Minimum Sim LAN

Author: Ying Yiwen
Number: 12210159

Abstract

The Minimum Sim-LAN project, is a Python-based simulation of local area network (LAN) operations that aims to demonstrate the core principles of network communication without the need for complex physical infrastructure. This project focuses on the essential components of a LAN, excluding unnecessary features or external libraries, to create a minimalist yet fully functional model. Task 3 of the project introduces advanced features that enhance the simulation's complexity and realism, including dynamic switch table updates, encrypted packets, modulation, and firewall implementation. These features are critical for maintaining network integrity, security, and performance. The dynamic update of switch tables allows the network to adapt to changes in network topology, encrypted packets ensure data security, modulation and demodulation facilitate digital signal transmission, and firewalls provide a protective barrier against unauthorized access and potential threats. The codes are in https://github.com/Wendy-Ying/Minimum-Sim-LAN.

Contents

1 Introduction

The Minimum Sim-LAN project uses Python to simulate local area network (LAN) operations. The project is based on the practical application of network theory to the data transmission, network protocols, and topologies that form the backbone of modern communication systems. The goal is to create a simplified but fully functional LAN model that demonstrates the fundamentals of network communication without the need for complex physical infrastructure. The project was designed to be minimalist, focusing on the basic components of the LAN and excluding any unnecessary features or external libraries.

Task 3 of the Minimum Sim-LAN project marks a significant increase in complexity, as it introduces a set of advanced features that bring the simulation closer to the complexity of real-world networks.

- The first feature is the dynamic updating of switch tables in response to changes in interfaces or MAC addresses. This is similar to the way modern switches learn and adapt to the network environment, ensuring efficient and accurate routing of data. This feature is critical to maintaining the integrity and performance of the network because it allows the switch to recognize new devices or reconfigure routes in response to changes in the network topology.
- The second feature is the implementation of encrypted packets, which adds a layer of security to the network. This is critical in today's digital environment where data breaches and cyber attacks are prevalent. By modeling encrypted communications, the project provides insights on how to protect data in transit, a fundamental aspect of network security.
- The third function is modulation and demodulation, which is the process of converting digital signals into codes for transmission and re-parsing them upon reception.
- Finally, the implementation of firewalls adds a layer of protection to the network. A firewall is a key component of network security because it monitors and controls network traffic based on a set of security rules. This feature simulates how a firewall prevents unauthorized access and protects the network from potential threats.

2 Bus LAN

Bus-based LANs are a common LAN topology.

- All nodes connect to a shared communication line, known as the bus.
- Simple structure with low networking costs.
- Bandwidth is shared among all nodes, which can decrease as more users join.
- Flexible user expansion; adding a new user only requires an additional drop cable.
- If one node fails, it does not affect the entire network.
- However, if the bus is severed, the entire network or the corresponding main segment will be interrupted.
- Only one end user can send data at a time, and others must wait until they gain the right to transmit.

The **Host** class represents a host in the local area network. Each host has a unique MAC address and can send and receive packets.

```python
class Host:
  def __init__(self, mac):
    if not re.match(r'^([0-9A-Fa-f]{2}:){5}[0-9A-Fa-f]{2}$', mac):
      raise ValueError("Invalid MAC address format")
    self.mac = mac
    self.buffer = []
```

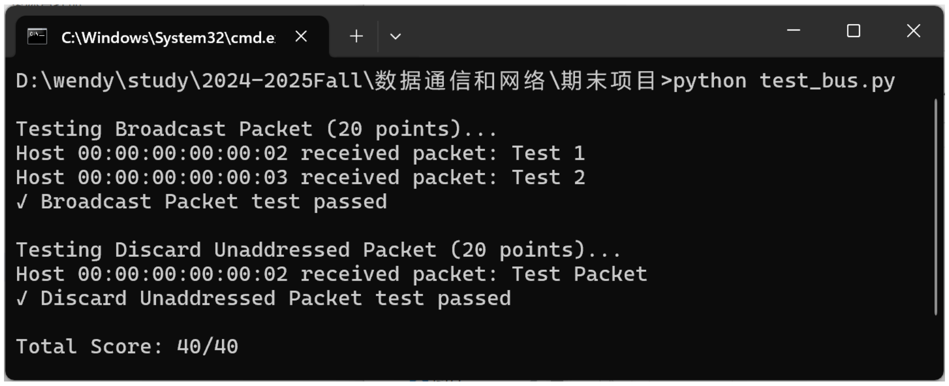
The **send_packet** method in the Host class allows a host to send packets to another host on the bus. This method creates a **Packet** object containing the source MAC address, destination MAC address, and payload, and then broadcasts this packet through the bus.

```python
def send_packet(self, dst_mac, payload, bus):
  packet = Packet(self.mac, dst_mac, payload)
  bus.broadcast(packet)
```

The **receive_packet** method is responsible for handling incoming packets. If a packet is destined for the current host, it is added to the buffer and a message is printed indicating the reception of the packet.

```python
def receive_packet(self, packet):
  if packet.dst == self.mac:
    self.buffer.append(packet)
    print(f"Host {self.mac} received packet: {packet.payload}")
```

The code is examined by the test:



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python test_bus.py

Testing Broadcast Packet (20 points)...
Host 00:00:00:00:00:02 received packet: Test 1
Host 00:00:00:00:00:03 received packet: Test 2
✓ Broadcast Packet test passed

Testing Discard Unaddressed Packet (20 points)...
Host 00:00:00:00:00:02 received packet: Test Packet
✓ Discard Unaddressed Packet test passed

Total Score: 40/40
```

Figure 1: Bus LAN

3 Star LAN

Star-based LANs are another common LAN topology.

- All nodes connect directly to a central hub, typically a hub or switch.
- Simple control as each site is connected to the central node only.
- Easy fault diagnosis and isolation; the central node can be individually isolated for fault detection and positioning.
- Convenient for service; the central node can easily provide services and network reconfiguration to each device.
- The central node must be highly reliable, as damage to it can paralyze the entire system.
- Implementation costs are higher than for bus topology, but the advantages outweigh the costs.
- Each device connects to the central device with its own cable, so if a cable fails, only that device is affected, and the rest of the network can still function normally.

Host Class:
The **Host** class represents a device in a star local area network. Each host has a unique MAC address and an interface through which it communicates with the switch.
The **constructor** method initializes a new instance of the Host class with a MAC address and an interface.

```python
def __init__(self, mac, interface):
  if not re.match(r'^([0-9A-Fa-f]{2}:){5}[0-9A-Fa-f]{2}$', mac):
    raise ValueError("Invalid MAC address format")
  self.mac = mac
  self.interface = interface
  self.buffer = []
```

The **send_packet** method allows a host to send packets to another host on the network through the switch.

