# Simulation of Driverless Cars on a Software Defined Network using A* Search

AUTHOR: Andrew Frost, Richard Millar

Version : 1.0
12/04/2017

# Table of Contents

Table of contents

# CPE400

A repository for the CPE400 Networking Programming Project

### Building and Running

### Building and Running on Linux

Building:

```
1 make
```

Running:

```
1 ./SDN Input.txt
```

Cleaning:

```
1 make clean
```

### Vehicles

Each vehicle on the network is an abstraction of a network packet. This allows each one to hold and share only basic information, such as identification, source, and destination addresses. The majority of routing of these packets is therefore completed by the intersection routers. The amount of time spend between each node is recorded by the packet, which is then read by the router and passed to the central device. This information helps the Central Node determine wait times at each individual node; this in turn allows it to scale and reroute packets accordingly.

Class **Vehicle**:

```
* Methods:

    * Constructor
    * Parameterized Constructor
    * Destructor
    * Set Start Time
    * Set Depart Time
    * Get Travel Time
    * Get Total Time
    * Get Next Destination
    * Time Remaining To Next Destination
    * Clear Route
    * Has Route
    * Has Node
    * Get ID
    * Get Source
    * Get Dest
    * Request Route
    * Set Route
    * Try Road Change
    * Get Lock
    * Release Lock

* Properties:

    * Device id
    * Source address
```

```
    * Dest address
    * Travel Time
    * Total Time
    * Travel Time Left
    * Route (a list of subnets to traverse)
    * Route Requested
    * mutex
```

### Central Compute Node

The central compute node is responsible for routing all traffic.

Class **CentralComputeNode**:

```
* Methods:

    * Constructor
    * Destructor
    * Build Subnet To Index Table
    * Get Map Index
    * Set Map
    * Set Subnet Properties
    * Queue Job
    * Compute Route
    * Direct Traffic
    * Join Network
    * Leave Network
    * Change Road
    * Get Lock
    * Release Lock
    * AStar
    * Reconstruct Path
    * Expand Node

* Properties:

    * Vehicle ID to Vehicle Object (the abstracted "route" to that vehicle)
    * Subnet Capacity
    * Vehicles at each subnet (map)
    * City Map (adjacency matrix)
    * Subnet To Index Table
    * Jobs (a queue of routes to be computed)

    * mutex
```

### Input Structure

Please see the Input.txt for notes on the input structure.

# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# File Index

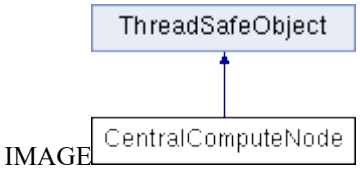
## File List

Here is a list of all documented files with brief descriptions:

# Class Documentation

## CentralComputeNode Class Reference

Inheritance diagram for CentralComputeNode:




IMAGE

### Public Member Functions

**CentralComputeNode** ()
*Default constructor.*

**~CentralComputeNode** ()
*Default destructor.*

void **buildSubnetToIndexTable** (std::vector< std::string > &subnets)
*Populates the subnetToIndexTable.*

int **getMapIndex** (const std::string &name)
*Returns index of ID.*

void **setMap** (std::vector< std::vector< double > > &map)
*Assign new map to object.*

void **setSubnetProperties** (std::string &name, int capacity)
*Assign properties to submet.*

void **queueJob** (**Job** &job)
*Adds a new job.*

bool **computeRoute** (**Route** &route)
*Computes route.*

void **directTraffic** (std::atomic_bool &running)
*Process waiting jobs for routes.*

void **joinNetwork** (**Vehicle** *vehicle)
*Add vehicle to network.*

void **leaveNetwork** (const std::string &id, const std::string &lastNode)
*Allow **Vehicle** to leave network.*

bool **changeRoad** (std::string &id, std::string &currentRoad, std::string &newRoad)
*Changes current road of vehicle.*

---

### Constructor & Destructor Documentation

#### CentralComputeNode::CentralComputeNode ()

Default constructor.

Constructs an empty **CentralComputeNode** object

**Note:**
None

### CentralComputeNode::~CentralComputeNode ()

Default destructor.

Destroys a **CentralComputeNode** object

**Note:**
None

---

## Member Function Documentation

### void CentralComputeNode::buildSubnetToIndexTable (std::vector< std::string > & *subnets*)

Populates the subnetToIndexTable.

Associates an index value to each subnet ID.

**Parameters:**

| | | |
|---|---|---|
| in | *subnets* | Vector of subnet IDS to assign |

**Note:**
None

### bool CentralComputeNode::changeRoad (std::string & *id*, std::string & *currentRoad*, std::string & *newRoad*)

Changes current road of vehicle.

Determines whether to allow **Vehicle** to change road, and if so, update vehicle location and count, else return false

**Parameters:**

| | | |
|---|---|---|
| in | *id* | vehicle ID |
| in | *currentRoad* | road vehicle is currently on |
| in | *newRoad* | road vehicle is requesting to switch to |

**Note:**
None

### bool CentralComputeNode::computeRoute (Route & *route*)

Computes route.

Computes the best route from start to end, and returns it

**Parameters:**

| | | |
|---|---|---|
| out | *route* | route to be computed and returned |

**Note:**
None

### void CentralComputeNode::directTraffic (std::atomic_bool & *running*)

Process waiting jobs for routes.

Processes all pending jobs in the queue, and if there are no vehicles present on the network, end the simulator.

**Parameters:**

| | | |
|---|---|---|
| out | *running* | boolean to determine whether the simulator is still running. |

**Note:**

None

### int CentralComputeNode::getMapIndex (const std::string & *name*)

Returns index of ID.

Gets and returns the associated index to the ID provided

**Parameters:**

| | | |
|---|---|---|
| in | *name* | ID to be searched for |

**Note:**

None

### void CentralComputeNode::joinNetwork (Vehicle * *vehicle*)

Add vehicle to network.

Appends vehicle to the vehicle map

**Parameters:**

| | | |
|---|---|---|
| in | *vehicle* | **Vehicle** to add to the network |

**Note:**

None

### void CentralComputeNode::leaveNetwork (const std::string & *id*, const std::string & *lastNode*)

Allow **Vehicle** to leave network.

Removes vehicle ID from network

**Parameters:**

| | | |
|---|---|---|
| in | *id* | ID of vehicle to remove |
| in | *lastNode* | last known node vehicle is at |

**Note:**

None

### void CentralComputeNode::queueJob (Job & *job*)

Adds a new job.

Appends a new job to the end of the queue

**Parameters:**

| | | |
|---|---|---|
| in | *job* | job to be appended |

**Note:**

None

### void CentralComputeNode::setMap (std::vector< std::vector< double > > & *map*)

Assign new map to object.

Set a new city map within the object

**Parameters:**

| | | |
|---|---|---|
| in | *map* | vector of vector of doubles that detail the distance from each node in the map |

**Note:**

None

### void CentralComputeNode::setSubnetProperties (std::string & *name*, int *capacity*)

Assign properties to submet.

Sets the subnet capacity specified by name

**Parameters:**

| | | |
|---|---|---|
| in | *name* | ID to be searched for |
| in | *capacity* | capacity of the subnet to assign |

**Note:**

None

---

**The documentation for this class was generated from the following files:**

0 **CentralComputeNode.h**
1 **CentralComputeNode.cpp**

# Job Struct Reference

### Public Member Functions

**Job** ()
*Default job constructor.*

**~Job** ()
*Default job desructor.*

### Public Attributes

std::string **start**
std::string **dest**
std::string **id**

---

### Constructor & Destructor Documentation

#### Job::Job ()

Default job constructor.

Constructs a job object

**Note:**
None

#### Job::~Job ()

Default job desructor.

Destroys job object

**Note:**
None

---

### The documentation for this struct was generated from the following files:

2 **CentralComputeNode.h**
3 **CentralComputeNode.cpp**

# Route Struct Reference

### Public Member Functions

**Route** ()
*__Route__ default constructor.*

**~Route** ()
*Default route destructor.*

### Public Attributes

std::string **start**
std::string **dest**
std::list< std::pair< std::string, double > > **route**

---

### Constructor & Destructor Documentation

#### Route::Route ()

**Route** default constructor.

Initializes route object

**Note:**
None

#### Route::~Route ()

Default route destructor.

