

计算机网络

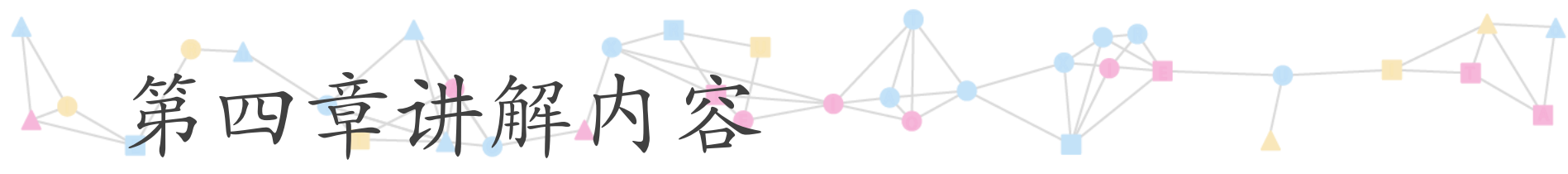
第四章 网络层

谢瑞桃

xie@szu.edu.cn

[rtxie.github.io](https://github.com/rtxie)

计算机与软件学院
深圳大学



1. 网络层概述与数据平面
2. 控制平面



网络层的功能

■ 转发

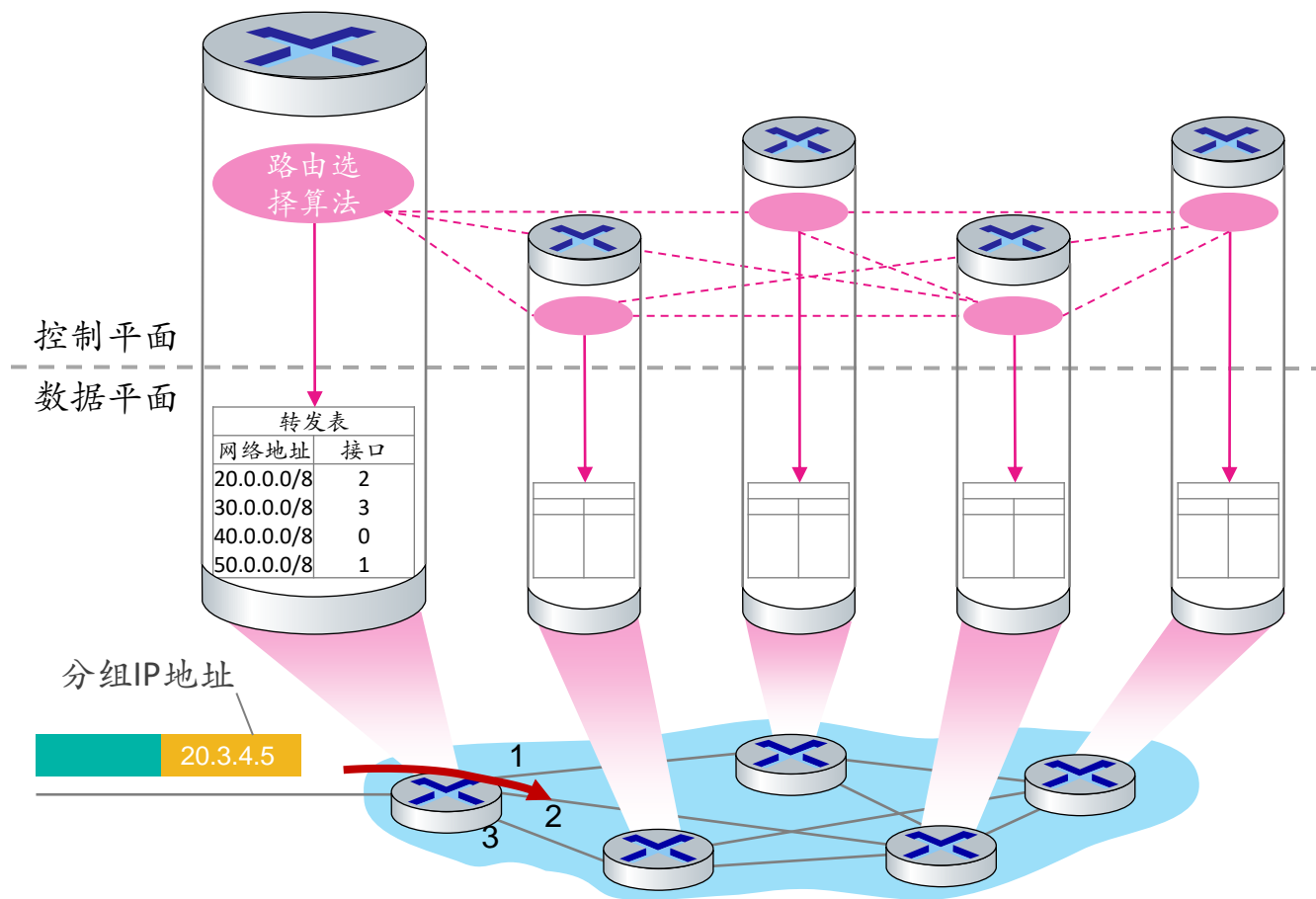
- 路由器将分组由一条输入链路移动到适当的输出链路
- 时间要求是纳秒级别
- 由硬件实现

■ 路由选择

- 决定将分组由源主机移动到目的主机所要经过的路由或路径
- 时间要求是秒级别
- 由软件实现

网络层的功能

- 每个路由器的路由选择组件相互通信，合作生成转发表



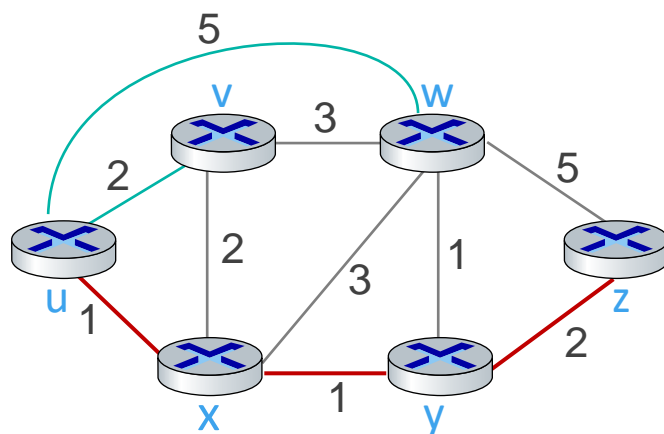


控制平面讲解内容

- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议

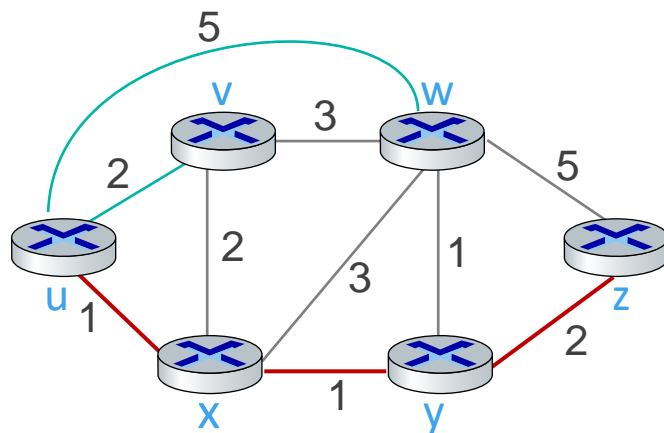
路由协议

- 目的是决定从发送主机到接收主机之间的好的网路路径
- 网络路径：一连串路由器
- 好：最少跳数，最低时延，最低开销



网络建模

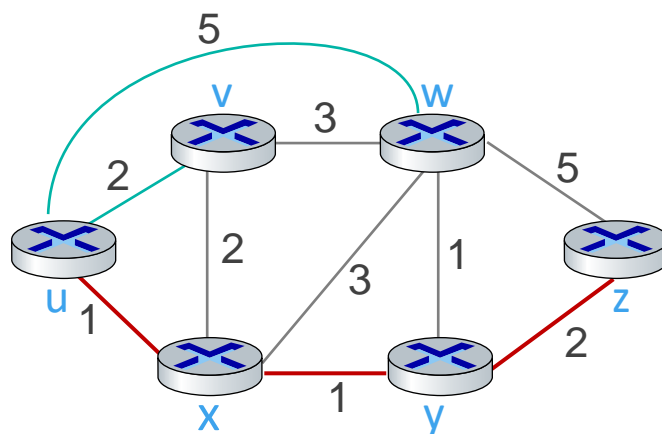
- 图 $G = \langle N, E \rangle$
- 节点集合 $N = \{u, v, w, x, y, z\}$
- 链路集合 $E = \{(u, v), (u, x), (u, w), (v, x), (v, w), (x, w), (x, y), (w, y), (w, z), (y, z)\}$



网络建模

- $c(x, y)$: 链路 (x, y) 的开销
- 路径 $(x_1, x_2, x_3, \dots, x_{n-1}, x_n)$ 的开销:

$$c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{n-1}, x_n)$$





路由算法分类

- 集中式

- 每个路由器都拥有完整的拓扑，链路开销
- 链路状态 (link state) 算法

- 分布式

- 每个路由器只知道邻居节点，以及与邻居节点之间的链路开销
- 迭代地与邻居交换信息，并进行计算
- 距离向量 (distance vector) 算法



链路状态路由选择算法

- 每个节点都知道网络拓扑和所有的链路开销
- 如何知道呢?
- 每个节点向网络中的所有其他节点广播其链路状态，包括链接的标识和开销
- 已知完整网络后，如何求解最低开销路径?
- Dijkstra 算法



第四章知识点汇总

- 了解路由协议的目的
- 了解链路状态路由选择算法的原理



控制平面讲解内容

- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议

八卦



Norman Rockwell (1894-1978), "The Gossips," 1948. Painting for "The Saturday Evening Post" cover, March 6, 1948. Oil on canvas. Private collection. ©SEPS: Curtis Publishing, Indianapolis, IN

<https://www.sothebys.com/en/auctions/ecatalogue/2013/american-art-n09048/lot.16.html>

距离向量路由选择算法

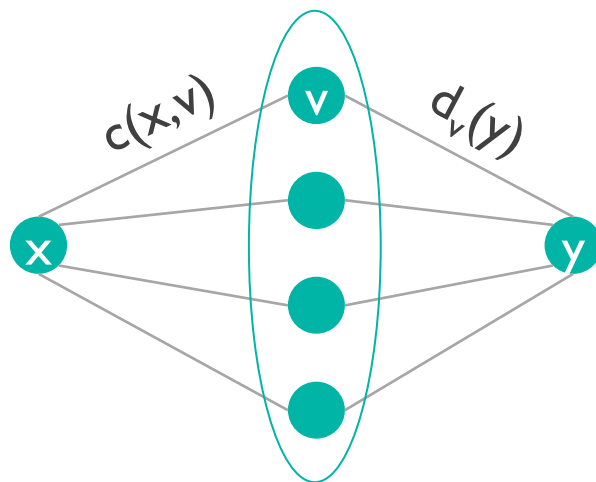
■ Bellman-Ford公式

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

节点x
到节点
y的最
短距离

x到邻居
节点v的
距离

节点v
到节点
y的最
短距离



节点x的邻居节点v的集合



距离向量路由选择算法

■ Bellman-Ford公式

$$d_x(y) = \min_v \{ c(x,v) + d_v(y) \}$$

节点x
到节点
y的最
短距离

x到邻居
节点v的
距离

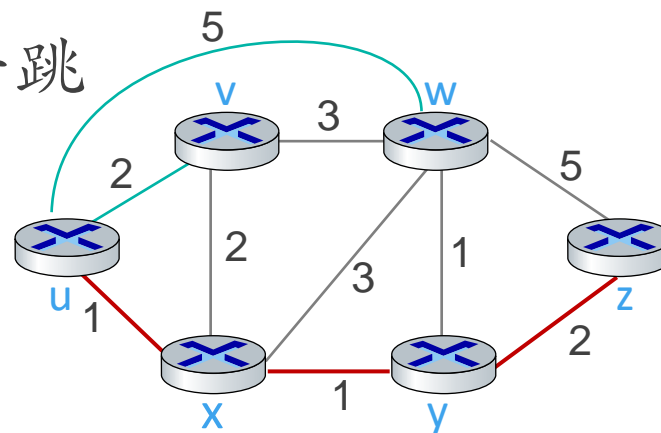
邻居v
到节点
y的最
短距离

- $\mathbf{d}_x = [d_x(y): y \in N]$ 称为节点x的距离向量(Distance Vector)

