

第四讲 网络路由

中国科学院计算技术研究所

网络技术研究中心

本讲提纲

- 网络路由

- 路由路径计算

- 距离向量方法

- 链路状态方法

- 路由协议

- 域内路由协议 RIP, OSPF

- 域间路由协议 BGP

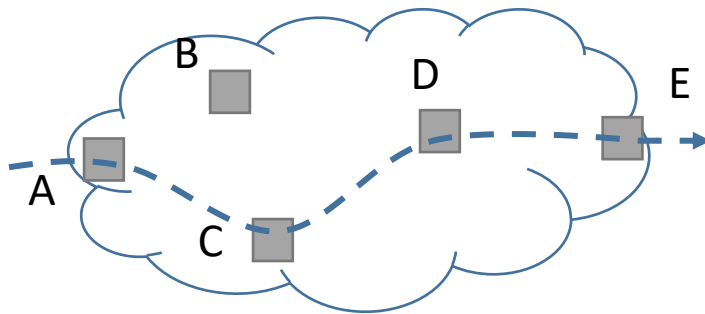
- 考虑主机移动的路由机制

- Mobile IP

- 基于扁平化标识的路由

什么是网络路由？

- 为每份数据传输**确定一条端到端的路径**
 - 通常是寻找两节点间代价最小的路径
 - 传输路径保存在路由器的转发表（FIB）中
- 网络路由通常是基于分布式计算的
 - 考虑节点、链路的变动

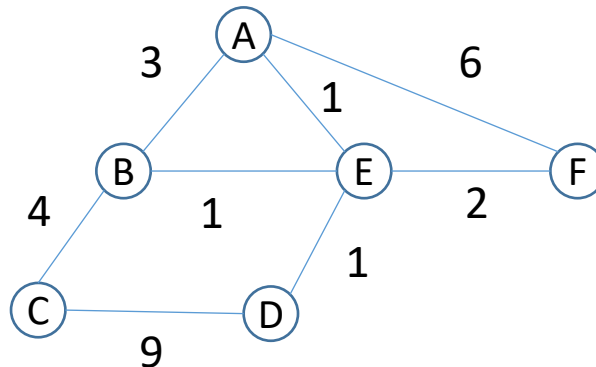


Router A's FIB

Dest	Cost	Next Hop
E	n	C

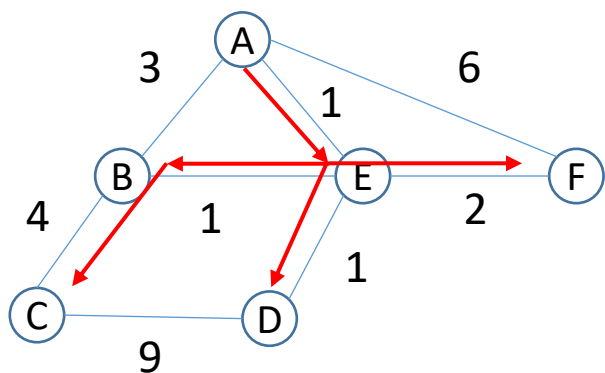
网络模型

- 使用无向图表示网络
 - 每个路由器是图中的一个节点
 - 节点间的链路是图中的一条带权重的边
 - 边的开销(cost)可以是延迟、拥塞程度，甚至价格等
 - 目标：为图中的任意两个节点寻找一条开销最小的边
 - $\text{cost } P(x, y) = \sum_{l \in P(x, y)} \text{cost}(l)$



路由路径例子

A节点的路由路径



A节点处的转发表

Dest	Cost	Next Hop
A	0	A
B	2	E
C	6	B
D	2	E
E	1	E
F	3	E

- 使用最短路径算法，计算到每个其它节点的最短路径
- 对于A节点，到其它节点的最短路径构成一个树
 - 最短路径树 (shortest path tree)

计算路由路径

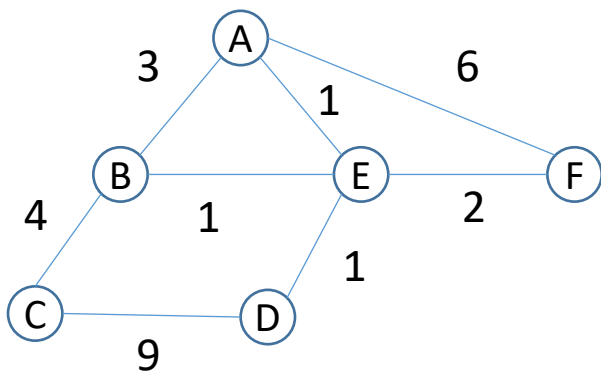
- 集中式方法
 - 集中节点收集网络拓扑信息
 - 使用最短路径算法计算最短路径
 - 将路由路径以转发表形式下发到各个节点
- 分布式方法