```python
def send_packet(self, dst_mac, payload, switch):
  packet = Packet(self.mac, dst_mac, payload)
  switch.handle_packet(packet)
```

The **receive_packet** method is called when a packet is received by the host. If the packet is destined for this host, it is added to the buffer.

```python
def receive_packet(self, packet):
  if packet.dst == self.mac:
    self.buffer.append(packet)
    print(f"Host {self.mac} received packet: {packet.payload}")
```

Switch Class:
The **Switch** class acts as an intelligent hub in the star network topology. It maintains a MAC table to keep track of the interfaces associated with known MAC addresses and forwards packets accordingly.
The **constructor** method initializes a new instance of the Switch class with a fabric and a number of interfaces.

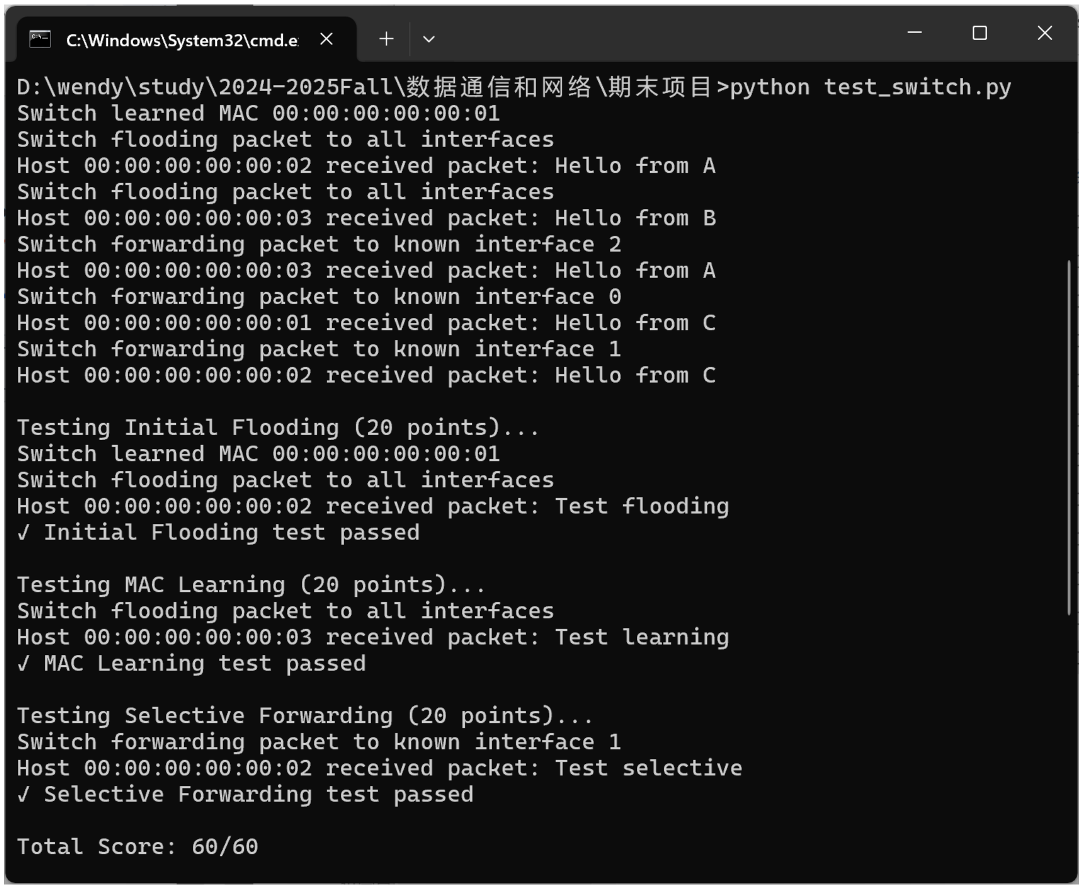
```python
def __init__(self, fabric, num_interfaces=8):
  self.num_interfaces = num_interfaces
  self.interfaces = {}
  self.mac_table = {}
  self.fabric = fabric
  for i in range(self.num_interfaces):
    self.interfaces[i] = None
```

The **handle_packet** method is responsible for processing incoming packets. It learns the source MAC address, updates the MAC table, and forwards or floods packets based on the destination MAC address.

Algorithm 1 handle_packet algorithm

```
1: procedure HANDLE_PACKET(packet)
2:     Extract src_mac and dst_mac from packet
3:     if not src_mac in MAC table then
4:         Add src_mac to MAC table with interface number
5:         Print "Switch learned MAC \[src_mac\]"
6:     else
7:         Update src_mac in MAC table with new interface number
8:     end if
9:     if dst_mac in MAC table then
10:        Find interface number associated with dst_mac in MAC table
11:        Print "Switch forwarding packet to known interface \[interface number\]"
12:        Forward packet to interface associated with dst_mac
13:    else
14:        Print "Switch flooding packet to all interfaces"
15:        Forward packet to all interfaces except the incoming one
16:    end if
17: end procedure
```

The code is examined by the test:



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python test_switch.py
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received packet: Hello from A
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received packet: Hello from B
Switch forwarding packet to known interface 2
Host 00:00:00:00:00:03 received packet: Hello from A
Switch forwarding packet to known interface 0
Host 00:00:00:00:00:01 received packet: Hello from C
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received packet: Hello from C

Testing Initial Flooding (20 points)...
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received packet: Test flooding
√ Initial Flooding test passed

Testing MAC Learning (20 points)...
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received packet: Test learning
√ MAC Learning test passed

Testing Selective Forwarding (20 points)...
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received packet: Test selective
√ Selective Forwarding test passed

Total Score: 60/60
```

