VanetURSim

使用手册

目錄

圖目錄

1. ns-3 安裝

在使用 VanetURSim 之前，首先必須要先安裝 ns-3，ns-3 必須要在 Linux 的環境下運行，Windows 的使用者可以透過 virtual machine 安裝 Linux 系統。

儘管 Linux 目前已經支援圖型介面模式，然而比較細微的設定上，我們仍然習慣使用傳統的文字介面，安裝好 Linux 後，我們要先啟動 command shell 或是 terminal 進入文字介面，在文字介面下安裝 ns-3 可以透過以下的步驟。

- 安裝 C++和 python。
  - sudo apt-get install gcc g++ python
  - sudo apt-get install gcc g++ python python-dev
- 安裝 mercurial。
  - sudo apt-get install mercurial
- 將 python 綁定 ns-3-dev 的組件。
  - sudo apt-get install bzr
- 下載 ns-3 (以 3.19 版本為例子)。
  - wget http://www.nsnam.org/release/ns-allinone-3.19.tar.bz2;tar xjf ns-allinone-3.19.tar.bz.2
- 進入 ns-allinone-3.19 目錄下，建立相關的 examples 和 tests 檔案。
  - cd ns-allinone-3.19
  - ./build.py --enable-examples --enable-tests
  - 看到'build' finished successfully 訊息則代表建立成功
- 進入 ns-allinone-3.19/ns-3.19 目錄下，執行 waf 的建構。
  - cd ns-3.19
  - ./waf clean;./waf –d optimized --enable-examples --enable-tests configure
  - ./waf clean;./waf –d debug --enable-examples --enable-tests configure

- ./waf
- ./waf configure –d debug --enable-sudo --enable-examples --enable-tests
- ./waf --help

- 在 ns-allinone-3.19/ns-3.19 目錄下，測試 ns-3 的相關核心檔案是否正確。
  - ./test.py –c core
- 在 ns-allinone-3.19/ns-3.19 目錄下，執行範例程式 hello-simulator，如果安裝成功，會出現'Hello Simulator'訊息。
  - ./waf --run hello-simulator

在此必須先說明，VanetURSim 是實作在 3.19 的 ns-3 版本底下，儘管在 2015 年五月 ns-3 已經更新到 3.23 版本，然而具體的 script 檔案的操作模式並未改變，在理論上應該是能夠相容的。

2. 新增 VanetURSim 模組 in ns-3

想要在 ns-3 上新增 VanetURSim 的模組，可以透過以下步驟：

- 進入 ns-3.19/src 目錄下，透過 ns-3 定義的 py 檔案，將自己的模組寫入 script 當中。
  - ./create-module.py VanetURSim
  - 如圖 2.1，完成後可以在 src 目錄下發現名稱為 vanetursim 的資料夾

