國 立 清 華 大 學

碩 士 論 文

在 ns-3 上實作都市道路環境的車載網路移動行為模擬模組

VanetURSim: An ns-3 VANET Mobility Module for Urban Road Networks

系所別：資訊工程研究所碩士班

學號姓名：102062645 陳俊州 (Chun-Chou Chen)

指導教授：楊舜仁 博士 (Prof. Shun-Ren Yang)

中華民國 104 年 8 月

VanetURSim: An ns-3 VANET Mobility Module for Urban Road Networks

Chun Chou Chen

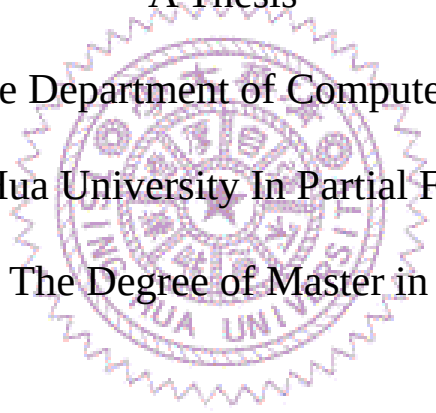
Advisor: Dr. Shun-Ren Yang

A Thesis

Submitted to The Department of Computer Science of The

National Tsing-Hua University In Partial Fulfillment of The

Requirements For The Degree of Master in Computer Science



Hsinchu, Taiwan, Republic of China

August, 2015

摘要

車輛在行進時，不會漫無目的的在道路上移動，透過這些有限行為能力的車輛和道路旁的路側單元所組成的網路，被稱之為車載網路 (Vehicular ad-hoc network)，車載網路被提出的目的，最主要是用於提升用車安全，避免意外事故發生，在研究車載網路時，一套新的傳輸機制、或是新的行車導航設計，在未經過模擬測試之前，是無法被大眾所使用的，所以當我們研究車載網路時，一個可以產生貼近於現實車載網路環境的模擬器是十分重要的。車載網路的研究者，最常設計的傳輸方式大多應用在都市或郊區，然而，在知名的網路模擬器 ns-3 上，並沒有都市或郊區的車載網路模擬環境，所以我們希望能夠在 *ns-3* 建立車載網路的模擬環境，並且能夠真實的反應出車輛在現實環境中的行為模式。本論文所提出之 VanetURSim 主要能讓使用者模擬車輛的現實移動行為模式，像是遵守基本的交通規、紅燈停、綠燈行，並且車輛會依照規畫好的軌跡移動，VanetURSim 提供了車子的移動模組與含有紅綠燈的都市道路環境，使用者可以透過 VanetURSim 簡易的在 ns-3 上創建車載網路環境。另外，我們提出了在路側單元與車輛傳輸時流量排程的方法，利用機率的方式計算，挑選最適合協助傳輸的車輛，以此來減少能量的消耗。

致謝

這篇論文能夠順利完成，是因為有許多人在背後的支持與幫助，在這些人當中，我的指導老師 楊舜仁教授，對我的幫助是無可取代的。

剛考上清華資訊工程研究所時，大約在四、五月時，老師就開始與我商討未來的研究方向，並且安排昱傑學長整理相關的文獻，由淺入深的交給我自行安排時間，然而，雖說是自行安排，卻也不是放牛吃草，老師會要求我每週回報進度，並且如果有遇到困難，可以與老師商討。在一定的程度上給予彈性，並且能依照個人的能力進行排程，我覺得能夠減少緊張的情緒與壓力。在找尋研究題目的過程中，老師投入了大量的時間與心血，一對一的帶著我搜尋論文和相關的研究，減少自行摸索上的茫然與困苦，並且每當我找到有興趣研究的主題時，老師也會幫助我釐清問題的本質，以及與我商量論文方向的可行性，透過這個方式，可以發現自己沒有考慮到的部分。最後在論文的修改與口試的預演，老師也給予我們很大的幫助，謝謝老師的指導與教誨。

在 WMNET 的時間裡，同一屆的小夥伴陪伴我一起克服了許多事情；感謝沈柏君的嚴厲督導與指責，還有幫助解答課程的問題，從他的身上，我看到了許多閃亮點，並且在我猶豫不決的時候，強硬的給予我理性的分析；感謝沈宜璇當小秘書，提醒我重大的事件與相關的學校公布資訊，並且還會陪聊、陪玩、陪發瘋，從她的身上，我看到了韓國文化對台灣人的影響，並且使我的阿宅之路不再孤單；感謝吳承儒帶給我八卦、音樂和花邊知識，以及文件的書寫方式，從他的身上，我找到了溫暖與陪伴，在我面臨緊張和徬徨時，能夠有人一起崩潰。

另外，我要感謝博班學長胖虎、米奇，碩班學長阿 Jay、勇昌、鈞閔、昱達、仲為，碩班學弟葉子、小賓、蘇蘇、家甄，研究助理大黃、勇安、章魚哥，沒有這些人的幫助，我們的碩士生活絕對不會這麼精彩。

目錄

圖目錄

1. 概論

車輛在行進時，不會漫無目的的在道路上移動，透過這些有限行為能力的車輛和道路旁的路側單元所組成的網路，被稱之為車載網路 (Vehicular ad-hoc network)，這些車輛和路側單元會組成節點，並利用無線通訊技術，形成行動網路，並且由於車輛的高機動性，行動網路的節點會快速變化，如何在這種情況下完成封包的傳輸是車載網路十分重要的議題，車載網路中，封包的傳輸方向大致上可以分成四類，分別是路側單元傳給路側單元 (*I2I*)、車輛傳給路側單元 (*V2I*)、路側單元傳給車輛 (*I2V*)、車輛傳給車輛 (*V2V*)，研究者通常會依據這四種傳輸的方向改良封包的傳送機制，並且在車載網路當中，封包也會被分成許多種類，大致上有廣告訊息、緊急訊息、商店資訊等，不同的封包所對應的需求也不盡相同，針對不同類型、甚至是同種類的封包，都存在許多種不同的傳輸機制，研究者針對不同的面向去改善、增進，[1-2]著重於在最短的時間內將封包送達、[3]著重於傳輸時有最高的封包送達率、[4]著重於傳輸時的耗能損耗，這些充滿多樣性的傳輸機制，在正是被採納、應用錢，研究者必須驗證他們設計的傳輸機制，確保他們的可行性，然而，實體的道路量測需要的時間以及花費太高，所以在初步的量測時，研究者必須透過虛擬的網路模擬，發現研究的價值與成果。

模擬器，或者特別說是網路模擬器，是透過軟體撰寫而成的，主要的目的是希望能夠模擬現實的網路環境，並且模擬出來的網路環境能夠越貼近於現實會越好。在研究車載網路時，一套新的傳輸機制、或是新的行車導航設計，在未經過模擬測試之前，是無法被大眾所使用，更甚者，不論是在商業或是學術界，模擬都是必不可少的一項環節，所以當我們研究車載網路時，一個可以產生貼近於現實車載網路環境的模擬器是十分重要的；然而，由於現實的環境過於複雜與瑣碎，過於龐大的資料與考慮條件，會使得程式的撰寫複雜度直線上升，於是，不同類型的網路模擬器被人所創建並且用於適應不同的情況。

目前存有很多的開放源代碼的網路模擬器[5-8]，其中 *ns-2*，*OMNeT++*都有相關的車載網路模擬環境。*ns-2* 由於開發時間較早的緣故，編譯的語言是透過 *C++*和 *OTcl* 結合而成，並且 *ns-2* 是物件的導向的網路模擬器，使用者可以自己編寫物件或是透過現存的物件文庫，創造自己新的網路物件，然而，儘管 ns-2 有非常多可以使用的車載網路模擬資源，卻因為其編譯的不方便性，以及對於傳輸模擬上的簡化，目前越來越少人在 ns-2 上測試。*OMNeT++*具有圖形化的介面，使用者可以簡易的使用當中定義的模組元件，然而必須要說的是，*OMNeT++*其實本身並不是網路模擬器，更像是網路模擬平台，使用者可以在上面建造建立自己的模擬，然而考量到現實的封包傳輸環境，*OMNeT++*的封包傳輸模擬就不太適合，使用者通常只會利用 *OMNeT++*進行簡易的封包傳輸模擬。

為了希望能夠更貼近於現實的封包傳輸環境，*ns-3* 在近幾年被設計出來，*ns-3* 在設計的概念上參考了其他的模擬器，並且 *ns-3* 與 *ns-2* 並沒有版本上的關係，*ns-3* 是一個獨立且公開的模組化的網路模擬器，其中在編譯語言上，可以全部使用 *C++*或是可以選擇結合 *python* 腳本，並且在封包傳輸方面的設計，*ns-3* 因為封包的規格與傳輸層級的模擬十分貼近於現實的情況，以及程式模組的層級架構，使得 *ns-3* 被廣泛的應用在學術以及教育上，封包的規格像是封包標頭的定義，封包的編碼與解碼過程，而傳輸層級則是仿照現實的 7 層式架構傳輸，*ns-3* 現存有許多的網路及相關的模組，使用者可以透過這些模組建立想要模擬的網路模擬的環境，[9-12]都是利用 *ns-3* 建立自己的網路模擬環境來驗證實驗成果，然而，由於 *ns-3* 是近期才被建立的網路模擬器，在官方認證且公開的程式碼當中並沒有足夠的模擬環境模組，車載網路的模擬環境目前就尚未被添加到 *ns-3* 當中，也正因為如此，儘管 *ns-3* 具備有十分完善的封包模擬與傳輸機制，車載網路的研究者並不常在 *ns-3* 上面實踐自己的模擬。

由於越來越多大眾使用 *ns-3* 作為模擬環境，研究者也注意到了 *ns-3* 上面缺乏車載網路環境的模組，開始有研究者著手設計模組，並且公開在實驗室或是機

構的網站上，[13]、[14]基於 *ns-3* 的架構，開發車載網路的模擬環境，[13]利用 *ns-3* 已有的模組，建立出車輛會隨機移動的模擬環境，[14]則是建立了高速公路的模擬環境，然而[13]並未考慮在現實的情況，車輛的行為模式並不是隨機移動，而是會有特定的行為模式，而[14]所產生的車載網路環境則過於單純，因為在高速公路上，車輛的方向不會產生變化，並且也缺少紅綠燈的存在，在現實的車載環境中，車輛的行為模式，除了會遵守紅燈停、綠燈行及一般的交通規則外，車輛在移動時，通常會有起始點與目的地，並且車輛會按照預先規劃的軌跡移動，不論[13]、[14]，均未考慮到都市的道路情況，並且缺少了紅綠燈的存在，然而車載網路的研究者，最常設計的傳輸方式大多應用在都市或郊區。

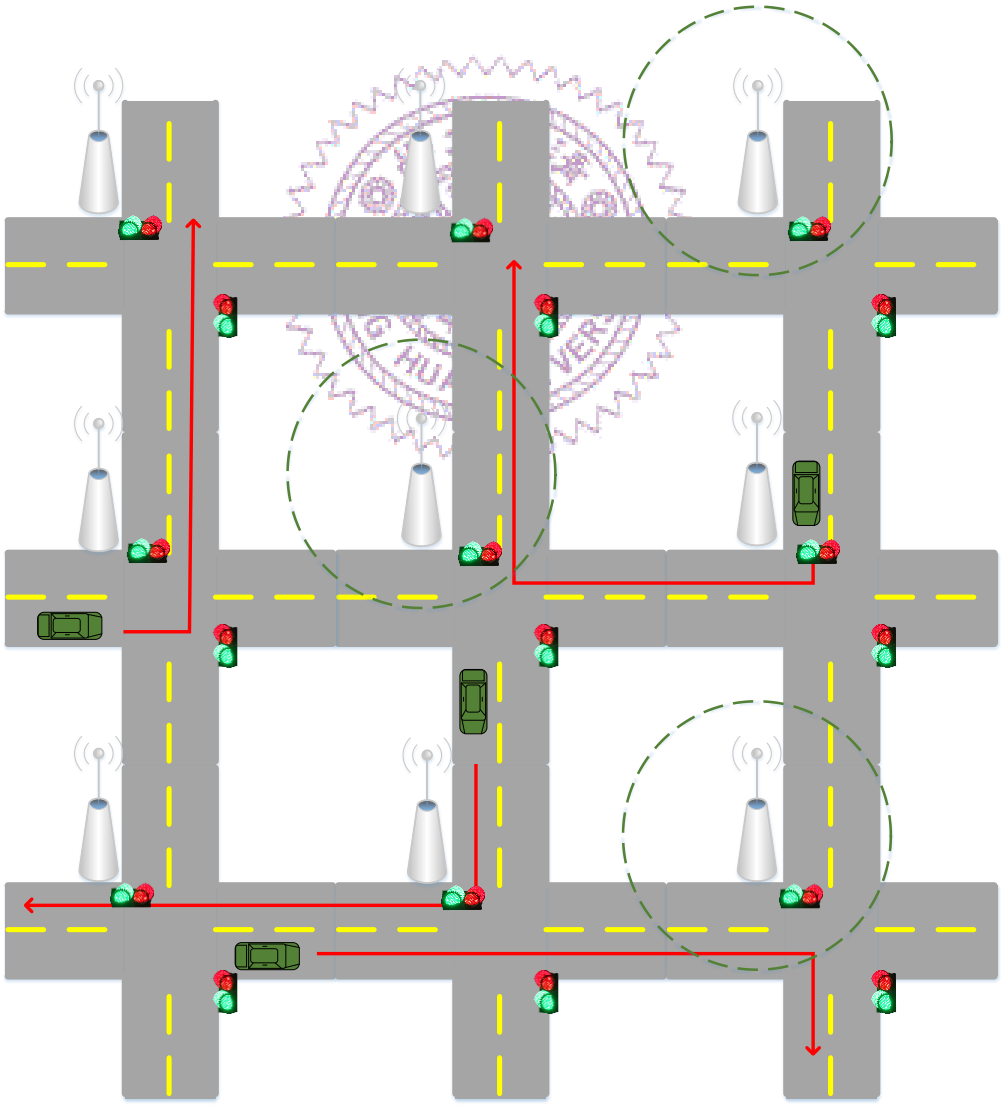


圖 1.1 VanetURSim 產生的車載網路模擬環境示意圖

考量了以上的因素，我們希望能夠在 *ns-3* 建立車載網路的模擬環境，並且能夠真實的反應出車輛在現實環境中的行為模式，所以我們基於 *ns-3* 的架構底下，實作了 *Vanet Mobility Module for Urban Road Networks* (VanetURSim)，VanetURSim 是可以直接掛載在 *ns-3* 上的模組，由於在設計時採用 ns-3 模組化的形式，使用者可以直接將 VanetURSim 與 *ns-3* 上其他模組結合，比起利用其他模擬器產生資料並且外掛入 *ns-3* 當中，VanetURSim 提供簡易的建立模擬環境方式，並且不需額外撰寫程式和考慮相容的問題，使用者能夠透過簡單的參數設定，產生需要的車載網路模擬環境，像是地圖大小、車流量等，如圖 1.1 所示，我們能夠在 *ns-3* 上產生車載網路的模擬環境，並且盡量符合現實的車輛行為模式，透過 VanetURSim，使用者能夠建立 manhattan grid map，並且在每個十字路口上都具備有路側單元以及紅綠燈，透過參數的設定可以創建車輛及指定車輛的出現間隔，而且車輛除了會在十字路口處被創建，也會從道段中的某個位置出現，這些車輛會在地圖上按照給定的路徑移動，並且遵守一般的交通規則。

