FLUERE

Traffic Routing Optimization with SDN and Fog Computing

Presented by Jishuo Yang & Yuqi Li

Abstract

Explore the integrated application of Software-Defined Networking (SDN) and Fog Computing in urban traffic management systems, specifically in the areas of traffic flow optimization and rapid response to emergency vehicles.

Objectives

Optimize emergency vehicle routes using SDN & fog computing

Efficient Routing

Ensure emergency vehicles can navigate city traffic with minimized delays

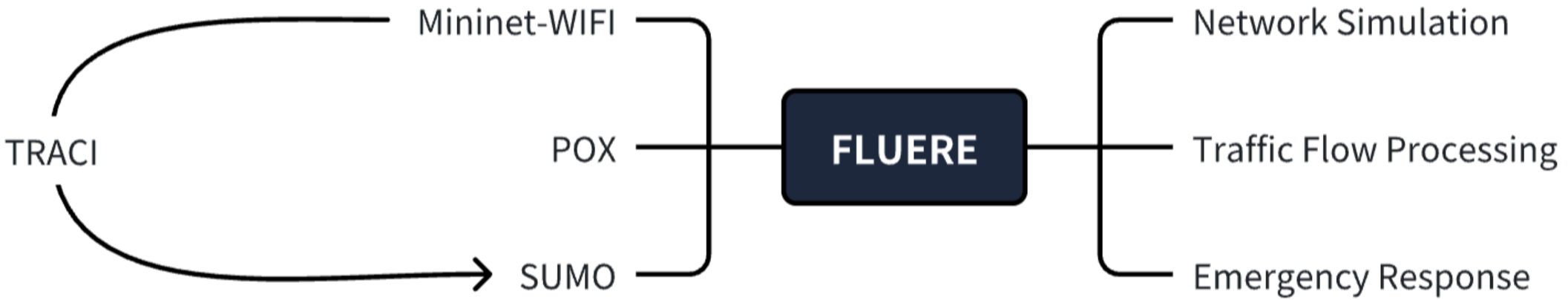
Flow Optimization

Improve overall urban traffic flow through intelligent rerouting based on real-time conditions.

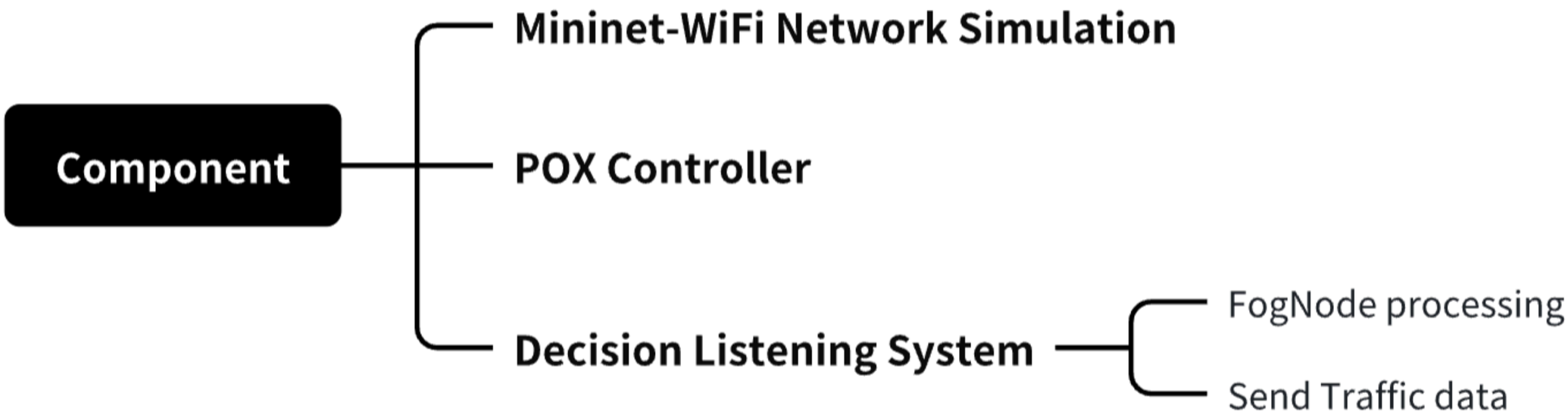
Reduced Response Time

Reduce the time it takes emergency services to reach their destinations.

