中華民國第四十三屆中小學科學展覽會參展作品專輯
高中組
生活與應用科學科

科別：生活與應用科學科

組別：高中組

作品名稱：電腦網路模型相變化之研究

關鍵詞：電腦網路模型、相變化

編號：040809

學校名稱：

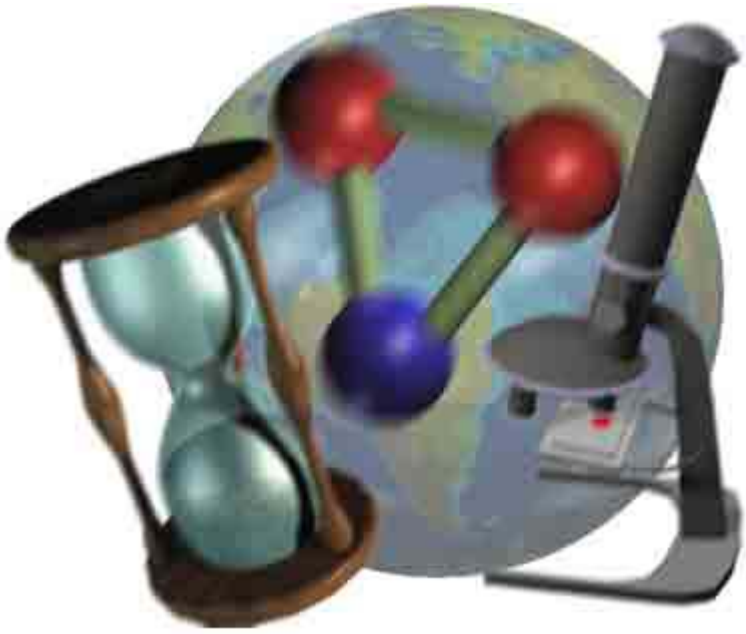
台北市立建國高級中學

作者姓名：

呂任棠

指導老師：

藍金興



摘要

近年來，電腦網路模型的研究，越來越熱門。我們設計一個簡單的網路模型，利用特殊的傳送法則，進行傳遞網路封包的工作，傳遞的過程，我們發現封包的生命期<L>與封包產生的機率 λ 兩者之間的關係具有物理的相變化的性質，也和當今網路塞車的情形頗爲類似。我們除了研究 Toru[1]所提出的模型:一個二維的晶格模型，四周爲能產生網路封包的節點，中心爲路由器。除此之外，我們也設法令模型中節點與路由器的比例爲 1:1，證明了隨著模型邊長的增加，相變化的情形就越早發生。藉由這兩個模型，我們發現網路塞車的情形、相變化發生的快慢，關鍵在於節點的個數，以及它在網路模型中的位置。

壹、研究動機

目前電腦網路廣泛的使用，也由於龐大的使用人數，網路常有「塞車」的情形，在因緣際會下看到了一篇名爲 Phase transition in a computer network traffic model 的論文，文中利用物理相變化的概念來分析網路塞車的情形，相變是大自然界隨時可見的現象，但用於解釋網路塞車的概念，我還是第一次看見，再加上我們發現文中其中的一個結論似乎有爭議，便引發我對現今電腦網路塞車真相的探索。

貳、研究目的

我們針對 Toru[1]等人所提出的模型 ，再加以作變化，進一步的分析物理網路系統相變化的概念。

參、研究設備器材

一、pentium (IV)1.6 GHz 個人電腦，WinXP 作業平台：一台

二、Borland C++ Builder 5.0 版軟體一套;

三、Matlab 5.1 版軟體一套;

四、Microcal Origin 6.0 版軟體一套

肆、研究過程及方法

一、模型介紹:

Toru 等人所提出的網路模型[1](下文以模型一代替)，是一個二維正方形陣列，見圖(1)，而在模型一的外圍(即圖型中的正方形方格)可以產生網路封包，也可以接收傳送其他的封包，稱爲節點(node)，然而中間的圓圈爲路由器，不能產生封包，但是可以接收並傳遞週遭傳來的封包。而節點產生網路封包的機率是依照亂數來產生， 產生網路封包的機率 λ (介於 0~1)來計算，例如: λ =0.5 時，四周的節點平均每兩個就有一個會產生封包，並以「隨機」的方式來選取封包的目的地，且封包的目的地只能爲節點。而我們可將每一個節點視爲四個部分(一)接收區(二)排列傳送區(三)產生區(四)紀錄區 見圖(2)

(一)接收區:接收周圍節點或路由器傳來的網路封包

(二)排列傳送區:依照封包接收的先後順序將封包依序傳出

(三)產生區:產生網路封包並即刻傳出

(四)紀錄區:紀錄已傳送過幾個封包

由於路由器不能產生網路封包故我們可視爲它有三個部分，即除了產生區外的三個部分。

二、傳送法則:

傳送法則我們先介紹由 Toru[1]所提出的兩種(一)決定性法則(二) Poisson 分佈法則，在本次科展中，我們只針對決定性法則進行探討，我們也將在討論的部分提出改變後的網路模型(下文