距离向量路由选择算法

- $d_v(z) = 5, d_x(z) = 3, d_w(z) = 3$
- $d_u(z) = \min \{c(u,v)+d_v(z), c(u,x)+d_x(z), c(u,w)+d_w(z)\}$
 $= \min \{2 + 5, 1 + 3, 5 + 3\} = 4$

- u到z的最短路径上：x是下一跳





距离向量路由选择算法

- $D_x(y)$ = 从x到y的最小开销估计
- $D_x = [D_x(y): y \in N]$ 称为节点x的距离向量估计
- 节点x维护以下信息:
 - x到邻居v的直达开销 $c(x, v)$
 - x的距离向量估计 $D_x = [D_x(y): y \in N]$
 - x的每个邻居的距离向量估计 $D_v = [D_v(y): y \in N]$



距离向量路由选择算法

每个节点独立运行以下程序

等待本地链路变化或收到邻居的DV估计

利用B-F公式重新计算DV估计
$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}, \forall y \in N$$

如果DV发生了变化, 通知邻居("传八卦")

距离向量路由选择算法

节点x的表

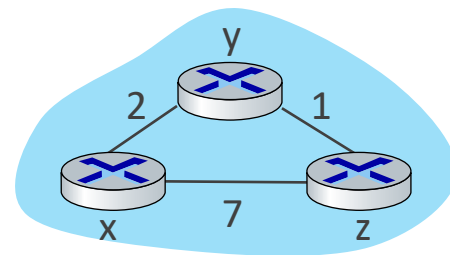
	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

节点z的表

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

节点z的表

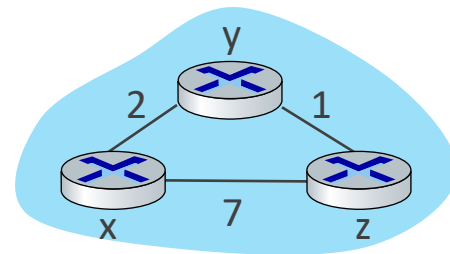
	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0		
y	2	0	1
z	7	1	0

$$D_x(y) = \min\{c(x, y) + D_y(y), c(x, z) + D_z(y)\}$$

来自节点自身的观测

来自邻居节点的传输



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

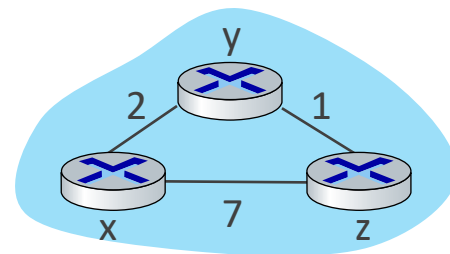
节点z的表

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

$$D_x(y) = \min\{c(x, y) + D_y(y), c(x, z) + D_z(y)\} \\ = \min\{2+0, 7+1\} = 2$$

$$D_x(z) = \min\{c(x, y) + D_y(z), c(x, z) + D_z(z)\} \\ = \min\{2+1, 7+0\} = 3$$



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

节点z的表

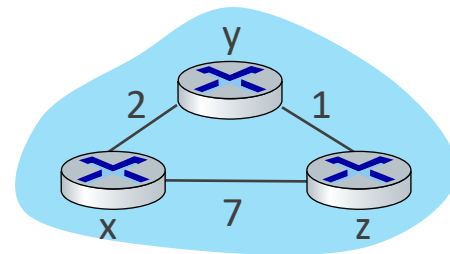
	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

$$D_y(x) = \min\{c(y, x) + D_x(x), c(y, z) + D_z(x)\} \\ = \min\{2+0, 1+7\} = 2$$

$$D_y(z) = \min\{c(y, x) + D_x(z), c(y, z) + D_z(z)\} \\ = \min\{2+7, 1+0\} = 1$$



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

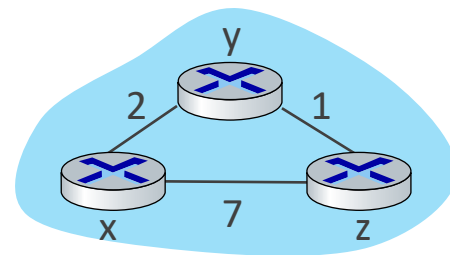
节点z的表

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

节点z的表

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

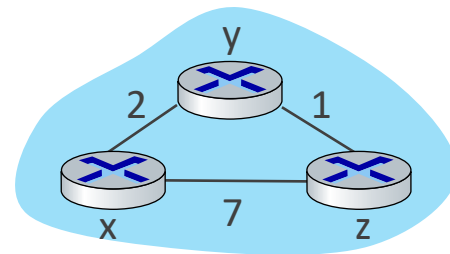
发生了变化，因此需要发给其邻居节点

	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

没发生变化，所以不需要发给邻居节点

	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

发生了变化，因此需要发给其邻居节点



距离向量路由选择算法

节点x的表

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

节点y的表

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

节点z的表

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

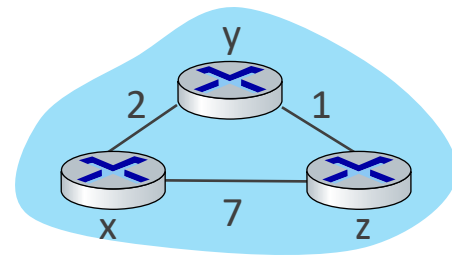
	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

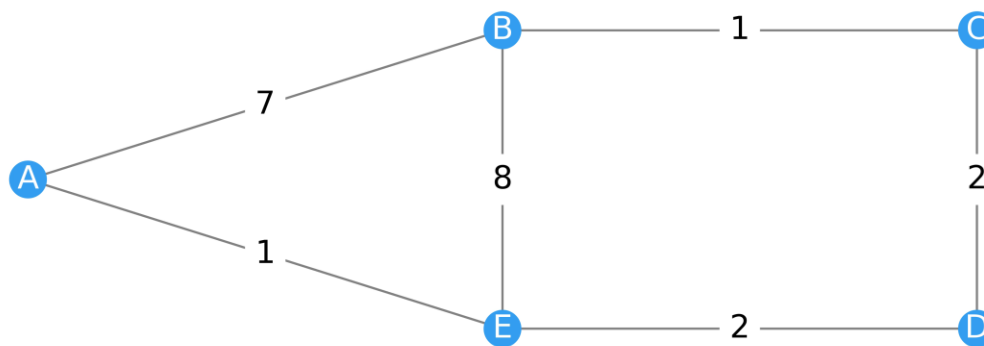




距离向量路由选择算法

- 分布式的
 - 每个节点独立运行自己的DV算法，并与邻居节点交换DV信息
- 迭代的
 - 每个节点从邻居节点接收信息，执行计算，然后再将结果分发给邻居，这一过程重复执行很多次
- 异步的
 - 不需要所有节点步调一致

DV 算法示例一



颜色说明

路由表:

添加新条目

修改旧条目

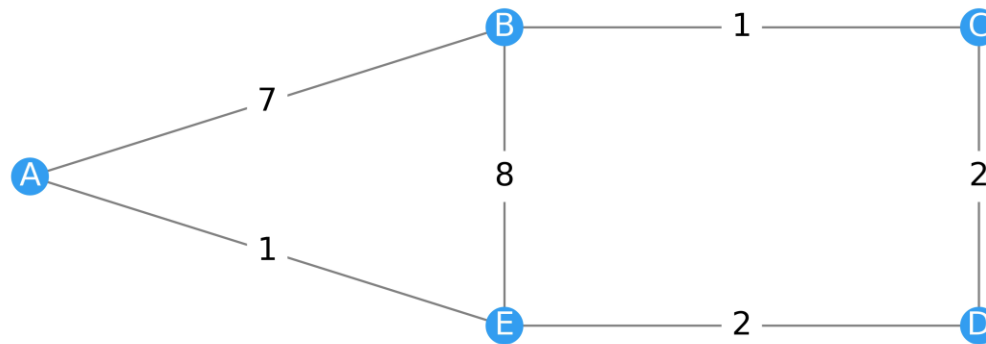
图:

----->

节点间传递DV

Frame 0

DV 算法示例二



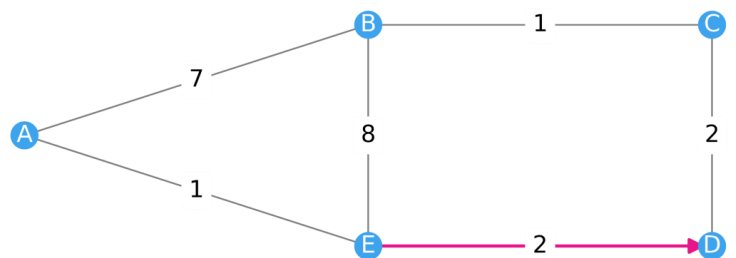
Frame 0

DV算法：殊途同归

- 对比示例一（左）和示例二（右）的第五次更新
- 可见两次运行过程不同

Node A, Round 1

Node	Dist	Next
A	0	A
B	7	B
D	3	E
E	1	E



Node E, Round 0

Node	Dist	Next
A	1	A
B	8	B
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

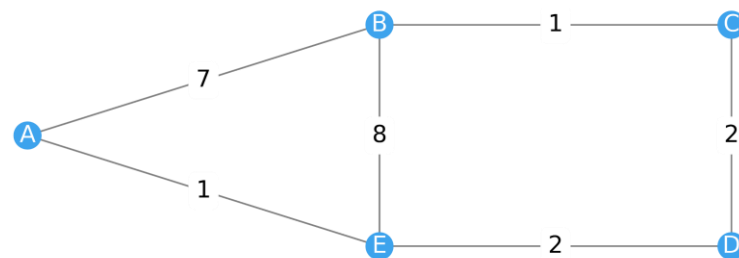
Frame 5

Node B, Round 1

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	8	E

Node C, Round 1

Node	Dist	Next
A	8	B
B	1	B
C	0	C
D	2	D
E	9	B



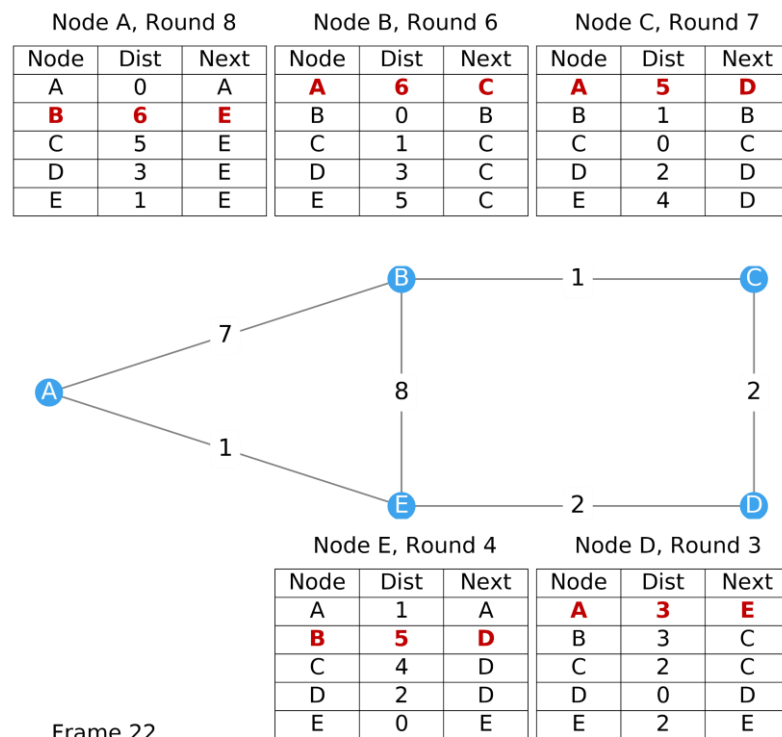
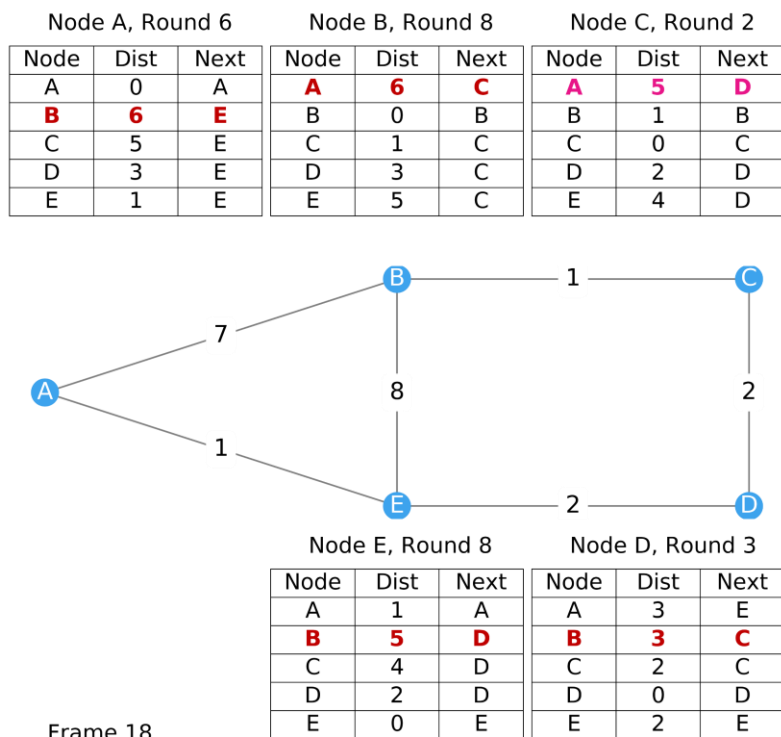
Node D, Round 0

Node	Dist	Next
C	2	C
D	0	D
E	2	E

Frame 5

DV算法：殊途同归

- 对比示例一（左）和示例二（右）收敛后的结果
- 可见两次运行结果相同

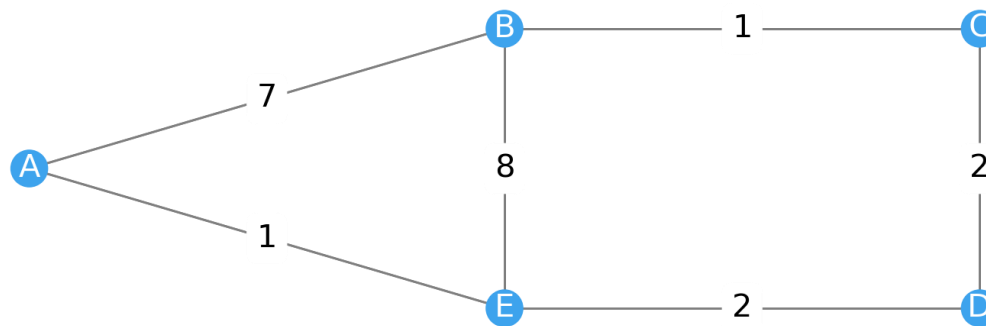




DV 算法示例

- python 实现DV算法
- 每个节点运行一个进程
- 动态显示每个节点的路由表
- 使用multiprocessing实现多进程，matplotlib绘图，matplotlib.animation实现动画
- 大概300行代码
- 感兴趣，你也可以试试看

DV 算法示例一：分解

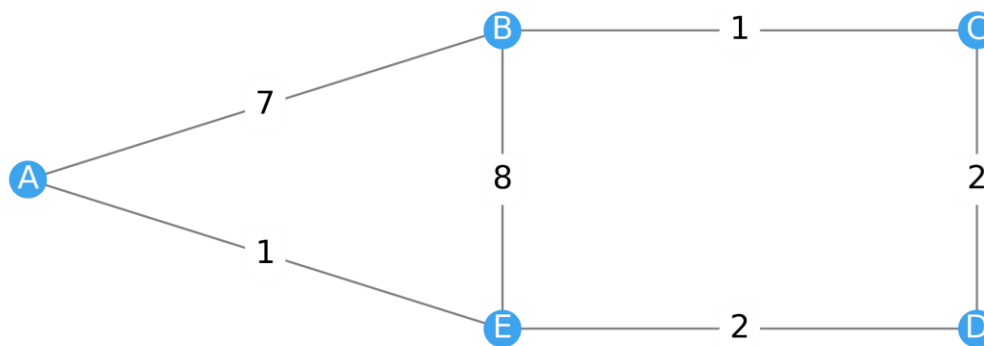


Frame 0

DV 算法示例 (1)

Node A, Round 0

Node	Dist	Next
A	0	A
B	7	B
E	1	E

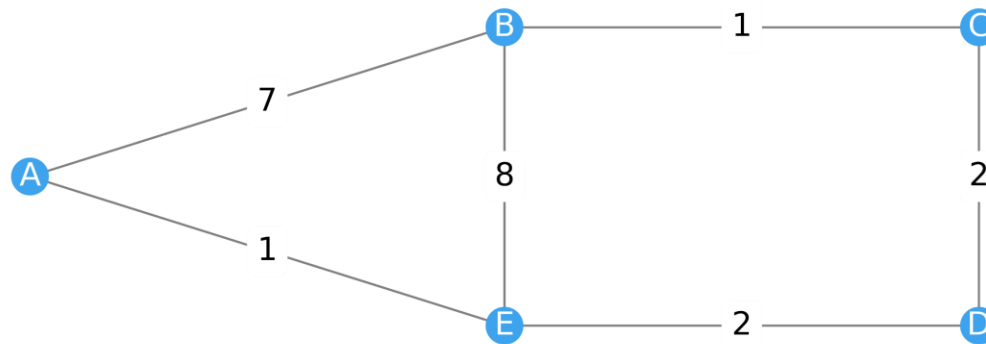


Frame 1

DV 算法示例 (2)

Node A, Round 0

Node	Dist	Next
A	0	A
B	7	B
E	1	E



Node D, Round 0

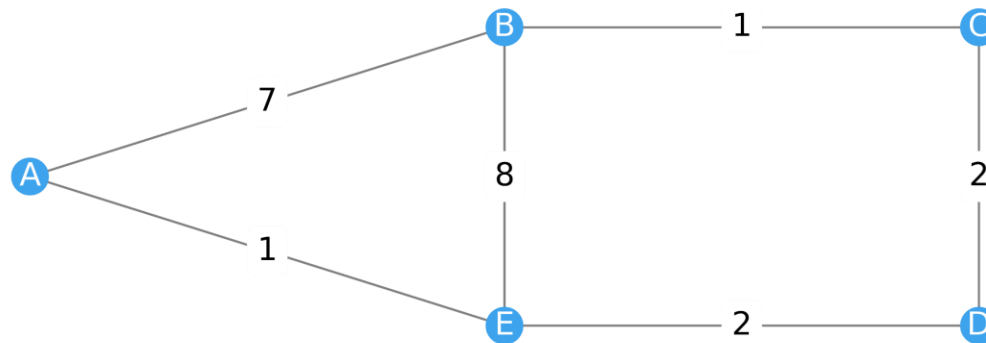
Node	Dist	Next
C	2	C
D	0	D
E	2	E

Frame 2

DV 算法示例 (3)

Node A, Round 0

Node	Dist	Next
A	0	A
B	7	B
E	1	E



Node E, Round 0

Node	Dist	Next
A	1	A
B	8	B
D	2	D
E	0	E

Node D, Round 0

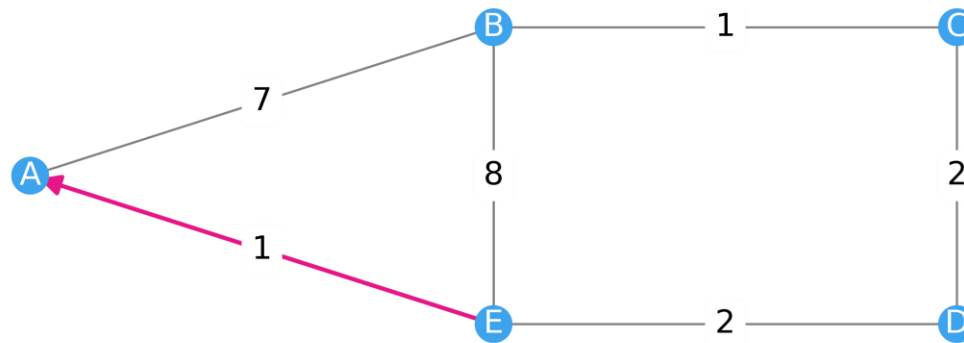
Node	Dist	Next
C	2	C
D	0	D
E	2	E

Frame 3

DV 算法示例 (4)

Node A, Round 1

Node	Dist	Next
A	0	A
B	7	B
D	3	E
E	1	E



Node E, Round 0

Node	Dist	Next
A	1	A
B	8	B
D	2	D
E	0	E

Node D, Round 0

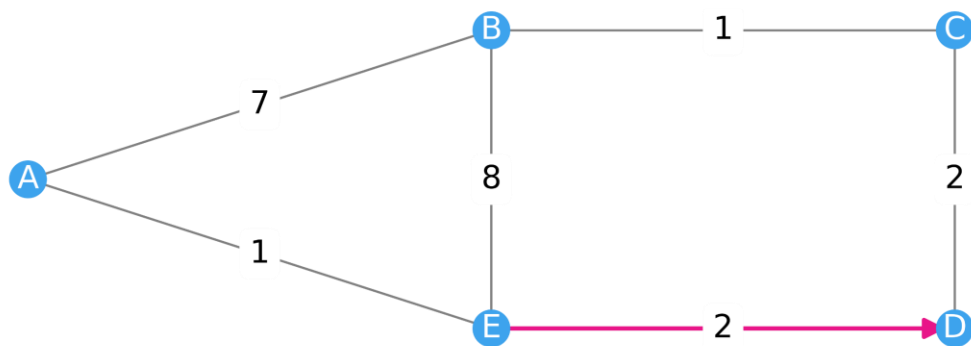
Node	Dist	Next
C	2	C
D	0	D
E	2	E

Frame 4

DV 算法示例 (5)