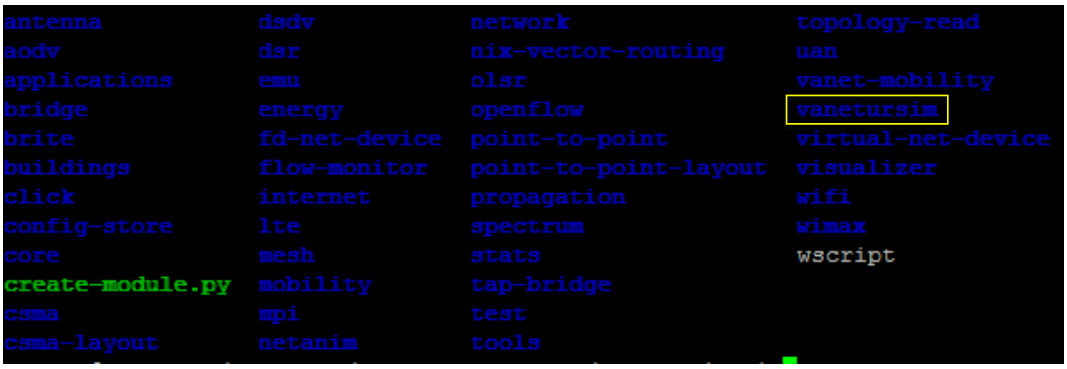

圖 2.1 src 目錄

- 進入 VanetURSim 的資料夾中，將寫好的 C++檔案以.h 和.cc 的副檔名放入 VanetURSim/model 當中。
- 在 VanetURSim 目錄下可以看到 wscript 檔案，初始的 wscript 檔案如圖 2.2，編輯 wscript 檔案，將.cc 檔案按照格式寫入 source include 當中，.h 則寫入 header include 當中，再將 test 與 examples 都註解掉。如圖 2.3 所示。

```
# -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

# def options(opt):
#     pass

# def configure(conf):
#     conf.check_nonfatal(header_name='stdint.h', define_name='HAVE_STDINT_H')

def build(bld):
    module = bld.create_ns3_module('vanetursim', ['core'])
    module.source = [
        'model/vanetursim.cc',
        'helper/vanetursim-helper.cc',
        ]

    module_test = bld.create_ns3_module_test_library('vanetursim')
    module_test.source = [
        'test/vanetursim-test-suite.cc',
        ]

    headers = bld(features='ns3header')
    headers.module = 'vanetursim'
    headers.source = [
        'model/vanetursim.h',
        'helper/vanetursim-helper.h',
        ]

    if bld.env.ENABLE_EXAMPLES:
        bld.recurse('examples')

    # bld.ns3_python_bindings()
```

source include

test

header include

examples

圖 2.2 wscript 原始檔案

```
# -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-

# def options(opt):
#     pass

# def configure(conf):
#     conf.check_nonfatal(header_name='stdint.h', define_name='HAVE_STDINT_H')

def build(bld):
    module = bld.create_ns3_module('vanetursim', ['core'])
    module.source = [
        'model/Link_List.cc',
        'model/Generate_Path.cc',
        'model/Mobility.cc',
        'model/Car.cc',
        'model/Create_Map.cc',
        'model/Infrastrcture.cc',
        ]

#    module_test = bld.create_ns3_module_test_library('vanetursim')
#    module_test.source = [
#        'test/vanetursim-test-suite.cc',
#        ]

    headers = bld(features='ns3header')
    headers.module = 'vanetursim'
    headers.source = [
        'model/Link_List.h',
        'model/Generate_Path.h',
        'model/Mobility.h',
        'model/Car.h',
        'model/Create_Map.h',
        'model/Infrastrucutre.h',
        ]

#    if bld.env.ENABLE_EXAMPLES:
#        bld.recurse('examples')

    # bld.ns3_python_bindings()
```

圖 2.3 wscript 檔案編寫

- 新增完 VanetURSim 模組後，回到 ns-3.19 目錄下，編譯 VanetURSim。
  - ./waf configure
  - ./waf
- 在 ns-3.19/scratch 目錄下，編寫自己的模擬程式，在最上面加入 VanetURSim 模組，則可以使用 VanetURSim 的功能。如圖 2.4 所示。
  - #include “ns3/vaneturSim-module.h”

```
#include <ctime>
#include "ns3/vaneturSim-module.h"
```

圖 2.4 在模擬中加入 VanetURSim

3. 產生模擬環境

在 VanetURSim 當中，我們提供了可以變動的參數包含：

- 起始的車子數量 (Ini_Car)、模擬的總時間 (T)、地圖長度 (width)、地圖寬度(height)、車輛長度 (Car_Length)、車輛速度 (Speed)、道路長度 (Road_Length)、紅綠燈轉換時間 (Traffic_Light_Period)、車輛產生間隔 (Car_Period)。
- 模擬的總時間單位為秒、地圖的長、寬表示十字路口的數量，假設長寬皆為 10，則地圖中會有 100 個十字路口，車輛長度的單位為公尺，車輛速度為公尺/秒，道路長度的單位為公尺，紅綠燈轉換時間的單位為秒，車輛產生間隔的單位為秒。

可以按照以下的步驟產生想要的模擬環境：

- 設定參數和車輛倒數計時器和紅綠燈的倒數計時器
- 產生地圖，並將地圖初始化
- 給予每個十字路口紅綠燈號誌，產生初始的車輛，
- 開始執行車輛的行為模擬，在移動模擬的部分，會先檢查紅綠燈是否需要轉換號誌，之後再檢查是否要產生車輛，接著將車輛的旗標做設定，旗標的設定是將車輛全部設定成尚未移動的狀態，再來才去更新整張地圖。
- 在 ns-3 上使用 VanetURSim 產生車載網路環境，具體的程式碼如圖 3.1 所示。

圖 3.1 利用 VanetURSim 產生車載模擬環境

4. VanetURSim 加入簡易的封包傳輸功能

另外如果沒有要做複雜的網路傳輸環境，我們提供了簡單的封包傳輸方式包含在 VanetURSim 當中，具體加入的變數有初始封包數量 (Ini_Packet)、封包大小 (bytes)、 封包產生間隔 (Packet_Period)、封包總數 (Total)，這些封包會隨機產生在除了邊界以外的路側單元(RSU)上，並且封包的目的地會是鄰近的四個十字路口中的一個，以下將會簡單的介紹我們添加入的功能：