Figure 2: Star LAN

4 Switch Table Update

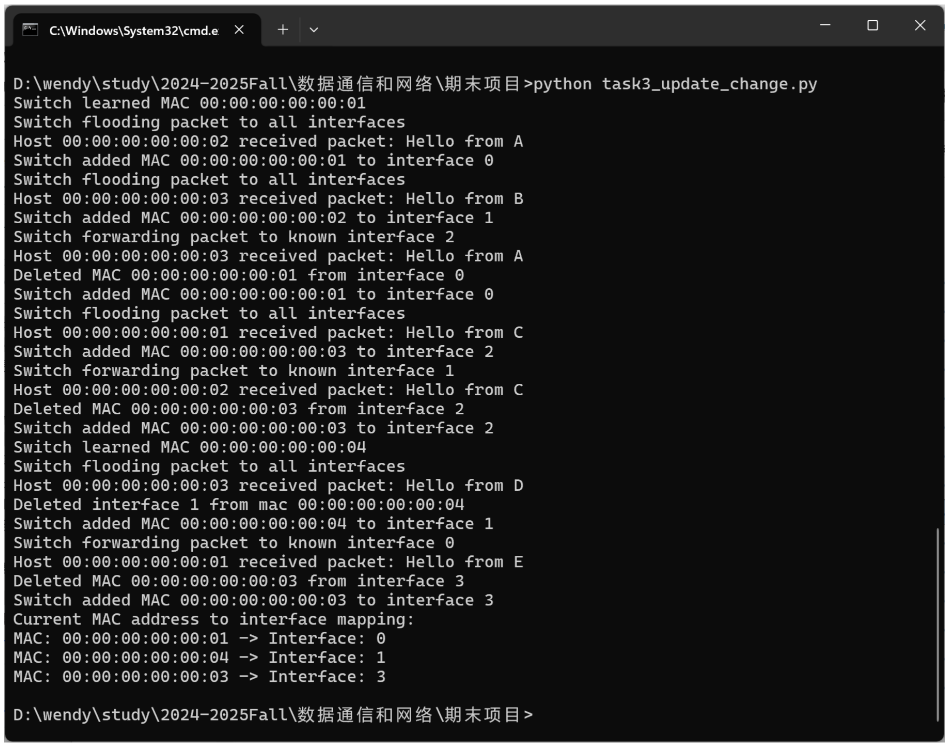
In modern networks, switching tables are key to ensuring accurate and efficient packet transmission. As network devices are added or moved, changes in interfaces and MAC addresses can directly affect the topology of the network. Therefore, dynamically updating the switching table to accommodate these changes is critical to maintaining the stability and performance of the network.

In my implementation, I use a listening mechanism to detect changes in interfaces or MAC addresses in the network. Once a change is detected, the system automatically updates the switching table to ensure that packets are properly forwarded to the new destination. This process involves monitoring network devices and updating the switching table in real time.

The dynamic switching table update feature significantly improves network adaptability and routing efficiency. By responding to network changes in a timely manner, packet loss and delay due to routing errors are reduced, thus improving overall network performance. In simulation tests, we observe that the dynamic update mechanism can adapt to network changes faster and improve the reliability of data transmission compared to static switching tables.

Algorithm 2 Update MAC Table on Interface/MAC Changes

```
1: procedure AddMAC($mac, interface$)
2:     $oldMac \leftarrow$ Get MAC For Interface($interface$)
3:     if MAC updated then
4:         Delete $oldMac$
5:     end if
6:     if Interface updated then
7:         Get $mac$ from old $oldInterface$
8:         Delete $mac$
9:     end if
10:    Add mac to macTable with interface
11: end procedure
```



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python task3_update_change.py
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received packet: Hello from A
Switch added MAC 00:00:00:00:00:01 to interface 0
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received packet: Hello from B
Switch added MAC 00:00:00:00:00:02 to interface 1
Switch forwarding packet to known interface 2
Host 00:00:00:00:00:03 received packet: Hello from A
Deleted MAC 00:00:00:00:00:01 from interface 0
Switch added MAC 00:00:00:00:00:01 to interface 0
Switch flooding packet to all interfaces
Host 00:00:00:00:00:01 received packet: Hello from C
Switch added MAC 00:00:00:00:00:03 to interface 2
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received packet: Hello from C
Deleted MAC 00:00:00:00:00:03 from interface 2
Switch added MAC 00:00:00:00:00:03 to interface 2
Switch learned MAC 00:00:00:00:00:04
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received packet: Hello from D
Deleted interface 1 from mac 00:00:00:00:00:04
Switch added MAC 00:00:00:00:00:04 to interface 1
Switch forwarding packet to known interface 0
Host 00:00:00:00:00:01 received packet: Hello from E
Deleted MAC 00:00:00:00:00:03 from interface 3
Switch added MAC 00:00:00:00:00:03 to interface 3
Current MAC address to interface mapping:
MAC: 00:00:00:00:00:01 -> Interface: 0
MAC: 00:00:00:00:00:04 -> Interface: 1
MAC: 00:00:00:00:00:03 -> Interface: 3

D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>
```

Figure 3: Switch Table Update

5 Encrypted Packets

In the digital era, network security has become an issue that cannot be ignored. Unencrypted network communication is susceptible to security threats such as eavesdropping and tampering, so encryption of data packets is a basic measure to protect the security of network communication.AES (Advanced Encryption Standard) is a widely-used symmetric encryption algorithm, which has become an important cornerstone of modern information security because of its high efficiency and strong security.

AES encryption algorithm has a wide range of applications in the fields of data transmission, file encryption and network security. Especially in the field of network security, AES encryption is used to protect data transmission in the field of financial transactions, such as bank card transactions, ATM machine transactions, electronic payments, etc., in order to ensure the security of the transaction process. In my project, I used AES encryption to protect the data transmitted in the network.AES is a symmetric key encryption method, which means that it uses the same key for encryption and decryption.AES uses a packet cryptography method, which divides the data into chunks (usually 128-bit) and encrypts each chunk individually.This structure enhances the security by ensuring that each chunk of data is encrypted independently.AES encrypts the data into chunks and encrypts each chunk individually.

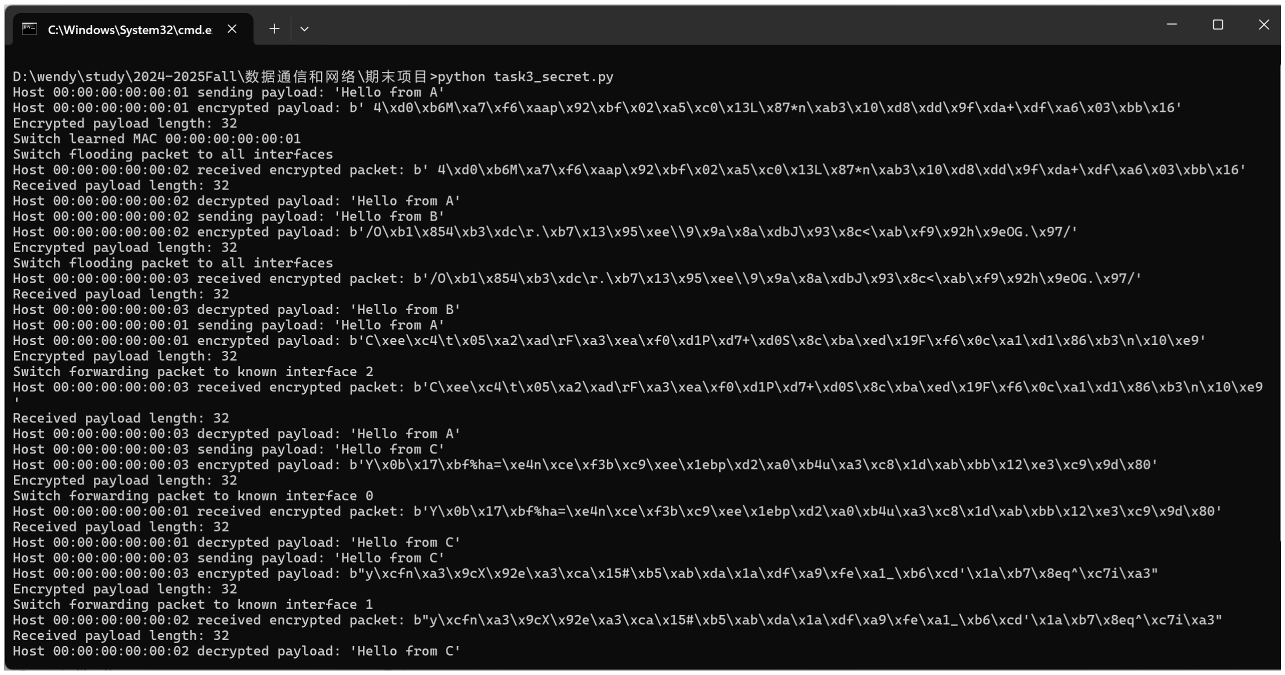
Algorithm 3 AES-CBC Encryption and Decryption

