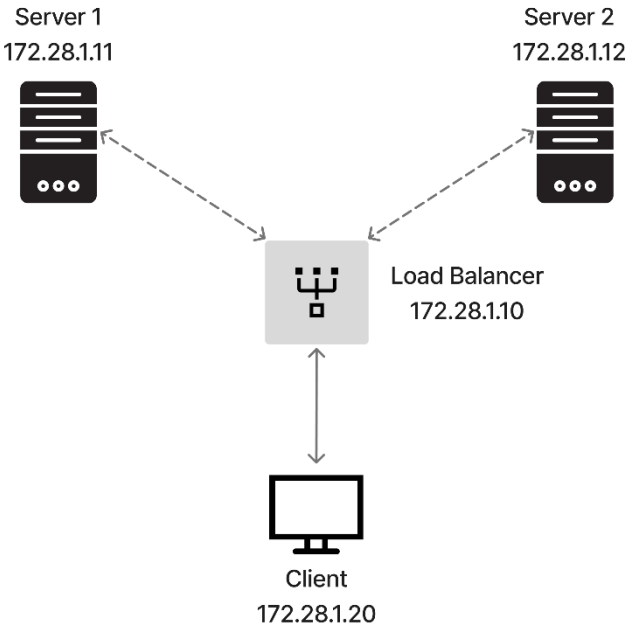
CO513: Advanced Computer Communication Networks - Lab 02

E/19/446: Wijerathna I.M.K.D.I.

09/05/2025

Network Configuration and Setup Summary



A client machine was added to the setup for a clearer and more realistic simulation. Instead of using iperf3, I used Python HTTP servers and clients due to issues with iperf3's dual-connection design (data and control) and single-client limitation, which made consistent load balancing difficult. Since, this approach consists of a single-connection design it provided better control and reliability during testing.

Docker and Docker Compose were used to simulate these nodes within the same virtual network. Please refer the Appendix 1 and 2 for the Docker configurations.

```
[~/CO513/Lab-2]
kalindu ⊟ sudo docker ps
[sudo] password for kalindu:
CONTAINER ID   IMAGE                COMMAND   CREATED              STATUS              PORTS     NAMES
6c3da392631c   lab-2-client         "bash"    About a minute ago   Up About a minute             client
ddca93d08801   lab-2-load_balancer  "bash"    About a minute ago   Up About a minute             load_balancer
dbcacde7622e   lab-2-server2        "bash"    9 days ago           Up About a minute             server2
25dc7e033916   lab-2-server1        "bash"    9 days ago           Up About a minute             server1

[~/CO513/Lab-2]
kalindu ⊟
```

Task 1: Set Up Server Simulations

Logged into the containers for Server 1 and Server 2 and started basic Python HTTP servers.

```
[~/C0513/Lab-2]
kalindu sudo docker exec -it server1 bash
root@25dc7e033916:/# mkdir /web && cd /web
root@25dc7e033916:/web# echo "Hello from Servers!" > index.html
root@25dc7e033916:/web# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

```
[~/C0513/Lab-2]
kalindu sudo docker exec -it server2 bash
root@dbcacde7622e:/# mkdir /web && cd /web
root@dbcacde7622e:/web# echo "Hello from Servers!" > index.html
root@dbcacde7622e:/web# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

From the client container, both servers were accessed using curl for initial availability testing

```
[~/C0513/Lab-2]
kalindu sudo docker exec -it client bash
root@6c3da392631c:/# curl http://server1
Hello from Servers!
root@6c3da392631c:/# curl http://server2
Hello from Servers!
root@6c3da392631c:/#
```

Since an external client was used, a proxy was required to forward traffic from the client through the load balancer to the servers, thus creating the proxy.py script attached in Appendix 3.

The following is a snapshot of logging into the load balancer Docker container and executing the proxy.py script.

```
[~/C0513/Lab-2]
kalindu sudo docker exec -it load_balancer bash
[sudo] password for kalindu:
root@ddca93d08801:/# sudo vim proxy.py
root@ddca93d08801:/# sudo python3 proxy.py
Proxy running on port 80...

=== Server Usage Stats ===
172.28.1.11: 0 requests
172.28.1.12: 0 requests
==========================
```

The tool "wrk" was used for load testing by generating high traffic rates to simulate realistic traffic conditions. The following command was executed to send concurrent requests:
"wrk -t4 -c50 -d30s http://172.28.1.10", which ran a 30-second test with 4 threads and 50 connections against the load balancer at 172.28.1.10.

Initially, traffic was sent directly from the client without a load balancer configured, and the following outputs confirm that only a single server handled the requests.

Client-side traffic generation:

```
root@6c3da392631c:/# wrk -t4 -c50 -d30s http://172.28.1.10
Running 30s test @ http://172.28.1.10
  4 threads and 50 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    23.54ms  111.58ms   1.67s    96.60%
    Req/Sec   316.68    192.53     1.03k    65.54%
  23783 requests in 30.06s, 6.31MB read
  Socket errors: connect 0, read 0, write 0, timeout 6
Requests/sec:    791.26
Transfer/sec:    214.81KB
root@6c3da392631c:/#
```

Proxy-server logs:

```
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:54:13] "GET / HTTP/1.1" 200 -

=== Server Usage Stats ===
172.28.1.11: 0 requests
172.28.1.12: 23790 requests
===========================
```

Task 2: Create Load Balancer Script (Bash)

Then the load_balancer.sh script was created which is attached in Appendix 4.

This script continuously simulates random load values for two servers and selects the server with the lower load by writing its IP to a file (/tmp/selected_server), updating the selection every 5 seconds; the selected server IP is then used by the proxy to forward traffic accordingly.

Load balancer logs: (Indicates simulated random loads and the selected server per each 5 seconds)

```
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 91
Server 2 (172.28.1.12) load: 9
Selected server: 172.28.1.12
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 54
Server 2 (172.28.1.12) load: 17
Selected server: 172.28.1.12
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 18
Server 2 (172.28.1.12) load: 79
Selected server: 172.28.1.11
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 70
Server 2 (172.28.1.12) load: 13
Selected server: 172.28.1.12
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 82
Server 2 (172.28.1.12) load: 51
Selected server: 172.28.1.12
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 45
Server 2 (172.28.1.12) load: 27
Selected server: 172.28.1.12
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 31
Server 2 (172.28.1.12) load: 36
Selected server: 172.28.1.11
------------------------------
Simulated random server loads:
Server 1 (172.28.1.11) load: 0
Server 2 (172.28.1.12) load: 5
Selected server: 172.28.1.11
```

Client-side traffic generation:

```
root@6c3da392631c:/# wrk -t4 -c50 -d30s http://172.28.1.10
Running 30s test @ http://172.28.1.10
  4 threads and 50 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency     9.23ms   47.80ms   1.67s    99.11%
    Req/Sec   295.31    192.56     1.33k    68.91%
  11664 requests in 30.05s, 3.08MB read
  Socket errors: connect 0, read 62, write 0, timeout 16
  Non-2xx or 3xx responses: 62
Requests/sec:    388.12
Transfer/sec:    105.12KB
root@6c3da392631c:/#
```

Proxy-server logs:

```
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:55:41] "GET / HTTP/1.1" 200 -

=== Server Usage Stats ===
172.28.1.11: 6720 requests
172.28.1.12: 4888 requests
: 62 requests
===========================
```

Task 3: Enhance Logic Using Python

Finally, the ai_loadbalancer.py was created which is attached in Appendix 5.

This Python script simulates a load balancer by selecting the server that has received fewer requests—tracked using counts derived from the proxy server logs—and writes the selected server's IP to a file every second. This helps distribute traffic more evenly, and intelligently between the two servers over time.

Load balancer logs: (Indicates the selected server per each 1 second)

```
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
------------------------------
Selected server: 172.28.1.12
------------------------------
------------------------------
Selected server: 172.28.1.11
------------------------------
```

Client-side traffic generation:

```
root@6c3da392631c:/# wrk -t4 -c50 -d30s http://172.28.1.10
Running 30s test @ http://172.28.1.10
  4 threads and 50 connections
  Thread Stats   Avg      Stdev     Max   +/- Stdev
    Latency    14.51ms   78.58ms   1.67s    98.01%
    Req/Sec   286.66    156.85   717.00     67.64%
  16912 requests in 30.04s, 4.48MB read
  Socket errors: connect 0, read 9, write 0, timeout 14
  Non-2xx or 3xx responses: 9
Requests/sec:    562.90
Transfer/sec:    152.78KB
root@6c3da392631c:/#
```

Proxy-server logs:

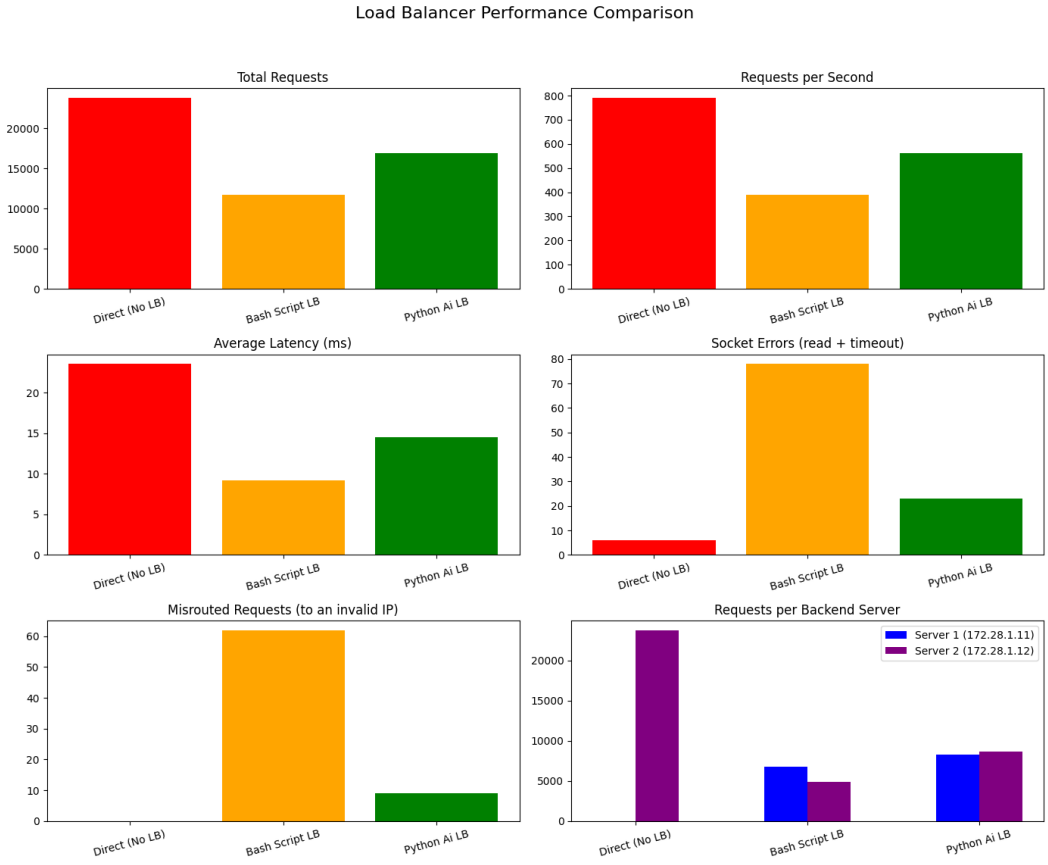
```
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:02] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -
172.28.1.20 - - [18/May/2025 13:59:04] "GET / HTTP/1.1" 200 -

=== Server Usage Stats ===
172.28.1.11: 8262 requests
172.28.1.12: 8651 requests
: 9 requests
===========================
```

Task 4: Analyze Performance and Decisions

The following is the comparison between

- No Load Balancer
- Bash script Load Balancer (Based on random loads)
- Python scrip Load Balancer (Based on history logs and intelligent selection)



Metric	Best Performer	Comments
Throughput	No LB > AI LB > Bash LB	Requests were consistently routed to a single server, avoiding selection overhead in No LB.
Latency	Bash LB > AI LB > No LB	Random selection led to more distributed load early, reducing queuing and keeping latency low in the Bash LB.
Reliability	No LB > AI LB > Bash LB	Fewer rerouting decisions and more consistent routing reduced response failures in No LB.
Distribution	AI LB > Bash LB >>> No LB	AI LB tracked usage and adjusted accordingly, ensuring nearly even server utilization.

Comparing the two Load Balancer implementations;

- AI LB handled load smarter, leading to better throughput and more even server distribution.
- Bash LB had slightly lower latency, likely due to early random load spreading.
- AI LB was more reliable, with fewer errors and failed responses.
- Overall, AI LB scaled better and is more adaptable for real-world use.

