Birzeit University
Faculty of Engineering & Technology
Department of Electrical & Computer Engineering

Socket Programming

Prepared By:
Abdallah Abbasi 1232951
Jamal Shehadeh 1192316
GROUP: T051

Supervised By:
ibrahim nemer

Birzeit
May , 2025

Abstract

Basic socket programming and computer networking skills were practiced in this project. Three primary tasks included the work. In the first task, Wireshark was used to capture packet information and explain and use key network commands. The second task involved using socket programming to create a basic web server that served HTML and CSS files. In the third task, a hybrid system of TCP and UDP sockets was used to create a multiplayer guessing game. Screenshots were taken and every component was tested. The project's objective was to teach students about network programming and collaboration. Every task was finished and recorded in this report.

Table of Contents

List of Tables

List of Figures

Chapter 1

Theory

Contents

1.1 Theory

1.1.1 Socket Programming

Socket programming was used to make communication between devices possible. A socket is a software object that was used to send and receive messages. In this project, TCP and UDP sockets were used. TCP was used when a reliable connection was needed, and UDP was used when fast communication was more important than accuracy.

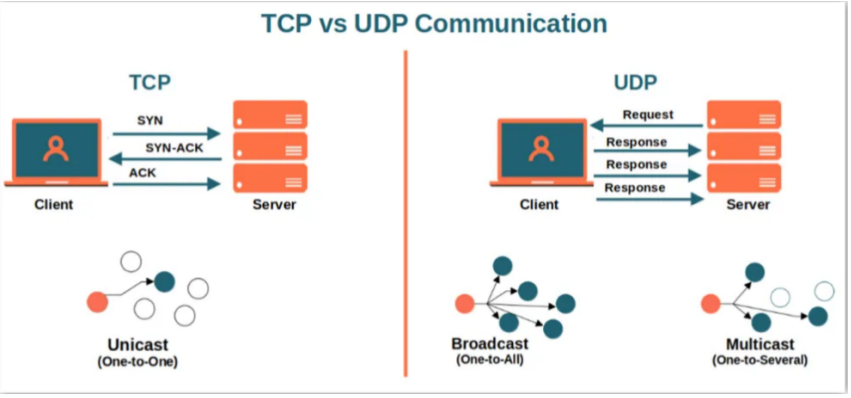


Figure 1.1: Difference between TCP and UDP communication

1.1.2 TCP and UDP

TCP (Transmission Control Protocol) is a connection-based protocol that was used to send data in order. It was helpful when messages had to arrive correctly. UDP (User Datagram Protocol) is a connectionless protocol that was used to send data quickly, but messages could be lost or arrive out of order. Both protocols were used in the game task.

1.1.3 HTTP Protocol

The HTTP (HyperText Transfer Protocol) was used to allow web browsers to talk to the server. When the browser sent a request, the server sent back a response with a file such as an HTML or CSS file. The response also included a status code like 200 (OK), 404 (Not Found), or 307 (Redirect).

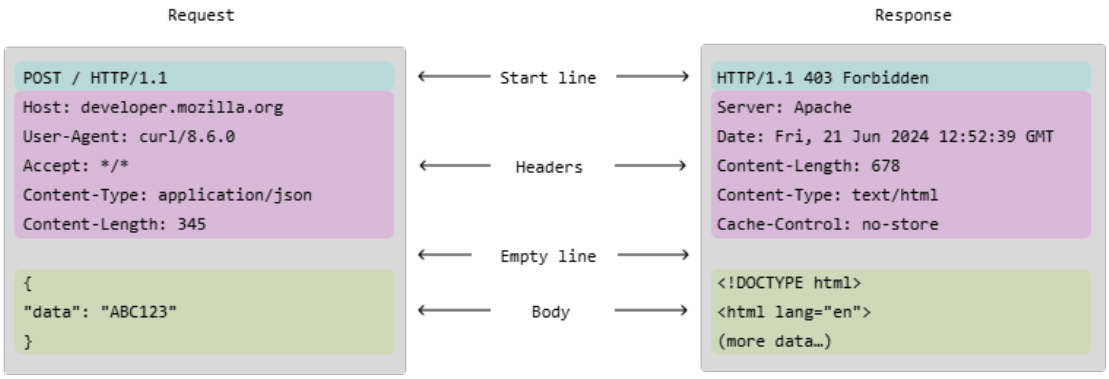


Figure 1.2: Example of HTTP request and response

1.1.4 HTML and CSS

HTML was used to build the content of the web pages. CSS was used to style the pages by changing colours, fonts, and layout. These files were stored in folders and were sent by the server when requested by the browser.

1.1.5 Wireshark and DNS

Wireshark is a tool used to capture data packets sent over the network. It's used in this project to capture a DNS (Domain Name System) query and response when a website was visited. DNS is a system that was used to change a website name into an IP address.

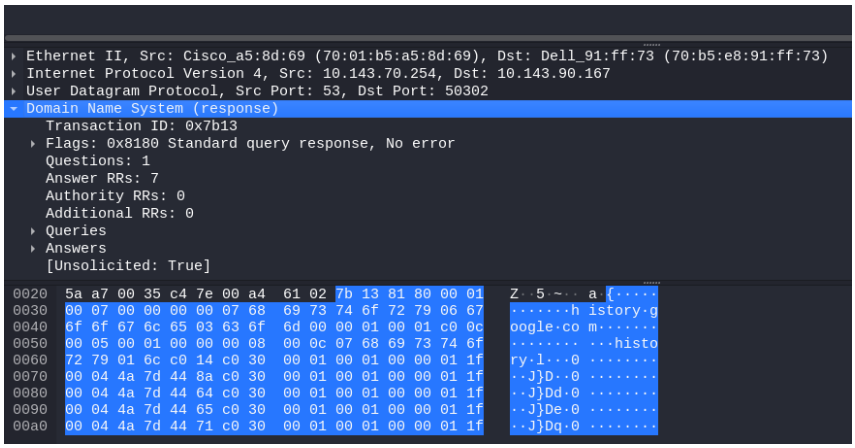


Figure 1.3: Captured DNS query and response using Wireshark

1.1.6 Multithreading

Multithreading is a technique that allows a program to do many things at the same time. In this project, threads were used so that the server could handle multiple players or requests at once without waiting.

1.1.7 HTTP Redirection

HTTP redirection was used when a file was not found. The server sent back a 307 redirect response. This told the browser to go to another URL. For example, if an image or video was not found, the browser was redirected to a Google search page.

Chapter 2

Results and Discussion

2.1 Task 1 – Network Commands and Wireshark

2.1.1 Part A – Explanation of Network Commands

- `ipconfig`: This command displays Internet protocol details for both IPv4 and IPv6 with their addresses, subnet masks, default gateway and DNS suffix. All of this information linked to network adapters.

Figure 2.1: ip config example

- `ping(Packet Internet Grouper)`: writing this command itself displays its manual ,how to use it and what it does if a parameter has been used. Its main use is testing

internet/host reachability, internet connectivity, network interface card and DNS issues.

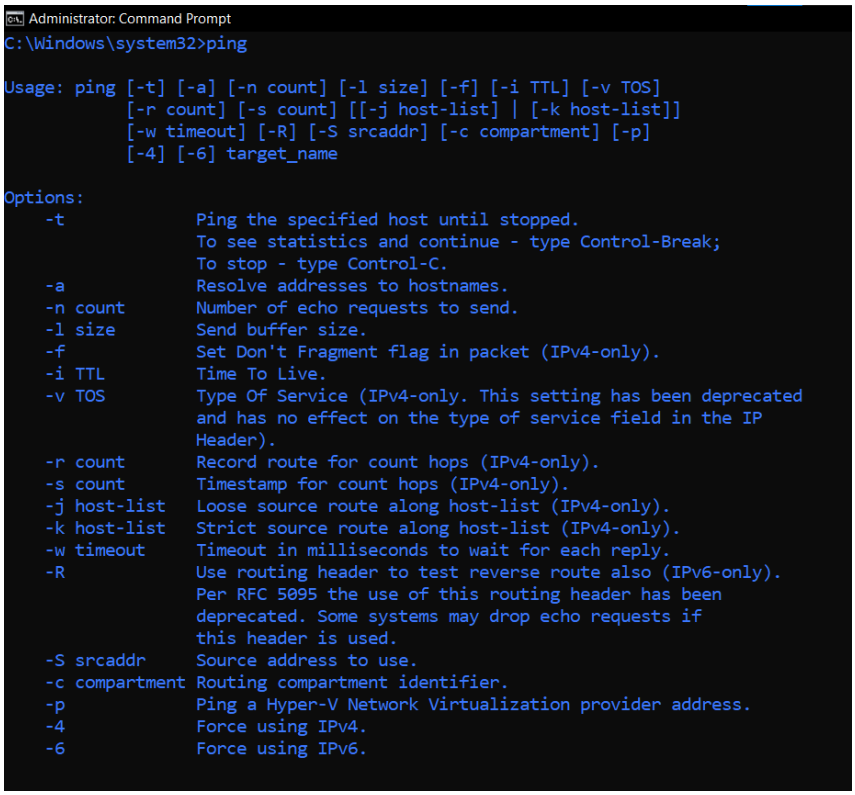


Figure 2.2: ping output

- `tracert` / `traceroute`:It displays the path that packets from my computer take to reach the destination. It is a step-by-step overview of each network hop along the route, using this command without parameters displays the manual.

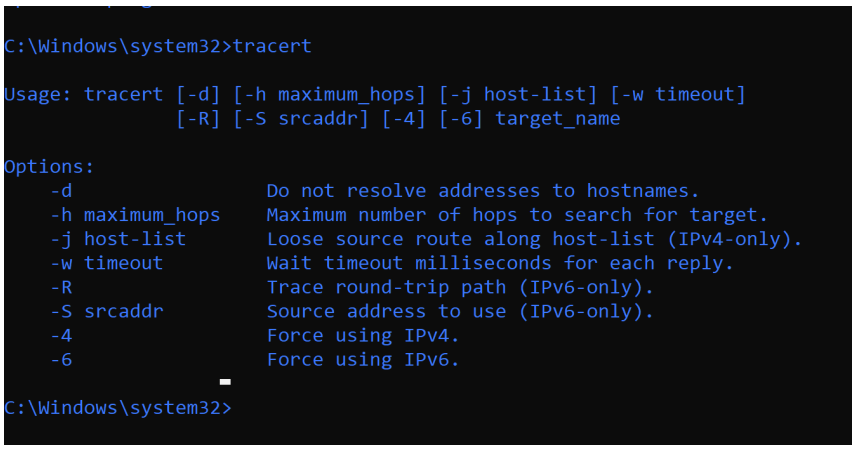


Figure 2.3: tracert output

- `telnet`:It is a terminal emulation program that is used to access remote servers and control them. writing the command without any parameters takes you to the telnet command line itself where you can write commands to connect to a specified server.

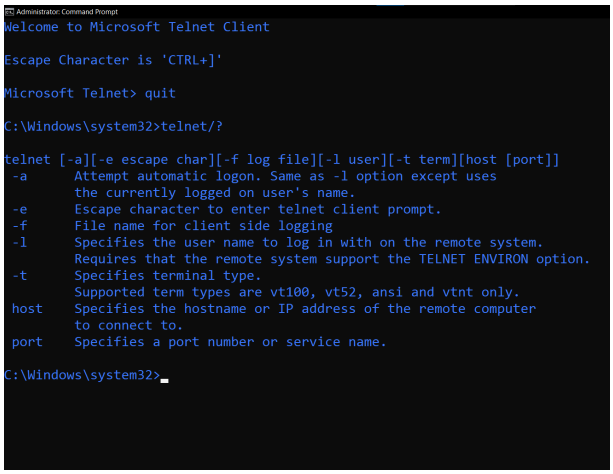


Figure 2.4: telnet output

- `nslookup`: Displays information about DNS (Domain Name System), including default DNS server, name and the address (IP).

Figure 2.5: ns lookup output

2.1.2 Part B – Practical Tests

1. IP Configuration The `ipconfig /all` command was executed to find network configuration details. The following data were obtained:

- IP Address: `192.168.1.18`
- Subnet Mask: `255.255.255.0`
- Default Gateway: `192.168.1.1`
- DNS Server: `192.168.1.1`

Figure 2.6: ipconfig result – page 1

```
Wireless LAN adapter Local Area Connection* 1:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter
   Physical Address. . . . . . . . . : F4-26-79-D5-00-A3
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Wi-Fi Direct Virtual Adapter #2
   Physical Address. . . . . . . . . : F6-26-79-D5-00-A2
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Wi-Fi 6 AX201 160MHz
   Physical Address. . . . . . . . . : F4-26-79-D5-00-A2
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::c697:df0b:8c6d:6baa%5(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.18(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Thursday, May 8, 2025 8:58:39 PM
   Lease Expires . . . . . . . . . . : Friday, May 9, 2025 8:58:38 PM
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCP Server . . . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 66332281
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-2F-46-45-8C-C8-5A-CF-B2-F7-F7
   DNS Servers . . . . . . . . . . . : 192.168.1.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Figure 2.7: ipconfig result – page 2