Destroys route object

**Note:**
None

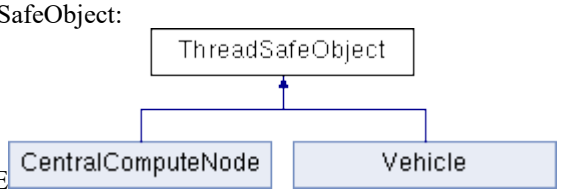
---

### The documentation for this struct was generated from the following files:

4 **CentralComputeNode.h**
5 **CentralComputeNode.cpp**

# ThreadSafeObject Class Reference

Inheritance diagram for ThreadSafeObject:

IMAGE 


### Public Member Functions

void **getLock** ()
void **releaseLock** ()

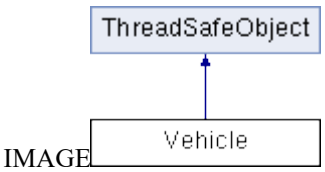
---

The documentation for this class was generated from the following files:

6 **ThreadSafeObject.h**
7 **ThreadSafeObject.cpp**

# Vehicle Class Reference

Inheritance diagram for Vehicle:




IMAGE

### Public Member Functions

**Vehicle** ()
*Default constructor.*

**Vehicle** (std::string newID, std::string newSource, std::string newDest)
***Vehicle** Constructor.*

**Vehicle** (const **Vehicle** &other)
***Vehicle** Copy Constructor.*

**~Vehicle** ()
***Vehicle** destructor.*

void **setStartTime** ()
*Sets the time the vehicle begins its journey.*

void **setDepartTime** ()
*Set the depart time of the vehicle.*

std::chrono::duration< double > **getTravelTime** () const
*Get the current travel time between nodes.*

std::chrono::duration< double > **getTotalTime** () const
*Get the total travel time.*

std::string **getNextDestination** () const
*Get the next vehicle destination.*

bool **timeRemainingToNextDestination** () const
*Shows whether the vehicle has arrived at a node.*

void **clearRoute** ()
*Clears the vehicle route.*

bool **hasRoute** () const
*Show whether the vehicle has a route.*

bool **hasNode** (const std::string &node) const
*Determine whether vehicle is traveling to node.*

std::string **getID** ()
*Get **Vehicle** ID.*

std::string **getSource** ()
*Get the source.*

std::string **getDest** ()
*Get the destination.*

void **requestRoute** (**CentralComputeNode** &ccn)
*Request a new route from ccn.*

void **setRoute** (std::list< std::pair< std::string, double >> newRoute)
*Sets the vehicle route.*

bool **tryRoadChange** (**CentralComputeNode** &ccn)

*Try to do a road change.*

---

### Constructor & Destructor Documentation

#### Vehicle::Vehicle ()

Default constructor.

Constructs **Vehicle** object

**Note:**

None

#### Vehicle::Vehicle (std::string *newID*, std::string *newSource*, std::string *newDest*)

**Vehicle** Constructor.

Constructs vehicle object with the specified values

**Parameters:**

| | | |
|---|---|---|
| in | *newID* | id to assign to object |
| in | *newSource* | source of the new object |
| in | *newDest* | destination of the new object |

**Note:**

None

#### Vehicle::Vehicle (const Vehicle & *other*)

**Vehicle** Copy Constructor.

Create a copy of the passed vehicle object

**Parameters:**

| | | |
|---|---|---|
| in | *other* | **Vehicle** to make a copy of |

**Note:**

None

#### Vehicle::~Vehicle ()

**Vehicle** destructor.

Destroys vehicle object and data within

**Note:**

None

---

### Member Function Documentation

#### void Vehicle::clearRoute ()

Clears the vehicle route.

Deletes the current vehicle route

**Note:**
None

### std::string Vehicle::getDest ()

Get the destination.

