CAH11-01 Networks and Operating Systems

Internet Layer

Aims of the Seminar

Welcome to the Python Workshop on the Internet Layer! This workshop focuses on the Internet Protocol (IP), the fundamental protocol responsible for addressing and routing packets across networks. By the end of this session, you will:

1. Understand IP's role in the internet layer (addressing,)
2. Build applications that simulate DHCP server

Workshop Outline

1. IP Basics: Working with IP addresses and subnets
2. Classful Addressing
3. Classless Addressing
4. Dynamic Host Configuration Protocol (DHCP)
5. DHCP Dora process

Prerequisites

- Basic Python programming skills
- Understanding of binary numbers and bitwise operations
- Familiarity with networking fundamentals

Feel free to discuss your work with peers, or with any member of the teaching staff.

Reminder

We encourage you to discuss the content of the workshop with the delivery team and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Exercises herein represent an example of what to do; feel free to expand upon this.

Helpful Resources

Python IP address library
https://realpython.com/python-ipaddress-module/
What is CIDR
https://aws.amazon.com/what-is/cidr/
Classful vs classless addressing
https://www.geeksforgeeks.org/classful-vs-classless-addressing/
How DORA Works?
https://www.geeksforgeeks.org/how-dora-works/

Week Recap Quiz

To Access the week quiz via:

Week #05: https://learn.gold.ac.uk/mod/quiz/view.php?id=1621521
Week #06: https://learn.gold.ac.uk/mod/quiz/view.php?id=1623714

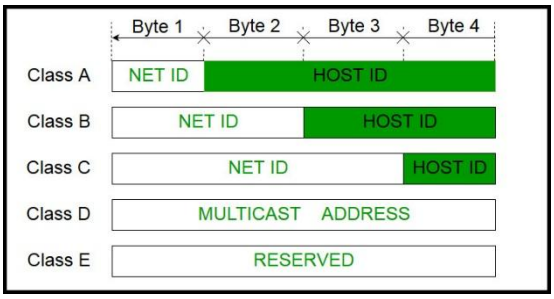
Exercises

You may find it useful to keep track of your answers from workshops in a separate document, especially for any research tasks.
Where questions are asked of you, this is intended to make you think; it would be wise to write down your responses formally.

1. Working with IP Addresses

a. Classful Addressing

IPv4 addresses are divided into five classes (A, B, C, D, and E) based on their first octet (the first 8 bits of the address). These classes determine the structure of the address, including how many bits are used for the network and host portions. Here's a breakdown:



Class	Format	Address Range
A	N.H.H.H	1.0.0.0 to 126.0.0.0
B	N.N.H.H	128.0.0.0 to 191.255.0.0
C	N.N.N.H	192.0.0.0 to 223.255.255.0
D	Multicast	224.0.0.0 to 239.255.255.255
E	Experimental	240.0.0.0. to 254.255.255.255

b. Classless Inter-Domain Routing (CIDR)

Classless Inter-Domain Routing (CIDR) is an IP addressing method that replaces the traditional classful system (Class A, B, C) to provide more efficient allocation of IP addresses. CIDR was introduced in 1993 (RFC 1519) to overcome the limitations of class-based addressing and to slow down the depletion of IPv4 addresses.

Instead of classful addresses, CIDR uses a **network prefix** followed by a **slash (/) and the number of network bits**.

- **Example:**
  - **192.168.1.0/24** → This means the first **24 bits** are the network portion, and the remaining **8 bits** are for hosts.
  - **10.0.0.0/16** → The first **16 bits** are for the network, and the remaining **16 bits** are for hosts.

Subnet Mask Representation
Each **CIDR prefix** corresponds to a **subnet mask**, which helps define how many bits are reserved for the network and how many for hosts.

CIDR Prefix	Subnet Mask	Number of Hosts
/8	255.0.0.0	16,777,214
/16	255.255.0.0	65,534
/24	255.255.255.0	254
/30	255.255.255.252	2

Comparison between Classful and Classless Addressing:

Feature	Classful Addressing	CIDR (Classless)
Address Allocation	Fixed (A, B, C)	Flexible (Custom)
Routing Table Size	Large	Small (Aggregated Routes)
Efficient Use of IPs	No (Wastes IPs)	Yes (Only Allocates Needed)
Scalability	Poor	Excellent

Objective: Create tools to manipulate and analyse IP addresses.

