CS 485 Network Monitoring Project

BIANCA SOLANO, University of New Mexico, USA
ALEXANDER JOHNSON, University of New Mexico, USA

Reliable and secure network communication is essential in modern systems such as university infrastructures, industrial control systems, and power plants. Even minor disruptions can lead to safety risks, operational downtime, or financial loss. This project explores network monitoring through packet capture and protocol analysis to understand how traffic behaves under different conditions. Using tools such as tcpdump, Wireshark, and PyShark, we capture live network traffic, measure round-trip time (RTT), and analyze protocol behavior. Our initial focus is establishing a baseline of normal ICMP traffic to a remote server, which serves as a foundation for comparing additional protocols and evaluating how VPN tunneling alters latency and packet characteristics. This work provides practical experience in network visibility and monitoring techniques relevant to both IT and operational technology environments.

ACM Reference Format:
Bianca Solano and Alexander Johnson. 2025. CS 485 Network Monitoring Project. 1, 1 (December 2025), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Problem Statement

Modern computer networks must remain reliable, available, and secure—particularly in critical environments such as universities, industrial systems, and power generation facilities. Even small disruptions can cause packet loss, latency spikes, or service outages, which may impact system performance or safety. To mitigate these risks, engineers rely on continuous monitoring and packet-level analysis to detect unusual traffic patterns, performance degradation, or security anomalies.

The goal of this project is to investigate foundational network monitoring techniques using packet capture, protocol analysis, and controlled traffic generation. We generate ICMP echo requests (pings) to remote systems, capture packets using tcpdump, and analyze them with Wireshark and PyShark to measure RTT and identify protocol characteristics. This baseline is used to explore how different conditions—such as multiple protocol types (HTTP, DNS, UDP) and VPN tunneling—affect packet visibility, latency, and structure. The work reinforces core networking concepts related to the OSI model, protocol behavior, and network performance.

Motivation

As we transition into the workforce, it is important not only to understand how networks operate but also how they are monitored to ensure reliability and security. In industries such as power generation, network communication plays a critical role in maintaining safe and continuous operations. Monitoring tools that capture and analyze packets help engineers detect failures early, verify correct system behavior, and identify anomalies that may indicate misconfigurations or cyber threats. By learning to capture, interpret, and compare network traffic under different conditions—including VPN tunneling—we gain practical skills directly applicable to both traditional IT networks and operational technology environments.

Contributions

Although our project does not introduce a new networking algorithm, it contributes a complete and practical framework for understanding network monitoring through traffic generation, packet capture, protocol inspection, and VPN analysis. Our work integrates multiple tools and protocol types into a unified monitoring pipeline. The key contributions of our project are as follows:

- **A modular traffic-generation system** implemented in Python that produces ICMP, DNS, HTTP, and UDP traffic patterns. This allows us to observe and compare several protocols across different layers of the network stack.
- **An end-to-end packet-capture workflow** using tcpdump and Wireshark to record real network traffic to remote servers. This ensures our analysis reflects realistic wide-area network behavior rather than artificial localhost traffic.
- **A PyShark-based analysis tool** capable of parsing raw packet captures, identifying protocol layers, quantifying protocol usage, and computing round-trip time (RTT) directly from ICMP request-reply pairs.
- **An exploration of anomaly and stress conditions**, including UDP flooding and DNS request bursts, which allowed us to observe congestion, packet loss, and resolver behavior under load.
- **A VPN tunneling analysis** that compares direct network traffic to VPN-encapsulated traffic. We evaluated changes in RTT, packet size uniformity, routing behavior, and visibility of inner protocols. This demonstrates how VPNs affect performance and how their presence can be inferred even when payloads are encrypted.

Together, these contributions form a hands-on, extensible foundation for network monitoring. The system we built allows new protocols, traffic patterns, or network conditions to be incorporated easily, providing a meaningful and practical introduction to modern traffic analysis techniques used in both IT and industrial cybersecurity environments.