以模型二代替)來搭配傳送法則。

(一)決定性法則:無論是節點或是路由器傳送封包時，皆會參考和它連接的節點或是路由器，以「已傳送封包數」數目較少者來傳遞，將網路封包傳送過去。但這個法則還有另一個前提:封包只走最短路徑，即表示當有最短路徑可走時，將以此條件爲優先，封包要走最短路徑。

(二)Poisson 分佈法則:和決定性法則比較，此種方法是利用 Poisson 的函式來決定如何傳送

$$P(A)= \frac{e^{-\beta X_A}}{e^{-\beta X_A}+e^{-\beta X_B}} \quad (1)$$

$$P(B)= \frac{e^{-\beta X_B}}{e^{-\beta X_A}+e^{-\beta X_B}} \quad (2)$$

$$1= P(A)+ P(B) \quad (3)$$

β是參數， 節點或路由器標示爲 A、B，而 X_A 、 X_B 是節點或路由器已傳送封包的數量。依照 P(A) P(B)的分配機率來傳送網路封包，這和決定性的法則最大的不同就是在於無論 A，B 都有可能接收到封包。

伍、研究結果

每單位時間，節點或路由器會接收並傳送網路封包，直到封包傳送至目的地爲止。其中所花的時間，我們稱之爲封包的「生命期<L>」。即表示<L>=$\frac{\text{總步數}}{\text{成功封包數}}$我們便是以封包的生命期<L>來了解電腦網路的擁塞現象。在圖(3)我們是利用決定性的法則來表現出<L>與 λ 的關係，其中的差別在於陣列模型邊長 N 的大小，而 λ 皆從 0.01 至 0.8 以每 0.01 爲間隔遞增。每次測試時間(步數)定爲 5000。我們可以很清楚的看到由於 λ 的漸漸增加，當超越了「臨界值 λ_C 」(critical rate)時，相變現象便發生。但從圖(3)可發現 λ_C 並不相同，N=8 的 λ_C 大約在 0.33，而 N=10 爲 0.31，N=15 則大約爲 0.26， λ_C 的不同，取決於陣列模型邊長 N 的大小。N 越大 λ_C 就越小，相變現象就越早發生，即越快達到飽和的情形;就實際網路而言，即表示越多的使用者使用電腦網路，便越容易發生「網路塞車」的情形。

雖然這個網路模型很簡易，也和實際的電腦網路不盡相同，然而，我們還是擷取到了網路可

能發生的問題。

陸、討論

從實驗結果中發現當網路模型邊長越大時，越早發生相變，關於這點是值得我們來討論的，因爲當邊長增大時，不產生封包的路由器成長的速率比節點還要快，卻越容易發生相變，這個結果似乎有點讓人難以理解。原本邊長爲 3 的模型其節點和路由器的比爲 8:1，而當邊長爲 5 時則變爲 16:9，路由器成長的速度遠遠超過節點，然而卻是邊長爲 5 的模型較快達到網路塞車的情形。

由於路由器和節點的比例 R 會隨著邊長 N 而改變，因此，無法看出 R 和 N 哪一個對 λ_C 的影響較大，我們爲了排除 R 的因素，我們將網路模型進行改變如圖(4)，稱之爲模型二。如此一來，路由器和節點的比例並不受到邊長的影響，皆爲 1:1。我們就可由此模型找出 λ_C 和邊長的關係，見圖(5)。由圖(5)足可以證明，隨著邊長的增加，網路塞車的情形就越早發生，意即 λ_C 取決於邊長 N,而非路由器及節點的比例 R。

此外,我們將圖(4)及圖(5)進行比較,兩張圖形雖極爲相似,但是可發現到在相同的邊長下,模型二的 λ_C 皆較模型一的小,即網路塞車的情形較早發生。

有鑑於此結果，我們先作個假設:封包傳遞的過程主要是集中在四周的節點，而非模型中央的路由器。意即節點所傳送的封包數大於路由器所傳送的封包數。也就是封包主要是塞在節點上，而非路由器。

作這個假設原因有二，第一、在於模型二是將節點打散充滿在整個模型中，如果，封包主要是塞在節點四周，比較容易塞車就可以解釋了。第二、爲了符合我之前證明的結果: 隨著邊長的增加，網路塞車的情形就越早發生，意即 λ_C 取決於邊長 N,而非路由器及節點的比例 R。也就是說，其實塞車的主因在於節點的個數，和路由器無關。

表(1)~表(6)分別爲在不同初值下時，節點與路遊器已傳送封包數一覽表。

由表(1)~表(3)得知，就模型一而言，我們所得到的結果與假設並沒有太大的出入，值得注意的，在相變化之前，路由器已傳送封包數比節點稍大，然而在相變化發生之後，反倒是節

點略大，這足以證明，模型一之所以發生網路塞車的現象，主要的原因在於網路封包「卡死」在節點之間所造成的。

由表(4)~表(6)則爲模型二的情形，將能產生封包的節點打散充滿在整個模型中，除了比較容易造成網路塞車外，在封包傳送的過程負擔的比例也較模型一重，無論在相變前、後皆是節點已傳送封包數較大，這是和模型一不同之處，不過，在相變後，節點所佔的比例越來越大，所以，仍然證明了塞車的主因，問題出在節點上。

