ДОДАТОК А

Графічний матеріал атестаційної роботи

Харківський національний університет радіоелектроніки

Кафедра ЕОМ

Атестаційна робота

Методи маршрутизації в корпоративній комп'ютерній мережі

Виконав:
ст. гр. СПзм-19-1
Топорков Є.О.

Перевірив:
доц., к.т.н., Носик А.М.

Об'єкт дослідження та мета роботи

Метою атестаційної роботи є дослідження моделей та методів маршрутизації в корпоративних комп'ютерних мережах.

Об'єктом дослідження є методи та алгоритми розподілення інформаційних потоків.

Завдання:

- розглянути існуючі методи маршрутизації, виявити недоліки;
- проаналізувати моделі та методи маршрутизації в мережах MPLS;
- розробити/модифікувати існуючий метод маршрутизації з метою усунення недоліків;
- провести моделювання та отримати результати.

Методи і алгоритми розподілу інформаційних потоків

3

1. Промислові протоколи маршрутизації:
   - *дистанційно-векторний протокол RIP;*
   - *протокол стану зв'язків OSPF;*
   - *протокол EIGRP.*
2. Графові алгоритми пошуку оптимальних маршрутів:
   - *алгоритм Дейкстри;*
   - *алгоритм Флойда;*
   - *пошук K-найкоротших шляхів;*
3. Розрахунок маршрутів методами математичного програмування
   - *формулювання мережевих завдань в термінах зв'язків і шляхів;*
   - *формулювання мережевих завдань в термінах вузлів та зв'язків;*
   - *рішення деяких мережевих оптимізаційних задач методом математичного програмування.*

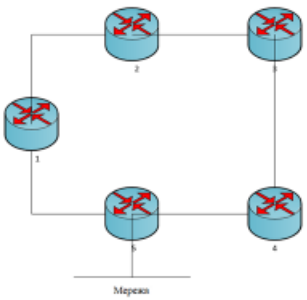
Методи і алгоритми розподілу інформаційних потоків

4

4. Методи реалізації багатоколійної маршрутизації. Технологія MPLS:
   - *протокол розповсюдження міток LDP;*
   - *завдання вибору оптимальних маршрутів;*
   - *технологія Traffic Engineering;*
   - *механізми MPLS, що реалізують Traffic Engineering.*
5. Повнооптичні мережі з комутацією каналів. Технологія GMPLS:
   - *мережа оптичної комутації блоків (OBS);*
   - *технологія DWDM;*
   - *архітектурні рішення комутаційного пристрою вузла мережі;*
   - *алгоритми встановлення каналу зв'язку;*
   - *існуючі методи розподілу потоків в мережі OBS.*

5

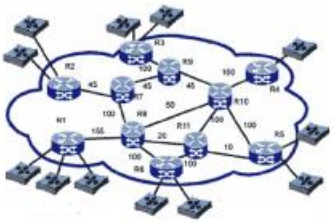
Ілюстрація роботи алгоритму DUAL

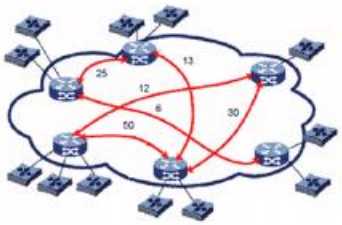


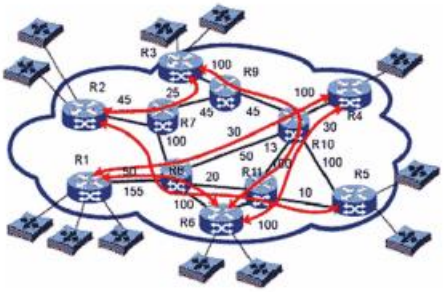
Мережа	Advertised Distance	Feasible Distance	Сусід
N	40	50	B
N	35	45	C
N	45	55	D

