# CPE 400: Software Defined Network

1.00

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# CPE400

A repository for the CPE400 Networking Programming Project

### Building and Running on Linux

Building:

```
make
```

Running:

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

Cleaning:

```
make clean
```

#### Network Structure

#### 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

The structure of the input file is fairly straight forward. Each object on the network has its own keyword that is recognized by the program:

- Car:
  - This keyword is used to specify a vehicle on the network.
  - car car-name source destination
- Intersect:
  - The intersection command specified a node within the network.
  - The command must contain an intersect name, as well as the maximum capacity of the node.
  - intersect intersect-name capacity
- Neighbor:
  - The neighbor keyword details a neighbor node to the preceding intersect.
  - The referenced neighbor node can be detailed either through a direct reference to the object, or by its index.
  - Along with a reference to the neighbor node, the length of time between the nodes must also be specified.
  - neighbor [node-index]|node-title timespan

The input file also allows for the use of comments, which begin with '#' at the beginning of the comment.

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

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

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

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

# Chapter 5

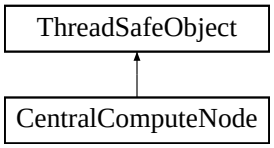
# Class Documentation

## 5.1 CentralComputeNode Class Reference

The centralized compute node for the entire network.

`#include "CentralComputeNode.h"`

Inheritance diagram for CentralComputeNode:




### 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.*

### 5.1.1 Detailed Description

The centralized compute node for the entire network.

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

### 5.1.2 Constructor & Destructor Documentation

#### 5.1.2.1 CentralComputeNode()

```
CentralComputeNode::CentralComputeNode ( )
```

Default constructor.

Constructs an empty CentralComputeNode object

**Note**

None

#### 5.1.2.2 ~CentralComputeNode()

```
CentralComputeNode::~CentralComputeNode ( )
```

Default destructor.

Destroys a CentralComputeNode object

**Note**

None

### 5.1.3 Member Function Documentation

#### 5.1.3.1 buildSubnetToIndexTable()

```
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

#### 5.1.3.2 changeRoad()

```
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

#### 5.1.3.3 computeRoute()

```
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

#### 5.1.3.4 directTraffic()

```
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

#### 5.1.3.5 getMapIndex()

```
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

#### 5.1.3.6 joinNetwork()

```
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

#### 5.1.3.7 leaveNetwork()

```
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

#### 5.1.3.8 queueJob()

```
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

#### 5.1.3.9 setMap()

```
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

#### 5.1.3.10 setSubnetProperties()

```
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:

- CentralComputeNode.h
- CentralComputeNode.cpp

## 5.2 Job Struct Reference

Request structure passed to the Compute Node.

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

### Public Member Functions

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

### Public Attributes

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

## 5.2.1 Detailed Description

Request structure passed to the Compute Node.

Object that is passed by Vehicles to the Compute Node to be processed.

## 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 Job()

```
Job::Job ( )
```

Default job constructor.

Constructs a job object

**Note**

None

#### 5.2.2.2 ~Job()

```
Job::~Job ( )
```

Default job desructor.

Destroys job object

**Note**

None

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

- CentralComputeNode.h
- CentralComputeNode.cpp

## 5.3 Route Struct Reference

Route object that contains a clear path between nodes.

`#include <CentralComputeNode.h>`

### 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**

### 5.3.1 Detailed Description

Route object that contains a clear path between nodes.

This object holds both a starting and end node, with a route that is computed by the Compute Node.

### 5.3.2 Constructor & Destructor Documentation

#### 5.3.2.1 Route()

`Route::Route ( )`

Route default constructor.

Initializes route object

**Note**

None

#### 5.3.2.2 ~Route()

```
Route::~Route ( )
```

Default route destructor.

Destroys route object

**Note**

None

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

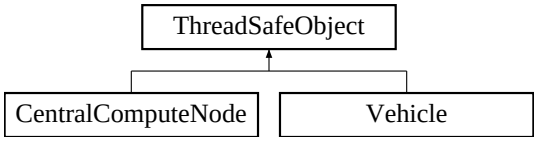
- CentralComputeNode.h
- CentralComputeNode.cpp

## 5.4 ThreadSafeObject Class Reference

The ThreadSafeObject class is inherited by others to allow for objects to lock their resources between threads.

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

Inheritance diagram for ThreadSafeObject:




### Public Member Functions

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

### 5.4.1 Detailed Description

The ThreadSafeObject class is inherited by others to allow for objects to lock their resources between threads.

This class provides two methods, one to lock the resource, and one to release it.

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

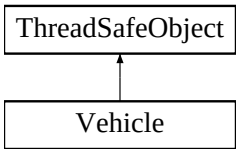
- ThreadSafeObject.h
- ThreadSafeObject.cpp

## 5.5 Vehicle Class Reference

This class represents the vehicles that make up the network of the SDN.

`#include "Vehicle.h"`

Inheritance diagram for Vehicle:




### 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.*

### 5.5.1 Detailed Description

This class represents the vehicles that make up the network of the SDN.

Each Vehicle is able to navigate between nodes from start to finish along a route precomputed for it by the Compute Node.