```
Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Bluetooth Device (Personal Area Network)
   Physical Address. . . . . . . . . : F4-26-79-D5-00-A6
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
```

Figure 2.8: ipconfig result – page 3

2. Ping to a Local Device A ping test was done to a phone connected to the same Wi-Fi network. All packets were received with no loss and an average latency of 22ms. This showed the connection was working well.

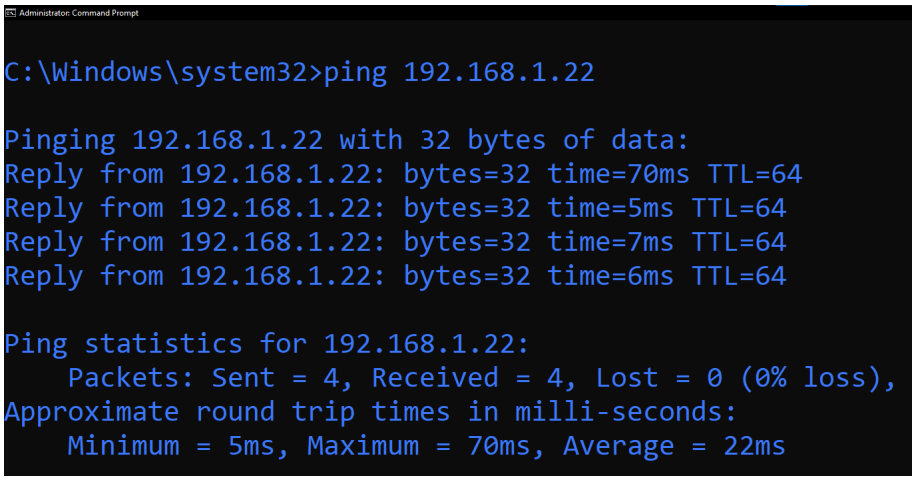


Figure 2.9: Ping to local device at 192.168.1.22

3. Ping to `gaia.cs.umass.edu` The server `gaia.cs.umass.edu` was pinged successfully. Four packets were received with 0% loss. The average round-trip time was 136ms. This means the server was reachable and responded directly.

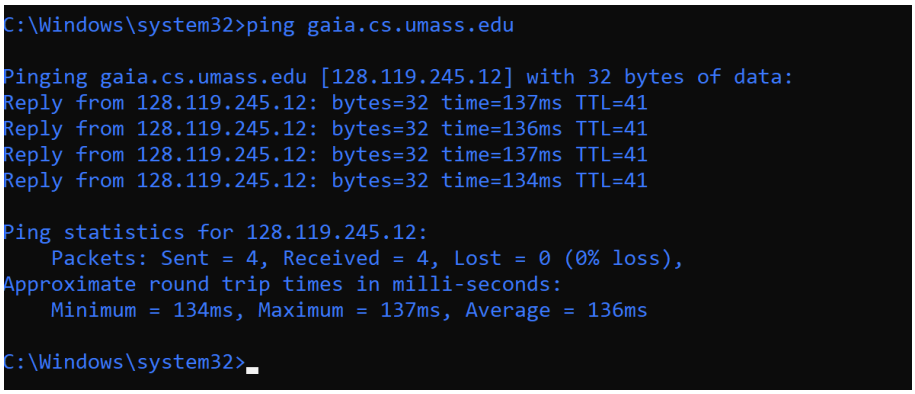


Figure 2.10: Ping to `gaia.cs.umass.edu` (128.119.245.12)

based on the information we received, here's what we learned: When we attempted to connect locally, the latency was very low, similar to that of the smartphone. However, when we tried connecting to a WAN server, the latency increased. This indicates that the packets successfully reached the server without encountering a firewall or any other obstacles, as all four packets were received with 0 loss.

4. Traceroute to `gaia.cs.umass.edu` The traceroute command showed that 25 hops were needed to reach the server. The route passed through different international networks and internet exchange points.

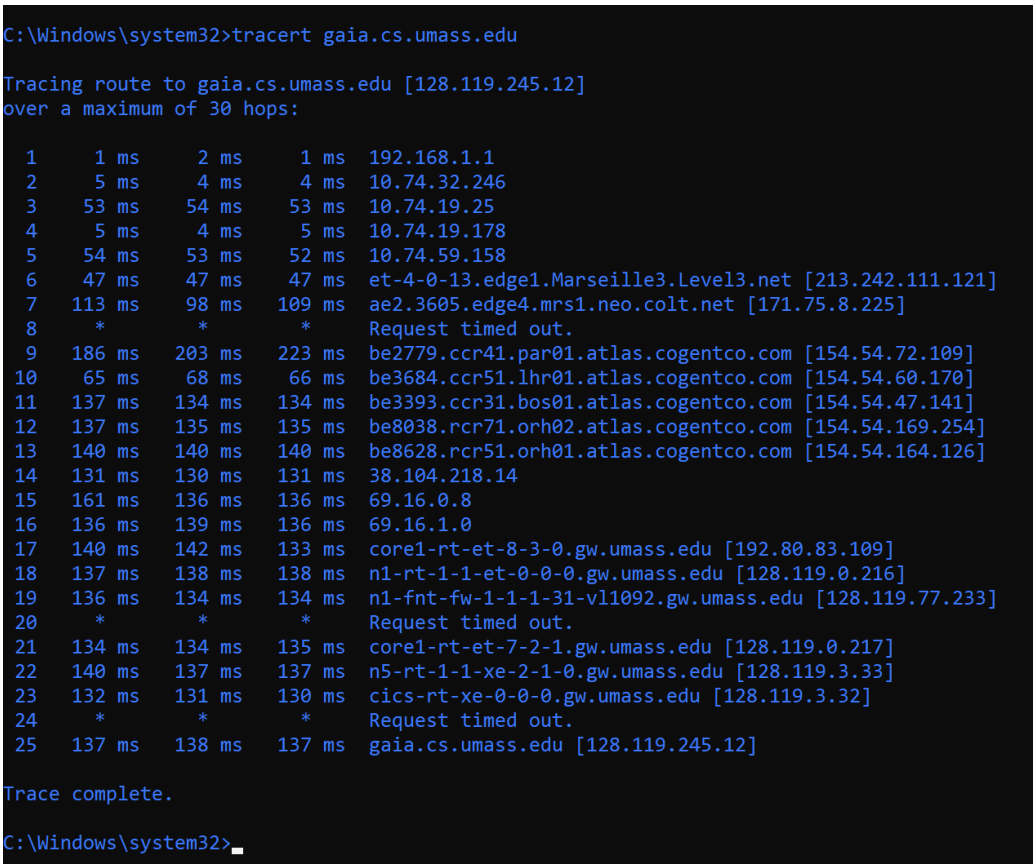


Figure 2.11: Traceroute showing path to `gaia.cs.umass.edu`

5. NSLookup of `gaia.cs.umass.edu` The `nslookup` command was used to get DNS information. The IP address `128.119.245.12` was found for `gaia.cs.umass.edu`. The DNS server used was `192.168.1.1`.

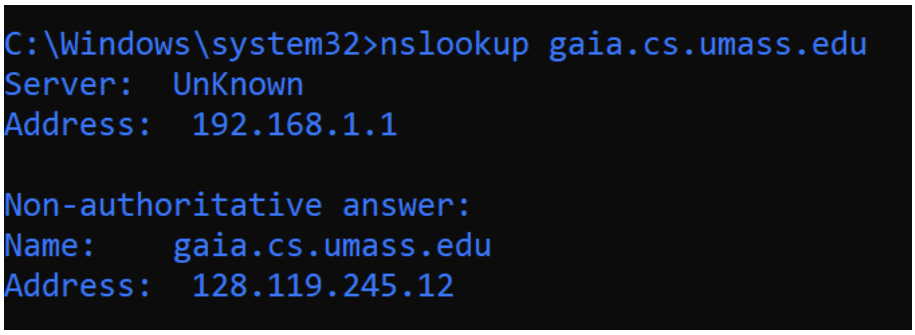


Figure 2.12: NSLookup of `gaia.cs.umass.edu` showing resolved IP

6. Telnet to Port 80 The `telnet gaia.cs.umass.edu 80` command was used. A black screen appeared, meaning the connection to port 80 was successful. This showed that the web server was active and accepting connections.

7. Autonomous System (AS) Information Information about the AS that owns `gaia.cs.umass.edu` was gathered using `https://bgpview.io` and `https://bgp.tools`. The ASN is AS1249 and it belongs to the University of Massachusetts – AMHERST.

- AS Number: AS1249
- AS Name: FIVE-COLLEGES-AS
- Description: University of Massachusetts – AMHERST
- IP Prefixes:
  - 128.119.0.0/16
  - 72.19.64.0/18
  - 192.80.83.0/24
  - 192.189.138.0/24
- Peers: AS1968 (UMASSNET), AS168 (UMass Amherst)
- Upstream: AS1968 (UMASSNET)
- Tier 1 ISP: Not directly connected, but reachable via upstreams like Cogent and GTT
- Number of IPs: 65,536

Figures:

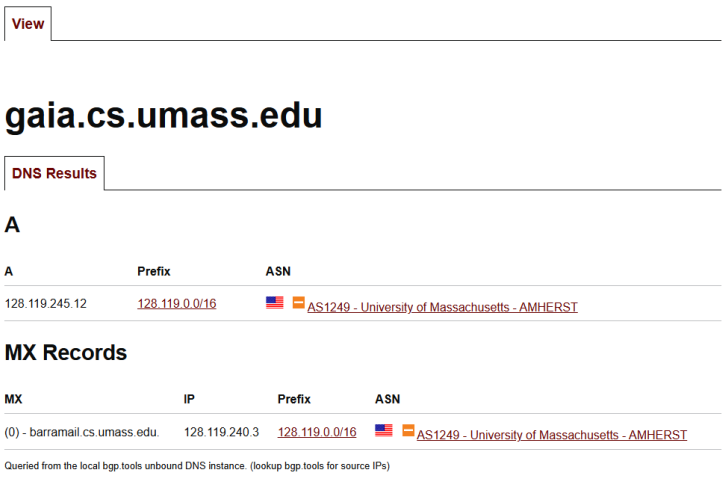


Figure 2.13: ASN topology and Tier 1 ISP connection overview

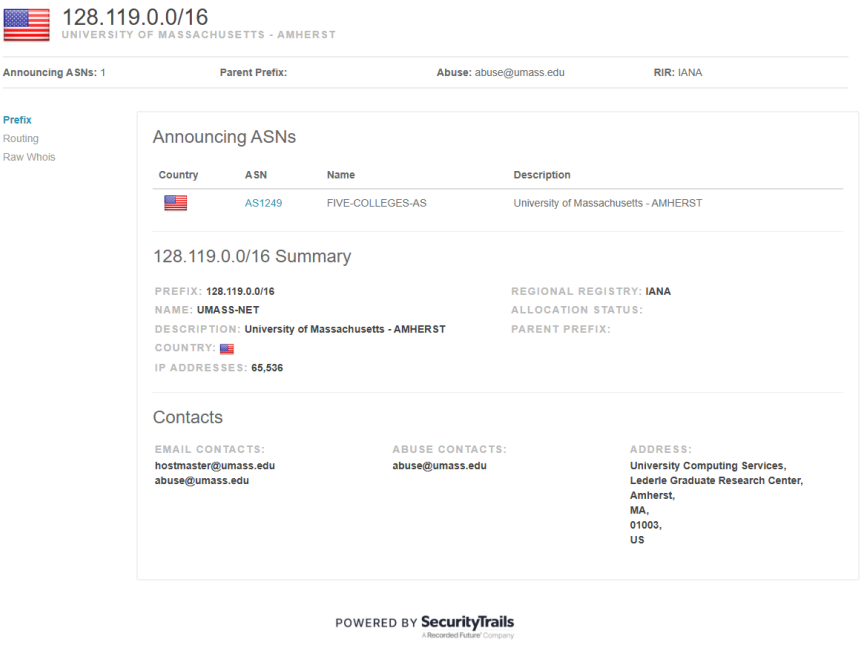


Figure 2.14: Raw WHOIS information for 128.119.0.0/16

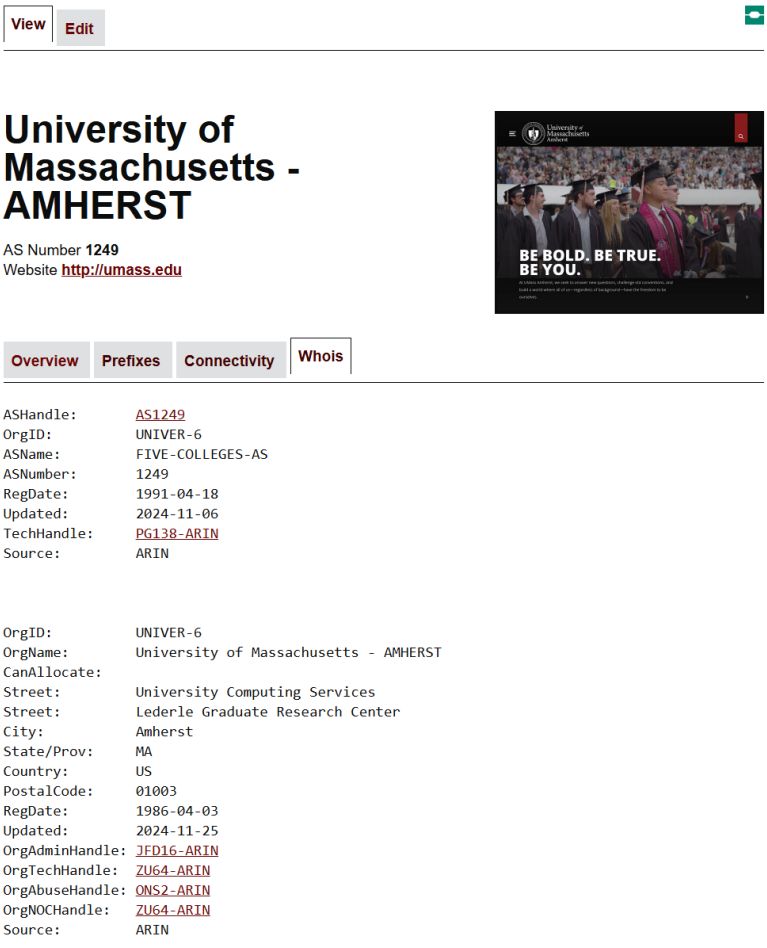


Figure 2.15: DNS response from BGP.tools for gaia.cs.umass.edu

```
OrgAdminHandle:       JFD16-ARIN
OrgAdminName:         John Doyle
# Email removed, Log into bgp.tools to see whois emails


OrgTechHandle:        ZU64-ARIN
OrgTechName:           Hostmaster
# Email removed, Log into bgp.tools to see whois emails


OrgAbuseHandle:       ONS2-ARIN
OrgAbuseName:          OIT Network Services
# Email removed, Log into bgp.tools to see whois emails


OrgNOCHandle:         ZU64-ARIN
OrgNOCName:            Hostmaster
# Email removed, Log into bgp.tools to see whois emails


