



Department of Computer Science and Engineering

*Lab Manual*

*Subject:Computer Network*

*Subject Code:*

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Experiment No. 1: Write a program for print the IP Address of a WWW.YAHOO.COM</b>                    | <b>7</b>  |
| 1.1      | Aim . . . . .   | 7         |
| 1.2      | Scope . . . . .   | 7         |
| 1.3      | Environmental Condition . . . . .   | 7         |
| 1.4      | Instruments . . . . .   | 7         |
| 1.5      | Procedure . . . . .   | 7         |
| 1.6      | Observation . . . . .   | 8         |
| 1.7      | Calculation . . . . .   | 8         |
| 1.8      | Output . . . . .  | 8         |
| 1.9      | Do's and Don'ts . . . . .   | 8         |
| 1.9.1    | Do's . . . . .  | 8         |
| 1.9.2    | Don'ts . . . . .  | 9         |
| 1.10     | References . . . . .  | 9         |
| <b>2</b> | <b>Experiment No. 2: Write a program for to print the IP Address of the local machine and hostname.</b> | <b>10</b> |
| 2.1      | Aim . . . . .   | 10        |
| 2.2      | Scope . . . . .   | 10        |
| 2.3      | Environmental Condition . . . . .   | 10        |
| 2.4      | Instruments . . . . .   | 10        |
| 2.5      | Procedure . . . . .   | 10        |
| 2.6      | Observation . . . . .   | 11        |
| 2.7      | Calculation . . . . .   | 11        |
| 2.8      | Output . . . . .  | 11        |
| 2.9      | Do's and Don'ts . . . . .   | 11        |
| 2.9.1    | Do's . . . . .  | 11        |
| 2.9.2    | Don'ts . . . . .  | 11        |
| 2.10     | References . . . . .  | 11        |
| <b>3</b> | <b>Experiment No. 3: Write HTML program to implement get( ) and post( ) methods.</b>                    | <b>12</b> |
| 3.1      | Aim . . . . .   | 12        |
| 3.2      | Scope . . . . .   | 12        |
| 3.3      | Environmental Condition . . . . .   | 12        |
| 3.4      | Instruments . . . . .   | 12        |
| 3.5      | Procedure . . . . .   | 12        |
| 3.5.1    | HTML File . . . . .   | 12        |
| 3.5.2    | Python Code . . . . .   | 15        |
| 3.6      | Observation . . . . .   | 16        |
| 3.7      | Calculation . . . . .   | 16        |
| 3.8      | Output . . . . .  | 16        |
| 3.9      | Do's and Don'ts . . . . .   | 16        |
| 3.9.1    | Do's . . . . .  | 16        |
| 3.9.2    | Don'ts . . . . .  | 16        |
| 3.10     | References . . . . .  | 16        |

|   |           |
|---|-----------|
| <b>4 Experiment No. 4: Write a program for to identify the well known ports on a Remote system.</b> | <b>17</b> |
| 4.1 Aim . . . . .   | 17        |
| 4.2 Scope . . . . .   | 17        |
| 4.3 Environmental Condition . . . . .   | 17        |
| 4.4 Instruments . . . . .   | 17        |
| 4.5 Procedure . . . . .   | 17        |
| 4.6 Observation . . . . .   | 18        |
| 4.7 Calculation . . . . .   | 18        |
| 4.8 Result . . . . .  | 18        |
| 4.9 Do's and Don'ts . . . . .   | 18        |
| 4.9.1 Do's . . . . .  | 18        |
| 4.9.2 Don'ts . . . . .  | 18        |
| 4.10 References . . . . .   | 18        |
| <b>5 Experiment No. 5: Write a program for to print the parts of URL.</b>                           | <b>19</b> |
| 5.1 Aim . . . . .   | 19        |
| 5.2 Scope . . . . .   | 19        |
| 5.3 Environmental Condition . . . . .   | 19        |
| 5.4 Instruments . . . . .   | 19        |
| 5.5 Procedure . . . . .   | 19        |
| 5.6 Observation . . . . .   | 20        |
| 5.7 Calculation . . . . .   | 20        |
| 5.8 Result . . . . .  | 20        |
| 5.9 Do's and Don'ts . . . . .   | 20        |
| 5.9.1 Do's . . . . .  | 20        |
| 5.9.2 Don'ts . . . . .  | 20        |
| 5.10 References . . . . .   | 21        |
| <b>6 Experiment No. 6:Write a program for to send &amp; receive data from datagram packet.</b>      | <b>22</b> |
| 6.1 Aim . . . . .   | 22        |
| 6.2 Scope . . . . .   | 22        |
| 6.3 Environmental Condition . . . . .   | 22        |
| 6.4 Instruments . . . . .   | 22        |
| 6.5 Procedure . . . . .   | 22        |
| 6.5.1 udp_server.py . . . . .   | 22        |
| 6.5.2 udp_client.py . . . . .   | 23        |
| 6.6 Observation . . . . .   | 23        |
| 6.7 Calculation . . . . .   | 23        |
| 6.8 Result . . . . .  | 23        |
| 6.8.1 Server Side . . . . .   | 23        |
| 6.8.2 Client Side . . . . .   | 23        |
| 6.9 Do's and Don'ts . . . . .   | 23        |
| 6.9.1 Do's . . . . .  | 23        |
| 6.9.2 Don'ts . . . . .  | 23        |