柒、結論

在我們修改了網路模型的形式後，終於可以確定 λ_C 與邊長N之間的關係:隨著N的增加 λ_C 便隨之減小，用最簡單的觀點，網路的使用者越多塞車的機率就變大，這是合理的。

在我們探討完已傳送封包數的問題後，也確定了節點的個數即在網路模型中的位置是造成網路塞車的主因、相變化發生的最重要因素。

儘管我們的網路模型雖然和現實世界的網路不盡相同，然而，我們在擷取到網路部分特性下，發現網路塞車和物理的相變的情形極爲類似，往後，我們可以改變模型的形狀，甚至直接模擬現實的網路系統，進而找出解決網路塞車的方法。

捌、參考資料及其他

一、 參考資料:

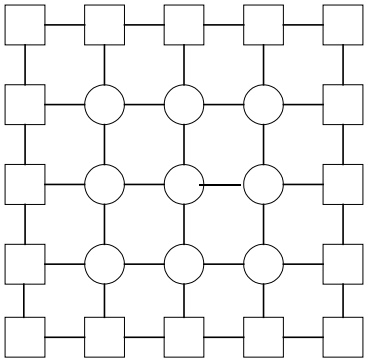
[1]Toru Ohira， Ryusuke Sawatari， Phase transition in a computer network traffic model， Phys. Rev. E 58， 193(1998)

[2]E.W. Leland， M.S Taqqu， W. Willinger and D.V. Wilson， ACM/SIGCOMM Comput. Commun. Rev. 23，183(1993);S. Yukawa and M. Kikuchi J. Phys. Soc.Jpn. 64，35 (1995);M.Takayasu， H. Takasu， and T. Sato， Physica A 233，824(1996); M. S. Taqqu， W. Willinger and R. Sherman， ACM/SIGCOMM Comput. Commun. Rev. 27， 5(1997); A. U. Tretyakov， H. Takayasu， and M. Takayasu， Physica A 233，315(1998).

[3]W. Leutzbach， Introduction to the Theory of Traffic Flow(Springer-Verlag， Berlin， 1988);O. Biham， A. A. Middleton， Hasebe， A. Nakayama， A. Shibata， and Y. Sugiyama， Phys. Rev. E 51， 1035 (1995).

[4]S. Kirkpatric， C. Gelatt， and M. Vecchi， Science 220，671 (1983).
[5]K. W. Huddart， Eighth International Conference on Road Traffic Monitoring and Control， IEE Conf. No.422(IEE， London，1996).
[6]W. Stallings，High-Speed NetWorks: TCP/IP and ATM Design Principles(Prentice-Hall，London，1998)
[7] L.Arbib ， The Handbook of Brain Theory and Neural Networks(MIT Press ， Cambridge，1995)

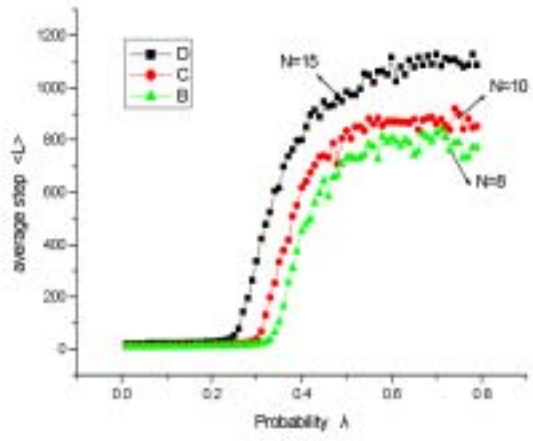
二、圖表:



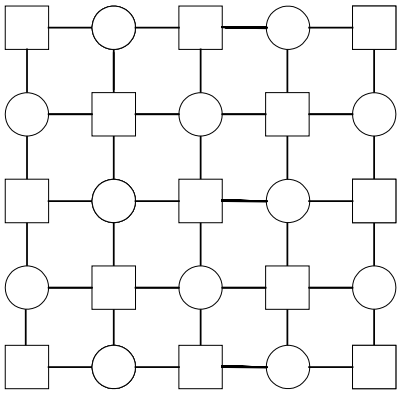
圖(1):網路模型一，四周爲節點中間爲路由器

接收區	排列傳送區
產生區	紀錄區

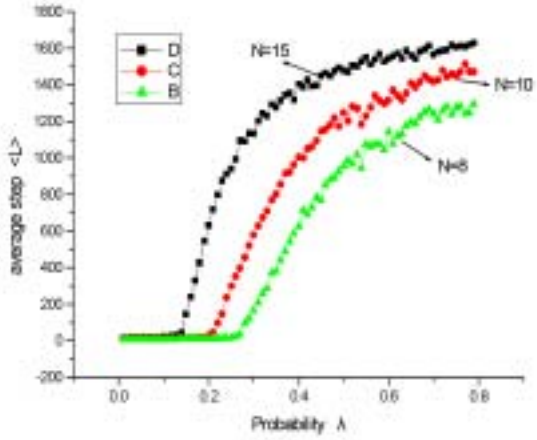
圖(2):節點的四個區域



圖(3):網路模型一在 N=8、10、15 之相變圖形



圖(4):網路模型二



圖(5):網路模型二在 N=8、10、15 之相變圖形

	相變前	相變後
節　點	2106	5038
路由器	2315	4967
比　例	0.9097	1.0143

表(1)模型一 N=8，已傳送封包數一覽表

	相變前	相變後
節　點	2081	5100
路由器	2287	4952
比　例	0.9099	1.0299

表(2)模型一 N=10，已傳送封包數一覽表

	相變前	相變後
節　點	1884	5145
路由器	2047	4875
比　例	0.9204	1.0554

表(3)模型一 N=15，已傳送封包數一覽表

	相變前	相變後
節　點	2261	6548
路由器	1924	4565
比　例	1.1752	1.4344

表(4)模型二 N=8，已傳送封包數一覽表

	相變前	相變後
節　點	2141	6615
路由器	1877	4619
比　例	1.1406	1.4321

表(5)模型二 N=10，已傳送封包數一覽表

	相變前	相變後
節　點	2011	6659
路由器	1855	4731
比　例	1.0841	1.4075

表(6)模型二 N=15，已傳送封包數一覽表

附錄: BCB 程式碼

```
//---------------------------------- Main.h ----------------------------------

#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
#include <algorithm>
#include <iostream>
#include <functional>
#include <vector>
using namespace std;
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TButton *Button1;
        TMemo *Memo1;
        TProgressBar *ProgressBar1;
        TEdit *Edit1;
        TEdit *Edit2;
        TLabel *Label1;
        TEdit *Edit3;
        TEdit *Edit4;
        TLabel *Label2;
        TLabel *Label3;
        TEdit *Edit5;
        TProgressBar *ProgressBar2;
        TButton *Button2;
        TButton *Button3;
```

```
        TSaveDialog *SaveDialog1;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);

private:    // User declarations
public:         // User declarations

vector<float> AvgStep;
vector<float> AvgStep_con;
vector<float> Node_sent;
vector<float> Router_sent;
int compare;
int Node_sent_size;
int Router_sent_size;
double Total_Num_sent(int Node_size,int Router_size,int compare);
float Avg_Node_sent;
float Avg_Router_sent;
float Avg_Node_sent_before;
float Avg_Router_sent_before;
int    mode_model;

        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

```
//--------Main.cpp-----------------------------------------------------------------

#include <vcl.h>
#include <time.h>
#include <fstream.h>
#pragma hdrstop

#include "Main.h"
#include "Control.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{


    time_t starttime, endtime;

    Controller TEST;
    TEST.InitializeSquare(5, Edit2->Text.ToInt());

    ofstream Out("d:\\data3\\test.txt", ios::out);

    double start, end, interval;
    start = Edit1->Text.ToDouble();
    end = Edit4->Text.ToDouble();
    interval = Edit3->Text.ToDouble();

    ProgressBar1->Min = 0;
    ProgressBar1->Max = Edit2->Text.ToInt();
    ProgressBar2->Min = 0;
```

```
ProgressBar2->Max = (end - start) / interval;
ProgressBar2->Position = 0;

for(double d = start; d < end; d += interval)
{
    TEST.Lambda = d;
    TEST.InitializeSquare(Edit5->Text.ToInt(), Edit2->Text.ToInt());

    ProgressBar1->Position = 0;
    starttime = time(NULL);

    for(int i = 0; i < Edit2->Text.ToInt(); i++)
    {
        TEST.OneSecond();

        ProgressBar1->Position++;
    }

    endtime = time(NULL);

    int success, sumtime;
    double averagetime;
    TEST.Statistics(&sumtime, &success, &averagetime);

    AnsiString str;
    str = sumtime;
    str = str + ", " + success + ", "+ TEST.Lambda + ", " +averagetime + ", "    +
          Edit5->Text;
    str = str + ", " + (endtime - starttime);
    AvgStep.push_back(averagetime);
    Memo1->Lines->Add(str);
    Out << str.c_str() << endl;

    ProgressBar2->Position++;
    TEST.Num_sent(Edit5->Text.ToInt());
    Node_sent.push_back(TEST.z);
    Router_sent.push_back(TEST.x);
}
```