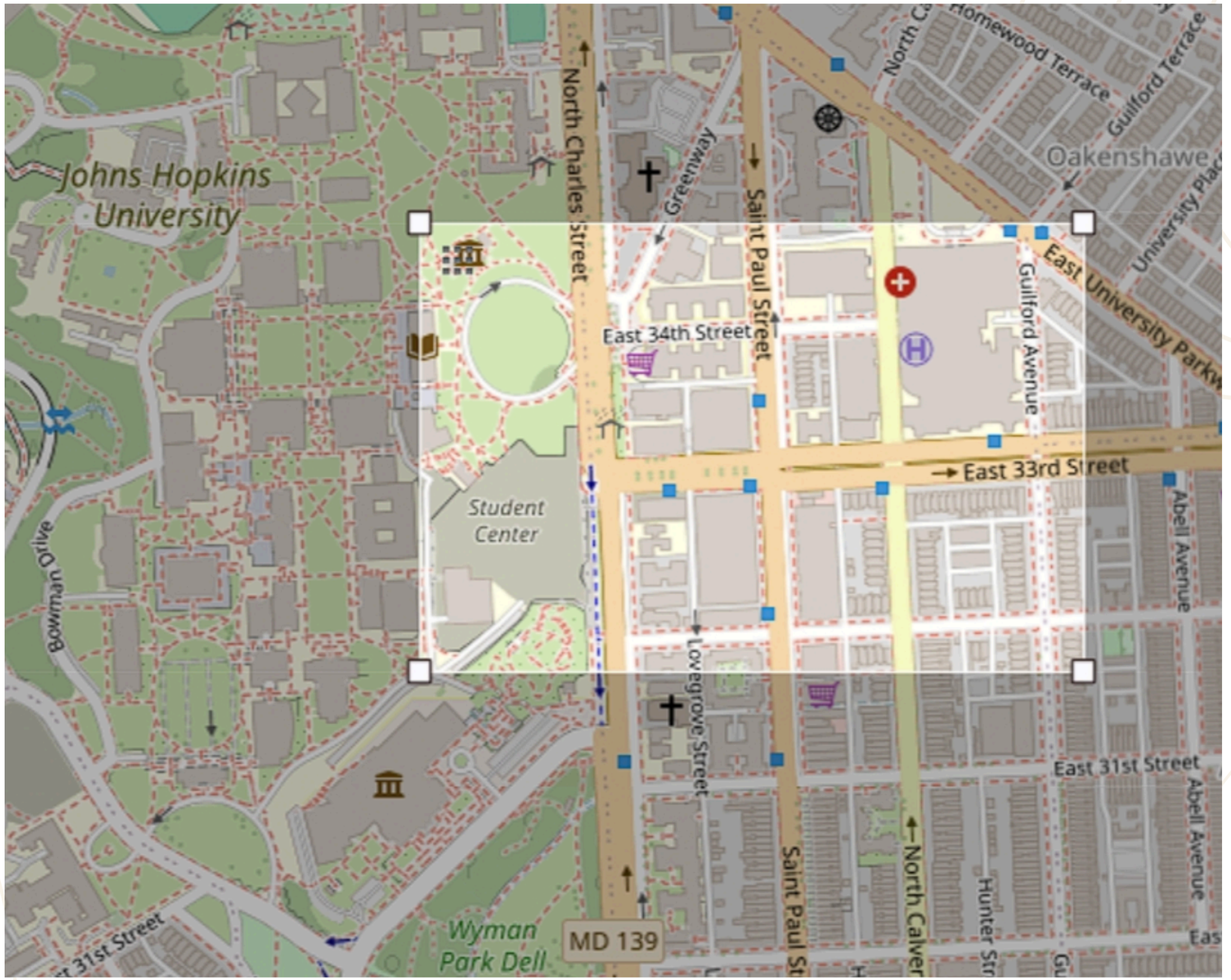
Overall Architecture



Component Breakdown



Network Simulation



```python
def create_network():
    net = Mininet_wifi(controller=Controller, accessPoint=OVSKernelAP)

    info("*** Creating nodes\n")
    ap1 = net.addAccessPoint('ap1', ssid='ap1-ssid', mode='g', channel='1', position='50,50,0', range=100)
    ap2 = net.addAccessPoint('ap2', ssid='ap2-ssid', mode='g', channel='2', position='100,50,0', range=100)
    ap3 = net.addAccessPoint('ap3', ssid='ap3-ssid', mode='g', channel='3', position='150,50,0', range=100)

    fogNode1 = net.addHost('fogNode1', ip='10.0.0.10')
    fogNode2 = net.addHost('fogNode2', ip='10.0.0.11')

    for i in range(1, 21):
        net.addStation(f'sta{i}', ip=f'10.0.0.{i}', position=f'{30 + i*10},60,0')

    emergency1 = net.addStation('emergency1', ip='10.0.0.101', position='10,20,0')
    emergency2 = net.addStation('emergency2', ip='10.0.0.102', position='20,20,0')

    info("*** Configuring wifi nodes\n")
    net.configureWifiNodes()

    info("*** Adding controller\n")
    net.addController('c0')

    info("*** Creating links\n")
    net.addLink(ap1, fogNode1)
    net.addLink(ap2, fogNode1)
    net.addLink(ap3, fogNode2)

    for i in range(1, 21):
        net.addLink(ap1 if i <= 7 else ap2 if i <= 14 else ap3, f'sta{i}')
    net.addLink(ap1, emergency1)
    net.addLink(ap2, emergency2)
```

```
ap1-eth2<->fogNode-eth0 (OK OK)
sta1-wlan0<->wifi (use iw/iwconfig to check connectivity)
sta2-wlan0<->wifi (use iw/iwconfig to check connectivity)
sta3-wlan0<->wifi (use iw/iwconfig to check connectivity)
emergency1-wlan0<->wifi (use iw/iwconfig to check connectivity)
emergency2-wlan0<->wifi (use iw/iwconfig to check connectivity)
mininet-wifi>
```

POX

- Dijkstra Algorithm: Write a function that uses the Dijkstra algorithm to find the shortest path between two nodes.
- Receive Decisions: Upon receiving an external decision regarding emergency vehicles, use the Dijkstra algorithm to find the path.
- Update Flow Rules: Based on the path found using the Dijkstra algorithm, use OpenFlow rules to clear the route for the emergency vehicle.

```
def dijkstra(self, graph, start, goal):
    queue = []
    heapq.heappush(queue, (0, start))
    distances = {start: 0}
    predecessors = {start: None}

    while queue:
        (current_distance, current_node) = heapq.heappop(queue)