Node A, Round 1

Node	Dist	Next
A	0	A
B	7	B
D	3	E
E	1	E



Node E, Round 0

Node	Dist	Next
A	1	A
B	8	B
D	2	D
E	0	E

Node D, Round 1

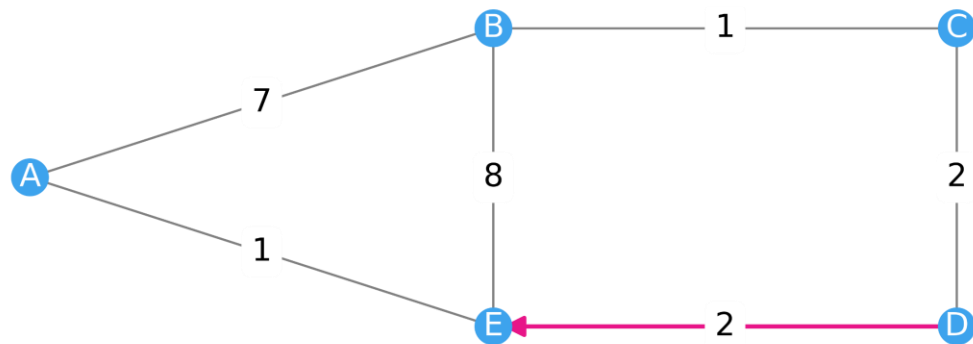
Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 5

DV 算法示例 (6)

Node A, Round 1

Node	Dist	Next
A	0	A
B	7	B
D	3	E
E	1	E



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

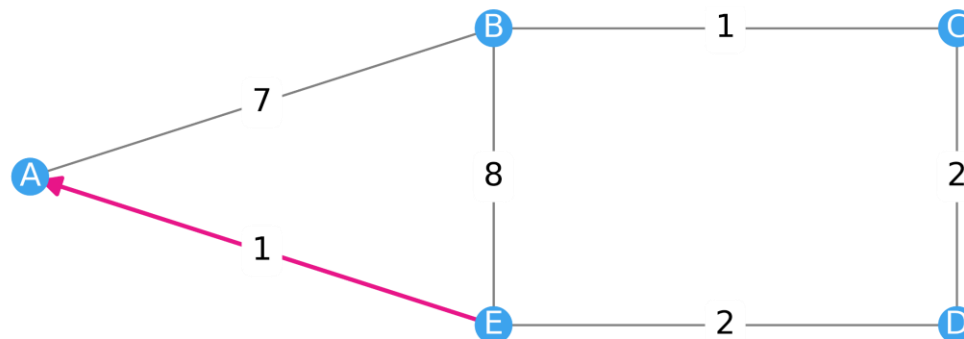
Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 6

DV 算法示例 (7)

Node A, Round 2

Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 7

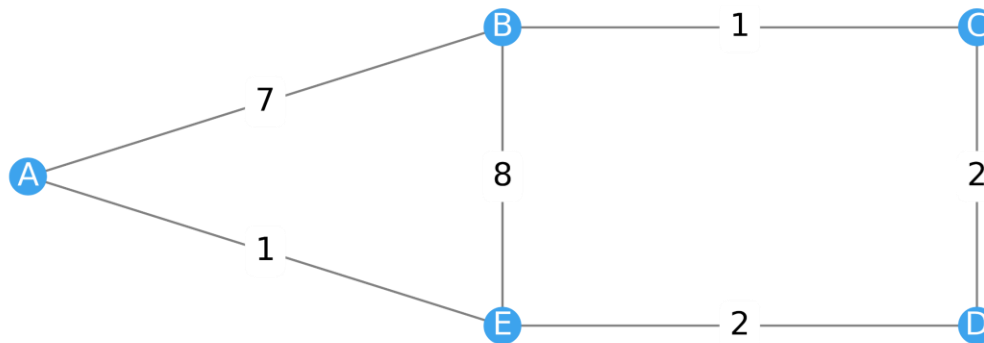
DV 算法示例 (8)

Node A, Round 2

Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 0

Node	Dist	Next
A	7	A
B	0	B
C	1	C
E	8	E



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 8

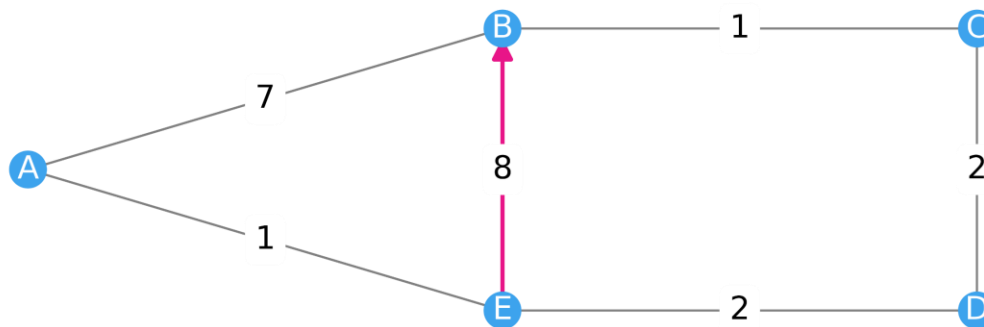
DV 算法示例 (9)

Node A, Round 2

Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 2

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	10	E
E	8	A



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 9

DV 算法示例 (10)

Node A, Round 2

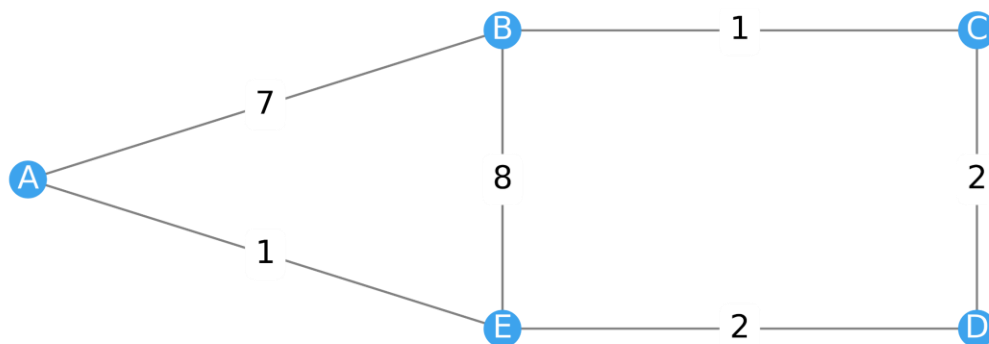
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 2

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	10	E
E	8	A

Node C, Round 0

Node	Dist	Next
B	1	B
C	0	C
D	2	D



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 10

DV 算法示例 (11)

Node A, Round 2

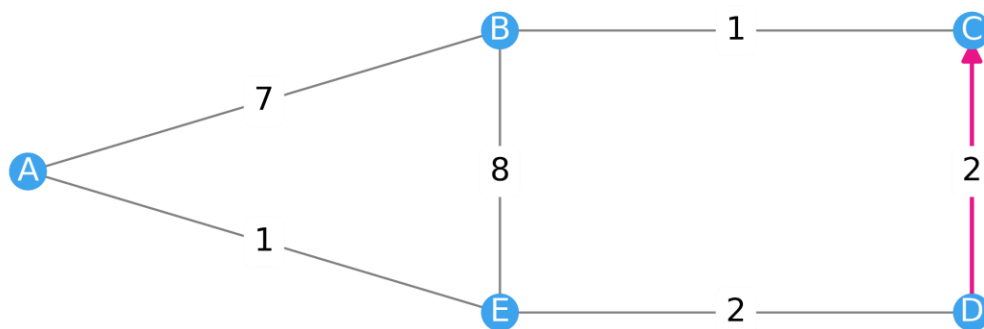
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 2

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	10	E
E	8	A

Node C, Round 1

Node	Dist	Next
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 11

DV 算法示例 (12)

Node A, Round 2

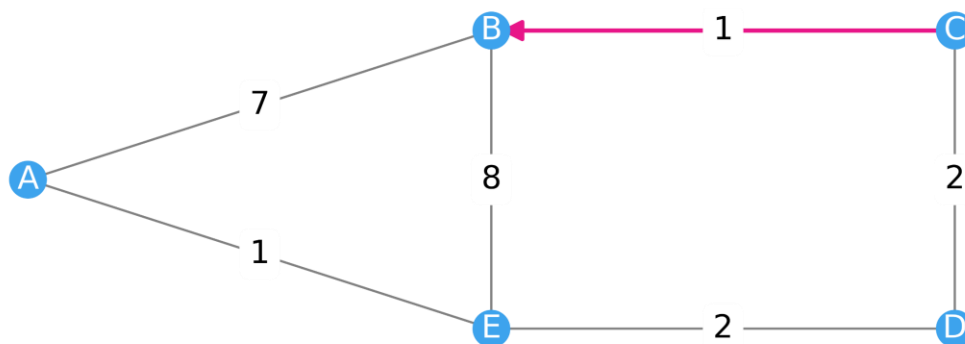
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 6

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	8	A

Node C, Round 1

Node	Dist	Next
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 1

Node	Dist	Next
A	3	E
B	10	E
C	2	C
D	0	D
E	2	E

Frame 12

DV 算法示例 (13)

Node A, Round 2

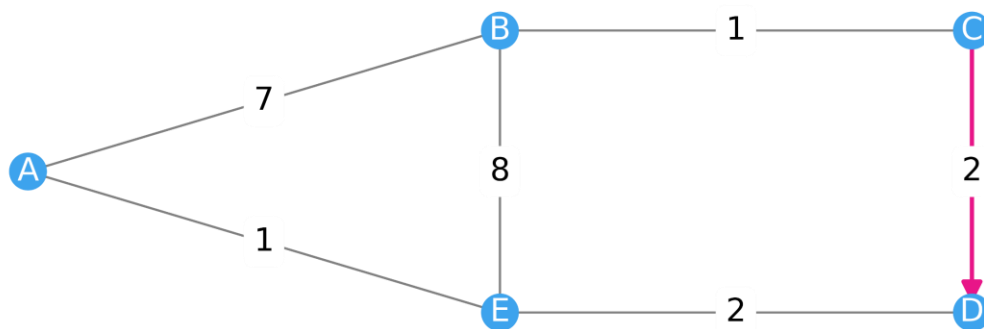
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 6

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	8	A

Node C, Round 1

Node	Dist	Next
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 13

DV 算法示例 (14)

Node A, Round 2

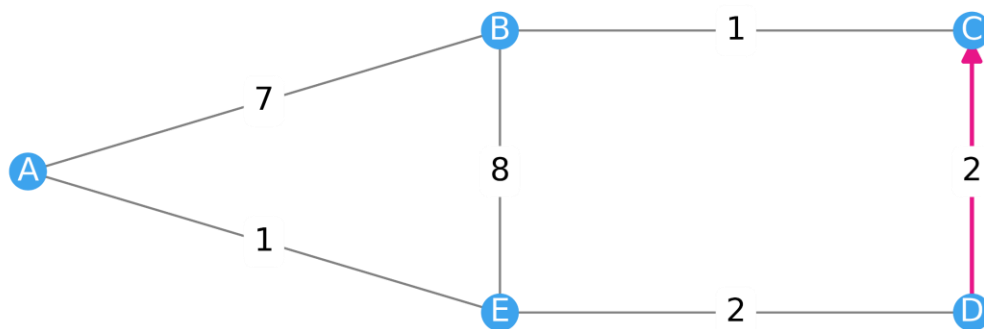
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 6

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	8	A

Node C, Round 2

Node	Dist	Next
A	5	D
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 2

Node	Dist	Next
A	1	A
B	8	A
C	4	D
D	2	D
E	0	E

Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 14

DV 算法示例 (15)