```
    double u=
      (( Edit4->Text.ToDouble()-Edit1->Text.ToDouble())/Edit3->Text.ToDouble())+1;
    for(int i=1;i<u;i++)
    AvgStep_con.push_back(AvgStep[i]/AvgStep[i-1]);
    typedef vector<float>::iterator iterator;
    iterator it1 =    max_element(AvgStep_con.begin() , AvgStep_con.end());
    for(int i=0;i<u-2;i++)
       if(*it1==AvgStep_con[i])
       {
            compare=i;
              break;
       }
    ProgressBar1->Position = 0;
    ProgressBar2->Position = 0;
    Node_sent_size=Node_sent.size();
    Router_sent_size=Router_sent.size();
    Total_Num_sent(Node_sent_size,Router_sent_size,compare);
    AnsiString str2;
    str2=AnsiString("相變前節點平均傳送封包數:") + Avg_Node_sent_before + "  相
      變前路由器平均傳送封包數:" +
      Avg_Router_sent_before + "相變後節點平均傳送封包數:" + Avg_Node_sent + "
      相變後路由器平均傳送封包數:" +
      Avg_Router_sent + "      The Critical Lambda is:" +
   (Edit1->Text.ToDouble()+(Edit3->Text.ToDouble()*compare)) ;
    Memo1->Lines->Add(str2);

    Out.close();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Memo1->Clear();
    AvgStep.clear();
    AvgStep_con.clear();
    Node_sent.clear();
    Router_sent.clear();
   }
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
     bool Success = SaveDialog1->Execute();
     if(Success == false)
          return;

     AnsiString OutFile = SaveDialog1->FileName;
     Memo1->Lines->SaveToFile(OutFile);
}
//---------------------------------------------------------------------------
double TForm1::Total_Num_sent(int Node_size,int Router_size,int compare)
{
     int k=0;
     int l=0;
     int a=0;
     int b=0;
     for(int i=compare;i<Node_size;i++)
        k+=Node_sent[i];
     Avg_Node_sent=k/(Node_size-compare);
     for(int i=compare;i<Router_size;i++)
        l+=Router_sent[i];
     Avg_Router_sent=l/(Router_size-compare);//相變後

    for(int i=0;i<compare;i++)
        a+=Node_sent[i];
     Avg_Node_sent_before=a/compare;
     for(int i=0;i<compare;i++)
        b+=Router_sent[i];
     Avg_Router_sent_before=b/compare;//相變前

     return Avg_Node_sent;
     return Avg_Router_sent;
     return Avg_Node_sent_before;
     return Avg_Router_sent_before;
}


//------------------------------------ Control.h-------------------------------------
```

```
#ifndef ControlH
#define ControlH
#define mode_deterministic 0
#define mode_probabilistic 1

//---------------------------------------------------------------------------
#include <vector>
using namespace std;
//---------------------------------------------------------------------------
class Package
{
public:    //暫時,爲了方便 => public
     double ID;
     double To;
     double From;
     double StartTime;
     double EndTime;
     bool IfEnd;
public:
     Package();
     ~Package();
     Package operator=(const Package &_right);
     Package operator+(const Package &_right);
};
//---------------------------------------------------------------------------
class Node
{
public:    //暫時,爲了方便 => public
     bool CanGenerate;
     double Lambda;
     double Beta;
     vector<Package> Wait;
     vector<Package> Queue;
     vector<Package> EndPackages;
     double AlreadySent;
     double index;
     double borderindex;
```

```
    Node *Next[8];
public:
    Node();
    ~Node();
    void Recieve(Package _recieve, double _nowtime, double _sumtime);
    void PushWait();
    void Send(int _mode, double _nowtime, int _width, double _sumtime); //
mode_deterministic, mode_probabilistic
    void Generate(int _mode, int _sumborder, int _nowtime, int _width, double _sumtime,
int *_table);
};
//---------------------------------------------------------------------------
class Controller
{
public:   //暫時,爲了方便 => public
    int Length;
    double Lambda;
    double SumTime;
    double Beta;
    double SumBlock;
    double SumBorder;
    double NowTime;
    Node *Nodes;
    int *Table;
public:
    Controller();
    ~Controller();
    void OneSecond();
    void ManySeconds(int _seconds);
    void InitializeSquare(int _length, double _sumtime);
    void Statistics(int *_sumtime, int *_success, double *_averagetime);
    double Num_sent(int _length);
    float z;
    float x;
    };
#endif
//---------------------------------------------------------------------------
//-------------------------------------Control.cpp-------------------------------------------------
```

```
#include <vcl.h>
#include <stdlib.h>
#pragma hdrstop
#include "Control.h"
//---------------------------------------------------------------------------------------------------
Package::Package()
{
    ID = -1;
    To = -1;
    IfEnd = false;
    From = -1;
    StartTime = -1;
    EndTime = -1;
}
//---------------------------------------------------------------------------
Package::~Package()
{
}
//---------------------------------------------------------------------------
Package Package::operator=(const Package &_right)
{
    ID = _right.ID;
    From = _right.From;
    To = _right.To;
    IfEnd = _right.IfEnd;
    StartTime = _right.StartTime;
    EndTime = _right.EndTime;
    return _right;

}
Node::Node()
{
    CanGenerate = true;
    Beta = 0.01;
    AlreadySent = 0;
    index = -1;
    borderindex = -1;
    for(int i = 0; i < 8; i++)
```