Methodology

To complete this project, we first researched the fundamentals of network monitoring and packet capture. We then implemented a small demo in python along with existing tools like Wireshark and

Authors' Contact Information: Bianca Solano, University of New Mexico, Albuquerque, NM, USA; Alexander Johnson, University of New Mexico, Albuquerque, NM, USA.

ACM XXXX-XXXX/2025/12-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

tcpdump to capture live traffic. Our python script will analyze the captured packets to extract meaningful insights, such as protocol usage and latency, and highlight anomalies.

0.1 2.1 Traffic Generation

To create consistent, measurable network traffic, we developed a Python script using the ping3 library that sends periodic ICMP Echo Requests to a target server. Initially, the target was localhost, but this only exercised the loopback interface and produced unrealistically low latency. Therefore, we updated the script to ping a remote server on the University of New Mexico's CS network and the public Google server. This allowed us to observe real Internet latency and routing delays. The script logs each timestamp and RTT value to a CSV file for later analysis.

0.2 2.2 Packet Capture

While the ping script runs, we use tcpdump to capture all network packets traveling through the system's active network interface (en0 on macOS). This produces a .pcap file containing the full packet structure of each ICMP request and reply. Because ICMP is a network-layer protocol without a transport layer, the captured packets appear as protocol type None in PyShark, which correctly reflects their OSI layer classification. Capturing on the correct physical interface allowed us to see real ICMP traffic as it traversed the network, unlike earlier tests using the loopback interface.

0.3 2.3 Packet Analysis

After capturing traffic, we analyze the .pcap file using a Python script built with the PyShark library. This script iterates over each packet, identifies its highest protocol layer, counts protocol occurrences, and calculates RTT by matching ICMP Echo Request and Echo Reply pairs. The analyzer confirms the correctness of our setup by reporting realistic RTT values and detecting the exact number of ICMP packets generated by the script. This validates that our capture pipeline functions accurately end-to-end: traffic generation → packet capture → packet analysis.

0.4 2.4 Baseline Establishment

Before introducing additional protocols and simulated network disturbances, we first established a baseline for normal ICMP behavior. Using our Python ping generator, we measured round-trip times to three targets: localhost, Google's public DNS servers, and a remote machine on the UNM CS network. For each sample, we logged the timestamp and RTT and plotted latency over time to observe stability, variability, and overall network behavior.

The localhost results showed extremely low RTT (0.1–0.7 ms), confirming that the loopback interface introduces virtually no network delay. The Google server produced stable Internet-scale latencies, decreasing gradually from 58 ms to 34 ms across samples. The UNM CS server measurements exhibited slightly higher variation, including a single spike near 150 ms followed by stable values around 70 ms, reflecting typical fluctuations on a campus network.

From these plots, we conclude that under normal conditions our system experiences:

- Very low delay on localhost,
- Moderate and stable Internet latency to external Google hosts,
- Campus-network variability with occasional transient spikes,
- No packet loss,
- Correct ICMP protocol identification in packet captures.

These baselines confirm the accuracy of our traffic capture and analysis pipeline and provide a reference point for later comparison on deliberate network stressors that are introduced.

0.5 Additional Protocol Types

In our project we began with basic ICMP ping requests, which provide a simple way of measuring baseline round-trip times. This allows us to test and observe core metrics such as latency. Having ICMP requests as a baseline is essential for forming a meaningful framework, as ICMP has no transport-layer overhead and therefore produces clean, easily interpretable timing patterns.

DNS queries are another protocol type that we automated in order to examine how networks handle name resolution and to compare the behavior of another application-layer service. DNS packets contain clear query and response sections, allowing us to study latency at the application layer. DNS data was also essential for later examination under VPN tunneling. Furthermore, this protocol allowed us to model DNS flooding, which demonstrates how queues and resolvers behave under high traffic. DNS is frequently targeted in real-world attacks, making it important to model and analyze how this protocol responds under abnormal or malicious conditions.