距离向量(Distance Vector)方法	链路状态(Link State)方法
节点没有全局拓扑信息	每个节点保存全局拓扑信息
节点将自己计算的表发送给邻居节点	节点将自己的邻接关系通告给所有其它节点
节点根据收到邻居的表，迭代计算自己的表	节点根据全局拓扑信息，独立计算转发表

距离向量方法

- 基本思想

- 任何时刻，每个节点保存到目的节点的**已知最优路径**的开销和对应下一跳节点；如果不可达，用 ∞ 表示
- 初始化：只有到相邻节点的开销和下一跳节点信息
- 迭代：收到相邻节点的消息后，比较选择更优的开销和下一跳节点

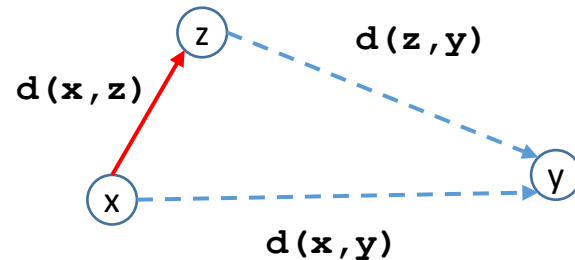


Dest	Cost	Next Hop
A	0	A
B	3	B
C	∞	-
D	∞	-
E	1	E
F	6	F

距离向量更新

- 对于节点 x ，其到 y 的开销为 $d(x,y)$ ，下一跳节点为 $nh(x,y)$
- 收到邻居节点 z 的距离向量消息后， x 更新路由表：

```
update(x, y, z):  
    d ← d(x,z) + d(z,y)  
    if d < d(x,y):  
        advertise to neighbors  
        return d, z  
    else:  
        return d(x,y), nh(x,y)
```



Bellman-Ford算法

- 距离向量方法本质上是一个分布式的Bellman-Ford最短路径算法

```
while not converged:
```

```
  for each node x:
```

```
    for each neighbor z:
```

```
      for each destination y:
```


```
        d <- d(x,z) + d(z,y)
```

```
        if d < d(x,y):
```

```
          d(x,y), nh(x,y) <- d, z
```

```
          advertise to neighbors
```

收到邻居节点消息

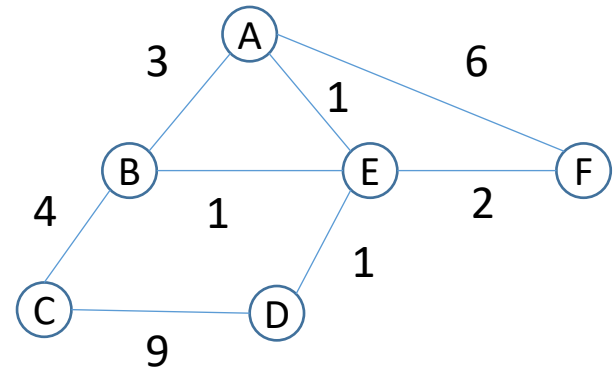


更新并通告路由信息



距离向量例子 – 初始化

- 距离向量初始化
 - 每个节点只有1跳以内的路由信息



节点A

Dest	Cost	Next H
A	0	A
B	3	B
C	∞	-
D	∞	-
E	1	E
F	6	F

节点B

Dest	Cost	Next H
A	3	A
B	0	B
C	4	C
D	∞	-
E	1	E
F	∞	-

节点C

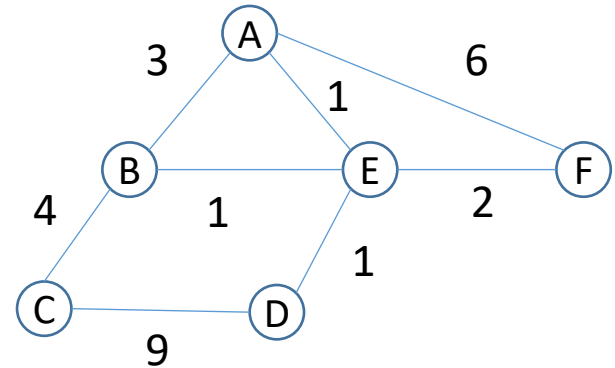
Dest	Cost	Next H
A	∞	-
B	4	B
C	0	C
D	9	D
E	∞	-
F	∞	-

