

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.

Contents

1	Introduction	2
2	Switch Table Update	2
3	Encrypted Packets	3
4	Modulation	5
5	Firewall	6
6	Conclusion	8
7	Appendix	9

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 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.

Algorithm 1 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
```

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.

```

C:\Windows\System32\cmd.e x + v
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 1: Switch Table Update

3 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.

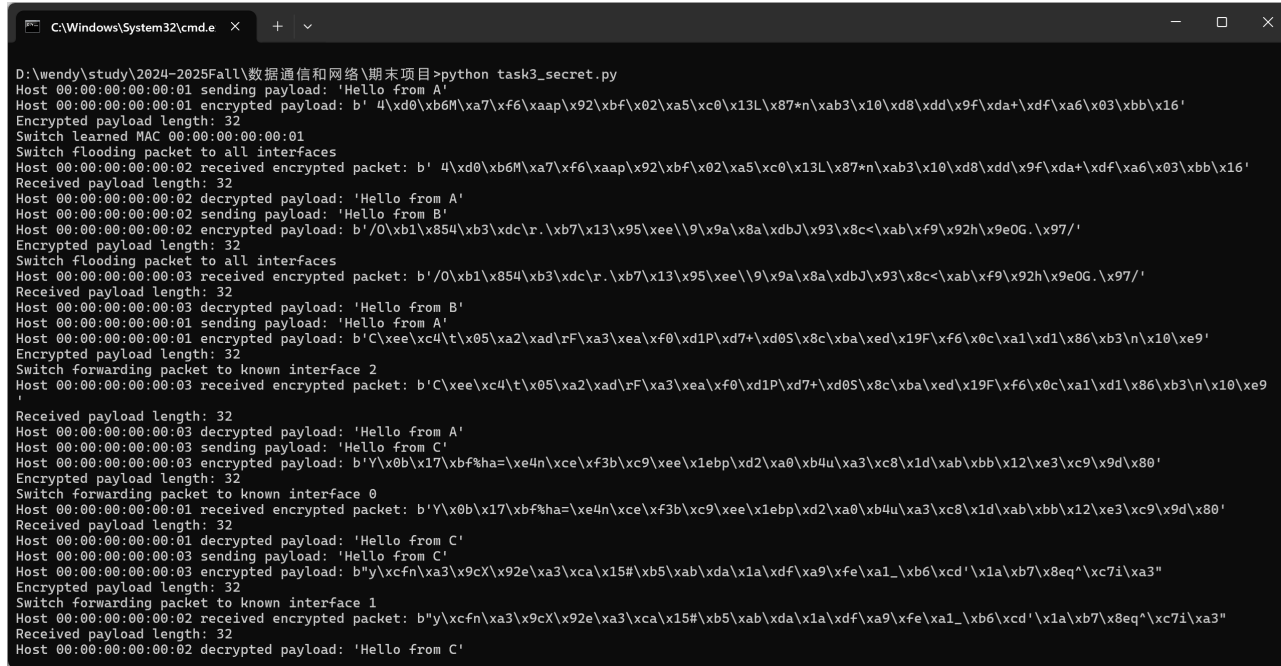
Algorithm 2 AES-CBC Encryption and Decryption

- 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 cryptanalysis. 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\xaaap\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\xaaap\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\xbl\x854\xb3\xdc\r.\xb7\x13\x95\xee\l9\x9a\x8a\xdbj\x93\x8c<\xab\x9f\x92h\x9e0G.\x97/'
Encrypted payload length: 32
Switch flooding packet to all interfaces
Host 00:00:00:00:00:03 received encrypted packet: b'/O\xbl\x854\xb3\xdc\r.\xb7\x13\x95\xee\l9\x9a\x8a\xdbj\x93\x8c<\xab\x9f\x92h\x9e0G.\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\xfa\x01P\xd7+\xd05\x8c\xba\xed\x19F\xfa\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\xfa\x01P\xd7+\xd05\x8c\xba\xed\x19F\xfa\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\xfb\x9c\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\xfb\x9c\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 2: Encrypted Packets