6

Технологія TE

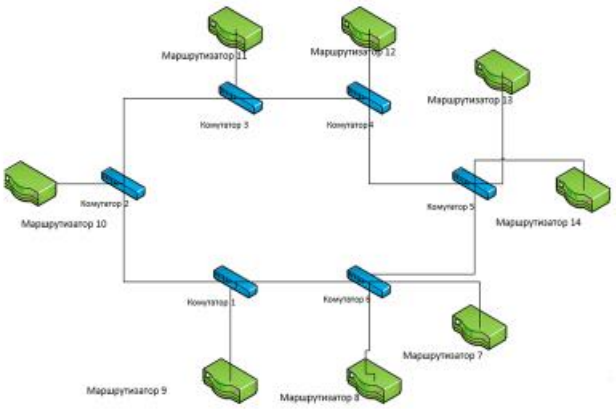






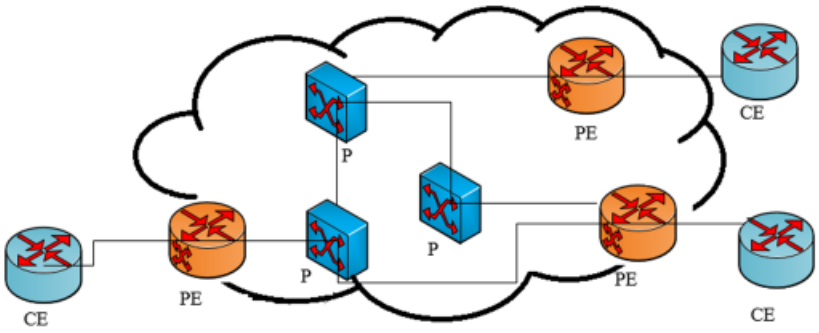
7

Схема топології досліджуваної мережі



8

Узагальнена схема мережі з багатопротокольною комутацією по мітках

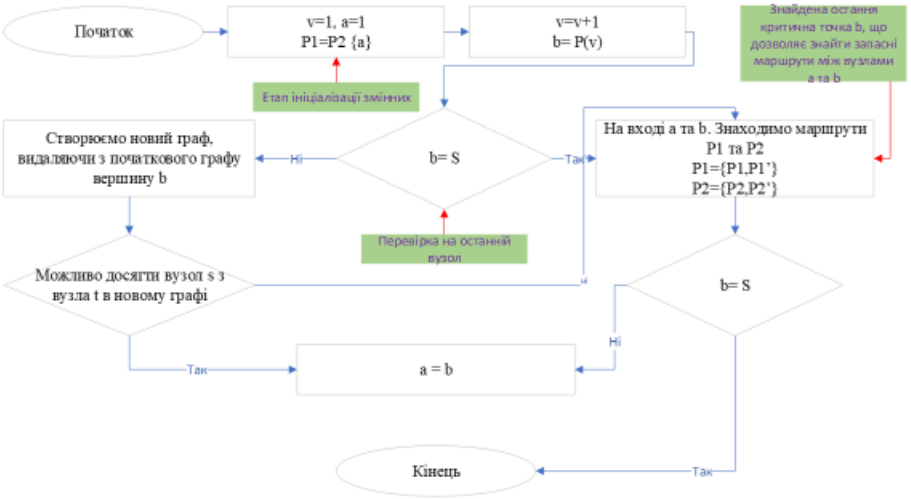


9

Алгоритм розрахунку поточного навантаження уздовж MP-BGP-сесії

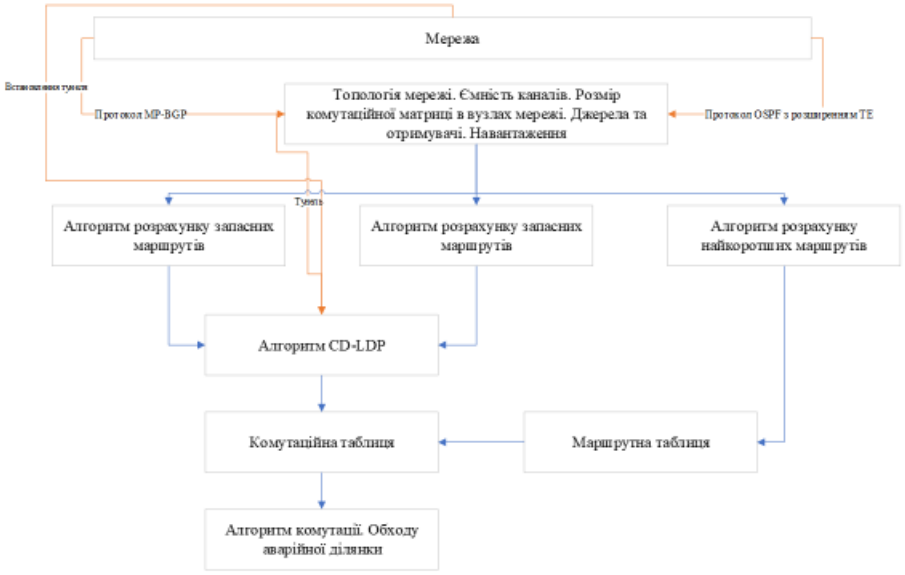
10

Модифікація алгоритму пошуку запасних маршрутів



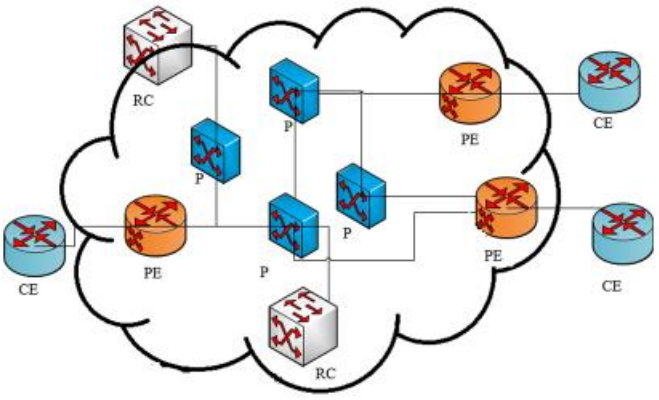
11

Функціональна схема методу динамічної маршрутизації