TechHandle:           PG138-ARIN
TechName:             Peter Gutierrez
# Email removed, Log into bgp.tools to see whois emails
```

Last Update: 2025-05-08T03:26:52Z UTC

Member of the following AS-SETs

-	Member	ASN Count
RADB	as10578:as-umassnet	9
LEVEL3	as-level3-origins-na	0
RADB	as-comcast-edia	2780
RADB	as198949:as-sd	3951
LEVEL3	as-as209-cust	1818
RADB	as-prole	1215
RIPE	as-aunanet	313
RIPE	as-ono-ups	316

Figure 2.16: BGP registration data for UMass ASN

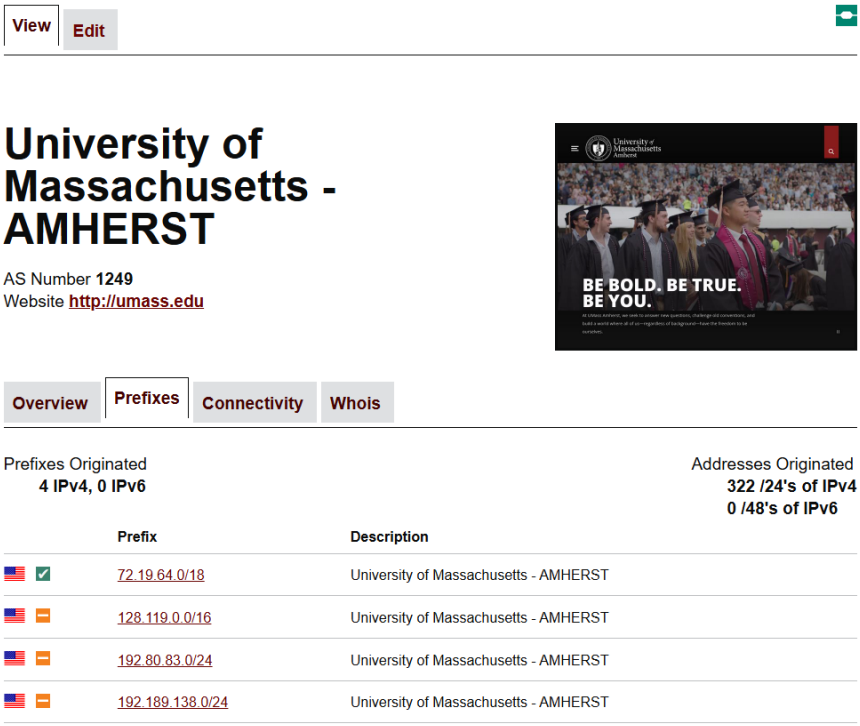


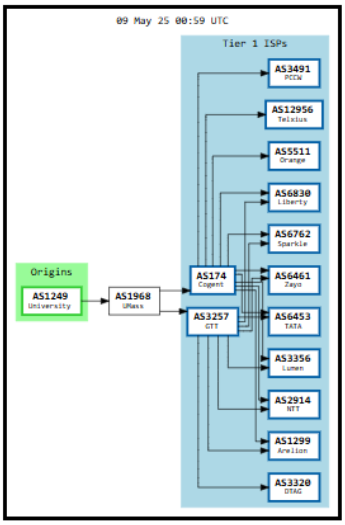
Figure 2.17: ASN prefixes originated from AS1249

Peers	Upstreams	Downstreams
2	1	1 (Cone: 2)

Network Policy

Click here to learn more about what this graph means and what makes up a network policy

Global Aggregation



▶ Chart Display Options

How are upstreams and downstreams calculated?

Upstreams

	ASN	Description	IPv4	IPv6
	AS1968	UMASSNET	✓	✓

Peers

	ASN	Description	IPv4	IPv6
	AS1968	UMASSNET	✓	✓
	AS168	University of Massachusetts - AMHERST	✗	✓

Downstreams

	ASN	Description	IPv4	IPv6
	AS168	University of Massachusetts - AMHERST	✗	✓

Figure 2.18: Contact and technical details for AS1249

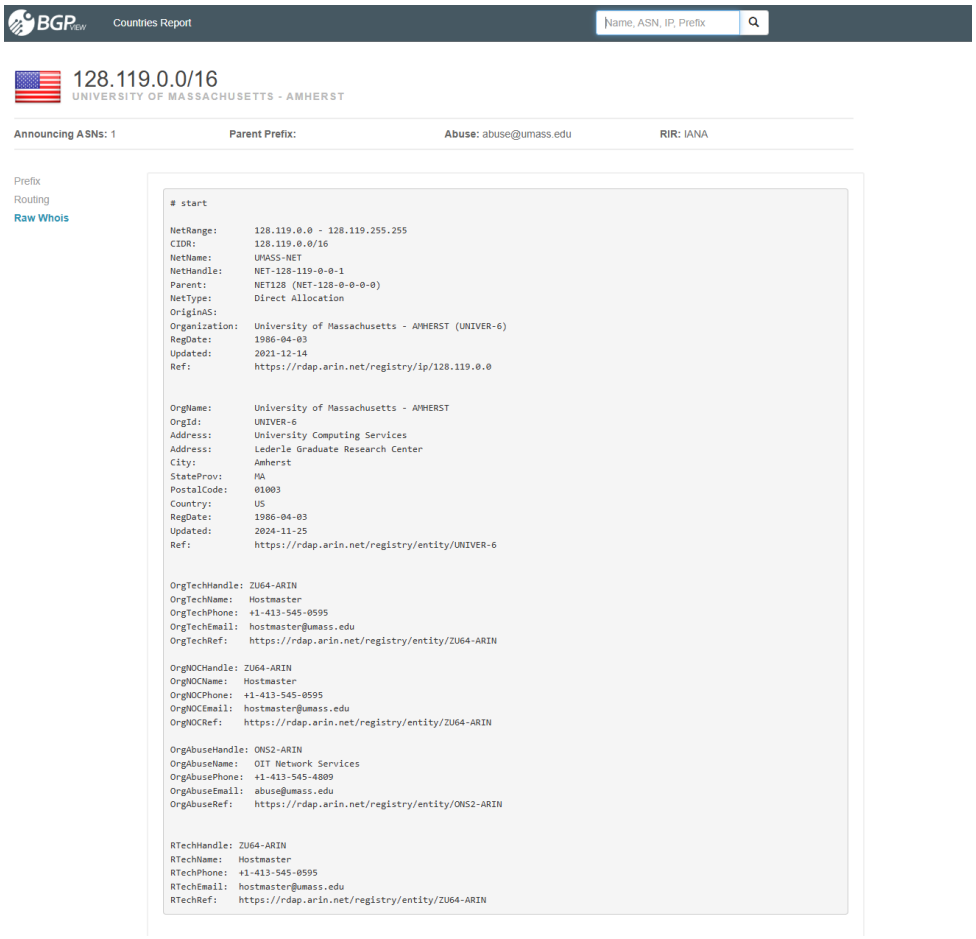


Figure 2.19: Detailed routing and registry metadata

Part C – DNS Query and Response (Wireshark)

Wireshark was used to capture DNS traffic between the local machine and the DNS server. Before starting the capture, the DNS cache was cleared to make sure that a new DNS query would be sent.

The website `gaia.cs.umass.edu` was typed into the browser. A DNS query was sent from the computer to the DNS server at `192.168.1.1`. The server responded with the IP address `128.119.245.12`.

This process can be seen in the following two figures. The first figure shows the DNS query that was sent. The second figure shows the DNS response with the resolved IP address.

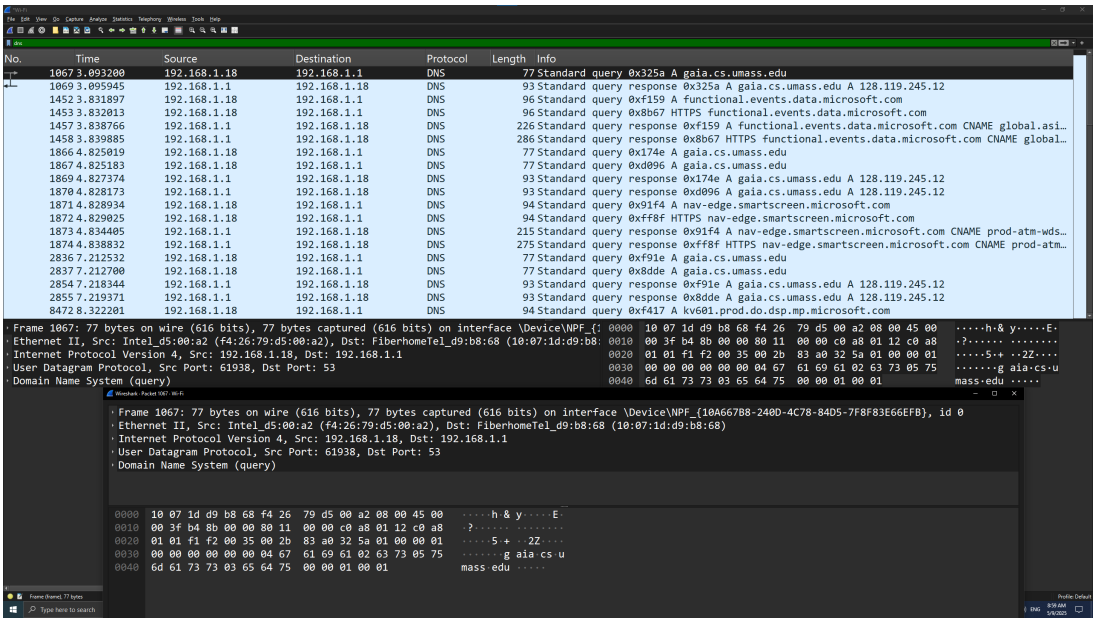


Figure 2.20: DNS query for `gaia.cs.umass.edu` sent to 192.168.1.1

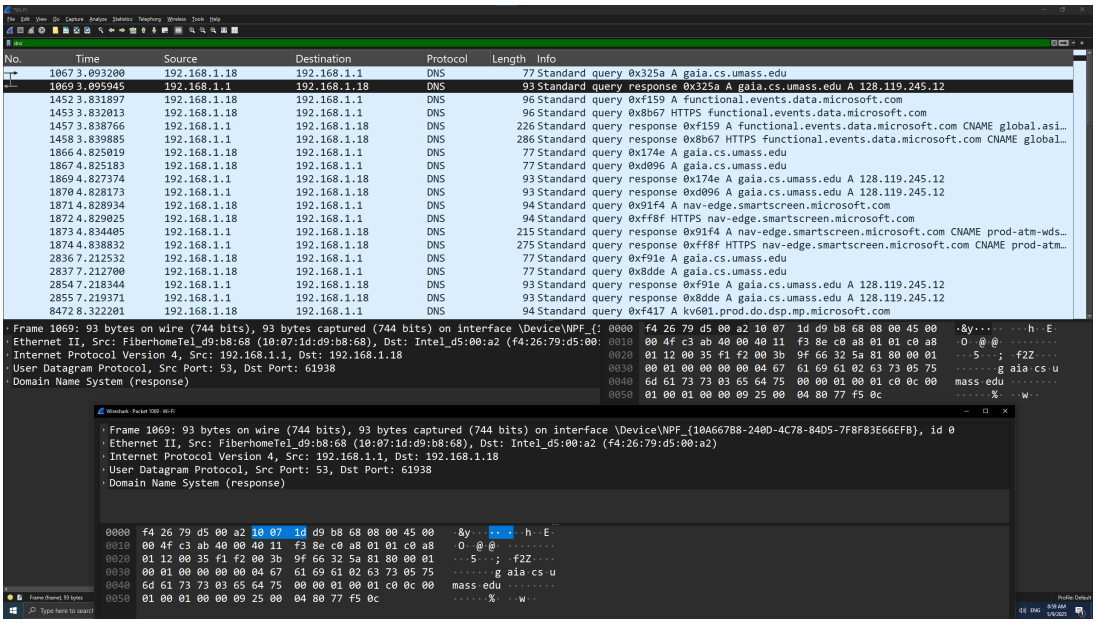


Figure 2.21: DNS response received with IP address 128.119.245.12

This capture confirmed that a DNS query and response were successfully exchanged, and the domain name was resolved properly by the local DNS server.

2.2 Task 2 – Implement a Simple Web Server Using Socket Programming

2.2.1 Task 2a – Main English Webpage (`main_en.html`)

A complete HTML and CSS webpage was created to serve as the main English page for the ENCS3320 web server. It included all the required components and was styled for clarity and visual appeal.

- The HTML page title was set as **"ENCS3320-Webserver"**. Although the browser tab title was written correctly in red style within the HTML, it could not be displayed in red due to browser theme limitations. Modern browsers render tab titles using system or browser-defined colours, which cannot be overridden with HTML or CSS.
- The header text **"Welcome to ENCS3320 – Computer Networks Webserver"** was displayed in blue at the top of the page using CSS.
- Team members were displayed in boxes with photos, names, student IDs, and a short list of projects, skills, and hobbies.
- A networking topic from the textbook's first chapter was included — comparing Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM). The section used headings, bullet points, and diagrams.
- An additional section showed the Internet Protocol Stack using an ordered list and an image.
- Three links were added at the bottom of the page:
  - Textbook site: `https://gaia.cs.umass.edu/kurose_ross/index.php`
  - Birzeit University: `https://www.birzeit.edu/`
  - Local page: `mySite_1232951_en.html`

Code Screenshots:

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <title>ENCS3320-Webserver</title>
    <link rel="stylesheet" href="Style/Style.css" />
</head>

<body>

    <nav class="navbar">
        <a class="lang-switch" href="main_ar.html">عربي</a>
        <h1 class="nav-title">Welcome to ENCS3320 - Computer Networks Webserver</h1>
        <div class="nav-spacer"></div>
    </nav>

    <main>

        <section class="team-section">
            <h2 class="section-heading">Team Members</h2>

            <div class="team-grid">
                <article class="team-card">
                    <div class="avatar-wrapper">
                        <img src="Images/Abdallah.png" alt="John Doe">
                    </div>

                    <ul class="member-list">
                        <li><strong>Name:</strong> Abdallah Abbasi</li>
                        <li><strong>ID:</strong> 1232951</li>
                        <li><strong>Projects:</strong> Shopping Cart System</li>
                        <li><strong>Skills:</strong> HTML, CSS, JavaScript, React, Node.js, MongoDB, Networking,
                            Cybersecurity, Operating Systems</li>
                        <li><strong>Hobbies:</strong> Programming, All things computer-related</li>
                    </ul>
                </article>

                <article class="team-card">
                    <div class="avatar-wrapper">
                        <img src="Images/JAMAL.png" alt="Jane Smith">
                    </div>

                    <ul class="member-list">
                        <li><strong>Name:</strong> Jamal Shehadeh</li>
                        <li><strong>ID:</strong> 1192316</li>
                        <li><strong>Projects:</strong> Smart Home System for People with Disabilities, Electrical
                            Projects</li>
                        <li><strong>Skills:</strong> Electrical installation, Home Assistant</li>

```

Figure 2.22: HTML code – Team members section

```
                    <li><strong>Hobbies:</strong> Reading, Piano</li>
                </ul>
            </article>
        </div>
    </section>

    <section class="topic-section">
        <h2 class="section-heading">
            Frequency Division Multiplexing (FDM)   vs   Time Division Multiplexing (TDM)
        </h2>

        <div class="topic-grid">
            <aside class="topic-card">
                <h3>Frequency Division Multiplexing (FDM)</h3>
                <ul>
                    <li>Optical / electromagnetic frequencies divided into narrow bands.</li>
                    <li>Each call allocated its own band and can transmit at the max rate of that band.</li>
                </ul>

                <h3>Time Division Multiplexing (TDM)</h3>
                <ul>
                    <li>Transmission time divided into slots.</li>
                    <li>Each call allocated periodic slot(s); it can use the full bandwidth only during its slot(s).
                    </li>
                </ul>
            </aside>

            <aside class="topic-card topic-image">
                <img src="Images/FMD-and-TDM-2.jpg" alt="FDM vs TDM diagram">
            </aside>
        </div>
    </section>


    <section class="topic-section">
        <h2 class="section-heading">Internet Protocol Stack</h2>

        <div class="topic-grid">
            <aside class="topic-card">
                <ol>
                    <li><strong>application:</strong> supporting network applications<br>IMAP, SMTP, HTTP</li>
                    <li><strong>transport:</strong> process-process data transfer<br>TCP, UDP</li>
                    <li><strong>network:</strong> routing of datagrams from source to destination<br>IP, routing
                        protocols</li>
                    <li><strong>link:</strong> data transfer between neighboring network elements<br>Ethernet,
                        802.11 (WiFi), PPP</li>
                    <li><strong>physical:</strong> bits “on the wire”</li>
                </ol>
            </aside>
```

Figure 2.23: HTML code – FDM vs TDM topic section

```
                <aside class="topic-card topic-image">
                    <img src="Images/layers.jpg" alt="Internet protocol stack diagram">
                </aside>
            </div>
        </section>
    </main>

    <footer class="page-footer">
        <p>
            External resources:
            <a href="https://gaia.cs.umass.edu/kurose_ross/index.php" target="_blank">Textbook site</a> |
            <a href="https://www.birzeit.edu/" target="_blank">Birzeit University</a> |
            <a href="mySite_1232951_en.html">Local page (mySite_1232951_en.html)</a>
        </p>
    </footer>

</body>

</html>
```

Figure 2.24: HTML code – Protocol stack section and external links

Rendered Page:

Figure 2.25: Final rendered output of `main_en.html` in browser

2.2.2 Task 2b – Local Webpage (`mySite_1232951_en.html`)

A secondary local webpage was created to allow users to request files related to the topic presented in the main page. The user was asked to enter the exact name of an image or video file into an input field.

The form used the GET method and submitted the input as a URL query. When the form was submitted, the file was searched for in the server's local directory.

If the requested file was not found, the server responded with a **307 Temporary Redirect**. The redirection was based on the file extension:

- If the file ended with `.png`, `.jpg`, or other image formats, the client was redirected to a Google Image search using the provided file name as a keyword.
- If the file ended with `.mp4` or other video formats, the client was redirected to a Google Video search.

The user was also shown a message indicating that if the file was not available locally, a redirection would occur.

Form Page HTML Code:

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <title>Media Finder – ENCS3320</title>
    <link rel="stylesheet" href="Style/Style.css">
    <link rel="stylesheet" href="Style/mySite.css">
</head>

<body>


    <nav class="navbar">
        <a class="lang-switch" href="mySite_1232951_ar.html">عربي</a>
        <h1 class="nav-title">Request an Image / Video</h1>
        <div class="nav-spacer"></div>
    </nav>

    <main>


        <section class="team-section" style="padding:2rem 0">
            <div class="team-grid" style="grid-template-columns:minmax(280px,400px)">
                <article class="team-card" style="padding:2rem 1rem">
                    <p style="margin-bottom:1rem">
                        Type the <strong>exact file name</strong> you want
                        (e.g. <em>osi_layers.png</em> or <em>intro.mp4</em>), then hit <b>Fetch</b>.
                        If the file isn't found locally, the server will redirect you to
                        a Google Images / Videos search for it.
                    </p>

                    <form action="/getImage" method="get" class="member-list" style="font-size:1rem">
                        <label for="imageName">File name:</label><br>
                        <input id="imageName" name="imageName" type="text" required
                            style="width:100%;padding:.5rem;border:1px solid #ccc;border-radius:4px;margin:.5rem 0 1rem">
                        <button type="submit"
                            style="padding:.5rem 1.25rem;border:none;border-radius:4px;background:var(--accent);color:#fff;cursor:pointer">
                            Fetch
                        </button>
                    </form>
                </article>
            </div>
        </section>

    </main>


    <footer class="page-footer">
        <p>
            <a href="main_en.html">Back to Home</a>
        </p>
    </footer>

</body>

</html>
```

Figure 2.26: HTML code of `mySite_1232951_en.html` – file request form

Rendered Page:

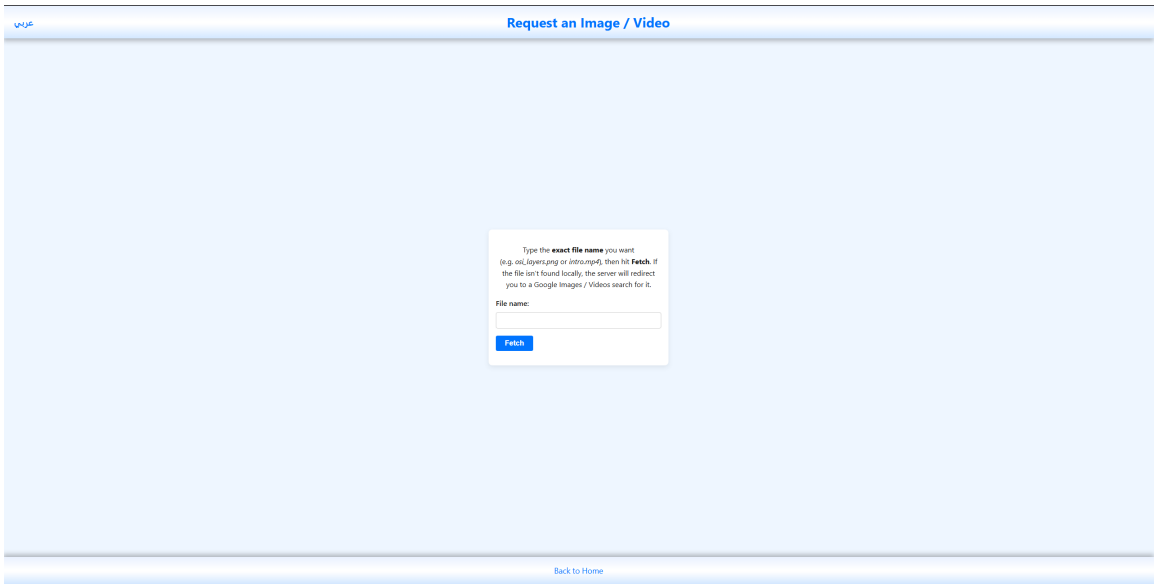


Figure 2.27: Rendered view of local file request page

The form was fully functional, and the redirection was correctly handled by the server logic. This helped simulate real-world search fallback behaviour for missing media content.

2.2.3 Task 2c – Arabic Versions (`main_ar.html` and `mySite_1232951_ar.html`)

Arabic versions of the main and local HTML pages were created. These pages were named `main_ar.html` and `mySite_1232951_ar.html`.

Both pages used Arabic text and followed a right-to-left (`rtl`) layout. The visual design, content, and structure were kept similar to the English versions for consistency.

- The direction was set using `<html lang="ar" dir="rtl">`.
- All text content was translated into Arabic, including headings, labels, button text, and descriptions.
- External links to the textbook site and Birzeit University were included.
- A language switch link allowed users to go back to the English version.

main_ar.html:

- Contained the same sections as the English version (team members, FDM vs TDM, Internet protocol stack).
- CSS styling and layout were preserved using mirrored flex/grid alignment.

mySite_1232951_ar.html:

- Allowed Arabic-speaking users to input a file name and request it.
- The same 307 redirect behaviour was implemented if the file was not found locally.

Code Screenshots:

```html
<!DOCTYPE html>
<html lang="ar" dir="rtl">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1.0" />
  <title>ENCS3320-خادم الويب</title>

  <link rel="stylesheet" href="Style/Style.css" />
  <link rel="stylesheet" href="Style/rtl.css" />
</head>

<body>


  <nav class="navbar">
    <a class="lang-switch" href="main_en.html">English</a>
    <h1 class="nav-title">مرحباً بكم في ENCS3320 – خادم شبكات الحاسوب</h1>
    <div class="nav-spacer"></div>
  </nav>

  <main>


    <section class="team-section">
      <h2 class="section-heading">أعضاء الفريق</h2>

      <div class="team-grid">
        <article class="team-card">
          <div class="avatar-wrapper">
            <img src="Images/Abdallah.png" alt="عبد الله">
          </div>
          <ul class="member-list">
            <li><strong>الاسم:</strong> عبد الله</li>
            <li><strong>الرقم:</strong> 1232951</li>
            <li><strong>المشاريع:</strong> نظام سلة تسوق</li>
            <li><strong>المهارات:</strong> HTML، CSS، JavaScript، React، Node.js، MongoDB، الشبكات، الأمن السيبراني،
              أنظمة التشغيل</li>
            <li><strong>الهوايات:</strong> البرمجة، كل ما يتعلق بالحاسوب</li>
          </ul>
        </article>

        <article class="team-card">
          <div class="avatar-wrapper">
            <img src="Images/JAMAL.png" alt="جمال">
          </div>
          <ul class="member-list">
            <li><strong>الاسم:</strong> جمال</li>
            <li><strong>الرقم:</strong> 1192316</li>
            <li><strong>المشاريع:</strong>مشروع بيت ذكي بلغة الهوم أسيستنت لذوي
```

Figure 2.28: HTML code – Arabic main page (team member section)

```
            <li><strong>المهارات:</strong> التعديات الكهربائية، Home Assistant</li>
            <li><strong>الهوايات:</strong> القراءة، العزف على البيانو</li>
          </ul>
        </article>
      </div>
    </section>


    <section class="topic-section">
      <h2 class="section-heading">
        تقسيم التردد (FDM)  <span style="font-weight:400">مقابل</span>  تقسيم الزمن (TDM)
      </h2>

      <div class="topic-grid">
        <aside class="topic-card">
          <h3>تقسيم التردد (FDM)</h3>
          <ul>
            <li>تُقسَّم الترددات الكهرومغناطيسية إلى نطاقات ضيّقة.</li>
            <li>يُخصَّص لكل اتصال نطاق مستقل ويمكنه الإرسال بأقصى معدل لذلك النطاق.</li>
          </ul>

          <h3>تقسيم الزمن (TDM)</h3>
          <ul>
            <li>يُقسَّم الزمن إلى خانات زمنية.</li>
            <li>يُخصَّص لكل اتصال خانة دورية ويمكنه استخدام كامل النطاق فقط خلالها.</li>
          </ul>
        </aside>

        <aside class="topic-card topic-image">
          <img src="Images/FMD-and-TDM-2.jpg" alt="مخطط FDM و TDM">
        </aside>
      </div>
    </section>


    <section class="topic-section">
      <h2 class="section-heading">طبقات بروتوكول الإنترنت</h2>

      <div class="topic-grid">
        <aside class="topic-card">
          <ol>
            <li><strong>Application:</strong> تطبيقات الشبكة<br>IMAP، SMTP، HTTP</li>
            <li><strong>Transport:</strong> نقل بيانات بين العمليات<br>TCP، UDP</li>
            <li><strong>Network:</strong> توجيه الحزم بين المصدر والوجهة<br>IP، بروتوكولات التوجيه</li>
            <li><strong>Link:</strong> نقل البيانات بين عناصر الشبكة المتجاورة<br>Ethernet، Wi-Fi، PPP</li>
            <li><strong>Physical:</strong> نبضات على السلك</li>
          </ol>
        </aside>

        <aside class="topic-card topic-image">

```

Figure 2.29: HTML code – Arabic main page (topic and image sections)

```
          <img src="Images/layers.jpg" alt="مخطط طبقات البروتوكول">
        </aside>
      </div>
    </section>

  </main>


  <footer class="page-footer">
    <p>
      مصادر خارجية:
      <a href="https://gaia.cs.umass.edu/kurose_ross/index.php" target="_blank">موقع الكتاب</a> |
      <a href="https://www.birzeit.edu/" target="_blank">جامعة بيرزيت</a> |
      <a href="mySite_1232951_ar.html">صفحة محلية (mySite_1232951_ar.html)</a>
    </p>
  </footer>

</body>

</html>
```

Figure 2.30: HTML code – Arabic main page footer

```html
<!DOCTYPE html>
<html lang="ar" dir="rtl">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1.0" />
    <title>ملتقط الوسائط – ENCS3320</title>


    <link rel="stylesheet" href="Style/Style.css">
    <link rel="stylesheet" href="Style/mySite.css">
</head>

<body>


    <nav class="navbar" style="direction:rtl">
        <a class="lang-switch" href="mySite_1232951_en.html">English</a>
        <h1 class="nav-title">طلب صورة / فيديو</h1>
        <div class="nav-spacer"></div>
    </nav>

    <main>


        <section class="team-section" style="padding:2rem 0">
            <div class="team-grid" style="grid-template-columns:minmax(280px,400px);direction:ltr">
                <article class="team-card" style="padding:2rem 1rem;direction:rtl">
                    <p style="margin-bottom:1rem">
                        اكتب <strong>اسم الملف بالضبط</strong>
                        (مثلاً <em>osi_layers.png</em> أو <em>intro.mp4</em>) ثم اضغط <b>بحث</b>.
                        إذا لم يُعثَر على الملف محلياً، سيعيد الخادم توجيهك إلى
                        بحث صور أو فيديو في Google.
                    </p>

                    <form action="/getImage" method="get" class="member-list" style="font-size:1rem">
                        <label for="imageName">اسم الملف:</label><br>
                        <input id="imageName" name="imageName" type="text" required
                            style="direction:ltr;width:100%;padding:.5rem;border:1px solid #ccc;border-radius:4px;margin:.5rem 0 1rem">
                        <button type="submit"
                            style="padding:.5rem 1.25rem;border:none;border-radius:4px;background:var(--accent);color:#fff;cursor:pointer">
                            جلب
                        </button>
                </article>
            </div>
        </section>

    </main>

    <footer class="page-footer" style="text-align:center">
        <p>
            <a href="main_ar.html">العودة إلى الصفحة الرئيسية</a>
        </p>
    </footer>

</body>

</html>
```

Figure 2.31: HTML code – Arabic local request page (`mySite_1232951_ar.html`)

Rendered Pages:

Figure 2.32: Rendered view of Arabic main page (`main_ar.html`)

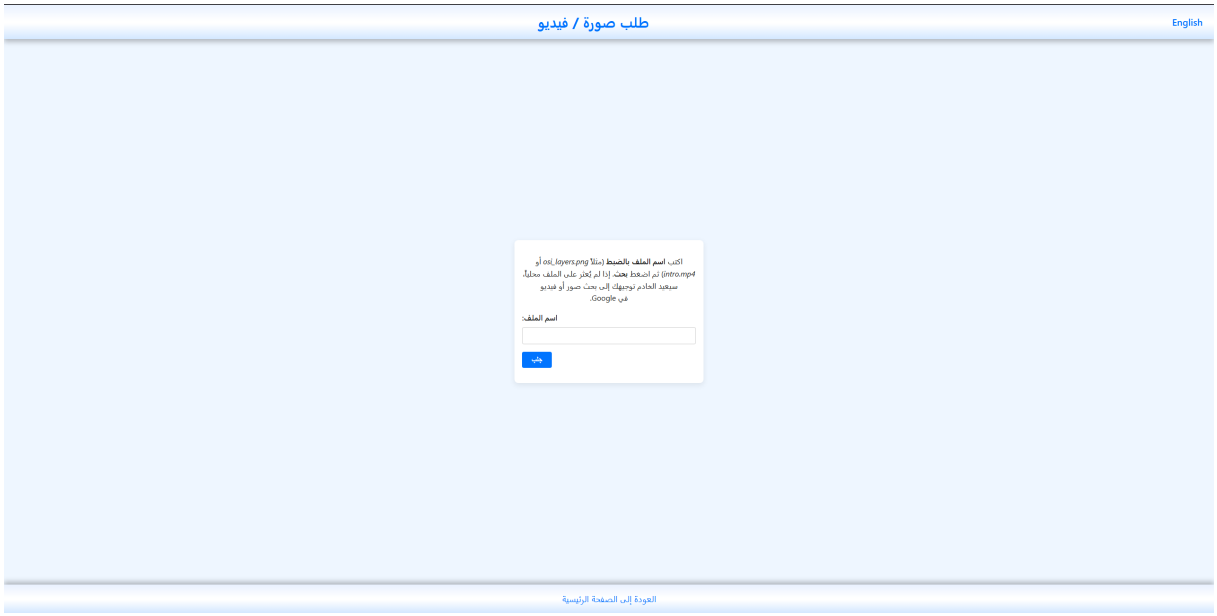


Figure 2.33: Rendered view of Arabic local page (`mySite_1232951_ar.html`)

CSS Styling

To improve the appearance and readability of both English and Arabic pages, external CSS files were created. These files defined consistent fonts, colours, spacing, shadows, and layout rules.

Two stylesheets were used:

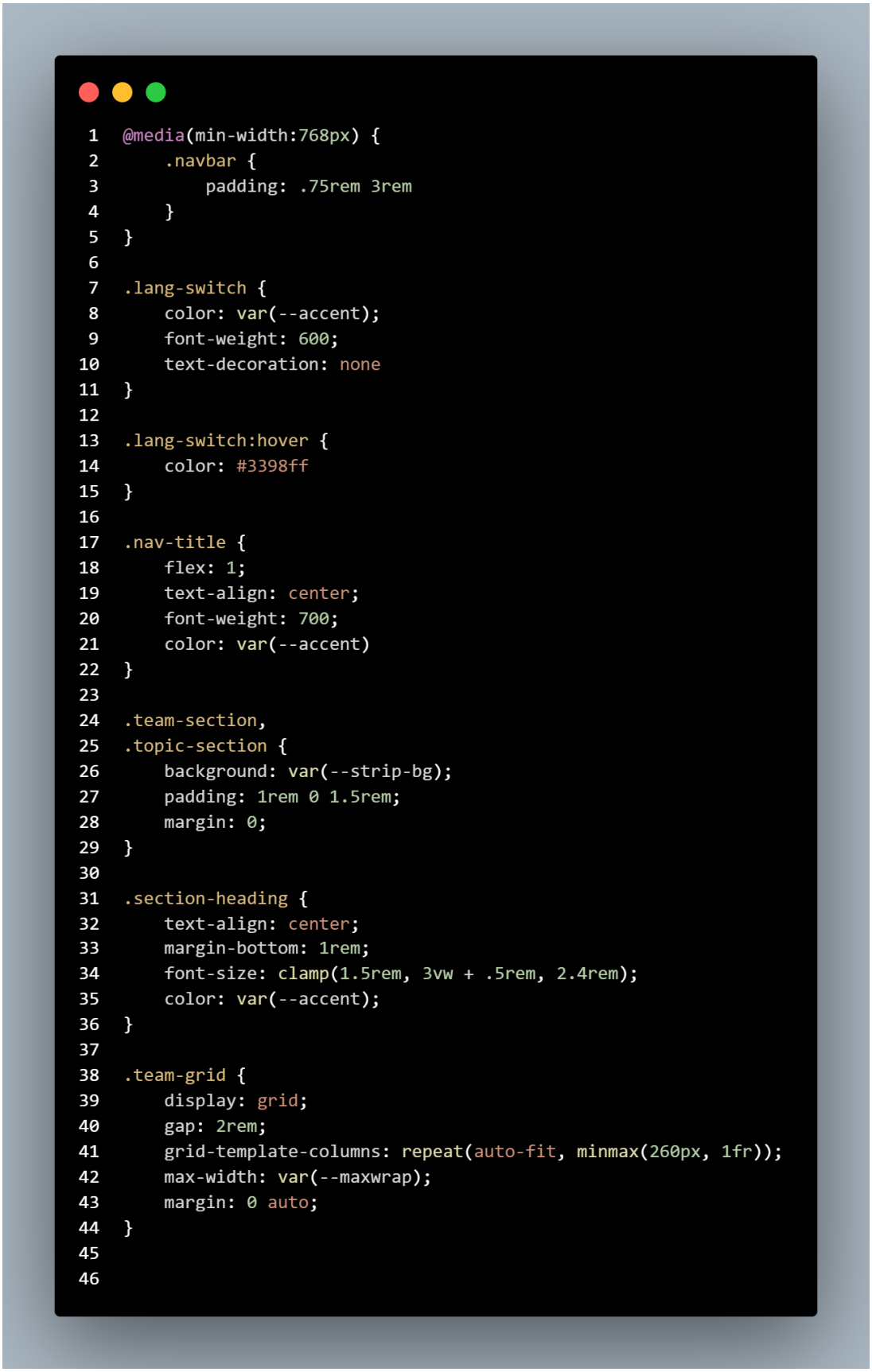
- `Style.css` – For general layout and design.
- `rtl.css` – For Arabic RTL (right-to-left) adjustments.

The CSS styling included:

- Custom variables using `:root` to control colours, shadows, max width, and spacing.
- Flexbox and CSS Grid used for layout control.
- Responsive design using `@media` queries.
- Separate styles applied to navbar, footer, team cards, topic sections, and image containers.
- RTL-specific CSS ensured Arabic text alignment and directionality were properly displayed.