Node A, Round 2

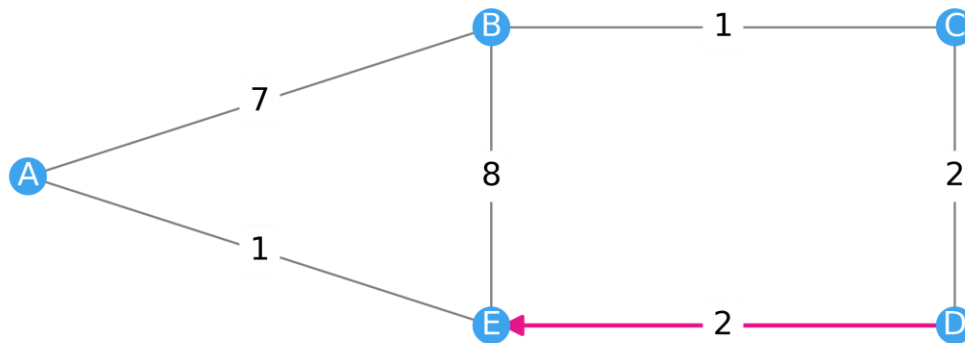
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 6

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	8	A

Node C, Round 2

Node	Dist	Next
A	5	D
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 8

Node	Dist	Next
A	1	A
B	5	D
C	4	D
D	2	D
E	0	E

Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 15

DV 算法示例 (16)

Node A, Round 2

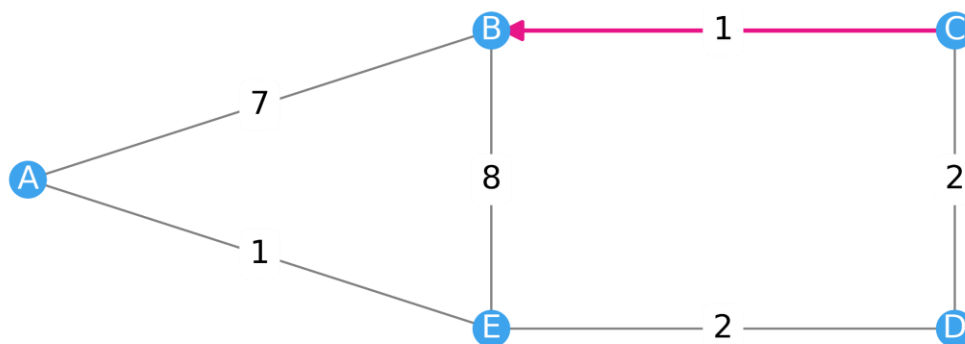
Node	Dist	Next
A	0	A
B	7	B
C	5	E
D	3	E
E	1	E

Node B, Round 7

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	5	C

Node C, Round 2

Node	Dist	Next
A	5	D
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 8

Node	Dist	Next
A	1	A
B	5	D
C	4	D
D	2	D
E	0	E

Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 16

DV 算法示例 (17)

Node A, Round 6

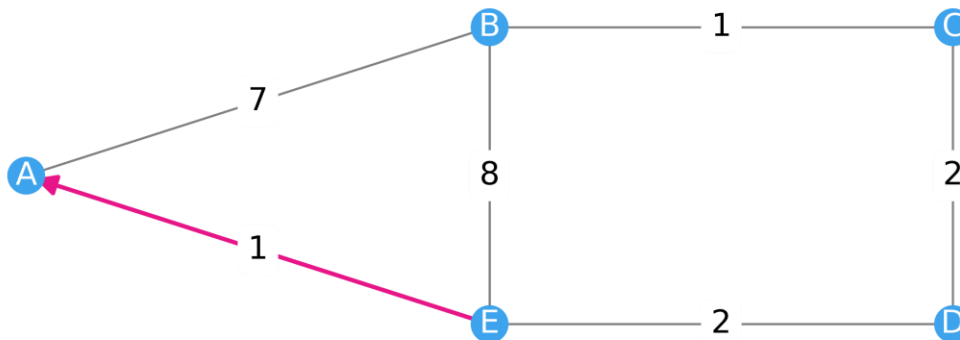
Node	Dist	Next
A	0	A
B	6	E
C	5	E
D	3	E
E	1	E

Node B, Round 7

Node	Dist	Next
A	7	A
B	0	B
C	1	C
D	3	C
E	5	C

Node C, Round 2

Node	Dist	Next
A	5	D
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 8

Node	Dist	Next
A	1	A
B	5	D
C	4	D
D	2	D
E	0	E

Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 17

DV 算法示例 (18)

Node A, Round 6

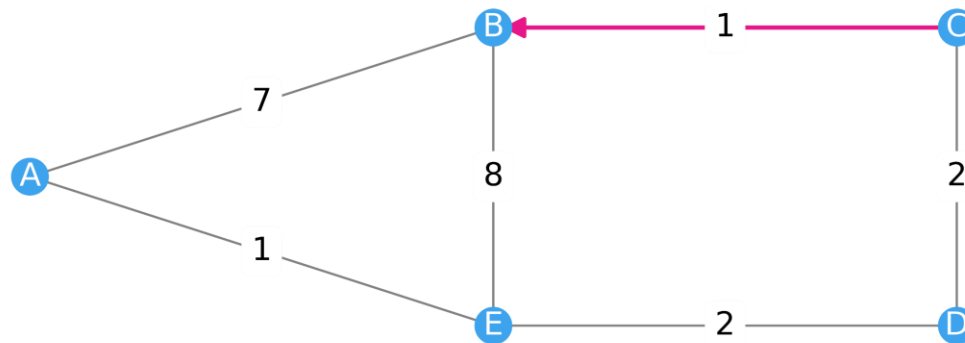
Node	Dist	Next
A	0	A
B	6	E
C	5	E
D	3	E
E	1	E

Node B, Round 8

Node	Dist	Next
A	6	C
B	0	B
C	1	C
D	3	C
E	5	C

Node C, Round 2

Node	Dist	Next
A	5	D
B	1	B
C	0	C
D	2	D
E	4	D



Node E, Round 8

Node	Dist	Next
A	1	A
B	5	D
C	4	D
D	2	D
E	0	E

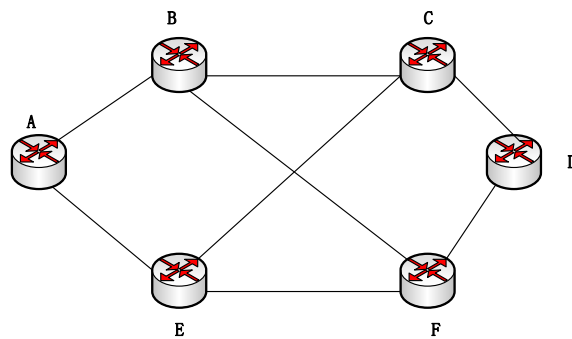
Node D, Round 3

Node	Dist	Next
A	3	E
B	3	C
C	2	C
D	0	D
E	2	E

Frame 18

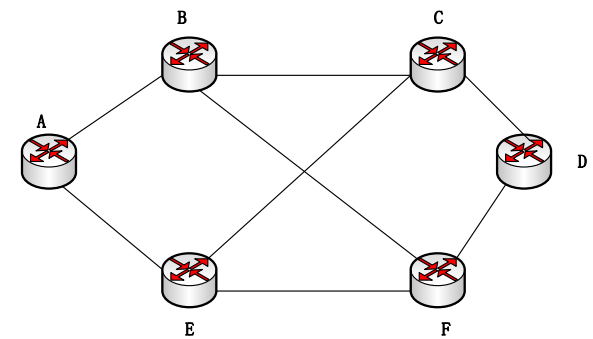
习题

- 考虑如下图所示的采用基于距离-矢量的路由选择算法的子网。假设路由器C刚启动，并测得到它的邻接路由器B、D和E的时延分别是4、5和6。此后，路由器C依次收到下列矢量：来自D的(10,8,5,0,8,9)、来自E的(7,5,3,7,0,3)以及来自B的(5,0,6,10,6,3)。上面的矢量表示的是发送该矢量的结点分别与结点A、B、C、D、E、F的延时。路由器C在收到3个矢量之后的新路由表是什么？



习题

目的路由器	最短距离估计	下一跳路由器
A	$\min(4 + 5, 5 + 10, 6 + 7) = 9$	B
B	$\min(4 + 0, 5 + 8, 6 + 5) = 4$	B
C	0	--
D	$\min(4 + 10, 5 + 0, 6 + 7) = 5$	D
E	$\min(4 + 6, 5 + 8, 6 + 0) = 6$	E
F	$\min(4 + 3, 5 + 9, 6 + 3) = 7$	B





习题

- 注意：
- 1、搞清楚什么是距离向量（估计）
- $D_C = [D_C(A), D_C(B), D_C(C), D_C(D), D_C(E), D_C(F)]$
- 其中既包含到相邻节点的距离，也包含到非相邻节点的距离。“相邻节点有哪些”是自己观测出的。而“非相邻节点有哪些”是靠“传八卦”知道的。
- 2、本题中只要求算节点C的路由表。其他节点的路由表是不要求算的，题目也没有提供足够的信息。



第四章知识点汇总

- 理解DV算法的运行原理
- 理解DV算法的特点



控制平面讲解内容

- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议

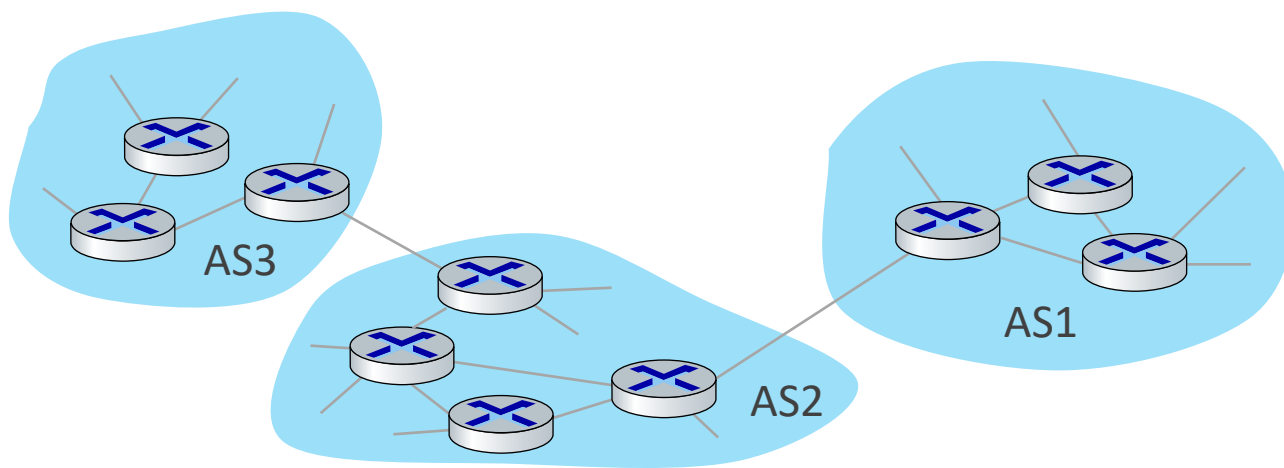


路由选择

- 理想化的LS算法和DV算法不适用于实际情况
- 管理自治
 - 因特网是ISP的网络
 - ISP按照自己的意愿管理网络，运行自己选择的路由选择算法
 - ISP希望对外隐藏网络内部结构
 - ISP希望保护网络免受攻击
- 规模
 - 网络规模越大，广播链路信息的代价越大，时间越久
 - 网络规模越大，迭代的DV算法收敛越慢

