Analyzing Performance: TCP vs UDP

CS 417 Assignment 1

Overview

In this assignment you will build a small client/server benchmark to analyze and compare the performance of TCP and UDP. You will implement the benchmark in Python and run a series of experiments that vary payload size and request volume. Your report must quantify the performance differences and explain why the observed results occur.

This assignment is not a coding practice. It is designed to encourage experimentation and critical thinking. This resembles a real-world analysis task where you are given a problem statement and must design your own experiments to explore it. Given that there are two solutions (TCP and UDP), there is no single "right answer" to the question of which is faster. You should be analytical and data-driven to conclude which protocol performs better under which conditions and why.

The focus is on experimental methodology and analysis. Your implementation should be correct, repeatable, and able to run on two different machines. You will measure connection setup time (TCP only), per-request turnaround time, and throughput, and use graphs to present your results.

1 Project Goals

- Understand the fundamental performance differences between TCP and UDP.
- Implement comparable TCP and UDP client/server benchmarks in Python.
- Measure and visualize latency and throughput across multiple configurations.
- Explain differences in results using network and transport-layer concepts.
- Follow experimentation practices on ilab machines (instructions for ilab)

2 Experiment Design

Because UDP does not have connections, we define a *work item* that is comparable across TCP and UDP:

- TCP work item: connect $\rightarrow$ send N bytes $\rightarrow$ receive N bytes (echo) $\rightarrow$ disconnect.
- UDP work item: send N bytes $\rightarrow$ receive N bytes (echo).

Repeat the work item X times for each configuration and measure total time, average/median latency, and throughput. This setup exposes TCP connection overhead for small N and large X while showing comparable performance for large N.

Suggested parameters (you may adjust if system limits require):

- Connections: 1, 10, 100, 1,000 (10,000 optional only if your host supports it).
- Requests per connection (TCP): 100.
- Work item repetitions (UDP): 100 per client.
- Payload sizes N: 64B, 512B, 1KB, 4KB, 8KB.

System limits: Avoid exhausting kernel resources or overloading shared iLab machines. If you hit per-user limits, reduce the maximum number of concurrent clients and document the limit in your report.

3 Stage 1: Benchmark Framework

Build a reusable framework that runs experiments and records measurements. Your code should support multiple payload sizes and client counts with minimal changes.

- Define a clear request/response format (e.g., echo) for both TCP and UDP.
- Record timestamps for connection setup (TCP), request send, and response receive.
- Save results in a machine-readable format (CSV or JSON).
- Log enough information to reproduce plots: protocol, client count, payload size, request count, and per-request RTT.

Note: Use `SO_REUSEADDR` (via `setsockopt`) before calling `bind` to allow rapid reuse of ports.

4 Stage 2: TCP Implementation

Implement the TCP client and server. The server should accept concurrent connections and respond with an echo of the payload. The client should support spawning multiple connections and issuing multiple requests per connection.

- Implement `run_tcp_server` and `run_tcp_client` exactly as specified in the boilerplate section.
- Measure connection establishment time (TCP 3-way handshake).
- Measure per-request turnaround time for 1, 10, 100, and 1,000 requests.
- Compute throughput as total bytes transferred divided by elapsed time.

5 Stage 3: UDP Implementation

Implement the UDP client and server with the same work item definition. UDP does not establish connections, so focus on request/response time and throughput.

- Implement `run_udp_server` and `run_udp_client` exactly as specified in the boilerplate section.
- Use a single UDP socket on the client to send requests to the server.
- Measure turnaround time for each work item and total throughput.
- If a payload size fails due to MTU limits, record the failure and explain it.

6 Getting Started

6.1 Requirements

1. Python 3.8+.
2. Two iLab machines with network connectivity.
3. A plotting tool (e.g., matplotlib, gnuplot, or spreadsheet software).

6.2 Required Project Folder and Boilerplate

Create a folder named `tcp_udp_benchmark`. This folder must contain the required boilerplate files listed below. You may add additional files, but you must not rename these files or change the required function signatures.

Required folder layout:

```
tcp_udp_benchmark/
  client.py
  server.py
  run_experiments.py
  results/
    *.csv
  plots/
  README.md
```

The `results/` folder should contain any CSV/JSON output and plots you generate.

6.3 Boilerplate Functions and CLI Interface

You must implement the following functions exactly (same names and parameters).