Returns where the vehicle is going

**Note:**
None

### std::string Vehicle::getID ()

Get **Vehicle** ID.

Returns the vehicle ID

**Note:**
None

### std::string Vehicle::getNextDestination () const

Get the next vehicle destination.

Returns the next route node if not null

**Note:**
None

### std::string Vehicle::getSource ()

Get the source.

Returns where the vehicle began from

**Note:**
None

### std::chrono::duration< double > Vehicle::getTotalTime () const

Get the total travel time.

Returns the totalTime

**Note:**
None

### std::chrono::duration< double > Vehicle::getTravelTime () const

Get the current travel time between nodes.

Returns the difference between now and the travelTime

**Note:**
None

### bool Vehicle::hasNode (const std::string & *node*) const

Determine whether vehicle is traveling to node.

Returns whether the node is within the vehicle route

**Parameters:**

| | | |
|---|---|---|
| in | *node* | node to search for in the route |

**Note:**
None

### bool Vehicle::hasRoute () const

Show whether the vehicle has a route.

Returns if route is null

**Note:**
None

### void Vehicle::requestRoute (CentralComputeNode & *ccn*)

Request a new route from ccn.

Send a job request to ccn to set a new route

**Parameters:**

| | | |
|---|---|---|
| in | *ccn* | Central compute node |

**Note:**
None

### void Vehicle::setDepartTime ()

Set the depart time of the vehicle.

Sets the travel time to the current time

**Note:**
None

### void Vehicle::setRoute (std::list< std::pair< std::string, double >> *newRoute*)

Sets the vehicle route.

Sets the route object to the new route

**Parameters:**

| | | |
|---|---|---|
| in | *newRoute* | new route to set the data to |

**Note:**
None

### void Vehicle::setStartTime ()

Sets the time the vehicle begins its journey.

Sets the total time to the current time

**Note:**

None

### bool Vehicle::timeRemainingToNextDestination () const

Shows whether the vehicle has arrived at a node.

Returns whether the travel time is less than the time left

**Note:**

None

### bool Vehicle::tryRoadChange (CentralComputeNode & *ccn*)

Try to do a road change.

If the object can change its current road do so, else wait

**Parameters:**

| | | |
|---|---|---|
| in | *ccn* | Central compute node to send requests to |

**Note:**

None

---

**The documentation for this class was generated from the following files:**

8 **Vehicle.h**
9 **Vehicle.cpp**

# File Documentation

## CentralComputeNode.cpp File Reference

Implementation file for the **CentralComputeNode** class.

```
#include "CentralComputeNode.h"
#include <atomic>
```

### Macros

#define **_INFINITY** 9999999

### Functions

template<typename Type > bool **GetCheapestNode** (std::unordered_set< Type > &set, std::map< Type, double > &fScore, Type &lowest)

*Finds the cheapest node.*

---

### Detailed Description

Implementation file for the **CentralComputeNode** class.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

---

### Function Documentation

#### template<typename Type > bool GetCheapestNode (std::unordered_set< Type > & *set*, std::map< Type, double > & *fScore*, Type & *lowest*)

Finds the cheapest node.

Scans the opened set to find the least f-score node.

**Parameters:**

| | | |
|---|---|---|
| in | *set* | set to be scanned for node |
| in | *fScore* | map of node IDs and associated f-scores |
| out | *lowest* | Cheapest node found |

**Note:**

None

# CentralComputeNode.h File Reference

Definition file for the **CentralComputeNode** class.

```
#include <unordered_set>
#include <vector>
#include <list>
#include <map>
#include <string>
#include <atomic>
#include "Vehicle.h"
#include "ThreadSafeObject.h"
```

### Classes

class **CentralComputeNode**
struct **Job**
struct **Route**

---

### Detailed Description

Definition file for the **CentralComputeNode** class.