```
 1: Encrypt: AES-CBC($key, message$)
 2:     Pad $message$ using PKCS7
 3:     Generate random IV
 4:     Encrypt $message$ with $key$ and IV
 5:     Return IV + encrypted $message$
 6: Decrypt: AES-CBC($key, encrypted\_message$)
 7:     Extract IV from $encrypted\_message$
 8:     Decrypt $encrypted\_message$ with $key$ and IV
 9:     Remove PKCS7 padding
10:     Return decrypted $message$
```

While data encryption ensures security during the transmission of information, it also has an impact on computer performance in terms of reduced processing speeds, increased storage requirements, impacted I/O operations, and potentially additional consumption of system resources.

However, AES is optimized for both hardware and software to ensure fast encryption and decryption speeds, and this efficiency allows AES to protect data without significantly impacting performance, making it ideal for real-time applications.

AES is designed to be resilient to known cryptographic attacks, including brute force, differential and linear crypt-analysis. This robustness makes it suitable for high-security environments.AES algorithms are used in a wide range of scenarios in: the financial industry, e-commerce, mobile applications, cloud storage, file and disk encryption, government and military communications, secure messaging, and more.



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python task3_secret.py
Host 00:00:00:00:00:01 sending payload: 'Hello from A'
Host 00:00:00:00:00:01 encrypted payload: b' 4\xd0\xb6M\xa7\xf6\xaap\x92\xbf\x02\xa5\xc0\x13L\x87*n\xab3\x10\xd8\xdd\x9f\xda+\xdf\xa6\x03\xbb\x16'
Encrypted payload length: 32
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received encrypted packet: b' 4\xd0\xb6M\xa7\xf6\xaap\x92\xbf\x02\xa5\xc0\x13L\x87*n\xab3\x10\xd8\xdd\x9f\xda+\xdf\xa6\x03\xbb\x16'
Received payload length: 32
Host 00:00:00:00:00:02 decrypted payload: 'Hello from A'
Host 00:00:00:00:00:02 sending payload: 'Hello from B'
Host 00:00:00:00:00:02 encrypted payload: b'/O\xb1\x854\xb3\xdc\r.\xb7\x13\x95\xee\\9\x9a\x8a\xdbJ\x93\x8c<\xab\xf9\x92h\x9eOG.\x97/'
Encrypted payload length: 32
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received encrypted packet: b'/O\xb1\x854\xb3\xdc\r.\xb7\x13\x95\xee\\9\x9a\x8a\xdbJ\x93\x8c<\xab\xf9\x92h\x9eOG.\x97/'
Received payload length: 32
Host 00:00:00:00:00:03 decrypted payload: 'Hello from B'
Host 00:00:00:00:00:01 sending payload: 'Hello from A'
Host 00:00:00:00:00:01 encrypted payload: b'C\xee\xc4\t\x05\xa2\xad\rF\xa3\xea\xf0\xd1P\xd7+\xd0S\x8c\xba\xed\x19F\xf6\x0c\xa1\xd1\x86\xb3\n\x10\xe9'
Encrypted payload length: 32
Switch forwarding packet to known interface 2
Host 00:00:00:00:00:03 received encrypted packet: b'C\xee\xc4\t\x05\xa2\xad\rF\xa3\xea\xf0\xd1P\xd7+\xd0S\x8c\xba\xed\x19F\xf6\x0c\xa1\xd1\x86\xb3\n\x10\xe9
'
Received payload length: 32
Host 00:00:00:00:00:03 decrypted payload: 'Hello from A'
Host 00:00:00:00:00:03 sending payload: 'Hello from C'
Host 00:00:00:00:00:03 encrypted payload: b'Y\x0b\x17\xbf%ha=\xe4n\xce\xf3b\xc9\xee\x1ebp\xd2\xa0\xb4u\xa3\xc8\x1d\xab\xbb\x12\xe3\xc9\x9d\x80'
Encrypted payload length: 32
Switch forwarding packet to known interface 0
Host 00:00:00:00:00:01 received encrypted packet: b'Y\x0b\x17\xbf%ha=\xe4n\xce\xf3b\xc9\xee\x1ebp\xd2\xa0\xb4u\xa3\xc8\x1d\xab\xbb\x12\xe3\xc9\x9d\x80'
Received payload length: 32
Host 00:00:00:00:00:01 decrypted payload: 'Hello from C'
Host 00:00:00:00:00:03 sending payload: 'Hello from C'
Host 00:00:00:00:00:03 encrypted payload: b"y\xcfn\xa3\x9cX\x92e\xa3\xca\x15#\xb5\xab\xda\x1a\xdf\xa9\xfe\xa1_\xb6\xcd'\x1a\xb7\x8eq^\xc7i\xa3"
Encrypted payload length: 32
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received encrypted packet: b"y\xcfn\xa3\x9cX\x92e\xa3\xca\x15#\xb5\xab\xda\x1a\xdf\xa9\xfe\xa1_\xb6\xcd'\x1a\xb7\x8eq^\xc7i\xa3"
Received payload length: 32
Host 00:00:00:00:00:02 decrypted payload: 'Hello from C'
```

Figure 4: Encrypted Packets

6 Modulation

In a communication system, modulation is the process of converting a digital signal into a form suitable for transmission over an analog channel, while demodulation is the opposite process, i.e., converting a received analog signal back into the original digital signal. In digital communications, modulation-demodulation usually involves the conversion between a digital signal (such as a character string) and a binary signal. This conversion is accomplished through ASCII, a character coding standard used to map specific digital values to characters.

In my project, the modulation process involves converting data of string type to binary form. This process can be achieved by following steps:

- Character to ASCII conversion: first, each character in the string is converted to its corresponding ASCII value.ASCII assigns a unique number to each character which can be used to represent the character.
- ASCII to Binary Conversion: Next, each ASCII value is converted to its binary equivalent. Since ASCII is a 7- or 8-bit code, this conversion process involves representing these bits in binary form.
- Transmission of Binary Data: The converted binary data can then be transmitted over a network, simulating the actual process of transmitting a digital signal over a communication channel.

The demodulation process is the inverse of modulation, and it consists of the following steps:

- Binary to ASCII conversion: The received binary data is first converted back to ASCII. This is achieved by parsing the binary data into its corresponding ASCII value.
- ASCII to Character Conversion: Once the ASCII values are obtained, they can be converted back to their corresponding characters to reconstruct the original string data.
- Reconstruction and validation of the data: Finally, the demodulated data needs to be validated to ensure that no errors occurred during transmission or that errors can be detected and corrected.

Algorithm 4 Host Communication Protocol

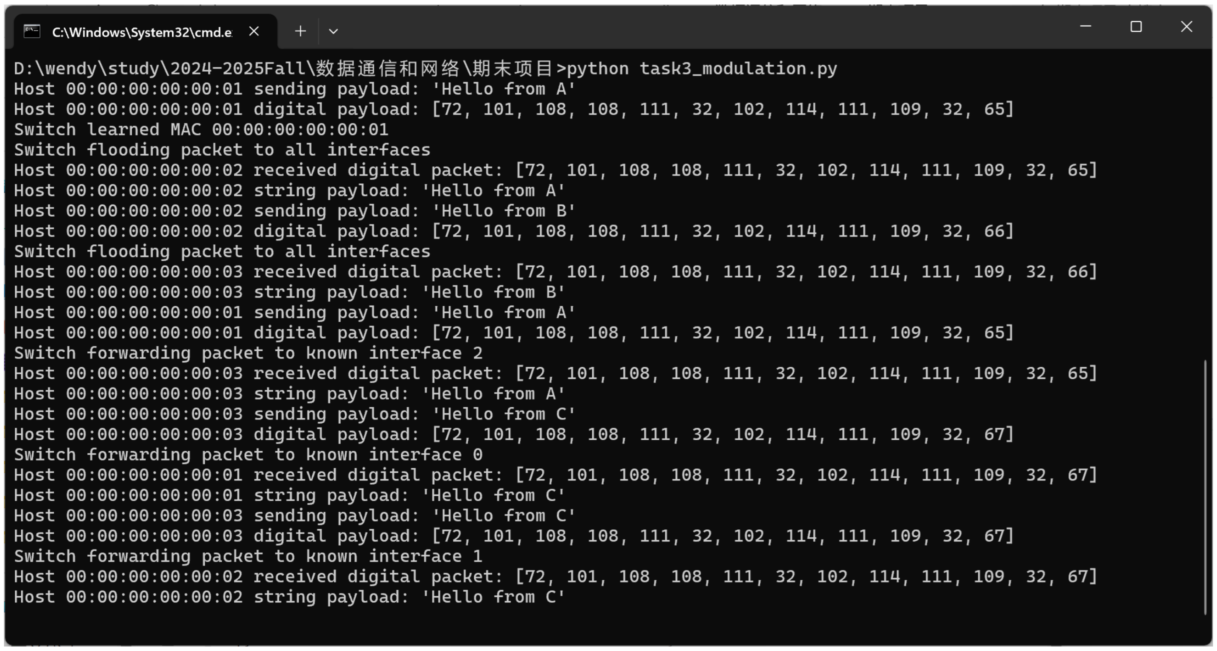
```
function String2Digital(message)
    return [ord(char) for char in message]
end function
function Digital2String(message)
    return ''.join(chr(num) for num in message)
end function
function send_packet(dst_mac, payload, switch)
    digital_payload = string2digital(payload)
    packet = Packet(self.mac, dst_mac, digital_payload)
    switch.handle_packet(packet)
end function
function receive_packet(packet)
    string_payload = digital2string(packet.payload)
    self.buffer.append(string_payload)
end function
```

Modulation and demodulation using ASCII codes for digital encoding has several advantages:

- Simplicity: ASCII is a widely understood and supported coding system, making the implementation process relatively simple.
- Compatibility: Due to the widespread use of ASCII, this modem method has good compatibility with a wide range of devices and systems.

- Efficiency: By directly converting characters to binary, this method is very efficient in processing text data.

Despite the above advantages of using ASCII for modem, challenges may be encountered in practical applications, such as handling data with non-ASCII character sets or ensuring the reliability of data transmission. To address these issues, more complex encoding schemes, such as UTF-8, as well as the introduction of error detection and correction mechanisms, such as parity check or more advanced correction codes, can be used to ensure data integrity and accuracy.



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python task3_modulation.py
Host 00:00:00:00:00:01 sending payload: 'Hello from A'
Host 00:00:00:00:00:01 digital payload: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 65]
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received digital packet: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 65]
Host 00:00:00:00:00:02 string payload: 'Hello from A'
Host 00:00:00:00:00:02 sending payload: 'Hello from B'
Host 00:00:00:00:00:02 digital payload: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 66]
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received digital packet: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 66]
Host 00:00:00:00:00:03 string payload: 'Hello from B'
Host 00:00:00:00:00:01 sending payload: 'Hello from A'
Host 00:00:00:00:00:01 digital payload: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 65]
Switch forwarding packet to known interface 2
Host 00:00:00:00:00:03 received digital packet: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 65]
Host 00:00:00:00:00:03 string payload: 'Hello from A'
Host 00:00:00:00:00:03 sending payload: 'Hello from C'
Host 00:00:00:00:00:03 digital payload: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 67]
Switch forwarding packet to known interface 0
Host 00:00:00:00:00:01 received digital packet: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 67]
Host 00:00:00:00:00:01 string payload: 'Hello from C'
Host 00:00:00:00:00:03 sending payload: 'Hello from C'
Host 00:00:00:00:00:03 digital payload: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 67]
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received digital packet: [72, 101, 108, 108, 111, 32, 102, 114, 111, 109, 32, 67]
Host 00:00:00:00:00:02 string payload: 'Hello from C'
```

Figure 5: Modulation

7 Firewall

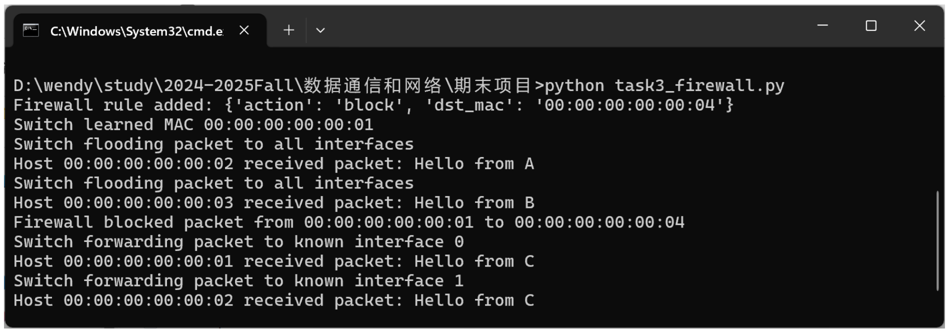
A firewall is a basic network security device whose main function is to create a protective barrier between internal and external networks. The core function of a firewall is to monitor and control the flow of data passing through it, ensuring that only packets that comply with a specific security policy are allowed to enter or leave the network. This security policy is usually based on a set of rules that define which types of traffic are allowed and which are prohibited.

The working principle of a firewall involves constant monitoring of network traffic and rule-based decision making. When a packet arrives at the firewall, the firewall examines the packet against a predefined set of rules. These rules may contain allow or deny instructions for specific IP addresses, ports, or protocols. If the packet's information matches one of the rules in the rule set, the firewall performs the appropriate action, which may be to release the packet or block it. In addition, the firewall has a logging function that records information about all blocked packets, which is very important for post-mortem analysis and security auditing.

This working mechanism of the firewall makes it a key component of network border security, which effectively prevents unauthorized access and protects the network from malicious traffic. By carefully scrutinizing traffic to and from the network, firewalls ensure the security of network resources and the integrity of data.

Algorithm 5 Firewall Rule Checking Algorithm