        if current_node == goal:
            path = []
            while current_node is not None:
                path.insert(0, current_node)
                current_node = predecessors[current_node]
            return path

        for neighbor, weight in graph.get(current_node, []):
            distance = current_distance + weight
            if neighbor not in distances or distance < distances[neighbor]:
                distances[neighbor] = distance
                predecessors[neighbor] = current_node
                heapq.heappush(queue, (distance, neighbor))

    return []
```

Data Transmission Processing

```
def start_server():
    host = ''
    port = 5000
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((host, port))
        s.listen()
        print('Server started. Waiting for connection...')
        while True:
            conn, addr = s.accept()
            with conn:
                print('Connected by', addr)
                while True:
                    data = conn.recv(1024)
                    if not data:
                        break
                    vehicle_data = data.decode()
                    vehicle_count, is_emergency = map(int, vehicle_data.split(","))
                    decision = process_traffic_data(vehicle_count, is_emergency)
                    print("Decision:", decision)
```

```
wifi@wifi-virtualbox:~/Desktop/sdnfinal/data_process$ su
Server started. Waiting for connection...
Connected by ('127.0.0.1', 59476)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59478)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59480)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59482)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59484)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59486)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59500)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59506)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59514)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59516)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59518)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59520)
Decision: Allow Traffic
Connected by ('127.0.0.1', 59522)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59524)
Decision: Clear Path for Emergency Vehicle
Connected by ('127.0.0.1', 59526)
Decision: Clear Path for Emergency Vehicle
```