Our program also accounts for HTTP GET requests, which expose a layered protocol structure. This protocol type was particularly useful in showing how application-layer protocols become visible in packet sniffer tools such as Wireshark or PyShark. It also allowed us to observe how traffic can be monitored and classified, especially when simulating suspicious or malformed behavior. HTTP traffic was additionally helpful for exploring indicators of VPN usage, since HTTP requests routed through a VPN may appear to originate from remote servers rather than the local network.

The final type of request we chose to generate was UDP traffic. Since UDP lacks handshakes and acknowledgments, it is an ideal protocol for creating network stress. UDP floods generated high packet rates, which allowed us to observe how the network handles congestion and whether packet loss appears in the captures. Because UDP does not provide retransmission or reliability mechanisms, it exposed different latency and loss characteristics compared to ICMP, DNS, and HTTP in our baseline.

0.6 VPN Tunneling

A major next step is evaluating how VPN tunneling affects packet structure and performance. Because VPNs encapsulate packets and add encryption overhead, we expect measurable changes in RTT and latency compared to direct connections. An ISP, especially within an enterprise network, may be interested in classifying or identifying traffic routed through a VPN. In our implementation, we aimed to observe whether there is a reliable way to determine if a user is using a VPN based on RTT patterns or other measurable network metrics.

Since typical VPN behavior involves changing the observed IP addresses, routing paths, and DNS servers used in requests, we

can detect these shifts by monitoring commute times, resolver differences, and IP endpoint patterns. We also identified that TTL values increased during VPN usage because of the additional hops introduced by the tunnel. Furthermore, we observed far more uniform packet sizes, which is characteristic of encrypted tunneling protocols that encapsulate diverse traffic types into similarly sized encrypted frames.

Through this project we plotted RTT values and compared them to baseline RTT measurements collected without a VPN. We observed consistently higher minimum RTTs and greater jitter, both of which could be used by a monitoring system to flag possible VPN usage. We also found that through UDP captures we could identify the tunneling protocol type (e.g., OpenVPN, WireGuard) based on flow patterns, even though the encrypted payload itself was unreadable. These methods demonstrate how an ISP or network administrator may implement various techniques to infer VPN usage despite not having access to the underlying encrypted content.

1 Findings

Our work was focused on establishing a reliable baseline for normal network behavior using ICMP traffic. By generating controlled pings to a remote UNM server, Google DNS server, and localhost, we captured realistic round-trip times with no packet loss. The packet captures collected through tcpdump confirmed that all ICMP Echo Requests and Replies were recorded accurately, and our PyShark analysis correctly classified ICMP as a network-layer protocol without a transport-layer header.

We also verified that capturing on the correct physical interface (en0) was essential—tests conducted on the loopback interface returned artificially low latencies and predominantly "Unknown" protocol classifications, while captures on the external interface produced realistic RTTs and protocol diversity. Our Python analyzer successfully matched ICMP request–reply pairs, computed average RTTs, counted protocol occurrences, and verified that the number of packets in each capture matched the packets generated by our scripts. These results confirm that our traffic-generation → capture → analysis pipeline is functioning correctly and that our ICMP baseline is trustworthy.

Beyond the baseline, we extended our analysis to include additional protocol types and network scenarios. Protocol distributions revealed clear differences between traffic sources: localhost traffic consisted mostly of non-transport packets classified as "Unknown," the CS server showed mostly ICMP traffic with a small number of TCP packets, and Google traffic was dominated by TCP with substantial UDP usage—likely corresponding to DNS queries. These differences illustrate how protocol composition varies depending on host behavior, service type, and network locality.