```
        Next[i] = NULL;
}
//---------------------------------------------------------------------------
Node::~Node()
{
}
//---------------------------------------------------------------------------
void Node::Recieve(Package _recieve, double _nowtime, double _sumtime)
{
    if(_recieve.To == index)
    {
        _recieve.EndTime = _nowtime + 1;
        EndPackages.push_back(_recieve);
        return;
    }

    double ElapseTime = _sumtime - _nowtime;
    if(Queue.size() + Wait.size() > ElapseTime)
        return;
    Wait.push_back(_recieve);
}
//---------------------------------------------------------------------------
void Node::PushWait()
{
    for(int i = 0; i < (int)Wait.size(); i++)
        Queue.push_back(Wait[i]);
    Wait.clear();
}
//---------------------------------------------------------------------------
void Node::Send(int _mode, double _nowtime, int _width, double _sumtime) //
mode_deterministic, mode_probabilistic
{
    if(Queue.size() == 0)
        return;

    if(_mode == mode_deterministic)
    {
        Package SendingNext;
```

```
SendingNext = Queue[0];
Queue.erase(Queue.begin());
AlreadySent += 1;

double NextAlreadySent[8] = {-1, -1, -1, -1, -1, -1, -1, -1};

for(int i = 0; i < 8; i++)
    if(Next[i] != NULL)
        NextAlreadySent[i] = Next[i]->AlreadySent;

int MinIndex = 0;
bool flag = true;

while(flag == true)
{
    flag = false;
    if(Next[MinIndex] == NULL)
    {
        flag = true;
        MinIndex++;

        if(MinIndex >= 8)
        {
            Queue.insert(Queue.begin(), SendingNext);
            AlreadySent++;
            return;
        }
    }

    if(MinIndex >= 8)
    {
        Queue.insert(Queue.begin(), SendingNext);
        AlreadySent++;
        return;
    }
}

bool Up = false;
```

```
bool Down = false;
bool Right = false;
bool Left = false;

int ToX = (int)SendingNext.To % _width;
int ToY = (SendingNext.To - ToX) / _width;
int HereX = (int)index % _width;
int HereY = (index - ToX) / _width;

if(ToX > HereX)
    Right = true;
else if(ToX < HereX)
    Left = true;

if(ToY > HereY)
    Down = true;
else if(ToY < HereY)
    Up = true;

if(Down == true && Right == true)
{
    if(Next[4] == NULL)
    {
        MinIndex = 2;
    }
    else if(Next[2] == NULL)
    {
        MinIndex = 4;
    }
    else if(NextAlreadySent[2] > NextAlreadySent[4] && NextAlreadySent[4] >=
            0)
    {
        MinIndex = 4;
    }
    else
    {
        MinIndex = 2;
    }
```

```
}
else if(Down == true && Left == true)
{
    if(Next[4] == NULL)
    {
        MinIndex = 6;
    }
    else if(Next[6] == NULL)
    {
        MinIndex = 4;
    }
    else if(NextAlreadySent[6] > NextAlreadySent[4] && NextAlreadySent[4] >=
            0)
    {
        MinIndex = 4;
    }
    else
    {
        MinIndex = 6;
    }
}
else if(Up == true && Left == true)
{
    if(Next[0] == NULL)
    {
        MinIndex = 6;
    }
    else if(Next[6] == NULL)
    {
        MinIndex = 0;
    }
    else if(NextAlreadySent[6] > NextAlreadySent[0] && NextAlreadySent[0] >=
            0)
    {
        MinIndex = 0;
    }
    else
    {
```

```
                MinIndex = 6;
            }
        }
        else if(Up == true && Right == true)
        {
            if(Next[0] == NULL)
            {
                MinIndex = 2;
            }
            else if(Next[2] == NULL)
            {
                MinIndex = 0;
            }
            else if(NextAlreadySent[2] > NextAlreadySent[0] && NextAlreadySent[0] >=
                    0)
            {
                MinIndex = 0;
            }
            else
            {
                MinIndex = 2;
            }
        }
        else if(Up == true)
            MinIndex = 0;
        else if(Down == true)
            MinIndex = 4;
        else if(Left == true)
            MinIndex = 6;
        else if(Right == true)
            MinIndex = 2;
        Next[MinIndex]->Recieve(SendingNext, _nowtime, _sumtime);
    }
    else if(_mode == mode_probabilistic)
    {
        Package SendingNext;
        SendingNext = Queue[0];
        Queue.erase(Queue.begin());
```

```
            AlreadySent += 1;

            double NextAlreadySent[8] = {-1, -1, -1, -1, -1, -1, -1, -1};
            for(int i = 0; i < 8; i++)
                if(Next[i] != NULL)
                    NextAlreadySent[i] = Next[i]->AlreadySent;

            int MaxIndex = 0;
            for(int i = 0; i < 8; i++)
                if(NextAlreadySent[i] > NextAlreadySent[MaxIndex])
                    MaxIndex = i;
            for(int i = 0; i < 8; i++)
                if(NextAlreadySent[i] == -1)
                    NextAlreadySent[i] = NextAlreadySent[MaxIndex] + 1000;
            int MinIndex = 0;
            for(int i = 0; i < 8; i++)
                if(NextAlreadySent[i] < NextAlreadySent[MinIndex])
                    MinIndex = i;

