are faster.

Are Protocols Everything? Clearly, protocols are just one aspect of complex system behavior. Take a web server as an example. It has a protocol interface, but also other complex functionalities such as interaction with the file system for content. A web server viewed only through the protocol lens would seemingly come up with content without any explanation why this particular content was provided. The behavior of the web server is not defined solely by the protocol interface. But we can observe the system across all of its external interfaces. We recorded the behavior of a web server both on network and file related system call interfaces using the Linux strace tool [8]. This enabled the LLM to learn the behavior of opening files requested on GET requests and providing the content of those files in responses.

Should Cyberdyne Systems access your files? There are many security concerns, such as what communication channels or operating system resources the LLM is allowed to access. We limited the LLM to specific resources, e.g., specific peers and file system parts.

3 CONCLUSIONS

It is intriguing that LLMs can learn protocol languages, at least in simple cases. There are potential benefits for several networking tasks. Given hallucination [5] and other limitations of LLMs, the likely use cases are in diagnostics, test generation, and simulation. We plan to further dive into systematic evaluation of LLM fidelity in these areas and more complex configurations, training methods, custom models, protocol-syntax driven tokenization [7], and the possibility of LLM-provided protocol optimizations.

This work was supported by Ericsson. F. Öström, D. Corcoran, J. Jiménez, F. Moradi, T. Kauppinen, P. Mátray, and A. Karapantelakis provided many insights.

REFERENCES

[1] Tom B. Brown, Benjamin Mann, and Nick Ryder et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

[2] Joao da Silva Damas, Michael Graff, and Paul A. Vixie. 2013. Extension Mechanisms for DNS (EDNS(0)). RFC 6891. https://doi.org/10.17487/RFC6891

[3] Martin Duclos, Ivan A. Fernandez, Kaneesha Moore, Sudip Mittal, and Edward Zieglar. 2024. Utilizing Large Language Models to Translate RFC Protocol Specifications to CPSA Definitions. arXiv:2402.00890 [cs.CR]

[4] Martin Holkovič, Ondřej Ryšavý, and Libor Polčák. 2019. Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics. In *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications Volume 1: DCNET, ICE-B, OPTICS, SIGMAP and WINSYS* (Praha, CZ). SciTePress - Science and Technology Publications, 37–47. https://doi.org/10.5220/0007929900370047

[5] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. arXiv:2311.05232 [cs.CL]

[6] Xi Jiang, Shinan Liu, Aaron Gember-Jacobson, Paul Schmitt, Francesco Bronzino, and Nick Feamster. 2023. Generative, High-Fidelity Network Traces. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (, Cambridge, MA, USA,) *(HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 131–138. https://doi.org/10.1145/3626111.3628196

[7] Franck Le, Mudhakar Srivatsa, Raghu Ganti, and Vyas Sekar. 2022. Rethinking data-driven networking with foundation models: challenges and opportunities. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 188–197. https://doi.org/10.1145/3563766.3564109

[8] Juan Lopez, Leonardo Babun, Hidayet Aksu, and A. Selcuk Uluagac. 2017. A Survey on Function and System Call Hooking Approaches. *Journal of Hardware and Systems Security* (2017), 114–136. https://doi.org/10.1007/s41635-017-0013-2

[9] Ruijie Meng, Martin Mirchev, Marcel Böhme, and Abhik Roychoudhury. 2024. Large Language Model guided Protocol Fuzzing. https://doi.org/10.14722/ndss.2024.24556

[10] Rasoul Nikbakht, Mohamed Benzaghta, and Giovanni Geraci. 2024. TSpec-LLM: An Open-source Dataset for LLM Understanding of 3GPP Specifications. arXiv:2406.01768 [id='cs.NI']

[11] OpenAI, Josh Achiam, Steven Adler, and et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[12] Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning. arXiv:2305.12295 [cs.CL]

[13] Prakhar Sharma and Vinod Yegneswaran. 2023. PROSPER: Extracting Protocol Specifications Using Large Language Models. *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (2023). https://api.semanticscholar.org/CorpusID:265158541

[14] Yipeng Wang, Zhibin Zhang, Danfeng (Daphne) Yao, Buyun Qu, and Li Guo. 2011. Inferring Protocol State Machine from Network Traces: A Probabilistic Approach. In *Applied Cryptography and Network Security*, Javier Lopez and Gene Tsudik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–18.

[15] Shangda Wu, Xu Tan, Zili Wang, Rui Wang, Xiaobing Li, and Maosong Sun. 2024. Beyond Language Models: Byte Models are Digital World Simulators. arXiv:2402.19155 [cs.LG]

[16] Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. 2021. Semi-Automated Protocol Disambiguation and Code Generation. arXiv:2010.04801 [cs.NI]

[17] Yucheng Yin, Zinan Lin, Minhao Jin, Giulia Fanti, and Vyas Sekar. 2022. Practical GAN-based synthetic IP header trace generation using NetShare. In *Proceedings of the ACM SIGCOMM 2022 Conference* (Amsterdam, Netherlands) *(SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 458–472. https://doi.org/10.1145/3544216.3544251