Network System Capstone
Homework 4
Report

For OSPF:

1. Show how you implement the flooding algorithm. (Do not just use direct transmission from all nodes to all other nodes) (10%)
   首先先創造一個Class OSPF_Router來模擬各個Router:

```python
class OSPF_Router:
    def __init__(self, num_router):
        self.rcv_buffer = {}
        self.fwd_buffer = {}
        self.link_state = []
        self.rcv_from_router = [False for i in range(num_router)]
```

   接下來用迴圈來跑，迴圈內主要分為兩個階段:

   a. 如果neighbor還沒有收過自己rcv_buffer的link_state data的話，就把自己rcv_buffer的link_state data送給還沒收過的neighbor

```python
do_iter = False
#! send my link state to my neighbor if it does not receive my link_state info yet
for src in range(n):
    neighbors = [neighbor for neighbor, cost in enumerate(link_cost[src]) if (cost != 999 and neighbor != src)]
    for buf_data in routers[src].fwd_buffer:
        # print("src:", src, " here!! ", buf_data)
        for neighbor in neighbors:
            if not routers[neighbor].rcv_from_router[buf_data]:
                routers[neighbor].rcv_buffer[buf_data] = link_cost[buf_data]
                routers[neighbor].rcv_from_router[buf_data] = True
                logs.append((src, buf_data, neighbor))
```

   b. 確定每個router有沒有都收過彼此的link_state，如果都有就結束，沒有就再來一次

```python
#! move buffer and examine if all routers get link_state info from others
for i in range(n):
    routers[i].fwd_buffer.clear()
    routers[i].fwd_buffer = copy.deepcopy(routers[i].rcv_buffer)
    # print("router", i, " rcv_buffer", routers[i].rcv_buffer, " fwd_buffer", routers[i].fwd_buffer)
    routers[i].rcv_buffer.clear()
    # print("router", i, " rcv_buffer", routers[i].rcv_buffer, " fwd_buffer", routers[i].fwd_buffer)
    if not all(routers[i].rcv_from_router):
        # print("router", i, "still has false.....")
        do_iter = True
if not do_iter:
    break
```

   離開迴圈後，每個router都計算最短路徑(用dijkstra, 我也有寫用bellman的):

```python
path_cost = []
for i in range(n):
    path_cost.append(routers[i].link_state)
#! do shortest path calculation
#? Dijastra
all_min_path = []
for src in range(n):
    min_path = [float('inf')] * n
    min_path[src] = 0
    heap = [(0, src)]
    visited = set()
    while heap:
        cost, router = heapq.heappop(heap)
        visited.add(router)
        for neighbor in range(len(path_cost[router])):
            cost = path_cost[router][neighbor]
            if (cost != 999) and (neighbor not in visited):
                new_cost = min_path[router] + cost
                if new_cost < min_path[neighbor]:
                    min_path[neighbor] = new_cost
                    heapq.heappush(heap, (new_cost, neighbor))
    all_min_path.append(min_path)
return_tuple = (all_min_path, logs)
#? Bellman-Ford
# for src in range(n):
#     for j in range(n-1):
#         for node in range(n):
#             for neighbor in range(len(path_cost[node])):
#                 cost = path_cost[node][neighbor]
#                 # print("Router:", node, " src:", src, " neighbor:", neighbor, " neighbor state:", neighbor_state)
#                 # for dst in range(len(neighbor_state)):
#                 #     cost = neighbor_state[dst]
#                 if path_cost[src][neighbor] > path_cost[src][node] + cost:
#                     # print(routers[node].link_state[neighbor], routers[node].link_state[src],cost)
#                     # print("update!")
#                     path_cost[src][neighbor] = path_cost[src][node] + cost
# return_tuple = (path_cost, logs)
```

