实验7　路由实验

1　实验目的

1. 通过Matlab仿真，熟悉并验证基本路由协议及其转发表的生成过程；

2. 通过仿真，进一步学习验证路由网络中的最短路算法，比较不同协议和算法的性能。

2　实验内容

1. 运行提供的仿真代码，观察并验证路由表的生成和收敛过程。

2. 运行提供的仿真代码，观察链路失效后的路由表重新收敛过程，结合讲授内容，观察并思考特殊情形下算法的不足。

3. 比较不同路由协议的性能。

3　实验环境

MATLAB 是美国MathWorks公司出品的商业数学软件，用于算法开发、数据可视化、数据分析以及数值计算的高级技术计算语言和交互式环境。

本次实验需要在MATLAB中完成，任意版本均可。

4　实验原理

4.1　RIP协议与距离矢量（DV）算法

4.1.1　RIP协议与DV算法概述

距离向量（Distance Vector，DV）算法是一种迭代的、异步的和分布式的动态路由算法。在此算法中，每个节点都维护一个向量，该向量包含了到网络中所有其他节点的开销或距离的估计。具体来说，距离向量算法的工作原理如下：每个路由器都维护一张距离向量表，该表列出了当前已知的到每个目标节点的最短距离，以及到达这些目标节点的下一跳路由器。路由器通过与其邻居交换信息，不断更新其内部的距离向量表。

定义$D_i(j)$和$c(x,y)$分别为从节点i到节点j的最短距离和节点x与相邻节点y之间的距离。此时，路由节点i到所有节点的最短距离组成的向量被称

Algorithm 1: Bellman-Ford算法

对于每个节点x：
初始化(Initialization)：
for 所有目标节点y **do**
  if 节点y不是x的邻接节点 **then** $c(x,y)=\infty$;
  $D_x(y)=c(x,y)$
end
对固定时间周期执行：
Loop
for 所有邻居节点w **do**
  传送节点x的距离向量给节点w，$\mathbf{D_x}=[D_x(y):y \text{ in } N]$
  节点w根据收到的x的距离矢量更新自己的距离矢量，
  $D_w(x)=\min_v\{c(w,v)+D_v(x)\}$
end
Forever

为距离向量（Distance Vector），记作D_i。可通过Bellman-Ford算法维护距离矢量，算法描述如算法1所示。

RIP（Routing Information Protocol）协议——路由信息协议，是一种分布式的基于距离向量的路由选择协议，最大的优点是简单。本次实验中对DV算法的实现，主要结合RIP协议来实现。在RIP协议中，每个路由器不断地和相邻路由器交换路由信息，最终使得自治系统中所有节点都得到正确的路由信息，即路由收敛。RIP协议对距离的定义进行了简化，距离的定义如下：

1) 从一路由器到直接连接的网络的距离定义为1。

2) 从一路由器到非直接连接的网络的距离定义为所经过的路由器数加1。

RIP协议的距离也称为跳数，RIP协议允许一条路径最多只能包含15个路由器。因此，距离等于16时即相当于不可达。RIP协议中，每个路由器仅和相邻路由器交换信息。每个路由节点维护一个定时器，每隔30秒，节点向所有邻居发送本路由器掌握的完整的距离矢量。RIP协议中基于收到的相邻路由器的路由信息和Bellman-Ford算法维护距离矢量，并计算得到路由转发表。

