Code Explanation

Packet Type

Declare the packet types used later in `handlePacket()`

```
class PacketType(Enum):
    ICMP_REQUEST = 0
    ICMP_RESPONSE = 1
    ARP_REQEUST = 2
    ARP_RESPONSE = 3
```

class Host

send()

Simulates the host sending packet, which told the node that the host is connected to handle the packet.
Will tell the receiver:

- self.mac: MAC source in ethernet header
- self.name: helps Switches to determine which port is this packet from
- dst_mac: MAC destination in ethernet header
- **kwargs: the payload
  - for ICMP packets:
    - src_ip: Source address in ipv4 header
    - dst_ip: Destination address in ipv4 header
  - for ARP packets:
    - src_ip: Sender protocol address(SPA) in ARP packet
    - ip: Target protocol address(TPA) in ARP packet
    - mac: Target hardware address(THA) in ARP packet
- packet_type: helps the receiver to determine how to handle the packet

```
def send(self, packet_type: PacketType, dst_mac: str, **kwargs):
        node = self.port_to
        node.handle_packet(packet_type, self.mac, self.name, dst_mac, **k
```

handlePacket()

Deal with different type of packets that the host received.

- case ARP_REQUEST:
  - Drop the packet if its destination isn't this host
  - Send an ICMP response back to the source
- ICMP_RESPONSE:
  - Do nothing
- ARP_REQUEST
  - Drop the packet if the target ip address isn't this host
  - Add the source ip/mac pair to the ARP table
  - send ARP response back to the source
- ARP_RESPONSE
  - Add the replied ip/mac pair to the ARP table

```
def handle_packet(self, packet_type: PacketType, src_mac: str, src_name:
        if packet_type == PacketType.ICMP_REQUEST:
            if kwargs['dst_ip'] != self.ip: return 1
            self.ping(kwargs['src_ip'], False)

        elif packet_type == PacketType.ICMP_RESPONSE:
            pass

        elif packet_type == PacketType.ARP_REQEUST:
            if(kwargs['ip'] != self.ip): return 1
            self.update_arp(kwargs['src_ip'], src_mac)
            self.send(PacketType.ARP_RESPONSE, src_mac, src_ip=kwargs['sr

        elif packet_type == PacketType.ARP_RESPONSE:
            if(kwargs['src_ip'] != self.ip): return 1
            self.update_arp(kwargs['ip'], kwargs['mac'])

        else:
            return 1
```

Others

Broadcast the ARP request.
Return 1 if the target_ip isn't added to the ARP table (didn't receive ARP reply).

```
def arp_request(self, target_ip: str):
        self.send(PacketType.ARP_REQEUST, 'ffff', src_ip=self.ip, ip=targ
        return target_ip not in self.arp_table
```

Send ICMP request/reply. Send ARP request first if the destination IP isn't in the ARP table.

```
def ping(self, dst_ip: str, isRequest = True):
        ptype = PacketType.ICMP_REQUEST if isRequest else PacketType.ICMP_

        if(dst_ip not in self.arp_table):
            if self.arp_request(dst_ip) == 1:
                return 1

        self.send(ptype, self.arp_table[dst_ip], src_ip=self.ip, dst_ip=d
        return 0
```

class Switch

send()

If the destination MAC address isn't found in the switch's MAC table, or it's a broadcast message, then flood the packet to all ports excluding the incoming port. Otherwise, send the packet to specific port.

```
def send(self, packet_type: PacketType, src_mac: str, dst_mac: str, in_pc
        if dst_mac in self.mac_table and dst_mac != 'ffff':
            port_idx = self.mac_table[dst_mac]
            node = self.port_to[port_idx]
            node.handle_packet(packet_type, src_mac, self.name, dst_mac,
        else:
            for i in range(self.port_n):
                if i == in_port: continue
                self.port_to[i].handle_packet(packet_type, src_mac, self.

```

handlePacket()

Upon receiving a packet, add the incoming port and the source MAC address to the MAC table, then forward the packet.

```
def handle_packet(self, packet_type: PacketType, src_mac: str, src_name:
    # Find out incoming port. This should be done by hardware in reality.
    in_port = -1
    for i in range(self.port_n):
        if self.port_to[i].name == src_name:
            in_port = i
            break
    if in_port == -1: return 1

    # Main function
    self.update_mac(src_mac, in_port)
    self.send(packet_type, src_mac, dst_mac, in_port, **kwargs)
```

Main Function

The main function is the same as the example code.

Questions

1. What is the difference between broadcasting and flooding in a network?

Broadcasting: Broadcast packet to all hosts in the network

Flooding: Unicast packet, only flood to all connected ports if destination not found in MAC table.

2. Explain the steps involved in the process of h1 ping h7 when there are no entries in the switch’s MAC table and the host’s ARP table.

- `h1` check `h7ip` in its ARP table -> Not found
- `h1` broadcast ARP request, target ip=h7ip
    - `s1~s7` received ARP request from `h1` , add `(h1mac, port)` to their MAC tables
    - s1~s7 flood the ARP request
- `h2~h6, h8` received the ARP request and dropped it.
- `h7` received the ARP request, add `(h1ip, h1mac)` to its ARP table
- `h7` send ARP response back to h1
    - `s6, s5, s7, s2, s1` received the ARP response from `h7` , added `(h7mac, port)` to their MAC tables

    - `s6, s5, s7, s2, s1` have `h1mac` in their mac tables so they send the packet directly to specific ports
- `h1` received the ARP response, add `(h7ip, h7mac)` to its ARP table
- `h1` sends ICMP request to `h7`
    - `s1, s2, s7, s5, s6` received the ICMP request from `h1`
    - `s1, s2, s7, s5, s6` have `h7mac` in their mac tables so they send the packet directly to specific ports
- `h7` received the ICMP request
- `h7` check `h1ip` in its ARP table -> Found
- `h7` sends ICMP response to `h1`
    - `s6, s5, s7, s2, s1` received the ICMP response from `h7`
    - `s6, s5, s7, s2, s1` have `h1mac` in their mac tables so they send the packet directly to specific ports
- `h1` received the ICMP response

3. What problem can arise when connecting s2 and s5 together and thus creating a switching loop? How can this issue be addressed?

For example, `h1 ping h2` and the MAC table of all switches are empty.

- s1 will flood the packet to s2.
- s2 will flood the packet to s5 and s7
- s7 will flood the packet to s5 while s5 flood the same packet to s7
- This goes on forever as long as s2, s5, s7 doesn't have h2mac on their MAC table

Packets will be stuck in the loop.
Can be dealt with the spanning tree protocol(STP).