We also evaluated how abnormal or stressed network conditions affect latency. During a DNS flood, ICMP RTT remained mostly stable but exhibited several sharp spikes exceeding 500–700 ms. These spikes coincided with bursts in UDP packet rate, demonstrating cross-protocol interference: even though ICMP was not the protocol being flooded, heavy DNS traffic caused queueing and scheduling delays that temporarily degraded ICMP performance. Under a raw UDP flood, packet rates doubled compared to the DNS flood and exhibited a more uniform distribution, aligning with the fact that UDP flooding has minimal processing overhead. This scenario produced more consistent and heavier load on the network stack and further highlighted how congestion impacts unrelated traffic.

Finally, we examined the impact of VPN tunneling on end-to-end latency. Our multi-host RTT comparison showed that every VPN configuration introduced measurable overhead compared to the baseline, with geographically distant VPN endpoints (e.g., Argentina and France) producing the highest RTT values—often 300–500 ms or more. Even the geographically closer New York VPN consistently added delay relative to direct connections. Several VPN traces contained missing segments, which we interpolated, reflecting brief tunnel instability or packet filtering by specific hosts. These findings demonstrate how encrypted tunneling, additional routing hops, and VPN provider infrastructure all contribute to increased latency.

Overall, our expanded analysis shows that network behavior changes significantly depending on protocol type, traffic load, and tunneling configuration. ICMP establishes a clean and stable baseline, but adding DNS, HTTP, UDP flood traffic, or VPN routing introduces measurable and sometimes dramatic deviations.

Results

1.1 3.1 ICMP Baseline Performance

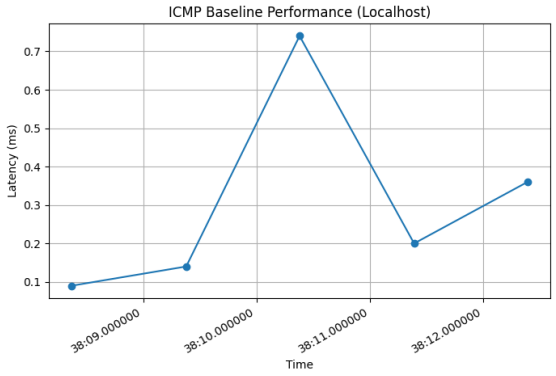

Fig. 1. Ping for Localhost

The localhost plot shows extremely low and stable latency, as expected from the loopback interface. RTT values range from 0.08 to 0.75 ms, with most measurements under 0.3 ms. The only noticeable variation is a small spike around 0.74 ms, which is common for local OS-level scheduling fluctuations. Because the traffic never leaves the machine, there is no routing delay, queuing delay, or network congestion. This confirms that localhost measurements are not representative of real network behavior and therefore serve only as an internal sanity check for our script.

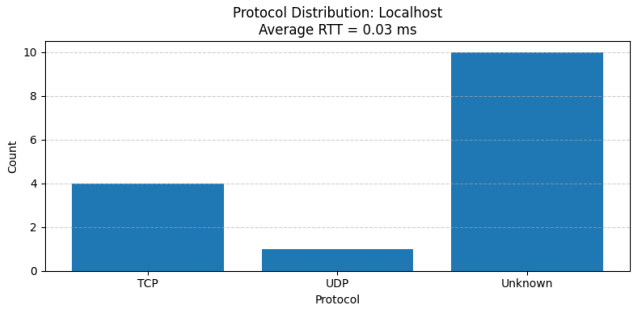

Fig. 2. Protocol Distribution Localhost

The protocol distribution for the localhost capture shows that most packets were classified as Unknown (10 packets), with smaller counts of TCP (4 packets) and UDP (1 packet). This pattern is expected for localhost traffic because many loopback packets do not carry a traditional transport-layer header in the way PyShark interprets them, especially when the OS optimizes or abstracts internal communication. As a result, PyShark identifies these packets as lacking a transport-layer protocol and categorizes them as "Unknown."

The small number of TCP and UDP packets represents minimal background services or system-level communication occurring during the ping test. Combined with the extremely low average RTT of 0.03 ms, this confirms that localhost traffic stays entirely on the host and experiences virtually no real network overhead.