可扩展的路由

- 将路由器组织成自治系统 (Autonomous System)
- 一个ISP构成一个AS
- 一个ISP将其网络划分成多个AS



CIDR REPORT for 24 May 21

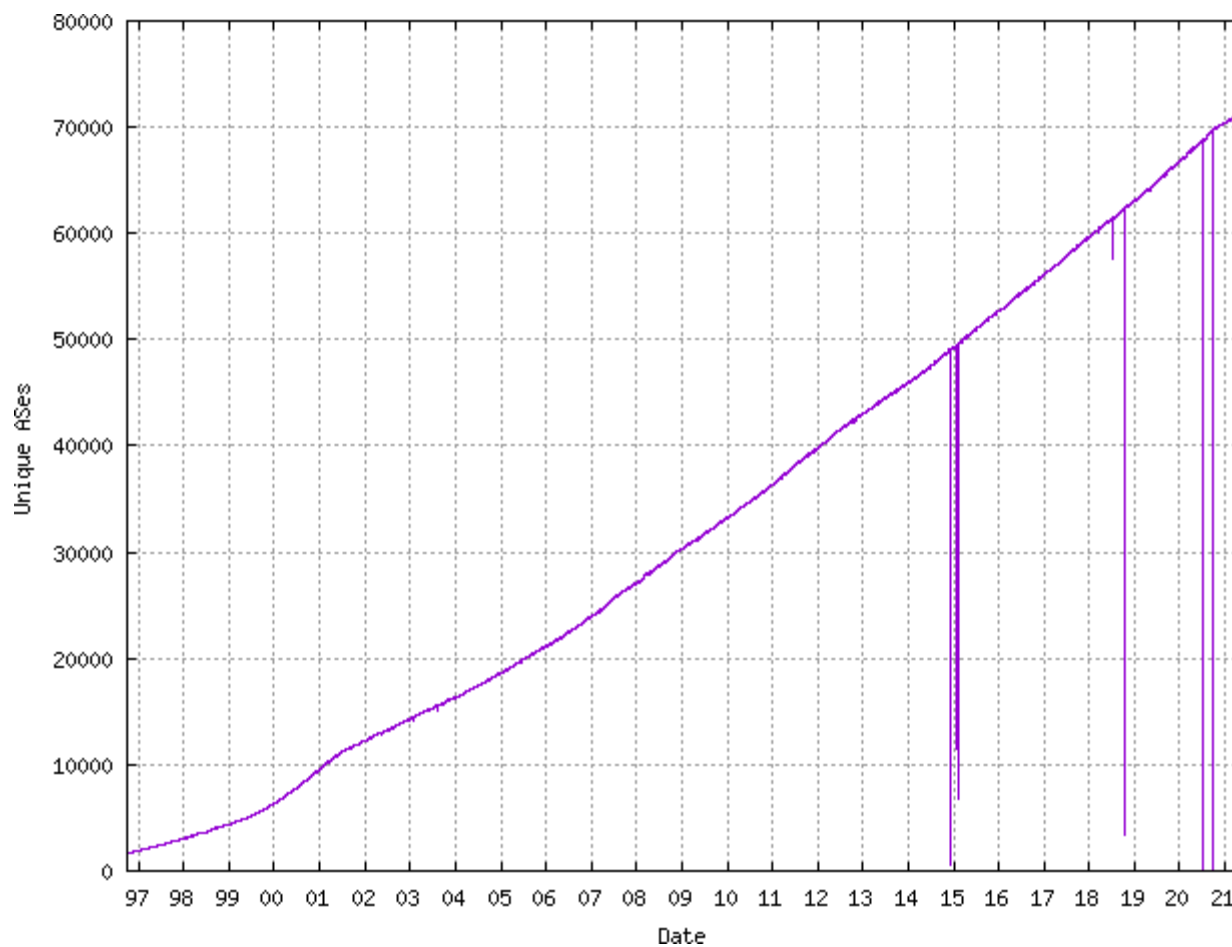
This report was generated at Mon May 24 03:14:39 2021 AEST.

AS Summary

71543	Number of ASes in routing system
25075	Number of ASes announcing only one prefix
8605	Largest number of prefixes announced by an AS AS8151 : Uninet S.A. de C.V., MX
178348544	Largest address span announced by an AS (/32s) AS8003 : GRS-DOD, US

<https://www.cidr-report.org/as2.0/>

AS 小知识



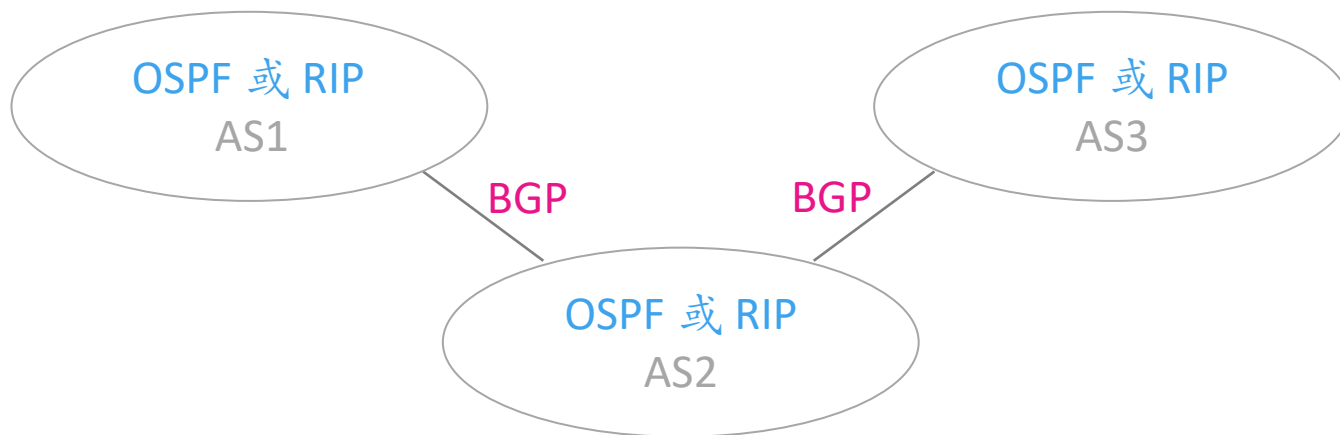
CIDR REPORT for 24 May 21 <https://www.cidr-report.org/as2.0/>



可扩展的路由

■ AS内路由选择协议

- 不同的AS可以选择不同的域内路由协议
- 例如：OSPF，RIP
 - OSPF：Open Shortest Path First (链路状态路由算法)
 - RIP：Routing Information Protocol (距离向量路由算法)

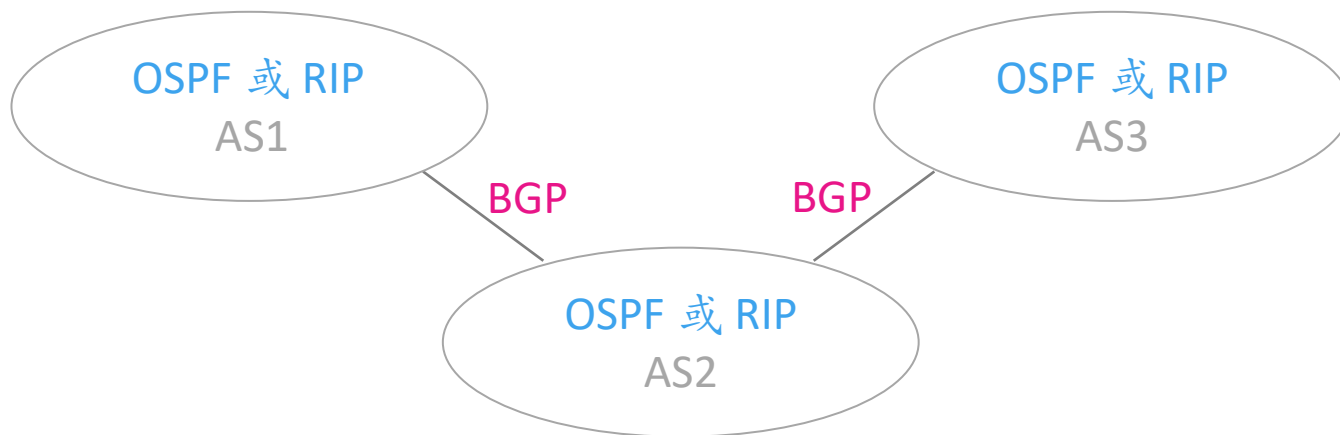




可扩展的路由

■ AS间路由选择协议

- 当源和主机位于不同的AS时，需要AS间路由协议
- Internet中，所有的AS运行相同的AS间路由协议：边界网关协议（Border Gateway Protocol，BGP）





第四章知识点汇总

- 了解自治系统存在的意义
- 了解AS内路由选择协议
- 了解AS间路由选择协议



控制平面讲解内容

- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议

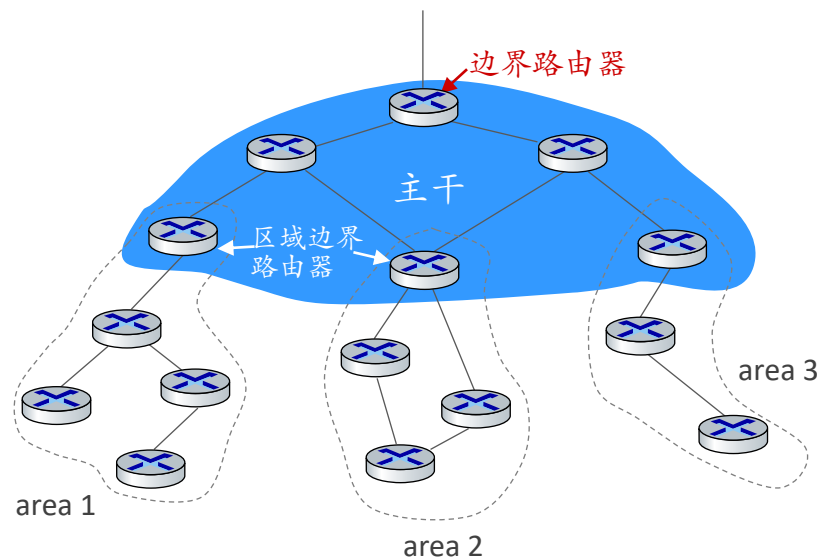


OSPF(开放最短路径优先)

- 利用链路状态算法
 - 需要广播链路状态分组
 - 每个节点构建网络拓扑
 - 利用Dijkstra算法计算路由
- 路由器向AS内的所有其他路由器广播链路状态信息
 - 封装成OSPF报文，直接由IP协议承载