在本实验中，为了便于同学们观察体会距离矢量算法，我们在RIP协议的基础上做如下改动：

1）路由器间的距离为非0值，而不是固定为1。

2）采用inf表达路由不可达的情况。

除此之外，本实验按照RIP协议定时交换路由信息、各个设备分布式独立运行的原理进行模拟，每个路由器在收到来自相邻路由器的路由信息后立即更新其路由表。

4.1.2 链路故障的情况

如图1, 在链路发生故障前，我们有$D_y(x) = D_x(y) = 4, D_y(z) = D_z(y) = 1, D_x(z) = D_z(x) = 5$。在链路改变后，$y$和$x$的距离由4变成了$\infty$，即$c(y,x)$从4变为$\infty$，此时$y$触发重新计算得到$D_y(x) = \min\{c(y,x)+D_x(x), c(y,z)+D_z(x)\} = \min\{\infty + 0, 1 + 5\} = 6$。情况似乎不像我们期望的那样，即$D_y(x)$变为51。这是因为$D_z(x)$的值是节点$z$之前通知给$y$的，而$z$通知给$y$的值，又是更早时候，$y$通知给$z$时计算的结果。不论如何$c(y,x)$发生变化，节点$z$是浑然无知的。当节点$z$收到$y$的更新时，触发节点$z$的重新计算，得到$D_z(x) = \min\{c(z,x) + D_x(x), c(z,y) + Dy(x)\} = \min\{50 + 0, 1 + 6\} = 7$。此时$z$又将更新发给$y$，$y$收到以后，得到$D_y(x)$的值为8，又将更新发给$z$；如此循环，直至 $D_y(x)$=51，$D_z(x)$=50为止。如果x到z的链路开销一开始为9999，则y到x的直连链路开销改变为10000会怎么样？权重会一直增加到10000，也就是导致了无限循环（count-to-infinity）的问题。

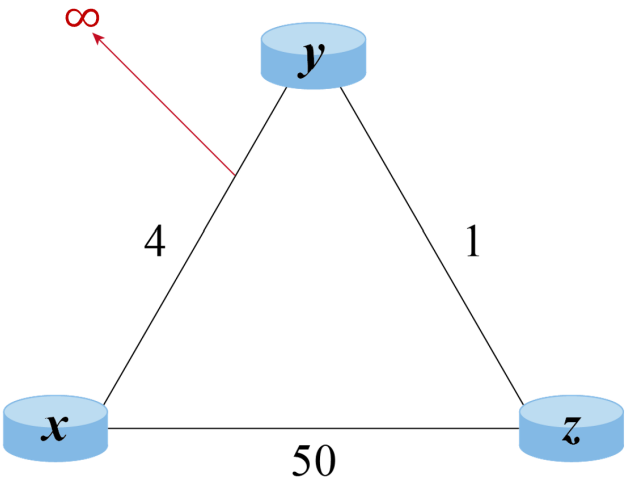

图 1: 链路故障

Algorithm 2: Dijkstra算法

初始化(Initialization)：
$N' = \{u\}$
for 所有节点 v **do**
  if v邻接于u **then** $D(v) = c(u, v)$;
  else $D(v) = \infty$;
end
执行：
while $N'! = N$ **do**
  找到不在N'中并且$D(w)$最小的节点w；
  将节点w添加到N'中；
  更新节点w的每个邻居v（v不在N'中）的$D(v)$：
  $D(v) = \min(D(v), D(w) + c(w, v))$；
end

4.2 OSPF协议与链路状态（LS）算法

链路状态（Link State，LS）算法是一种需要全局状态信息的路由算法，即需要了解网络中每条链路的开销。在实际应用中，每个节点会向网络中的所有其他节点广播链路状态包。每个路由器需要在数据库中保存一份它所收到的链路状态包的备份。完整的拓扑数据库，也称为链路状态数据库，是Dijkstra算法计算网络图中到每个路由器的最短路径的基础。基于链路状态数据库，Dijkstra算法计算当前路由节点到其他各个路由节点的最短路径。链路状态算法中最短路径算法的典型代表是Dijkstra算法，如算法2所示。

OSPF（Open Shortest Path First）是一种基于链路状态算法的路由协议。在OSPF协议中，每个路由器（节点）与其邻居之间建立联系，这种联系被称为邻接关系。每个路由器向每个邻居发送链路状态通告（Link State Advertisement, LSA）。每条链路都会生成一个LSA，用于标识链路、链路状态、路由器接口到链路的代价度量值以及链路所连接的邻居。每个邻居在收到通告后，将向其邻居转发（泛洪, flooding）这些通告。每个路由节点收到泛洪的LSA数据包后，若发现LSA是新的包，就向所有相邻的非来源的路由节点泛洪该LSA通告。否则，该路由节点将丢弃该LSA数据包。基于LSA通告数据，每个路由器维护其链路状态数据库，并根据Dijkstra算法计算当前路由节点到其他各个路由节点的最短路径。