```
procedure AddFirewallRule(rule)
    $firewall\_rules.append(rule)$
    Print "Firewall rule added: rule"
end procedure
function CheckFirewall(packet)
    allowed ← True
    for rule in $firewall\_rules$ do
        if rule.action ← 'block' and (rule.src_mac ← packet.src or rule.dst_mac ← packet.dst) then
            allowed ← False
            break
        end if
    end for
    return allowed
end function
procedure HandlePacket(packet)
    if CheckFirewall(packet) then
        Forward packet
    else
        Print "Firewall blocked packet"
    end if
end procedure
```



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python task3_firewall.py
Firewall rule added: {'action': 'block', 'dst_mac': '00:00:00:00:00:04'}
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Host 00:00:00:00:00:02 received packet: Hello from A
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received packet: Hello from B
Firewall blocked packet from 00:00:00:00:00:01 to 00:00:00:00:00:04
Switch forwarding packet to known interface 0
Host 00:00:00:00:00:01 received packet: Hello from C
Switch forwarding packet to known interface 1
Host 00:00:00:00:00:02 received packet: Hello from C
```

Figure 6: Firewall

Firewalls have many advantages:

- Enhanced Security: Firewalls effectively prevent unauthorized access and protect internal networks from external threats.
- Access Control: Through precise rule settings, firewalls can restrict network access to specific users or services, enhancing network access control.
- Intrusion Prevention: Modern firewalls often integrate Intrusion Prevention System (IDS) features to identify and block potential attacks.
- Packet Filtering: Firewalls filter packets at the network layer, providing a first line of defense for network communications.

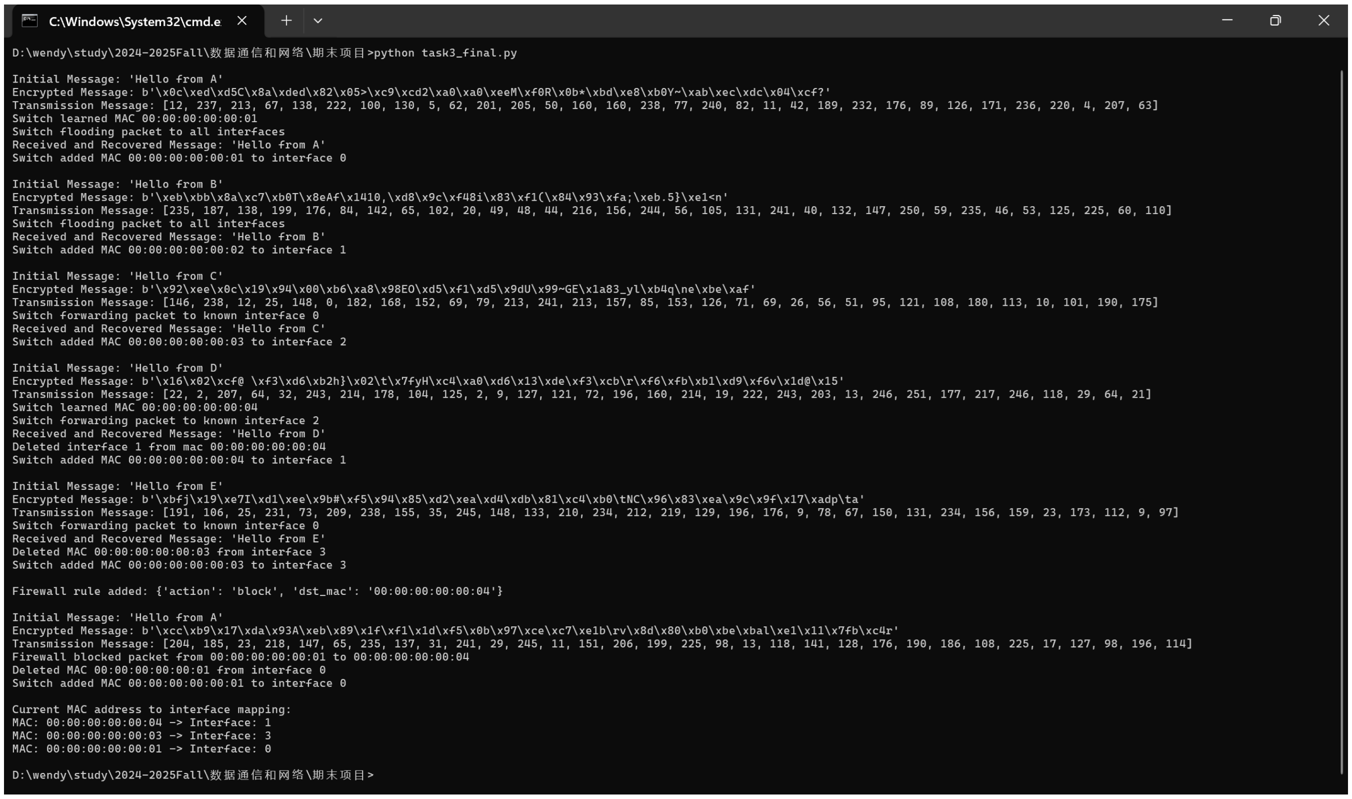
Inevitably, firewalls have certain drawbacks:

- Performance impact: In high-traffic networks, firewalls can become a performance bottleneck, especially when performing deep packet inspection.

- Complexity of rule management: As the network environment changes, firewall rules need to be constantly updated and maintained, which can lead to management complexity.