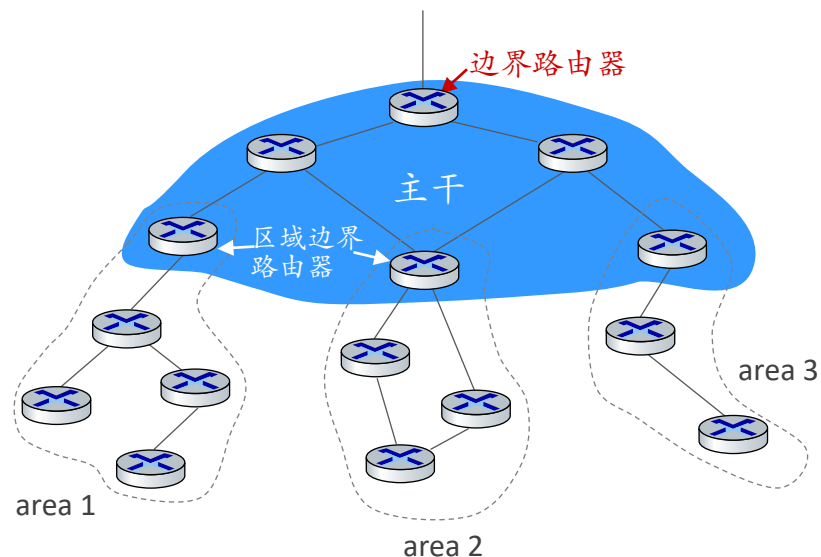
OSPF

- 两层结构：多个区域 + 一个主干



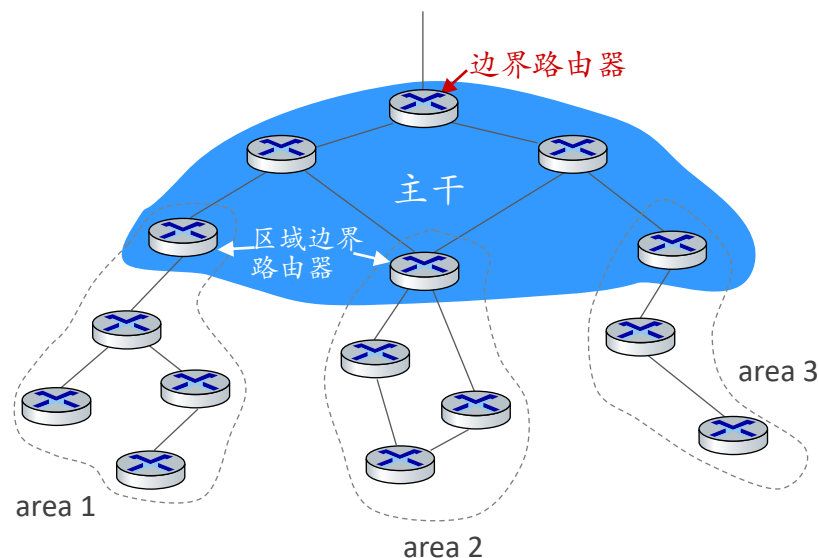
OSPF

- 两层结构：多个区域 + 一个主干
- 每个区域运行自己的OSPF协议，链路状态广播仅限于区域内
- 主干：负责区域之间的路由选择，包含所有区域边界路由器



OSPF

- 两层结构：多个区域 + 一个主干
- 区域边界路由器：负责为流向本区域外的分组提供路由选择
- 边界路由器：连接其他AS





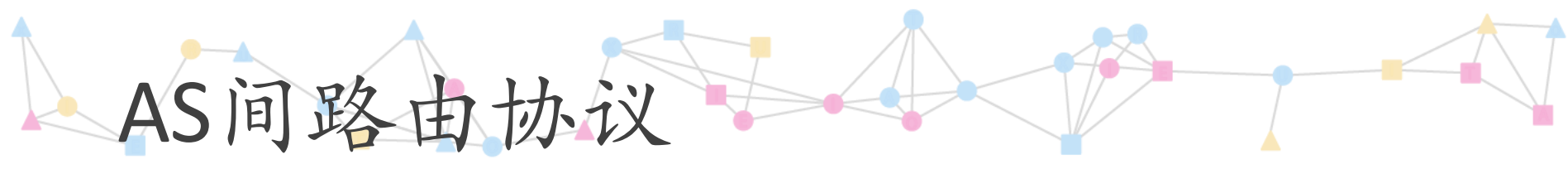
第四章知识点汇总

- 了解OSPF路由协议的特点



控制平面讲解内容

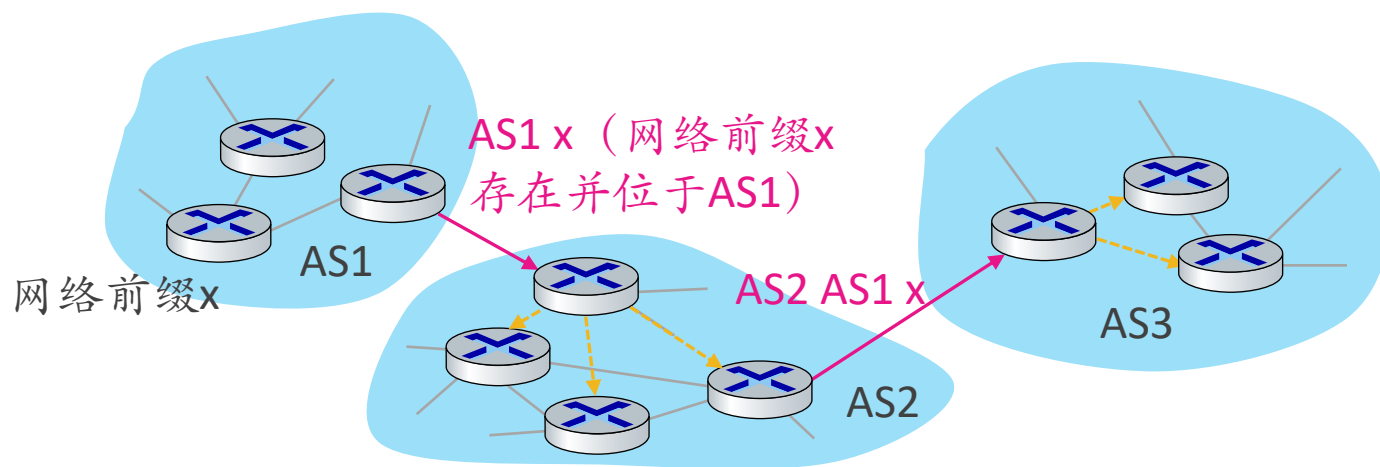
- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议



- BGP: Border Gateway Protocol
 - 对于任意一个网络, BGP使得Internet上的所有AS都知道其存在并可达
 - 确定到达网络最好的路由

AS间路由协议

- 通告网络的存在
 - 使得网络中的所有路由器都知道网络地址x的存在以及通向x的AS路径
 - 利用TCP传输路由选择信息



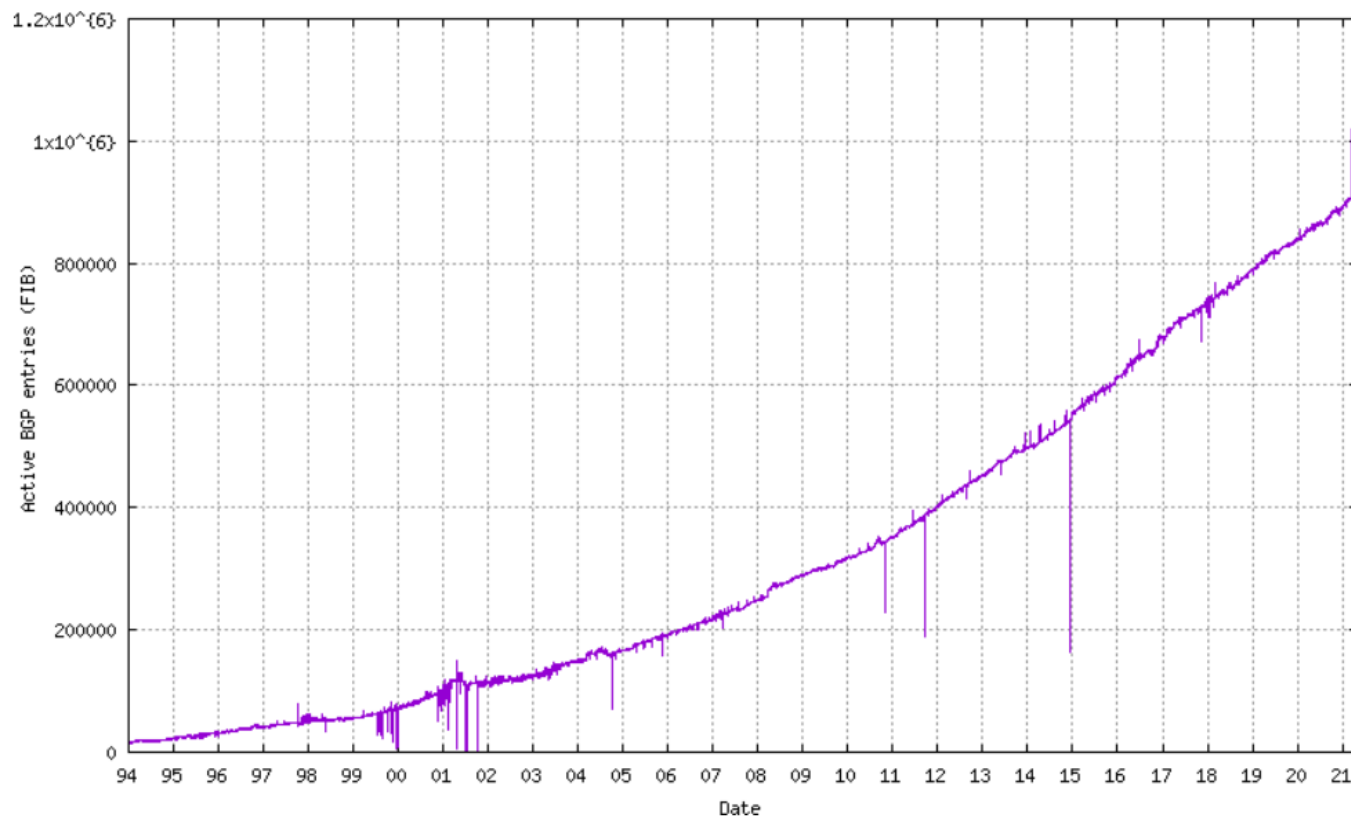


AS间路由协议

- 确定到达网络最好的路由
 - 基于策略
 - 选择能尽快离开当前AS的路径
 - 会用到距离向量算法
 - 需要AS内路由信息

BGP 转发表规模

Active BGP entries (FIB)



<https://bgp.potaroo.net/as6447/>



第四章知识点汇总

- 了解BGP中的网络通告
- 理解路由汇聚的理由

习题

- 1970-2020，为什么研究人员不断地提出新的路由选择算法？

The screenshot shows the IEEE Xplore Digital Library interface. At the top, there is a navigation bar with links for 'Browse', 'My Settings', 'Get Help', and 'Subscribe'. Below this is a search bar with the text 'Enter keywords or phrases (Note: Searches metadata only by default. A search for 'smart grid' = 'smart AND grid')'. The search results are displayed for the keyword 'routing', showing 26-50 of 102,435 results. The results are categorized by document type: Conferences (85,191), Journals (14,290), Magazines (2,002), Books (480), Early Access Articles (320), Standards (103), and Courses (49).

IEEE Xplore®
Digital Library

> Institutional Sign In

IEEE

Browse ▾ My Settings ▾ Get Help ▾ Subscribe

All ▾ Enter keywords or phrases (Note: Searches metadata only by default. A search for 'smart grid' = 'smart AND grid')

