# Network System Capstone

110550039 劉詠

## Code Explanation

**FOR HOST:**

```
class host:
    ......
    def clear(self):
        # clear ARP table entries for this host
        self.arp_table.clear()
    def update_arp(self, ip, mac):
        # update ARP table with a new entry
        self.arp_table[ip] = mac
    def handle_packet(self, packet):
        if packet['type'] == 'arp' and packet['reply'] != 'arpreply':
            source_ip = packet['source_ip']
            source_mac = packet['source_mac']
            # Check if the destination IP matches its own IP address
            if packet['destination_ip'] == self.ip:
                # Send ARP reply to the source (h1)
                # Update ARP table
                self.update_arp(source_ip, source_mac)
                arp_reply = {'type': 'arp', 'source_ip': self.ip,
               'source_mac': self.mac,'destination_ip': source_ip,
               'destination_mac': source_mac, 'reply': 'arpreply'}
                arp_reply['incoming_port'] = self.name
                self.send(arp_reply)
            else:
               # If destination IP not match, ignore the ARP request
                pass
        if packet['reply'] == 'arpreply':
            if packet['destination_ip'] == self.ip:
                source_ip = packet['source_ip']
                source_mac = packet['source_mac']
                self.update_arp(source_ip, source_mac)
                icmp_packet = {'type': 'icmp', 'source_ip': self.ip,
               'destination_ip': source_ip, 'destination_mac':
              source_mac, 'reply': 'ping_icmp'}
                icmp_packet['incoming_port'] = self.name
```

```
                self.send(icmp_packet)
        if packet['type'] == 'icmp' and packet['reply'] !=
'icmpreply':#need to send icmp reply
            if packet['destination_ip'] == self.ip:
                source_ip = packet['source_ip']
                source_mac = packet['source_mac']
                self.update_arp(source_ip, source_mac)#
                icmp_packet = {'type': 'icmp', 'source_ip': self.ip,
                'destination_ip': source_ip, 'destination_mac':
                source_mac, 'reply': 'icmpreply'}
                icmp_packet['incoming_port'] = self.name
                self.send(icmp_packet)
        if packet['reply'] == 'icmpreply':
            if packet['destination_ip'] == self.ip:
                pass
    def ping(self, dst_ip):
        # handle a ping request
        if dst_ip in self.arp_table:
            # Send ICMP request to the destination
            icmp_packet = {'type': 'icmp', 'source_ip': self.ip,
            'destination_ip': dst_ip, 'reply': 'ping_icmp'}
            self.send(icmp_packet)
        else:
            # Broadcast ARP request to all hosts
            arp_packet = {'type': 'arp', 'source_ip': self.ip,
            'destination_ip': dst_ip,'reply': 'ping_arp'}
            self.send(arp_packet)

    def send(self, packet):
        # determine the destination MAC here
        if packet['type'] == 'arp' and packet['reply'] != 'arpreply':
            destination_mac = 'ffff' # Broadcast MAC address
        else:
            destination_mac =
            self.arp_table.get(packet['destination_ip'], None)
        # Create a packet with source and destination MAC addresses
        packet['source_mac'] = self.mac
        packet['destination_mac'] = destination_mac
```

```
        node = self.port_to  # Get node connected to this host
        packet['incoming_port'] = self.name
        node.handle_packet(packet)
```

In the above code of host, in ping function, check if dst_ip is in current host's ARP table, if is, call the send function to send ICMP packet. If not, send ARP request packet. In send function, first check if the packet is ARP request, let its destination mac as 'ffff', if not, look up current host's ARP table to get the particular destination mac. Attach the 'incoming_port' information to the packet that is going to be send to switch for later using. In handle_packet() of host, if a host receive an ARP request, it will first check the destination_ip is as the same as its ip. If is, update the current host's ARP table, and send ARP reply packet with packet information 'reply' be 'arpreply'. For other hosts whose ip and destination_ip does not match, they will ignore the packet. If a host receives a packet with information 'reply' being 'arpreply', update the current host's ARP table and then send ICMP request packet. For a host receives ICMP request, I make the 'reply' in the ICMP reply packet be 'icmpreply', and send the packet. For a host that receives an ICMP reply packet, since it won't going to do something, I just pass this part.

