EE315 – 数据通信与网络期末项目

张怡程 Zhang Yicheng 12210714

2024-12-28

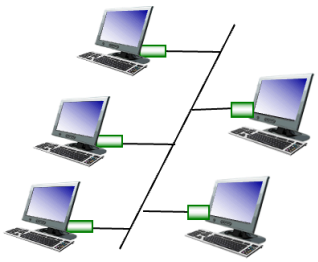
---

1. 项目简介

使用 Python 实现了简单的局域网通信，使用总线结构和星型结构。在基础功能中，实现了广播、舍弃含错误地址的包、MAC 地址学习、选择性转发等功能。在附加功能中，实现了 VLAN、加密解密、调制解调功能。保持了全系统的相互兼容，且测试方式清晰规范。

2. 内容

2.1. 总线结构



bus: coaxial cable

在总线结构中，所有设备通过一条共享的总线连接。某个设备发送数据时，所有总线上的设备都会接收到这个信号，但是只有目标设备会对这个数据做出相应。这一结构逻辑简单。

发送主机直接将信号广播：

```
1 def send_packet(self, dst_mac, payload, bus):
2     packet = Packet(src=self.mac, dst=dst_mac, payload=payload)
3     bus.broadcast(packet)
4     pass
```

广播函数如下。注意到广播排除了本机 MAC。

```
1 def broadcast(self, packet):
2     self.log_event(f"Broadcasting packet: {packet}")
3     for host in self.hosts:
4         if host.mac != packet.src:
5             host.receive_packet(packet)
```

总线上所有主机尝试读取数据。只有目标 MAC 的主机会将数据载入内存。

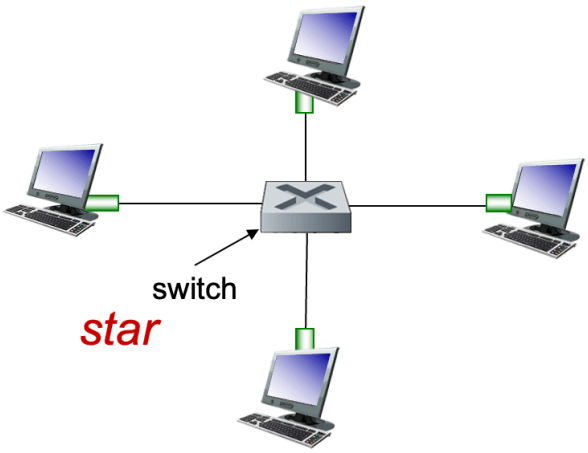
```
1 def receive_packet(self, packet):
2     if packet.dst == self.mac:
```

```
3         self.buffer.append(packet)
4     pass
```

经测试，达成目标。

```
Testing Broadcast Packet (20 points)...
✓ Broadcast Packet test passed

Testing Discard Unaddressed Packet (20 points)...
✓ Discard Unaddressed Packet test passed

Total Score: 40/40
```

2.2. 星型结构



在星型结构中，所有设备直接连接到中心节点（在本实验中为交换机）。发送主机将数据发送到交换机，然后交换机将数据转发给目标设备。

发送主机让交换机处理数据包：

```
1 def send_packet(self, dst_mac, payload, switch):
2     packet = Packet(src=self.mac, dst=dst_mac, payload=payload)
3     switch.handle_packet(packet)
```

交换机如下处理数据包：

```
1 def handle_packet(self, packet):
2     # 1. MAC Learning
3     # interface -> host mapping
4 
5     for interface, host in self.interfaces.items():
6         if host and host.mac == packet.src:
7             self.mac_table[packet.src] = interface
8             break
9 
10     dst_interface = self.mac_table.get(packet.dst)
11     if dst_interface is not None:
```

```
12          self.fabric.forward_to_interface(packet, dst_interface)
13      else:
14          # Broadcast
15          for interface, host in self.interfaces.items():
16              if host and host.mac != packet.src:  # Don't send to self
17                  self.fabric.forward_to_interface(packet, interface)
```

- 其中，`Switch.inferfaces` 维护了一个接口-MAC 地址的映射表。`self.mac_table[packet.src] = interface` 用来更新这个映射关系，称为“MAC 学习”。
- 若 MAC 表中没有发现目标主机，则进行泛洪。