另外，我們提出了在路側單元與車輛傳輸時流量排程的方法，利用機率的方式計算，挑選最適合協助傳輸的車輛，以此來減少不必要的 *beacon message* 傳送，透過這個排成方法，我們希望能夠減少傳輸封包時的能量消耗，我們利用 VanetURSim 產生車載網路的模擬環境，並且結合 *ns-3* 上封包的傳輸模組，就可以觀察流量排程方法的模擬成果，更甚者，利用 VanetURSim，其他設計傳輸機制的研究者，可以檢測傳輸機制的使用情況和探討效能，透過流量排成的模擬，我們不僅能呈現 VanetURSim 的設計成果，也連帶的驗證 VanetURSim 的正確性與可行性。

2. 研究背景

當我們在做研究時，模擬器是一個很重要的部分，不論是在工業、商業或是學術方面，模擬器是被大量使用且十分重要的，模擬器，特別是指網路模擬器，是希望透過軟體的設計，模擬網路的行為並且盡可能的貼近現實的網路環境。

我們的 VanetURSim，是實作在廣為人知的網路模擬器 *ns-3* 上，在這一章節，我們會先介紹 ns-3 的基本特色和架構，之後，我們會介紹如何利用 *ns-3* 產生想要的模擬環境。

2.1. *ns-3* 網路模擬器的特色

ns-3 是公開資源的且獨立的網路模擬器，主要是作為研究與教育使用。*ns-3* 的設計概念是從許多不同的網路模擬器中啟發而來，像是 *ns-2* 和 *GTNetS*，*ns-3* 的特色主要包括了以下幾點：

- **編譯語言**：可以整體使用 *C++*編寫，或是使用 *python* 寫 *script* 檔案
- **記憶體管理**：可以在程式執行的過程中，動態的釋放沒有使用的記憶體空間，可以有效的執行較為大型的模擬。
- **封包與網路架構**：封包標頭各自獨立，每個封包只會有屬於自己相關的標頭，並且網路的傳輸架構符合現實中多層級式的網路傳輸架構。
- **追蹤技術**： *ns-3* 提供了追蹤封包或是節點的技術，能夠產生 *log* 檔案方便使用者監看模擬的數據。

2.2. *ns-3* 的軟體設計架構

ns-3 的官方網站提供關於 *ns-3* 的基礎模組介紹與架構的教程如下：

ns-3 的程式資源主要都放置在 *src* 的目錄底下，並且可以從圖 2.1 中看出 *ns-3* 的軟體設計架構，我們在 *ns-3* 建立模擬環境時是採用由下往上的方式。

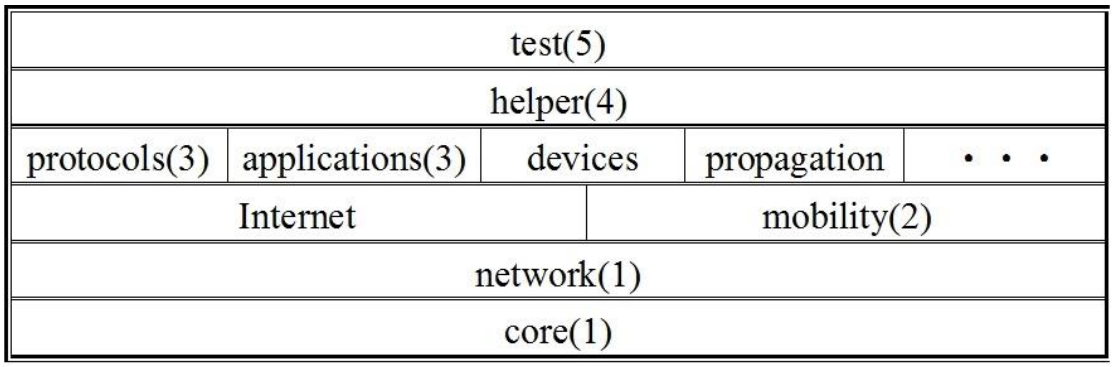


圖 2.1 *ns-3* 的軟體設計架構

core 和 **network** 模組計畫用來產生模擬的核心部分，不僅僅是以網際網路為基底的網路環境，透過 **core** 和 **network** 也可以產生不同類型的網路環境，其餘在上面的模組都是各自獨立的，並且能夠產生特定的網路或裝置。

VanetURSim 是建立在屬於 mobility 的層級，使用者可以透過 VanetURSim 產生車載網路地圖，並賦予車輛趨近於現實的行為模式。

2.3. 在 *ns-3* 上建立模擬的例子

在了解 *ns-3* 軟體設計的架構後，我們來看該如何根據 *ns-3* 的架構建立模擬，舉例來說，假設我們想要創建兩個節點，而這兩個節點會隨機移動，並且在移動的過程中，這兩個節點會互相的傳送封包。

根據 ns-3 的架構，我們採用由下而上的方式建立模擬，第一步，透過 **core** 模組 (如圖 3.1 (1))和 **network** 模組 (如圖 3.1 (1))創建兩個具有網路功能的節點，第二步，透過 **mobility** 模組 (如圖 3.1 (2))賦予這兩個節點隨機移動的能力，第三步，透過 **protocols** 模組 (如圖 3.1 (3))和 **applications** 模組 (如圖 3.1 (3))選擇我們想要的傳輸機制以及應用，第四步，透過 **helper** 模組 (如圖 3.1 (4))將之前的模組互相連接起來，最後，在 **test** 模組 (如圖 3.1 (1))當中設計我們想要模擬的情況，例如，節點會不停的隨機移動，並且在移動的過程中會互相傳送封包，之後，我們可以透過 *ns-3* 的追蹤技術來觀察模擬的結果。

3. VanetURSim

VanetURSim 是實作在 *ns-3* 上的一個模組，它提供了貼近於現實的車載網路環境，使用者可以透過 VanetURSim 在 *ns-3* 上模擬自己實驗數據。基於 *ns-3* 的架構與車載網路的組成元素，我們將 VanetURSim 切割成許多組件，在這一章節，我們會先介紹 VanetURSim 的組織架構，之後會介紹 VanetURSim 的設計理念。

3.1. VanetURSim 組織架構

VanetURSim 是由許多組件所建構而成，在看整體的系統架構之前，我們必須了解什麼是 Manhattan grid map，曼哈頓位於美國的紐約，如圖 3.1 所示，由於其道路規劃從空中鳥瞰猶如棋盤一般呈現方格狀，所以一般稱方格狀的地圖為 Manhattan grid map。

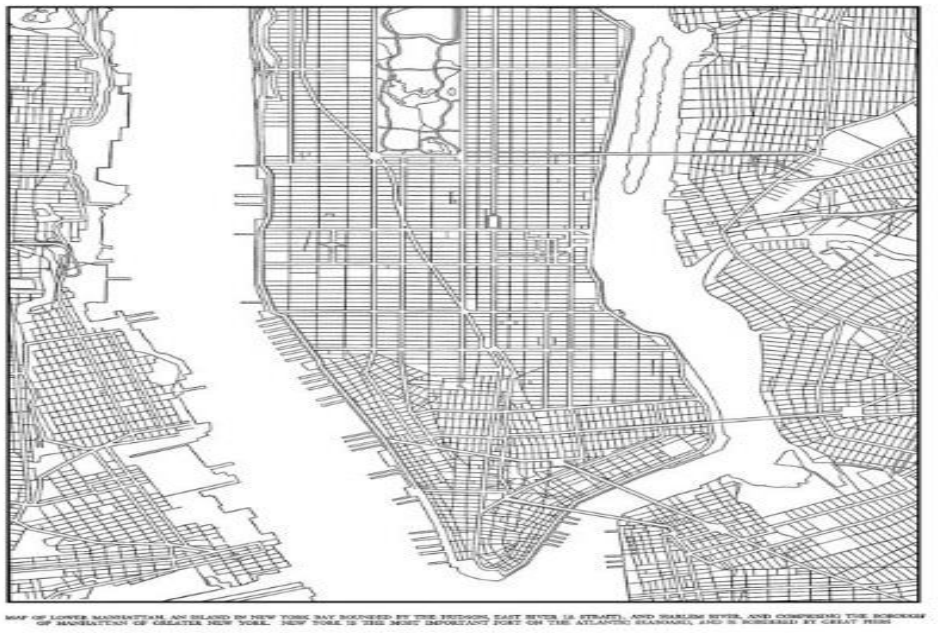


圖 3.1 1944 年紐約曼哈頓地圖

一般的車載網路環境是由地圖、路側單元、紅綠燈、車輛所組成，有鑑於此，我們在設計時考量了各個組件的關聯性與具備的行為能力，將 VanetURSim 分成許多組件，如圖 3.2 所示， 在我們的模組當中，會先建立一張 Manhattan grid map，在 Manhattan grid map 中的每一個十字路口，我們都設置有路側單元，而每一個

路側單元都負責控管紅綠燈與鄰近的道路，在每一段道路上，都存有一張車輛表單，車輛表單內的車輛含有自己的移動路徑與移動模式。

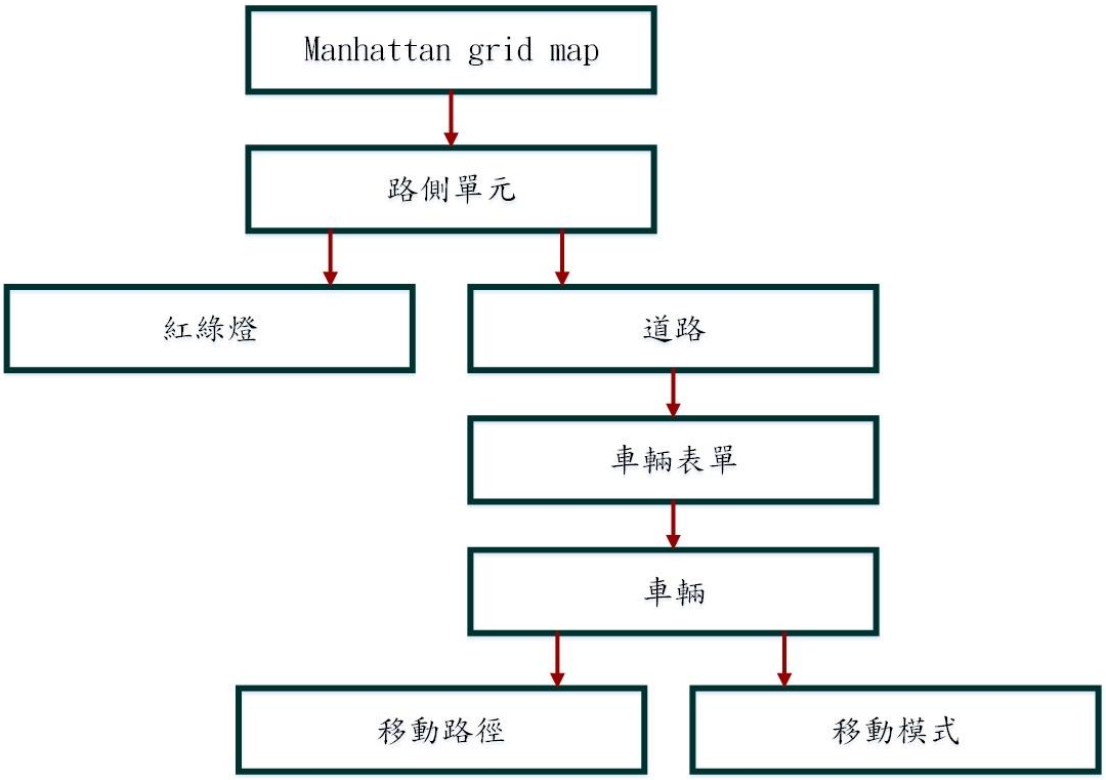


圖 3.2 VanetURSim 組織架構

3.2. 設計理念

在這一小節，我們會描述各個組件的設計理念與基本功能，我們將採取由下到上的方式介紹各個組件並且敘述結合的方式和功能

- 移動路徑 (Trajectory)

因為一般駕駛並不會漫無目的地開車遊蕩，駕駛者通常會有一定的目的性存在，所以我們的移動路徑定義為從起始點到目的地的移動軌跡。在移動路徑的組件中我們定義了一些基本功能：

  - Create Trajectory：透過傳入參數地圖的長、寬，以及起始點和目的地，並給予選定車輛，創建一條從起始點到目的地的移動軌跡。
  - Delete Trajectory：透過傳入參數路徑，刪除整條路徑。

- 移動模式 (Moving Pattern)

通常在一個陌生的環境，駕駛必須借助導航來規劃行車路線，而後駕駛透過得到的行車路線行進，並遵守相關的交通規則，才能順利的抵達目的地。有鑑於此，我們將移動模式設定為駕駛會遵守交通規則，並且按照給定的移動路徑行進。在移動模式的組件中我們定義了一些基本功能：

- Check Direction：透過傳入參數車輛位置、路徑，將車輛位置與當前欲前往的十字路口做距離的比對，確認目前的車輛的移動方向。
- Check Traffic Light：透過傳入參數車輛位置、路徑，比對車輛位置與欲前往的十字路口，確認目前的紅綠燈號誌。
- Move：透過傳入參數車輛位置、時速、移動方向，使得車輛根據車輛的時速沿著移動方向行進。
- Acceleration：透過傳入參數車輛、加速度，設定車輛的加速度。
- Deceleration：透過傳入參數車輛、減速度，設定車輛的減速度。

● 車輛 (Vehicle)

如圖 3.3 所示，透過定義的移動路徑和移動模式，我們可以創建出具有起始點與目的地的車輛與對應移動軌跡，並且車輛會按照給定的移動軌跡行進。我們設定每一臺車輛會包含一些資訊，像是車輛的編號、車輛的長度、車輛所在位置和車輛的速度。在車輛的組件中我們定義了一些基本功能：

- Create Vehicle：透過傳入參數地圖的長、寬，以及車輛的速度、路徑、移動模式，隨機在地圖的某一點創建車輛。
- Delete Vehicle：透過傳入參數車輛編號和車輛表單，根據指定車輛的編號將車輛移除。
- Set Speed：透過傳入參數車輛、移動速度，設定車輛的時速。
- Set ID：透過傳入參數車輛、編號，設定車輛的編號。
- Set Location：透過傳入參數車輛、位置，設定車輛所在的位置。

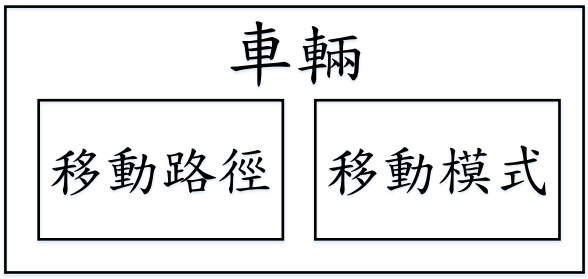


圖 3.3 車輛的結構組成

- 車輛表單 (Vehicle List)

為了方便尋找到任一道路上的任一車輛，我們在每一段道路上都存有一張車輛表單，表單中含有該路段上的每一臺車輛，透過指定道路與車輛編號，能夠將搜尋的時間有效的降低。在車輛表單的組件中我們定義了一些基本功能：