8 Conclusion

In conclusion, the Minimum Sim-LAN project successfully simulates the critical operations and security aspects of a local area network through its minimalist design and advanced features. The dynamic updating of switch tables, implementation of encrypted packets using AES encryption, modulation and demodulation processes, and the integration of firewalls, all contribute to a robust simulation that closely mirrors the complexities of real-world networks. This project not only provides a practical understanding of LAN operations but also highlights the importance of network security and data integrity in modern communication systems. The simulation's ability to adapt to network changes, protect against cyber threats, and maintain efficient data transmission demonstrates its educational and practical value. As network technologies continue to evolve, the principles and techniques implemented in this project remain foundational to the development of secure and efficient network systems.



```
D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>python task3_final.py

Initial Message: 'Hello from A'
Encrypted Message: b'\x0c\xed\xd5C\x8a\xded\x82\x05>\xc9\xcd2\xa0\xa0\xeeM\xf0R\x0b*\xbd\xe8\xb0Y~\xab\xec\xdc\x04\xcf?'
Transmission Message: [12, 237, 213, 67, 138, 222, 100, 130, 5, 62, 201, 205, 50, 160, 160, 238, 77, 240, 82, 11, 42, 189, 232, 176, 89, 126, 171, 236, 220, 4, 207, 63]
Switch learned MAC 00:00:00:00:00:01
Switch flooding packet to all interfaces
Received and Recovered Message: 'Hello from A'
Switch added MAC 00:00:00:00:00:01 to interface 0

Initial Message: 'Hello from B'
Encrypted Message: b'\xeb\xbb\x8a\xc7\xb0T\x8eAf\x1410,\xd8\x9c\xf48i\x83\xf1(\x84\x93\xfa;\xeb.5}\xe1<n'
Transmission Message: [235, 187, 138, 199, 176, 84, 142, 65, 102, 20, 49, 48, 44, 216, 156, 244, 56, 105, 131, 241, 40, 132, 147, 250, 59, 235, 46, 53, 125, 225, 60, 110]
Switch flooding packet to all interfaces
Received and Recovered Message: 'Hello from B'
Switch added MAC 00:00:00:00:00:02 to interface 1

Initial Message: 'Hello from C'
Encrypted Message: b'\x92\xee\x0c\x19\x94\x00\xb6\xa8\x98EO\xd5\xf1\xd5\x9dU\x99~GE\x1a83_yl\xb4q\ne\xbe\xaf'
Transmission Message: [146, 238, 12, 25, 148, 0, 182, 168, 152, 69, 79, 213, 241, 213, 157, 85, 153, 126, 71, 69, 26, 56, 51, 95, 121, 108, 180, 113, 10, 101, 190, 175]
Switch forwarding packet to known interface 0
Received and Recovered Message: 'Hello from C'
Switch added MAC 00:00:00:00:00:03 to interface 2

Initial Message: 'Hello from D'
Encrypted Message: b'\x16\x02\xcf@ \xf3\xd6\xb2h}\x02\t\x7fyH\xc4\xa0\xd6\x13\xde\xf3\xcb\r\xf6\xfb\xb1\xd9\xf6v\x1d@\x15'
Transmission Message: [22, 2, 207, 64, 32, 243, 214, 178, 104, 125, 2, 9, 127, 121, 72, 196, 160, 214, 19, 222, 243, 203, 13, 246, 251, 177, 217, 246, 118, 29, 64, 21]
Switch learned MAC 00:00:00:00:00:04
Switch forwarding packet to known interface 2
Received and Recovered Message: 'Hello from D'
Deleted interface 1 from mac 00:00:00:00:00:04
Switch added MAC 00:00:00:00:00:04 to interface 1

Initial Message: 'Hello from E'
Encrypted Message: b'\xbfj\x19\xe7I\xd1\xee\x9b#\xf5\x94\x85\xd2\xea\xd4\xdb\x81\xc4\xb0\tNC\x96\x83\xea\x9c\x9f\x17\xadp\ta'
Transmission Message: [191, 106, 25, 231, 73, 209, 238, 155, 35, 245, 148, 133, 210, 234, 212, 219, 129, 196, 176, 9, 78, 67, 150, 131, 234, 156, 159, 23, 173, 112, 9, 97]
Switch forwarding packet to known interface 0
Received and Recovered Message: 'Hello from E'
Deleted MAC 00:00:00:00:00:03 from interface 3
Switch added MAC 00:00:00:00:00:03 to interface 3

Firewall rule added: {'action': 'block', 'dst_mac': '00:00:00:00:00:04'}

Initial Message: 'Hello from A'
Encrypted Message: b'\xcc\xb9\x17\xda\x93A\xeb\x89\x1f\xf1\x1d\xf5\x0b\x97\xce\xc7\xe1b\rv\x8d\x80\xb0\xbe\xbal\xe1\x11\x7fb\xc4r'
Transmission Message: [204, 185, 23, 218, 147, 65, 235, 137, 31, 241, 29, 245, 11, 151, 206, 199, 225, 98, 13, 118, 141, 128, 176, 190, 186, 108, 225, 17, 127, 98, 196, 114]
Firewall blocked packet from 00:00:00:00:00:01 to 00:00:00:00:00:04
Deleted MAC 00:00:00:00:00:01 from interface 0
Switch added MAC 00:00:00:00:00:01 to interface 0

Current MAC address to interface mapping:
MAC: 00:00:00:00:00:04 -> Interface: 1
MAC: 00:00:00:00:00:03 -> Interface: 3
MAC: 00:00:00:00:00:01 -> Interface: 0

D:\wendy\study\2024-2025Fall\数据通信和网络\期末项目>
```

Figure 7: All Works Integrated Together

In the end, all the function was combined in a single script to realize a complete Minimum Sim LAN.