目标主机接受过程同总线结构中的过程。

经测试，达成目标。

```
Testing Initial Flooding (20 points)...
✓ Initial Flooding test passed

Testing MAC Learning (20 points)...
✓ MAC Learning test passed

Testing Selective Forwarding (20 points)...
✓ Selective Forwarding test passed

Total Score: 60/60
```

2.3. 附加内容 1：VLAN

VLAN（虚拟局域网）在逻辑上分割网络，将不同设备划分到不同的虚拟网络中。可以提高网络的安全性和效率。

在 `Switch` 中维护一个 `interface_vlan_table` 字段来储存端口和 VLAN 的映射关系。

在将主机接入网络的时候，配置其 VLAN：

```
1 def configure_interface_vlan(self, interface, vlan_id):
2     self.interface_vlan_table[interface] = vlan_id
```

发送主机同前进行发送。在交换机处，判断发送主机和目标主机是否处于同一 VLAN，是则进行转发。

```
1 if src_vlan == dst_vlan:
2     self.fabric.forward_to_interface(packet, dst_interface)
```

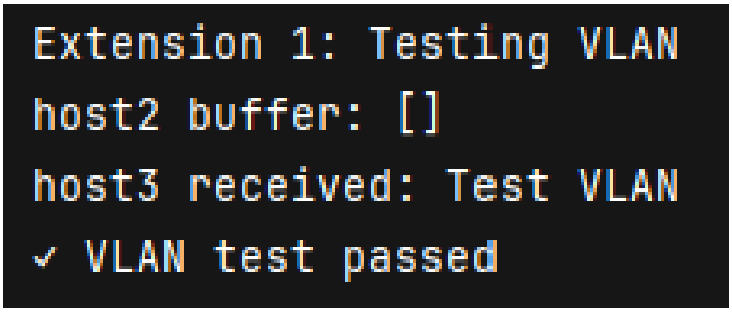
以下为测试代码：

```
1 print("\n\nExtension 1: Testing VLAN")
2
3 host1 = Host("00:00:00:00:00:01", 0)
```

```
host2 = Host("00:00:00:00:00:02", 1)
host3 = Host("00:00:00:00:00:03", 2)

switch.configure_interface_vlan(0, 10)  # host1 on VLAN 10
switch.configure_interface_vlan(1, 20)  # host2 on VLAN 20
switch.configure_interface_vlan(2, 10)  # host3 on VLAN 10

host1.buffer = []
host2.buffer = []
host3.buffer = []

try:
    open(fabric.log_file, 'w').close()  # Clear log
    host1.send_packet("00:00:00:00:00:02", "Test VLAN", switch)
    print(f"host2 buffer: {host2.buffer}")
    host1.send_packet("00:00:00:00:00:03", "Test VLAN", switch)
    print(f"host3 received: {host3.buffer[0].payload}")
    with open(fabric.log_file, 'r') as f:
        log_content = f.read()
        assert "Packet forwarded - Interface: 1" not in log_content, "Packet
incorrectly forwarded to different VLAN"
        assert "Packet forwarded - Interface: 2" in log_content, "Packet
failed to forwarded to same VLAN"
    print("✓ VLAN test passed")
except AssertionError as e:
    print(f"✗ VLAN test failed: {str(e)}")
```

在此测试中，`host1`、`host3` 处于同一个 VLAN 中（`VLAN 10`），`host2` 不在此 VLAN 中（`VLAN 20`）。那么，应当预期 `host1` 可以向 `host3` 发送信息，而不能向 `host2` 发送信息。

测试结果成功。



2.4. 附加内容 2：加密、解密

加密、解密用语保护数据安全。加密将原始数据转换为不可读的密文，解密反之。

采用 Fernet 加密协议，使用 128 为 AES 对称加密算法进行加密解密。使用时，可自行指定密钥种子生成密钥。

```
def generate_key(string_key):
    padded_key = string_key.ljust(32)[:32]
    return base64.urlsafe_b64encode(padded_key.encode())
```

`padded_key` 将输入的字符串种子调整为 32 字符大小以符合 Fernet 协议要求。随后，对其进行 Base64 编码令其可在 URL 中安全传输。

```
Class Host:
    ...
    def update_key(self, key):
        self.key = key
        self.cipher = Fernet(self.key)
```

将生成的 Base64 编码传入 `update_key()`中设置主机持有的密钥和解密对象。

发送方如下进行加密：

```
payload = self.cipher.encrypt(payload.encode()).decode()
```

为保证前向兼容，在 `Packet` 类中添加 `is_encrypted` 字段表征其是否被加密，方便接收方决定是否进行解密。该字段默认为 `False`。

```
def __init__(self, src, dst, payload, is_encrypted=False):
    ...
    self.is_encrypted = is_encrypted
```