Advanced Search | Other Search Options ▾

Search within results 🔍

Per Page: 25 ▾ | Export ▾ | Set Search Alerts ▾ | Search History

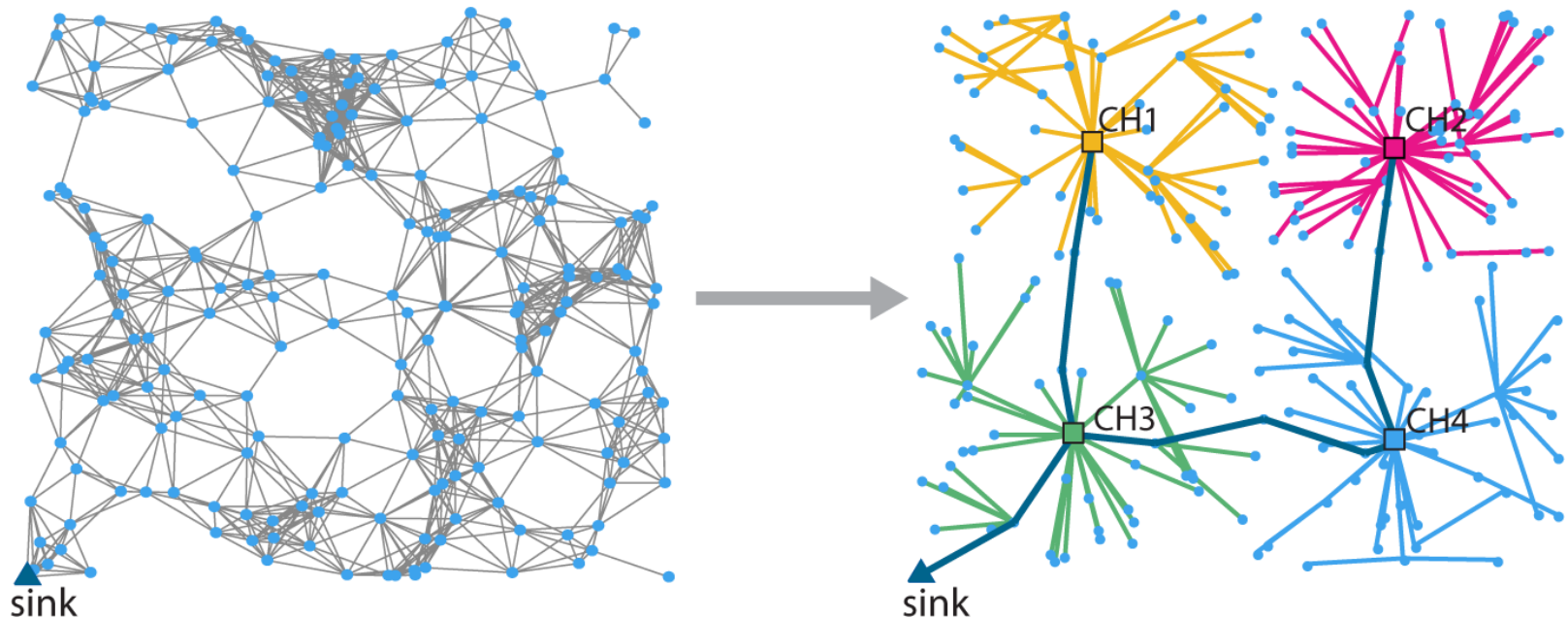
Showing 26-50 of 102,435 for **routing** ✕

<input type="checkbox"/> Conferences (85,191)	<input type="checkbox"/> Journals (14,290)	<input type="checkbox"/> Magazines (2,002)	<input type="checkbox"/> Books (480)
<input type="checkbox"/> Early Access Articles (320)	<input type="checkbox"/> Standards (103)	<input type="checkbox"/> Courses (49)	

Transmission-Efficient Clustering Method for Wireless Sensor Networks Using Compressive Sensing

Ruitao Xie and Xiaohua Jia, *Fellow, IEEE, Computer Society*

https://rtxie.github.io/rtxie.github.io/wp-content/uploads/2017/12/cs_clustering_13.pdf





控制平面讲解内容

- 路由协议的目的与算法分类
- 距离向量路由选择算法
- 可扩展的路由选择
- AS内路由协议：OSPF
- AS间路由协议：BGP
- ICMP协议



ICMP

- Internet Control Message Protocol [RFC 792]
- 主机和路由器用来沟通网络层的信息
 - 差错报告
 - Ping程序：回显请求/应答
 - Traceroute程序
- ICMP报文由IP承载
 - IP的上层协议号为1



ICMP

- ICMP报文

- 类型字段

- 编码字段

- 应答报文还包含：引发ICMP报文生成的IP数据报的首部
ICMP报文的首部和前8个字节



ICMP报文类型

ICMP 类型	编码	描述
0	0	回显应答
3	0	目的网络不可达
3	1	目的主机不可达
3	2	目的协议不可达
3	3	目的端口不可达
3	6	目的网络未知
3	7	目的主机未知
4	0	源抑制
8	0	回显请求
9	0	路由器通告
10	0	路由器发现
11	0	TTL过期
12	0	IP首部损坏

ICMP

■ 利用ICMP实现Traceroute的原理

```
Windows PowerShell
PS C:\Users\ruitao> tracert ieee.org

通过最多 30 个跃点跟踪
到 ieee.org [140.98.193.152] 的路由:

  1    1 ms    <1 毫秒    <1 毫秒 RT-AC68U-BA48 [192.168.2.1]
  2    2 ms    1 ms      1 ms    192.168.1.1
  3    3 ms    3 ms      3 ms    100.64.0.1
  4    7 ms    4 ms      3 ms    202.105.153.237
  5   11 ms   10 ms    10 ms    183.56.65.62
  6   12 ms    9 ms      *      202.97.94.138
  7   28 ms   21 ms    22 ms    202.97.94.98
  8  174 ms  171 ms   182 ms    202.97.51.154
  9  238 ms  169 ms   169 ms    202.97.50.78
 10  171 ms  170 ms   171 ms   TenGigE0-1-0-5. GW6. SJC7. ALTER. NET [152.179.48.149]
 11  244 ms  248 ms   248 ms   Bundle-Ether11. GW8. EWR6. ALTER. NET [140.222.235.239]
 12  251 ms  251 ms   253 ms   ieee-gw.customer.alter.net [152.193.14.46]
 13  251 ms  267 ms   252 ms   140.98.207.204
 14  241 ms  240 ms    *      anakin-ext.ieee.org [140.98.210.1]
 15  242 ms  275 ms   245 ms   ieee.net [140.98.193.152]

跟踪完成。
```

ICMP

- 源节点向目的节点发送ICMP报文（类型是回显请求），IP报文的TTL=1，重复发三次
- 到达第一个路由器时，TTL减1，为0，触发类型为TTL过期的ICMP报文。注意该报文的源地址就是该路由器。

Time	Source	Source Port	Destination	Destination Port	Protocol	Info
17.666202	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=790/5635, ttl=1 (no response found!)
17.667933	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
17.668265	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=791/5891, ttl=1 (no response found!)
17.668978	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
17.669239	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=792/6147, ttl=1 (no response found!)
17.669892	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.674473	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=793/6403, ttl=2 (no response found!)
18.676835	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.678335	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=794/6659, ttl=2 (no response found!)
18.679347	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.680713	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=795/6915, ttl=2 (no response found!)
18.681809	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.687983	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)
20.189246	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)
21.734860	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)

> Internet Protocol Version 4, Src: 192.168.2.178, Dst: 140.98.193.152

▼ Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0x4a49 [correct]
- [Checksum Status: Good]
- Identifier (BE): 1 (0x0001)
- Identifier (LE): 256 (0x0100)
- Sequence number (BE): 786 (0x0312)
- Sequence number (LE): 4611 (0x1203)
- [\[Response frame: 20\]](#)

wireshark_Ethernet_3_20200401190951_a12956.pcapng

分组: 795 · 已显示: 102 (12.8%) · 已丢弃: 0 (0.0%)

配置: Default

ICMP

- 源节点向目的节点发送ICMP报文（类型是回显请求），IP报文的TTL=2，重复发三次
- 到达第二个路由器时，TTL减为0，触发类型为TTL过期的ICMP报文。注意该报文的源地址就是该路由器。

Time	Source	Source Port	Destination	Destination Port	Protocol	Info
17.666202	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=790/5635, ttl=1 (no response found!)
17.667933	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
17.668265	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=791/5891, ttl=1 (no response found!)
17.668978	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
17.669239	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=792/6147, ttl=1 (no response found!)
17.669892	192.168.2.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.674473	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=793/6403, ttl=2 (no response found!)
18.676835	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.678335	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=794/6659, ttl=2 (no response found!)
18.679347	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.680713	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=795/6915, ttl=2 (no response found!)
18.681809	192.168.1.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
18.687983	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)
20.189246	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)
21.734860	192.168.1.1	137	192.168.2.178	137	ICMP	Destination unreachable (Port unreachable)

> Internet Protocol Version 4, Src: 192.168.2.178, Dst: 140.98.193.152

▼ Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0x4a49 [correct]
- [Checksum Status: Good]
- Identifier (BE): 1 (0x0001)
- Identifier (LE): 256 (0x0100)
- Sequence number (BE): 786 (0x0312)
- Sequence number (LE): 4611 (0x1203)
- [\[Response frame: 20\]](#)

wireshark_Ethernet_3_20200401190951_a12956.pcapng

分组: 795 · 已显示: 102 (12.8%) · 已丢弃: 0 (0.0%)

配置: Default

ICMP

- 有些TTL过期类型的ICMP报文丢失了，例如第14跳，发送了三个ping请求报文，只收到两个TTL过期报文。

Time	Source	Source Port	Destination	Destination Port	Protocol	Info
76.350366	140.98.206.204		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
76.352223	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=828/15363, ttl=13 (no response found!)
76.604264	140.98.207.204		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
81.869473	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=829/15619, ttl=14 (no response found!)
82.110687	140.98.210.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
82.111488	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=830/15875, ttl=14 (no response found!)
82.351455	140.98.210.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
82.353372	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=831/16131, ttl=14 (no response found!)
86.139773	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=832/16387, ttl=15 (reply in 693)
86.381775	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=832/16387, ttl=239 (request in 690)
86.382215	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=833/16643, ttl=15 (reply in 717)
86.657797	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=833/16643, ttl=239 (request in 694)
86.658223	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=834/16899, ttl=15 (reply in 741)
86.903580	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=834/16899, ttl=239 (request in 718)

> Internet Protocol Version 4, Src: 192.168.2.178, Dst: 140.98.193.152

▼ Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0x4a49 [correct]
[Checksum Status: Good]
- Identifier (BE): 1 (0x0001)
- Identifier (LE): 256 (0x0100)
- Sequence number (BE): 786 (0x0312)
- Sequence number (LE): 4611 (0x1203)
- [\[Response frame: 20\]](#)

wireshark_Ethernet_3_20200401190951_a12956.pcapng | 分组: 795 · 已显示: 102 (12.8%) · 已丢弃: 0 (0.0%) | 配置: Default

ICMP

- 当TTL增加到一定值时，源节点到目的节点发送的ICMP回显请求类型的报文成功到达了目的节点。
- 目的节点回复ICMP回显应答类型的报文。源节点收到以后结束程序。

Time	Source	Source Port	Destination	Destination Port	Protocol	Info
76.350366	140.98.206.204		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
76.352223	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=828/15363, ttl=13 (no response found!)
76.604264	140.98.207.204		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
81.869473	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=829/15619, ttl=14 (no response found!)
82.110687	140.98.210.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
82.111488	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=830/15875, ttl=14 (no response found!)
82.351455	140.98.210.1		192.168.2.178		ICMP	Time-to-live exceeded (Time to live exceeded in transit)
82.353372	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=831/16131, ttl=14 (no response found!)
86.139773	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=832/16387, ttl=15 (reply in 693)
86.381775	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=832/16387, ttl=239 (request in 690)
86.382215	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=833/16643, ttl=15 (reply in 717)
86.657797	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=833/16643, ttl=239 (request in 694)
86.658223	192.168.2.178		140.98.193.152		ICMP	Echo (ping) request id=0x0001, seq=834/16899, ttl=15 (reply in 741)
86.903580	140.98.193.152		192.168.2.178		ICMP	Echo (ping) reply id=0x0001, seq=834/16899, ttl=239 (request in 718)

> Internet Protocol Version 4, Src: 192.168.2.178, Dst: 140.98.193.152

▼ Internet Control Message Protocol

- Type: 8 (Echo (ping) request)
- Code: 0
- Checksum: 0x4a49 [correct]
[Checksum Status: Good]
- Identifier (BE): 1 (0x0001)
- Identifier (LE): 256 (0x0100)
- Sequence number (BE): 786 (0x0312)
- Sequence number (LE): 4611 (0x1203)
- [\[Response frame: 20\]](#)

wireshark_Ethernet_3_20200401190951_a12956.pcapng

分组: 795 · 已显示: 102 (12.8%) · 已丢弃: 0 (0.0%)

配置: Default



第四章知识点汇总

- 了解ICMP协议的用途
- 了解Traceroute程序实现的原理

If you shut your door to all errors truth will be shut out.

如果你把所有的错误都关在门外，
真理也要被关在门外了。

——*Tagore*