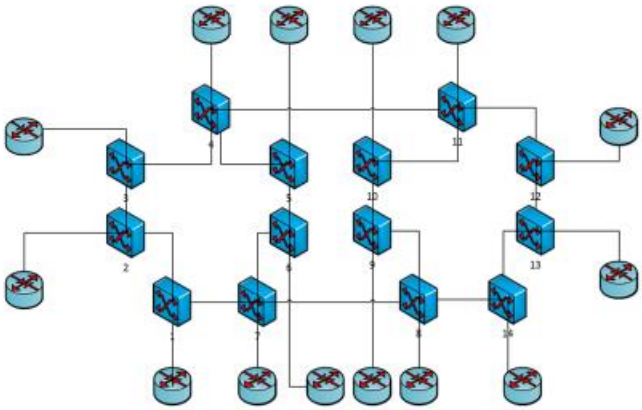
12

Узагальнена схема мережі з виділеними обчислювачами



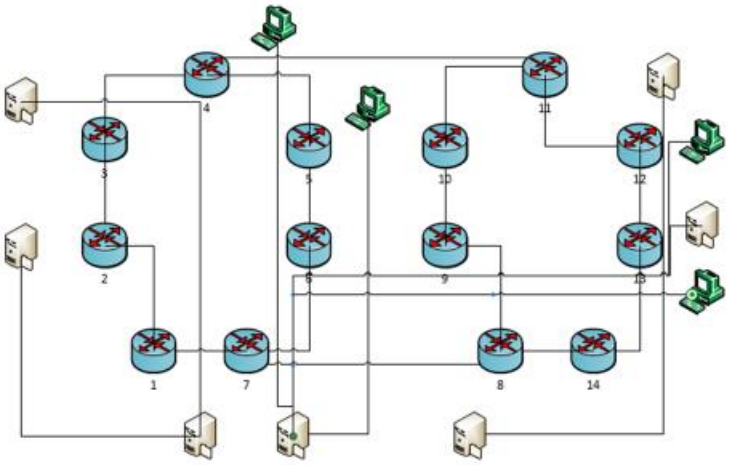
13

Топологія досліджуваної мережі



14

Напрямок руху найбільш інтенсивного трафіку



Моделювання мережі. Розрахункові навантаження і маршрути проходження потоків

15

Источник	Получатель	Нагрузка,Мбит/с	Маршрут
2	2	336	2 1 7
3	3	168	32 17
3	3	168	3 4 5 6 7
13	13	504	13 14 8 7
7	7	3	7 6 5 4
7	7	333	7 1234
7	7	335	7 6 5
7	7	169	7 8 14 13 12
7	7	166	7 6 5 4 11 12
7	7	335	7 8 14 13
12	12	501	12 11 1098