- Insert In List：透過傳入參數車輛表單、車輛、車輛編號、位置，讓車輛根據距離十字路口的遠近插入車輛表單。
- Delete From List：透過傳入參數車輛表單、車輛編號，根據車輛編號將車輛移出表單。
- Search List：透過傳入參數車輛表單、車輛編號，根據車輛的編號找出車輛在表單中的位置。

- 道路 (Roadway)

每一條道路都具有方向性，假設是雙向道，則在我們的道路模組中會記錄成兩條不同方向的道路。每一條道路透過車輛表單，可以監控屬於該道路的車輛，方便使用者去做調整或監看。在道路的組件中我們定義了一些基本功能：

- Display All Vehicle：透過傳入參數道路，顯示該路段上所有車輛的所在位置。
- Refresh Vehicle：透過傳入參數道路，將該路段上的所有車輛做位置、速度或方向的更新。
- Destroy List：透過傳入參數道路，將該路段上所有的車輛移除。

- 紅綠燈 (Traffic Light)

在現實的環境，通常每一個十字路口上都有設置紅綠燈，並且車輛會以當前的顯示燈號做為判斷是否移動的依據，根據現實的考量，我們在每一個十字路口上都設置紅綠燈，並且車輛必須遵守紅燈停、綠燈行的準則。在紅綠燈的組件中我們定義了一些基本功能：

- Initial Traffic Light：透過傳入參數路側單元與起始燈號，並給定更換燈號間隔，設定每一個紅綠燈的初始號誌。
- Change Traffic Light：透過傳入參數路側單元、當前的紅綠燈號誌，更換紅綠燈的號誌。

- 路側單元 (Road-Side Unit)

每一個路側單元都包含四條道路和一些資訊，像是所在位置、紅綠燈號誌、通訊範圍和鄰近的道路資訊，如圖 3.4 所示，路側單元可以透過道路中的車輛表單，快速地找出在自己通訊範圍內的車輛，並且根據選定的 Routing Scheme 來決定是否要與車輛做傳輸，並且路側單元可以控管屬於該十字路口的紅綠燈、道路以及車輛。在路側單元的組件中我們定義了一些基本功能：

- Show Light：透過傳入參數路側單元，顯示該路側單元的紅綠燈號誌。
- Vehicle Amount：透過傳入參數路側單元，統計屬於該路側單元的車輛。
- Refresh Infrastructure：透過傳入參數路側單元，更新屬於該路側單元內的紅綠燈、車輛表單和車輛。

- Manhattan grid map

Manhattan grid map 是一張方格狀的地圖，它可以透過底下的組件來控管整個模擬，並且含有一些地圖設定的資訊，像是道路長度、每條道路的速限、產生車輛的時間間隔、紅綠燈號誌轉換的時間間隔等。在 Manhattan grid map 的組件中我們定義了一些基本功能：

- Refresh Map：透過傳入參數地圖長、寬，更新整個地圖的情況。
- Initial Map：透過傳入參數地圖長、寬和初始參數，設定地圖的初始參

數與創建包含的物件，像是路側單元、紅綠燈等。

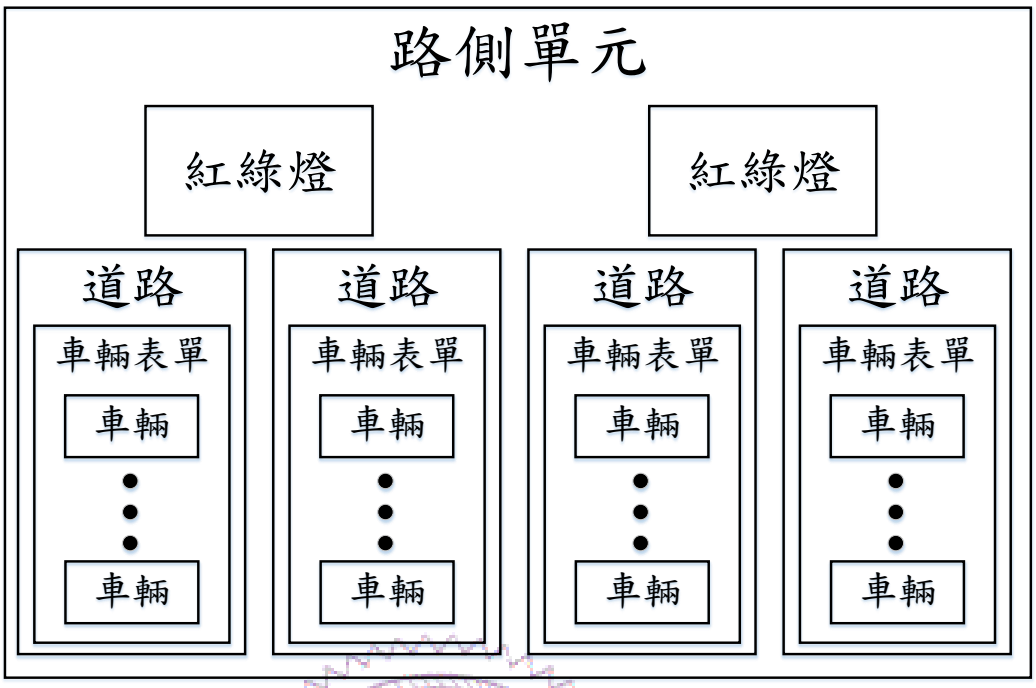


圖 3.4 路側單元的結構組成

4. VanetURSim 實作

在這一章節，我們會先詳細說明各個模組的實作方式，之後會介紹如何利用 VanetURSim 實際產生車載網路的模擬環境。

4.1. 模組實作

- 移動路徑

在描述起始點到目的地的軌跡時，都需要依照某些參考的物件，例如描述餐廳的位置時，我們說距離此地兩個街區，大約 200 公尺，參照物就是道路長度，或是當爬山時，我們會說，再走 10 分鐘左右就到山頂，參照物就是人的移動速度。因此，在我們設計移動路徑模組時，我們可以有許多不同的設計方式，然而，在 VANETURSIM 的模擬環境中，車輛速度或是道路長度都是屬於使用者自定義的參數，為了方便比較與查看數據，我們希望移動路徑能夠有固定的顯示型態，不會隨著使用者定義的參數而改變，因而在實作移動路徑時，如圖 4.1 所示，我們將地圖座標化，並且設定地圖左下角的十字路口座標為$I_{0,0}$，透過這些設定，移動路徑可以看成是一連串座標的組合，如圖 4.1 中，橘色線代表從起始點 S 到目的地 D 的移動路徑，就會顯示成$I_{0,0} \rightarrow I_{1,0} \rightarrow I_{2,0} \rightarrow I_{2,1}$。由於考量到駕駛有可能會臨時改變移動路徑，為了方便插入或刪除新的移動路徑，我們採用鏈接串列(Link-list)的方式儲存移動路徑。

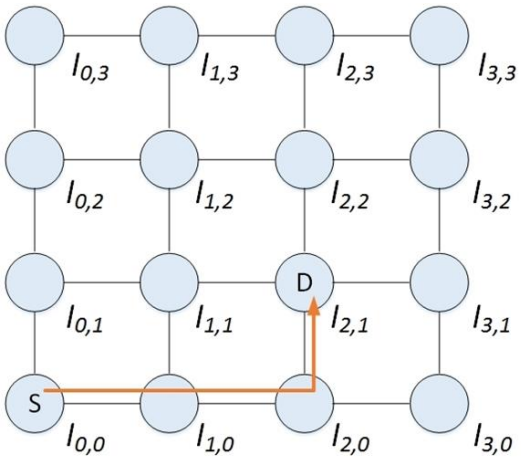


圖 4.1 座標化的地圖

另外，在移動路徑的部分，我們提供了兩種創建移動路徑的方式供使用者選擇，一種是給定起始點和目的地，再產生最短的移動軌跡，另一種是隨機產生起始點和目的地，再產生最短的移動軌跡，所謂最短的移動軌跡，指的是經過的十字路口數最少的路線，如圖 4.1 中，起始點 S 到目的地 D 經過的十字路口數，假設包含目的地，最少的十字路口數量為 3，在建立移動路徑時，我們會統計經過的十字路口數，並將其轉換成對應的移動方向，如圖 4.1，從起始點 S 到目的地 D 總共經過 3 個十字路口，轉換的對應方向為右、右、上，我們將這三個方向做隨機排列，並從 6 種當中亂數取出一組作為移動路徑。

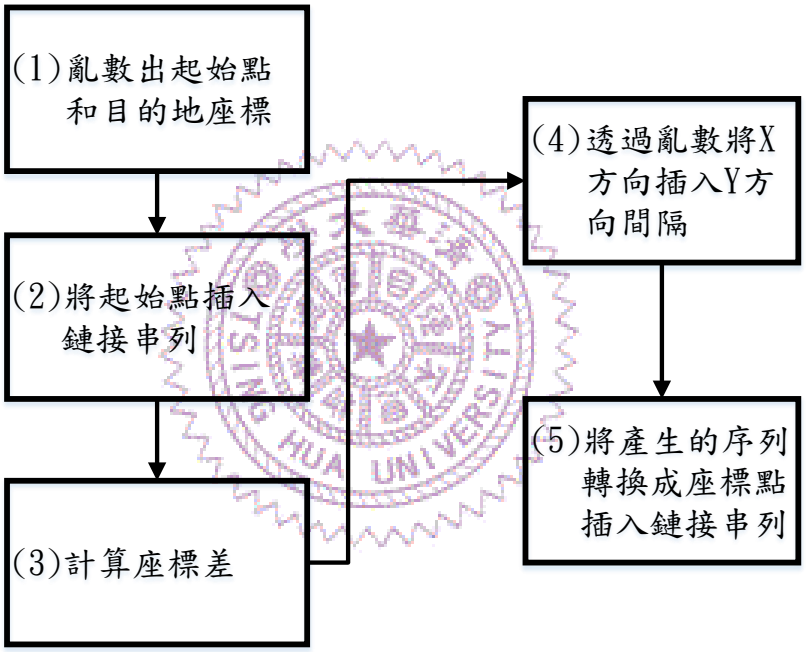


圖 4.2 隨機產生移動路徑流程圖

更進一步說，移動路徑的資料組態包含三個部分，分別是座標點 X、座標點 Y 和下一個十字路口的座標點，透過這個資料鏈接的形式，可以方便使用者搜尋整條路徑，在創建移動路徑時，我們提供函式 *Random_Create_Path*，實際的運作如圖 4.2 所示：

- 步驟一：亂數出起始點和目的地座標，分別存入參數 *Source_X*、*Source_Y*、*Destination_X*、*Destination_Y*。
- 步驟二：將起始點座標透過函式 *Insert_Node* 插入鏈接串列中，函式

Insert_Node 會找出鏈接串列的最後一個位置，並將傳入的座標點插入鏈接串列。

■ 步驟三：計算座標差並儲存，座標差分為水平與垂直，分別是 *X_Difference* 和 *Y_Difference*，

■ 步驟四：利用亂數從 *0* 到 *Y_Difference+1* 隨機取出 *X_Difference* 組數值，按照順序將 *X* 方向插入對應的 *Y* 間隔中。

■ 步驟五：將產生出來的序列透過方向的轉換，變成座標點，透過 *Insert_Node* 依次放入鏈接串列中。

- 移動模式

一般車輛的移動模式，通常會按照規畫好的移動路徑移動，並且會遵守交通規則，為了實作此類的移動模式，我們創造了兩個變數，一個是綁定的十字路口 (*Binding RSU*)，一個是暫時目的地 (*Temp Destination*)，綁定的十字路口是當前車輛所需遵從紅綠燈號誌的十字路口，暫時目的地則是車輛將要前往的十字路口，當車輛剛被創建的時候，綁定的十字路口就是車輛的初始位置，而暫時目的地則是接下來要前往的十字路口，這樣的設計方式是為了判斷車輛的移動方向，並防止車輛忽略當前十字路口的紅綠燈號誌。

如圖 4.3 所示，首先我們會根據目前車輛的位置與暫時目的地設定車輛的移動方向，接下的部分，根據我們的觀察將車輛分成兩成類型，一種是距離十字路口最近的車輛，另一種則是其餘排在同一條道路上的車輛，會分成這兩種是因為我們發現車輛的行為模式在排頭與其餘位置會有所區隔，在排頭的車輛，移動時由於前面不會有其他車輛，主要需要考慮的是與十字路口之間的距離和紅綠燈號誌，而剩餘的車輛，由於前面存有其他的車輛，在移動時僅僅需要考慮與前車的距離即可。

如果是第一輛車，在車輛位置加上移動速度後，假如不會超過十字路口，則不需要考慮紅綠燈號誌，只須根據車輛的行進方向改變車輛所在的位置，另一方

面來說，假如會超過十字路口，則必須檢查綁定的十字路口的紅綠燈號誌，如果是紅燈，則表示這次的移動距離，理論上無法超過十字路口，考量現實的駕駛情況，通常駕駛在前方紅綠燈號誌為紅燈的時候，會將速度減緩，並且停靠在十字路口處，有鑑於此，在我們的模組設計上，如果前方的紅綠燈號誌為紅燈，且這次的移動距離會超過十字路口，便將移動距離更改為最遠可到達的位置，對排頭的車輛而言，就是十字路口處。如果不是第一輛車，在移動時僅需要考慮與前車的距離即可，同理，如果移動距離超過與前方車子的間隔距離，則將車輛移動到最遠可到達的位置，也就是緊靠在前方車輛的後面。

當車輛從一個十字路口離開，不論是直行、左轉或右轉，車輛的行進方向、綁定的十字路口和暫時目的地都必須做更新，在更新之後假如暫時目的地與所在位置相同，則表示車輛沒有下一個目標，也就是車輛已經抵達目的地，這個時候我們會將車輛從車輛表單移除，如果還未抵達目的地，則將車輛轉移到另一條路段的車輛表單內。

更進一步說，我們提供函式 *Vehicle_Moving*，利用 *Check_Direction* 確認車輛目前的移動方向，*Check_Direction* 會根據車輛的暫時目的地，比對車輛的位置，回傳 *0/1/2/3* 四個數值，分別對應到移動方向上/下/左/右，之後利用變數 *Distance* 檢測是否屬於第一輛車，*Distance* 設有初始值*-10*，每當檢查完一部車輛就會變更為該車輛的位置，在檢查完車輛是不是屬於排頭後，會呼叫函式 *Check_Location*，*Check_Location* 會根據傳入的 *Distance* 決定是否需要考量紅綠燈號誌，如果不需要則會直接根據車輛移動速度和 *Distance* 決定車輛位置，如果需要考量紅綠燈號誌，則會呼叫 *Check_Traffic_Light* 檢查綁定的十字路口的紅綠燈號誌，*Check_Traffic_Light* 會回傳兩個數值，分別是 0 或 1，0 表示紅燈，1 表示綠燈，並且在移動後會呼叫函式 *Change_Trajectory*，將移動過的十字路口從移動路徑中清除，*Change_Trajectory* 會根據傳入的車輛位置比對移動路徑當中的第 一個十字路口，如果車輛已經經過該十字路口，則會將該十字路口座標從移動路徑中清

除，之後會更改車輛的暫時目的地和綁定的十字路口，最後，會呼叫函式 *Check_Trajectory*，如果發現當前的路徑已經是 *NULL*，則會呼叫函式 *Delete_Vehicle* 將車輛從車輛表單中移除，*Delete_Vehicle* 會根據傳入的車輛編號，將對應的車輛刪除。