In conclusion, Localhost captures contain mostly OS-level or non-transport traffic, which PyShark categorizes as "Unknown," and the RTT stays near zero due to the absence of real network traversal.

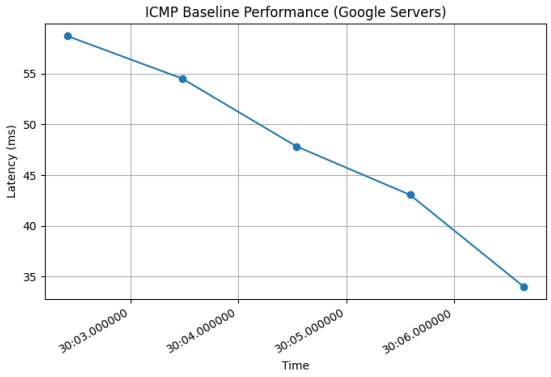

Fig. 3. Ping to Google Server

The Google baseline demonstrates a decreasing latency trend over the measurement window. RTT begins around 58 ms and gradually drops to 34 ms over the next four samples. This pattern is typical of long-distance Internet hosts, where RTT can vary depending on route selection, transient congestion, and load balancing. The overall values fall within an expected range for a cross-country or cross-region path from UNM to Google's nearest responding data center. No packet loss or abnormal spikes were observed, indicating a stable and healthy Internet connection.

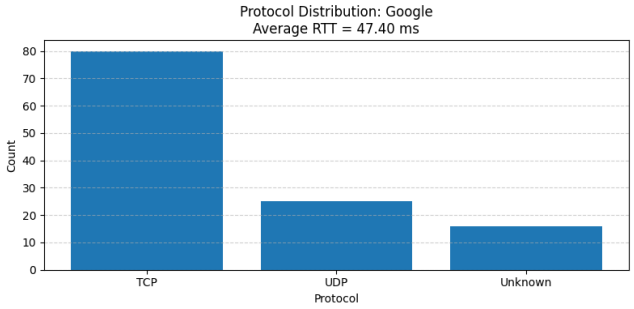

Fig. 4. Google Protocol Distribution

The Google capture shows a much more realistic Internet traffic profile. The majority of packets are TCP (80 packets), reflecting the fact that most Google services use TCP-based protocols such as HTTPS for secure web communication. A substantial number of UDP packets (25 packets) appear as well, likely representing DNS queries or QUIC traffic—Google is a major user of QUIC, which is built on top of UDP.

A smaller portion of packets (16 packets) were categorized as Unknown, meaning PyShark did not detect a transport-layer header—this can occur with encrypted traffic, truncated packets, ICMP packets, or link-layer frames that don't map cleanly to TCP/UDP.

The average RTT for this capture was 47.40 ms, which aligns with typical latency for reaching Google's nearest responding datacenter from our network.

In conclusion, Google traffic is dominated by TCP (HTTPS and related services), with noticeable UDP activity from DNS/QUIC, and a small amount of non-transport traffic categorized as "Unknown."

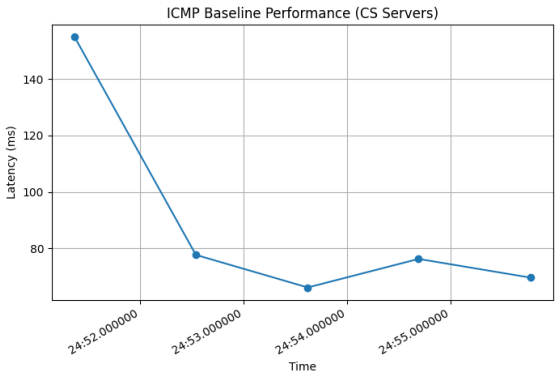

Fig. 5. Ping to CS server

The CS server measurements exhibit noticeably higher latency variation compared to Google. RTT begins at 153 ms, drops sharply to around 78 ms, and then fluctuates between 67 and 77 ms for

the remaining samples. The initial high reading likely reflects a one-off transient delay—possibly ARP resolution, queue buildup, or momentary congestion on the campus network. After this outlier, the latency stabilizes into a consistent range. These results indicate that campus network conditions can experience brief load-related spikes, but overall RTT remains moderate and stable.