- **Create_Packet**(*地圖初始位置，道路長度，封包存活時間，封包大小，地圖高度，地圖寬度，車輛速度*)：在路側單元產生給定封包大小的封包，如果想要修改封包的產生方式和目的地，可以在這當中做修改。
- **Refresh_Car_Packet**(*地圖初始位置，地圖高度，地圖寬度，道路長度*)：檢查車輛的封包是否有要傳給路側單元，如果有要傳送給路側單元，則檢查封包是否在時限內送達，有的話將封包的 Success 設為 1。
- **Refresh_Infrastrcuture_Packet**(*地圖初始位置，地圖高度，地圖寬度，道路長度*)：檢查車輛是否在通訊範圍內，如果有則執行 Routing Scheme，決定是否傳送封包給車輛。
- **Set_Car_Packet**(*地圖初始位置，地圖高度，地圖寬度，道路長度，成功送達的封包數目*)：更新車輛內封包的經過時間，如果經過的時間超過存活時間，則將封包移除，並且函式也會將 Success 為 1 的封包移除，並回傳目前成功送達的封包數量。
- **Set_Infrastructure_Packet**(*地圖初始位置，地圖高度，地圖寬度，道路長度*)：更新路側單元內封包的經過時間，並將經過的時間超過存活時間的封包移除。
- **Display_All_Packet**(*地圖初始位置，地圖高度，地圖寬度*)：可以顯示當前地圖的所有封包與封包所存放的位置和狀態，封包的顯示方式如圖 4.1 所示

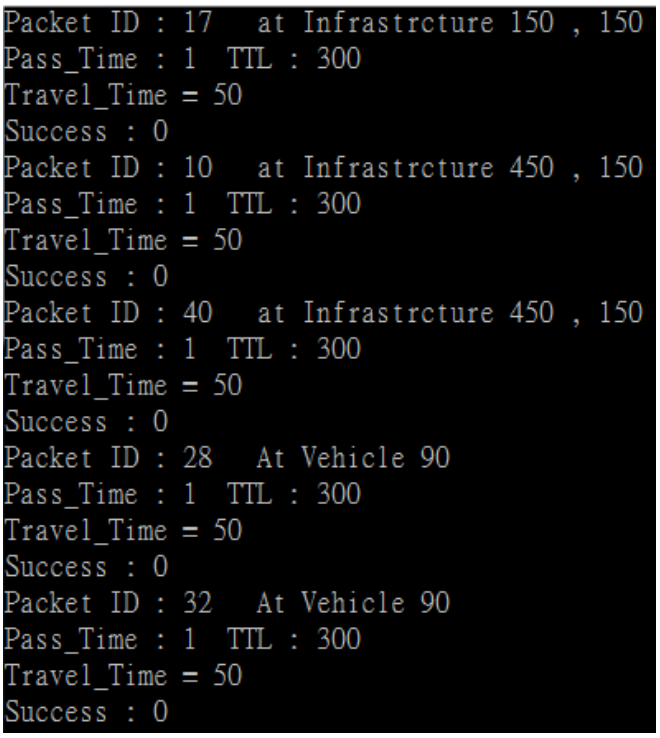
Packet ID : 17 at Infrastrcture 150 , 150
Pass_Time : 1 TTL : 300
Travel_Time = 50
Success : 0
Packet ID : 10 at Infrastrcture 450 , 150
Pass_Time : 1 TTL : 300
Travel_Time = 50
Success : 0
Packet ID : 40 at Infrastrcture 450 , 150
Pass_Time : 1 TTL : 300
Travel_Time = 50
Success : 0
Packet ID : 28 At Vehicle 90
Pass_Time : 1 TTL : 300
Travel_Time = 50
Success : 0
Packet ID : 32 At Vehicle 90
Pass_Time : 1 TTL : 300
Travel_Time = 50
Success : 0

圖 4.1 封包的顯示情況

5. 實作 PTS 模擬

Probabilistic Traffic Schedule (PTS)在論文的第五章節有詳細的介紹，主要是作為 I2V 封包傳輸的決策依據，會透過封包剩餘時間的計算決定是否何時啟動，並與車輛交換 beacon message，主要目的是希望能夠減少能量的消耗。

以下的程式碼是實作 PTS 的模擬，透過 VanetURSim 與額外附加的簡易封包模組，可以產生約 2 公里*2 公里的地圖大小，並且每隔五秒會在隨機的路側單元產生封包，並且透過車輛的協助，傳輸到目的地。