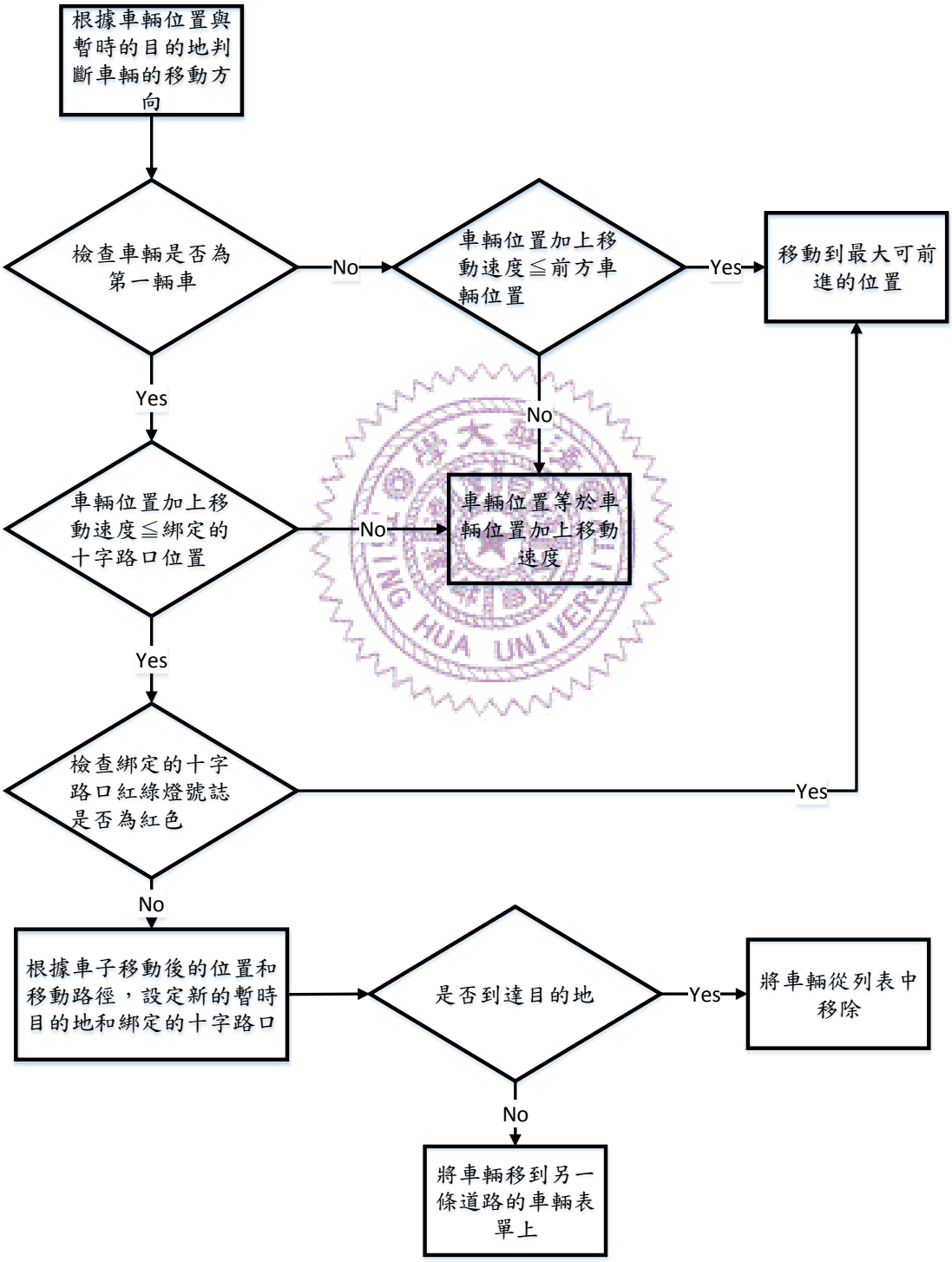


圖 4.3 實作移動模式流程圖

- 車輛

車輛的模組可以看成是一個物件，在創建車輛時，我們會先亂數出一條移動路徑，並根據移動路徑的起始點設定車輛所存在的位置，起始點座標所轉換的車輛初始位置，會隨著使用者自定義的道路長度而有所調變，在創建車輛時會根據傳入的參數設定使用者的車輛長度和車輛速度，另外，在創建車輛時會給定車輛編號，車輛編號的數值代表車輛在模擬中是第幾臺被創建的車輛，由於車輛的起始點設定在十字路口上，並且必須遵從該十字路口的紅綠燈號誌，而每一個十字路口包含四條道路，該如何選擇將車輛放入哪一條道路是必須解決的問題，如圖 4.3 所示，紅色箭號表示必須遵從該十字路口紅綠燈號誌的道路，根據箭號可以發現，屬於該道路的車輛行進方向必然與箭號相同，所以當我們在決定該將車輛放入哪條道路上時，會將車輛的移動路徑取出，比對移動路徑內第一與第二的座標點，計算出車輛的初始移動方向，並根據車輛的初始移動方向將車輛放入對應的道路中，如圖 4.1 中的橘色路徑，車輛的起始位置為$I_{0,0}$，當我們要決定放入十字路口$I_{0,0}$的哪條道路上時，會取出移動路徑的前二座標點，也就是$I_{0,0}$與$I_{1,0}$，所以車輛會放入十字路口$I_{0,0}$中方向向右的道路當中。

更進一步說，車輛模組內含有車輛編號 *ID*、車輛位置 *Location_X*、*Location_Y*、行進方向 *Direction*、移動速度 *Speed*、車輛長度 *Vehicle_Length*、旗標 *Flag* 移動路徑 *Path*，我們提供創建車輛函示 *Create_Vehicle*，實際運作方式如圖 4.4 所示：

- 步驟一：車輛創建時，會先呼叫 *Random_Create_Path*，並將移動路徑存入 *Path* 當中。
- 步驟二：透過函示 *Check_Trajectroy* 取出移動路徑的第一個座標點，將取出的座標點乘上道路長度，存入 *Location_X* 和 *Location_Y*，如此車子的實際位置就會適應變化的道路長度。
- 步驟三：將使用者定義的車輛初始速度和車輛長度分別存入 *Speed* 和 *Vehicle_Length* 中，並設定 *Flag* 為 *0*。

■ 步驟四：透過呼叫函式 *Change_Trajectory*，將車輛的暫時目的地設為下一個要移動到的十字路口，再呼叫函式 *Check_Direction* 得出車輛的初始移動方向。

■ 步驟五：呼叫函式 *Insert_Vehicle* 將車輛插入對應道路上的車輛表單內。*Insert_Vehicle* 找到車輛表單的最後面，並將車輛插入。

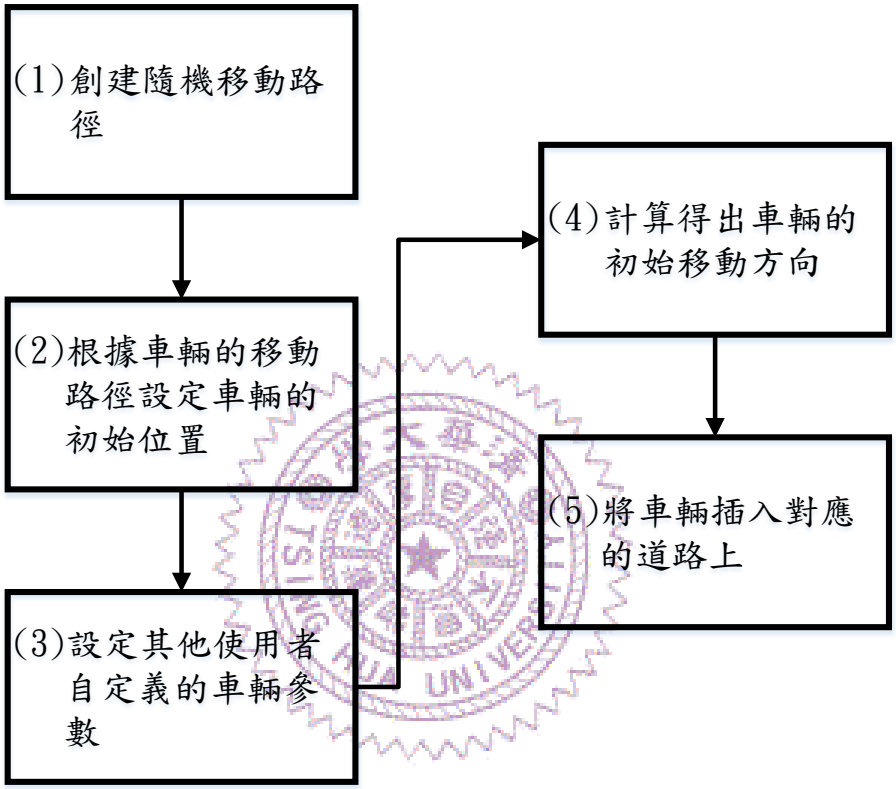


圖 4.4 實作車輛創建流程圖

隨著車輛創建的設計，一個新的問題產生了，當有許多車輛被創建時，車輛的初始位置有可能會相同，並且車輛的初始移動方向也有可能會相同，在這樣的情況下，車輛在地圖的顯示上，會有車輛重疊的情況發生，為了解決這個問題，當我們在將車輛插入表單前，會先在車輛表單中確認該位置是空的，如果不是空的，則會去搜尋車輛表單中的空位，將車輛插入該位置，同時，車輛的起始點也會跟著變動，並且將暫時目的地設回對應的十字路口。這樣的設計會使得我們的模擬變得更具變化性，也更為貼合現實，車輛的起始點並不單純的全部都是十字路口處，也會產生在路邊啟動的車輛。

更進一步的說，我們在步驟四與步驟五之間額外呼叫函式 *Check_Empty*，*Check_Empty* 會搜尋傳入的車輛表單，並且回報當前可插入車輛的位置，當收到回傳值後，會與車輛位置 *Location_X* 和 *Location_Y* 做比較，如果值不相同表示有車輛重疊的狀況發生，*Location_X* 和 *Location_Y* 會將數值變更為空位的位置，並且將暫時的目的地設定成當前的十字路口，Check_Empty 的實際運作方式如圖 4.5 所示：

- 步驟一：創建變數 *Empty_X* 和 *Empty_Y*，表示在車輛表單中的空位位置，並且給定初始值為新創建車輛的初始位置。
- 步驟二：檢查車輛表單中的車輛位置，如果 *Location_X*、*Location_Y* 與 *Empty_X*、*Empty_Y* 相同，則表示發生車輛重疊問題。
- 步驟三：如果發生重疊問題，會將 *Empty_X*、*Empty_Y* 的數值往道路方向的反方向調整一個 *Vehicle_Length*
- 步驟四：重複步驟二、三直到新創建的車輛有空位插入為止。

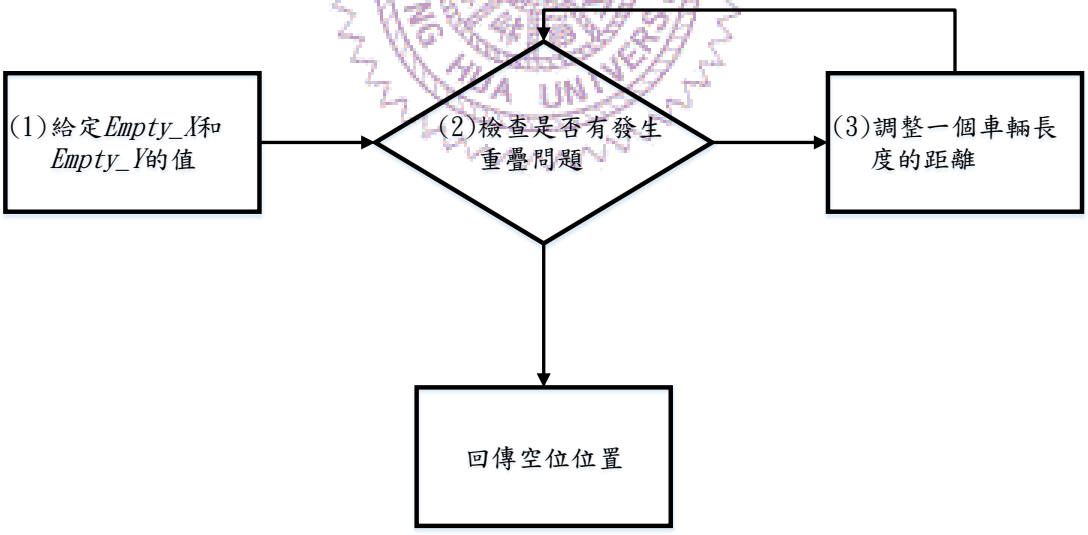


圖 4.5 實作車輛插入表單流程圖

- 車輛表單

在車輛表單的設計上，由於車輛頻繁的進入與離開路段，為了方便插入與刪除車輛，我們採用 link-list 的方式儲存車輛，然而，車輛在移動時，並不是全部

同步移動，因而會產生車輛移動時誤判的可能，如圖 4.6，假如車輛 1 與車輛 2 具有相同的速度，且車輛位置更新的順序為車輛 1、車輛 2，則當車輛 1 判斷是否能移動時，由於已經緊靠車輛 2，所以不會移動，之後車輛 2 則會往前移動，然而，如果從整體來看，車輛 1 與車輛 2 都應該向前移動，這就是由於更新順序所引起的誤判。為了解決這個問題，我們將車輛表單更動為 order-link-list，車輛插入的順序會依照距離十字路口的遠近做排序，更新的順序就是從第一輛車開始依序向後，如此便可以避免誤判的問題。

更進一步說，我們修改函式 *Insert_Vehicle*，在函式 *Insert_Vehicle* 中不會單純的搜尋到表單的最後就將車輛插入，而是會按照距離綁定十字路口的距離作排序，車輛距離綁定的十字路口越近，則在車輛表單中就越靠前，所以在插入車輛時，會比較車輛的位置，將車輛插入到對應的表單位置當中。

另一方面，當車輛在車輛表單中做轉移的時候，亦即從一個路段換到另一個路段，由於更新的順序存有先後的關係，車輛有可能會被更新到兩次而連續移動兩次，為了防止此類問題的發生，車輛設置有旗標記錄車輛是否被更新。

車輛表單的更新方式，詳細的運作方式如下：

- 對屬於該路段的車輛從排頭開始，直到最後一臺車為止。
- 檢查車輛的 Flag 值，Flag 的值分為 0、1，分別代表尚未移動與移動過的車輛。如果車輛的 Flag 值為 0，則呼叫 *Vehicle_Moving* 更新該車輛的位置。
- 每臺車輛移動後，都會更改車輛內 *Flag* 的值為 *1*。
- 當表單內的車輛全數移動完畢後，會計算該道路上的車輛總數，方便模擬時的統計或監看。

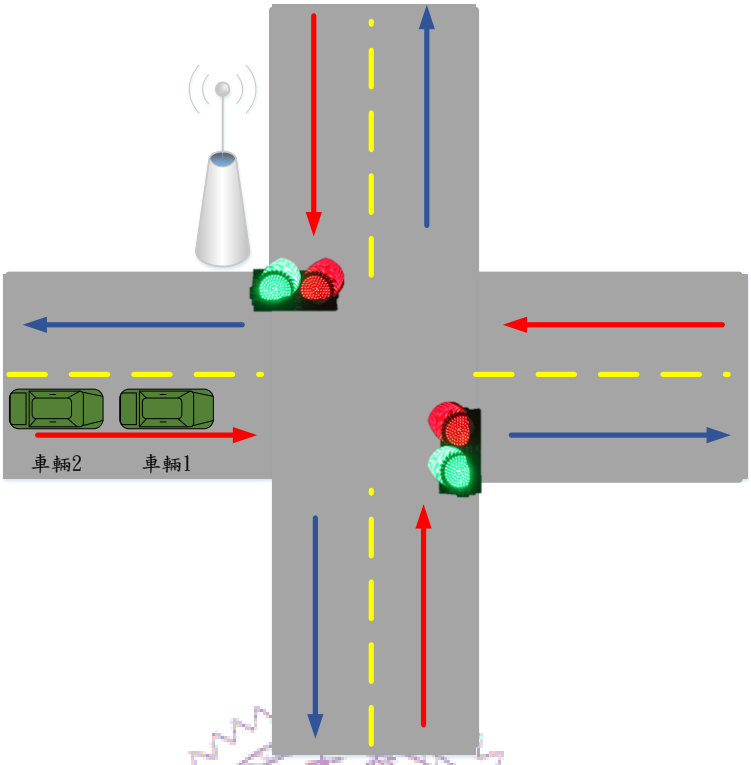


圖 4.6 單一路側單元內的物件

- 道路

在道路的設計上，我們讓道路具備有方向性，如圖 4.6 所示，傳統的雙向道在我們的模組內會顯示成兩條道路，道路總共具有四個方向，上、下、左、右，每條道路上只有含有與其同個方向的車輛，並且每個路側單元內包含四條道路，這四條道路上的車輛都必須遵從當前路側單元上的紅綠燈號誌，如圖 4.6 所示，紅色箭號道路為一路側單元內包含的四條道路，而藍色箭號道路則為鄰近的路側單元所包含的道路，透過這些具有方向性的道路，可以將地圖視為每個路段皆為雙向道的道路環境。

這樣的設計乍看之下似乎忽略了藍色箭號道路上的車輛與路側單元溝通的可能性，然後仔細觀察後可以發現，藍色箭號的車輛其實可以看成是紅色箭號道路的延伸，車輛如果能夠行進到藍色道路上，必然會經過紅色箭號的十字路口，所以路側單元只需控管四條道路即可。

- 紅綠燈

在紅綠燈的設計上，我們將一組紅綠燈簡易的分為兩個方向，一個是水平方向，一個是鉛直方向，分別給道路方向為左、右與上、下的車輛作為移動依據，透過一個倒數計時器，每組紅綠燈會按照給定的時間做號誌變換。

在一開始，我們必須要給紅綠燈初始號誌，我們提供了函式 *Initial_Traffic_Light*，在 *Initial_Traffic_Light* 當中，根據傳入的地圖長、寬，*Map_Height* 和 *Map_Width*，針對每一個十字路口的紅綠燈分別做號誌的亂數處理，紅燈與綠燈分別表示成 0、1，先對垂直方向的紅綠燈亂數出紅綠燈號誌，水平方向則設置為相反的號誌燈號。

另一方面，我們提供函式 *Is_Change_Traffic_Light*，實際的運作方式如圖 4.7 所示：

- 步驟一：我們在函式中定義 *static* 的計數器 *Timer*，給定初始值 *0*。
- 步驟二：將 Timer 加 1，並且與紅綠燈號誌轉換間隔 *Traffic_Light_Period* 比較，如果相等則呼叫函式 *Change_Traffic_Light*。
- 步驟三：*Change_Traffic_Light* 會檢測當前的紅綠燈號誌，並且將紅綠燈號誌對調。
- 步驟四：如果 Timer 與 *Traffic_Light_Period* 相等，則將 Timer 回歸 0，

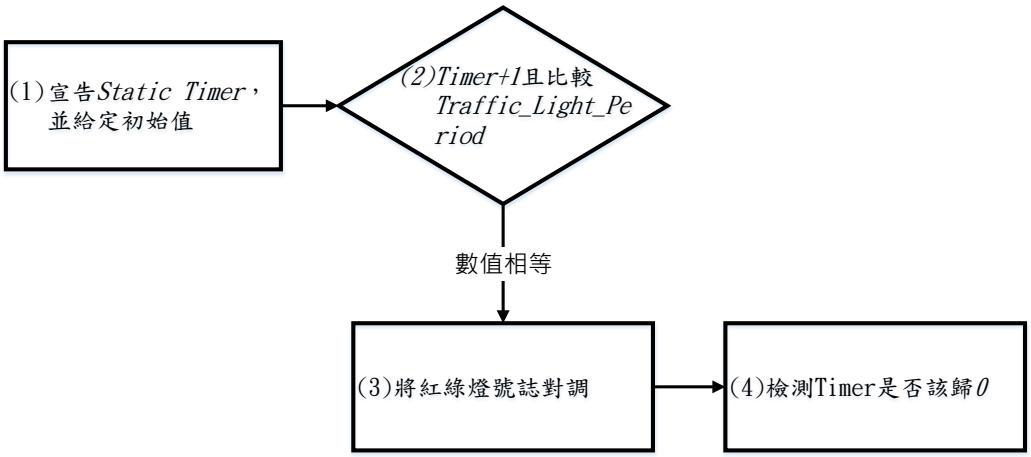


圖 4.7 實作紅綠燈號誌流程圖

- 路側單元

由於路側單元會含有自定義的 *Routing Scheme*，必須要能夠準確的與車輛做溝通，且紅綠燈通常加裝在十字路口上，我們希望路側單元能夠快捷的掌握這些資訊，所以在路側單元的設計上，我們採用上下層的方式，路側單元會包含 Routing Scheme 和底下所有的組件，使用者可以自己定義 *Routing Scheme*，以此做為路側單元與其底下控管的車輛的傳輸決策機制。

我們提供函式 *Refresh_Infrastructure*，可以對底下每一條道路及車輛做控管，實際的運作方式如圖 4.8 所示：

- 步驟一：將每條道路的車輛的 *Flag* 設為 *0*。
- 步驟二：按照上、下、左、右的順序呼叫車輛表單，讓車輛表單對底下的車輛座位置的更新。
- 步驟三：將每個車輛表單內的車輛做加總，確認目前在道路上的車輛總數。
- 步驟四：檢查在通訊範圍內的車輛是否滿足 *Routing Scheme* 的傳輸條件。

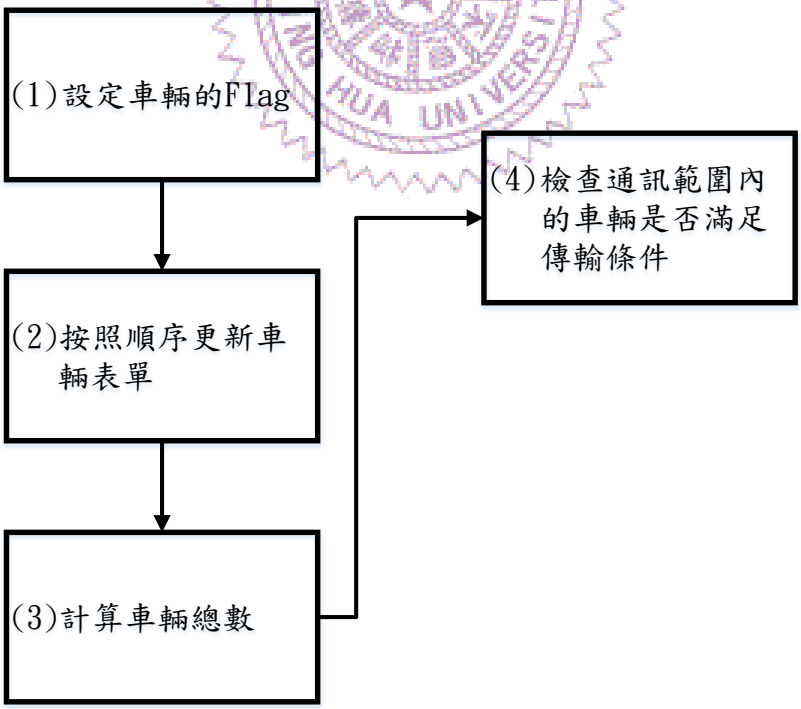


圖 4.8 實作路側單元流程圖

- Manhattan grid map

透過給定地圖的長、寬，可以產生一張 Manhattan grid map，並且會產生路側單元和紅綠燈，透過參數的設定可以快速地產生想要的模擬環境。

我們提供了函示 Initial_Map，Initial_Map 的實際的運作方式如圖 4.9 所示：

- 步驟一：呼叫函式 Initial_Traffic_Light，對每一個十字路口的紅綠燈做初始化燈號的動作。
- 步驟二：將十字路口內的車輛表單中的車輛全部清空。
- 步驟三：將十字路口的座標根據道路長度轉換成實際地圖的位置。
- 步驟四：設定每個路側單元的傳輸範圍。

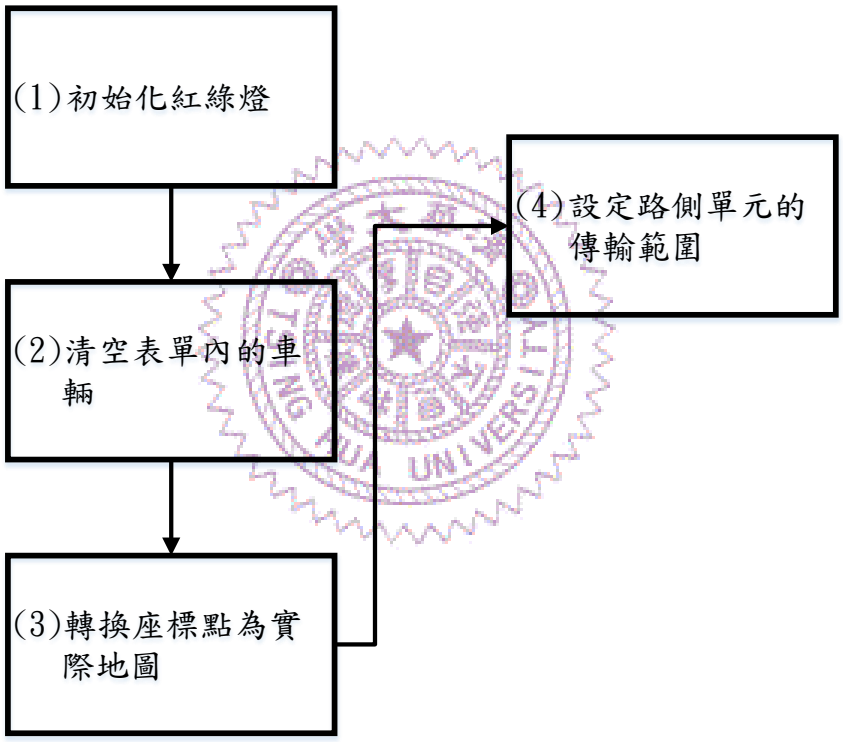


圖 4.9 實作 *Initial_Map* 流程圖

另外我們提供觀察模擬環境數據的函式，分別是 *Show_Light*、*DisPlay_All_Vehicle_With_ID*、*DisPlay_All_Vehicle_Trajectroy*、*Vehicle_Amount*：

- ***Show_Light***：顯示每一個十字路口的當前時間對應的紅綠燈號誌
- ***DisPlay_All_Vehicle_With_ID***：從地圖最左下角的十字路口，向右而上的顯示每一臺車輛當前的車輛編號、綁定的十字路口、暫時的目的地，當前綁定十字路口的紅綠燈號誌、車輛所在的位置、車輛的行進方向、車輛屬於

的道路方向、目的地。

- ***DisPlay_All_Vehicle_Trajectory***：會列出當前模擬環境中的所有車輛 ID，並且會顯示車輛的移動路徑。

- ***Vehicle_Amount***：會回傳當前模擬環境中的車輛總數。

各個觀察數據函式的顯示結果如圖 4.10 所示。

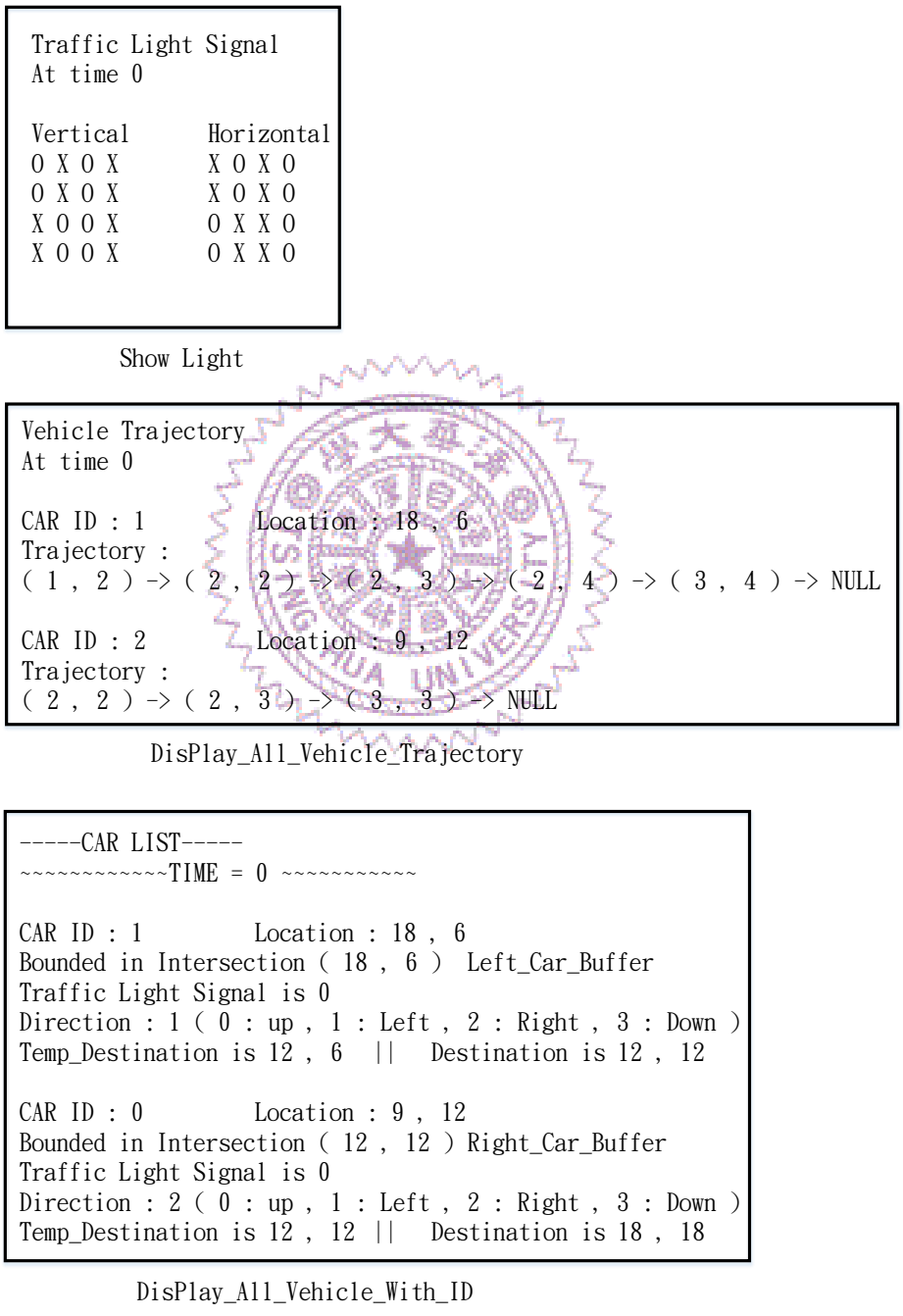


圖 4.10 觀察數據函式顯示圖

4.2. 產生模擬環境

在這一小節，我們將會描述如何使用 VanetURSim 產生對應的模擬環境。如圖 4.11 所示，剛開始我們要設定我們的模擬參數，再來要創建地圖與初始車輛，接著將地圖初始化，地圖的初始化包括將參數匯入、創建路側單元、創建紅綠燈並亂數出初始紅綠燈號誌，接著，判斷使否到達想要模擬的總時間，並且根據倒數計時器決定是否產生車輛以及更換紅綠燈號誌，之後將車輛表單內的旗標還原，在呼叫地圖更新，就可以使得地圖上的車輛按照時間做位置的移動。圖 4.12 為在 *ns-3* 上建立模擬環境的 *pseudo code*。

```
#include "ns3/VANETURSIM.h"

  int main(void){

  int Run_Time = 10000;
  int Vehicle_Period = 5;
  int Traffic_Light_Period = 30;
  int Map_Height = 15;
  int Map_Width = 15;
  int Road_Length = 150;
  int Vehicle_Speed = 15;
  int Vehicle_Length = 3;
  int Initial_Vehicle = 100;
  int Time = 0;

  C_INFRASTRUCTURE MAP[Map_Height][Map_Width];
  Initial_Map(Map);

  while(Time <= Run_Time){
     Is_Create_Vehicle( Time, Vehicle_Period);
     Is_Change_Traffic_Light( Time, Traffic_Light_Period);
     Refresh_Map(Map);
  }

}
```

