01-switching
==========

My name: 郑凯琳 TAY KAI LIN

My Student ID: 205220025

This lab took me about 5 hours to do.

Implementation Explanation:

(一) `iface_info_t *lookup_port(u8 mac[ETH_ALEN])`

<u>功能：</u>
在 `mac_port` 表中查找给定 MAC 地址的接口信息

<u>思路：互斥锁 & 遍历哈希桶和链表</u>
1. 获取 `mac_port_map` 的互斥锁，以确保在查找过程中表的一致性。
2. 循环遍历 `mac_port_map` 中的所有哈希桶（0 到 HASH_8BITS-1）。
3. 在每个哈希桶中，使用 `list_for_each_entry` 宏遍历链表中的每个条目。
4. 对于每个条目，使用 `memcmp` 函数比较条目的 MAC 地址和给定的 MAC 地址。
5. 如果找到匹配的条目，则释放互斥锁并返回条目对应的接口信息。
6. 如果在整个表中未找到匹配的条目，则释放互斥锁并返回 `NULL`，表示未找到匹配的接口信息。

(二) `void insert_mac_port(u8 mac[ETH_ALEN], iface_info_t *iface)`

<u>功能：</u>
将 MAC 地址和接口信息插入到 `mac_port` 表中

<u>思路：互斥锁 & 哈希表</u>
1. 获取 `mac_port_map` 的互斥锁，以确保在插入过程中表的一致性。
2. 分配一个新的 `mac_port_entry_t` 结构体，并进行初始化。
3. 获取当前时间，并将其设置为新条目的 `visited` 字段。
4. 使用 `memcpy` 函数将给定的 MAC 地址复制到新条目的 `mac` 字段。
5. 使用 `calculate_hash` 函数计算给定 MAC 地址的哈希值。
6. 使用 `list_add_tail` 函数将新条目插入到对应哈希桶的链表末尾。
7. 释放互斥锁，完成插入操作。

(三) `int sweep_aged_mac_port_entry()`

<u>功能：</u>
清理 `mac_port` 表中那些在过去 30 秒内没有访问过的条目

<u>思路：互斥锁 & 遍历链表</u>
1. 获取 `mac_port_map` 的互斥锁，以确保在清理过程中表的一致性。

2. 初始化一个变量 `num` 用于计算被清理的条目数量。
3. 获取当前时间。
4. 对于每个哈希桶，使用 `list_for_each_entry_safe` 宏遍历链表中的条目。
5. 检查每个条目的 `visited` 字段与当前时间的差值是否大于 `MAC_PORT_TIMEOUT` ，即是否超过了 30 秒。
6. 如果是超过 30 秒未访问的条目，则从链表中删除该条目，释放其内存，并将 `num` 计数加一。
7. 释放互斥锁，完成清理操作。
8. 返回被清理的条目数量。

(四) `void handle_packet(iface_info_t *iface, char *packet, int len)`

<u>功能：</u>
处理接收到的数据包

<u>思路：查找目标 MAC 地址并进行转发或广播 & 更新源 MAC 地址和接口的映射关系</u>
1. 解析数据包的以太网头部，获取目标 MAC 地址。
2. 使用 `lookup_port` 函数在 `mac_port` 表中查找目标 MAC 地址对应的接口。
3. 如果找到了目标接口(`tx_iface`)，则使用该接口将数据包转发出去，调用 `iface_send_packet`函数。
4. 如果没有找到目标接口，则表示该数据包需要进行广播，调用 `broadcast_packet` 函数将数据包从接收接口广播出去。
5. 检查源 MAC 地址是否存在于 `mac_port` 表中。
6. 如果源 MAC 地址不存在于表中，则将源 MAC 地址和接口信息插入到 `mac_port` 表中，调用 `insert_mac_port` 函数。
7. 记录日志，输出目标 MAC 地址信息。
8. 释放数据包的内存。

Screenshots:

```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2
h2 -> h1
*** Results: 0% dropped (2/2 received)
mininet> h1 ping -c 2 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=1.08 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.102 ms

--- 10.0.0.2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1013ms
rtt min/avg/max/mdev = 0.102/0.589/1.077/0.487 ms
mininet> h1 ping -c 3 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.168 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.072 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.083 ms

--- 10.0.0.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2027ms
rtt min/avg/max/mdev = 0.072/0.107/0.168/0.042 ms
mininet>
```

```
*** Starting CLI:
mininet> h1 ping -c 4 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=4.87 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.390 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.131 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.050 ms

--- 10.0.0.2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3040ms
rtt min/avg/max/mdev = 0.050/1.360/4.869/2.029 ms
mininet> h1 ping -c 5 h2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.064 ms
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.075 ms
64 bytes from 10.0.0.2: icmp_seq=3 ttl=64 time=0.046 ms
64 bytes from 10.0.0.2: icmp_seq=4 ttl=64 time=0.091 ms
64 bytes from 10.0.0.2: icmp_seq=5 ttl=64 time=0.081 ms

--- 10.0.0.2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4076ms
rtt min/avg/max/mdev = 0.046/0.071/0.091/0.015 ms
mininet>
```

Remaining Bugs: