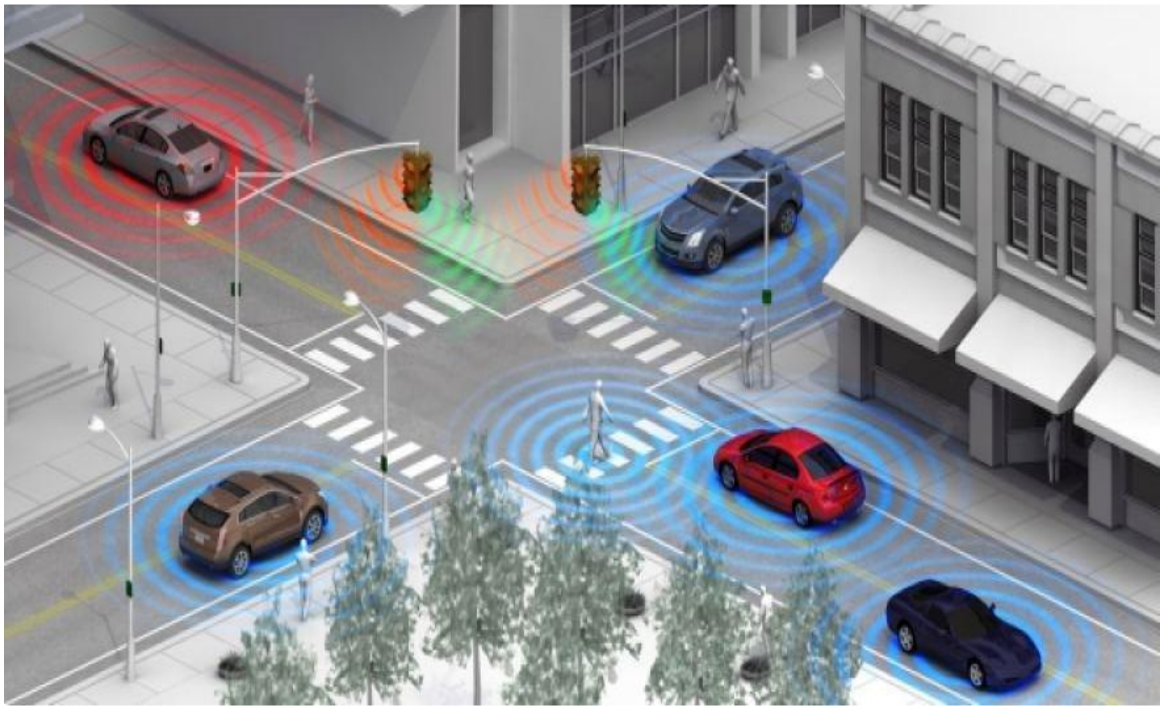


ConnectiCab

A MOBILE TAXI NETWORK DESIGN FOR P2P NETWORKING

CPE 400 - 12/07/16

Team Members:

Brett Knadle

Randy Gonzalez

Phillip Vong

Table of Content

Abstract

Taxi cabs are not communicating as effectively as they could be. In order to help facilitate communication between taxi’s, a P2P network system would be ideal as it is most efficient. In order to maximize throughput for the P2P network an optimal routing algorithm was implemented. The algorithm will take into account Bluetooth and Wi-Fi as primary sources for broadcast, as well as the distance between each taxicab in order to create an optimal P2P network.

Introduction

ConnectiCab presents a peer-to-peer network communication between taxi cabs on city streets. Each taxicab is equipped with Wi-Fi and Bluetooth technology to communicate amongst each other. In order to optimize the transfer of packets within the network, bluetooth is used when available because it is much faster than wifi, however because bluetooth has a very limited range wifi is used most of the time within the network. ConnectiCab offers a solution to a peer-to-peer taxi network by showing an implementation as well as a graphic simulation of how an example city would look. The simulation provides a visual aid to present how a packet is transferred along the network from a source taxi to its destination taxi.

Dependencies

All code was tested using Ubuntu 16.04. Following must be installed on machine:

- gcc/g++ compiler:

```
-$ sudo apt-get install g++
```

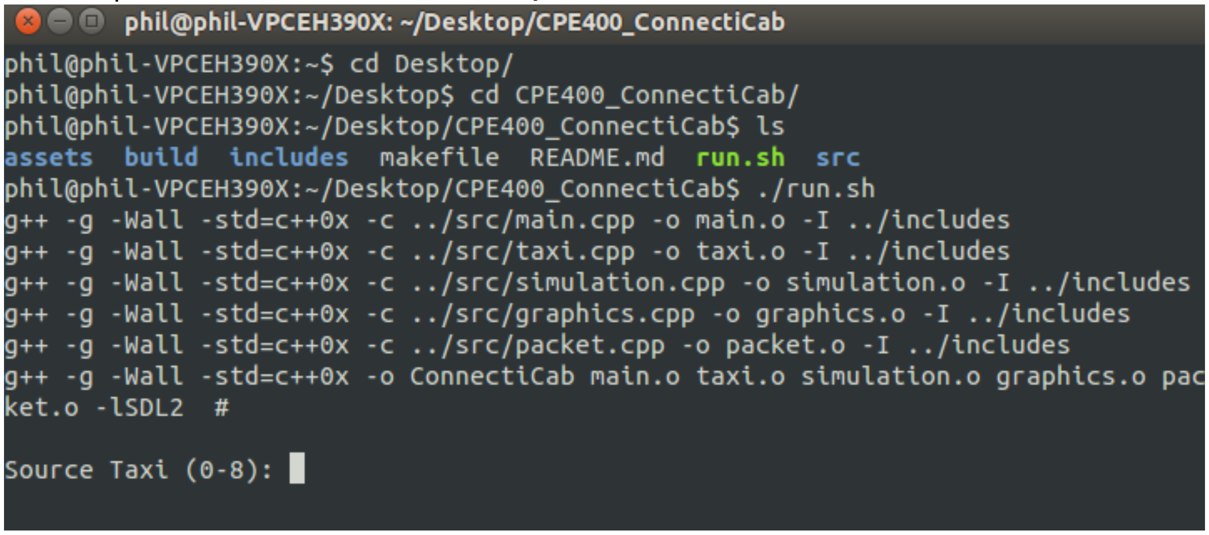
- SDL2 : used for graphical interface and event handling.

```
-$ sudo apt-get install libsdl2-2.0
```

Compilation

Two methods for compilation:

1. Run the provided bash file in terminal: **./run.sh**

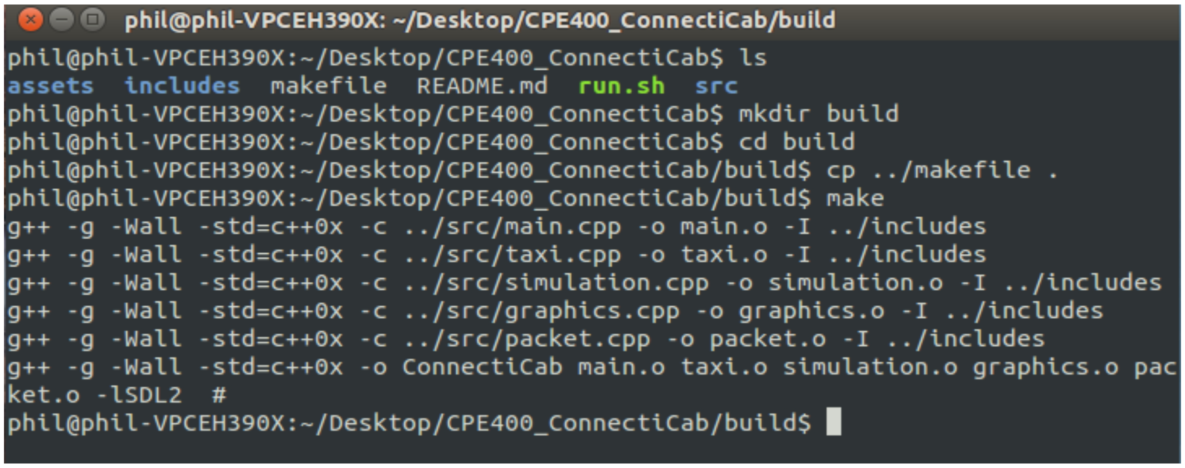


```
phil@phil-VPCEH390X: ~/Desktop/CPE400_ConnectiCab
phil@phil-VPCEH390X:~$ cd Desktop/
phil@phil-VPCEH390X:~/Desktop$ cd CPE400_ConnectiCab/
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab$ ls
assets  build  includes  makefile  README.md  run.sh  src
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab$ ./run.sh
g++ -g -Wall -std=c++0x -c ../src/main.cpp -o main.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/taxi.cpp -o taxi.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/simulation.cpp -o simulation.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/graphics.cpp -o graphics.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/packet.cpp -o packet.o -I ../includes
g++ -g -Wall -std=c++0x -o ConnectiCab main.o taxi.o simulation.o graphics.o pac
ket.o -lSDL2  #

Source Taxi (0-8): 
```

Fig 1: Compile and run with method 1, already included the makefile context inside the bash script.

2. Use Makefile:

````bash

**mkdir build**

**cd build**

**cp ../makefile .**

**make**

**./ConnectiCab**

``````



```
phil@phil-VPCEH390X: ~/Desktop/CPE400_ConnectiCab/build
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab$ ls
assets  includes  makefile  README.md  run.sh  src
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab$ mkdir build
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab$ cd build
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab/build$ cp ../makefile .
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab/build$ make
g++ -g -Wall -std=c++0x -c ../src/main.cpp -o main.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/taxi.cpp -o taxi.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/simulation.cpp -o simulation.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/graphics.cpp -o graphics.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/packet.cpp -o packet.o -I ../includes
g++ -g -Wall -std=c++0x -o ConnectiCab main.o taxi.o simulation.o graphics.o pac
ket.o -lSDL2  #
phil@phil-VPCEH390X:~/Desktop/CPE400_ConnectiCab/build$ 
```

Fig 2: Compiled successful with method 2, manually create a build folder and copy makefile then make command to compile the program.

Overview of Code

- **XML File:** A configuration file is provided under the assets folder. The XML file specifies the ranges for Wi-Fi and Bluetooth broadcast for all taxis, as well as each taxi's start and end coordinates.

- **Main:** The main program opens and reads the configuration file to start building the data structure for the taxi. All taxi information is fed onto the simulation. Main program runs the simulation to display on to the SDL screen. Program exits after SDL window is closed and all memory allocated returns back to the system.

- **Taxi Class:** The taxi class serves as a data structure to hold all the information from configuration file. Each taxi has source/destination coordinates and a speed value. All taxis have the same broadcast radius for transmitting packets via Wi-Fi or Bluetooth. One specific taxi holds a packet for the network.

- **Simulation Class:** This class is responsible for the routing algorithm. Each taxi is initialized and loaded onto the simulation. The simulation takes taxi information as parameter, updating the location of each taxi with respect to their start coordinates, destination, speed, and packet. The routing algorithm keeps track of the source taxi containing the packet and the desired destination taxi. Taxi's communicate with each other when their distances are within range of broadcast with either Wi-Fi or Bluetooth. The packet is transmitted through Wi-Fi or Bluetooth communication and is routed to its specified destination taxi.

- **Packet Class:** The packet serves as message that is being transmitted between taxis. The packet contains the information of the desired destination taxi. The packet is distributed via Wi-Fi and/or Bluetooth broadcast if source taxi is within range of another taxi. The routing algorithm successfully transmits the packet to its specified destination at the end of the simulation.

- **Graphics Class:** The graphics class handles the entire graphical interface. Each taxi is drawn on screen and represented as a yellow square. The red taxis represent the taxis that contain the packet. A line between two squares indicates that the two taxis have established a communication link via Wi-Fi and/or Bluetooth. The class creates a visual representation of the implemented routing protocol showing the taxis move and transmit the message to its destination as it updates every second.

Functionality of Protocol

The routing algorithm implemented assumes a traffic-free network with a single packet to distribute. Broadcast is achieved through Wi-Fi or Bluetooth when two or more taxis are within the radius range. Prior to executing the protocol, the user is prompted for the packet source and destination. The packet source is assigned to a single taxi at the beginning of the program and broadcasted among neighboring taxis until the packet reaches its destination taxi.

Realistic implementations of this protocol would allow taxis to communicate their destination or source with other taxi's. This would improve taxi coverage within a city and make the taxi network more efficient.

Examples of error handling are based on if a taxi isn't in range of the broadcast for long enough it won't receive the packet and the program should return a failure. Also, if no route is found between two taxis a failure is returned that no route was found.

Deviation from existing research

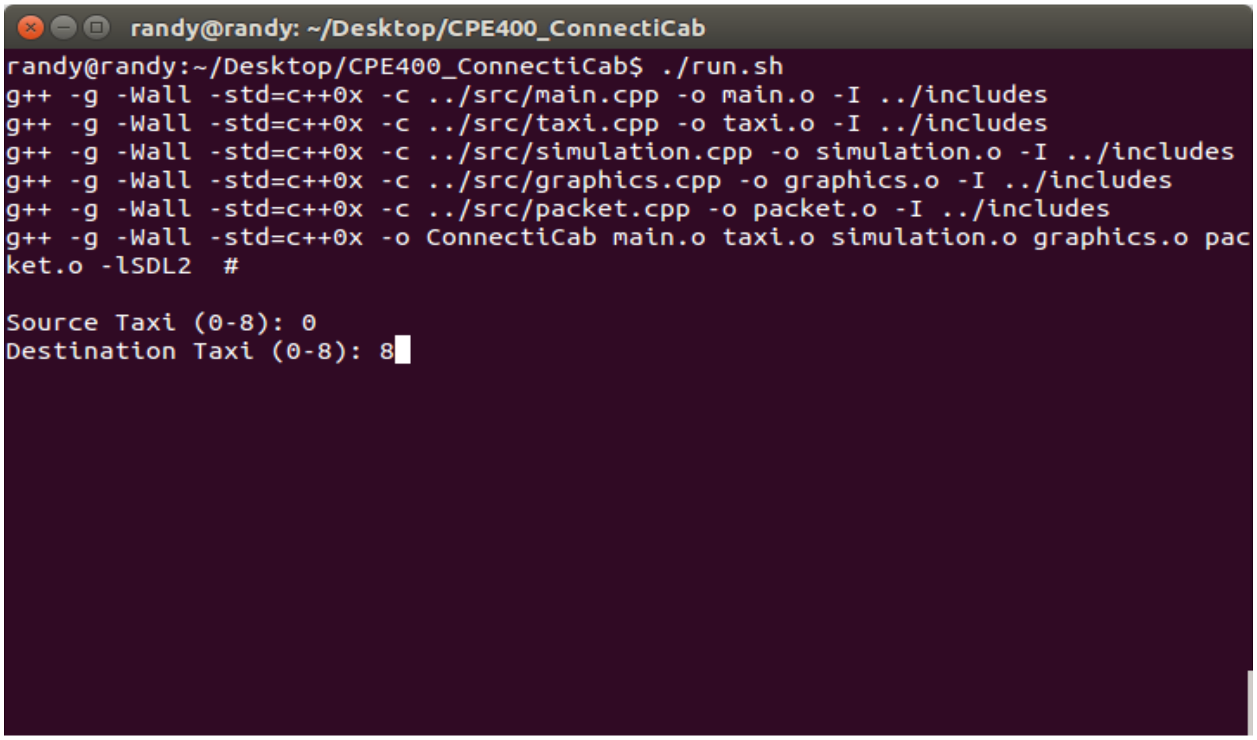
Developed & Implemented functionality:

- A routing algorithm was implemented that allows a packet to broadcast to neighboring taxicabs that are in range with either Wi-Fi or Bluetooth only.
- Graphical Interface was achieved that integrated a dynamic P2P network similar to a real world scenario.
- Each taxi was able to broadcast the received packet to every other neighboring nodes within range of transfer.
- User may choose the source and destination of the packet to be delivered.
- Logging feature in terminal displays all of broadcasting transactions between the taxi cabs and the connections established through a broadcast source. Program notifies the user that the packet has reached its destination taxi.

Possible Future advancements:

- Take into account network traffic by allowing multiple packets to be transferred to multiple locations.
- Allow the user to specify the packet size to differentiate the broadcast rate between Wi-Fi and Bluetooth.
- Add LTE broadcasting capability allowing the taxi cabs to transfer the packet at a longer range than Wi-Fi but at a lower throughput rate.

Results and Analysis



randy@randy: ~/Desktop/CPE400_ConnectiCab

```
randy@randy:~/Desktop/CPE400_ConnectiCab$ ./run.sh
g++ -g -Wall -std=c++0x -c ../src/main.cpp -o main.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/taxi.cpp -o taxi.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/simulation.cpp -o simulation.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/graphics.cpp -o graphics.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/packet.cpp -o packet.o -I ../includes
g++ -g -Wall -std=c++0x -o ConnectiCab main.o taxi.o simulation.o graphics.o pac
ket.o -lSDL2  #

Source Taxi (0-8): 0
Destination Taxi (0-8): 8
```

Fig 3: Program prompting user to input the 'Source Taxi (0-8)' and 'Destination Taxi (0-8)' to simulate a routing algorithm from a source to a destination.

In **figure 3**, the user is prompted to specify the packet source and destination. Integer values range from 0 to 8. After user input, the program begins with a display screen as shown in **figure 4**.

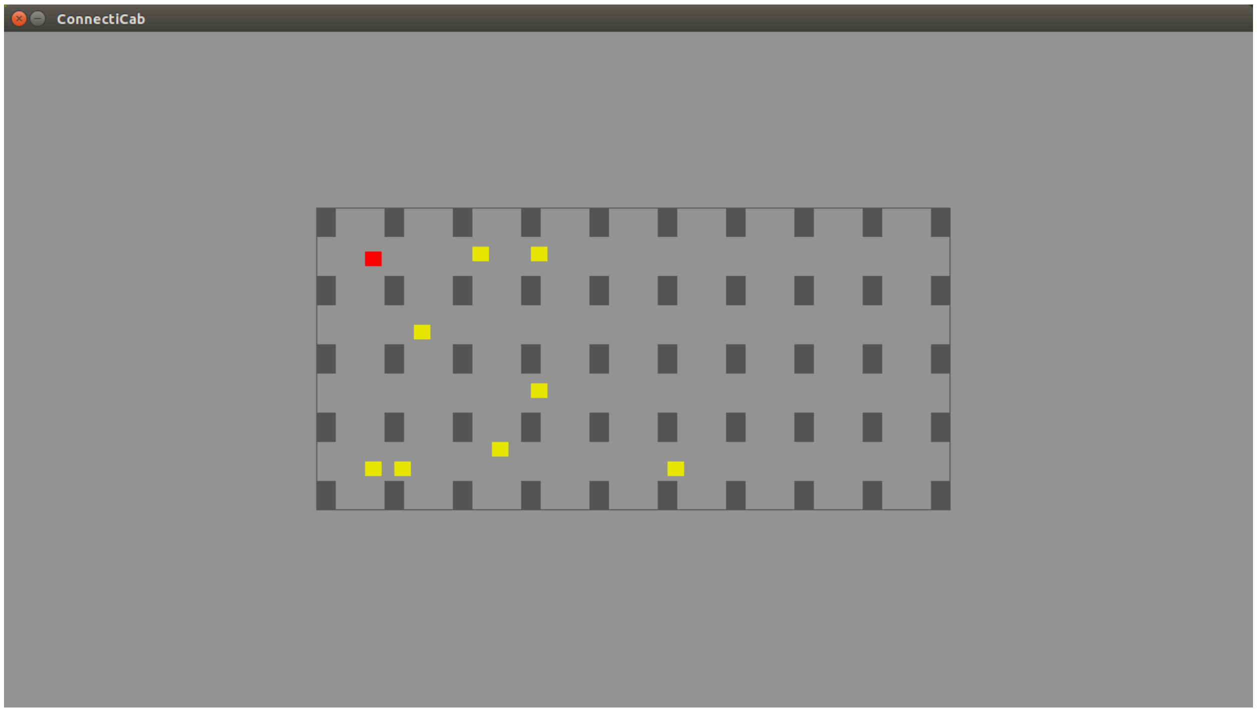


Fig 4: A graphical interface was implemented to visualize a P2P network of taxis in a city street scenario. Red boxes represent the source taxi with a packet containing the destination taxi information. Yellow squares represent neighboring taxis in the network.

In **figure 4**, all taxis are depicted on screen as yellow/red blocks. The image is the initial state of all taxis when the program executed. In this particular case the user has entered the source taxi to be the "0"th taxi in the network. As one can see the source taxi is represented as a red block. When the program continues, every taxi in the network will update its current location by one second. As the taxi aim to reach their destination goal the packet is transferred via Bluetooth or Wi-Fi whenever the source taxi is within range. The packet gets routed throughout the network until it has successfully reached its desired destination taxi.

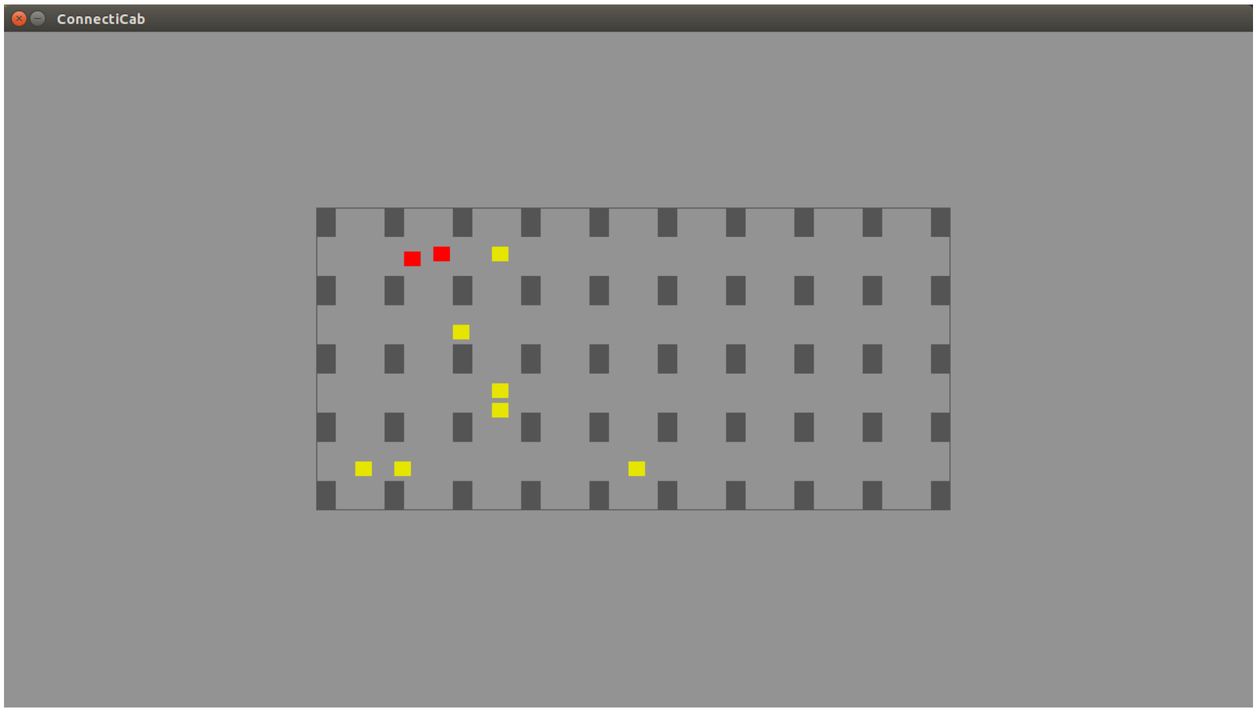
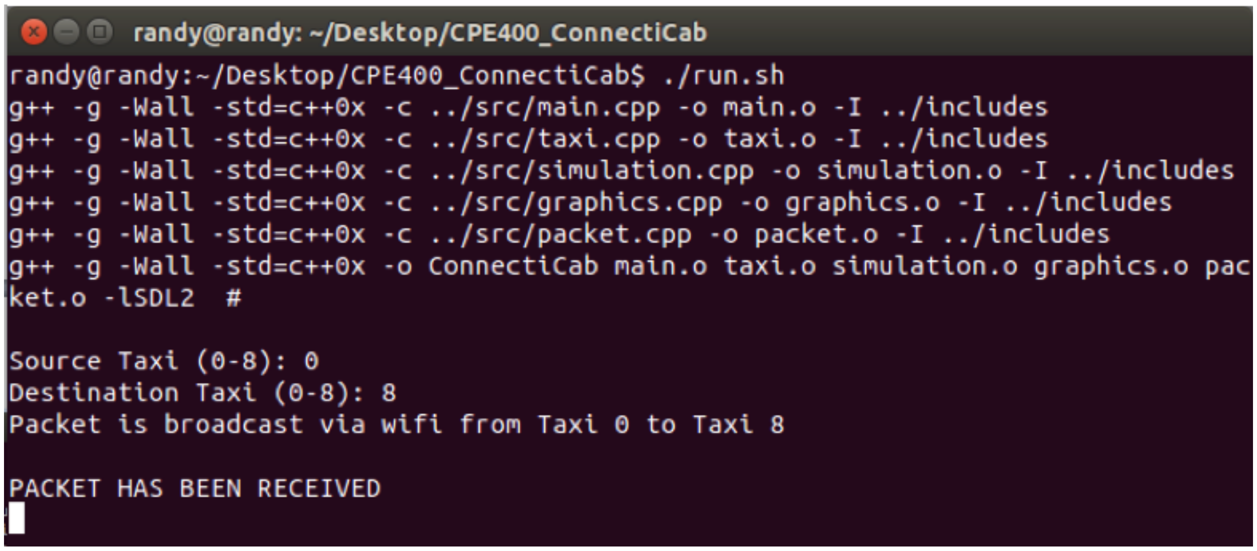


Fig 5: Packet transferred to another taxi cab and change color from yellow to red.

