University of Crete | Computer Science Department

Bachelor Thesis

WiresharkMCP: Automating Network Diagnostics through Dynamic TShark Integration and LLM Agents

Anastasia Dourou

Supervisor: Xenofontas Dimitropoulos
Co Supervisor: Adonis Chatzivasileiou

Heraklion, 2025-2026

Abstract

This research investigates the automation of network diagnostics by bridging Large Language Models (LLMs) with raw packet data through a custom Model Context Protocol (MCP) server. By interfacing directly with the Tshark CLI, the proposed implementation enables autonomous agents to perform dynamic deep packet inspection and protocol analysis using standardized tool calls. The methodology was evaluated using GPT-5, Claude 4.5 Sonnet, and Gemini 3 Pro across three case studies involving academic protocols, security forensics, and network troubleshooting. While all models achieved a 100% success rate in foundational academic labs, complex scenarios revealed a critical divergence in investigative archetypes. Specifically, the iterative reasoning strategy employed by Gemini 3 Pro allowed it to identify a massive 19.25 second retransmission timeout that other models overlooked due to their reliance on predictive batching and standard timing metrics. Despite challenges such as field hallucinations and buffer constraints, the study concludes that a reactive, adaptive diagnostic workflow is essential for LLM agents to reach parity with human expert analysis in real-world network environments.

Table of Contents

Abstract 1

Table of Contents 2

1. **Introduction** 4

1.1 Model Context Protocol 4

1.2 Our implementation 5

1.3 Model Profiles 6

2. **Methodology** 7

2.1 The Technical Stack 7

2.2 Tool Architecture 7

2.2.1 Tool Description 8

2.2.2 Input Schema & Validation 8

2.2.3 Data Extraction & Heuristic Defaults 8

2.2.4 Output Format 9

2.3 The Analysis Workflow 9

2.4 Model Selection and Dataset Specifications 10

2.4.1 Frontier Model Profiles 10

2.4.2 Dataset and Case Study Selection 10

3. **Results & Analysis** 11

3.1 Case Study I: Academic Protocol Foundations 11

3.2 Case Study II: Security Forensics and Malware Detection 12

3.2.1 Unified Diagnostic Findings 12

3.2.2 Tool Interaction Patterns: Batching vs Iterative Reasoning..... 12

3.2.3 Evidence Correlation Table……..... 13

3.2.4 Comparative Analysis of Investigative Archetypes 14

3.3 Case Study III: Network Troubleshooting 14

3.3.1 Diagnostic Findings 14

3.3.2 Tool Interaction Patterns: Batching vs Iterative Reasoning 15

3.3.2 Comparative Analysis of Diagnostic Depth 15

3.4 Technical Limitations & Agentic Error Analysis 15

3.4.1 Field Hallucinations and Syntactic Errors 16

3.4.2 Buffer Constraints 16

3.4.3 Impact on the Efficiency 16

4. **Conclusions** 17

5. **Sources** 18

6. **Appendix** 19

Test Prompts (Case Studies I-III) 19

1. Introduction

In the world of network engineering, performing measurements and capturing traffic is only half the battle, the ability to interpret these findings clearly is equally critical. For decades, protocol analysis has been a manual and labor intensive process. Tools like Wireshark provide deep visibility into network traffic, but extracting meaningful insights from thousands of packets, diagnosing network latency issues, requires significant domain expertise and time.

The recent rise of Large Language Models (LLMs) offers a promising path toward automating network diagnostics. However, a fundamental gap still persists: despite their reasoning capabilities, LLMs lack a native, stateful interface to interact with raw network data formats. This makes the direct analysis of PCAP files a significant challenge.

1.1 Model Context Protocol

This is where the Model Context Protocol (MCP), introduced by Anthropic in late 2024, enters the picture. MCP provides a universal, client-server interface that allows AI models to access external data sources (such as databases or other files), via standardized tool calls.

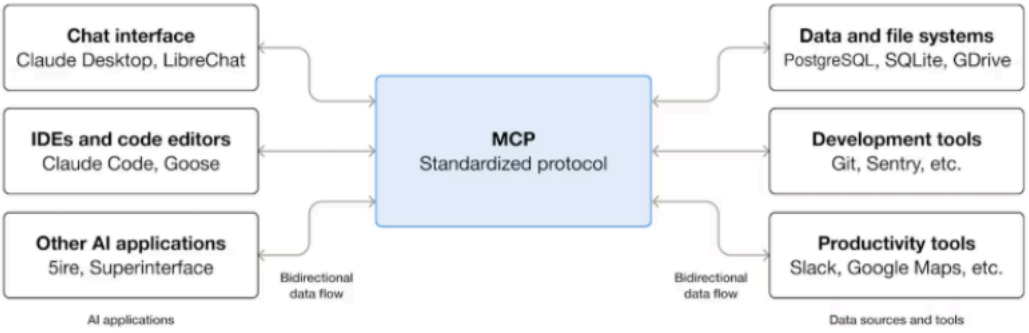

Image 1: The MCP Architecture and Examples

Previous methods of tool integration presented various limitations such as:

- Each tool needed was made from scratch with custom code, authentication, and error handling. This led to inflexible architectures that lacked robustness and scalability.
- Previous systems were “stateless” as they could not retain memory or share information between different tools.
- Security defenses were weak because there were no standardized mechanisms in place.

- As a natural consequence, as more tools were added, the systems became overwhelmed, requiring increasingly more tokens just for tool descriptions.
- Previous methods are too technical and require programming expertise.

MCP improved significantly upon these limitations by introducing a universal, single, efficient interface with diverse data sources.

As mentioned above, the way it works is via tool implementation and a call. More specifically, a tool allows the server to perform an action. Each tool takes a specific input schema and produces an output. MCP uses JSON Schema for validation.

```
{
  name: "searchFlights",
  description: "Search for available flights",
  inputSchema: {
    type: "object",
    properties: {
      origin: { type: "string", description: "Departure city" },
      destination: { type: "string", description: "Arrival city" },
      date: { type: "string", format: "date", description: "Travel date" }
    },
    required: ["origin", "destination", "date"]
  }
}
```

Image 2: Example of tool definition

The tools are model-provoked, that is, models know them and can call them automatically.

Complementing tools, MCP introduces resources that provide the information to be analyzed by the AI models. Resources expose data from APIs, databases, files, or any other type of information AI needs context to understand. In the context of this project, the primary resource consists of network traffic data from pcap or pcapng files. Tshark is utilized as the underlying engine to parse these files and expose their content in a structured JSON format, enabling the LLM to “see” and interpret the network packets accurately.

1.2 Our implementation

While other MCP servers for Tshark exist, they often restrict the agent to basic summaries. Our implementation introduces a higher level of granularity, empowering the LLM to dynamically define any specific field or filter it requires for its analysis. This method allows the AI to act as a principal investigator, navigating through raw .pcap files with the same flexibility as a human expert.

To evaluate the effectiveness of this dynamic integration, we conducted a comparative study using three frontier models, Gemini 3 Pro, GPT- 5, and Claude 4.5 Sonnet, across three case studies:

1. Academic Foundations: Solving standardized questions from Wireshark labs from the Kurose & Ross computer networking textbook.
2. Security Forensics: Identifying infection patterns using real-world traffic samples from malware-traffic-analysis.net.
3. Network Latency & Troubleshooting: Diagnosing complex performance bottlenecks based on case studies from industry expert Chris Greer.

1.3 Model Profiles

The creation of an MCP server that offers access to the specific attributes of network traffic provides the visibility required for a thorough investigation. While the server serves as the technical interface, the final success of a diagnosis depends on the internal logic and the investigative strategies that the agent employs. To evaluate how well this workflow operates, it is necessary to use models that represent the leading edge of technology in their ability to use tools and process vast amounts of information. This section outlines the profiles of the three models selected for this study, highlighting why their different approaches to reasoning are vital for the analysis of complex protocols.

Gemini 3 Pro (Google) is a natively multimodal reasoning model introduced in November 2025. It is designed to handle complex tasks by comprehending vast datasets from diverse sources, including text, images, audio, video, and entire code repositories. Built on a sparse mixture-of-experts (MoE) transformer architecture, the model dynamically routes tokens to specialized parameter subsets to optimize performance. A significant feature is its optional "Deep Think" mode, which is engineered to enhance problem-solving capabilities during inference. The model supports a token context window of up to 1 million tokens and is specifically suited for agentic performance, advanced coding, and long-context applications. In technical evaluations, Gemini 3 Pro demonstrated high proficiency in agentic tool use and coding, scoring 85.4% on Tool-bench and 76.2% on SWE-Bench Verified.

Claude 4.5 Sonnet (Anthropic) is a hybrid reasoning model trained on a mix of public and proprietary data as of July 2025. It is specifically engineered for high performance in software coding, "agentic" tasks where the model operates in an autonomous loop, and direct computer use. A core feature of this model is its "extended thinking mode," which allows it to generate a detailed internal thought process for solving complex or difficult problems. In terms of security and safety, Claude 4.5 Sonnet is deployed under the AI Safety Level 3 (ASL-3) standard, reflecting its advanced capabilities and the associated precautionary safeguards. In agentic evaluations, it demonstrated remarkable resilience against prompt injection attacks, achieving the lowest successful attack rate on the Gray Swan Agent Red Teaming (ART) benchmark compared to all other models tested. Furthermore, it shows a high degree of safety in coding tasks, refusing 98.7% of malicious coding requests in specialized safety evaluations. Its integration with the Model Context

Protocol (MCP) and other tool-use surfaces is bolstered by multilayered defense strategies, including real time classifier systems to prevent unauthorized actions.