```css
:root {
    --accent: #0074ff;
    --strip-bg: #eef6ff;
    --card-bg: #ffffff;
    --text: #1a1a1a;
    --nav-grad: linear-gradient(180deg, #eaf2ff 0%, #ffffff 45%, #ffffff 55%, #d4e8ff 100%);
    --shadow: rgba(0, 0, 0, .08);
    --maxwrap: 1200px;
}

*,
*::before,
*::after {
    box-sizing: border-box;
    margin: 0;
    padding: 0
}

html,
body {
    height: 100%
}

body {
    font-family: system-ui, -apple-system, "Segoe UI", sans-serif;
    background: var(--strip-bg);
    color: var(--text);
    line-height: 1.6;
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}

main {
    flex: 1 0 auto;
    padding: 0
}

.navbar {
    display: flex;
    align-items: center;
    gap: 1rem;
    padding: .75rem 1.5rem;
    background: var(--nav-grad);
    box-shadow: 0 6px 10px rgba(0, 0, 0, .25);
    font-size: calc(1rem + 4px);
}

```

Figure 2.34: CSS base variables and body styling (`Style.css`)



```
@media(min-width:768px) {
    .navbar {
        padding: .75rem 3rem
    }
}

.lang-switch {
    color: var(--accent);
    font-weight: 600;
    text-decoration: none
}

.lang-switch:hover {
    color: #3398ff
}

.nav-title {
    flex: 1;
    text-align: center;
    font-weight: 700;
    color: var(--accent)
}

.team-section,
.topic-section {
    background: var(--strip-bg);
    padding: 1rem 0 1.5rem;
    margin: 0;
}

.section-heading {
    text-align: center;
    margin-bottom: 1rem;
    font-size: clamp(1.5rem, 3vw + .5rem, 2.4rem);
    color: var(--accent);
}

.team-grid {
    display: grid;
    gap: 2rem;
    grid-template-columns: repeat(auto-fit, minmax(260px, 1fr));
    max-width: var(--maxwrap);
    margin: 0 auto;
}
```

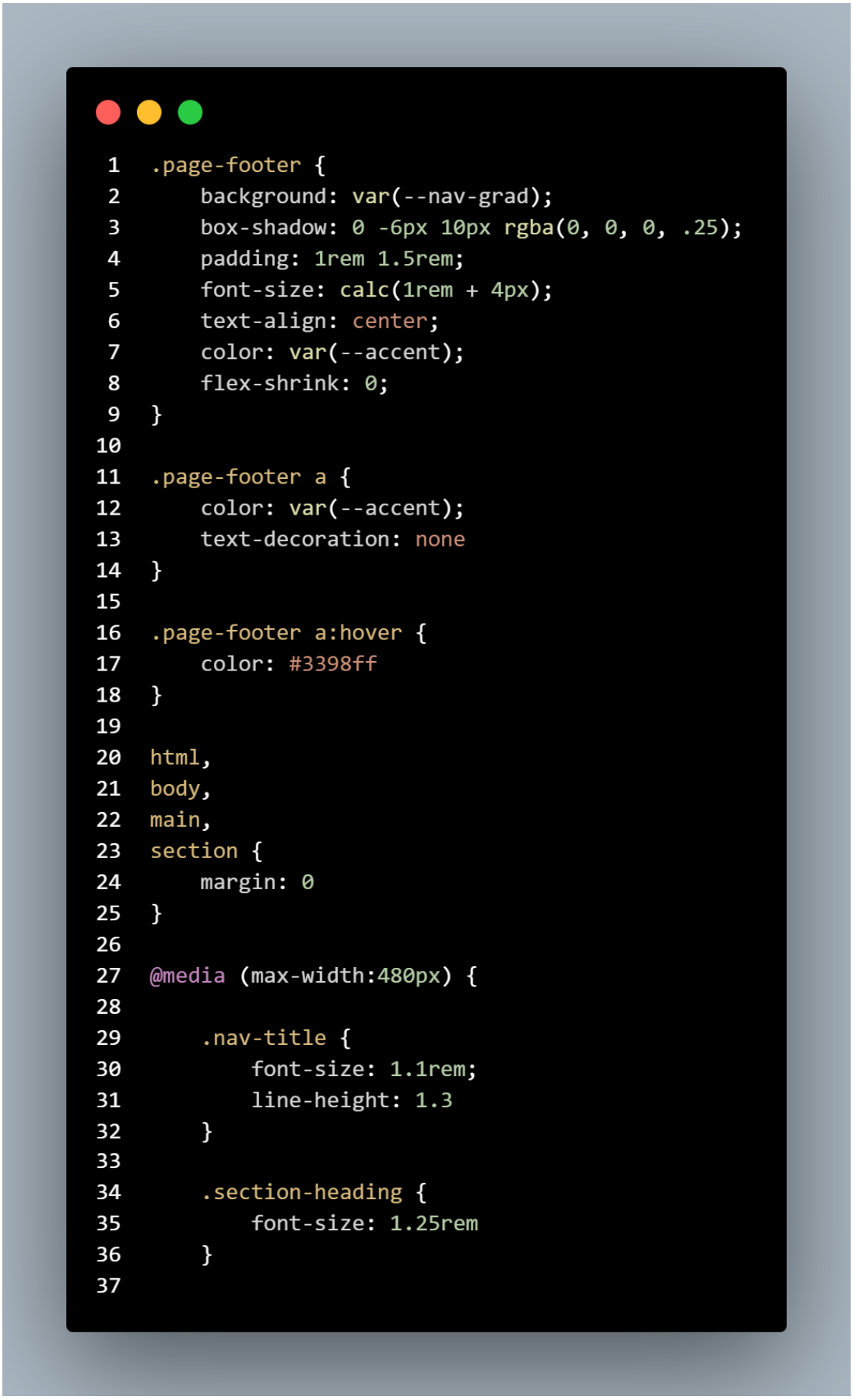
Figure 2.35: Navigation bar, team sections, and headings

```css
.team-card {
    position: relative;
    text-align: center;
    background: var(--card-bg);
    padding: 3.5rem 1rem 2rem;
    border-radius: 8px;
    box-shadow: 0 4px 12px var(--shadow);

}

.avatar-wrapper {
    position: absolute;
    top: -45px;
    left: 50%;
    transform: translateX(-50%);
    width: 90px;
    height: 90px;
    border-radius: 50%;
    overflow: hidden;
    border: 4px solid var(--card-bg);
    box-shadow: 0 0 0 2px var(--accent);
}

.avatar-wrapper img {
    width: 100%;
    height: 100%;
    object-fit: cover
}

.member-list {
    list-style: none;
    font-size: 1.25rem;
    text-align: left;
    margin-top: .5rem
}

.topic-grid {
    display: grid;
    gap: 2rem;
    grid-template-columns: 4fr 3fr;
    align-items: stretch;
    max-width: var(--maxwrap);
    margin: 0 auto;
}

@media(max-width:700px) {
    .topic-grid {
        grid-template-columns: 1fr
    }
}

.topic-card {
    background: var(--card-bg);
    padding: 1.5rem;
    border-radius: 8px;
    box-shadow: 0 4px 12px var(--shadow);
    position: relative;
}

```


Figure 2.36: Team card design, avatar styling, and responsive layout

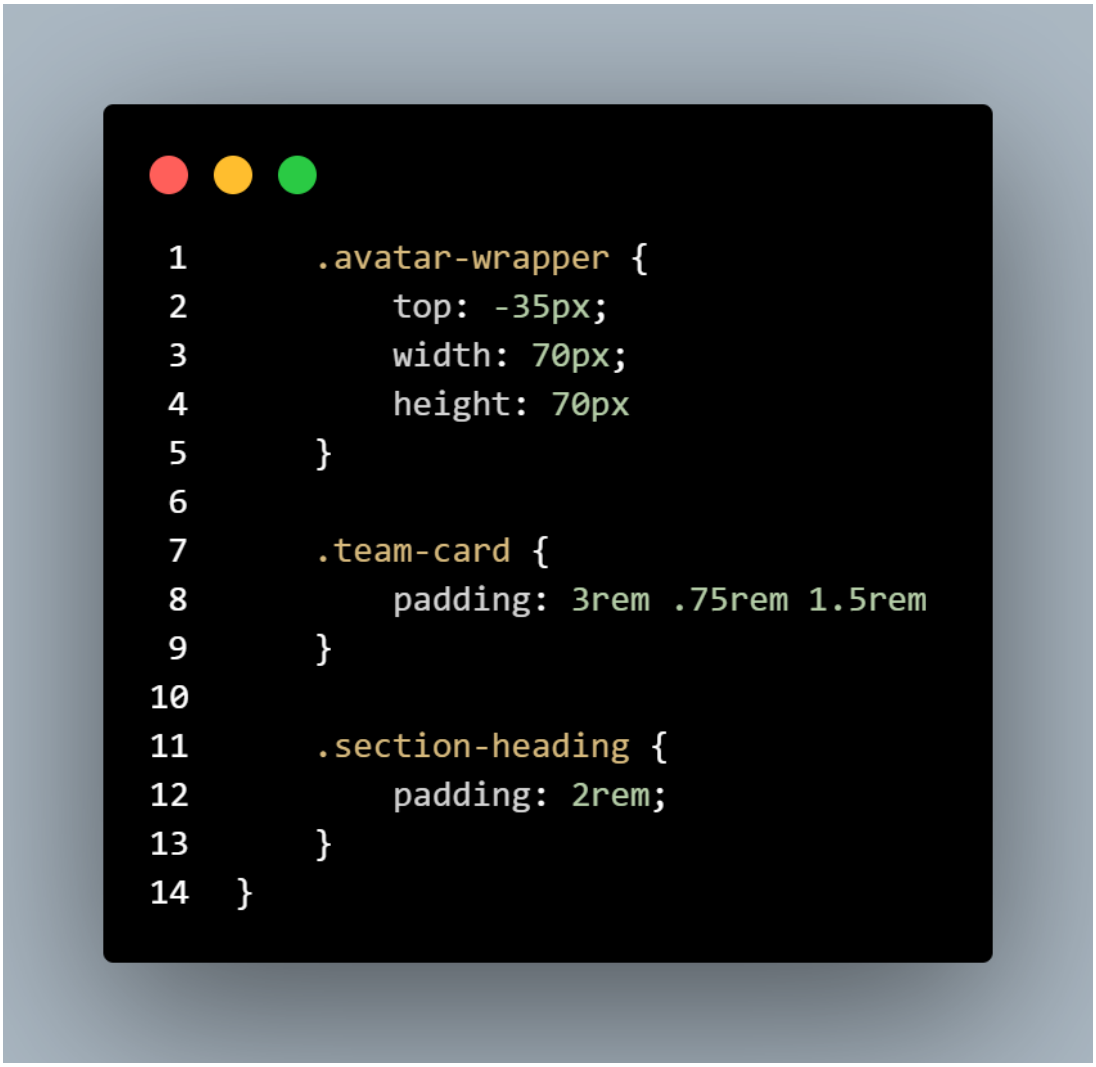
```css
.topic-card h3 {
    color: var(--accent);
    margin: .5rem 0
}

.topic-card ul,
.topic-card ol {
    margin-left: 1rem;
    margin-bottom: .5rem
}

.topic-card li {
    margin: .25rem 0
}

.topic-card:not(.topic-image) {
    font-size: calc(1rem + 4px)
}

.topic-image img {
    width: 100%;
    display: block;
    border-radius: 4px
}

.topic-image::before,
.topic-image::after {
    content: '';
    position: absolute;
    left: 0;
    width: 100%;
    height: 14px;
    background: var(--strip-bg);
    pointer-events: none;
}

.topic-image::before {
    top: 0
}

.topic-image::after {
    bottom: 0
}

```

Figure 2.37: Topic section layout and styling



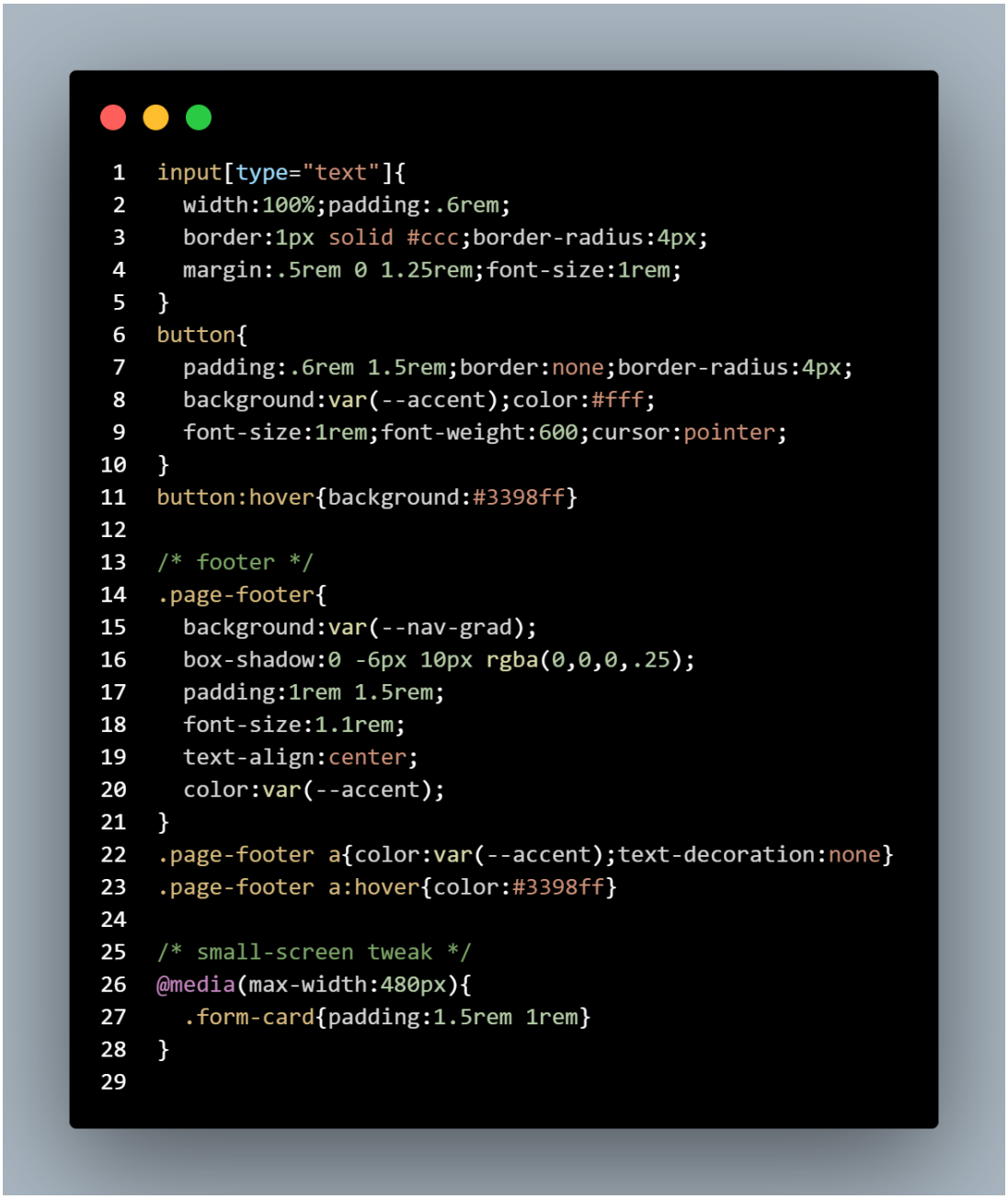
```
.page-footer {
    background: var(--nav-grad);
    box-shadow: 0 -6px 10px rgba(0, 0, 0, .25);
    padding: 1rem 1.5rem;
    font-size: calc(1rem + 4px);
    text-align: center;
    color: var(--accent);
    flex-shrink: 0;
}

.page-footer a {
    color: var(--accent);
    text-decoration: none
}

.page-footer a:hover {
    color: #3398ff
}

html,
body,
main,
section {
    margin: 0
}

@media (max-width:480px) {

    .nav-title {
        font-size: 1.1rem;
        line-height: 1.3
    }

    .section-heading {
        font-size: 1.25rem
    }

```

Figure 2.38: Topic content spacing, image layout, and hover styles



```
1       .avatar-wrapper {
2           top: -35px;
3           width: 70px;
4           height: 70px
5       }
6
7       .team-card {
8           padding: 3rem .75rem 1.5rem
9       }
10
11      .section-heading {
12          padding: 2rem;
13      }
14  }
```

Figure 2.39: Footer styles and small screen adjustments



```
input[type="text"]{
  width:100%;padding:.6rem;
  border:1px solid #ccc;border-radius:4px;
  margin:.5rem 0 1.25rem;font-size:1rem;
}
button{
  padding:.6rem 1.5rem;border:none;border-radius:4px;
  background:var(--accent);color:#fff;
  font-size:1rem;font-weight:600;cursor:pointer;
}
button:hover{background:#3398ff}

/* footer */
.page-footer{
  background:var(--nav-grad);
  box-shadow:0 -6px 10px rgba(0,0,0,.25);
  padding:1rem 1.5rem;
  font-size:1.1rem;
  text-align:center;
  color:var(--accent);
}
.page-footer a{color:var(--accent);text-decoration:none}
.page-footer a:hover{color:#3398ff}

/* small-screen tweak */
@media(max-width:480px){
  .form-card{padding:1.5rem 1rem}
}

```

Figure 2.40: General layout and team member section CSS

```
:root{
  --accent:#0074ff;
  --strip-bg:#eef6ff;
  --card-bg:#ffffff;
  --text:#1a1a1a;
  --shadow:rgba(0,0,0,.08);
  --nav-grad:linear-gradient(180deg,#eaf2ff 0%,#ffffff 45%,#ffffff 55%,#d4e8ff 100%);
}

/* reset + base */
*,*::before,*::after{box-sizing:border-box;margin:0;padding:0}
body{
  font-family:system-ui,-apple-system,"Segoe UI",sans-serif;
  background:var(--strip-bg);
  color:var(--text);
  line-height:1.6;
  display:flex;flex-direction:column;min-height:100vh;
}

/* navbar */
.navbar{
  display:flex;align-items:center;gap:1rem;
  padding:.75rem 1.5rem;
  background:var(--nav-grad);
  box-shadow:0 6px 10px rgba(0,0,0,.25);
  font-size:1.25rem;
}
.nav-title{flex:1;text-align:center;font-weight:700;color:var(--accent)}
.lang-switch{color:var(--accent);font-weight:600;text-decoration:none}
.lang-switch:hover{color:#3398ff}