地圖參數設定

創建地圖和初始化

更新整張地圖

圖 4.11 簡易的車載網路模擬環境產生範例

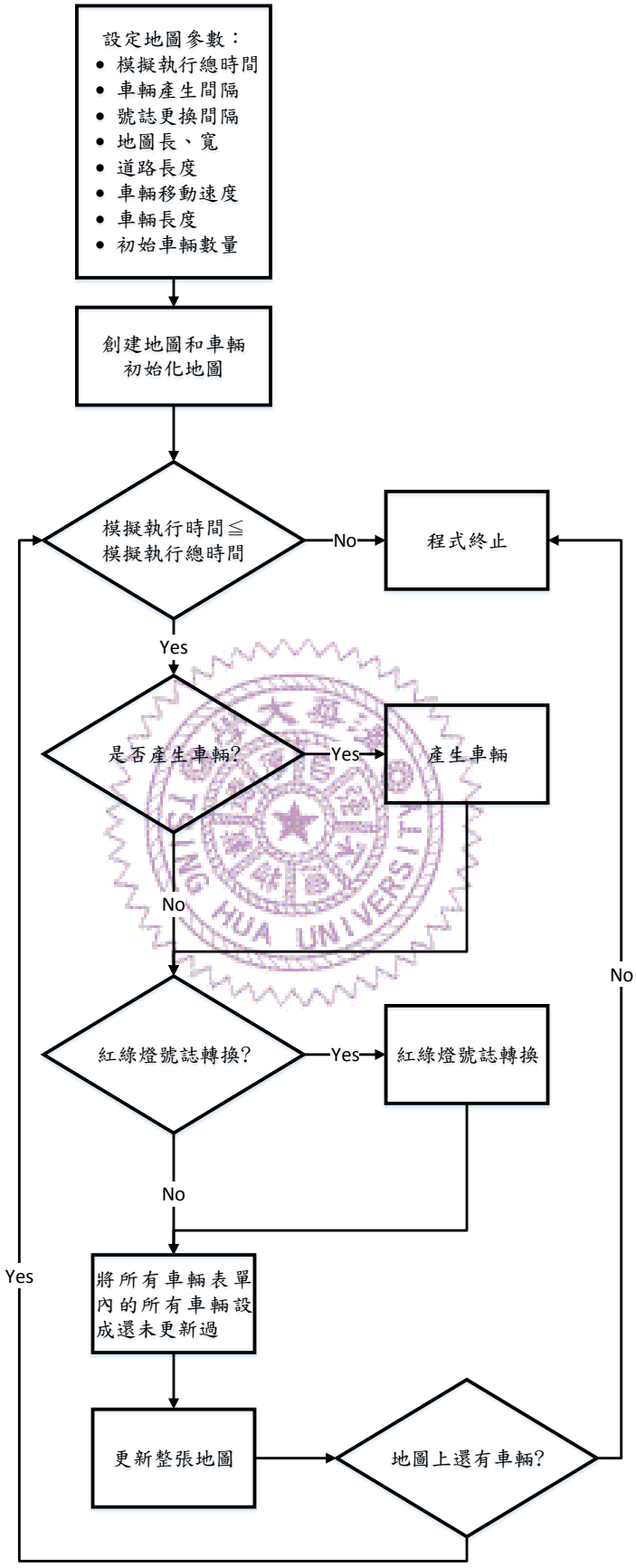


圖 4.12 透過 VanetURSim 建立模擬環境

4.3. 模擬數據分析

圖 4.15 是模擬的實驗結果，圖上的編號為產生數據的時間順序，透過這些數據，可以將數據對應到模擬環境中，如圖 4.13 為模擬環境的示意圖，我們產生的地圖為 4 乘 4 的 Manhattan grid map，地圖上建有兩台車輛，並設定道路的長度為 6 公尺，車輛的速度為 3 公尺每秒。

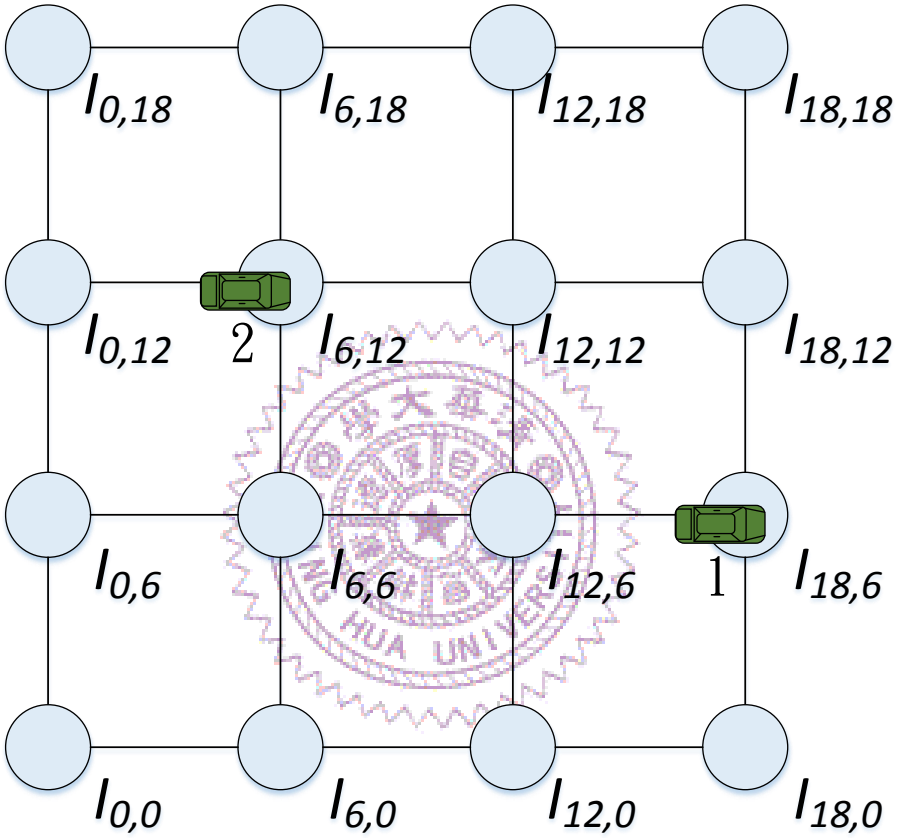


圖 4.13 模擬環境示意圖

在圖 4.15 中可以監看地圖中車輛的訊息，其中 *TIME* 是當前的時間點，*CAR ID* 表示的是車輛編號，*Location* 代表的是車輛當前的位置，*Bounded in Intersection* 代表車輛當前綁定的十字路口，*Left_Car_Buffer* 則代表車輛位於方向為向左的道路上，*Traffic Light signal* 為車輛當前必須遵從的紅綠燈號誌，*Direction* 為車輛的行進方向，*Temp_Destination* 是車輛的暫時目的地，*Destination* 為車輛的目的地。

如圖 4.15.1 和圖 4.14 所示，*TIME = -1* 為剛產生實驗環境時的時間點，以車

輛 1 為例，在第一個時間點，車輛 1 被創建在十字路口(18,6)，並且放置於向左的道路上，當前路口的紅綠燈號誌為紅燈，車輛 1 的行進方向為向左，從暫時的目的地(12,6)與車輛所在的位置可以檢查車輛的行進方向是否正確。在圖 4.15.1 的數據中可以顯示出車輛 1 在當前的時間點由於紅綠燈號誌為紅燈，車輛 1 應該無法移動，為了驗證車輛的行動模式是否正確，我們要檢查在下一個時間點車輛的位置是否有發生變化，從圖 4.15.2 中車輛 1 所在的位置，可以清楚地發現車輛 1 的位置並沒有發生改變。

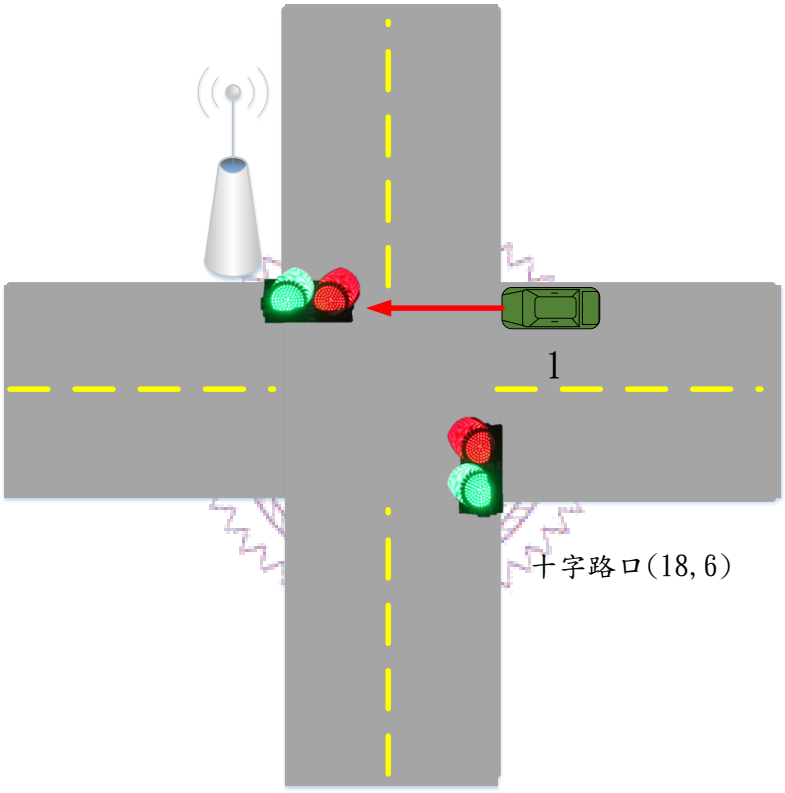


圖 4.14 透過 VANETURSIM 建立模擬環境

另外，在圖 4.5.5 中有顯示 TRAFFIC LIGHT CHANGE，當出現這行字的時候，代表地圖中的紅綠燈號誌發生改變，以車輛 1 為例子，也就是說車輛 1 所需要遵從的十字路口號誌已經變為綠燈，所以當 TIME = 3 時，車輛 1 應該要進行移動，並且移動的距離為車輛 1 所在的位置加上速度，從圖 4.5.5 可以看出車輛 1 的位置從(18,6)移動到了(15,6)的位置，並且由於車輛 1 已經經過十字路口(18,6)，所以車輛 1 所需遵守的十字路口號誌燈號應該為下一個路段的十字路口位置，從

圖 4.15.4 和 4.15.5 可以看出車輛 1 綁定的十字路口從(18,6)換成了(12,6)，此時顯示的紅綠燈號誌為十字路口(12,6)的紅綠燈號誌。

在圖 4.15.7 中，我們可以發現車輛 1 所在的位置(12,9)，已經距離目的地(12,12)只差一次移動的距離，在圖 4.15.8 中可以看出車輛 1 因為到達目的地而被移除。

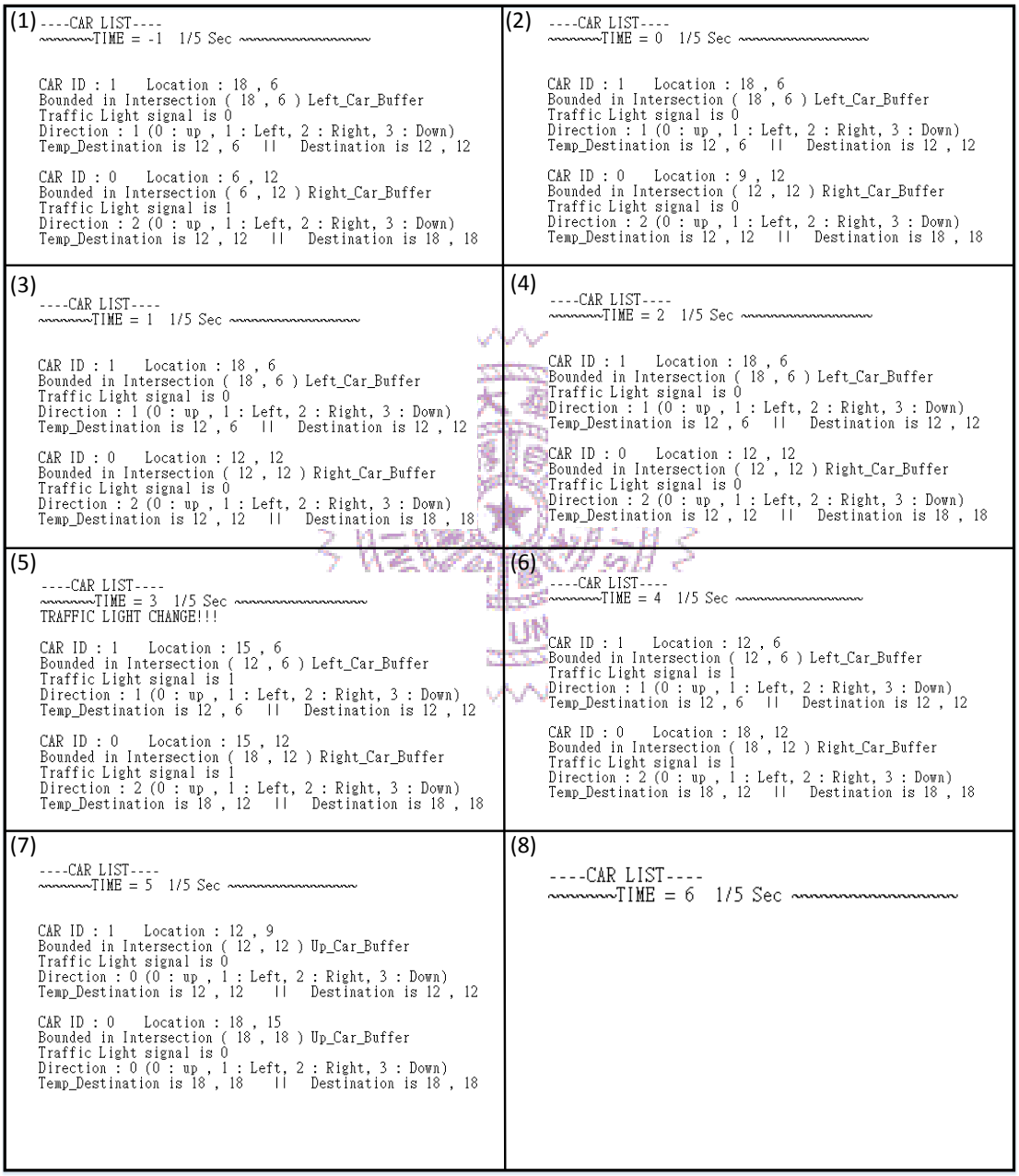


圖 4.15 透過 VANETURSIM 建立模擬環境

5. 在 VDTN 環境下封包傳輸的能量消耗

在車載網路環境中，有些封包會具備較長的存活時限，像是廣告訊息、地方資訊、當天油價等等，隨著駕駛車輛的人數上升，這些類型的封包也隨之遽增，為了能更有效率的處理這些封包，一個新的議題，vehicular delay tolerant networks (VDTN)也隨之誕生。