GPT-5 (OpenAI) is a unified system composed of a fast, high-throughput model (gpt-5-main), a deeper reasoning model for complex problem-solving (gpt-5-thinking), and a real time router that dynamically directs queries based on task complexity and tool requirements. The reasoning models are trained using reinforcement learning to produce an internal chain of thought, allowing them to refine strategies, recognize mistakes, and follow complex guidelines more effectively. Significant technical improvements have been made to reduce factual hallucinations, with gpt-5-thinking showing a 65% reduction compared to its predecessor, and to enhance instruction following. For autonomous operations, the model supports advanced tool calling, web browsing, and external connectors, achieving a 74% success rate on the SWE-bench Verified benchmark for resolving real world software issues. Security is managed through an "Instruction Hierarchy" that prioritizes system level instructions over user prompts, and a "safe-completions" approach that focuses on the safety of the output rather than binary refusals of intent.

2. Methodology

To ensure reproducibility and scientific validity, our methodology focuses on the autonomous interaction between Large Language Models and raw packet data. We established a local research environment designed to evaluate how different AI architectures interpret network phenomena without manual filtering during the analysis phase.

2.1 The Technical Stack

Interface: Visual Studio Code serves as the primary workspace, utilizing GitHub Copilot as the bridge to various frontier models.

The Bridge (MCP): A custom MCP server was built on Node.js, using TypeScript SDK and Zod for schema validation. The server operates via Stdio transport, allowing seamless communication between the LLM and the local environment.

The Engine (Tshark): The server interfaces directly with the Tshark CLI, allowing the models to perform deep packet inspection on .pcap and .pcapng files.

2.2 Tool Architecture

The centerpiece of this implementation is a specialized MCP tool named *analyze_pcap_details*. This tool was designed with three primary objectives: data efficiency, structured reasoning, and most importantly autonomous investigation.

2.2.1 Tool Description

The description provided to the LLM is a critical component of the methodology, as it defines the "rules of engagement" for the AI:
"Parses and extracts protocol level data from PCAP files for network analysis and troubleshooting. Use this to query specific packet fields or to get a summary of network traffic. Always use 'displayFilter' to narrow down packets before extraction to avoid excessive data. If 'fields' is not provided, the tool returns a default summary (frame number, time, source, destination, protocol, and info). Ideal for diagnosing connectivity issues, latency, packet loss, or analyzing protocol specific behavior."
This description forces the model to think like a human analyst: it encourages the use of filters to minimize noise and highlights the tool's versatility in both high-level summaries and field extraction.

2.2.2 Input Schema & Validation

To ensure reliability, we implemented a rigorous input schema using Zod. The schema defines the following parameters:

Table 1: Input Schema and Parameter Specifications for the analyze_pcap_details Tool

Parameter	**Type**	**Description**
pcapFilePath	string	The absolute path to the target .pcap or .pcapng file.
displayFilter	string(opt)	Standard Wireshark display filter syntax (e.g., http.request or tcp.flags.syn == 1).
fields	array<string>	A dynamic list of specific Tshark packet fields to extract (e.g., ["ip.src", "tcp.seq", "http.host"]).
tsharkPath	string(opt)	Path to the executable Tshark CLI (defaults to "tshark").

2.2.3 Data Extraction & Heuristic Defaults

The server's logic is designed to handle two distinct analysis modes:

1. The "Packet List" Mode (Default): If the LLM does not specify fields, the server automatically injects a predefined set of fields (*frame.number, frame.time_relative, ip.src, ip.dst, _ws.col.Protocol, _ws.col.Info*). This provides a broad overview, similar to the Wireshark GUI's main window.

2. The “Deep Inspection” Mode: When the LLM identifies a specific problem, it can request a dynamic array of fields (e.g., *tcp.analysis.initial_rtt* for latency). The server then dynamically builds the Tshark command using the -e flag for each requested field.

2.2.4 Output Format

The use of the *-T json* output format ensures high data integrity. The JSON format provides a standardized object structure, which allows the LLM to navigate protocol properties as key-value pairs and accurately correlate data across multiple packets.

2.3 The Analysis Workflow

Each measurement task followed a straightforward path:

- Technical Inquiry: A prompt is issued (e.g., "Identify the HTTP GET requests and show the hostnames in this pcap").
- Tool Call and Reasoning: The LLM evaluates the question and independently generates a specific tshark command, defining the necessary display filters (e.g., “http.request.method == 'GET'”) and metadata fields (e.g., frame.number, http.host).
- Data Extraction: The MCP server executes the command and feeds the raw JSON output back to the model.
- Synthesis and Response: The model correlates the extracted packet data with networking principles to provide a final answer, based on the evidence it received.