Analysis

Optimize emergency vehicle routes using SDN & fog computing

Emergency Handling Time

Average Handling Time: handle all emergency events to gain a general

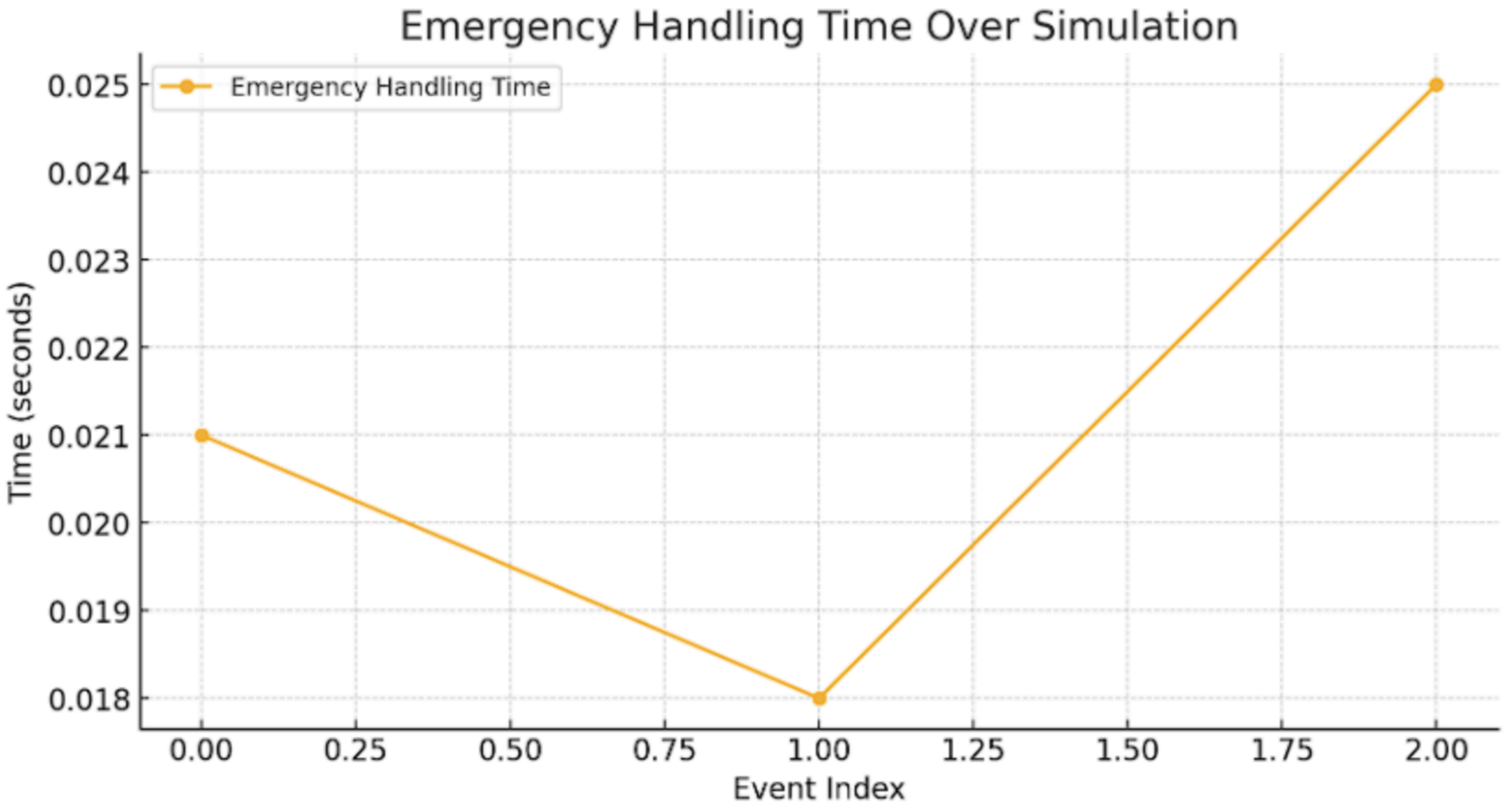
Longest and Shortest Handling Time: Identify the quickest and slowest handled events

Vehicle Notification Time

Average Notification Time
execute all notification operations, reflecting the system's speed in communicating emergency information to other vehicles

Path Length

Average Path Length: The average path length calculated for emergency events can be used to analyze the efficiency of the path selection.