4 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 3 Host Communication Protocol

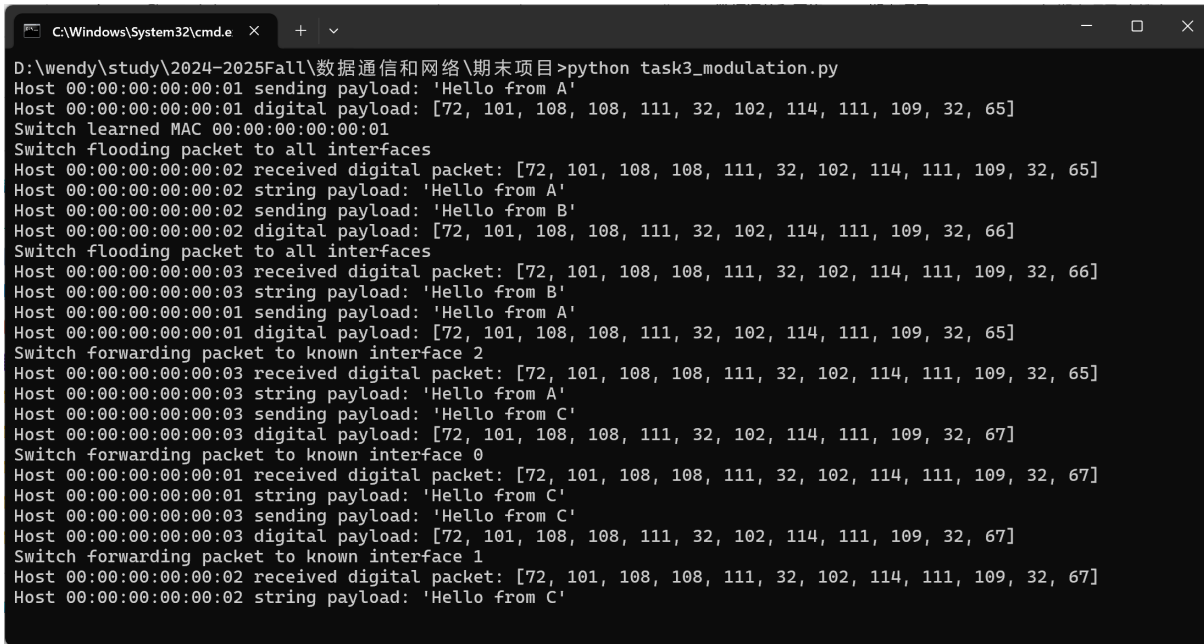
```
1: function STRING2DIGITAL(message)
2:   return [ord(char) for char in message]
3: end function
4: function DIGITAL2STRING(message)
5:   return ''.join(chr(num) for num in message)
6: end function
7: function SEND_PACKET(dst_mac, payload, switch)
8:   digital_payload = STRING2DIGITAL(payload)
9:   packet = PACKET(self.mac, dst_mac, digital_payload)
10:  switch.HANDLE_PACKET(packet)
11: end function
12: function RECEIVE_PACKET(packet)
13:  string_payload = DIGITAL2STRING(packet.payload)
14:  self.buffer.append(string_payload)
15: 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.



```

C:\Windows\System32\cmd.e  x  +  v
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 3: Modulation