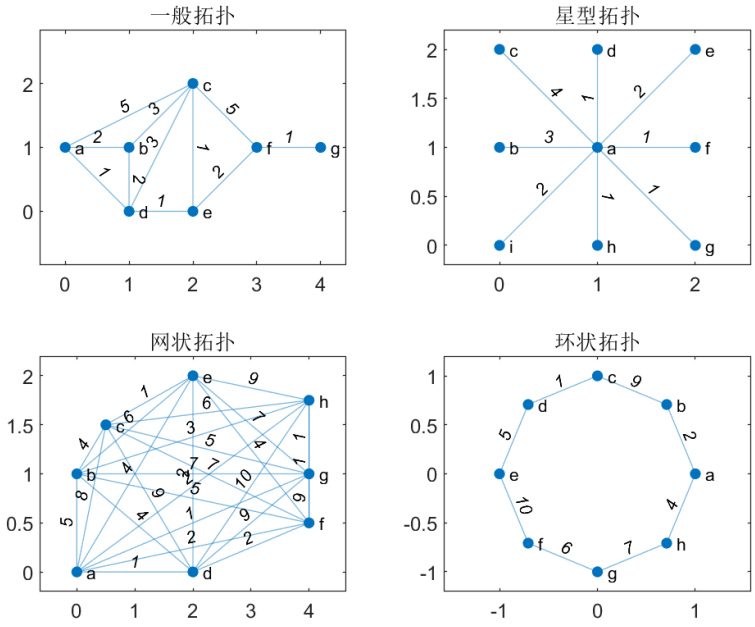

图 2: 四种网络拓扑

5 实验内容

5.1 网络拓扑

如图4所示，在本次实验中，我们提供了4种网络拓扑，即一个一般网络拓扑、一个星型网络拓扑、一个网状网络拓扑和一个环状网络拓扑。在网络拓扑图中，路由节点由小写英文字母“a,b,c,...,z”表示。为便于后续实验表示，我们对路由节点“a,b,c,...,z”分别编号为“1,2,3,....,26”。四种网络拓扑分别定义在m文件“topology_general.m”、“topology_star.m”、“topology_mesh.m”和“topology_ring.m”中。运行该网络拓扑文件，可通过Matlab画出相应网络拓扑结构图，并将相应网络拓扑结构数据，以graph数据结构形式分别保存在“topology_general.mat”、“topology_star.mat”、“topology_mesh.mat”和“topology_ring.mat”中。

调用不同网络拓扑数据时，仅将相应路由实验文件第4行代码更改为“load('topology_XXX.mat');” （“XXX”为对应拓扑结构的名称）。需要注意的是，在所有拓扑结构文件中，网络拓扑都是固定的。在提供的一般网络拓扑（“topology_general.m”），路由连边的权重（或代价）是固定的。在提供的星型网络拓扑（“topology_star.m”）、网状网络拓扑（“topology

_mesh.m”）和环状网络拓扑（“topology_ring.m”）中，路由连边的权重是随机生成的。每运行一次“topology_star.m”、“topology_mesh.m”和“topology_ring.m”可得到具有不同连边权重的路由网络。

5.2 RIP协议与DV算法仿真实验

1. 打开实验文件“exp7_1a.m”，阅读实验代码并运行实验文件，记录Matlab命令行输出的不同迭代轮次下的距离矩阵和下一跳矩阵。选择感兴趣某一次迭代，结合代码对Bellman-Ford算法的实现，对距离矩阵和下一跳矩阵的更新给出解释。