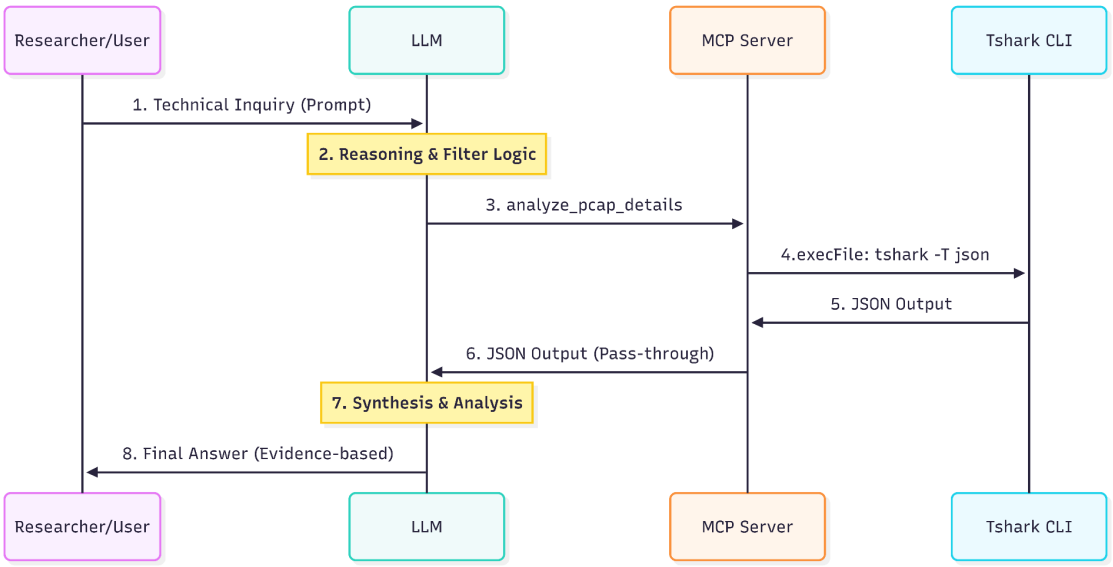

Image 3: Sequence diagram of the PCAP analysis interaction. The LLM dynamically invokes Tshark via the MCP server, receiving a transparent JSON relay of packet metadata to perform deep protocol inspection.

2.4 Model Selection and Dataset Specifications

To evaluate the limits of autonomous network analysis, we designed a comparative study across different AI architectures. This section details the rationale behind our model selection and the specific characteristics of the datasets used for testing.

2.4.1 Frontier Model Profiles

To ensure the study reflects the current capabilities of LLMs in network automation, we selected the leading "frontier" models available as of late 2025. As mentioned above, these models represent the state-of-the-art (SOTA) in terms of reasoning, complex tool calling, and context management.

The selection includes:

- GPT-5 (OpenAI)
- Claude 4.5 Sonnet (Anthropic)
- Gemini 3 Pro (Google)

By using the most advanced models from the three major AI providers, we aim to provide a comprehensive cross-provider evaluation of autonomous packet analysis.

2.4.2 Dataset and Case Study Selection

To evaluate the limits of autonomous network analysis, we established a framework consisting of three distinct Case Studies. The primary objective is to verify the accuracy of the models' findings and analyze their reasoning process when interpreting raw network data.

Case Study I: Academic Protocol Foundations

- Source: Standardized Wireshark labs from the 9th Edition of *"Computer Networking: A Top-Down Approach"* (Kurose & Ross).
- Objective: To check if the AI agent can provide correct answers to textbook-level questions. We evaluate the accuracy of the extracted metadata (e.g., sequence numbers, RTT, flags) and the reasoning the model uses to correlate these values with standard networking principles.

Case Study II: Security Forensics

- Source: Real-world infection traffic samples from *malware-traffic-analysis.net.*

- Objective: To evaluate the model's ability to correctly identify infection patterns. The focus is on whether the AI correctly identifies the "who-what-where" of an attack and if its reasoning for labeling a specific flow as malicious is technically sound.

Case Study III: Network Troubleshooting

- Source: Packet captures from industry expert Chris Greer.
- Objective: To verify the accuracy of the AI's diagnosis in complex "slow network" scenarios. We examine if the model can pinpoint the root cause of a performance bottleneck and analyze the reasoning it follows to reach that conclusion.

3. Results & Analysis

3.1 Case Study I: Academic Protocol Foundations

The initial phase of our evaluation served as a baseline validation for the MCP server's integration. We tested the models against a subset of 17 technical questions derived from the HTTP, DNS, and TLS Wireshark labs of the Kurose & Ross (9th Ed.) textbook. This allowed us to verify if the LLMs could autonomously map natural language inquiries to specific protocol fields and stateful interactions.

Across the three foundational protocols, all three frontier models, GPT-5, Claude 4.5 Sonnet, and Gemini 3 Pro, achieved a 100% success rate. They correctly identified frame numbers, IP addresses, and complex protocol states (such as TLS cipher suite negotiation) without any human intervention in the query formulation.