|          |   |           |
|----------|---|-----------|
| 6.10     | References . . . . .  | 24        |
| <b>7</b> | <b>Experiment No. 7: Write a program for a chat application.</b>  | <b>25</b> |
| 7.1      | Aim . . . . .   | 25        |
| 7.2      | Scope . . . . .   | 25        |
| 7.3      | Environmental Condition . . . . .   | 25        |
| 7.4      | Instruments . . . . .   | 25        |
| 7.5      | Procedure . . . . .   | 25        |
| 7.5.1    | chat_server.py . . . . .  | 25        |
| 7.5.2    | chat_client.py . . . . .  | 26        |
| 7.6      | Observation . . . . .   | 26        |
| 7.7      | Calculation . . . . .   | 26        |
| 7.8      | Result . . . . .  | 26        |
| 7.8.1    | Server Side . . . . .   | 26        |
| 7.8.2    | Client Side . . . . .   | 26        |
| 7.9      | Do's and Don'ts . . . . .   | 26        |
| 7.9.1    | Do's . . . . .  | 26        |
| 7.9.2    | Don'ts . . . . .  | 27        |
| 7.10     | References . . . . .  | 27        |
| <b>8</b> | <b>Experiment No. 8: Write a program for the simple file transfer between two systems by opening socket connection to out server on one system and sending a file from one system to another.</b> | <b>28</b> |
| 8.1      | Aim . . . . .   | 28        |
| 8.2      | Scope . . . . .   | 28        |
| 8.3      | Environmental Condition . . . . .   | 28        |
| 8.4      | Instruments . . . . .   | 28        |
| 8.5      | Procedure . . . . .   | 28        |
| 8.5.1    | file_server.py . . . . .  | 28        |
| 8.5.2    | file_client.py . . . . .  | 29        |
| 8.6      | Observation . . . . .   | 29        |
| 8.7      | Calculation . . . . .   | 29        |
| 8.8      | Result . . . . .  | 30        |
| 8.8.1    | Server Side . . . . .   | 30        |
| 8.8.2    | Client Side . . . . .   | 30        |
| 8.9      | Do's and Don'ts . . . . .   | 30        |
| 8.9.1    | Do's . . . . .  | 30        |
| 8.9.2    | Don'ts . . . . .  | 30        |
| 8.10     | References . . . . .  | 30        |
| <b>9</b> | <b>Experiment No. 9: Write a program for the HTTP server.</b>   | <b>31</b> |
| 9.1      | Aim . . . . .   | 31        |
| 9.2      | Scope . . . . .   | 31        |
| 9.3      | Environmental Condition . . . . .   | 31        |
| 9.4      | Instruments . . . . .   | 31        |
| 9.5      | Procedure . . . . .   | 31        |
| 9.5.1    | Python HTTP Server Code . . . . .   | 31        |

|           |   |           |
|-----------|---|-----------|
| 9.5.2     | index.html . . . . .  | 32        |
| 9.6       | Observation . . . . .   | 32        |
| 9.7       | Calculation . . . . .   | 32        |
| 9.8       | Result . . . . .  | 32        |
| 9.9       | Do's and Don'ts . . . . .   | 32        |
| 9.9.1     | Do's . . . . .  | 32        |
| 9.9.2     | Don'ts . . . . .  | 32        |
| 9.10      | References . . . . .  | 33        |
| <b>10</b> | <b>Experiment No. 10: Implement the concept of static routing.</b>                      | <b>34</b> |
| 10.1      | Aim . . . . .   | 34        |
| 10.2      | Scope . . . . .   | 34        |
| 10.3      | Environmental Condition . . . . .   | 34        |
| 10.4      | Instruments . . . . .   | 34        |
| 10.5      | Procedure . . . . .   | 34        |
| 10.5.1    | Conceptual Network Topology . . . . .   | 34        |
| 10.6      | Observation . . . . .   | 35        |
| 10.7      | Calculation . . . . .   | 35        |
| 10.8      | Result . . . . .  | 35        |
| 10.9      | Do's and Don'ts . . . . .   | 35        |
| 10.9.1    | Do's . . . . .  | 35        |
| 10.9.2    | Don'ts . . . . .  | 36        |
| 10.10     | References . . . . .  | 36        |
| <b>11</b> | <b>Experiment No. 11: Implement the concept of dynamic routing (RIP, OSPF, BGP).</b>    | <b>37</b> |
| 11.1      | Aim . . . . .   | 37        |
| 11.2      | Scope . . . . .   | 37        |
| 11.3      | Environmental Condition . . . . .   | 37        |
| 11.4      | Instruments . . . . .   | 37        |
| 11.5      | Procedure . . . . .   | 37        |
| 11.5.1    | Conceptual Network Topology . . . . .   | 37        |
| 11.6      | Observation . . . . .   | 39        |
| 11.7      | Calculation . . . . .   | 39        |
| 11.8      | Result . . . . .  | 39        |
| 11.9      | Do's and Don'ts . . . . .   | 41        |
| 11.9.1    | Do's . . . . .  | 41        |
| 11.9.2    | Don'ts . . . . .  | 41        |
| 11.10     | References . . . . .  | 41        |
| <b>12</b> | <b>Experiment No. 12: Packet capture and header analysis by wire-shark (TCP,UDP,IP)</b> | <b>42</b> |
| 12.1      | Aim . . . . .   | 42        |
| 12.2      | Scope . . . . .   | 42        |
| 12.3      | Environmental Condition . . . . .   | 42        |
| 12.4      | Instruments . . . . .   | 42        |
| 12.5      | Procedure . . . . .   | 42        |
| 12.6      | Observation . . . . .   | 43        |

|                                |    |
|--------------------------------|----|
| 12.7 Calculation . . . . .     | 43 |
| 12.8 Result . . . . .          | 44 |
| 12.9 Do's and Don'ts . . . . . | 45 |
| 12.9.1 Do's . . . . .          | 45 |
| 12.9.2 Don'ts . . . . .        | 45 |
| 12.10References . . . . .      | 45 |

# 1 Experiment No. 1: Write a program for print the IP Address of a WWW.YAHOO.COM

## 1.1 Aim

The aim of this experiment is to write a simple Python program that can resolve a given hostname (like www.yahoo.com) into its corresponding Internet Protocol (IP) address. This demonstrates how the Domain Name System (DNS) works in practice to translate human-readable website names into numerical addresses that computers use to locate each other on the internet.

## 1.2 Scope

This program resolves a single hostname to its IPv4 address, demonstrating basic DNS lookup; it's not a comprehensive network utility.