/* main wrapper */
main{
  flex:1 0 auto;
  display:flex;
  align-items:center;
  justify-content:center;
  padding:2rem 1rem;
}

/* form card */
.form-card{
  background:var(--card-bg);
  max-width:420px;width:100%;
  padding:2rem 1.5rem;
  border-radius:8px;
  box-shadow:0 4px 12px var(--shadow);

```

Figure 2.41: Form design and padding for mobile screens

```css
html[dir="rtl"] body {
    direction: rtl;
    text-align: right
}


html[dir="rtl"] .navbar {
    direction: rtl
}

html[dir="rtl"] .lang-switch {
    direction: rtl
}

html[dir="rtl"] .nav-title {
    text-align: center
}

html[dir="rtl"] .team-grid {
    direction: rtl
}

html[dir="rtl"] .member-list {
    direction: rtl;
    padding-right: 1rem;
}


html[dir="rtl"] ul,
html[dir="rtl"] ol {
    direction: rtl;
    list-style-position: inside;
    padding-right: 1.2rem;
    margin-right: 0;
}

html[dir="rtl"] li {
    text-align: right
}


html[dir="rtl"] .topic-grid {
    direction: ltr
}


html[dir="rtl"] .topic-card {
    direction: rtl
}

html[dir="rtl"] .topic-card:not(.topic-image) {
    text-align: right
}

html[dir="rtl"] .page-footer {
    text-align: center
}
```

Figure 2.42: Right-to-left (RTL) layout rules for Arabic pages (`rtl.css`)

2.3 Task 2 (d) – Server Functionality

The server was implemented in Python using the `socket` module, listening on port 9991 (derived from student ID). Upon receiving an HTTP request, it performs the following steps:

- Parses and logs the full HTTP request (method, path, headers) on the terminal.
- Identifies the file extension to set the appropriate `Content-Type` (e.g., `text/html`, `image/png`, `text/css`, `video/mp4`).
- Returns a valid HTTP response using status codes such as `200 OK`, `307 Temporary Redirect`, or `404 Not Found`.
- Logs the response code and both the client's and server's socket addresses.

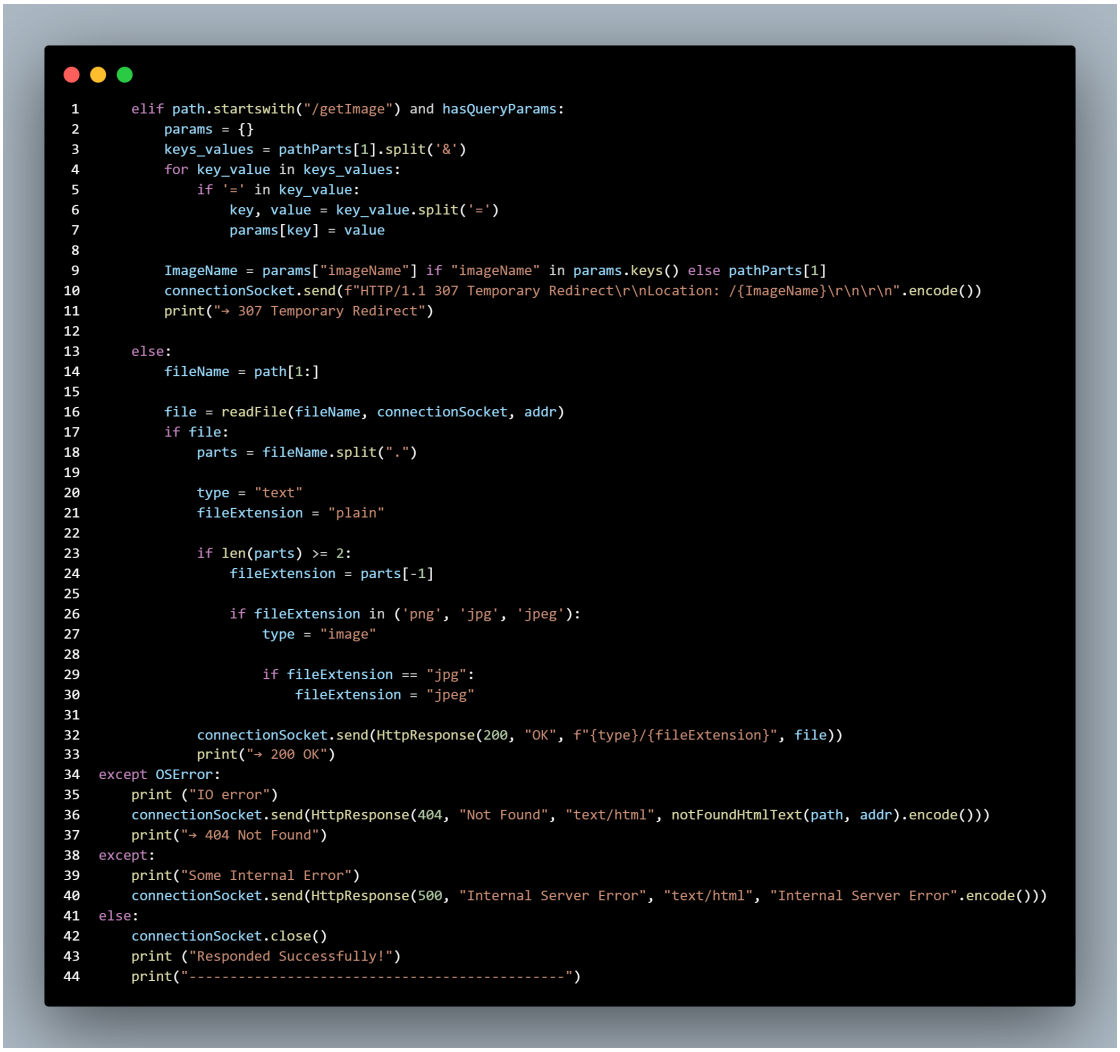
Figure 2.43 shows the part of the code responsible for setting up the server and handling 404 responses. **Figure 2.44** illustrates the request parsing, default routing, and 307 redirection to external sites. **Figure 2.45** displays image/video redirect handling and MIME type assignment.

```python

from socket import gethostbyname, socket, AF_INET, SOCK_STREAM, gethostname

mainAddress = (gethostbyname(gethostname()), 9991)

serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(mainAddress)
serverSocket.listen()

print("The server is up!")
print(f"Listening to http://{mainAddress[0]}:{mainAddress[1]}")

def notFoundHtmlText(file, addr):
    return f"""
            <html>
                <head><title>Error 404</title></head>
                <body>
                    <h1 style='color: red;'>The file "{file}" is not found</h1>
                    <p><b>1192316 - Jamal Shehadeh<br />1232951 - Abdallah Ababsi</p>
                    <p>Client IP: {addr[0]}</p>
                    <p>Client Port: {addr[1]}</p>
                </body>
            </html>
        """

def HttpResponse(code, status, type, body):
    return (f"HTTP/1.1 {code} {status}\r\n"
        f"Content-Type: {type}; charset=utf-8\r\n"
        f"Content-Length: {len(body)}\r\n\r\n").encode() + body

def readFile(filePath, connectionSocket, addr):
    try:
        file = open(f"static/{filePath}", "rb")
        return file.read()
    except OSError:
        ext = filePath.lower().split('.')[-1]
        if ext in ('png', 'jpg', 'jpeg', 'gif'):
            url = f"https://www.google.com/search?q={filePath}&tbm=isch"
            connectionSocket.send(
                f"HTTP/1.1 307 Temporary Redirect\r\nLocation: {url}\r\n\r\n".encode()
            )
            print("→ 307 Temporary Redirect")
        elif ext in ('mp4', 'webm'):
            url = f"https://www.google.com/search?q={filePath}&tbm=vid"
            connectionSocket.send(
                f"HTTP/1.1 307 Temporary Redirect\r\nLocation: {url}\r\n\r\n".encode()
            )
            print("→ 307 Temporary Redirect")
        else:

```

Figure 2.43: Server setup and 404 response generation

```
                HttpResponse(404, "Not Found", "text/html",
                notFoundHtmlText(filePath, addr).encode())
            )
            print("→ 404 Not Found")
        return

while True:
    try:
        connectionSocket, addr = serverSocket.accept()
        request = connectionSocket.recv(2048).decode()

        if not request or len(request) == 0:
            print("Heartbeating....")
            continue

        print("----------------------------------------------")
        print("Request From " + str(addr))
        print(request)

        lines = request.split("\r\n")

        requestLine = lines[0].strip()
        method, fullPath, version = requestLine.split(" ")

        pathParts = fullPath.split("?")
        hasQueryParams = len(pathParts) != 1
        path = pathParts[0] if hasQueryParams else fullPath

        path = path[:-1] if path.endswith("/") and len(path) != 1 else path

        if path == '/' or path.startswith(('/index.html', '/en')):
            htmlText = readFile("main_en.html", connectionSocket, addr)
            if htmlText:
                connectionSocket.send(HttpResponse(200, "OK", "text/html", htmlText))
                print("→ 200 OK")

        elif path.startswith('/ar'):
            htmlText = readFile("main_ar.html", connectionSocket, addr)
            if htmlText:
                connectionSocket.send(HttpResponse(200, "OK", "text/html", htmlText))
                print("→ 200 OK")

        elif path.startswith("/so"):
            connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\nLocation: https://stackoverflow.com\r\n\r\n".encode())
            print("→ 307 Temporary Redirect")

        elif path.startswith("/itc"):
            connectionSocket.send("HTTP/1.1 307 Temporary Redirect\r\nLocation: https://itc.birzeit.edu\r\n\r\n".encode())
            print("→ 307 Temporary Redirect")


```

Figure 2.44: Routing logic and redirection handling



```
    elif path.startswith("/getImage") and hasQueryParams:
        params = {}
        keys_values = pathParts[1].split('&')
        for key_value in keys_values:
            if '=' in key_value:
                key, value = key_value.split('=')
                params[key] = value

        ImageName = params["imageName"] if "imageName" in params.keys() else pathParts[1]
        connectionSocket.send(f"HTTP/1.1 307 Temporary Redirect\r\nLocation: /{ImageName}\r\n\r\n".encode())
        print("→ 307 Temporary Redirect")

    else:
        fileName = path[1:]

        file = readFile(fileName, connectionSocket, addr)
        if file:
            parts = fileName.split(".")

            type = "text"
            fileExtension = "plain"

            if len(parts) >= 2:
                fileExtension = parts[-1]

                if fileExtension in ('png', 'jpg', 'jpeg'):
                    type = "image"

                    if fileExtension == "jpg":
                        fileExtension = "jpeg"

            connectionSocket.send(HttpResponse(200, "OK", f"{type}/{fileExtension}", file))
            print("→ 200 OK")
except OSError:
    print ("IO error")
    connectionSocket.send(HttpResponse(404, "Not Found", "text/html", notFoundHtmlText(path, addr).encode()))
    print("→ 404 Not Found")
except:
    print("Some Internal Error")
    connectionSocket.send(HttpResponse(500, "Internal Server Error", "text/html", "Internal Server Error".encode()))
else:
    connectionSocket.close()
    print ("Responded Successfully!")
    print("----------------------------------------------")
```

Figure 2.45: Image/video redirect logic and MIME detection

2.4 Task 2 (e) – Default and Error Responses

The server responds appropriately based on the requested path:

- For requests to `/`, `/en`, `/index.html`, or `/main_en.html`, the server responds with `main_en.html`.
- For requests to `/ar` or `/main_ar.html`, the response is `main_ar.html`.
- For invalid URLs or non-existent files, the server generates a custom 404 page with:
  - Status line: `HTTP/1.1 404 Not Found`
  - Browser tab title: `Error 404`
  - Red body text: "The file is not found"
  - Client IP and port

This behaviour was verified through browser testing and command-line logs

2.5 Task 2 – Experimental Cases

To verify the proper functionality of the web server, we tested different types of requests and confirmed that the expected responses were generated according to MIME type logic and redirect/error handling.

2.5.1 Case 1: Successful Image Request

An image file named `test1.png` was requested from the server. The server responded with a 200 OK message and correctly returned the image.

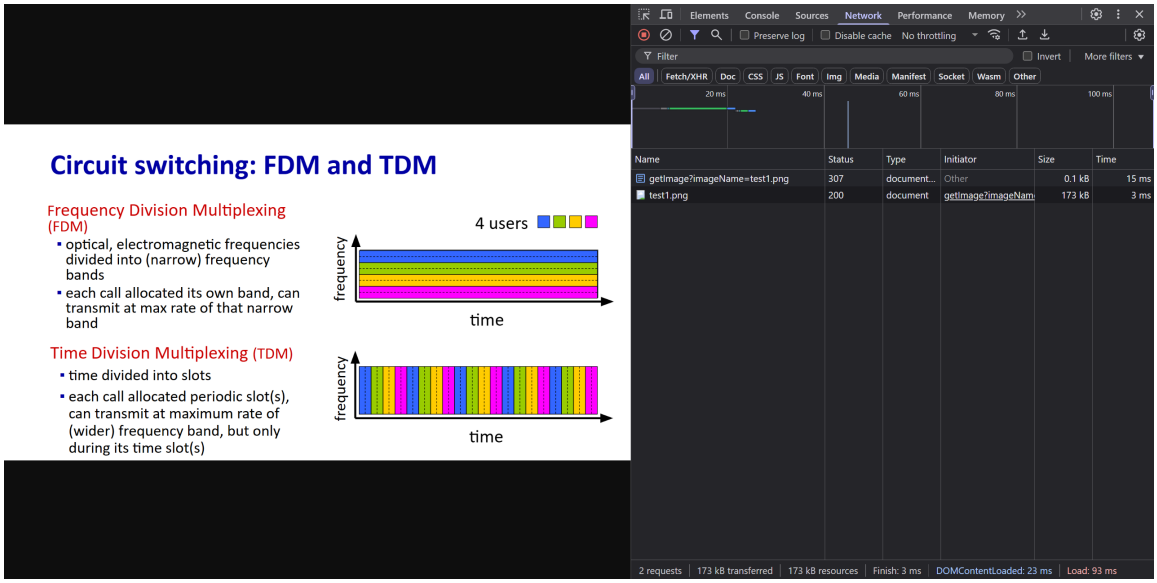


Figure 2.46: Image successfully retrieved (test1.png)

```
-----------------------------------------------
-----------------------------------------------
Request From ('192.168.56.1', 63462)
GET /test1.png HTTP/1.1
Host: 192.168.56.1:9991
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.6
Referer: http://192.168.56.1:9991/mySite_1232951_en.html
Accept-Encoding: gzip, deflate


→ 200 OK
Responded Successfully!
-----------------------------------------------
```

Figure 2.47: Terminal log showing successful image request and 200 OK response

2.5.2 Case 2: Successful Google Image Redirect (PNG)

The file `abdallah.png` was requested. The server could not locate the file locally and responded with a 307 Temporary Redirect to Google Image Search.

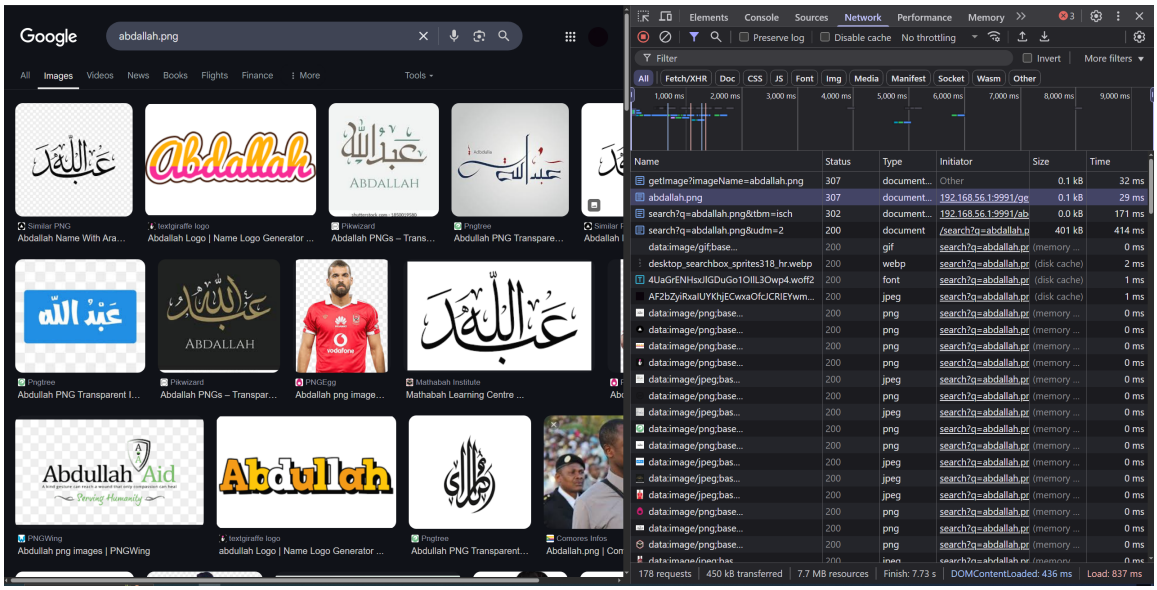


Figure 2.48: Image redirection example for abdallah.png (status 307)

2.5.3 Case 3: Successful Google Video Redirect (MP4)

A media file named `jamal.mp4` was requested. The file does not exist in the server directory. The server identified the `.mp4` MIME type and redirected the client to a YouTube search using a 307 response.