5 Firewall

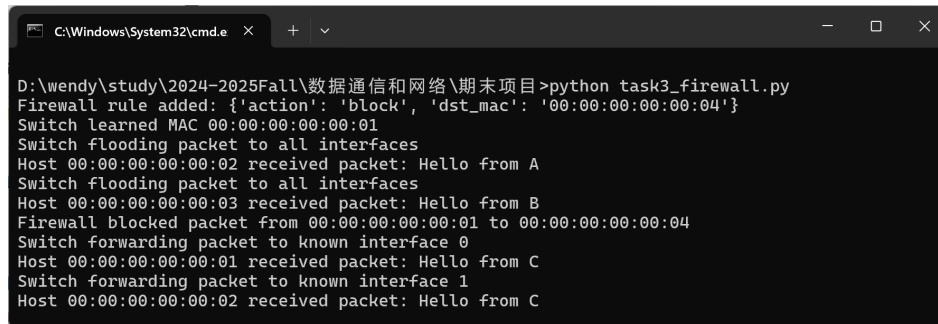
A firewall is a basic network security device whose main function is to create a protective barrier between internal and external networks. The core function of a firewall is to monitor and control the flow of data passing through it, ensuring that only packets that comply with a specific security policy 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 4 Firewall Rule Checking Algorithm

```
1: procedure ADDFIREWALLRULE(rule)
2:   firewall_rules.append(rule)
3:   Print "Firewall rule added: rule"
4: end procedure
5: function CHECKFIREWALL(packet)
6:   allowed  $\leftarrow$  True
7:   for rule in firewall_rules do
8:     if rule.action  $\leftarrow$  'block' and (rule.src_mac  $\leftarrow$  packet.src or rule.dst_mac  $\leftarrow$  packet.dst) then
9:       allowed  $\leftarrow$  False
10:      break
11:    end if
12:  end for
13:  return allowed
14: end function
15: procedure HANDLEPACKET(packet)
16:  if CHECKFIREWALL(packet) then
17:    Forward packet
18:  else
19:    Print "Firewall blocked packet"
20:  end if
21: end procedure
```



```
C:\Windows\System32\cmd.e x + v
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 4: 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.

6 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.

```
C:\Windows\System32\cmd.exe
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\x05\x8a\xde\x82\x05\xcd\x0a\x0a\xeeM\xf0R\x0b*\xbd\x08\x0b~\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'\xab\xbb\xba\x0a\x07\x00T\x0eAf\x1d1g,\x0d\x0c\x0f0B1x03\xf1(\x0d\x03\xfa;\x0b.5}\x01c-'
Transmission Message: [235, 187, 138, 199, 176, 84, 142, 65, 102, 20, 49, 46, 44, 216, 156, 240, 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\x0e\x0c\x19\x94\x00\x06\x0a\x89E0\x05\xf1\x05\x09dU\x99-GE\x1a83_yl\x04q\n\x0e\x0a'f'
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, 161, 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'\x10\x02\xcf0 \xf3\x06\x02h}\x02\t\x7fyH\x0c\xa0\x06\x13\x0e\xf3\xcb'r\x0f6\xfb\x01\x0d\xf6v\x1d0\x15'
Transmission Message: [2, 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'\xbffj\x19\x0e7I\x01\x0e\x09b*\xf5\x94\x85\x02\x0a\x04\x0b\x01\x0c\x0b\x01\x0c\x06\x83\x0e\x0c\x09\x0f\x17\x0adp\ta'
Transmission Message: [191, 106, 25, 231, 73, 289, 238, 155, 35, 245, 148, 133, 218, 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'\x0c\x0b\x17\x0a\x93A\x0b\x089\x1f\xf1\x1d\xf5\x0b\x97\x0c\x07\x0e1b\r\n\x0d\x08\x0b\x0e\x0a1\x01\x11\x07fb\x04r'
Transmission Message: [20, 10, 23, 2, 107, 65, 235, 137, 31, 241, 29, 245, 11, 151, 286, 199, 225, 98, 13, 118, 141, 128, 176, 190, 186, 108, 225, 17, 127, 98, 196, 114]
Finally blocked packet from 00:00:00:00:00:04 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 5: All Works Integrated Together