## 1.3 Environmental Condition

Not Specifically Required

## 1.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

## 1.5 Procedure

```
import socket

def get_ip_address(hostname):
    """
    Retrieves the IP address for a given hostname.

    Args:
        hostname (str): The domain name (e.g., "www.yahoo.com").

    Returns:
        str: The IP address of the hostname, or an error message if not found.
    """
    try:
        # Get the IP address associated with the hostname
```

```
# gethostbyname returns the IP address as a string
ip_address = socket.gethostbyname(hostname)
return ip_address
except socket.gaierror:
    # Handle the error if the hostname cannot be resolved
    return f"Error: Could not resolve hostname '{hostname}'.
    Please check the spelling or your internet connection."
except Exception as e:
    # Catch any other unexpected errors
    return f"An unexpected error occurred: {e}"

# Define the hostname we want to look up
target_hostname = "www.yahoo.com"

# Call the function to get the IP address
resolved_ip = get_ip_address(target_hostname)

# Print the result
print(f"The IP address of {target_hostname} is: {resolved_ip}")
```

## 1.6 Observation

Not specifically Required

## 1.7 Calculation

Not specifically Required

## 1.8 Output

The IP address of [www.yahoo.com](http://www.yahoo.com) is: 69.147.88.8

## 1.9 Do's and Don'ts

### 1.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor



### **1.9.2 Don'ts**

- Don't Run Without Internet
- Don't Ignore Errors

### **1.10 References**

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 2 Experiment No. 2: Write a program for to print the IP Address of the local machine and hostname.

### 2.1 Aim

The aim of this experiment is to write a simple Python program that can retrieve and display the hostname and the primary IP address of the local machine on which the program is executed. This demonstrates how a computer can identify itself on a network and obtain its own network address.

### 2.2 Scope

The scope of this practical identifies the local machine's hostname and primary private IP, demonstrating basic self-identification, not public IPs or diagnostics.

### 2.3 Environmental Condition

Not Specifically Required

### 2.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 2.5 Procedure

```
import socket

def get_local_network_info():
    """
    Prints the IP address and hostname of the local machine.
    """
    try:
        hostname = socket.gethostname()
        ip_address = socket.gethostbyname(hostname)

        print(f"Hostname: {hostname}")
        print(f"IP Address: {ip_address}")
    except socket.error as e:
        print(f"Error getting network information: {e}")
```

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

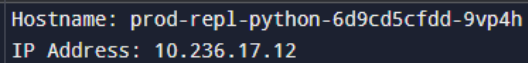
## 2.6 Observation

Not specifically Required

## 2.7 Calculation

Not specifically Required

## 2.8 Output



```
Hostname: prod-repl-python-6d9cd5cfdd-9vp4h  
IP Address: 10.236.17.12
```

## 2.9 Do's and Don'ts

### 2.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 2.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 2.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

### 3 Experiment No. 3: Write HTML program to implement get( ) and post( ) methods.

#### 3.1 Aim

The aim of this experiment is to write a simple Python program that can retrieve and display the hostname and the primary IP address of the local machine on which the program is executed. This demonstrates how a computer can identify itself on a network and obtain its own network address.

#### 3.2 Scope

#### 3.3 Environmental Condition

Not Specifically Required

#### 3.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

#### 3.5 Procedure

##### 3.5.1 HTML File

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>GET vs. POST Demonstration</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      background-color: #f4f4f4;
      color: #333;
    }
    .container {
      background-color: #fff;
      padding: 30px;
```

```
        border-radius: 8px;
        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        max-width: 600px;
        margin: 30px auto;
    }
    h2 {
        color: #0056b3;
        border-bottom: 2px solid #eee;
        padding-bottom: 10px;
        margin-bottom: 20px;
    }
    form {
        margin-bottom: 30px;
        padding: 20px;
        border: 1px solid #ddd;
        border-radius: 5px;
        background-color: #f9f9f9;
    }
    label {
        display: block;
        margin-bottom: 8px;
        font-weight: bold;
    }
    input[type="text"],
    input[type="email"] {
        width: calc(100% - 22px);
        padding: 10px;
        margin-bottom: 15px;
        border: 1px solid #ccc;
        border-radius: 4px;
        font-size: 16px;
    }
    input[type="submit"] {
        background-color: #007bff;
        color: white;
        padding: 12px 20px;
        border: none;
        border-radius: 5px;
        cursor: pointer;
        font-size: 16px;
        transition: background-color 0.3s ease;
    }
    input[type="submit"]:hover {
        background-color: #0056b3;
    }
    p {
```

```
        line-height: 1.6;
    }
    .note {
        background-color: #e6f7ff;
        border-left: 5px solid #2196F3;
        padding: 15px;
        margin-top: 20px;
        border-radius: 4px;
        color: #333;
    }
</style>
</head>
<body>
    <div class="container">
        <h1>Understanding GET and POST Methods</h1>

        <div class="note">
            <p><strong>Important Note:</strong> HTML forms send data
            to a server. To fully see the 'GET' and 'POST' methods in
            action (how data is processed), you would need a server-side
            script (e.g., in Python, PHP, Node.js) to receive and display
            the submitted data. This HTML page only demonstrates how the
            forms are structured to send the data.</p>
            <p>For this example, the 'action' attribute points to placeholder
            paths. If you were to implement this, you'd replace '/process_get'
            and '/process_post' with the actual URLs of your server-side
            scripts.</p>
        </div>

        <h2>GET Method Example</h2>
        <p>The 'GET' method appends form data to the URL as query string
        parameters. This means the data will be visible in the browser's
        address bar. It's generally used for retrieving data and when the
        form submission should be bookmarkable or shareable.</p>
        <p><strong>Observe the URL after submission!</strong></p>

        <form action="/process_get" method="GET">
            <label for="get_name">Name:</label>
            <input type="text" id="get_name" name="name" required>

            <label for="get_email">Email:</label>
            <input type="email" id="get_email" name="email" required>

            <input type="submit" value="Submit (GET)">
        </form>
```

```
<h2>POST Method Example</h2>
<p>The 'POST' method sends form data in the body of the HTTP
request. This data is not visible in the URL, making it more
suitable for sensitive information (like passwords) or when
submitting large amounts of data. 'POST' requests are generally
used for creating or updating resources on the server.</p>
<p><strong>The URL will not change significantly after submission.
</strong></p>

<form action="/process_post" method="POST">
    <label for="post_username">Username:</label>
    <input type="text" id="post_username" name="username" required>

    <label for="post_password">Password:</label>
    <input type="text" id="post_password" name="password" required>

    <input type="submit" value="Submit (POST)">
</form>
</div>
</body>
</html>
```

### 3.5.2 Python Code

```
from flask import Flask, request, render_template

app = Flask(__name__)