Моделювання мережі. Розрахункові навантаження і маршрути проходження потоків

16

Джерело	Отримувач	Величина навантаження	
		Відповідно до запропонованої методики, Мбіт/сек	По найкоротших маршрутах, Мбіт/сек
2	7	311	284
3	7	334	205
13	7	466	121
7	4	335	276
7	5	317	209
7	12	299	197
7	13	309	225
12	8	485	328
Загальна пропускна здатність мережі		2853	1845

17

Висновки

Був розроблений модифікований метод динамічної маршрутизації для мереж з багатопротокольною комутацією по мітках, заснованого на багатоколійній маршрутизації. Зроблено огляд сучасних технологій, що дозволяють застосувати методи оптимального розподілу трафіку в мережах передачі даних (MPLS і GMPLS з розширеннями Traffic Engineering), що дозволяє підійти до проблеми збільшення продуктивності повнооптичних мереж передачі даних. Зроблено порівняльний аналіз існуючих алгоритмів розподілу інформації. Розглянуто графові алгоритми, а також алгоритми, в основі яких лежить оптимізація мережевих потоків з використанням методів математичного програмування.

ДОДАТОК Б

Рішення завдання мінімізації втрат блоків градієнтним методом

```
% Формат отображения найденных значений
format short;
global T L XO Tc G fid Xtr
% Условия задачи
W = 32; % Количество линий в одном DWDM – канале
LineSpeed = 2048000000; % Скорость потока одной линии – 2.5 Gbit/sec
MeanBurstLength = 2048000*8; % Средняя длина блока данных 1 Мбайт
MeanHandlingTime = MeanBurstLength/LineSpeed; % среднее время
обслуживания
MaxFlow = (W * LineSpeed) / MeanBurstLength;
MaxFlow = (W * LineSpeed) / MeanBurstLength;
% P – предполагаемый трафик между узлами сети в блоках в секунду
T = [4 11 6894.85;
2 14 1415.25;
6 13 3019.38;
5 9 8670.51; ];
%T(1:Tc,3) = T(1:Tc,3) .* 0.3 ;
L = length(G); % узлов
K = sum(sum(G)); % ребер
Tc = length(T); % трафиков
Xc = K * Tc; % переменных
к = 1; % присваиваем ребрам порядковые номера
for i = 1:L
for j = 1:L
if G(i,j) == 1
G(i,j) = k;
k = k + 1;
end
end
end
% создадим матрицу Aeq*x = Beq
% правила сохранения потока
Aeq = [];
Beq = [ ] ;
for i = 1:L % по всем узлам
for k = l:Tc % по всем трафикам
Line = zeros(1, Xc) ;
for j = 1:L
if G(i,j) -= 0 % по всем ребрам данного узла
Line((G(i,j) – 1)*Tc + k) = 1;
Line((G(j,i) – l)*Tc + k) = –1;
end
end
Aeq = [Aeq; Line];
if T(k,l) == i
Beq = [Beq; T(k,3)];
elseif T(k,2) == i
Beq = [Beq; –T(k,3)];
else
Beq = [Beq; 0] ;
end
end
end
XO = zeros(1, Xc);
Xzeros = zeros(1, Xc) ;
```

```
      options =
      optimset{'Display','iter','GradObj','on','Maxlter',40,'LargeScale','off,
'TolCon
      ',0,'TolX',0);
      [XO FVAL] = fmincon(ctarget,X0, [],[] ,Aeq,Beq,Xzeros,Inf, [],options,
      MeanHandlingTime, Tc, K, W) ;
      for i = 1:L % по всем узлам
      for j = 1:L % по всем ребрам
      if G(i,j) ~= 0
      for k = l:Tc % по всем трафикам
      Lnkl = (G(i,j) - l)*Tc + k;
      Lnk2 = (G(j,i) - l)*Tc + k;
      if (XO(Lnkl) > 0) & (X0(Lnk2) > 0)
      if XO(Lnkl) > X0(Lnk2)
      XO(Lnkl) = XO(Lnkl) - X0(Lnk2);
      X0(Lnk2) = 0;
      elseif XO(Lnkl) < X0(Lnk2)
      X0(Lnk2) = X0(Lnk2) - XO(Lnkl);
      XO(Lnkl) = 0;
      end
      end
      end
      end
      end
      end
      Xtr = X0(:);
      disp(sprintf{'Целевая функция: %.4f',FVAL));
      fid = fopen('obs_nonlin.txt','w');
      fprintf(fid, 'LineSpeed: %i\n', LineSpeed);
      fprintf(fid, 'MeanBurstLength: %i\n', MeanBurstLength/8);
      fprintf(fid, 'NodeCount: %i\n', L) ;
      fprintf(fid, 'ChannelCount: %i\n', W) ;
      for i=l:L
      for j=i:L
      if G(i,j) -= 0
      fprintf(fid, 'DuplexLink: %i %i\n', i, j ) ;
      end
      end
      end
      for i=l:Tc
      fprintf(fid, 'TrafficDemand: %i %i %i\n', i, T(i,l), T(i,2));
      end
      for i=l:Tc
      disp (sprintf ('Источник: %d Получатель: %d Нагрузка: % . 2 f , T(i,l), T(i,2),
      T ( i , 3 ) ) ) ;
      get_routes(i, T(i,l), [T(i,l)]* Inf, 0 ) ;
      end
      fclose(fid);
```