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

7 Appendix

```
1 from ee315_24_lib import SwitchFabric, Packet
2 from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
3 from cryptography.hazmat.primitives import padding
4 from cryptography.hazmat.backends import default_backend
5 import os
6 import re
7
8 # Generate AES key
9 def generate_aes_key():
10     return os.urandom(16) # Generate a 32-byte random key
11
12 # AES encryption function
13 def aes_encrypt(key, message):
14     # Use PKCS7 padding to ensure the message length is a multiple of the AES block
15     # size
16     padder = padding.PKCS7(128).padder()
17     padded_message = padder.update(message.encode()) + padder.finalize()
18
19     # Generate a random IV (Initialization Vector)
20     iv = os.urandom(16)
21
22     # Set up the AES cipher, using CBC mode
23     cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
24     encryptor = cipher.encryptor()
25
26     # Encrypt the padded message
27     encrypted_message = encryptor.update(padded_message) + encryptor.finalize()
28
29     # Return the IV and the encrypted message (IV + encrypted message)
30     return iv + encrypted_message
31
32 # AES decryption function
33 def aes_decrypt(key, encrypted_message):
34     # Extract the IV (first 16 bytes) and the encrypted message
35     iv = encrypted_message[:16]
36     encrypted_message = encrypted_message[16:]
37
38     # Set up the AES decryptor, using CBC mode
39     cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=default_backend())
40     decryptor = cipher.decryptor()
41
42     # Decrypt the message
43     padded_message = decryptor.update(encrypted_message) + decryptor.finalize()
44
45     # Remove padding
46     unpadder = padding.PKCS7(128).unpadder()
47
48     try:
49         message = unpadder.update(padded_message) + unpadder.finalize()
50         return message.decode() # Decode to a string
51     except ValueError as e:
52         return None
```

```

52
53 def string2digital(message):
54     return list(message)
55
56 # Convert digital to a byte string (digital payload)
57 def digital2string(digital_payload):
58     return bytes(digital_payload)
59
60 class Host:
61     def __init__(self, mac, interface, key):
62         if not re.match(r'^([0-9A-Fa-f]{2}:){5}([0-9A-Fa-f]{2})$', mac):
63             raise ValueError("Invalid MAC address format")
64         self.mac = mac
65         self.interface = interface
66         self.buffer = []
67         self.aes_key = key
68
69     def send_packet(self, dest_mac, message, switch):
70         print()
71         print(f"Initial Message: '{message}'")
72         encrypted_message = aes_encrypt(self.aes_key, message) # Encrypt the message
73         digital_payload = string2digital(encrypted_message) # Convert to digital
74             payload
75         print(f"Encrypted Message: {encrypted_message}")
76         print(f"Transmission Message: {digital_payload}")
77
78         packet = Packet(self.mac, dest_mac, digital_payload)
79         switch.handle_packet(packet) # Send the packet through the switch
80         switch.add_mac(self.mac, self.interface)
81
82     def receive_packet(self, packet):
83         if packet.dst == self.mac:
84             byte_payload = digital2string(packet.payload) # Convert to byte string
85
86             # Decrypt the message
87             decrypted_payload = aes_decrypt(self.aes_key, byte_payload) # Decrypt the
88                 digital payload
89
90             if decrypted_payload:
91                 self.buffer.append(decrypted_payload)
92                 print(f"Received and Recovered Message: '{decrypted_payload}'")
93             else:
94                 print("Failed to decrypt the packet.")
95
96 class Switch:
97     def __init__(self, fabric, num_interfaces=8):
98         self.num_interfaces = num_interfaces
99         self.interfaces = {}
100         self.mac_table = {}
101         self.mac = {}
102         self.fabric = fabric
103         self.firewall_rules = [] # New firewall rules list
104         for i in range(self.num_interfaces):
105             self.interfaces[i] = None