As shown in **figure 5**, a red taxi cab contains the packet that was successfully transferred via Wi-Fi and a change of color occurs specifying that the taxi cab had received a packet. Each taxi cab will keep updating its location the packet has been routed to its destination. In this case scenario, the destination taxi cab received the packet and the simulation came to an end. In the terminal, all log information is displayed which details all the different communication links established during that run. The terminal also specifies broadcast method, whether it was Wi-Fi or Bluetooth. As shown below in **figure 6** the program ends with the packet being received by the destination taxi.



```
randy@randy:~/Desktop/CPE400_ConnectiCab$ ./run.sh
g++ -g -Wall -std=c++0x -c ../src/main.cpp -o main.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/taxi.cpp -o taxi.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/simulation.cpp -o simulation.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/graphics.cpp -o graphics.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/packet.cpp -o packet.o -I ../includes
g++ -g -Wall -std=c++0x -o ConnectiCab main.o taxi.o simulation.o graphics.o pac
ket.o -lSDL2  #

Source Taxi (0-8): 0
Destination Taxi (0-8): 8
Packet is broadcast via wifi from Taxi 0 to Taxi 8

PACKET HAS BEEN RECEIVED
```

Fig 6: Display log of history of where the packet is being transferred and prompt user when packet had been delivered to destination taxi cab.

In another scenario, the packet is being broadcasted to another taxi cab nearby and happen to be in range as Wi-Fi. There is a line to indicate that a connection link was established and a change of color from yellow to red as shown below in **figure 7**.

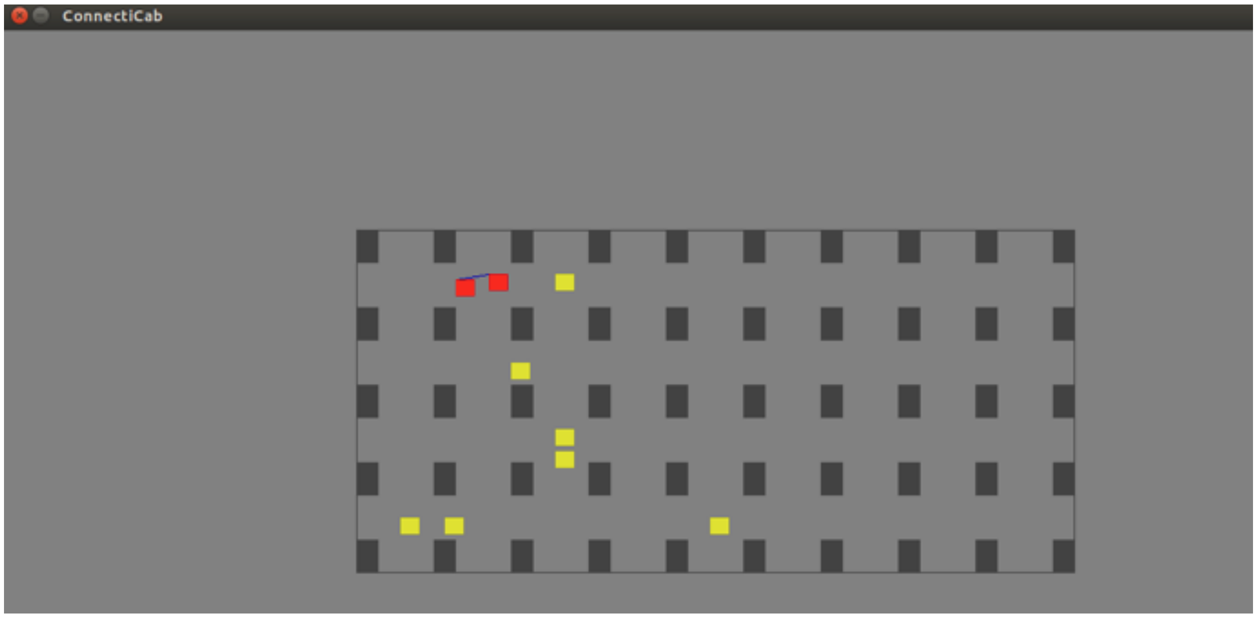


Fig 7: Packet is broadcasted from a taxi cab to another taxicab.

The routing algorithm continues to work until its end condition is satisfied. If the packet has not been received by the destination taxi specified then the packet will continue its route. In this case a communication and transfer was established but the packet had not reached its destination. The simulation continues as the red taxis will broadcast the packet even further. **Figures 8, 9,** and **10** demonstrate this process very well. From the images on may see that the routing algorithm is very effective. The terminal again, displays the log information of the communication links that were establish with either Wi-Fi or Bluetooth.