ДОДАТОК В

Рішення завдання пошуку оптимальних маршрутів

```
% Формат отображения найденных значений
format short;
global T L X0 Tc G MeanHandlingTime fid
% Условия задачи
W = 32; % Количество линий в одном DWDM – канале
LineSpeed = 2048000000; % Скорость потока одной линии – 2.5 Gbit/sec
MeanBurstLength = 2048000*8; % Средняя длина блока данных 1 Мбайт
MeanHandlingTime = MeanBurstLength/LineSpeed; % среднее время
обслуживания
MaxFlow = (W * LineSpeed) / MeanBurstLength;
% P – предполагаемый трафик между узлами сети в блоках в секунду
T = [4 11;
175
2 14;
б 13;
5 9; ];
L = length(G); % узлов
K = sum(sum(G)); % ребер
Tc = length(T); % трафиков
Xc = K * Tc; % переменных
к = 1;
for i = 1:L
for j = 1:L
if G(i,j) == 1
G(i,j) = k;
к = к + 1;
end
end
end
% Сформируем матрицы A и B, такие что: A*x <= B
A = [ ] ; B = [ ] ;
for i = 1:L % по всем узлам
for j = 1:L
if G(i,j) ~= O % по всем ребрам узла
Line = zeros(1, Xc + Tc);
Line((G(i,j)-l)*Tc+l:G(i,j)*Tc) = 1;
A = [A; Line];
B = [B; MaxFlow] ;
end
end
end
% создадим матрицу Aeq*x = Beq
Aeq = [];
Beq = [];
% сумма частичек одного и того же трафика,
% входящего в узел равна сумме частичек трафика исходящего из узла.
% иначе говоря сумма(исход) – сумма(вход) = O
% если узел – источник данного трафика,
% то сумма(исход) – сумма(вход) = 1
% если получатель, то сумма(исход) – сумма(вход) = –1
for i = 1:L % по всем узлам
for к = 1:Tc % по всем трафикам
Line = zeros(1, Xc + Tc);
for j = 1:L
if G(i,j) ~ = 0 % по всем ребрам данного узла
```

```
Line((G(i,j) - 1)*Tc + k) = 1;
Line((G(j,i) - l)*Tc + k) = -1;
end
end
if T(k,l) == i
Line(Xc + k) = -1;
elseif T(k,2) == i
Line(Xc + k) = 1;
end
Aeq = [Aeq; Line];
Beq = [Beq; 0];
end
end
% Сформируем оптимизируемую функцию f
% Количество коэффициентов равно количеству искомых переменных.
% Поскольку коэффициенты зависят от нагрузки на конкретной дуге,
% то они одинаковы для частей трафика на данной дуге.
f = zeros(1, Xc + Tc);
Xzeros = zeros(Xc + Tc,1);
f(Xc+1:Xc+Tc) = -1;
XO = linprog(f,A,B,Aeq,Beq,Xzeros,Inf);
% T(l:Tc,3) = fix(XO(Xc+l:Xc+Tc));
T(l:Tc,3) = X0(Xc+1:Xc+Tc).*0.5;
% Сформируем матрицы A и B, такие что: A*x <= B
A = [ ] ; B = [ ] ;
for i = 1:L % по всем узлам
for j = 1: L
if G(i,j) ~= O %по всем ребрам узла
Line = zeros(1, Xc) ;
Line((G(i,j)-l)*Tc+l:G(i,j)*Tc) = 1;
A = [A; Line];
B = [B; MaxFlow] ;
end
end
end
% создадим матрицу Aeq*x = Beq
Aeq = [];
Beq = [] ;
% сумма частичек одного и того же трафика,
% входящего в узел равна сумме частичек трафика исходящего из узла.
% иначе говоря сумма(исход) - сумма(вход) = O
% если узел - источник данного трафика,
% то сумма(исход) - сумма(вход) = 1
% если получатель, то сумма(исход) - сумма(вход) = -1
for i = 1:L % по всем узлам
for к = 1:Tc % по всем трафикам
Line = zeros(1, Xc) ;
for j = 1:L
if G(i,j) ~ = 0 % по всем ребрам данного узла
Line((G(i,j) - 1)*Tc + k) = 1;
Line((G(j,i) - l)*Tc + k) = -1;
end
end
Aeq = [Aeq; Line];
if T(k,l) == i
Beq = [Beq; T(k,3)];
elseif T(k,2) == i
Beq = [Beq; -T(k,3)];
else
Beq = [Beq; 0] ;
end
end
end
f = ones(1, Xc);
```

```
        Xzeros = zeros(Xc,l);
        X0 = linprog(f,A,B,Aeq,Beq,Xzeros, Inf);
        177
        fid = fopen('obs_simple.txt','w');
        fprintf(fid, 'LineSpeed: %i\n', LineSpeed);
        fprintf(fid, 'MeanBurstLength: %i\n', MeanBurstLength/8);
        fprintf(fid, 'NodeCount: %i\n', L ) ;
        fprintf(fid, 'ChannelCount: %i\n', W) ;
        for i=l:L
        for j=i:L
        if G(i,j) -== 0
        fprintfffid, 'DuplexLink: %i %i\n', i, j);
        end
        end
        end
        for i=l:Tc
        fprintf(fid, 'TrafficDemand: %i %i %i\n', i, T(i,l), T(i,2));
        end
        for i=l:Tc
        disp(sprintf('Источник: %d Получатель: %d Нагрузка:
        %.2f',T(i,l),T(i,2),T(i,3)));
        get_routes(i, T(i,l), [T(i,l)], Inf, 0);
end
```