2. 打开实验文件“exp7_2.m”，阅读实验代码并运行实验文件。在本次实验中，我们在路由表收敛后，将路由节点6和7之间的链路断开（链路代价无穷大）。运行实验文件，观察matlab命令行输出的不同迭代轮次下的距离矩阵和下一跳矩阵。基于命令行的输出，观察并记录路由重新收敛的过程。结合D-V协议，分析该路由重新收敛的过程中的问题。

5.3 OSPF协议与LS算法

打开实验文件“exp7_3.m”，阅读实验代码，结合实验代码理解链路状态协议。补齐第8行实验代码，通过将变量“viewNode”设置为1到N之间的一个整数，选中某一路由节点，观察该路由节点的最小生成树随链路状态传播而变化的情况。

运行实验文件，观察Matlab命令行输出和生成的图片。观察并记录，链路状态协议下链路状态的泛洪过程。记录生成的图片中，“viewNode”路由节点的最小生成树随链路状态传播而变化的情况，结合Dijkstra算法，给出解释。

5.4 路由协议性能对比分析（选做实验）

更改“exp7_1a.m”文件和“exp7_3.m”文件中第四行代码，更换不同的路由拓扑文件。记录不同路由拓扑结构下，路由表收敛后Matlab命令行输出的“路由总传输次数”和“总传输包大小”。结合路由协议理论内容，思考合适的评价指标（如按节点数平均的总传输次数），给出你的阐释。

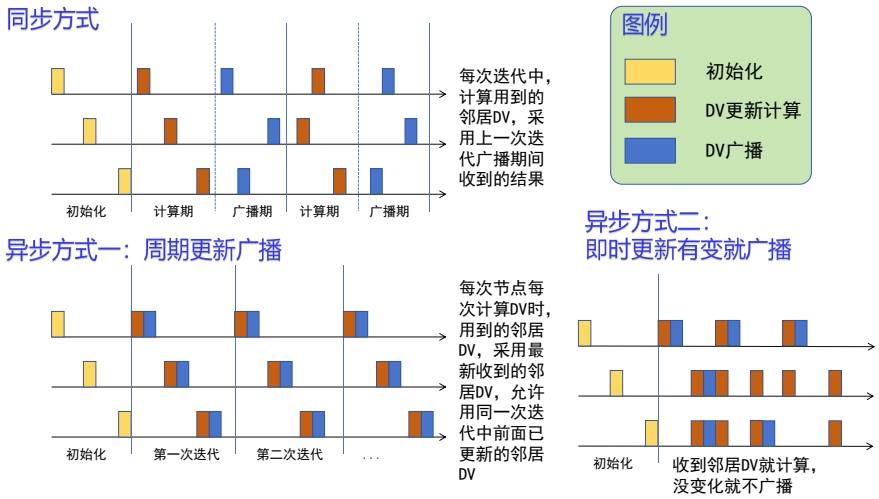

图 3: 更新方式

5.5 Distance Vector更新方式分析（选做实验）

5.5.1 同步更新与异步更新对比

三种可能的更新方式的时序在图3中展示。本次实验“exp7_1a.m”中采取的是异步方式一、在“exp7_1b.m”中提供了同步方式。可以考虑修改已有的“topology.m”文件，生成规模更复杂的拓扑网络（更多的节点数目，也可直接使用“topology_complex.mat”），比较两种更新方式下路由算法的收敛时间及通信量的比较，并分析原因。

5.5.2 异步更新顺序对路由表的影响分析

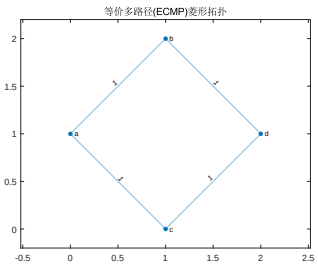

图 4: Ecmp拓扑

采用异步方式二，不同的异步更新顺序可能会导致最终不同的路由表。运行“topology_ecmp.m”程序，生成图4所示的拓扑图，并运行“exp7_1c.m”程序，观查路由表的差异，进行分析。