圖 5.1 為參數設定的部分，黃色框內為都市環境模擬的參數，綠色框內為封包的參數，這個程式產生的模擬，封包的時限為 5 分鐘，並且地圖中初始封包有 50 個，每個的大小為 100 bytes，封包產生的時間間隔為 5 秒，Total 為總共的封包數量，Success 紀錄有幾個封包成功傳輸。

```
#include <ctime>
#include "ns3/VanetURSim-module.h"

int main(){
        int Ini_Car = 100;
        int T = 18000;
        int width = 10;
        int height = 10;
        int Car_Length = 3;
        int speed = 15;
        int Road_Length = 225;
        int Traffic_Light_Period = 30;
        int Car_Period = 5;

        int Ini_Packet = 50;
        int TTL = 300;
        int bytes = 100;
        int Packet_Period = 5;
        int Success = 0;
        int Total = 0;

        srand( (unsigned int)time(NULL) );
        int Car_Counter = Car_Period;
        int Traffic_Light_Counter = Traffic_Light_Period;
        int Packet_Counter = Packet_Period;

        C_INFRASTRUCTURE MAP[height][width];
        Ini_Map(&MAP[0][0], height, width, Road_Length);
        Random_Traffic_Light(&MAP[0][0], height, width);
        for(i=0; i<Ini_Car; i++){
                Create_Car(&MAP[0][0], height, width, Road_Length, Speed, Car_Length);
        }
        for(i=0; i<Ini_Packet; i++){
                Create_Packet(&MAP[0][0], Road_Length, TTL, bytes, height, width, Speed);
                Total++;
        }
```

圖 5.1 PTS 參數設定

圖 5.2 為 PTS 的模擬部分，並且在最後會貼出每一個時間點的封包送達率，黃框內為車輛的移動模擬，綠框內為封包的處理，在這個部分的設計上，我們在車輛的移動之後，加入封包傳送的部分，首先檢查車輛和路側單元內的封包是否有要傳輸，再來會更新並且檢查路側單元內的封包時限，接著檢查車輛中封包的時限和封包成功傳輸的狀況。

```
for(i=0; i<T; i++){

        if(Traffic_Light_Counter == 0){
                Traffic_Liht_Change(&MAP[0][0], height, width);
                Traffic_Light_Counter = Traffic_Light_Period;
        }
        else Traffic_Light_Counter--;
        if(Car_Counter == 0){
                Create_Car(&MAP[0][0], height, width, Road_Length, Speed, Car_Length);
                Car_Counter = Car_Period;
        }
        else Car_Counter--;
        if(Packet_Counter == 0){
                Create_Packet(&MAP[0][0], Road_Length, TTL, bytes, height, width, Speed);
                Total++;
                Packet_Counter = Packet_Period;
        }
        else Packet_Counter--
        Set_Vehicle_Flag(&MAP[0][0], height, width);
        Refresh_Map(&MAP[0][0], height, width, Road_Length);
        Refresh_Car_Packet(&MAP[0][0], height, width, Road_Length);
        Refresh_Infrastructure_Packet(&MAP[0][0], height, width, Road_Length);
        Set_Infrastructure_Packet(&MAP[0][0]. height, width, Road_Length);
        Success = Set_Car_Packet(&MAP[0][0], height, width, Road_Length, Success);

        cout << "Delivery Ratio : " << (float)Success/Total << endl;

}
}
```

圖 5.2 PTS 模擬

6. 注意事項

在 ns-3 當中，如果 include 的檔案重複的話，會造成重複引入的情況，ns-3 會將這個看成是編譯錯誤，所以在撰寫.h 檔案時，我們必須要在前後加入引用的條件，假設要引用 Mobility.h 的話，在 Mobility.h 檔案的撰寫上，如圖 6.1 所示，必須要在程式碼的前後加入條件判斷。

```
#ifndef MOBILITY_H_
#define MOBILITY_H_

#include "Generate_Path.h"
typedef struct positon {
    int x;
    int y;
}Position;

int Check_Dir(T_NODE *temp, int Cur_X, int Cur_Y);
T_NODE *Trajectory_Change(T_NODE *Path, int Cur_X, int Cur_Y);
#endif
```

圖 6.1 .h 檔案的條件寫法

另外，每次只要對 VanetURSim 模組內的程式做修改，就必須重新編譯一次，才能夠使用到模擬上，在修改完模組後要進入到 ns-3.19 的目錄下，輸入以下程式

- ./waf configure
- ./waf