```

```

104
105 def add_firewall_rule(self, rule):
106     self.firewall_rules.append(rule)
107     print()
108     print(f"Firewall rule added: {rule}")
109
110 def handle_packet(self, packet):
111     # Learn the source MAC address and store the interface number
112     src_mac = packet.src
113     dst_mac = packet.dst
114
115     # Check if the source MAC is already in the MAC table
116     if src_mac not in self.mac_table:
117         # Learn the source MAC address and associate it with the incoming interface
118         self.mac_table[src_mac] = 0
119         print(f"Switch learned MAC {src_mac}")
120     else:
121         self.mac_table[src_mac] = 1
122
123     # Check firewall rules before forwarding
124     if self.check_firewall(packet):
125         # Selective forwarding or flooding
126         if dst_mac in self.mac_table:
127             # Forward to the known destination interface
128             dst_interface = None
129             for interface, mac in self.fabric.physical_map.items():
130                 if mac == dst_mac:
131                     dst_interface = interface
132                     break
133             self.mac_table[dst_mac] = 1
134             print(f"Switch forwarding packet to known interface {dst_interface}")
135             self.fabric.forward_to_interface(packet, dst_interface)
136         else:
137             # Update the MAC table with the new interface
138             self.mac_table[dst_mac] = 1
139             # Flood to all interfaces except the incoming one
140             print(f"Switch flooding packet to all interfaces")
141             for i, host in self.interfaces.items():
142                 if host and i != src_mac:
143                     self.fabric.forward_to_interface(packet, i)
144         else:
145             print(f"Firewall blocked packet from {src_mac} to {dst_mac}")
146
147 # Key changes
148 def add_mac(self, mac, interface):
149     # Delete the old interface
150     old_mac = self.get_mac_for_interface(interface)
151     if old_mac != mac and old_mac != None:
152         del self.mac[old_mac]
153         del self.mac_table[old_mac]
154         print(f"Deleted interface {interface} from mac {mac}")
155     # Delete the old mac
156     if mac in self.mac:
157         del self.mac[mac]

```

```

158     del self.mac_table[mac]
159     print(f"Deleted MAC {mac} from interface {interface}")
160     self.mac[mac] = interface
161     print(f"Switch added MAC {mac} to interface {interface}")
162
163 def get_mac_for_interface(self, interface):
164     for mac, intf in self.mac.items():
165         if intf == interface:
166             return mac
167     return None
168
169 def print_mac(self):
170     print()
171     print("Current MAC address to interface mapping:")
172     for mac, interface in self.mac.items():
173         print(f"MAC: {mac} -> Interface: {interface}")
174
175 def check_firewall(self, packet):
176     # Default to allow all packets through
177     allowed = True
178     for rule in self.firewall_rules:
179         if rule['action'] == 'block':
180             # If the rule includes src_mac, check the source MAC address
181             if 'src_mac' in rule and rule['src_mac'] == packet.src:
182                 allowed = False
183                 break
184             # If the rule includes dst_mac, check the destination MAC address
185             if 'dst_mac' in rule and rule['dst_mac'] == packet.dst:
186                 allowed = False
187                 break
188     return allowed
189
190 # Create the network
191 shared_fabric = SwitchFabric()
192 switch = Switch(shared_fabric)
193
194 # Generate AES key
195 key = generate_aes_key()
196
197 # Create hosts
198 host1 = Host("00:00:00:00:00:01", 0, key)
199 host2 = Host("00:00:00:00:00:02", 1, key)
200 host3 = Host("00:00:00:00:00:03", 2, key)
201
202 # Connect hosts to the switch
203 shared_fabric.connect_host_to_switch(host1, switch)
204 shared_fabric.connect_host_to_switch(host2, switch)
205 shared_fabric.connect_host_to_switch(host3, switch)
206
207 # Simulate communication
208 host1.send_packet("00:00:00:00:00:02", "Hello from A", switch)
209 host2.send_packet("00:00:00:00:00:03", "Hello from B", switch)
210 host3.send_packet("00:00:00:00:00:01", "Hello from C", switch)
211

```

```

212 # Add a new MAC address
213 host4 = Host("00:00:00:00:00:04", 1, key)
214 shared_fabric.connect_host_to_switch(host4, switch)
215 host4.send_packet("00:00:00:00:00:03", "Hello from D", switch)
216
217 # Add a new interface
218 host5 = Host("00:00:00:00:00:03", 3, key)
219 shared_fabric.connect_host_to_switch(host5, switch)
220 host5.send_packet("00:00:00:00:00:01", "Hello from E", switch)
221
222 # Add a firewall rule to block communication from 00:00:00:00:00:04
223 switch.add_firewall_rule({'action': 'block', 'dst_mac': '00:00:00:00:00:04'})
224 host1.send_packet("00:00:00:00:00:04", "Hello from A", switch) # This package
    will be blocked
225
226 # print the dictionary
227 switch.print_mac()

```