VDTN 在近期受到越來越多研究者的關注，一般的 VDTN 環境包含互相不在通訊範圍內的路側單元，且這些路側單元必須要靠車輛協助資訊的傳輸，在這樣的環境下，如何讓封包在最短的時間內送到目的地已經有研究者提出了他們的看法，然而另外有研究者指出，由於在 VDTN 的環境中，封包具備較長的存活時間，讓封包在最短的時間內送到目的地並不具有更大的效益，這類型的封包由於其存活時間較長的特性，幾乎都能夠成功的送達目的地，也正因為如此，在滿足封包能夠送達的情況下，去減少傳輸封包時的能量消耗開始被研究者所重視，為了要減少能量的消耗，我們必須要先能夠量測傳輸封包時的能量消耗，在現實的環境中，能量的耗損是很容易透過儀器測量而得，然而現實的測量需要耗費的資源成本相對較大，正因為如此，我們希望能夠有數學模型幫助我們計算封包傳輸的能量消耗，[15]提出一個數學的計算方式，能夠在 VDTN 的環境下，計算出路側單元與車輛之間傳送封包的能量消耗，並且透過驗證，確認與現實的測量差距十分細微。

5.1. Optimal Traffic Scheduling in Vehicular Delay Tolerant Networks

[15]希望能夠透過數學的分析，定義出路側單元與車輛之間傳輸封包的能量消耗，一般在傳輸封包的能量消耗，可以分成兩個部分，一個是路側單元與車輛之間的訊息交換，這些基本訊息會透過 *beacon message* 傳送，另一個部分則是路側單元啟動，並且傳送封包所需要的能量，[15]假設路側單元會隨著時間累積資

訊，而且每一輛來到路側單元的車輛，移動軌跡都含有連接到後端網路的路側單元，當車輛來到路側單元時，路側單元會評估是否選擇此臺車輛協助傳輸，如果選擇此臺車輛協助傳輸的話，路側單元會將含有的所有封包做一次性的傳送，[15]考量了這種情況，提出當車輛來到路側單元時，路側單元決定以此車輛協助傳輸時，傳輸封包的能量消耗，具體的公式如下：

$$y_n = wP\left(\frac{S_n}{R} + T_0\right) + BI(n = C), n \leq C$$

P 表示在傳輸封包時傳輸功率的負擔，w 表示每單位時間傳輸所需消耗的能量，單位為焦耳，S_n表示當選擇第 n 台車時，累積的封包流量總數，R 表示封包的傳輸率，T_0為封包傳輸的時間，從這裡可以看出，前半部分表示路側單元在與車輛之間傳輸封包時，所需要花費的能量，而後半部分，*BI* 表示當選擇第 *n* 台車時，傳送 *beacon message* 的能量消耗，不論 *n* 值為多少，*BI* 的值都是固定的，*C* 則是在封包的存活時間內，所能選擇的最後一台車輛，結合這兩部分，y_n就表示選擇第 *n* 輛車協助傳輸的能量消耗。

假設封包都能夠被傳送出去，前半部分的能量消耗其實總和是一個定值，例如，封包的總量是 1600 位元組，則不管選擇多少臺車輛協助傳輸，總共需要維持 1600 位元組傳輸的能量是固定的，從這裡可以看出，能量的消耗取決於挑選車輛協助傳送的次數，所以為了要減少傳輸時的能量消耗，如何選擇適當的時間點開始找尋車輛協助傳輸是十分重要的。

5.2. 系統模組

路側單元與車輛傳輸的車載網路環境如圖 5.1 所示，在地圖中設置有路側單元，且這些路側單元互相都不在彼此的通訊範圍內，然而只存在幾個路側單元有連結到後端網路，每一個路側單元都會定期的蒐集鄰近的道路資訊，像是車流量、路段平均速度等，這些蒐集來的資料會儲存在路側單元內，透過經過路側單元的車輛協助傳輸給有連結到後端網路的路側單元，假設路側單元可以取得在它通訊

範圍內車輛的移動路徑，透過分析決定是否選擇車輛協助傳輸封包，並且每次選定一臺車輛協助傳輸時，路側單元會將其存有的封包一次性的傳輸給該車輛。

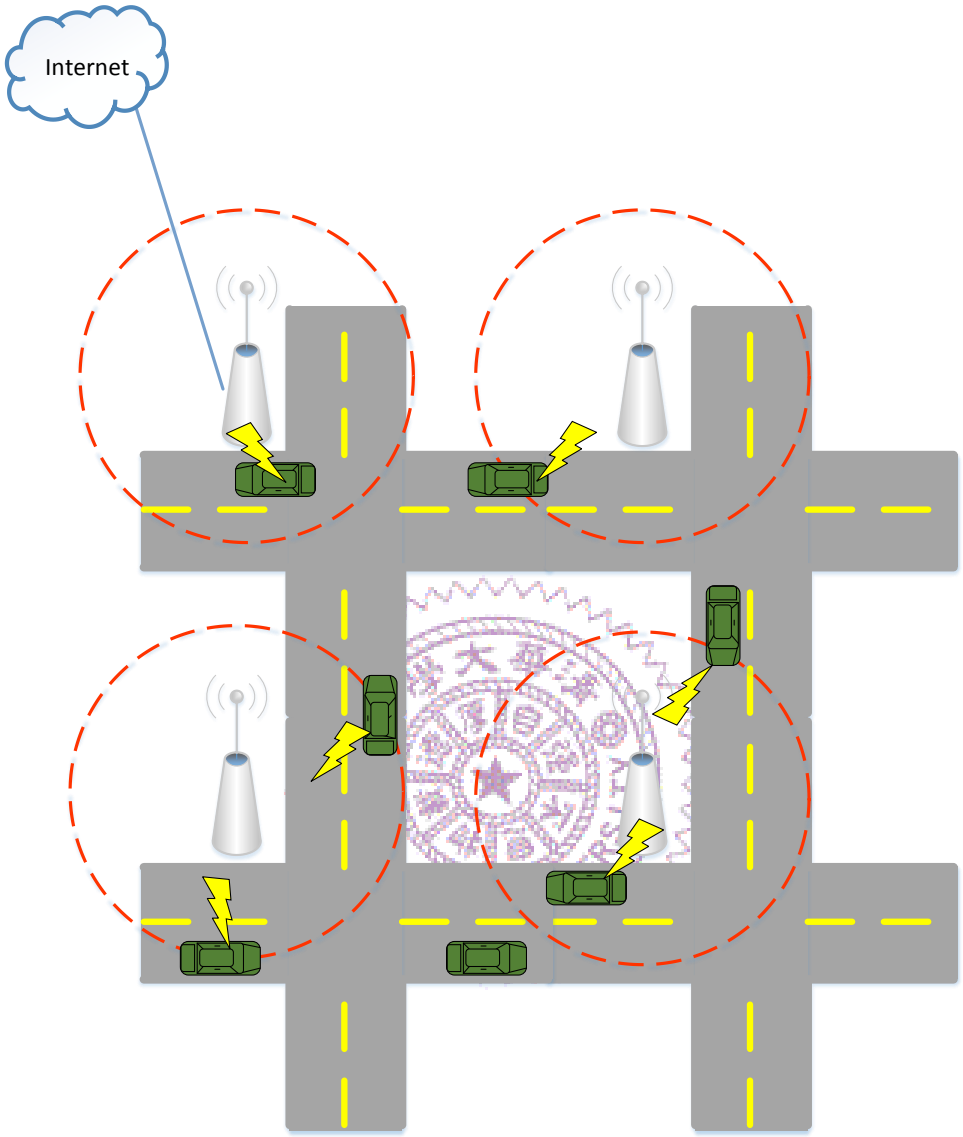


圖 5.1 車載網路環境

5.3. Probabilistic Traffic Scheduling

為了減少 *beacon message* 的能量消耗，[3]提出了兩個挑選車輛協助傳輸的方式，分別是 *Yan's traffic scheduling_1* 和 *Yan's traffic scheduling_2*，這兩個方法考量了車輛來到十字路口的時間間隔，透過預留一定倍數的時間間隔，使得封包能夠在時限內送達，並且減少傳輸次數。

如圖 5.2 所示，假設封包必須在剩餘時間內找到車輛協助傳輸，隨著封包在

路側單元內待得越久，經過的時間就會越長，相對的，剩餘時間也會越短，*Yan's traffic scheduling* 會將封包存活時間預先扣除到達間隔時間，當封包的剩餘時間扣除小於等於 *0* 時，路側單元會啟動並開始尋找車輛協助傳輸，假設只預留一個到達時間間隔，則表示預留一臺車輛到達的時距，可想而知，當預留的到達時間間隔越多，則封包的送達率也會越高，然而也會使得路側單元的啟動次數越多，並且相對的耗能也越高。

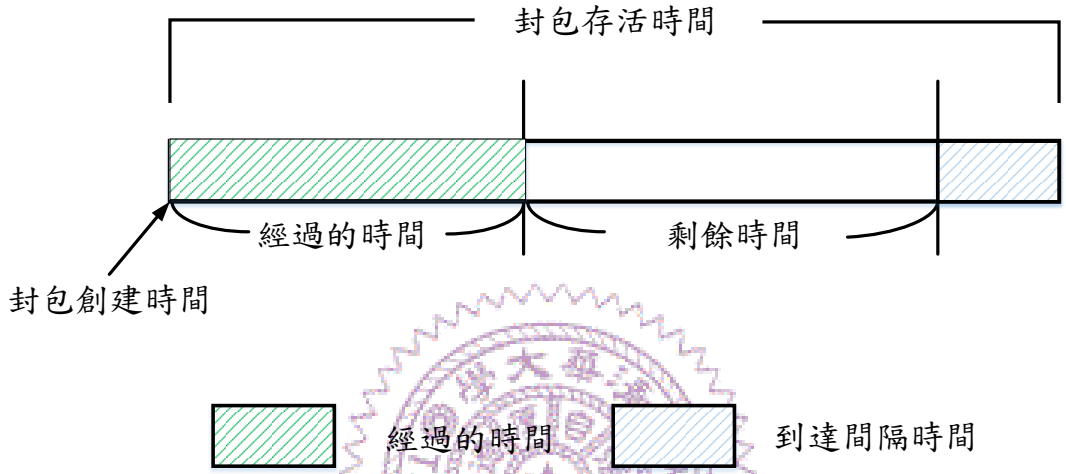


圖 5.2 封包存活時間在 YAN’S TRAFFIC SCHEDULING 示意圖

Yan's traffic scheduling_1 和 *Yan's traffic scheduling_2* 分別是預留一個到達時間間隔與兩個到達時間間隔，[15]在研究中發現，預留兩個以上的到達時間間隔，並不會有顯著的提升封包的送達率，反而會使得耗能大幅度的上升，在之後的模擬實驗中，我們將與這兩個方法做比較。

[15]所提出的兩個方法，僅僅只考慮了車輛的到達時間間隔，然而卻忽略了車輛的到達時間間隔並不是固定的，並且也忽略了當車輛收到封包後，移動到目的地所需要的行車時間，*Yan's traffic scheduling* 單純的從車輛的到達時間間隔作為判斷的依據，其實與實際的道路情況有非常的大的差距。考量了這點，我們提出了 *probabilistic traffic scheduling* (PTS)希望能夠透過機率的分析，尋找到較為精確的路側單元啟動時間，並且在兼顧封包送達率的前提下，有效的減少傳輸時

的能量消耗。考慮車輛的到達時間間隔與變異，我們採用 *chernoff inequality* 去較為全面的分析路側單元應該啟動的時間，*chernoff inequality* 的公式如下：

$$P = \Pr(X \geq a) = \Pr(e^{tX} \geq e^{ta}) \leq \frac{E[e^{tX}]}{e^{ta}}$$

Chernoff inequality 可以預測事件發生的時間點會大於某個時間點的機率，對應到 PTS 當中，我們可以預測下一輛車來的時間點，會大於封包剩餘時間的機率，也就是封包的遺失機率，當遺失的機率大於某個數值時，路側單元便會啟動，開始尋找車輛協助傳輸。

當車輛收到封包後，行駛到目的地的行車時間，由於當前路側單元所在的位置與能與後端網路連接的路側單元所在的位置，是能夠預先得知的，假定車輛的行駛速度不變，車輛的行車時間是可以被預測的，如圖 5.3 所示，首先我們會先將封包的存活時間預先扣除一段行車時間，接著透過 *chernoff inequality* 分析在剩餘的時間內是否要啟動路側單元，尋找車輛協助傳輸，具體的運算方式如下。

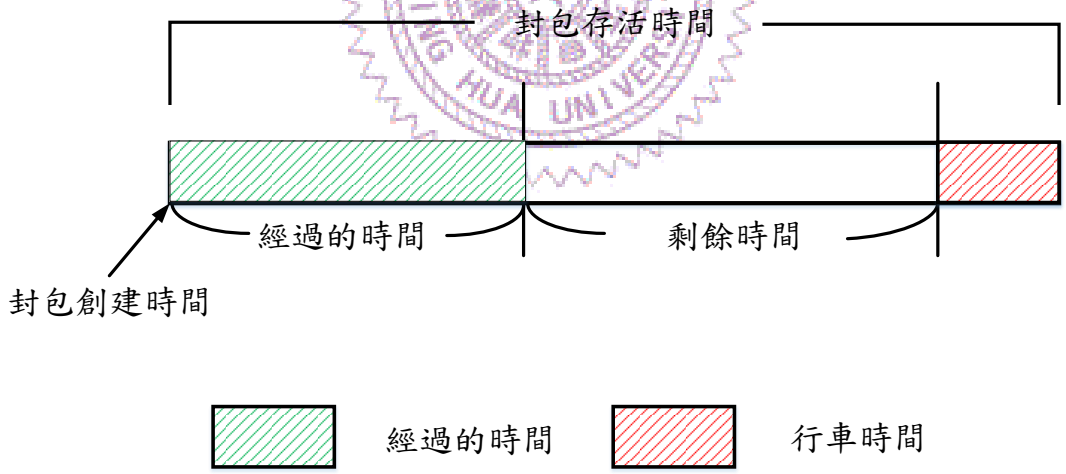


圖 5.3 封包存活時間在 PTS 的示意圖

假設車輛的到達會符合伽瑪分布，Pr(X ≥ a)中 X 代表車輛的到達時間間隔，*a* 則表示封包的剩餘時間，更進一步說，Pr(X ≥ a)中就是從現在開始計算，下一臺車輛抵達的時間，會超過剩餘時間的機率，我們將之定為 *P*，*P* 表示的就是封