Table 2: Accuracy Rates for Academic Baseline (Case Study I)

Model	**Labs Tested**	**Accuracy(%)**
GPT-5	HTTP (7 questions)	100
Claude 4.5 Sonnet	DNS (6 questions)	100
Gemini 3 Pro	TLS (4 questions)	100

Beyond simple data extraction, the models displayed advanced contextual reasoning by interpreting protocol sequences rather than just reading raw fields. In the TLS lab, all models correctly deduced that the first encrypted packet (Frame 41) following the Server Hello (Frame 32) was an HTTP GET request. This demonstrates an ability to correlate the sequence of network events with higher layer protocol behavior despite the payload being encrypted. Finally, the models exhibited granular identification in the DNS lab by successfully correlating packet pairs, identifying Packet 15 as the query and Packet 17 as the corresponding response, while accurately identifying the specific resolver IP address as 75.75.75.75.

The interaction with the *analyze_pcap_details* MCP tool was characterized by a primarily iterative approach. Gemini 3 Pro and Claude 4.5 Sonnet maintained a sequential reasoning strategy, executing “Reasoning → Tool Call → Observation” cycles for each of the prompts in the HTTP, DNS, and TLS labs.

A notable observation was the behavior of GPT-5, which in this phase diverged from the "Predictive Batching" strategy seen in later stages of the study. Specifically, when a prompt required more than one tool call to identify complex protocol states, GPT-5 opted for iterations rather than batching.

The successful performance of the analysis tool in this foundational phase served as a critical baseline validation. By achieving 100% accuracy across these academic tests, the tool proved to be a reliable interface for enabling frontier LLMs to master standardized protocol interpretation. With the technical efficacy of the tool secured, the investigation transitioned from academic baselines to real-world applications, including security forensics and advanced network troubleshooting.

3.2 Case Study II: Security Forensics and Malware Detection

In the second phase of our evaluation, the AI agents were tasked with performing a comprehensive forensic investigation of a network capture containing real-world Trickbot infection traffic. This case study aimed to test whether the models could transition from simple data extraction to complex technical reasoning, identifying not just the “what” but also the “how” and “why” of a security compromise.

3.2.1 Unified Diagnostic Findings

Despite their different architectural backgrounds, all three frontier models achieved a 100% success rate in identifying the critical components of the breach. They correctly identified the primary infected workstation (10.5.28.229) and the secondary compromise of the Domain Controller (10.5.28.8). Furthermore, the agents successfully mapped the command-and-control (C2) infrastructure, pinpointing external IP addresses and non standard ports used for malicious beaconing.

3.2.2 Tool Interaction Patterns: Batching vs Iterative Reasoning

A critical observation in this case study was the divergence in how the models orchestrated their interaction with the *analyze_pcap_details* tool via the MCP server.

Predictive Batching (GPT-5): GPT-5 adopted a "breadth-first" strategy. At the onset of the investigation, the model performed a reasoning phase to map out all potentially relevant filter. It executed its tool calls as a consolidated sequence, triggering both DNS and HTTP queries as a single investigative block, before synthesizing the final report. While

computationally efficient, this approach relied heavily on the model's ability to predict the infection chain without seeing intermediate data.

Iterative State (Gemini 3 Pro & Claude 4.5): In contrast, Gemini and Claude followed a "depth-first" iterative loop. Their workflow followed a continuous cycle of *Reasoning → Tool Call → Observation → Refined Reasoning*. For instance, after an initial DNS overview, these models paused to evaluate the results before issuing targeted queries for specific C2 URI patterns or SMB filenames. This methodology more closely mirrors the workflow of a human "Principal Investigator" who adjusts their search based on findings.

3.2.3 Evidence Correlation Table

To further quantify the diagnostic depth, Table 3 summarizes the specific forensic artifacts autonomously extracted by each agent using the MCP server.

Table 3: Forensic Artifact Extraction by Model

Evidence Category	**Artifact Example**	**GPT-5**	**Claude 4.5**	**Gemini 3 Pro**
C2 Beacons	HTTP POST to 36.89.106.69	Yes	Yes	Yes
Lateral Movement	SMB/RPC (Port 49155)	Yes	Yes	Yes
Stealth Indicators	Tor .onion resolution	No[1]	Yes	Yes
Evasion Tactics	RBL (Blacklist) Reputation Checks	Yes	Yes	Yes
Payload Analysis	Executable size/type (imgpaper.png)	Yes	Yes	Yes

[1] While GPT-5's batch plan successfully triggered the DNS queries that captured the Tor resolution in the raw tool output, the model failed to include this artifact in its final executive summary.

3.2.4 Comparative Analysis of Investigative Archetypes

The forensic depth achieved in the Trickbot investigation revealed a compelling compromise between investigative breadth and narrative structure, driven by the internal reasoning logic of each agent.