In `server.py`:

```
def run_tcp_server(bind: str, port: int, log_path: str,
                   payload_bytes: int, requests: int, clients: int) -> None

def run_udp_server(bind: str, port: int, log_path: str,
                   payload_bytes: int, requests: int, clients: int) -> None

def parse_args() -> argparse.Namespace
def main() -> None
```

In `client.py`:

```
def run_tcp_client(host: str, port: int, log_path: str,
                   payload_bytes: int, requests: int, clients: int) -> None

def run_udp_client(host: str, port: int, log_path: str,
                   payload_bytes: int, requests: int, clients: int) -> None

def parse_args() -> argparse.Namespace
def main() -> None
```

Your `main()` must call `parse_args()` and dispatch to the correct TCP/UDP function based on `--proto`.

Required CLI flags (both client and server):

- `--proto tcp|udp`: selects the protocol.
- `--bind` (server only): bind address, e.g., `0.0.0.0`.
- `--host` (client only): server IP or hostname.

- `--port`: TCP/UDP port number.
- `--payload-bytes`: number of bytes per request payload.
- `--requests`: number of requests per client (TCP: per connection).
- `--clients`: number of concurrent client connections (TCP) or senders (UDP).
- `--log`: path to a log file (CSV or JSON).

Your client should generate an echo request with a payload of exactly `payload_bytes` and measure RTT. Your server should echo the payload back unchanged.

6.4 Execution Guidance

- Do not run all clients from a single host if it overloads the machine.
- Limit concurrency if you hit per-user socket or file descriptor limits.
- Record the OS, CPU model, and network type used for your experiments.

7 Deliverables

- Source code for TCP and UDP clients/servers and any scripts used to run experiments.
- A short report (PDF) with methodology, graphs, and analysis.
- Raw measurement data (CSV/JSON) used to generate the graphs.
- A reproducibility script (e.g., `run_experiments.py` or `run_benchmark.sh`) that, from a clean checkout, re-runs the experiments and regenerates the graphs used in the report with a single command; include usage in your README and write outputs to `results/`.

8 Report Requirements

- Describe your experimental setup and machine details.
- Explain your work item definition and justify chosen parameters.
- Include plots with labeled axes and units.
- Discuss why TCP and UDP differ, referencing connection setup, reliability, and MTU limits.

9 Report

Submit a short PDF report that clearly explains your experimental findings. At minimum, your report must include:

- Experimental setup: which ilab machines you used
- test methodology: how you defined the work item and structured your experiments
- Rationale for your chosen parameters (payload sizes, clients, requests).

- Graphs showing the performance differences between TCP and UDP
- A discussion of TCP connection overhead and why it impacts small payloads.
- One paragraph on any anomalies or surprising results and how you verified them.
- Suggested graphs:
  - Latency vs. payload size.
  - Throughput vs. payload size.
  - Time-to-finish vs. payload size.
  - Latency vs. number of connections.
  - Throughput vs. number of connections.
  - Time-to-finish vs. number of connections.
  - Combined latency/throughput/time-to-finish vs. payload size (multiple bars for different client connection counts).
- *Note:* You do not need to include all of the above graphs, but you should include enough to support your analysis and conclusions. The main goal for graphing is to clearly show the performance differences between TCP and UDP across different payload sizes and client counts.
- A detailed discussion of performance difference between TCP and UDP.
  - From materials you read about TCP and UDP, what performance differences did you expect?
  - What's the actual data showing?
  - If the data matches your expectations, explain why. If it doesn't, explain why not.
  - Draw conclusion on when TCP and UDP perform similarly and when they differ, and what are the real-world use cases can benefit from each protocol.
  - *Note:* We encourage to ask questions on course's forum. Hints will be provided if you get stuck.
- One paragraph on any anomalies or surprising results and how you verified them.

Grading focus: correctness, experimental rigor, clarity of presentation, and quality of analysis. A detailed rubric will be provided separately.

Rubric (100 pts total):

- Runnable implementation (30 pts): Client/server run with required CLI, logs produced, and experiments complete without manual patching.
- Experimental methodology & data (20 pts): Clear workload definition, parameter coverage, and raw data saved with enough detail to reproduce findings.
- Report quality & analysis (20 pts): Clear setup, rationale, anomalies discussion, and coherent narrative analysis.
- Reproducible graphs from scripts/results (10 pts): Plots can be regenerated from provided scripts and raw results.
- Quantitative analysis tied to graphs (10 pts): Claims supported by specific values/trends from graphs.

- Real-world TCP/UDP use-case conclusions (10 pts): Strong, evidence-based conclusions on when each protocol is appropriate in practice.

P/S: This assignment is designed to be open-ended to encourage critical thinking and experimentation. You are not expected to perfectly predict all results.

10 Submission

Submit a single ZIP file containing your source code, report PDF, and data files.

11 Due date:

Monday, March 2, 2026 at 11:59 PM EST.

If you have questions, post on the course forum and include details about your environment and steps to reproduce.