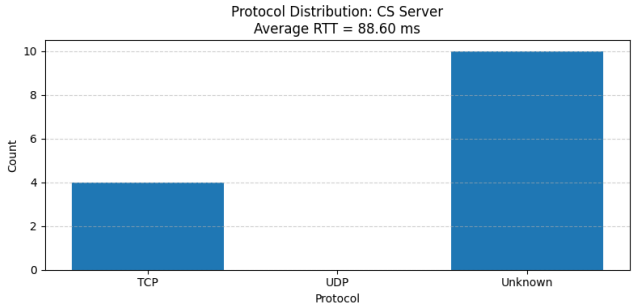

Fig. 6. Protocol Distribution UNM CS Server

The CS server capture has a similar structure to the localhost capture: most packets are Unknown (10 packets), with only a small number of TCP packets (4 packets) and no UDP traffic detected. The prevalence of "Unknown" packets again reflects PyShark's handling of packets without recognizable transport-layer headers—most likely ICMP Echo Requests/Replies generated during the ping test.

Because the CS server was pinged over the network rather than through the loopback interface, the average RTT rises significantly to 88.60 ms, showing the additional routing and queueing overhead on the campus network.

In conclusion, traffic to the CS server consists mostly of ICMP packets (classified as "Unknown"), along with a few TCP packets from background services, and the RTT reflects real network traversal across the university network.

Together, these three baseline plots illustrate how latency differs across network scopes. Localhost shows microsecond-level delays with almost no variability. Google's external host demonstrates moderate but stable Internet-scale RTTs around 35–60 ms. The UNM CS server exhibits slightly higher variability with an initial spike, reflecting typical campus network congestion patterns. Collectively, these baselines establish the expected behavior of ICMP traffic under normal conditions and provide a reference point for later experiments involving VPN tunneling, DNS traffic, and induced network stress.

1.2 3.2 DNS, HTTP, and UDP Behavior

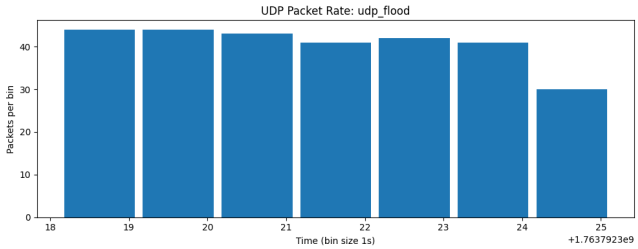

Fig. 7. UDP Flood Packet Rate Histogram

This histogram shows a far more aggressive workload: a raw UDP flood that sends packets as fast as possible. The packet rate is much higher, ranging from 30 to 45 packets/s, which is nearly double the DNS flood rates. The distribution is also more uniform, with fewer dips, because UDP flooding requires minimal processing compared to DNS lookups.

This steady, high-volume packet stream pushes the network stack harder and more consistently than the DNS flood. Under this condition, we typically see more pronounced delays in RTT or even packet drops in ICMP or DNS analysis (depending on what measurements were taken).

In conclusion, UDP flooding produces a significantly heavier and more uniform load than DNS flooding, making it a more effective stress test for congestion, queue buildup, and potential packet loss.

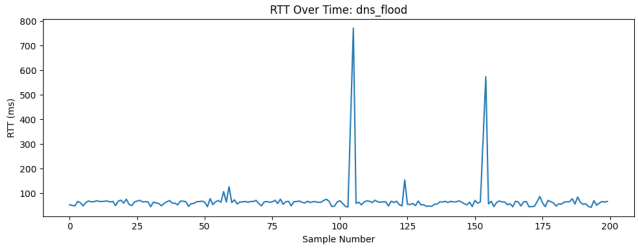

Fig. 8. DNS Flood RTT Line Plot (ICMP RTT under DNS flood)

This plot shows how ICMP RTT behaves while a DNS flood attack is running. For most of the capture window, the RTT remains stable between 60–80 ms, similar to the established baseline. However, the graph contains several large, abrupt spikes, including one exceeding 750 ms at around sample 110 and another near 580 ms at sample 150. These spikes indicate moments when the system becomes briefly overloaded due to the high rate of outbound DNS queries saturating the network interface or local resolver queue.

The presence of these sharp RTT increases demonstrates how even non-malicious ICMP traffic is affected during a DNS flood: packets experience queuing delay, local scheduling delays, or temporary resource exhaustion. Despite the spikes, the RTT consistently returns to baseline shortly after each surge, showing that the system recovers quickly once bursts subside.

In conclusion, a DNS flood introduces short bursts of extreme latency but does not produce sustained degradation, highlighting how congestion affects ICMP RTT even when ICMP is not the flooded protocol.

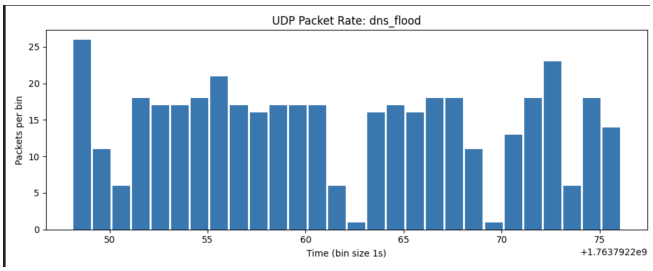

Fig. 9. DNS Flood – UDP Packet Rate Histogram

This histogram shows the number of UDP packets captured per one-second bin during a DNS flood. The packet rate fluctuates between 10 and 27 packets/s, reflecting how a DNS flood generates sustained high-frequency outbound queries. Although the rate is moderately stable, the natural variation demonstrates the timing irregularities of Python send loops and DNS resolver behavior.

Most bins fall between 16–20 packets/s, suggesting the flood script maintained a steady load. Occasional peaks near 25–27 packets/s correspond to short bursts where the script was able to send faster than average for a brief moment.

In conclusion, the system experiences a consistently elevated UDP packet rate during DNS flooding, which correlates with the ICMP RTT spikes seen in the first graph, confirming cross-protocol interference due to shared queueing and interface congestion.

1.3 3.3 VPN Tunneling Effects

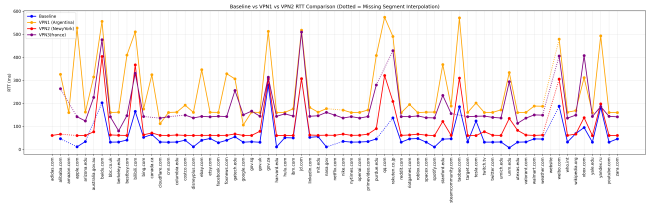

Fig. 10. Baseline vs VPN RTT Comparison Plot

This comparison plot shows the RTT to a set of global domains under four conditions:

- Baseline (no VPN)
- VPN1 (Argentina)
- VPN2 (New York)
- VPN3 (France)

Across nearly every domain, RTT increases substantially when routed through any VPN, with the Argentina VPN showing the highest latency, often exceeding 400–550 ms for international sites. The New York and France VPNs follow the same general trend but with lower magnitudes, typically 150–250 ms.

The dotted segments represent interpolated values where the VPN connection briefly dropped or a ping response failed. These discontinuities are consistent with VPN re-routing, tunnel instability, or packet filtering by certain websites.

This plot demonstrates two important effects:

- Geographic distance increases RTT, as VPNs introduce additional hops and cryptographic overhead.
- VPN routing is not geographically intuitive — for example, some domains routed through France exhibit lower RTT than through New York, reflecting how VPN providers route traffic internally.