GPT-5 acted as a broad pattern matcher. Its initial batch captured the RBL (Blacklist) reconnaissance activity, but possibly because the Tor hidden service resolution (5efxqhk2zhgnc24l.onion) was not part of its initial heuristic plan, the model failed to prioritize it during the report synthesis phase. This highlights a "needle in a haystack" scenario where critical stealth indicators are technically retrieved but analytically overlooked.

Claude reconstructed a precise timeline consisting of six stages of the compromise. This breakdown moved from IP discovery and C2 Registration, through secondary communications and payload downloads, to the final stages of Lateral Movement and TOR network indicators. This approach ensures technical completeness by treating every protocol anomaly as a sequential step in the attack chain.

Gemini utilized a structure driven by narrative that prioritized the functional intent of the malware. Unlike the other models, it specifically categorized the RBL queries not just as network activity, but as "strategic sandbox evasion" and "reconnaissance". This highlights an investigative style that anchors technical artifacts (like DNSBL lookups) to their likely purpose within the malware's lifecycle.

The divergence in these results suggests that while the a*nalyze_pcap_details* tool provides the necessary visibility, the agent’s internal prioritization, whether focused on broad discovery (GPT-5) or sequential reconstruction (Claude), dictates the ultimate granular depth of the forensic investigation.

3.3 Case Study III: Network Troubleshooting

This final case study shifted to performance diagnostics using the dataset featured in the expert analysis video “Wireshark Tutorial // Fixing SLOW APPLICATIONS” by Chris Greer. The models were tasked with identifying the primary delays in the trace, determining their root causes, and pinpointing the single largest delay found in the capture . This scenario tested the models' ability to distinguish between standard network latency (RTT), application processing time (think time), and critical protocol timeouts (RTO).

3.3.1 Diagnostic findings

The evaluation revealed a significant performance gap in “investigative intuition”. While GPT-5 and Claude 4.5 Sonnet identified delays in the range of 1.1 seconds, they both failed to detect the massive 19.25 second stall discovered by Gemini 3 Pro.

3.3.2 Tool Interaction Patterns: Batching vs Iterative Reasoning

The reasoning findings in this case study align with those identified in the previous analysis.

As before, GPT-5 utilized a strategy of predictive batching. At the onset, the model performed a reasoning phase to map out all potentially relevant filters, including *frame.time_delta*, DNS, and HTTP timings. It executed its tool calls as a consolidated sequence of parallel analyses. While theoretically efficient, this approach relied on the model's ability to predict where the delay might be without seeing intermediate results. This once again led to a "needle in a haystack" scenario where the model retrieved large amounts of data but missed the specific protocol anomaly in Stream 0.

In contrast, Gemini 3 Pro and Claude 4.5 maintained the same iterative state observed earlier, refining their search based on incoming observations. For instance, Claude 4.5 used progressive RTT thresholding, while Gemini 3 Pro pivoted to investigate Retransmission Timeouts (RTO) after detecting initial flags. This reinforces the conclusion that a reactive, adaptive diagnostic workflow is superior for pinpointing complex protocol anomalies compared to a static heuristic.

3.3.3 Comparative Analysis of Diagnostic Depth

The discrepancy in findings highlights a fundamental difference in how each model defines and searches for "delays." Claude 4.5 and GPT-5 primarily identified "slowness", measurable network latency where traffic is still moving. Claude 4.5 pinpointed a 1.155s delay in Stream 25 as its maximum find, while GPT-5 identified a 1.167s interframe gap in a TLS flow as the largest "silence". Both models also noted a significant DNS outlier for an ad-service domain with a resolution time of approximately 1.115 seconds.

The reason these models missed the 19.25s delay lies in their reliance on standard timing metrics like RTT. RTT requires a successful packet-ACK pair. In the case of Stream 0, the initial packet was lost, meaning no RTT could be calculated for that interval. Gemini 3 Pro's diagnostic depth allowed it to see through this "silence". It determined that packet #1 was sent at the start of the capture but never received an acknowledgment, forcing the sender to wait for a full Retransmission Timeout (RTO) of 19.25 seconds before attempting packet #9804. Gemini also differentiated between this major timeout and a 1.05s delay in Stream 25, which it correctly attributed to a lost SYN packet during the initial connection handshake. Ultimately, Gemini 3 Pro's ability to correlate retransmissions with absolute timing allowed it to identify a critical "frozen" connection that the other models overlooked as mere background silence.

3.4 Technical Limitations & Agentic Error Analysis

Throughout the investigation of all case studies, several recurring technical challenges were observed regarding how the AI agents interacted with the *analyze_pcap_details* tool. These

errors highlight the current limitations of LLMs when operating as autonomous agents in environments with a lot of data.

3.4.1 Field Hallucinations and Syntactic Errors

A notable limitation was the frequent hallucination of packet field names and syntax. While all models experienced these issues, Claude 4.5 exhibited a significantly higher frequency of syntactic errors, often replacing underscores with spaces or misspelling protocol prefixes. These hallucinations directly led to tool failures, with the system returning "some fields are not valid" as an error message.