Emergency Handling Time Over Simulation
Emergency Handling Time
Time (seconds)
0.025
0.024
0.023
0.022
0.021
0.020
0.019
0.018
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Event Index

Tech Stack

SUMO

Simulation of Urban MObility

OpenStreetMap

Database of digital map

TraCI

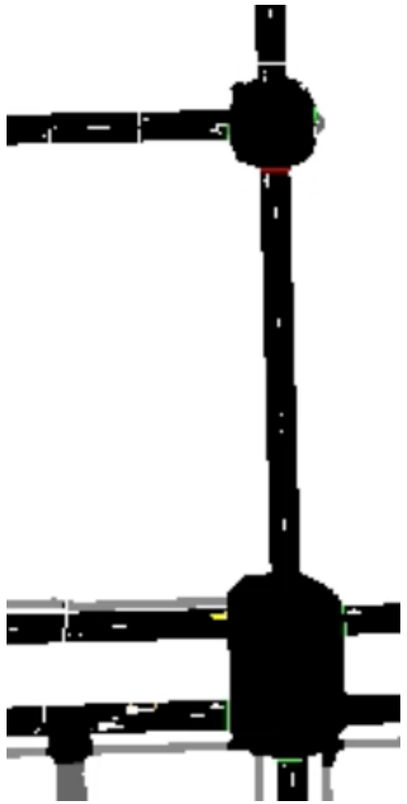
Communication for MNWIFI & SUMO by simulating objects and manipulating behaviors