```
------------------------------------------------
------------------------------------------------
Request From ('192.168.56.1', 63190)
GET /getImage?imageName=jamal.mp4 HTTP/1.1
Host: 192.168.56.1:9991
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/136.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8
Sec-GPC: 1
Accept-Language: en-US,en;q=0.6
Referer: http://192.168.56.1:9991/mySite_1232951_en.html
Accept-Encoding: gzip, deflate


→ 307 Temporary Redirect
Responded Successfully!
------------------------------------------------
------------------------------------------------
```

Figure 2.49: Redirected search result for `jamal.mp4` to YouTube (307 Temporary Redirect)

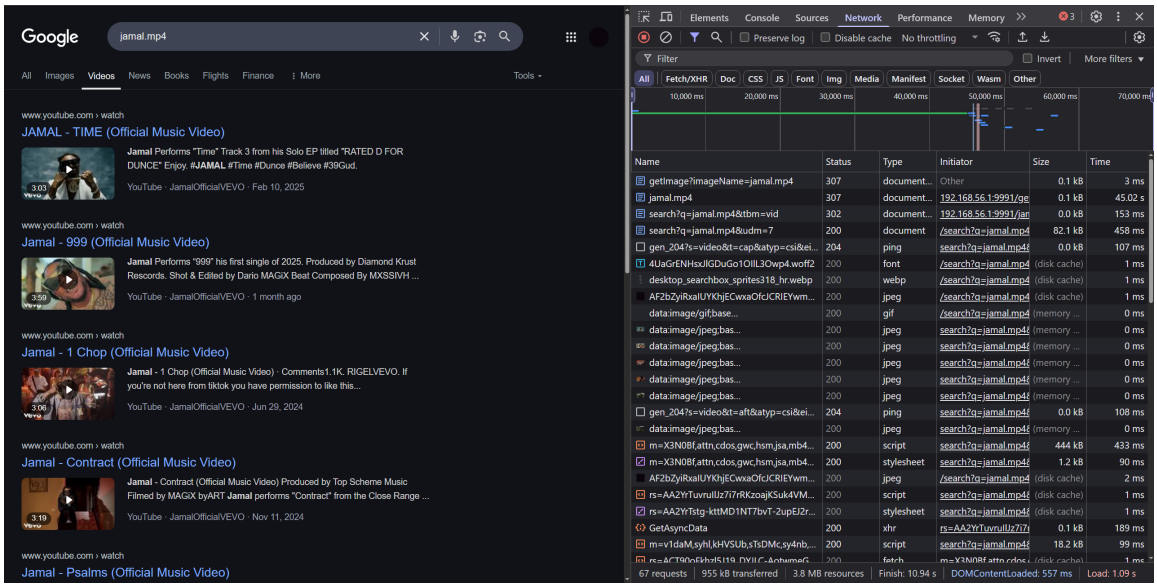


Figure 2.50: Redirect behaviour for video file (jamal.mp4) shown in browser and developer tools

2.5.4 Case 4: File Not Found (404 Error)

A text file named `jamal.txt` was requested. Since the file does not exist and doesn't match any redirect logic, the server returned a 404 error page with client details.

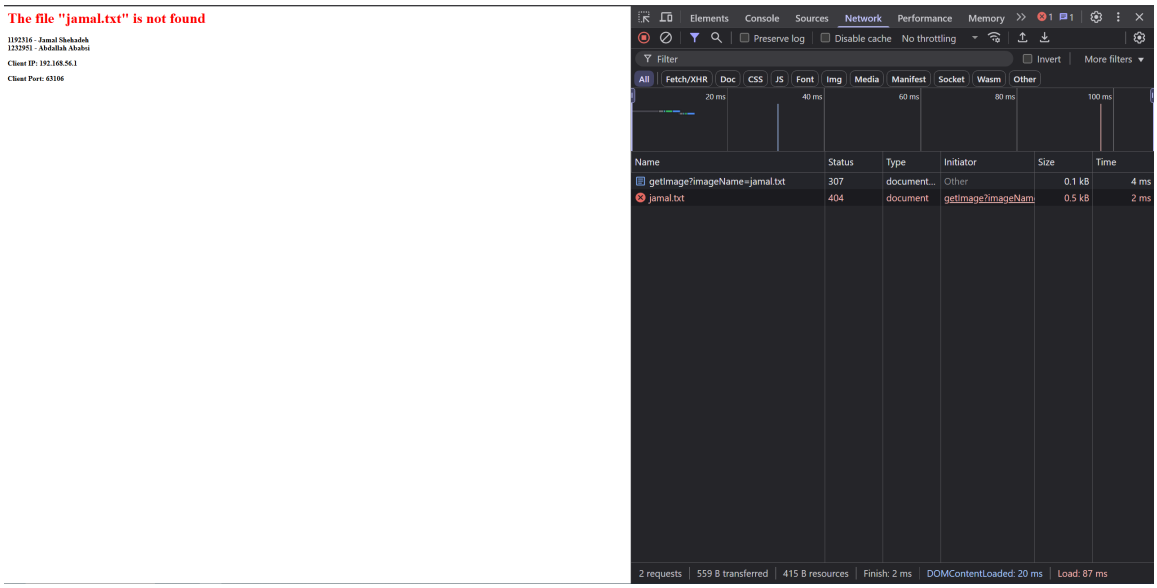


Figure 2.51: Client-side display of 404 Not Found error for jamal.txt

Figure 2.52: Server-side log output for 404 Not Found (jamal.txt)

2.6 Task 3 – TCP/UDP Hybrid Client-Server Game Using Socket Programming

2.6.1 a) Game Features

This is a multiplayer number-guessing game where clients compete to guess a secret number generated by the server. The game uses a hybrid network architecture:

- **TCP** is used for player registration, control messages, and results.
- **UDP** is used for fast real-time guessing to reduce latency.

Features include:

- Support for 2 to 4 simultaneous players.
- Unique username registration with validation.
- Game rounds with real-time feedback: "Higher", "Lower", "Correct".
- Result announcement and round restart via TCP.

b) Components and Protocol Design

TCP Phase:

- Clients connect to the server (port 6000) and send `JOIN <username>`.
- Server checks uniqueness and accepts or rejects the username.
- When enough players have joined, the game begins.

UDP Phase:

- Server selects a random number within a defined range (e.g., 1–100).
- Players send guesses to the server on UDP port 6001.
- Server replies to each player via UDP with feedback.
- The first player to guess correctly wins; results are broadcast to all clients via TCP.

2.6.2 c) Game Workflow

Server Side:

- Initializes TCP and UDP sockets.
- Waits for player registration.
- Starts game round and generates secret number.

- Collects UDP guesses, checks correctness, and announces winner.

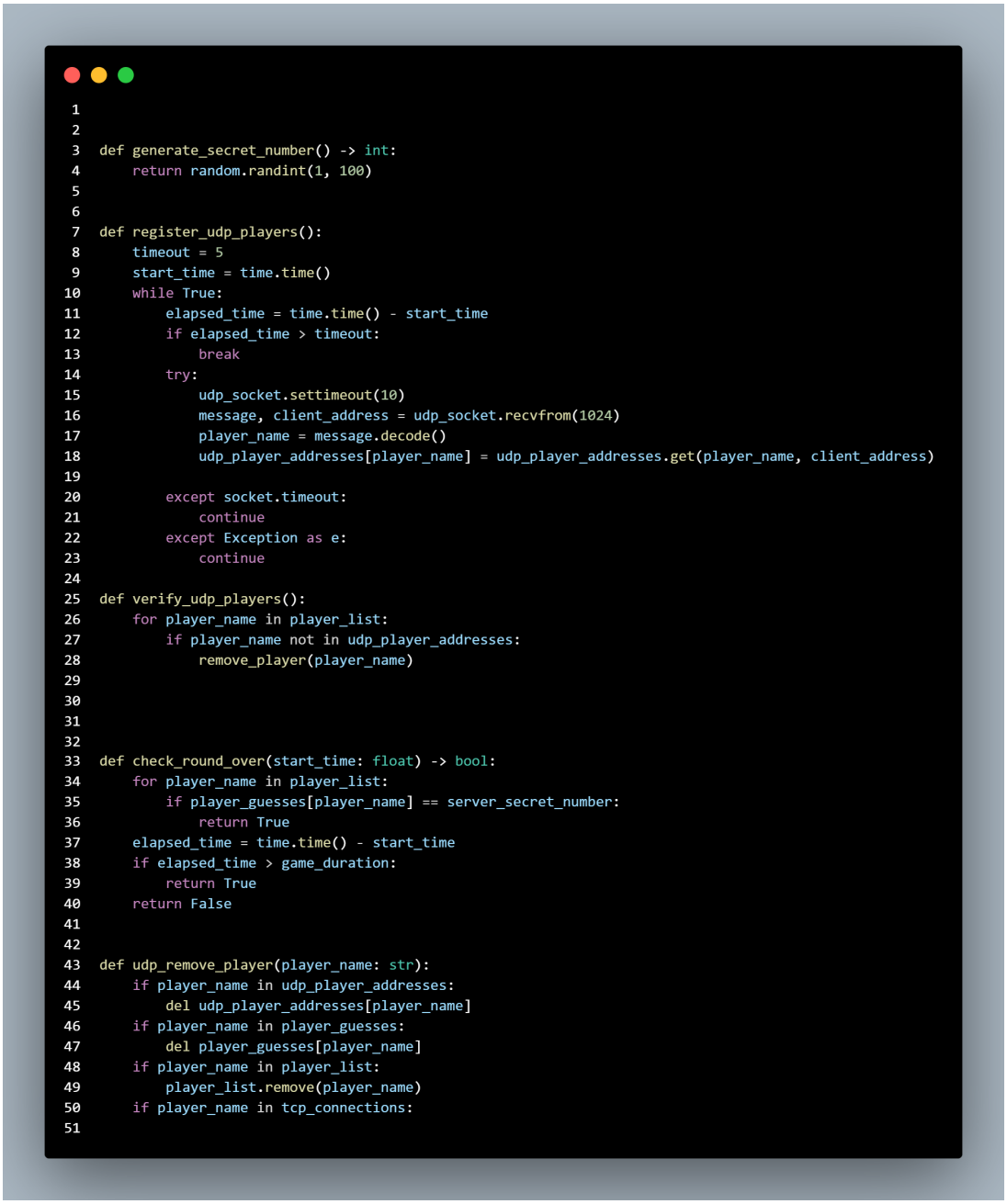
Client Side:

- Connects via TCP, submits username.
- Sends guesses via UDP.
- Listens for game updates and final results.

Code Screenshots:

```
1
2  # generate random secret number by server (done)
3  # register players with TCP (done)
4  # players attempt to guess using UDP (done)
5  # server responds toi each guess with a message using UDP (e.g., "too high", "too low", "correct") (done)
6  # if player wins, send them a message using TCP to all players (done)
7
8  import socket, random, threading, time, collections
9
10
11
12 tcp_port = 6000
13 udp_port = 6001
14 server_name = "localhost"
15 min_players = 2
16 max_players = 4
17 to_enter_your_guess = 10.0
18 game_duration = 60.0
19 player_reg_timeout = 120.0
20 player_kick_timeout = 5.0
21
22 player_list = []
23 tcp_connections = {}
24
25 udp_player_addresses = {}
26
27 server_secret_number = 0 # initialized to 0
28
29 tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
30 udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
31
32 player_guesses = {}
33
34
35
36
37
38
39 def udp_pingpong(player_name: str):
40     try:
41         udp_socket.sendto(b"ping", udp_player_addresses[player_name])
42         udp_socket.settimeout(5)
43         if udp_socket.recvfrom(1024)[0] != b"pong":
44             raise Exception("No pong received")
45     except Exception as e:
46         udp_player_addresses[player_name].close()
47         return False
48     return True
49
50
51
```

Figure 2.53: Client-side socket setup

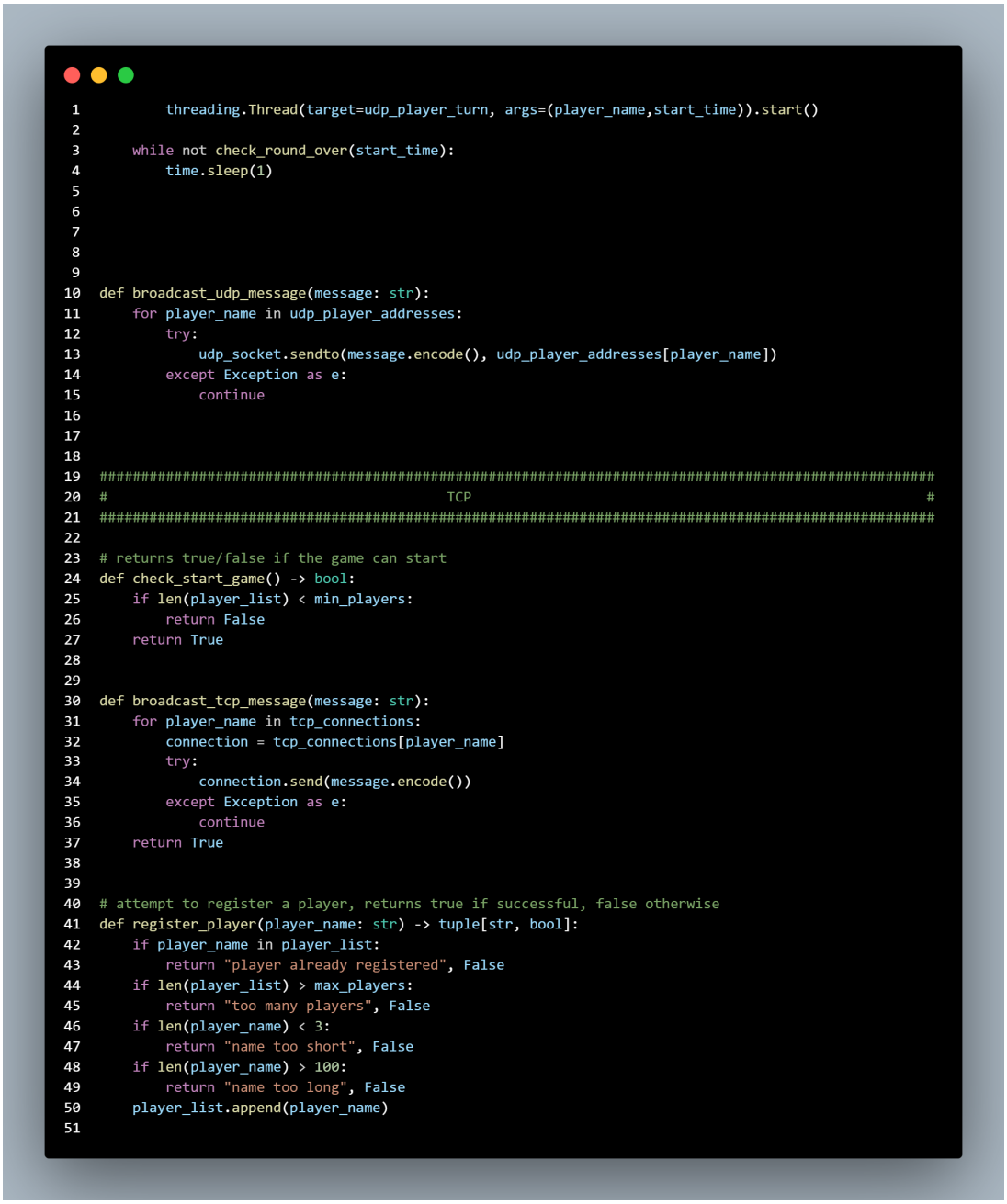