Table 4: Observed field hallucinations

Hallucinated field	Attempted by	In case study
http.response_for.uri	Claude 4.5 Sonnet	I
transport_layer.port	Claude 4.5 Sonnet	I
tls.cipher_suite tls.handshake.cipher_suites	Claude 4.5 Sonnet	I
_ws.col.Info	GPT-5	III

3.4.2 Buffer Constraints

Another significant challenge was the models' occasional failure to apply sufficiently granular display filters. When agents executed broad queries, such as filtering for all "ip" (Claude in the second case study) or "http" (GPT in the third case study) traffic without constraints or with no filter at all (Claude in case study III), the volume of data exceeded the environment's technical limits. This resulted in multiple *stdout maxbuffer exceeded* errors. GPT-5, in particular, acknowledged the tool's warnings regarding the lack of filters but proceeded with broad queries regardless, trusting the tool to return the expected output despite the ambiguity. These buffer overflows forced the models to restart their investigative process or synthesize reports based on incomplete data fragments.

3.4.3 Impact on the efficiency

The combination of field hallucinations and poor filtering strategy significantly impacted the “Predictive batching” approach. Because models like GPT-5 planned multiple queries in advance without intermediate verification, a single invalid field or a buffer overflow in the first

step of the batch could compromise the entire investigative chain. For example in the case study III, GPT-5, because of these errors, only had 2 successful tool calls out of 4 total[2].

Conversely, the “Iterative reasoning” models (Gemini and Claude) were better equipped to recover from these errors. They could observe a “field not valid” error and immediately refine their next tool call to correct the syntax. This underscores the necessity for robust error handling and more precise “protocol aware” training for agents tasked with autonomous network forensics.

4. Conclusions

The implementation of the custom MCP server successfully established a robust and reliable bridge between frontier LLMs and raw network data, enabling autonomous packet inspection with remarkable precision. During the initial validation phase, the 100% success rate achieved across all academic labs confirmed the tool's efficacy in translating natural language technical inquiries into accurate TShark commands and stateful protocol interpretations. However, as the investigation transitioned to complex, real-world scenarios, a critical divergence emerged regarding the models' investigative archetypes. Specifically, GPT-5 exhibited a hybrid archetype, while it initially employed iterative reasoning for foundational validation, it pivoted toward a predominantly "predictive batching" strategy when tasked with more complex, multistage scenarios. This preference for preplanned investigative blocks, while computationally efficient, proved less effective than the consistent, adaptive "iterative reasoning" loops employed by Gemini 3 Pro and Claude 4.5. This was most evident in the security forensics case study, where GPT-5's preplanned queries successfully retrieved stealth indicators such as Tor resolution but ultimately failed to include them in the final synthesis, highlighting a gap between batch driven data retrieval and analytical prioritization.

The superiority of a reactive, adaptive diagnostic workflow was further solidified in the network troubleshooting analysis, where Gemini 3 Pro’s investigative intuition allowed it to identify a massive 19.25s retransmission timeout. This critical performance bottleneck was overlooked by other models, which relied on standard timing metrics and interpreted the resulting silence as mere background inactivity. Despite these advanced diagnostic capabilities, the study also identified persistent technical limitations, most notably the frequent hallucination of packet field names and the occasional failure to apply granular display filters. These errors often led to buffer overflows and compromised the efficiency of the predictive batching approach. Ultimately, this research demonstrates that for an AI agent to achieve true parity with a human expert, it must possess more than just an iterative workflow, it requires a diagnostic depth capable of questioning "missing" data and interpreting protocol behavior beyond the limits of standard automated metrics.

To further advance the capabilities of autonomous network diagnostic agents, future research should focus on evolving the MCP server from having a single tool into a more

[2] Although this is a significant amount of errors, the failure to detect the 19.25s delay was not just a result of these tool errors, but a fundamental flaw in the preplanned diagnostic logic, as mentioned in the Case Study III section of the Results.

diverse tool ecosystem. Building on this foundation, introducing specialized tools, such as those dedicated exclusively to high-level statistical summaries or protocol distribution, would allow models to perform initial reconnaissance without triggering the stdout buffer overflows encountered during broad heavy queries.

Furthermore, to overcome the frequent field hallucinations that hindered diagnostic, it is essential to move beyond prompt based instructions toward the specialized training of LLMs on comprehensive Tshark field schemas. Given that the sheer volume of specific packet fields is too vast for standard context windows, training models to internalize these syntaxes would significantly enhance reliability. By combining these structural improvements with the advanced reasoning capabilities observed in models like Gemini 3 Pro, future autonomous agents will be better equipped to navigate the complexities of real-world network forensics and troubleshooting with the diagnostic proficiency of experts.