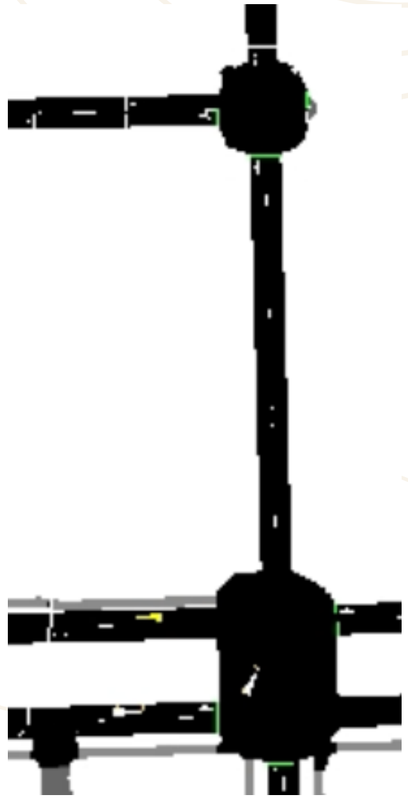
Emergency Response

LcKeepRight

ChangeLane

Traffic Light





SetUp

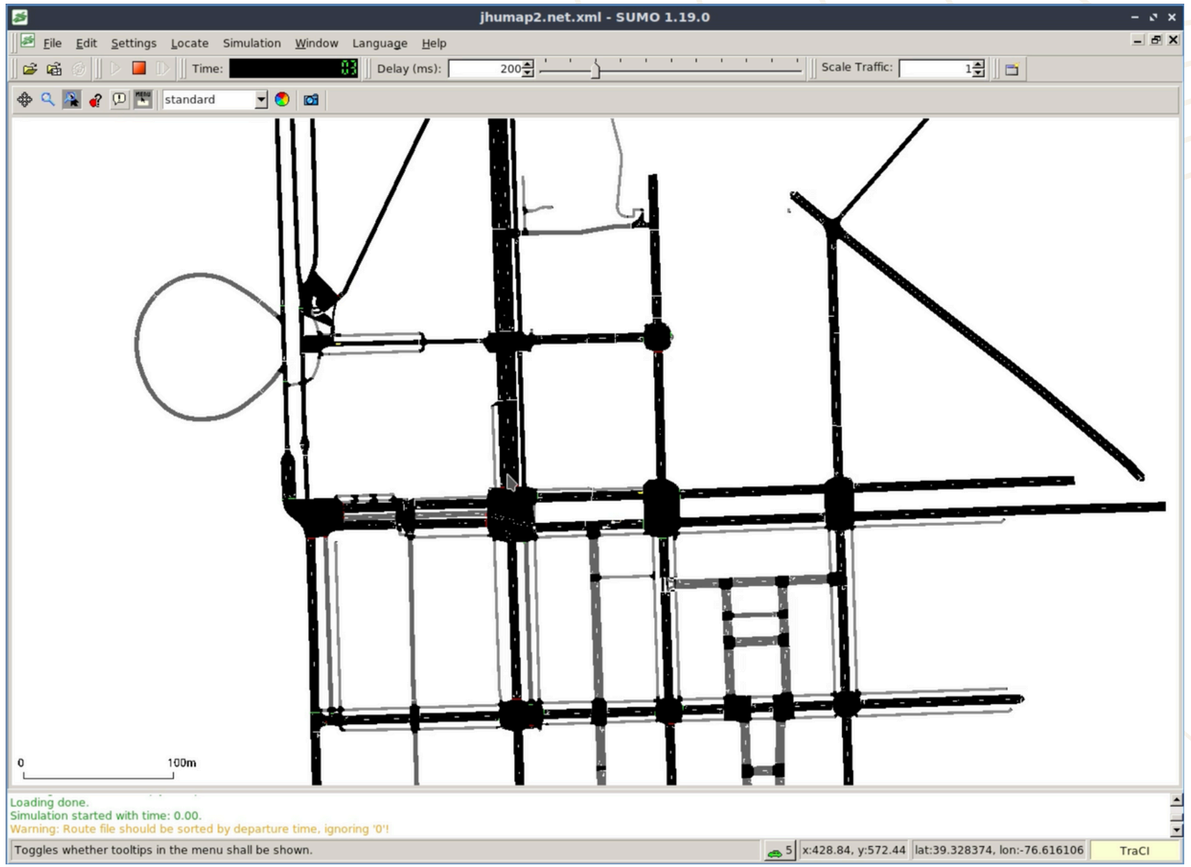
100 cars

2 ambulances

RandomTrips.py

net.xml for map

rou.xml for nodes



Analysis

```
jsy@JsYs-MacBook-Pro result % /usr/local/bin/python3 /Users/jsy/Desktop/result/analysis.py
Comparison between 'traffic_data_without.csv' (original) and 'traffic_data.csv' (modified):

Average Speed:
Original: 11.221935311480722
Modified: 11.26842827233788
Percentage increase: 0.41430430283796876 %

Average Occupancy:
Original: 0.012611423194734815
Modified: 0.025286301077326594
Percentage increase: 100.5031524743651 %
jsy@JsYs-MacBook-Pro result %
```

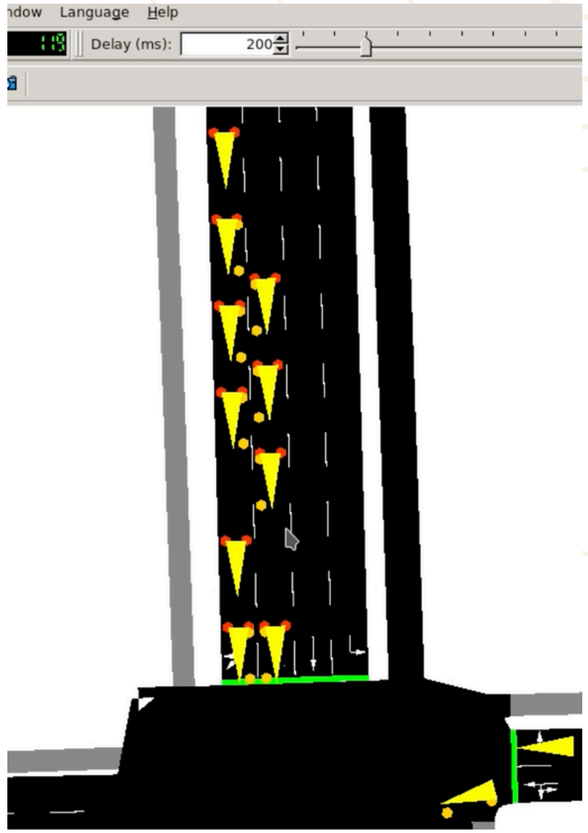
The traffic load is increased and speed is slightly faster.

The ambulance took 1/2 of the original time to finish all the routes.

What to work on next

Better Logic to handle the traffic

Better Connection between MNwifi & SUMO



Thank You

Q&A?