            Next[MinIndex]->Recieve(SendingNext, _nowtime, _sumtime);
        }
        else
            return;
    }
    //---------------------------------------------------------------------------
    void Node::Generate(int _mode, int _sumborder, int _nowtime, int _width, double
    _sumtime, int *_table)
    {
        if(CanGenerate == false)
            return;

        long IfGenerate = random(10000000);
        if(IfGenerate > Lambda * 10000000)
            return;

        if(_mode == mode_deterministic)
        {
            Package NewPackage;
```

```
NewPackage.ID = 1;
NewPackage.From = index;
NewPackage.StartTime = _nowtime;
NewPackage.EndTime = -1;
NewPackage.IfEnd = false;

do
    NewPackage.To = random(_sumborder);
while(NewPackage.To == borderindex);

AlreadySent += 1;

double NextAlreadySent[8] = {-1, -1, -1, -1, -1, -1, -1, -1};

for(int i = 0; i < 8; i++)
    if(Next[i] != NULL)
        NextAlreadySent[i] = Next[i]->AlreadySent;

int MinIndex = 0;
bool flag = true;

while(flag == true)
{
    flag = false;
    if(Next[MinIndex] == NULL)
    {
        flag = true;
        MinIndex++;

        if(MinIndex >= 8)
        {
            Queue.insert(Queue.begin(), NewPackage);
            AlreadySent++;
            return;
        }
    }
```

```
        if(MinIndex >= 8)
        {
            Queue.insert(Queue.begin(), NewPackage);
            AlreadySent++;
            return;
        }
    }

    bool Up = false;
    bool Down = false;
    bool Right = false;
    bool Left = false;

    int ToX = _table[(int)NewPackage.To] % _width;
    int ToY = (_table[(int)NewPackage.To] - ToX) / _width;
    int HereX = (int)index % _width;
    int HereY = (index - ToX) / _width;

      if(ToX > HereX)
        Right = true;
    else if(ToX < HereX)
        Left = true;

    if(ToY > HereY)
        Down = true;
    else if(ToY < HereY)
        Up = true;

    if(Down == true && Right == true)
    {
        if(Next[4] == NULL)
            MinIndex = 2;
        else if(Next[2] == NULL)
            MinIndex = 4;
        else if(NextAlreadySent[2] > NextAlreadySent[4] && NextAlreadySent[4] >=
0)
            MinIndex = 4;
        else
```

```
            MinIndex = 2;
        }
        else if(Down == true && Left == true)
        {
            if(Next[4] == NULL)
                MinIndex = 6;
            else if(Next[6] == NULL)
                MinIndex = 4;
            else if(NextAlreadySent[6] > NextAlreadySent[4] && NextAlreadySent[4] >=
0)
                MinIndex = 4;
            else
                MinIndex = 6;
        }
        else if(Up == true && Left == true)
        {
            if(Next[0] == NULL)
                MinIndex = 6;
            else if(Next[6] == NULL)
                MinIndex = 0;
            else if(NextAlreadySent[6] > NextAlreadySent[0] && NextAlreadySent[0] >=
0)
                MinIndex = 0;
            else
                MinIndex = 6;
        }
        else if(Up == true && Right == true)
        {
            if(Next[0] == NULL)
                MinIndex = 2;
            else if(Next[2] == NULL)
                MinIndex = 0;
            else if(NextAlreadySent[2] > NextAlreadySent[0] && NextAlreadySent[0] >=
0)
                MinIndex = 0;
            else
                MinIndex = 2;
        }
```

```
            else if(Up == true)
                MinIndex = 0;
            else if(Down == true)
                MinIndex = 4;
            else if(Left == true)
                MinIndex = 6;
            else if(Right == true)
                MinIndex = 2;

            Next[MinIndex]->Recieve(NewPackage, _nowtime, _sumtime);
            }



        else
            return;
}
//---------------------------------------------------------------------------
Controller::Controller()
{
    Beta = 0.01;
    Lambda = 0.1;
    Length = 5;
    Nodes = new Node[1];
    Table = new int[1];
}
//---------------------------------------------------------------------------
Controller::~Controller()
{
    delete[] Table;
    delete[] Nodes;
}
//---------------------------------------------------------------------------
void Controller::OneSecond()
{
    for(int i = 0; i < SumBlock; i++)
    {
        Nodes[i].Send(mode_deterministic, NowTime, Length, SumTime);
```

```
            Nodes[i].Generate(mode_deterministic, SumBorder, NowTime, Length, SumTime,
Table);
            Nodes[i].PushWait();
        }

        NowTime++;
}
//---------------------------------------------------------------------------
void Controller::ManySeconds(int _seconds)
{
        for(int i = 0; i < _seconds; i++)
            OneSecond();
}
//---------------------------------------------------------------------------
void Controller::InitializeSquare(int _length, double _sumtime)
{

        //正方形
        delete[] Nodes;
        delete[] Table;
        SumTime = _sumtime;
        Length = _length;
        SumBlock = _length * _length;