```
1
2
3   def generate_secret_number() -> int:
4       return random.randint(1, 100)
5
6
7   def register_udp_players():
8       timeout = 5
9       start_time = time.time()
10      while True:
11          elapsed_time = time.time() - start_time
12          if elapsed_time > timeout:
13              break
14          try:
15              udp_socket.settimeout(10)
16              message, client_address = udp_socket.recvfrom(1024)
17              player_name = message.decode()
18              udp_player_addresses[player_name] = udp_player_addresses.get(player_name, client_address)
19
20          except socket.timeout:
21              continue
22          except Exception as e:
23              continue
24
25  def verify_udp_players():
26      for player_name in player_list:
27          if player_name not in udp_player_addresses:
28              remove_player(player_name)
29
30
31
32
33  def check_round_over(start_time: float) -> bool:
34      for player_name in player_list:
35          if player_guesses[player_name] == server_secret_number:
36              return True
37      elapsed_time = time.time() - start_time
38      if elapsed_time > game_duration:
39          return True
40      return False
41
42
43  def udp_remove_player(player_name: str):
44      if player_name in udp_player_addresses:
45          del udp_player_addresses[player_name]
46      if player_name in player_guesses:
47          del player_guesses[player_name]
48      if player_name in player_list:
49          player_list.remove(player_name)
50      if player_name in tcp_connections:
51
```

Figure 2.54: UDP game loop and message handling

```
        tcp_connections[player_name].close()
        del tcp_connections[player_name]


def udp_player_turn(player_name: str, start_time):
    while True:
        try:
            if check_round_over(start_time):
                return

            if not udp_pingpong(player_name):
                print(f"Player {player_name} disconnected.")
                udp_remove_player(player_name)
                return



            udp_socket.sendto("Enter your guess (1-100): ".encode(), udp_player_addresses[player_name])
            message, client_address = udp_socket.recvfrom(1024)
            raw_guess = message.decode()
            try:
                guess = int(raw_guess)
                if guess < 1 or guess > 100:
                    udp_socket.sendto("Warning: Out of range, miss your chance.".encode(), client_address)
                    time.sleep(to_enter_your_guess)
                    continue
            except ValueError:
                udp_socket.sendto(f"Warning: invalid input. miss your chance.".encode(), client_address)
                time.sleep(to_enter_your_guess)
                continue

            player_guesses[player_name] = guess

            if guess == server_secret_number:
                udp_socket.sendto(f"Feedback: CORRECT".encode(), client_address)
                broadcast_tcp_message(f"{player_name} guessed the number {server_secret_number} correctly!")
                return
            elif guess < server_secret_number:
                udp_socket.sendto(f"Feedback: Higher".encode(), client_address)
                time.sleep(to_enter_your_guess)
            else:
                udp_socket.sendto(f"Feedback: Lower".encode(), client_address)
                time.sleep(to_enter_your_guess)
        except Exception as e:
            continue


def game_round():
    start_time = time.time()

```

Figure 2.55: Server TCP ping-pong and client monitoring



```
        threading.Thread(target=udp_player_turn, args=(player_name,start_time)).start()

    while not check_round_over(start_time):
        time.sleep(1)




def broadcast_udp_message(message: str):
    for player_name in udp_player_addresses:
        try:
            udp_socket.sendto(message.encode(), udp_player_addresses[player_name])
        except Exception as e:
            continue



#############################################################################################
#                                        TCP                                                #
#############################################################################################

# returns true/false if the game can start
def check_start_game() -> bool:
    if len(player_list) < min_players:
        return False
    return True


def broadcast_tcp_message(message: str):
    for player_name in tcp_connections:
        connection = tcp_connections[player_name]
        try:
            connection.send(message.encode())
        except Exception as e:
            continue
    return True


# attempt to register a player, returns true if successful, false otherwise
def register_player(player_name: str) -> tuple[str, bool]:
    if player_name in player_list:
        return "player already registered", False
    if len(player_list) > max_players:
        return "too many players", False
    if len(player_name) < 3:
        return "name too short", False
    if len(player_name) > 100:
        return "name too long", False
    player_list.append(player_name)

```

Figure 2.56: UDP player management and gameplay round

```
    waiting_room = '\n'.join(player_list)
    return f"\nConnected as {player_name}\nWaiting room:\n{waiting_room}", True



def remove_player(player_name: str):
    if player_name in player_list:
        player_list.remove(player_name)



def check_player_connections():
    for player_name in player_list:
        try:
            tcp_connections[player_name].settimeout(player_kick_timeout)
            tcp_connections[player_name].send(b"ping")
            if tcp_connections[player_name].recv(1024) != b"pong":
                raise Exception("No pong received")
        except socket.timeout:
            print(f"Player {player_name} timed out. Removing from game.")
            tcp_connections[player_name].close()
            del tcp_connections[player_name]
            remove_player(player_name)
        except Exception as e:
            print(f"Error checking connection for player {player_name}: {e}")
            tcp_connections[player_name].close()
            del tcp_connections[player_name]
            remove_player(player_name)

def close_tcp_connections():
    for con in tcp_connections:
        try:
            connection = tcp_connections[con]
            connection.close()
        except Exception as e:
            print(f"Error closing connection: {e}")
            continue
    tcp_connections.clear()

def init_sockets():
    tcp_socket.bind((server_name, tcp_port))
    tcp_socket.listen(5)
    tcp_socket.setblocking(False)
    print(f"TCP socket listening on port {tcp_port}...")


    udp_socket.bind((server_name, udp_port))
    udp_socket.setblocking(False)
    print(f"UDP socket listening on port {udp_port}...")


```

Figure 2.57: UDP guess handling and feedback logic

```
1
2  def pingpong(connection_socket: socket.socket):
3      try:
4          connection_socket.send(b"ping")
5          connection_socket.settimeout(5)
6          if connection_socket.recv(1024) != b"pong":
7              raise Exception("No pong received")
8      except socket.timeout:
9          connection_socket.close()
10         return False
11     except Exception as e:
12         connection_socket.close()
13         return False
14     return True
15
16 def handle_client(connection_socket: socket.socket, addr):
17     try:
18         # send the welcome message
19         connection_socket.send(f"Welcome to the game server! Please enter your name: ".encode())
20         connection_socket.settimeout(player_reg_timeout)
21
22         # receive the player's name
23         player_name = connection_socket.recv(1024).decode()
24         print(f"Received player name: {player_name}")
25
26         if not player_name:
27             print("No player name received. Closing connection.")
28             connection_socket.close()
29             return
30         response_message, response_status = register_player(player_name)
31
32
33         while not response_status:
34             connection_socket.send(f"{response_message}".encode())
35             player_name = connection_socket.recv(1024).decode()
36             response_message, response_status = register_player(player_name)
37
38         connection_socket.send(f"{response_message}".encode())
39
40         connection_socket.settimeout(None)
41         tcp_connections[player_name] = connection_socket
42
43         return
44
45     except TimeoutError:
46         print("Player registration timed out. Closing connection.")
47         connection_socket.close()
48         return
49
50 def accept_tcp_connections_threaded():
51
```

Figure 2.58: Round broadcasting and result distribution

```
    while True:
        # accept a new connection
        try:
            connection_socket, addr = tcp_socket.accept()
        except BlockingIOError:
            time.sleep(2)
            check_player_connections()
            if check_start_game():
                print("Game can start...")
                return
            continue
        except Exception as e:
            print(f"Error accepting connection: {e}")
            continue

        # start a new thread for each connection
        threading.Thread(target=handle_client, args=(connection_socket,addr)).start()


def ask_users_to_play():
    for player_name in player_list:
        try:
            tcp_connections[player_name].send(b"Do you want to play? (yes/no): ")
            tcp_connections[player_name].settimeout(5)
            response = tcp_connections[player_name].recv(1024).decode()
            if response.lower() == "no":
                remove_player(player_name)
                continue
        except socket.timeout:
            print(f"Player {player_name} timed out. Removing from game.")
            tcp_connections[player_name].close()
            del tcp_connections[player_name]
            remove_player(player_name)
        except Exception as e:
            print(f"Error checking connection for player {player_name}: {e}")
            tcp_connections[player_name].close()
            del tcp_connections[player_name]
            remove_player(player_name)



##############################################################################################
#                                          MAIN                                              #
##############################################################################################

def main():
    init_sockets()

    # print start message

```

Figure 2.59: TCP handling thread logic

```python
    # print start message
    print(f"Server started on {server_name}: TCP {tcp_port}, UDP {udp_port}")

    accept_tcp_connections_threaded()
    still_playing = True

    while still_playing:
        for player_name in player_list:
            player_guesses[player_name] = 0

        print(f"Game started with {len(player_list)} players...")
        broadcast_tcp_message(f"Game started with players: {', '.join(player_list)}")

        # initialize game logic
        server_secret_number = generate_secret_number()

        register_udp_players()
        time.sleep(5)
        verify_udp_players()

        broadcast_udp_message("You have 60 seconds to guess the number (1-100)!")

        print("round begin")
        game_round()

        for player_name in player_list:
            if player_guesses[player_name] == server_secret_number:
                message = """==GAME RESULTS==
                Target number: {server_secret_number}
                Winner: {player_name}
                """
                broadcast_tcp_message(f"{player_name} guessed the number {server_secret_number} correctly!")
            else:
                message = f"==GAME RESULTS==\nTarget number: {server_secret_number}\nWinner: None"
                broadcast_tcp_message(message)

        ask_users_to_play()
        still_playing = check_start_game()

    close_tcp_connections()


main()


```

Figure 2.60: Final game loop and result tracking

d) Test Cases and Expected Output

EXP – Two Players Join

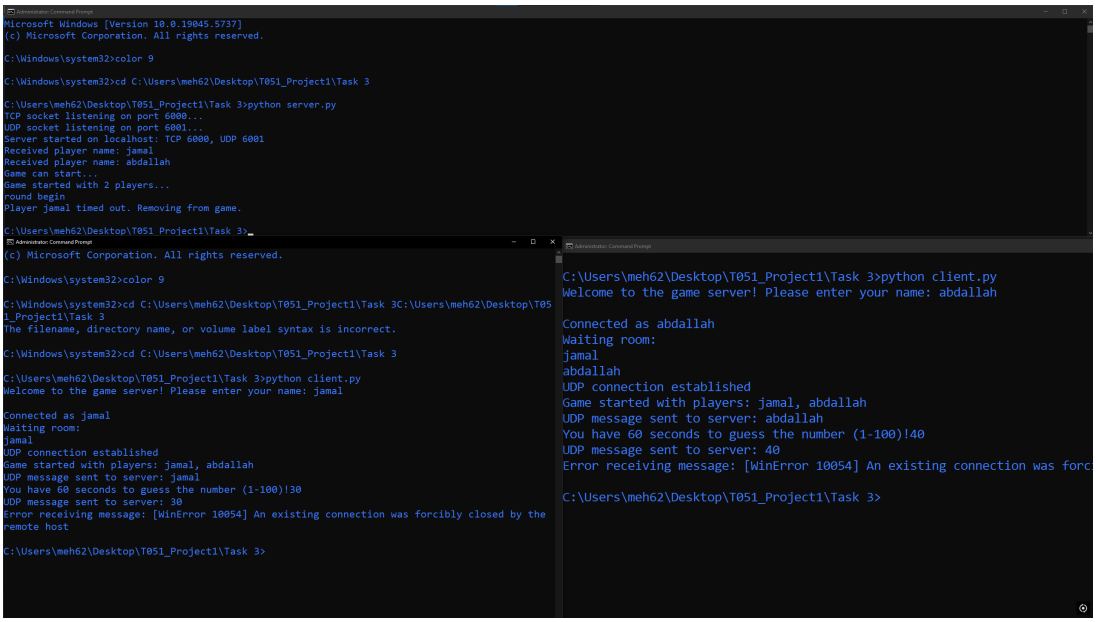


Figure 2.61: Two players join and the game starts. A bug happens.

```python

import socket
import time

udp_port = 6001
tcp_port = 6000
server_name = 'localhost'

player_patience_timeout = 60.0


player_udp_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
player_tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

def init_tcp_sockets():
    player_tcp_socket.connect((server_name, tcp_port))


def init_udp_socket():
    player_udp_socket.connect((server_name, udp_port))
    print(f"UDP connection established")



if __name__ == "__main__":
    init_tcp_sockets()

    server_message = player_tcp_socket.recv(1024).decode('utf-8')

    player_name = input(server_message)
    player_tcp_socket.sendall(player_name.encode('utf-8'))

    # receive the response from the server
    response = player_tcp_socket.recv(1024).decode('utf-8')
    print(response)
    init_udp_socket()

    start_time = time.time()
    while True:
        elapsed_time = time.time() - start_time
        if elapsed_time > player_patience_timeout:
            print("Player patience timeout reached. Exiting...")
            break
        try:
            # receive the message from the server
            message = player_tcp_socket.recv(1024).decode('utf-8')
            if not message:
                continue
            if message == "ping":
                player_tcp_socket.sendall(b"pong")
            else:
                print(message)
                break
        except socket.error as e:
            print(f"Socket error: {e}")
            break


    player_udp_socket.sendto(player_name.encode('utf-8'), (server_name, udp_port))
    print(f"UDP message sent to server: {player_name}")


    # game loop
    while True:
        try:
            # receive the message from the server
            message, addr = player_udp_socket.recvfrom(1024)
            message = message.decode('utf-8')
            if not message:
                continue
            if message == "ping":
                player_udp_socket.sendto(b"pong", (server_name, udp_port))
                continue
            guess = input(message)
            player_udp_socket.sendto(guess.encode('utf-8'), (server_name, udp_port))
            print(f"UDP message sent to server: {guess}")
            response = player_udp_socket.recv(1024).decode('utf-8')
            print(response)
        except Exception as e:
            print(f"Error receiving message: {e}")
            break


    player_tcp_socket.close()
    player_udp_socket.close()
```

Figure 2.62: Client inputs a guess outside valid range and receives a warning.

```
    # print start message
    print(f"Server started on {server_name}: TCP {tcp_port}, UDP {udp_port}")

    accept_tcp_connections_threaded()
    still_playing = True

    while still_playing:
        for player_name in player_list:
            player_guesses[player_name] = 0

        print(f"Game started with {len(player_list)} players...")
        broadcast_tcp_message(f"Game started with players: {', '.join(player_list)}")

        # initialize game logic
        server_secret_number = generate_secret_number()

        register_udp_players()
        time.sleep(5)
        verify_udp_players()

        broadcast_udp_message("You have 60 seconds to guess the number (1-100)!")

        print("round begin")
        game_round()

        for player_name in player_list:
            if player_guesses[player_name] == server_secret_number:
                message = """==GAME RESULTS==
                Target number: {server_secret_number}
                Winner: {player_name}
                """
                broadcast_tcp_message(f"{player_name} guessed the number {server_secret_number} correctly!")
            else:
                message = f"==GAME RESULTS==\nTarget number: {server_secret_number}\nWinner: None"
                broadcast_tcp_message(message)

        ask_users_to_play()
        still_playing = check_start_game()

    close_tcp_connections()


main()
```

Figure 2.63: Server detects disconnected player and handles it gracefully.

e) Initial Settings

- `tcp_port` = 6000
- `udp_port` = 6001
- `server_name` = 'localhost'
- `min_players` = 2, `max_players` = 4
- `guess_timeout` = 10 seconds
- `game_duration` = 60 seconds
- `Player Names: Player_1 = Abdallah_Abbasi, Player_2 = Jamal_Shehadeh`
- `Libraries used: socket, random, threading, time, collections`

Code Overview and Architecture

The game is implemented using:

- `server.py` – Sets up TCP/UDP sockets, manages game rounds, handles registration, guessing, and result broadcasting.
- `client.py` – Handles TCP registration and UDP-based guessing; receives feedback and results from server.

Server Highlights (`server.py`)

- Player registration is handled through dedicated TCP threads.
- UDP guesses are processed in real-time using a listener thread.
- A separate monitoring thread checks TCP connection health using ping messages.
- Results are broadcast to all clients upon a correct guess.

Client Highlights (`client.py`)

- Connects to TCP server and submits unique username.
- Sends guesses via UDP and listens for game status updates.
- Replies with "pong" when receiving TCP ping to prevent disconnection.

Technical Challenge

Despite proper hybrid implementation, a critical issue persists:

- The server periodically sends "ping" over TCP to verify clients.
- The client doesn't respond quickly enough with "pong", triggering `WinError 10054`.
- The server then closes the TCP connection and ends the round, even though UDP communication continues.

Attempts were made to fix this by implementing a TCP pong reply handler on the client side. However, due to timing or threading limitations, the issue remains. Future work should ensure a persistent TCP listener on the client side to catch and reply to pings immediately.

Chapter 3

Conclusion

Conclusion

A complete and simple network system was created and put into use for this project. A simple web server, HTML pages in Arabic and English, and a multiplayer guessing game that uses both TCP and UDP were all part of the project.

Every feature that was needed was created successfully. The web server managed redirects and errors while returning the appropriate pages. Players could register, guess a number in real time, and get feedback while playing the game.

The main logic and gameplay were finished correctly, even though a small issue in the TCP connection during the game. This project helped in understanding of socket programming and client-server communication management.