包丟失的機率，要計算出 P 的值，我們必須要考慮動差生成函數，找出能夠使得 $E[e^{tX}]/e^{ta}$ 會有最小值所對應的 t 值，伽瑪分布的動差生成函數定義為

$$(1-\theta t)^{-k}, t < \frac{1}{\theta},$$

整合 $E[e^{tX}]/e^{ta}$，我們可以得到

$$e^{\ln(1-\theta t)^{-k}}/e^{ta},$$

其中 k 表示伽瑪分布的形狀，θ 則為車輛的到達時間間隔，為了算出上述式子的最小值，我們將 t 細分成 100 萬等分，並將每一個 t 值帶入式子中，計算出所有的值後，取出最小的一個，這個值就是 P。

有了封包在當前時間的丟失機率，另一方面來看，我們也就得到了當前時間路側單元如果啟動，尋找車輛協助傳輸的封包送達機率，也就是 $1-P$，如何選擇一個適當的值來決定是否傳輸，是十分重要的事情，從圖 5.6 中可以看出，不同的界值對於封包送達率的影響，越高的 $1-P$ 值自然會有越高的封包送達率，然而相對的 *beacon message* 的數量也會上升，因為這個現象，考慮了送達率以及 *beacon message* 的數量，在我們的 PTS 當中選定 0.7 作為界值，以此來決定是否要開始發送 *beacon message*。

1－P 的臨界值	車輛總數	beacon message 數量	送達率
0.9	2243	2063	89%
0.8	2243	1873	85%
0.7	2243	1467	82%
0.6	2243	1325	73%
0.5	2243	1263	67%

圖 5.4 不同界值對應的實驗結果

6. 模擬實驗結果

我們的模擬是實作在 *ns-3* 上，透過 VanetURSim 產生車載網路環境，再結合 *ns-3* 的封包模組，以此來測試 PTS 的成果。圖 6.1 為我們透過 VanetURSim 產生的車載網路環境的程式。而這個程式所產生的車載網路環境的參數如圖 6.2 所示。

```
#include  "ns3/VANETURSIM.h"

int main(void){

int Run_Time = 90000;
int Vehicle_Period = 120;
int Traffic_Light_Period = 30;
int Map_Height = 15;
int Map_Width = 15;
int Road_Length = 225;
int Vehicle_Speed = 15;
int Vehicle_Length = 3;
int Initial_Vehicle = 100;
int Transmission_Range = 150;
int Time = 0;

C_INFRASTRUCTURE MAP[Map_Height][Map_Width];
Initial_Map(Map);

while(Time <= Run_Time){
        Is_Create_Vehicle(Time, Vehicle_Period);
        Is_Change_Traffic_Light(Time, Traffic_Light_Period);
        Refresh_Map(Map);

        //封包更新
        }
}
```

圖 6.1 透過 VanetURSim 建立車載網路模擬環境

模擬執行時間	90000 秒
車輛產生間隔	2 分鐘
紅綠燈號誌變換間隔	30 秒
地圖大小	3150 公尺 x 3150 公尺
道路長度	225 公尺
車輛移動速度	15 公尺/秒
車輛長度	3 公尺
地圖初始車輛數量	100 輛
通訊範圍	150 公尺
車輛在每一時刻總數	90～120 輛

圖 6.2 車載網路模擬環境參數

我們希望透過 PTS，能夠使得封包維持一定的送達率，並且能夠有效的減少傳輸時能量的消耗，我們將 PTS 與 *Yan's traffic scheduling* 的方法做比較，*Yan's traffic scheduling_1* 為保留一臺車輛的到達時間間隔，*Yan's traffic scheduling_2* 為保留兩臺車輛的到達時間間隔。

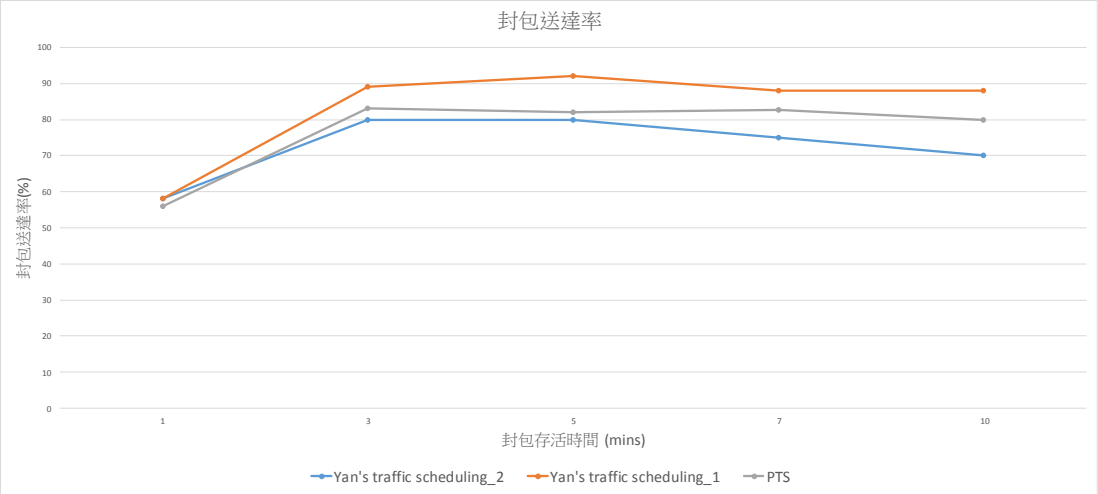


圖 6.3 封包送達率

圖 6.3 表示不同的封包存活時間，三種方法的封包送達率。從圖 6.3 中可以發現，PTS 的封包送達率會界於 *Yan's traffic scheduling_1* 與 *Yan's traffic scheduling_2* 之間，並且封包的送達率能夠維持在 80%的水準，比起 *Yan's traffic scheduling_1*，PTS 能夠有效的提升封包的送達率，然而比起 PTS，雖然 *Yan's traffic scheduling_2* 能夠有很高的封包送達率，但是其所花費的代價是能量的消耗幾乎沒有減少，在後面的圖中會進一步的說明，至於當封包存活時間為 1 分鐘時，不論哪一種方法，封包送達率都十分的低，造成這種現象的原因是由於車輛的到達時間間隔為 2 分鐘，當封包的存活時間比車輛的到達時間間隔來得小的時候，可以預見的是封包被產生後，在封包的存活時間內，有很大的可能會沒有任何車輛經過，所以封包將會傳送失敗，以至於產生的封包送達率不盡理想。

圖 6.4 表示不同的封包存活時間，路側單元啟動並且與路側單元交換 *beacon message* 的車輛總數，從圖 6.4 中可以發現，PTS 傳送 *beacon message* 的車輛總數隨著封包存活時間越長，與 *Yan's traffic scheduling_1* 的車輛總數會越接近，而

與 *Yan's traffic scheduling_2* 則一直有接近 *1/4* 的車輛總數差距，從這張圖中可以發現，PTS 能夠有效的減少 beacon message 的傳送數量。至於當封包存活時間為 1 分鐘的時候，由於車輛到達時間小於封包存活時間，基本上不論哪一種方法，路側單元都會一直啟動並交換 *beacon message*。

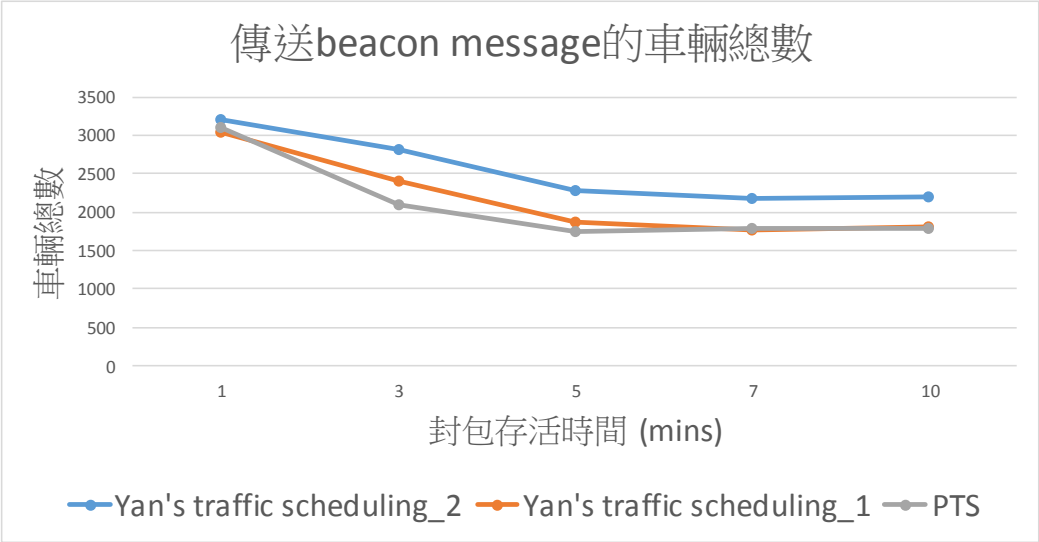


圖 6.4 傳送 *beacon message* 的車輛總數

圖 6.5 表示在封包的傳輸過程中，消耗的總能量，在圖中可以很明顯的看出，比起 *Yan's traffic scheduling*，PTS 更能夠有效的降低傳輸封包時的能量消耗，尤其是與 *Yan's traffic scheduling_2* 相比，PTS 幾乎能夠減少 *1/4* 的能量消耗。

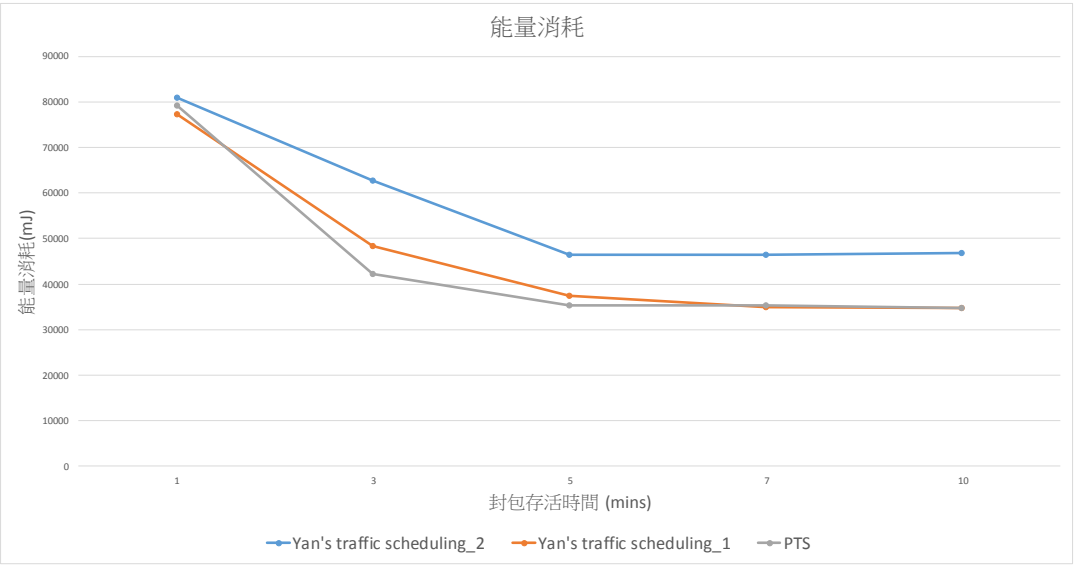


圖 6.5 傳送封包的能量消耗

7. 結論

由於 *ns-3* 是近期才開發出來的網路模擬器，在開放資源上並未有太多的素材可以使用，為了能夠在 ns-3 產生適合車載網路的模擬環境，我們首先提出了 VanetURSim，VanetURSim 是實作在 *ns-3* 上的模組，可以讓使用者在 *ns-3* 上產生車載網路的模擬環境，VanetURSim 產生的模擬環境是 Manhattan grid map，並且車輛的移動模式會貼近於現實的情況，之後，我們提出了 PTS，PTS 是路側單元與車輛之間的傳輸排程方式，PTS 不僅僅能驗證 VanetURSim 的可行性，並且能夠有效的減少封包傳輸的能量消耗。

8. 參考文獻

[1] Suriyapaibonwattana, K.; Pomavalai, C., "An Effective Safety Alert Broadcast Algorithm for VANET," *Communications and Information Technologies, 2008. ISCIT 2008. International Symposium on* , vol., no., pp.247,250, 21-23 Oct. 2008

[2] Suriyapaiboonwattana, K.; Pornavalai, C.; Chakraborty, G., "An adaptive alert message dissemination protocol for VANET to improve road safety," *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on* , vol., no., pp.1639,1644, 20-24 Aug. 2009

[3] Buchenscheit, A.; Schaub, F.; Kargl, F.; Weber, M., "A VANET-based emergency vehicle warning system," *Vehicular Networking Conference (VNC), 2009 IEEE* , vol., no., pp.1,8, 28-30 Oct. 2009
doi: 10.1109/VNC.2009.5416384

[4] Barba, C.T.; Mateos, M.A.; Soto, P.R.; Mezher, A.M.; Igartua, M.A., "Smart city for VANETs using warning messages, traffic statistics and intelligent traffic lights," *Intelligent Vehicles Symposium (IV), 2012 IEEE* , vol., no., pp.902,907, 3-7 June 2012

[5] ns-2 simulator, http://www.isi.edu/nsnam/ns/

[6] ns-3 simulator, https://www.nsnam.org/

[7] OMNeT++ simulator, https://omnetpp.org/

[8] OPNET simulator, http://www.opnet.com.tw/

[9] Ikeda, M.; Kulla, E.; Barolli, L.; Takizawa, M.; Miho, R., "Performance Evaluation of Wireless Mobile Ad-Hoc Network via NS-3 Simulator," *Network-Based Information Systems (NBiS), 2011 14th International Conference on* , vol., no., pp.135,141, 7-9 Sept. 2011

[10] Molloy, T.; Zhenhui Yuan; Muntean, G.-M., "Real time emulation of an LTE network using NS-3," *Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014). 25th IET* , vol., no., pp.251,257, 26-27 June 2013

[11] Lakkakorpi, J.; Ginzboorg, P., "ns-3 Module for routing and congestion control studies in mobile opportunistic DTNs," *Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2013 International Symposium on* , vol., no., pp.46,50, 7-10 July 2013

[12] Ikeda, M.; Hiyama, M.; Kulla, E.; Barolli, L.; Takizawa, M., "Multi-hop Wireless Networks Performance Evaluation via NS-3 Simulator," *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2011 International Conference on* , vol., no., pp.243,249, 26-28 Oct. 2011

[13] http://dl.acm.org/citation.cfm?id=2263074

[14] http://arxiv.org/abs/1004.4554

[15] Zhongjiang Yan; Zhou Zhang; Hai Jiang; Zhong Shen; Yilin Chang, "Optimal Traffic Scheduling in Vehicular Delay Tolerant Networks," *Communications Letters, IEEE* , vol.16, no.1, pp.50,53, January 2012