The **CentralComputeNode** class is the central computer of the network. It manages all incoming requests to it, and updates routes according to network conditions.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

# main.cpp File Reference

Main processing file for the Software Defined Network simulator.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstdlib>
#include <chrono>
#include <thread>
#include <atomic>
#include <functional>
#include <vector>
#include <string>
#include "ThreadSafeObject.h"
#include "Vehicle.h"
#include "CentralComputeNode.h"
```

### Functions

bool **FetchInput** (std::string &fileName, **CentralComputeNode** &ccn, std::vector< **Vehicle** > &cars)

*Process input file.*

void **EndSimulator** (std::vector< std::thread > &simulatorThreads)

*End the simulator.*

void **WaitFor** (long long timeMS)

*Wait for a specified time.*

void **ComputeNode** (**CentralComputeNode** &ccn, std::atomic_bool &running, **ThreadSafeObject** &consoleLock)

*Begins the compute node processing.*

void **Car** (**CentralComputeNode** &ccn, std::atomic_bool &running, **ThreadSafeObject** &consoleLock, **Vehicle** car, long long timeStep)

*Operations done by each **Vehicle** object.*

int **main** (int argc, char *argv[])

---

### Detailed Description

Main processing file for the Software Defined Network simulator.

Takes in user input file and starts the simulator, processing each vehicle route until all are finished

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

---

### Function Documentation

#### void Car (CentralComputeNode & *ccn*, std::atomic_bool & *running*, ThreadSafeObject & *consoleLock*, Vehicle *car*, long long *timeStep*)

Operations done by each **Vehicle** object.

This function runs the car and all its operations.

**Parameters:**

| | | |
|---|---|---|
| in | *ccn* | central compute node |
| in | *running* | flag to show simulator is running |
| in | *consoleLock* | lock for the console output |
| in | *car* | main thread object |
| in | *timeStep* | |

### void ComputeNode (CentralComputeNode & *ccn*, std::atomic_bool & *running*, ThreadSafeObject & *consoleLock*)

Begins the compute node processing.

Runs the compute node and checks for open jobs periodically

**Parameters:**

| | | |
|---|---|---|
| in | *ccn* | Main compute node of simulator |
| in | *running* | flag to show that the simulator is running |
| in | *consoleLock* | Lock assigned to the console for output |

### void EndSimulator (std::vector< std::thread > & *simulatorThreads*)

End the simulator.

Wait for each thread to join

**Parameters:**

| | | |
|---|---|---|
| in | *simulatorThreads* | list of threads |

### bool FetchInput (std::string & *fileName*, CentralComputeNode & *ccn*, std::vector< Vehicle > & *cars*)

Process input file.

Parses out input file and places the data into the compute node

**Parameters:**

| | | |
|---|---|---|
| in | *fileName* | file to parse |
| in | *ccn* | Central node |
| in | *cars* | List of vehicles |

### void WaitFor (long long *timeMS*)

Wait for a specified time.

puts the current thread to sleep

**Parameters:**

| | | |
|---|---|---|
| in | *timeMS* | time period to wait in milliseconds |

# ThreadSafeObject.cpp File Reference

Implementation file for the **ThreadSafeObject** class.

```
#include "ThreadSafeObject.h"
```

---

### Detailed Description

Implementation file for the **ThreadSafeObject** class.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

# ThreadSafeObject.h File Reference

Definition file for the **ThreadSafeObject** class.

```
#include <mutex>
```

### Classes

class **ThreadSafeObject**

### Typedefs

using **Lock** = std::unique_lock< std::mutex >

---

### Detailed Description

Definition file for the **ThreadSafeObject** class.

This class is extended by the **CentralComputeNode** and **Vehicle** classes to provide a unique lock and functionality per object.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

# Vehicle.cpp File Reference

Implementation file for the **Vehicle** class.

```
#include "Vehicle.h"
#include "CentralComputeNode.h"
```

---

### Detailed Description

Implementation file for the **Vehicle** class.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

# Vehicle.h File Reference

Definition file for the **Vehicle** class.

```
#include <list>
#include <string>
#include <chrono>
#include "ThreadSafeObject.h"
#include "CentralComputeNode.h"
```

### Classes

class **Vehicle**

---

### Detailed Description

Definition file for the **Vehicle** class.

This class is the object that represents all vehicles on the network.

**Author:**

Andrew Frost, Richard Millar

**Version:**

1.00

# Index

INDE