        Nodes = new Node[_length * _length];
        NowTime = 0;

        if(Length % 2 == 1)
            SumBorder = ((Length * Length) / 2) + 1;
        else
            SumBorder = ((Length * Length) / 2);

        Table = new int[SumBorder];

        //左上
        Nodes[0].Next[2] = &Nodes[1];
        Nodes[0].Next[4] = &Nodes[Length];
```

```
//左下
Nodes[(Length-1)*Length].Next[0] = &Nodes[(Length-2)*Length];
Nodes[(Length-1)*Length].Next[2] = &Nodes[(Length-1)*Length + 1];

//右上
Nodes[Length-1].Next[4] = &Nodes[2*Length - 1];
Nodes[Length-1].Next[6] = &Nodes[Length - 2];

//右下
Nodes[Length*Length - 1].Next[6] = &Nodes[Length*Length - 2];
Nodes[Length*Length - 1].Next[0] = &Nodes[Length*Length - 1 - Length];

//上
for(int i = 1; i < Length - 1; i++)
{
    Nodes[i].Next[2] = &Nodes[i + 1];
    Nodes[i].Next[4] = &Nodes[i + Length];
    Nodes[i].Next[6] = &Nodes[i - 1];
}

//下
for(int i = Length * (Length - 1) + 1; i < Length * Length - 1; i++)
{
    Nodes[i].Next[6] = &Nodes[i - 1];
    Nodes[i].Next[0] = &Nodes[i - Length];
    Nodes[i].Next[2] = &Nodes[i + 1];
}

//左
for(int i = Length; i < Length * (Length - 1); i += Length)
{
    Nodes[i].Next[0] = &Nodes[i - Length];
    Nodes[i].Next[2] = &Nodes[i + 1];
    Nodes[i].Next[4] = &Nodes[i + Length];
}

//右
for(int i = 2 * Length - 1; i < Length * Length - 1; i += Length)
```

```
{
    Nodes[i].Next[4] = &Nodes[i + Length];
    Nodes[i].Next[6] = &Nodes[i - 1];
    Nodes[i].Next[0] = &Nodes[i - Length];
}

//中間
for(int i = 1; i < Length - 1; i++)
{
    for(int j = 1; j < Length - 1; j++)
    {
        int index = i * Length + j;

        Nodes[index].Next[0] = &Nodes[index - Length];
        Nodes[index].Next[2] = &Nodes[index + 1];
        Nodes[index].Next[4] = &Nodes[index + Length];
        Nodes[index].Next[6] = &Nodes[index - 1];

    }
}

int count = 0;

for(int i = 1; i < Length; i += 2)
{
    for(int k = 0; k < Length; k += 2)
    {
        Nodes[i+k*Length].CanGenerate = false;
        Nodes[i+k*Length].borderindex = count;
        Table[count] = i + k * Length;
        count++;
    }
}
for(int j = 0; j < Length; j += 2)
{
    for(int y = 1; y < Length; y += 2)
    {
        Nodes[j+y*Length].CanGenerate = false;
```

```
                Nodes[j+y*Length].borderindex = count;
                Table[count] = j + y * Length;
                count++;
            }
        }


        //All
        for(int i = 0; i < SumBlock; i++)
        {
            Nodes[i].index = i;
            Nodes[i].Beta = Beta;
            Nodes[i].Lambda = Lambda;
            Nodes[i].AlreadySent = 0;
        }

}
//---------------------------------------------------------------------------
void Controller::Statistics(int *_sumtime, int *_success, double *_averagetime)
{
    *_sumtime = NowTime;
    *_success = 0;
    for(int i = 0; i < SumBlock; i++)
        *_success += Nodes[i].EndPackages.size();

    double SumTime = 0;
    for(int i = 0; i < SumBlock; i++)
        for(int j = 0; j < (int)Nodes[i].EndPackages.size(); j++)
            SumTime += (Nodes[i].EndPackages[j].EndTime -
Nodes[i].EndPackages[j].StartTime);

    if(*_success != 0)
        *_averagetime = SumTime / *_success;
    else
        *_averagetime = -1;
}
//---------------------------------------------------------------------------
double Controller::Num_sent(int _length)
```

```
{
    int m=0;
    int n=0;
    int count1=0;
    int count2=0;
    Length=_length;
    SumBlock=_length*_length;
    for(int i=0;i<SumBlock;i++)
    {
        if(Nodes[i].CanGenerate==true)
        {
            count1++;
            m+=Nodes[i].AlreadySent;
        }
        else
        {
           count2++;
           n+=Nodes[i].AlreadySent;
        }
    }

    z=m/count1;
    x=n/count2;
    return z;
    return x;
}
//---------------------------------------------------------------------------
#pragma package(smart_init)
```

評語

提高網路的傳輸效能，減少擁塞現象一直是我們追求的目標，以高中生能以 c 語言撰寫模擬程式，觀察網路擁塞之臨界現象，實在不容易。然而所選用網路拓樸結構並不實際，如能考量各種不同結構，甚至是實際的網路架構，則會更有價值。