@app.route('/')
def index():
    # Renders the HTML form
    return render_template('index.html')

@app.route('/process_get', methods=['GET'])
def process_get():
    name = request.args.get('name')
    email = request.args.get('email')
    return f"<h1>GET Request Received!</h1><p>Name: {name}</p><p>Email: {email}</p><p>Data was in the URL query string.</p>"

@app.route('/process_post', methods=['POST'])
def process_post():
    username = request.form.get('username')
    password = request.form.get('password')
    return f"<h1>POST Request Received!</h1><p>Username: {username}</p>
<p>Password: {password}</p>"
```

<p>Data was in the request body (not visible in URL).</p>”

```
if __name__ == '__main__':  
    # Make sure 'index.html' is in a folder named 'templates' in  
    the same directory as app.py  
    app.run(debug=True)
```

### 3.6 Observation

Not specifically Required

### 3.7 Calculation

Not specifically Required

### 3.8 Output

```
* Serving Flask app 'app'  
* Debug mode: on  
WARNING: This is a development server. Do not use it in a production deployment  
* Running on http://127.0.0.1:5000  
Press CTRL+C to quit  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: XXX-XXX-XXX
```

### 3.9 Do's and Don'ts

#### 3.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

#### 3.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

### 3.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall



## 4 Experiment No. 4: Write a program for to identify the well known ports on a Remote system.

### 4.1 Aim

The aim of this experiment is to identify open well-known ports on a remote system using Python socket programming, demonstrating how port scanning can reveal active services and assess network visibility.

### 4.2 Scope

The scope of this port-scanning experiment includes identifying open well-known ports (0–1023) on a remote system using TCP connections in Python, understanding service discovery, interpreting socket behavior, and exploring basic cybersecurity concepts such as network visibility and reconnaissance techniques. It also lays the groundwork for more advanced network scanning and monitoring tools.

### 4.3 Environmental Condition

Not Specifically Required

### 4.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 4.5 Procedure

```
import socket

# Remote target (change to the actual IP or hostname)
remote_host = 'example.com' # Replace with your target hostname or IP

# Well-known ports range
well_known_ports = range(0, 1024)

print(f"Scanning well-known ports on {remote_host}...")

for port in well_known_ports:
    try:
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.settimeout(0.5) # Optional: speed up slow scans
    result = s.connect_ex((remote_host, port))
    if result == 0:
        print(f"[+] Port {port} is open")
except socket.error:
    print(f"[-] Could not connect to {remote_host}")
    break
```

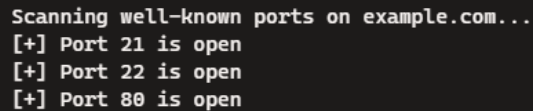
## 4.6 Observation

Not specifically Required

## 4.7 Calculation

Not specifically Required

## 4.8 Result



```
Scanning well-known ports on example.com...
[+] Port 21 is open
[+] Port 22 is open
[+] Port 80 is open
```

## 4.9 Do's and Don'ts

### 4.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 4.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 4.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 5 Experiment No. 5: Write a program for to print the parts of URL.

### 5.1 Aim

The aim of this experiment is to extract and display the individual components of a URL using Python's `urllib.parse` module, enhancing understanding of URL structure for web development and data processing.

### 5.2 Scope

The scope of this URL parsing experiment includes breaking down a URL into its components—such as scheme, hostname, port, path, query, and fragment—using Python's `urllib.parse` module. It introduces basic web concepts, supports applications like web scraping and data validation, and lays a foundation for more advanced URL handling in networked and web-based applications.

### 5.3 Environmental Condition

Not Specifically Required

### 5.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 5.5 Procedure

```
from urllib.parse import urlparse

# Sample URL
url = 'https://example.com:8080/path/to/page?name=John&age=30#section2'

# Parse the URL
parsed_url = urlparse(url)

# Print components
print("Scheme      :", parsed_url.scheme)
print("Netloc       :", parsed_url.netloc)
print("Hostname     :", parsed_url.hostname)
print("Port         :", parsed_url.port)
```

```
print("Path          :", parsed_url.path)
print("Params       :", parsed_url.params)
print("Query        :", parsed_url.query)
print("Fragment     :", parsed_url.fragment)
```

## 5.6 Observation

Not specifically Required

## 5.7 Calculation

Not specifically Required

## 5.8 Result



```
Scheme      : https
Netloc      : example.com:8080
Hostname    : example.com
Port        : 8080
Path        : /path/to/page
Params      :
Query       : name=John&age=30
Fragment    : section2
```

## 5.9 Do's and Don'ts

### 5.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 5.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 5.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 6 Experiment No. 6: Write a program for to send & receive data from datagram packet.

### 6.1 Aim

The aim of this experiment is to implement communication using datagram packets via the UDP protocol in Python, showcasing connectionless data exchange, message transfer, and real-time interaction between two systems.

### 6.2 Scope

The scope of this datagram packet communication experiment includes enabling connectionless communication between two systems using the UDP protocol, transmitting messages with minimal overhead, understanding real-time data exchange concepts, and applying socket programming principles in Python. It also introduces the practical differences between TCP and UDP through hands-on implementation.

### 6.3 Environmental Condition

Not Specifically Required

### 6.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 6.5 Procedure

#### 6.5.1 udp\_server.py

```
import socket
HOST = '0.0.0.0' PORT = 5003
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as server_socket: server_socket.bind((HOST,
PORT)) print(f'UDP Server listening on port PORT...')
while True: data, addr = server_socket.recvfrom(1024) print(f'Received from addr: {data.decode()}')
```

### 6.5.2 udp\_client.py

```
import socket
SERVER_IP = '192.168.1.5' # Replace with server's IP PORT = 5003
with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as client_socket: while True: msg
= input("Enter message to send (or 'exit'): ") if msg.lower() == 'exit': break client_socket.sendto(msg.encode()),
(SERVER_IP, PORT))
```

## 6.6 Observation

Not specifically Required

## 6.7 Calculation

Not specifically Required

## 6.8 Result

### 6.8.1 Server Side

```
UDP Server listening on port 5003...
Received from ('192.168.1.20', 56789): Hello, Server!
Received from ('192.168.1.20', 56789): Testing UDP chat.
```

### 6.8.2 Client Side