ДОДАТОК Г

Методика оптимального розподілу навантаження в мережі GMPLS

```
% нагрузка на сеть
T = [ 2 7; 3 7; 13 7; 7 4; 7 5; 7 12; 7 13; 12 8 ];
% количество искомых кратчайших маршрутов
MaxPath = 4;
% количество разных нагрузок
DemandCount = length(T) ;
% найти все маршруты между всеми парами
PathCount = 0; Paths = [];
for i = 1:DemandCount
Paths{i} = yen(G, T(i,l), T(i,2), MaxPath);
PathCount = PathCount + length(Paths{i});
end;
% количество ребер
K = sum(sum(G));
% количество узлов
NodeCount = length(G);
% емкость линий
LSpeed = 504;
% Aeq*x = Beq
A = [ ] ; B = [ ] ;
% реализация ограничения на максимальную нагрузку
for i = 1:NodeCount
for j = 1:NodeCount
if G(i,j) ~= 0
% найти все пути, проходящие через ребро i,j
Line = zeros(1, PathCount + DemandCount);
Pos = 0;
for к = 1:DemandCount
% no всем нагрузкам k
for n = l:length(Paths{k})
% no всем маршрутам n нагрузки k
for h = l:length(Paths{k}{n})-l
if ((Paths{k}{n}(h) == i) && (Paths{k}{n}(h+1) == j))
% часть маршрута Paths{k}{n} проходит через ребро
Line(Pos + n) = 1;
break;
end;
end;
end;
Pos = Pos + n;
end;
A = [A; Line];
B = [B; LSpeed*0.95];
end;
end;
end;
Aeq = []; Beq = [];
Pos = 0;
for k = 1:DemandCount
% по всем нагрузкам k
Line = zeros(1, PathCount + DemandCount);
for n = 1:length(Paths{k})
% по всем маршрутам n нагрузки k
Line(Pos + n) = 1;
end;
```

```
Pos = Pos + n;
Line(PathCount + k) = -1;
Aeq = [Aeq; Line];
Beq = [Beq; 0] ;
end;
func = zeros(1, PathCount + DemandCount);
func(PathCount + 1:PathCount + DemandCount) = -1;
Opts = optimset('Display', 'off');
LB = zeros(PathCount + DemandCount, 1);
Precision = LSpeed / 1000;
UnV = [ ] ,-
for i = 1:DemandCount
UnV(i) = PathCount + i;
end;
LoCon = 0;
while true
UpCon = LSpeed;
while true
step = (UpCon - LoCon) / 2;
if step < Precision
break;
end;
for i = 1:length(UnV)
LB(UnV(i)) = LoCon + step;
end;
[X Z ExitFlag] = linprog(func,A,B,Aeq,Beq,LB,[],[],Opts);
if ExitFlag > 0
Xbest = X;
Zbest = Z;
LoCon = LoCon + step;
else
UpCon = UpCon - step;
end;
end;
% найти минимальное значение нагрузки и зафиксировать его
for i = 1:length(UnV)
for j = 1:length(UnV)
LB(UnV(j)) = LoCon;
end;
LB(UnV(i)) = LoCon + step;