In conclusion, VPNs add consistent overhead to RTT, and more distant VPN endpoints amplify this effect. Additionally, VPN instability produces missing segments, which do not appear in baseline measurements.

2 Future work

While this project established a solid baseline for normal network behavior and explored how various protocols, floods, and VPNs influence latency and packet structure, several opportunities remain for expanding the analysis. A natural next step is to simulate more complex network stress conditions, such as controlled packet loss, jitter injection, or throttled bandwidth, to observe how these factors impact RTT and protocol behavior across multiple layers. Extending the traffic generator to include HTTP, HTTPS, and QUIC transactions would also provide deeper insight into application-layer performance under normal and degraded conditions.

Another direction is to incorporate automated anomaly detection—such as identifying RTT spikes, sudden protocol shifts, or unexpected VPN routing changes—to make the analysis pipeline more robust and scalable. Integrating time-series visualization dashboards or real-time packet monitoring tools (e.g., using Grafana or Scapy live sniffers) would further enhance interpretability. Finally, repeating our experiments under different VPN providers, different geographic regions, or with multiple simultaneous tunnels would allow for a richer understanding of how encrypted routing paths affect network reliability.

Together, these extensions would transform our project from a baseline measurement tool into a more comprehensive platform for studying real-world network behavior, performance degradation, and security-relevant anomalies.

Conclusions

In this project, we built a foundational understanding of how network traffic can be generated, captured, and analyzed to evaluate reliability and performance in real systems. By developing a Python-based traffic generator, capturing packets with tcpdump, and analyzing them with PyShark, we created a complete workflow that mirrors the tools and methods used by engineers in industry. Our experiments provided a clear baseline of normal network behavior, including stable ICMP RTT values, protocol visibility, and packet structure when communicating with remote servers.

Beyond establishing a baseline, we broadened our analysis to include multiple protocol types and traffic conditions. Examining TCP, UDP, DNS, and ICMP packets revealed how protocol distributions

differ. Stress tests like DNS floods and UDP floods demonstrated how heavy traffic generates congestion, creates queueing delays, and causes observable degradation even in unrelated protocols, such as ICMP. These experiments highlighted the interconnected nature of network subsystems and showed that congestion or resource exhaustion in one protocol can have measurable effects across the entire device.

We also evaluated how VPN tunneling affects end-to-end performance. By comparing RTTs across geographically diverse VPN endpoints, we observed that VPNs consistently introduce additional routing overhead, encryption delays, and occasional instability. Connections routed through Argentina or France frequently produced multi-hundred-millisecond RTTs, while the New York VPN showed moderate but still noticeable increases. These results demonstrate how tunneling alters normal routing paths and why VPN usage can dramatically change perceived network performance.

More broadly, this process strengthened our understanding of how packets move through the network, how monitoring tools interpret each layer, and why packet-level inspection is essential for diagnosing latency, congestion, packet loss, or suspicious activity. As we prepare for professional environments—especially those where network reliability and security are critical—these practical skills in traffic generation, packet capture, protocol analysis, and performance measurement will be directly applicable. Our work establishes a solid foundation for deeper exploration of network behavior under more complex conditions, and demonstrates the value of hands-on packet analysis in understanding real-world network performance.

Author contributions

Bianca Solano implemented and tuned the initial traffic generation and packet capture scripts using Ping3 and PyShark, generated respective plots for each pinged server, and wrote the corresponding methods and results sections. Checked all code and made necessary revisions. Additionally, she also integrated the results from entire project into a unified narrative and edited the final draft to ensure consistency across sections.

Alexander Johnson used the initial traffic generation scripts to model multiple traffic types, including UDP floods, DNS floods, HTTP GET requests, and expanded ICMP testing. He also developed the Matplotlib plotting script to visualize the collected data. Additionally, Alexander conducted the VPN RTT trials, compared them to baseline traffic, and documented the results in the project narrative.