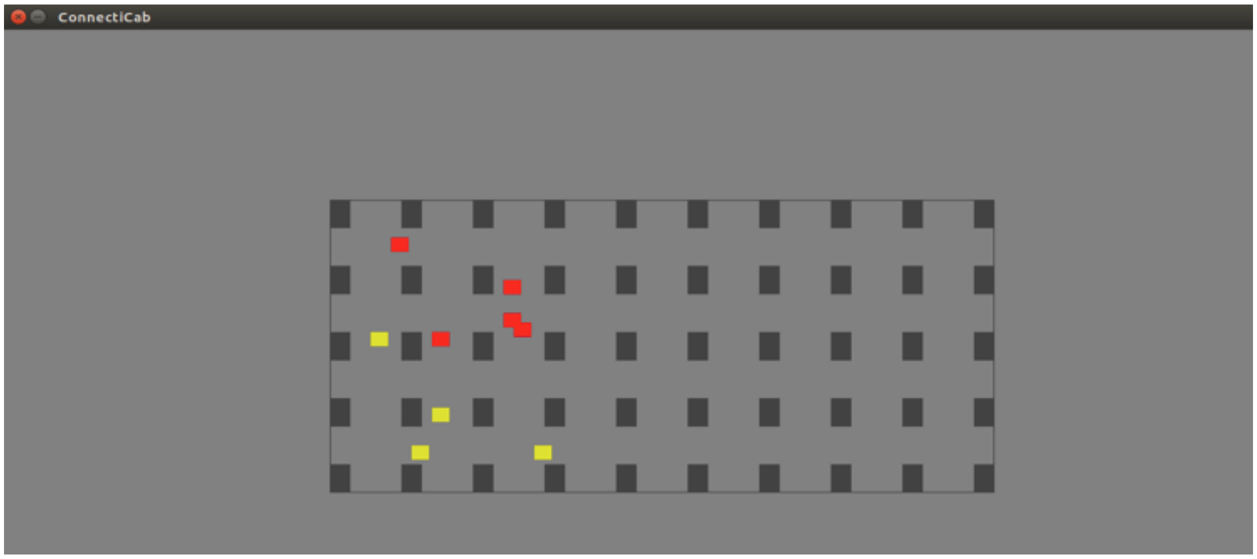


Fig 8: The packet being broadcast to other taxicabs in range of Wi-Fi and Bluetooth

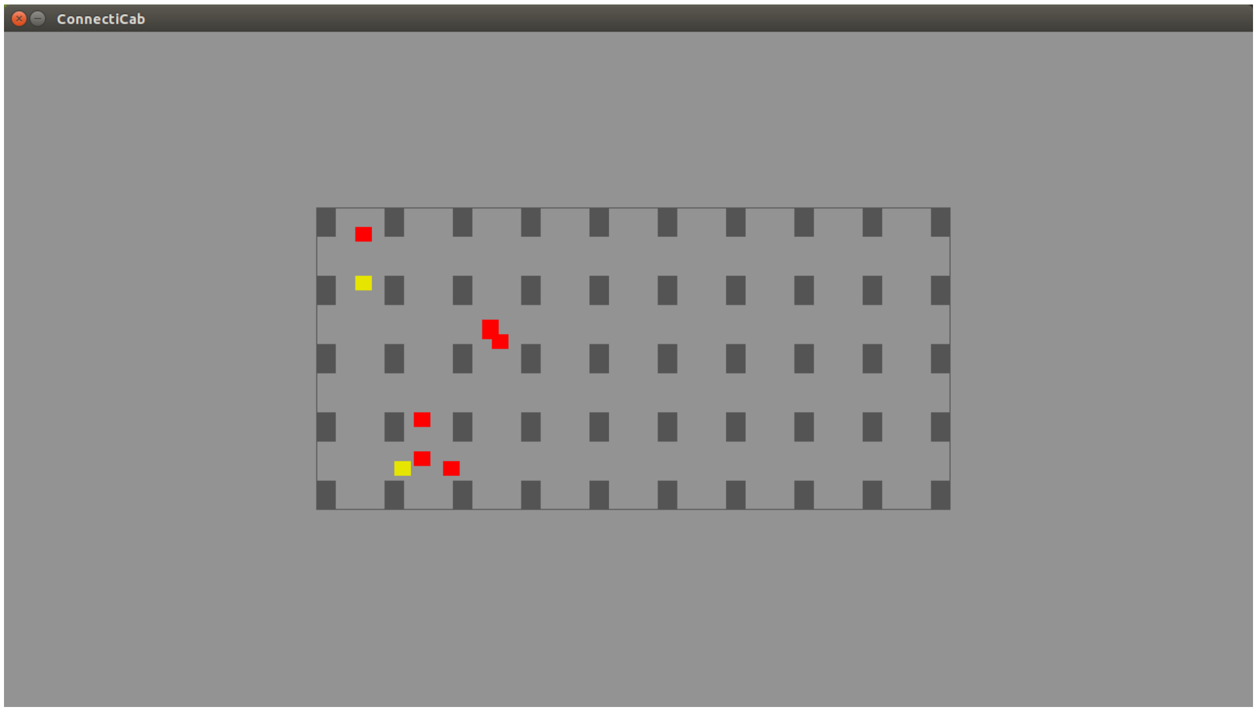
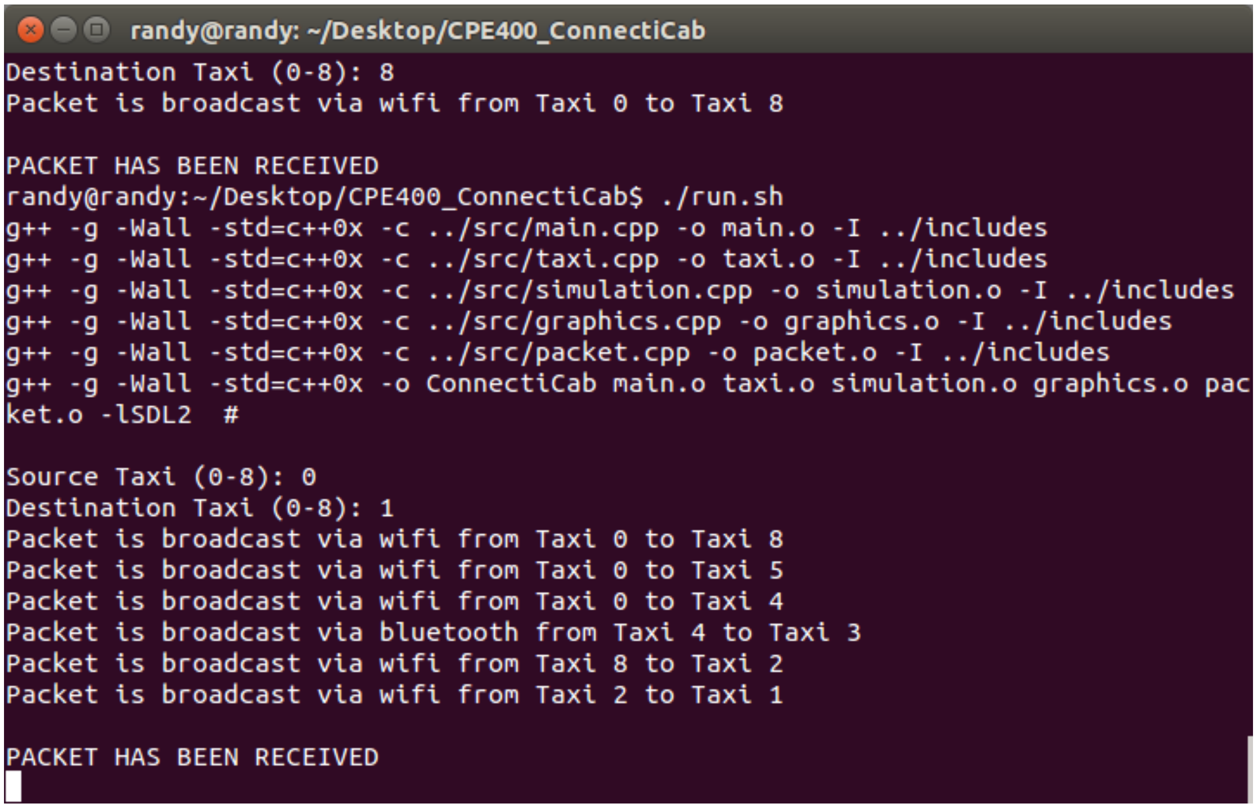