[X Z ExitFlag] = linprog(func,A,B,Aeq,Beq,LB,[],[].Opts);
if ExitFlag <= 0
break;
end;
end;
LB(UnV(i)) = Xbest(UnV(i));
UnV(i) = [];
if length(UnV) < 1
break;
end;
end;
T(1:DemandCount, 3) = Xbest(PathCount+1:PathCount+DemandCount);
global f LB A B Aeq Beq DemandCount PathCount;
UsedPath = 4;
% Aeq*x = Beq
Aeq = [] ; Beq = [] ;
% сумма потоков по найденным маршрутам равна исходному потоку
Pos = 0;
for i = 1:DemandCount
% по всем нагрузкам
A_line = zeros(1,PathCount);
for j = 1:length(Paths{i})
% по всем маршрутам данной нагрузки
A_line(Pos + j) = 1;
```

```
end;
Aeq = [Aeq; A_line];
Beq = [Beq; T(i,3)];
Pos = Pos + j;
end ;
% A*x <= B
A = [ ] ; B = [ ] ;
% сумма нагрузок на каждой дуге не должна превышать пропускную
способность
% этой дуги
for i = l:NodeCount
for j = l:NodeCount
if G(i,j) ~= 0
% найти все пути, проходящие через ребро i,j
A_line = zeros(1,PathCount);
Pos = 0;
for k = 1:DemandCount
% по всем нагрузкам k
for n = 1:length(Paths{k})
% по всем маршрутам n нагрузки k
for h = 1:length(Paths{k}{n})-l
, if ((Paths{k}{n}(h) == i) && (Paths{k}{n}(h+1) == j))
% кусок маршрута Paths{k}{n} проходит через ребро i,j
A_line(Pos + n) = 1;
break;
end;
end;
end;
Pos = Pos + n;
end;
A = [A; A_line] ;
B = [B; LSpeed];
end;
end;
end;
% необходимо, чтобы пути были кратчайшими
f = zeros(1, PathCount);
Pos = 1;
for i = 1:DemandCount
for j = 1:length(Paths{i})
f(Pos) = length(Paths{i}{j});
Pos = Pos + 1;
end;
end;
% ограничение снизу: все переменные неотрицательные
LB = zeros(PathCount, 1);
global BinVars;
% вначале необходимо сформировать набор переменных
BinVars = [];
for i = 1:DemandCount
BinVars(i).vars = prepare(UsedPath, length(Paths{i}));
[BinVars(i).count n] = size(BinVars(i).vars);
end;
global Zbest Xbest Opts;
Zbest = Inf;
Xbest = [];
Opts = optimset('Display', 'off');
[X Z ExitFlag] = linprog(f,A,B,Aeq,Beq,LB,[],[],Opts);
BBB(0, []);
if Zbest == Inf
disp('Решений нет!');
return;
end;
pos = 0;
```

```
for i=l:DemandCount
disp(sprintf('Источник: %2.d Получатель: %2.d', T(i,l), T(i,2)));
for j=1:length(Paths{i})
pos = pos + 1 ;
if Xbest(pos) > 0.000001
disp(strcatfsprintf('%3.0f:', Xbest(pos)), sprintf(' %2.0f ',
Paths{i}{j})));
end;
end;
```