Code Example:

```
import ipaddress

def analyse_ip(ip_str):
    # Create an IP interface object
    ip = ipaddress.ip_interface(ip_str)

    print(f"Address: {ip.ip}")
    print(f"Network: {ip.network}")
    print(f"Netmask: {ip.netmask}")
    print(f"Is private: {ip.ip.is_private}")
    print(f"Is global: {ip.ip.is_global}")

    # List all hosts in the network
    if ip.network.num_addresses < 256:  # Only for small networks
        print("\nHosts in network:")
        for host in ip.network.hosts():
            print(host)

# Example usage
analyse_ip('192.168.1.1')
```

Exercise 1: Extend the script to calculate:

- Broadcast address
- First and last usable host addresses
- Number of usable hosts
- Change the IP address to have it CIDR prefix (e.g. /24) and compare resulting networks

To find your device's IP address, first identify its name, then retrieve the address by:

```
import socket
hostname = socket.gethostname()
IPAddr = socket.gethostbyname(hostname)

print("Your Computer Name is:" + hostname)
```

```
print("Your Computer IP Address is:" + IPAddr)
```

Exercise 2: Analyse your IP address to know if it is private/public and the network details.

Exercise 3: Get the university website IP address and analyse it. Hint you can use seminar of week2 to know how to get the address.

Exercise 4: (Challenge 🤔) Create a subnetting plan for a company with 4 departments requiring the following hosts:

- Engineering: 30 hosts
- Marketing: 15 hosts
- Finance: 10 hosts
- HR: 5 hosts

Use the network address 172.16.0.0/16 and determine appropriate subnet masks for each department.

2. Dynamic Host Configuration Protocol (DHCP)

DHCP is a network management protocol used to automatically assign IP addresses and other network configuration parameters to devices on a network.

a. DHCP Operation (4-Step Process)

1. **DHCP Discover (DHCPDISCOVER)**
   - Client broadcasts a request for an IP address
   - Source: 0.0.0.0, Destination: 255.255.255.255
   - Contains client's MAC address for identification
2. **DHCP Offer (DHCPOFFER)**
   - Server responds with an available IP address
   - Includes lease time and other configuration information
   - Multiple DHCP servers can respond with different offers
3. **DHCP Request (DHCPREQUEST)**
   - Client broadcasts which offer it accepts
   - This informs all DHCP servers, allowing non-selected servers to reclaim offered IP addresses
4. **DHCP Acknowledgment (DHCPACK)**
   - Selected server confirms the assignment
   - Finalizes lease duration and provides complete configuration

b. Simulate DHCP operations

The following script simulate the process of DHCP.

```
# dhcp_simple.py - Simplified DHCP Simulator

# Server Configuration
server = {
    "ip_pool": ["192.168.1.100", "192.168.1.101", "192.168.1.102"],
    "leases": {}
}

# Client Configuration
client = {
    "mac": "AA:BB:CC:DD:EE:FF",
    "ip": None
}

def send_discover():
    print("\n[CLIENT] Step 1: Sending DHCP DISCOVER")
    return {
        "type": "DISCOVER",
        "mac": client["mac"]
    }

def make_offer(discover):
    print("\n[SERVER] Step 2: Making DHCP OFFER")
    if not server["ip_pool"]:
        print("No IPs available!")
        return None

    offered_ip = server["ip_pool"].pop(0)
    return {
        "type": "OFFER",
        "mac": discover["mac"],
        "ip": offered_ip
    }

def send_request(offer):
    print("\n[CLIENT] Step 3: Sending DHCP REQUEST")
    return {
        "type": "REQUEST",
        "mac": offer["mac"],
        "ip": offer["ip"]
    }
```

```python
def send_ack(request):
    print("\n[SERVER] Step 4: Sending DHCP ACK")
    server["leases"][request["mac"]] = request["ip"]
    return {
        "type": "ACK",
        "mac": request["mac"],
        "ip": request["ip"]
    }

def main():
    print("=== Simple DHCP Simulation ===")

    # Client starts process
    discover = send_discover()

    # Server responds
    offer = make_offer(discover)
    if not offer:
        return

    # Client continues
    request = send_request(offer)

    # Server finalizes
    ack = send_ack(request)

    # Update client IP
    client["ip"] = ack["ip"]

    print("\n=== Result ===")
    print(f"Client {client['mac']} got IP: {client['ip']}")
    print("Server leases:", server["leases"])

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

run it and trace the process steps.

Exercise 5: (Challenge 😳) use the client sever example from previous week to have a TCP server as DHCP server and create a client that would get the IP address from the server following the process above.

Happy coding! 🚀