```
Enter message to send (or 'exit'): Hello, Server!
Enter message to send (or 'exit'): Testing UDP chat.
Enter message to send (or 'exit'): exit
```

## 6.9 Do's and Don'ts

### 6.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 6.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 6.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall



## 7 Experiment No. 7: Write a program for a chat application.

### 7.1 Aim

The aim of this experiment is to build a basic real-time chat application using socket programming in Python, enabling two systems to exchange messages over a network and understand fundamental client-server communication.

### 7.2 Scope

The scope of this chat application experiment includes implementing real-time communication between two systems using socket programming in Python, exploring multithreading for simultaneous sending and receiving of messages, and understanding client-server architecture fundamentals in network programming. This experiment lays the foundation for developing more advanced messaging systems.

### 7.3 Environmental Condition

Not Specifically Required

### 7.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 7.5 Procedure

#### 7.5.1 chat\_server.py

```
import socket
import threading

HOST = '0.0.0.0'
PORT = 5002

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen(1)

print(f"Server listening on port {PORT}...")
conn, addr = server_socket.accept()
print(f"Connected by {addr}")

def receive():
    while True:
        data = conn.recv(1024)
        if not data:
            break
        print(f"Client: {data.decode()}")

def send():
    while True:
        msg = input()
        conn.send(msg.encode())

threading.Thread(target=receive).start()
threading.Thread(target=send).start()
```

### 7.5.2 chat\_client.py

```
import socket
import threading

SERVER_IP = '192.168.1.5' # Replace with server's IP
PORT = 5002

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((SERVER_IP, PORT))

def receive():
    while True:
        data = client_socket.recv(1024)
        if not data:
            break
        print(f'Server: {data.decode()}")

def send():
    while True:
        msg = input()
        client_socket.send(msg.encode())

threading.Thread(target=receive).start()
threading.Thread(target=send).start()
```

## 7.6 Observation

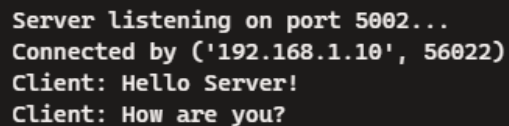
Not specifically Required

## 7.7 Calculation

Not specifically Required

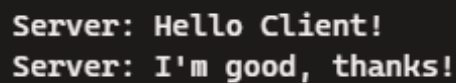
## 7.8 Result

### 7.8.1 Server Side



```
Server listening on port 5002...
Connected by ('192.168.1.10', 56022)
Client: Hello Server!
Client: How are you?
```

### 7.8.2 Client Side



```
Server: Hello Client!
Server: I'm good, thanks!
```

## 7.9 Do's and Don'ts

### 7.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### **7.9.2 Don'ts**

- Don't Run Without Internet
- Don't Ignore Errors

### **7.10 References**

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 8 Experiment No. 8: Write a program for the simple file transfer between two systems by opening socket connection to out server on one system and sending a file from one system to another.

### 8.1 Aim

The aim of this experiment is to demonstrate simple file transfer between two systems using socket programming in Python, showcasing network communication, data transmission, and basic client-server interaction principles.

### 8.2 Scope

The scope of this file transfer experiment includes establishing socket-based communication between two systems, transmitting a file over a network, understanding client-server architecture, and demonstrating basic network programming using Python.

### 8.3 Environmental Condition

Not Specifically Required

### 8.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 8.5 Procedure

#### 8.5.1 file\_server.py

```
import socket

HOST = '0.0.0.0' # Listen on all interfaces
PORT = 5001

with socket.socket() as server_socket:
    server_socket.bind((HOST, PORT))
    server_socket.listen(1)
    print(f"Server listening on port {PORT}...")
```

```
conn, addr = server_socket.accept()
with conn:
    print(f"Connection from {addr}")
    with open('received_file.txt', 'wb') as file:
        while True:
            data = conn.recv(1024)
            if not data:
                break
            file.write(data)

    print("File received successfully!")
```

### 8.5.2 file\_client.py

```
import socket

SERVER_IP = '192.168.1.5' # Replace with the server's IP address
PORT = 5001
FILENAME = 'file_to_send.txt' # File to be sent

with socket.socket() as client_socket:
    client_socket.connect((SERVER_IP, PORT))
    with open(FILENAME, 'rb') as file:
        data = file.read(1024)
        while data:
            client_socket.send(data)
            data = file.read(1024)

    print("File sent successfully!")
```

## 8.6 Observation

Not specifically Required

## 8.7 Calculation

Not specifically Required

## 8.8 Result

### 8.8.1 Server Side

```
Server listening on port 5001...  
Connection from ('192.168.1.10', 54723)  
File received successfully!
```

### 8.8.2 Client Side

```
File sent successfully!
```

## 8.9 Do's and Don'ts

### 8.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 8.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 8.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 9 Experiment No. 9: Write a program for the HTTP server.

### 9.1 Aim

To build a basic HTTP server that serves static content using Python's standard library, demonstrating fundamental networking concepts, server-client communication, and how web servers handle HTTP requests.

### 9.2 Scope

The scope of this HTTP server experiment includes creating a lightweight web server using Python, serving static files like HTML, understanding basic server-client architecture, and enabling local testing of web content.

### 9.3 Environmental Condition

Not Specifically Required

### 9.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Python Interpreter: Python 3.x installed on your computer.
- Internet Connection: Essential to perform the DNS lookup.
- Text Editor or IDE: To write and save the Python code (e.g., Notepad, VS Code, PyCharm, Sublime Text).
- Command Line/Terminal: To execute the Python script.

### 9.5 Procedure

#### 9.5.1 Python HTTP Server Code

```
from http.server import SimpleHTTPRequestHandler, HTTPServer

PORT = 8080

class MyHandler(SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.path = 'index.html'
        return super().do_GET()

with HTTPServer(('', PORT), MyHandler) as server:
    print(f"Server running on http://localhost:{PORT}")
    server.serve_forever()
```

### 9.5.2 index.html