接收方如下进行解密：

```
packet.payload = self.cipher.decrypt(packet.payload.encode()).decode()
```

以下为测试代码：

```
print("\n\nExtension 2: Testing Encryption")
host1.buffer = []
host2.buffer = []
host3.buffer = []
key1 = generate_key("114514")
key2 = generate_key("1919810")
host1.update_key(key1)
host2.update_key(key1)
host3.update_key(key2)

host1.send_packet("00:00:00:00:00:02", "Correct Encryption Key", switch,
encrypt=True)

print(f"host2 buffer: {host2.buffer[0].payload}")
try:
    host1.send_packet("00:00:00:00:00:03", "Wrong Encryption Key", switch,
encrypt=True)
except Exception as e:
    print(f"Encryption failed as expected: {e}")
print(f'host3 buffer: {host3.buffer}')
print("Failure as expected")
```

`host1`、`host2` 持有相同密钥（种子为 `114514`）。`host3` 持有的密钥不同（种子为 `1919810`）。应当期待 `host2` 可以正常解码恢复 `host1` 发送的信息，而 `host3` 无法成功。

```
Extension 2: Testing Encryption
host2 buffer: Correct Encryption Key
Failed to decrypt packet.
host3 buffer: []
Failure as expected
✓ Encryption test passed
```

测试结果成功。host3 按预期在解码过程中抛出异常。

2.5. 附加功能 3：调制解调

调制将原始数据转换为特定格式令其适合传输；解调用于将这种信号还原为原始信号。

采用幅度调制，将信号转换为二进制 ASCII 编码调制为高幅度（1）和低幅度（0）。

```
class Modem:
    @staticmethod
    def modulate(data):
        return ''.join(format(ord(char), '08b') for char in data)

    @staticmethod
    def demodulate(signal):
        return ''.join(chr(int(signal[i:i + 8], 2)) for i in range(0,
len(signal), 8))
```

`modulate()`将字符串中的每个字符转换为 8 位二进制，随后拼接为二进制串。`demodulate()`反之。

发送方如此调制：

```
payload = Modem.modulate(payload)
```

类似加密解密，在 `Packet` 中添加 `is_modulated` 字段。接收方如此解调：

```
packet.payload = Modem.demodulate(packet.payload)
```

测试代码如下：

```
print("\n\nExtension 3: Testing Modulation")
host1.buffer = []
host2.buffer = []

host1.send_packet("00:00:00:00:00:02", "Encrypted and modulated signal",
switch, encrypt=True, modulate=True)
print(f"host2 buffer: {host2.buffer[0].payload}")
assert host2.buffer[0].payload == "Encrypted and modulated signal",
"Demodulation failed"

print("✓ Modulation test passed")
```

```
Extension 3: Testing Modulation
Modulated payload: 011001110100000101000001010000010100
host2 buffer: Encrypted and modulated signal
✓ Modulation test passed
```

测试结果成功。

2.6. 其他：代码兼容性、评估

本项目附加代码之间相互兼容，也与基础内容代码兼容。主要原因是增加字段时设置了默认值，比如。

```python
class Packet:
    def __init__(self, src, dst, payload, is_encrypted=False,
is_modulated=False):
        ...
        # Extension: Encryption
        self.is_encrypted = is_encrypted
        # Extension: Modulation
        self.is_modulated = is_modulated
```

```
Testing Initial Flooding (20 points)...
✓ Initial Flooding test passed

Testing MAC Learning (20 points)...
✓ MAC Learning test passed

Testing Selective Forwarding (20 points)...
host2.buffer[0].payload: Test flooding
✓ Selective Forwarding test passed

Total Score: 60/60

Testing Extensions


Extension 1: Testing VLAN
host2 buffer: []
host3 received: Test VLAN
✓ VLAN test passed


Extension 2: Testing Encryption
host2 buffer: Correct Encryption Key
Failed to decrypt packet.
host3 buffer: []
Failure as expected
✓ Encryption test passed


Extension 3: Testing Modulation
Modulated payload: 011001110100000101000001010000
host2 buffer: Encrypted and modulated signal
✓ Modulation test passed
```

在附加内容测试代码中，使用 `assert` 评估 `buffer`、 `log_file` 来准确评估代码是否正确运行。