Fig 9: The packet has been received.



```
randy@randy: ~/Desktop/CPE400_ConnectiCab
Destination Taxi (0-8): 8
Packet is broadcast via wifi from Taxi 0 to Taxi 8

PACKET HAS BEEN RECEIVED
randy@randy:~/Desktop/CPE400_ConnectiCab$ ./run.sh
g++ -g -Wall -std=c++0x -c ../src/main.cpp -o main.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/taxi.cpp -o taxi.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/simulation.cpp -o simulation.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/graphics.cpp -o graphics.o -I ../includes
g++ -g -Wall -std=c++0x -c ../src/packet.cpp -o packet.o -I ../includes
g++ -g -Wall -std=c++0x -o ConnectiCab main.o taxi.o simulation.o graphics.o pac
ket.o -lSDL2  #

Source Taxi (0-8): 0
Destination Taxi (0-8): 1
Packet is broadcast via wifi from Taxi 0 to Taxi 8
Packet is broadcast via wifi from Taxi 0 to Taxi 5
Packet is broadcast via wifi from Taxi 0 to Taxi 4
Packet is broadcast via bluetooth from Taxi 4 to Taxi 3
Packet is broadcast via wifi from Taxi 8 to Taxi 2
Packet is broadcast via wifi from Taxi 2 to Taxi 1

PACKET HAS BEEN RECEIVED
```

Fig 10: Display of the log information specifying the method of broadcast between two taxis.

The results can be better and implement could be improved; however, the team did achieve the proposal goal of a peer-to-peer networking system. The protocol allows taxi cabs to broadcast a packet in the network between other neighboring nodes via Wi-Fi or Bluetooth. The routing algorithm allows packet to go from the source taxi cab to any tax cab until it reaches the destination taxi cab. The results are easy to see by including the screen and live simulation similar to the real world. Using different color indication helps user follow along of how the packet is being broadcasted. In conclusion, the routing algorithm is working as expected in this P2P representation. In the real world, more scenarios would have to be accounted for and further functionality must be implemented onto the algorithm. For the span of this project, the implementation achieved was appropriate for a simple simulation of a peer-to-peer taxi network.

References

A. M. E. Ejmaa, S. Subramaniam, Z. A. Zukarnain and Z. M. Hanapi, "Neighbor-Based Dynamic Connectivity Factor Routing Protocol for Mobile Ad Hoc Network," in IEEE Access, vol. 4, no. , pp. 8053-8064, 2016. doi: 10.1109/ACCESS.2016.2623238
keywords: {Ad hoc networks;Disasters;Emergency services;Energy consumption;Information systems;Mobile computing;Mobile handsets;Routing protocols;AODV;Mobile ad hoc networks;flooding;probabilistic rebroadcast;routing overhead},
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7736158&isnumber=7419931