Appendix-1: docker-compose.yaml

```yaml
version: '3.8'

services:
  load_balancer:
    container_name: load_balancer
    networks:
      mynetwork:
        ipv4_address: 172.28.1.10
    tty: true
    stdin_open: true
    command: bash
    privileged: true
    build:
      context: .
      dockerfile: Dockerfile

  server1:
    container_name: server1
    networks:
      mynetwork:
        ipv4_address: 172.28.1.11
    tty: true
    stdin_open: true
    command: bash
    privileged: true
    build:
      context: .
      dockerfile: Dockerfile

  server2:
    container_name: server2
    networks:
      mynetwork:
        ipv4_address: 172.28.1.12
    tty: true
    stdin_open: true
    command: bash
    privileged: true
    build:
      context: .
      dockerfile: Dockerfile

  client:
```

```
    container_name: client
    networks:
      mynetwork:
        ipv4_address: 172.28.1.20
    tty: true
    stdin_open: true
    command: bash
    privileged: true
    build:
      context: .
      dockerfile: Dockerfile

networks:
  mynetwork:
    driver: bridge
    ipam:
      config:
        - subnet: 172.28.1.0/24
```

Appendix-2: Dockerfile used for all the nodes (servers, clients, load balancer)

```
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive

RUN apt-get update && apt-get install -y \
    python3 \
    iperf3 \
    iproute2 \
    net-tools \
    iputils-ping \
    bash \
    sudo \
    build-essential \
    libssl-dev \
    git \
    unzip \
    && apt-get clean

# Install wrk from source
RUN git clone https://github.com/wg/wrk.git /opt/wrk && \
    make -C /opt/wrk && \
    ln -s /opt/wrk/wrk /usr/local/bin/wrk
```

Appendix-3: proxy.py

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import requests
import time
import threading

SELECTED_FILE = "/tmp/selected_server"
PORT = 80

# In-memory usage counter
request_count = {
    "172.28.1.11": 0,
    "172.28.1.12": 0
}

class ProxyHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        try:
            # Read selected server IP
            with open(SELECTED_FILE, 'r') as f:
                backend = f.read().strip()

            # Track request count
            if backend in request_count:
                request_count[backend] += 1
            else:
                request_count[backend] = 1  # in case new IP appears

            url = f"http://{backend}{self.path}"

            # Forward request to server
            resp = requests.get(url, timeout=2)

            # Send response headers
            self.send_response(resp.status_code)
            for k, v in resp.headers.items():
                if k.lower() != 'transfer-encoding':
                    self.send_header(k, v)
            self.end_headers()

            # Send response content
            try:
                self.wfile.write(resp.content)
            except BrokenPipeError:
```

```
                pass

        except Exception as e:
            self.send_response(502)
            self.end_headers()
            try:
                self.wfile.write(f"Error: {e}".encode())
            except BrokenPipeError:
                pass


def log_stats():
    while True:
        time.sleep(5)
        print("\n=== Server Usage Stats ===")
        for server, count in request_count.items():
            print(f"{server}: {count} requests")
        print("==========================\n")


if __name__ == "__main__":
    print(f"Proxy running on port {PORT}...")

    # Start background logging thread
    stats_thread = threading.Thread(target=log_stats, daemon=True)
    stats_thread.start()

    # Start HTTP proxy server
    server = HTTPServer(('', PORT), ProxyHandler)
    server.serve_forever()
```

Appendix-4: load_balancer.sh

```
#!/bin/bash

SERVER1="172.28.1.11"
SERVER2="172.28.1.12"
PORT="80"
SELECTED_FILE="/tmp/selected_server"

# Initial target
echo "$SERVER1" > "$SELECTED_FILE"

while true; do
    # Simulate random load values between 0 and 100
    LOAD1=$(( RANDOM % 101 ))
    LOAD2=$(( RANDOM % 101 ))

    echo "------------------------------"
    echo "Simulated random server loads:"
    echo "Server 1 ($SERVER1) load: $LOAD1"
    echo "Server 2 ($SERVER2) load: $LOAD2"

    if [ "$LOAD1" -le "$LOAD2" ]; then
        TARGET="$SERVER1"
    else
        TARGET="$SERVER2"
    fi

    echo "$TARGET" > "$SELECTED_FILE"
    echo "Selected server: $TARGET"

    sleep 5
done
```

Appendix-5: ai_load_balancer.py

```python
import time

SERVER1 = "172.28.1.11"
SERVER2 = "172.28.1.12"
SELECTED_FILE = "/tmp/selected_server"

# Track how many times each server was selected
request_history = {
    SERVER1: 0,
    SERVER2: 0
}

def select_server():
    # Pick the server with fewer requests
    if request_history[SERVER1] <= request_history[SERVER2]:
        return SERVER1
    else:
        return SERVER2

def write_selected_server(server):
    with open(SELECTED_FILE, "w") as f:
        f.write(server)

def main():
    print("AI Load Balancer started using request history tracking...\n")

    while True:
        selected = select_server()
        request_history[selected] += 1
        write_selected_server(selected)

        print("-------------------------------")
        print(f"Selected server: {selected}")
        print("-------------------------------")

        time.sleep(1)

if __name__ == "__main__":
    main()
```