5. Sources

[1] Xinyi Hou, Yanjie Zhao, Shenao Wang, Haoyu Wang (2025). Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. arXiv.org

[2] Padmanabhan Venkiteela (2025). The New Interoperability Paradigm: Model Context Protocol (MCP), APIs, and the Future of Agentic AI. Computer fraud & security

[3] Xiang Fei, Xiawu Zheng, Hao Feng (2025). MCP-Zero: Active Tool Discovery for Autonomous LLM Agents. arXiv.org

[4] Anthropic. (2024). Introduction - Model Context Protocol. *Model Context Protocol Documentation*. [Online]. Available: https://modelcontextprotocol.io/docs/getting-started/intro. [Accessed: March, 2026].

[5] Anthropic. (2024). Understanding MCP servers. *Model Context Protocol Documentation*. [Online]. Available:https://modelcontextprotocol.io/docs/learn/server-concepts. [Accessed: March, 2026].

[6] Kurose, J. F., & Ross, K. W. (2024). Computer Networking: A Top-Down Approach (9th ed.). Pearson.

[7] Duncan, B. (2020). 2020-05-28 Traffic Analysis Exercise. *Malware Traffic Analysis*. [Online]. Available: https://www.malware-traffic-analysis.net/. [Accessed: April, 2026].

[8] Greer, C. (2022). Wireshark Tutorial: Fixing Slow Applications. *YouTube*. [Online]. Available: https://www.youtube.com/watch?v=DDMOY3RTfGw. [Accessed: April, 2026].

[9] Google. (2025). Gemini 3 Pro Model Card. *Google DeepMind*. [Online]. Available: https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf . [Accessed: March, 2026].

[10] Anthropic. (2025). Claude 4.5 Sonnet System Card. *Anthropic*. [Online]. Available: https://www-cdn.anthropic.com/963373e433e489a87a10c823c52a0a013e9172dd.pdf. [Accessed: March, 2026].

[11] OpenAI. (2025). GPT-5 System Card. *OpenAI*. [Online]. Available: https://arxiv.org/html/2601.03267v1. [Accessed: March, 2026].

6. APPENDIX

Prompts

Case Study I

I.1Parse the file with path C:\Users\pathto\wireshark-labs-9e\http-wireshark-trace1-1.pcapng and answer the following questions:

1. Is my browser running HTTP version 1.0, 1.1, or 2? What version of HTTP is

the server running?

2. What languages (if any) does my browser indicate that it can accept to the

server?

3. What is the IP address of my computer? What is the IP address of the

gaia.cs.umass.edu server?

4. What is the status code returned from the server to my browser?

5. When was the HTML file that I am retrieving last modified at the server?

6. How many bytes of content are being returned to my browser?

7. By inspecting the raw data in the packet content window, do you see any headers

within the data that are not displayed in the packet-listing window? If so, name

one.

I.2. Parse the file with path C:\Users\pathto\wireshark-labs-9e\dns-wireshark-trace1-1.pcapng and answer the following questions:

1. Locate the first DNS query message resolving the name gaia.cs.umass.edu. What

is the packet number6 in the trace for the DNS query message? Is this query

message sent over UDP or TCP?

2. Now locate the corresponding DNS response to the initial DNS query. What is the packet number in the trace for the DNS response message? Is this response message received via UDP or TCP?

3. What is the destination port for the DNS query message? What is the source port of the DNS response message?

4. To what IP address is the DNS query message sent?

5. Examine the DNS query message. How many “questions” does this DNS message contain? How many “answers” answers does it contain?

6. Examine the DNS response message to the initial query message. How many “questions” does this DNS message contain? How many “answers” answers does it contain?

I.3 Parse the file with path C:\Users\pathto\wireshark-labs-9e\tls-wireshark-trace1.pcapng and answer the following questions:

1. What symmetric key cryptography algorithm is being used by the client and server to encrypt application data (in this case, HTTP messages)?

2. In which of the TLS messages is this symmetric key cryptography algorithm finally decided and declared?

3. What is the packet number in your trace for the first encrypted message carrying application data from client to server?

4. What do you think the content of this encrypted application-data is, given that this trace was generated by fetching the homepage of www.cics.umass.edu?

Case Study II

Act as a Senior Network Security Analyst. Access the packet capture file at the following path: C:\Users\pathto\2020-05-28-traffic-analysis-exercise.pcap.

Perform a comprehensive forensic investigation of the traffic. Your goal is to determine if this network has been compromised by malware.

Please provide:

An executive summary of your findings.

A detailed breakdown of any suspicious activities, identifying the infected host(s) and the nature of the threat.

The specific evidence that led to your conclusion.

Case Study III

C:\Users\pathto\WiresharkTutorial.pcapng"

Βρες τις καθυστερήσεις που υπάρχουν στο παραπάνω αρχείο και πες μου σε τι οφείλονται? Ποια είναι η μεγαλύτερη καθυστέρηση που βρίσκεις?

Originally asked in Greek, but translates to:
Find the delays present in the above file and tell me what they are due to?

What is the longest delay you find?