9 Appendix

```
from ee315_24_lib import SwitchFabric, Packet
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.backends import default_backend
import os
import re

# Generate AES key
def generate_aes_key():
  return os.urandom(16)  # Generate a 32-byte random key

# AES encryption function
def aes_encrypt(key, message):
  # Use PKCS7 padding to ensure the message length is a multiple of the AES block
      size
  padder = padding.PKCS7(128).padder()
  padded_message = padder.update(message.encode()) + padder.finalize()

  # Generate a random IV (Initialization Vector)
  iv = os.urandom(16)

  # Set up the AES cipher, using CBC mode
  cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
  encryptor = cipher.encryptor()

  # Encrypt the padded message
  encrypted_message = encryptor.update(padded_message) + encryptor.finalize()

  # Return the IV and the encrypted message (IV + encrypted message)
  return iv + encrypted_message

# AES decryption function
def aes_decrypt(key, encrypted_message):
  # Extract the IV (first 16 bytes) and the encrypted message
  iv = encrypted_message[:16]
  encrypted_message = encrypted_message[16:]

  # Set up the AES decryptor, using CBC mode
  cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
  decryptor = cipher.decryptor()

  # Decrypt the message
  padded_message = decryptor.update(encrypted_message) + decryptor.finalize()

  # Remove padding
  unpadder = padding.PKCS7(128).unpadder()

  try:
    message = unpadder.update(padded_message) + unpadder.finalize()
    return message.decode()  # Decode to a string
  except ValueError as e:
    return None
```

```python

def string2digital(message):
  return list(message)

# Convert digital to a byte string (digital payload)
def digital2string(digital_payload):
  return bytes(digital_payload)

class Host:
  def __init__(self, mac, interface, key):
    if not re.match(r'^([0-9A-Fa-f]{2}:){5}[0-9A-Fa-f]{2}$', mac):
      raise ValueError("Invalid MAC address format")
    self.mac = mac
    self.interface = interface
    self.buffer = []
    self.aes_key = key

  def send_packet(self, dest_mac, message, switch):
    print()
    print(f"Initial Message: '{message}'")
    encrypted_message = aes_encrypt(self.aes_key, message)  # Encrypt the message
    digital_payload = string2digital(encrypted_message)  # Convert to digital
        payload
    print(f"Encrypted Message: {encrypted_message}")
    print(f"Transmission Message: {digital_payload}")

    packet = Packet(self.mac, dest_mac, digital_payload)
    switch.handle_packet(packet)  # Send the packet through the switch
    switch.add_mac(self.mac, self.interface)

  def receive_packet(self, packet):
    if packet.dst == self.mac:
      byte_payload = digital2string(packet.payload)  # Convert to byte string

      # Decrypt the message
      decrypted_payload = aes_decrypt(self.aes_key, byte_payload)  # Decrypt the
          digital payload

      if decrypted_payload:
        self.buffer.append(decrypted_payload)
        print(f"Received and Recovered Message: '{decrypted_payload}'")
      else:
        print("Failed to decrypt the packet.")

class Switch:
  def __init__(self, fabric, num_interfaces=8):
    self.num_interfaces = num_interfaces
    self.interfaces = {}
    self.mac_table = {}
    self.mac = {}
    self.fabric = fabric
    self.firewall_rules = []  # New firewall rules list
    for i in range(self.num_interfaces):
      self.interfaces[i] = None

  def add_firewall_rule(self, rule):
    self.firewall_rules.append(rule)
    print()
    print(f"Firewall rule added: {rule}")

  def handle_packet(self, packet):
    # Learn the source MAC address and store the interface number
    src_mac = packet.src
    dst_mac = packet.dst

    # Check if the source MAC is already in the MAC table
    if src_mac not in self.mac_table:
      # Learn the source MAC address and associate it with the incoming interface
      self.mac_table[src_mac] = 0
      print(f"Switch learned MAC {src_mac}")
    else:
      self.mac_table[src_mac] = 1

    # Check firewall rules before forwarding
    if self.check_firewall(packet):
      # Selective forwarding or flooding
      if dst_mac in self.mac_table:
        # Forward to the known destination interface
        dst_interface = None
        for interface, mac in self.fabric.physical_map.items():
          if mac == dst_mac:
            dst_interface = interface
            break
        self.mac_table[dst_mac] = 1
        print(f"Switch forwarding packet to known interface {dst_interface}")
        self.fabric.forward_to_interface(packet, dst_interface)
      else:
        # Update the MAC table with the new interface
        self.mac_table[dst_mac] = 1
        # Flood to all interfaces except the incoming one
        print(f"Switch flooding packet to all interfaces")
        for i, host in self.interfaces.items():
          if host and i != src_mac:
            self.fabric.forward_to_interface(packet, i)
    else:
      print(f"Firewall blocked packet from {src_mac} to {dst_mac}")

  # Key changes
  def add_mac(self, mac, interface):
    # Delete the old interface
    old_mac = self.get_mac_for_interface(interface)
    if old_mac != mac and old_mac != None:
      del self.mac[old_mac]
      del self.mac_table[old_mac]
      print(f"Deleted interface {interface} from mac {mac}")
    # Delete the old mac
    if mac in self.mac:
      del self.mac[mac]
```

```
            del self.mac_table[mac]
            print(f"Deleted MAC {mac} from interface {interface}")
        self.mac[mac] = interface
        print(f"Switch added MAC {mac} to interface {interface}")

    def get_mac_for_interface(self, interface):
        for mac, intf in self.mac.items():
            if intf == interface:
                return mac
        return None

    def print_mac(self):
        print()
        print("Current MAC address to interface mapping:")
        for mac, interface in self.mac.items():
            print(f"MAC: {mac} -> Interface: {interface}")

    def check_firewall(self, packet):
        # Default to allow all packets through
        allowed = True
        for rule in self.firewall_rules:
            if rule['action'] == 'block':
                # If the rule includes src_mac, check the source MAC address
                if 'src_mac' in rule and rule['src_mac'] == packet.src:
                    allowed = False
                    break
                # If the rule includes dst_mac, check the destination MAC address
                if 'dst_mac' in rule and rule['dst_mac'] == packet.dst:
                    allowed = False
                    break
        return allowed

# Create the network
shared_fabric = SwitchFabric()
switch = Switch(shared_fabric)

# Generate AES key
key = generate_aes_key()

# Create hosts
host1 = Host("00:00:00:00:00:01", 0, key)
host2 = Host("00:00:00:00:00:02", 1, key)
host3 = Host("00:00:00:00:00:03", 2, key)

# Connect hosts to the switch
shared_fabric.connect_host_to_switch(host1, switch)
shared_fabric.connect_host_to_switch(host2, switch)
shared_fabric.connect_host_to_switch(host3, switch)

# Simulate communication
host1.send_packet("00:00:00:00:00:02", "Hello from A", switch)
host2.send_packet("00:00:00:00:00:03", "Hello from B", switch)
host3.send_packet("00:00:00:00:00:01", "Hello from C", switch)

```

```python
# Add a new MAC address
host4 = Host("00:00:00:00:00:04", 1, key)
shared_fabric.connect_host_to_switch(host4, switch)
host4.send_packet("00:00:00:00:00:03", "Hello from D", switch)

# Add a new interface
host5 = Host("00:00:00:00:00:03", 3, key)
shared_fabric.connect_host_to_switch(host5, switch)
host5.send_packet("00:00:00:00:00:01", "Hello from E", switch)

# Add a firewall rule to block communication from 00:00:00:00:00:04
switch.add_firewall_rule({'action': 'block', 'dst_mac': '00:00:00:00:00:04'})
host1.send_packet("00:00:00:00:00:04", "Hello from A", switch)  # This package
    will be blocked

# print the dictionary
switch.print_mac()
```