### 5.5.2 Constructor & Destructor Documentation

#### 5.5.2.1 Vehicle() [1/3]

```
Vehicle::Vehicle ( )
```

Default constructor.

Constructs Vehicle object

**Note**

None

#### 5.5.2.2 Vehicle() [2/3]

```
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

#### 5.5.2.3 Vehicle() [3/3]

```
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

#### 5.5.2.4 ~Vehicle()

```
Vehicle::~Vehicle ( )
```

Vehicle destructor.

Destroys vehicle object and data within

**Note**

None

### 5.5.3 Member Function Documentation

#### 5.5.3.1 clearRoute()

```
void Vehicle::clearRoute ( )
```

Clears the vehicle route.

Deletes the current vehicle route

**Note**

None

#### 5.5.3.2 getDest()

```
std::string Vehicle::getDest ( )
```

Get the destination.

Returns where the vehicle is going

**Note**

None

#### 5.5.3.3 getID()

```
std::string Vehicle::getID ( )
```

Get Vehicle ID.

Returns the vehicle ID

**Note**

None

#### 5.5.3.4 getNextDestination()

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

Get the next vehicle destination.

Returns the next route node if not null

**Note**

None

#### 5.5.3.5 getSource()

```
std::string Vehicle::getSource ( )
```

Get the source.

Returns where the vehicle began from

**Note**

None

#### 5.5.3.6 getTotalTime()

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

Get the total travel time.

Returns the totalTime

**Note**

None

#### 5.5.3.7 getTravelTime()

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

Get the current travel time between nodes.

Returns the difference between now and the travelTime

**Note**

None

#### 5.5.3.8 hasNode()

```
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

#### 5.5.3.9 hasRoute()

```
bool Vehicle::hasRoute ( ) const
```

Show whether the vehicle has a route.

Returns if route is null

**Note**

None

#### 5.5.3.10 requestRoute()

```
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

#### 5.5.3.11 setDepartTime()

```
void Vehicle::setDepartTime ( )
```

Set the depart time of the vehicle.

Sets the travel time to the current time

**Note**

None

#### 5.5.3.12 setRoute()

```
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

#### 5.5.3.13 setStartTime()

```
void Vehicle::setStartTime ( )
```

Sets the time the vehicle begins its journey.

Sets the total time to the current time

**Note**

None

#### 5.5.3.14 timeRemainingToNextDestination()

```
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

#### 5.5.3.15 tryRoadChange()

```
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:

- Vehicle.h
- Vehicle.cpp

# Chapter 6

# File Documentation

## 6.1 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.*

### 6.1.1 Detailed Description

Implementation file for the CentralComputeNode class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

### 6.1.2 Function Documentation

#### 6.1.2.1 GetCheapestNode()

```
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

## 6.2 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

  *The centralized compute node for the entire network.*
- struct Job

  *Request structure passed to the Compute Node.*
- struct Route

  *Route object that contains a clear path between nodes.*

### 6.2.1 Detailed Description

Definition file for the CentralComputeNode class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

## 6.3 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 (const char *fileName, CentralComputeNode &ccn, std::vector< Vehicle > &cars)

  *Process input file.*
- void RunSimulator (CentralComputeNode &ccn, std::vector< Vehicle > &vehicles)

  *Run the simulator until end.*
- 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[])

### 6.3.1 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

### 6.3.2 Function Documentation

#### 6.3.2.1 Car()

```
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* | time from beginning of sim to start of car |

#### 6.3.2.2 ComputeNode()

```
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 |

#### 6.3.2.3 EndSimulator()

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

End the simulator.

Wait for each thread to join

**Parameters**

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

#### 6.3.2.4 FetchInput()

```
bool FetchInput (
            const char * 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 |

#### 6.3.2.5 RunSimulator()

```
void RunSimulator (
            CentralComputeNode & ccn,
            std::vector< Vehicle > & vehicles )
```

Run the simulator until end.

Initializes the simulator by launching the vehicle threads and starting the compute node.

**Parameters**

| | | |
|---|---|---|
| in | *ccn* | Compute Node of the simulator |
| in | *vehicles* | List of vehicles in the simulator |

#### 6.3.2.6 WaitFor()

```
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 |

## 6.4 ThreadSafeObject.cpp File Reference

Implementation file for the ThreadSafeObject class.

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

### 6.4.1 Detailed Description

Implementation file for the ThreadSafeObject class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

## 6.5 ThreadSafeObject.h File Reference

Definition file for the ThreadSafeObject class.

```
#include <mutex>
```

### Classes

- class ThreadSafeObject

  *The ThreadSafeObject class is inherited by others to allow for objects to lock their resources between threads.*

### Typedefs

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

#### 6.5.1 Detailed Description

Definition file for the ThreadSafeObject class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

## 6.6 Vehicle.cpp File Reference

Implementation file for the Vehicle class.

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

#### 6.6.1 Detailed Description

Implementation file for the Vehicle class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

## 6.7 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

  *This class represents the vehicles that make up the network of the SDN.*

### 6.7.1 Detailed Description

Definition file for the Vehicle class.

**Author**

Andrew Frost, Richard Millar

**Version**

1.00

# Index