```
<!DOCTYPE html>
<html>
<head>
  <title>My HTTP Server</title>
</head>
<body>
  <h1>Hello from your custom HTTP server!</h1>
</body>
</html>
```

## 9.6 Observation

Not specifically Required

## 9.7 Calculation

Not specifically Required

## 9.8 Result



Server running on http://localhost:8080

## 9.9 Do's and Don'ts

### 9.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 9.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors



## 9.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 10 Experiment No. 10: Implement the concept of static routing.

### 10.1 Aim

The primary goal of this practical is to demonstrate and observe the manual configuration of routes on routers to establish connectivity between non-directly connected networks. It focuses on understanding the explicit, administrator-defined nature of static routes and their advantages/disadvantages compared to dynamic routing.

### 10.2 Scope

The scope is to configure and verify manual routes on routers, establishing connectivity to non-directly connected networks, highlighting static routing's explicit nature and lack of automatic adaptation to topology changes. This experiment visually demonstrates manually configuring static routes on virtual routers, establishing connectivity between networks and highlighting their fixed, non-adaptive nature.

### 10.3 Environmental Condition

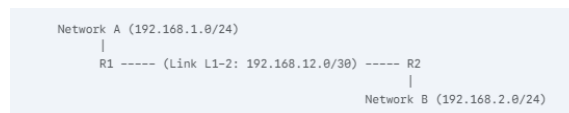
Not Specifically Required

### 10.4 Instruments

- Cisco Packet Tracer: (Easier for beginners, GUI-based, Cisco-specific but concepts apply generally).
- GNS3 / EVE-NG: (More powerful, emulates real router OS, higher learning curve).
- Virtual Machines (e.g., using VirtualBox/VMware with Linux VMs running Quagga/FR-Routing or VyOS): Allows you to build a network of virtual routers.

### 10.5 Procedure

#### 10.5.1 Conceptual Network Topology



- Design the Topology: Drag and drop routers and end devices (PCs) onto the canvas. Connect them with appropriate cable types.
- Assign IP Addresses:
  - R1:
    - \* Interface to Network A: 192.168.1.1 255.255.255.0

- \* Interface to L1-2: 192.168.12.1 255.255.255.252
- R2:
  - \* Interface to L1-2: 192.168.12.2 255.255.255.252
  - \* Interface to Network B: 192.168.2.1 255.255.255.0
- PC in Network A: 192.168.1.10 255.255.255.0 (Default Gateway: 192.168.1.1)
- PC in Network B: 192.168.2.10 255.255.255.0 (Default Gateway: 192.168.2.1)
- Configure Static Routes: This is the core of the experiment.
- Verify Routing Tables: Use commands like show ip route on each router to see the manually configured static routes.
- Test Connectivity: From the PC in Network A, ping the PC in Network B (ping 192.168.2.10). The ping should be successful.
- Simulate a Link Failure

## 10.6 Observation

Not specifically Required

## 10.7 Calculation

Not specifically Required

## 10.8 Result

```
R1>en
R1#conf t
R1(config)#int g0/0
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#no shut
R1(config-if)#int g0/1
R1(config-if)#ip address 192.168.12.1 255.255.255.252
R1(config-if)#no shut
R1(config-if)#exit
R1(config)#end
```

## 10.9 Do's and Don'ts

### 10.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### **10.9.2 Don'ts**

- Don't Run Without Internet
- Don't Ignore Errors

### **10.10 References**

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 11 Experiment No. 11: Implement the concept of dynamic routing (RIP, OSPF, BGP).

### 11.1 Aim

The primary goal of this practical is to demonstrate and observe the automated route discovery and path selection capabilities of dynamic routing protocols (RIP, OSPF, BGP) in a multi-router network, contrasting them with static routing approaches.

### 11.2 Scope

This experiment visually demonstrates dynamic routing (RIP, OSPF, BGP) on virtual routers, observing automated route discovery, path selection, and network convergence upon topology changes.

### 11.3 Environmental Condition

Not Specifically Required

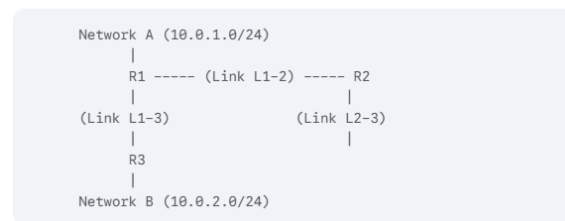
### 11.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Cisco Packet Tracer: (Easier for beginners, GUI-based, Cisco-specific but concepts apply generally).
- GNS3 / EVE-NG: (More powerful, emulates real router OS, higher learning curve).
- Virtual Machines (e.g., using VirtualBox/VMware with Linux VMs running Quagga/FR-Routing or VyOS): Allows you to build a network of virtual routers.

### 11.5 Procedure

#### 11.5.1 Conceptual Network Topology

Imagine a simple network with three routers (R1, R2, R3) connected as follows:



- Design the Topology: Drag and drop routers, switches (if needed for LAN segments), and end devices (PCs) onto the canvas. Connect them with appropriate cable types.

- Assign IP Addresses: Configure IP addresses and subnet masks on all active interfaces of the routers and end devices. Ensure interfaces on the same link are in the same subnet.
- No Static Routes (Initially): Do not configure any static routes between the routers at first.
- Enable Routing Protocol (One at a time):
  - For RIP:
    - \* On each router, enable RIP (e.g., router rip command in Cisco IOS).
    - \* Advertise directly connected networks using the network command (e.g., network 10.0.0.0 for all 10.x.x.x networks).
  - For OSPF:
    - \* On each router, enable OSPF (e.g., router ospf 1 - process ID 1).
    - \* Advertise networks with their wildcard masks and specify the area (e.g., network 10.0.1.0 0.0.0.255 area 0). For a simple setup, put everything in Area 0 (backbone).
  - For BGP:
    - \* Note: BGP is typically used between Autonomous Systems (AS). This would require at least two "ASes" in your simulation, each running an IGP (like OSPF or RIP) internally and BGP externally. This is more complex than a simple IGP setup.
    - \* On each router that acts as an AS Border Router (ASBR), enable BGP (e.g., router bgp  $i$ AS\_NUMBER $_i$ ).
    - \* Define neighbors (eBGP peers in different ASes) and advertise networks (neighbor  $i$ IP\_ADDRESS $_i$  remote-as  $i$ REMOTE\_AS\_NUMBER $_i$ ).
    - \* Redistribute routes from the IGP into BGP, or use the network command directly under BGP (carefully).
- Verify Adjacencies/Neighbors: Check if routers form neighbor relationships (e.g., show ip protocols for RIP, show ip ospf neighbor for OSPF, show ip bgp summary for BGP).
- Examine Routing Tables: Use commands like show ip route on each router to observe the routes learned dynamically by the protocol.
- Test Connectivity: From an end device (PC), try to ping a device in a different network.
- Simulate a Link Failure: Disable an interface on one of the links (e.g., shutdown on a router interface).
- Observe Convergence:
  - Watch the routing tables update automatically (show ip route).
  - Re-test connectivity to see if traffic reroutes.
  - Observe the time it takes for the network to converge (traffic to resume flowing along a new path).
- Cleanup: Remove the routing protocol configurations before starting with the next one.

## 11.6 Observation

Not specifically Required

## 11.7 Calculation

Not specifically Required

## 11.8 Result