**FOR SWITCHES:**

```
class switch:
    ......
    def clear(self):
        # clear MAC table entries for this switch
        self.mac_table.clear()
    def update_mac(self, mac, port):
        # update MAC table with a new entry
        self.mac_table[mac] = port
    def send(self, idx, packet): # send to the specified port
        node = self.port_to[idx]
        node.handle_packet(packet)  # Pass the incoming port index
    def handle_packet(self, packet):
        # Extract source MAC address from the packet
        source_mac = packet['source_mac']
        # Update MAC table and the incoming port number
        count = 0
        for i in self.port_to:
            if i.name == packet['incoming_port']:
                break
            count += 1
```

```
        incoming_port = count
        self.update_mac(source_mac, incoming_port)
        # Check if the destination MAC address is 'ffff'
        if packet['destination_mac'] == 'ffff':
            # Flood the packet out of all ports except the incoming port
            for i, port in enumerate(self.port_to):
                if i != incoming_port:
                    packet['incoming_port'] = self.name
                    self.send(i, packet)
        else:
            destination_mac = packet['destination_mac']
            # Check if the destination MAC address is in the MAC table
            if destination_mac in self.mac_table:
                # Send the packet out of the port associated with the
                 destination MAC address
                if incoming_port != self.mac_table[destination_mac]:
                    packet['incoming_port'] = self.name
                    self.send(self.mac_table[destination_mac], packet)
            else:
                # Flood the packet by sending it on every port except
                 the incoming one
                for i, port in enumerate(self.port_to):
                    if i != incoming_port:
                        packet['incoming_port'] = self.name
                        self.send(i, packet)
```

for switches, if a switch receives a packet, it first search which port the packet comes from, and using that port number to update current switch's mac table. And for a packet with destination_mac address equals 'ffff', it is an ARP request packet, we need to flood it to all nodes connected to current switch except the incoming one. Then, if the destination_mac is not 'ffff', check if the destination_mac is in current switch's mac table, if is, check whether the incoming_port is not equal to the current switch's mac_table[destination_mac] to avoid recursion loop, later, update the information 'incoming_port' of the packet as the current switch's name and send. If destination_mac is not in current switch's mac table, flood the packet to the nodes that connect to the current switch except the incoming one.

The above is the whole of my code explanation.

## Question Explanation

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

   A “Broadcast” is where a device will send out controlled requests in order to obtain the IP address of the host that it is looking for. A “Flood” is an uncontrolled broadcast, which is a technique where a message is forwarded to every neighbor node except the one from which it was received, and flooding does not require any routing information or configuration; it simply sends the message out on all available links.

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.**

   a. h1 first ping h7, send ARP request packet to switch 1 and switch updates its mac table.

   b. since destination mac address is ‘ffff’, s1 send the ARP request packet to its connected nodes (s2 and h2), h2’s ip address did not correspond to the destination ip address, it ignored the request.

   c. For s2, it will send the ARP packet to s7 and s3, for s3, it will send the packet to h3 and h4, again, both h3 and h4 ignored the packet. For s7, it will send the packet to s5. Then, s5 will keep sending packet to s4 and s6, for s4, and for h5 and h6, they ignore the ARP request packet. For s6, it will then send the packet to h7 and h8, for h8, it ignored the packet.

   d. For h7, since its ip address is the ARP request destination ip address, h7 updates its ARP table. And send ARP reply packet with destination mac address be the source address of the original packet back to h1.

   e. Since switches that the ARP request packet went through have update their mac tables, therefore, when h7 want to send ARP reply back to h1, it will according those switches mac tables to go through the path.

   f. When h1 receives the ARP reply, it will update its ARP table and send ICMP request to h7.

   g. After h7 receives the ICMP request from h1, it will send the ICMP reply back to h1.

3. **What problem can arise when connecting s2 and s5 together and thus creating a switching loop? How can this issue be addressed? (You should mention the specific algorithm or protocol used.)**

   **A.**

   a. Broadcast Storms: Since switches forward broadcast packets out of all ports except the one they were received on, in a looped network, a broadcast packet can circulate endlessly, resulting in a broadcast storm, which can overwhelm

the network with unnecessary traffic, leading to performance degradation or even network failure.

b. Duplicate Frame Transmission: In a switching loop, a frame may circulate endlessly between switches, leading to the duplication of frames. This can consume network bandwidth and cause delays in delivering packets to their destinations and also put unnecessary load on network devices.
c. MAC Address Table Instability: Switches maintain MAC address tables to determine the appropriate port for forwarding frames. In a looping scenario, switches may continuously update their MAC address tables as frames circulate through the network, leading to instability and incorrect forwarding decisions.

**B.**

Using spanning tree algorithm to shut off some ports so that the resulting topology is a loop-free tree.