2. What factor will affect the convergence time of OSPF? (10%)
    a. 網路拓撲：網路中的拓撲結構可以影響 OSPF 的收斂時間。當網路拓撲復雜時，收斂時間可能會更長，因為需要更多的時間來傳遞和處理 LSA信息。
    b. 鏈路狀態廣告（LSA）更新：OSPF 使用 LSA 更新來維護網路中的路由表。當 LSA 更新頻繁時，收斂時間可能會更長，因為路由器需要花更多的時間處理和應用 LSA。
    c. OSPF 區域：OSPF 區域的大小也會影響收斂時間。在大型區域中，收斂時間可能會更長，因為需要更多的時間來傳遞 LSA。
    d. OSPF 路由器性能：路由器性能也可能影響 OSPF 的收斂時間。當路由器處理能力較弱時，可能需要更長的時間來處理 LSA 更新。
    e. 網路負載：當網路負載較重時，OSPF 的收斂時間可能會更長，因為路由器需要處理更多的流量和 LSA 更新。

For RIP:

1. Show how you implement the distance vector exchange mechanism. (10%)
    首先先創造一個Class RIP_Router來模擬各個Router:

```python
class RIP_Router:
    def __init__(self, num_router):
        self.link_state = []
        self.old_link_state = []
        self.rcv_buffer = {}
```

接下來用迴圈來跑，迴圈內主要分為三個階段:

a. 如果自己的link_state有變的話，就把自己的link_state data送給所有neighbor

```python
#! send my link state to my neighbor if my link_state change
for src in range(n):
    neighbors = [neighbor for neighbor, cost in enumerate(link_cost[src]) if (cost != 999 and neighbor != src)]
    for neighbor in neighbors:
        if (routers[src].link_state != routers[src].old_link_state):
            routers[neighbor].rcv_buffer[src] = copy.deepcopy(routers[src].link_state)
            logs.append((src, neighbor))
#! copy all routers' link_state to old_link_state
for i in range(n):
    routers[i].old_link_state.clear()
    routers[i].old_link_state = copy.deepcopy(routers[i].link_state)
```

b. 算最短路徑

```python
#! do shortest path calculation
for node in range(n):
    for neighbor, neighbor_state in routers[node].rcv_buffer.items():
        for dst in range(len(neighbor_state)):
            cost = neighbor_state[dst]
            if routers[node].link_state[dst] > routers[node].link_state[neighbor] + cost:
                # print(routers[node].link_state[neighbor], routers[node].link_state[src],cost)
                # print("update src:", node, " neighbor:", neighbor, " dst:", dst)
                routers[node].link_state[dst] = routers[node].link_state[neighbor] + cost
```

c. 確定每個router的link_state有沒有變化，如果都沒有就結束，有的話就再來一次

```python
#! examine if all routers' link_state change or not
for i in range(n):
    if routers[i].old_link_state != routers[i].link_state:
        do_iter = True
#! clear buffer
for i in range(n):
    routers[i].rcv_buffer.clear()
```

2. What factor will affect the convergence time of RIP? (10%)
跟ospf大同小異:

a. 網絡規模：網絡規模是影響RIP收斂時間的一個重要因素。當網絡規模較大時，RIP 協議需要更多的時間來計算distance vector並傳遞信息，因此收斂時間可能會更長。

b. Router計算能力：RIP協議需要router進行計算和轉發信息，因此router計算能力的差異會影響RIP收斂時間。如果router計算能力較弱，則可能需要更長時間計算distance vector。
c. 更新頻率：RIP中路由表的更新頻率也會影響收斂時間。當路由表更新頻繁時，RIP需要更多的時間來傳遞信息和計算distance vector，因此收斂時間可能會更長。
d. 網絡拓撲：網絡拓撲結構也會影響 RIP的收斂時間。當網絡拓撲結構複雜時，RIP需要更多的時間來計算distance vector並傳遞信息，因此收斂時間可能會更長。

e. 丟包率：在packet傳輸過程中出現的丟包情況會對RIP的收斂時間產生影響。丟包會導致路由表信息傳遞失敗，從而導致RIP重新計算distance vector並傳遞信息，從而增加了收斂時間。