```
// On R1's CLI
R1>en
R1#conf t
R1(config)#int g0/0
R1(config-if)#ip address 10.0.1.1 255.255.255.0
R1(config-if)#no shut
R1(config-if)#int g0/1
R1(config-if)#ip address 192.168.12.1 255.255.255.252 // Link to R2
R1(config-if)#no shut
R1(config-if)#int g0/2
R1(config-if)#ip address 192.168.13.1 255.255.255.252 // Link to R3
R1(config-if)#no shut
R1(config-if)#exit
```

Figure 1: Initial Configuration Output (Example - Cisco IOS-like syntax)

```
// On R1, R2, R3
R1(config)#router rip
R1(config-router)#version 2 // Use RIPv2 for classless routing
R1(config-router)#network 10.0.1.0 // For directly connected LAN
R1(config-router)#network 192.168.12.0 // For point-to-point links
R1(config-router)#network 192.168.13.0
R1(config-router)#exit
```

Figure 2: Enabling RIP and Observing Output

```
Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2
       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
       ia - IS-IS inter area, * - candidate default, U - per-user static route
       o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

10.0.0.0/24 is subnetted, 2 subnets
C    10.0.1.0 is directly connected, GigabitEthernet0/0
R    10.0.2.0 [120/1] via 192.168.12.2, 00:00:23, GigabitEthernet0/1 // Learned f
    [120/2] via 192.168.13.2, 00:00:15, GigabitEthernet0/2 // Learned
192.168.12.0/30 is subnetted, 1 subnets
C    192.168.12.0 is directly connected, GigabitEthernet0/1
192.168.13.0/30 is subnetted, 1 subnets
C    192.168.13.0 is directly connected, GigabitEthernet0/2
```

Figure 3: Enabling OSPF and Observing Output

```
// On R1, R2, R3
R1(config)#router ospf 1
R1(config-router)#network 10.0.1.0 0.0.0.255 area 0
R1(config-router)#network 192.168.12.0 0.0.0.3 area 0
R1(config-router)#network 192.168.13.0 0.0.0.3 area 0
R1(config-router)#exit
```

Output from `show ip ospf neighbor` (on R1):

| Neighbor ID  | Pri | State   | Dead Time | Address      | Interface         |
|--------------|-----|---------|-----------|--------------|-------------------|
| 192.168.12.2 | 1   | FULL/DR | 00:00:33  | 192.168.12.2 | GigabitEthernet0, |
| 192.168.13.2 | 1   | FULL/DR | 00:00:35  | 192.168.13.2 | GigabitEthernet0, |

Figure 4: Output from `show ip route` (on R1 after convergence)

```
Codes: ... (similar to RIP output, but with O for OSPF) ...

Gateway of last resort is not set

 10.0.0.0/24 is subnetted, 2 subnets
C    10.0.1.0 is directly connected, GigabitEthernet0/0
O    10.0.2.0 [110/2] via 192.168.12.2, 00:00:10, GigabitEthernet0/1 // Learned fr
192.168.12.0/30 is subnetted, 1 subnets
C    192.168.12.0 is directly connected, GigabitEthernet0/1
192.168.13.0/30 is subnetted, 1 subnets
C    192.168.13.0 is directly connected, GigabitEthernet0/2
```

Figure 5: Enabling BGP and Observing Output (More Complex - Conceptual)

```
// On R1 (ASBR for AS 65001), assuming R2 is in AS 65002
R1(config)#router bgp 65001
R1(config-router)#neighbor 192.168.12.2 remote-as 65002
R1(config-router)#network 10.0.1.0 mask 255.255.255.0 // Advertise R1's LAN
R1(config-router)#redistribute ospf 1 // If OSPF is running internally
R1(config-router)#exit
```

Output from `show ip bgp summary` (on R1):

```
BGP router identifier 192.168.12.1, local AS number 65001
BGP table version is 2
Main routing table version 2
0 network entries and 0 path entries at this time
BGP current NDP activity: 0:0:0 active, 0:0:0 passive
Neighbor      V    AS MsgRcvd MsgSent  TblVer  InQ OutQ Up/Down State/PfxRcd
192.168.12.2  4  65002      5      5        2    0    0 00:00:10      0
```

Figure 6: Output from `show ip bgp summary` (on R1)

```
Codes: ... (similar, but with B for BGP) ...

Gateway of last resort is not set

 10.0.0.0/24 is subnetted, 2 subnets
C    10.0.1.0 is directly connected, GigabitEthernet0/0
B    10.0.2.0 [20/0] via 192.168.12.2, 00:00:05 // BGP learned route
...
```

Figure 7: Output from `show ip route` (on R1, showing a BGP learned route)



## **11.9 Do's and Don'ts**

### **11.9.1 Do's**

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### **11.9.2 Don'ts**

- Don't Run Without Internet
- Don't Ignore Errors

## **11.10 References**

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall

## 12 Experiment No. 12: Packet capture and header analysis by wire-shark (TCP, UDP, IP)

### 12.1 Aim

- Understand Network Protocols in Action: Gain practical insight into how the TCP, UDP, and IP protocols function at a low level by observing their headers in live network traffic.
- Identify Key Header Fields: Learn to locate and interpret the significant fields within TCP, UDP, and IP headers (e.g., source/destination IP addresses, port numbers, sequence/acknowledgment numbers, flags, checksums).
- Differentiate Protocol Behaviors: Observe the differences in how TCP (connection-oriented, reliable) and UDP (connectionless, unreliable) handle data transmission by analyzing their respective headers.

### 12.2 Scope

This experiment visually analyzes fundamental TCP, UDP, and IP protocol headers and their network communication roles using Wireshark.

### 12.3 Environmental Condition

Not Specifically Required

### 12.4 Instruments

- A Computer: Any desktop or laptop (Windows, macOS, Linux).
- Wireshark installed (download from [www.wireshark.org](http://www.wireshark.org)).
- Internet Connection

### 12.5 Procedure

- Open Wireshark: Launch the Wireshark application.
- Select an Interface:
  - On the Wireshark home screen, you'll see a list of network interfaces (Ethernet, Wi-Fi, Loopback, etc.).
  - Choose the interface that is currently connected to the internet or your local network (e.g., "Wi-Fi" if you're on a wireless connection, or "Ethernet" if you're wired). You'll usually see activity graphs next to active interfaces.
- Start Capturing:
  - Click on the selected interface.
  - Click the blue "Start capturing packets" button icon (or go to Capture > Start).

- You'll immediately see packets appearing in real-time.
- Generate Network Traffic:
  - While Wireshark is capturing, open your web browser.
  - For TCP:
    - \* Visit a few websites (e.g., google.com, wikipedia.org). This will generate a lot of TCP (HTTP/HTTPS) traffic.
  - For UDP:
    - \* If possible, use an application that relies on UDP (e.g., a simple DNS query, streaming a video from a non-HTTP source, or playing an online game that uses UDP).
    - \* A simple way to generate DNS traffic (which uses UDP) is to clear your DNS cache (if you know how) and then visit a new website.
    - \* Alternatively, open your command prompt/terminal and type `nslookup google.com`
      - this often generates UDP DNS queries.
- Stop Capturing:
  - Go back to Wireshark and click the red "Stop capturing packets" square icon (or Capture & Stop).
- Apply Display Filters:
  - The captured traffic will be overwhelming. Use display filters to focus on the protocols you want to analyze.
  - In the "Apply a display filter..." bar at the top, type:
    - \* `ip` (to see all IP packets)
    - \* `tcp` (to see all TCP packets)
    - \* `udp` (to see all UDP packets)
    - \* You can also combine them, e.g., `tcp or udp`
    - \* Or filter by port: `tcp.port == 80` (HTTP) or `udp.port == 53` (DNS)
- Analyze Headers:
  - Select a packet from the "Packet List" pane (top pane).
  - The "Packet Details" pane (middle pane) will show the decoded layers of the selected packet. Expand the "Internet Protocol Version 4" (for IP), "Transmission Control Protocol" (for TCP), or "User Datagram Protocol" (for UDP) sections.
  - The "Packet Bytes" pane (bottom pane) shows the raw hexadecimal data of the packet.

## 12.6 Observation

Not specifically Required

## 12.7 Calculation

Not specifically Required

## 12.8 Result

```
Frame 1234: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
Ethernet II, Src: Vmware_ab:cd:ef (00:0c:29:ab:cd:ef), Dst: Tp-Link_01:23:45 (00:11:32:01:23:45)
Internet Protocol Version 4, Src: 192.168.1.100, Dst: 172.217.160.100

0100 .... = Version: 4
.... 0101 = Header Length: 20 bytes (5)
Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
Total Length: 52
Identification: 0x48d5 (18645)
Flags: 0x4000, Don't fragment
Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0x7623 [validation successful]
Source Address: 192.168.1.100
Destination Address: 172.217.160.100
[Source GeoIP: Not available]
[Destination GeoIP: Not available]
```

Figure 8: IP

```
Frame 1235: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0
... (Ethernet and IP layers) ...
Transmission Control Protocol, Src Port: 54321, Dst Port: 80, Seq: 0, Ack: 0, Len: 0

Source Port: 54321
Destination Port: 80 (http)
[Stream index: 0]
[TCP Segment Len: 0]
Sequence number: 0 (relative sequence number)
Sequence number (raw): 0x12345678
Acknowledgment number: 0 (relative acknowledgment number)
Acknowledgment number (raw): 0x88888888
0101 ..... = Header Length: 20 bytes (5)
Flags: 0x002 (SYN)
.... 0... = Reserved: Not set
.... ..0. = Nonce: Not set
.... ...0 = Congestion Window Reduced (CWR): Not set
.... ....0 = ECN-Echo: Not set
.... .....0 = Urgent: Not set
.... ....1. = Acknowledgement: Not set
.... ....0 = Push: Not set
.... ....0.. = Reset: Not set
.... ....1. = Syn: Set
.... ....0 = Fin: Not set
Window size value: 65535
[Calculated window size: 65535]
[Window size scaling factor: -1 (no scaling applied)]
Checksum: 0xABCD [validation successful]
[Checksum Status: Good]
Urgent pointer: 0
Options: (4 bytes), Maximum Segment Size
Maximum Segment Size: 1460 bytes
[Timestamps]
[Sequence number: 1 (relative sequence number)]
[Next sequence number: 1 (relative sequence number)]
```

Figure 9: TCP

```
Frame 1236: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on 0
... (Ethernet and IP layers) ...
User Datagram Protocol, Src Port: 54322, Dst Port: 53

Source Port: 54322
Destination Port: 53 (domain)
Length: 28
Checksum: 0x487d [validation successful]
[Checksum Status: Good]
[Stream index: 0]
[Timestamps]
Data (28 bytes)
```

Figure 10: UDP

## 12.9 Do's and Don'ts

### 12.9.1 Do's

- Do Ensure an Internet Connection
- Do Observe the Output
- Do Check for Python Installation
- Do Use a Clean Text Editor

### 12.9.2 Don'ts

- Don't Run Without Internet
- Don't Ignore Errors

## 12.10 References

- Computer Networking- A Top-Down approach (6th edition), Kurose and Ross, Pearson
- Computer Networks- A Top-Down approach, Behrouz Forouzan, McGraw Hill
- Computer Networks (5th edition), Andrew Tanenbaum, Prentice Hall