节点E

Dest	Cost	Next H
A	1	A
B	1	B
C	∞	-
D	1	D
E	0	E
F	2	F

距离向量例子 – 第1次更新

- 节点收到邻居节点的路由信息
 - 更新本地路由表
 - 路由表中保存2跳可达的路由信息



节点A

Dest	Cost	Next H
A	0	A
B	2	E
C	7	B
D	2	E
E	1	E
F	3	E

节点B

Dest	Cost	Next H
A	2	E
B	0	B
C	4	C
D	2	E
E	1	E
F	3	E

节点C

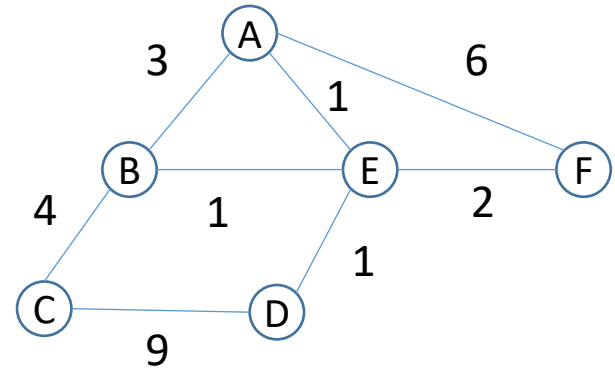
Dest	Cost	Next H
A	7	B
B	4	B
C	0	C
D	9	D
E	5	B
F	∞	-

节点E

Dest	Cost	Next H
A	1	A
B	1	B
C	5	B
D	1	D
E	0	E
F	2	F

距离向量例子 – 第2次更新

- 节点收到邻居节点的路由信息
 - 更新本地路由表
 - 路由表中保存3跳可达的路由信息



节点A

Dest	Cost	Next H
A	0	A
B	2	E
C	6	E
D	2	E
E	1	E
F	3	E

节点B

Dest	Cost	Next H
A	2	E
B	0	B
C	4	C
D	2	E
E	1	E
F	3	E

节点C

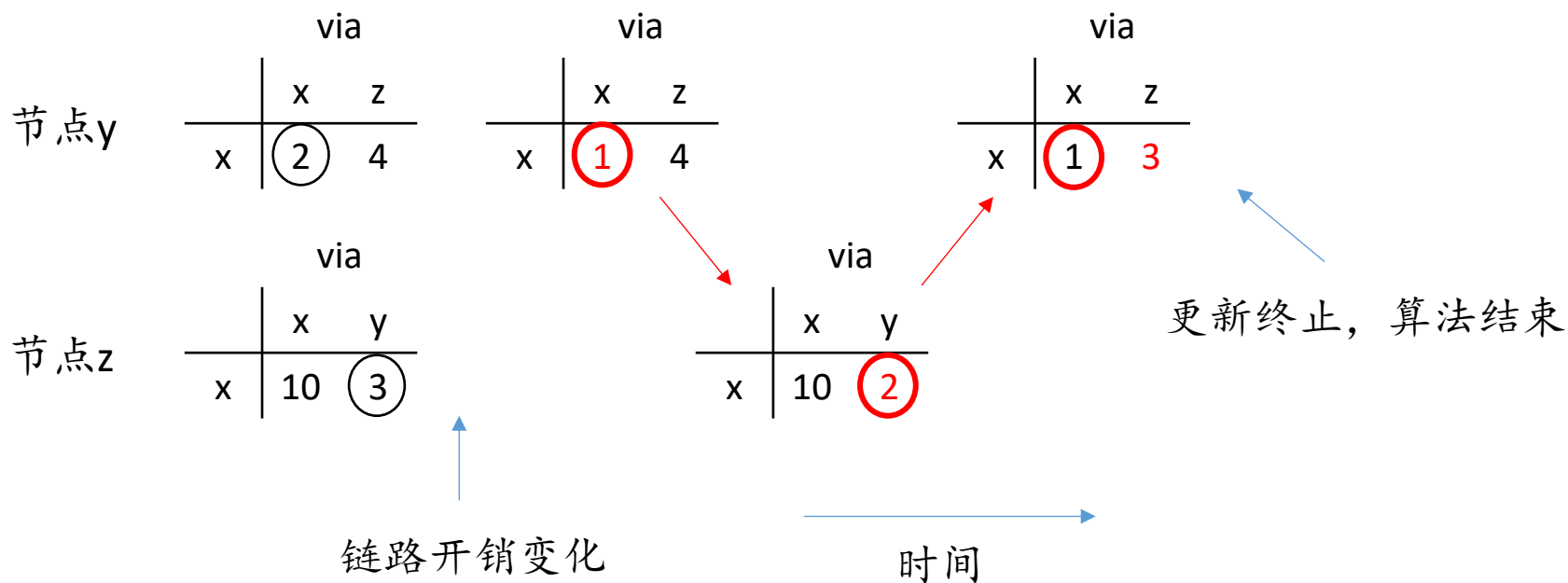
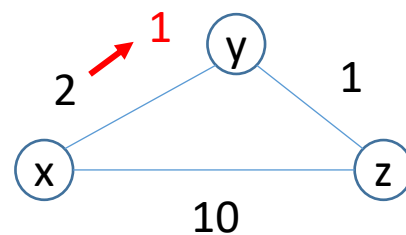
Dest	Cost	Next H
A	7	B
B	4	B
C	0	C
D	6	B
E	5	B
F	7	B

节点E

Dest	Cost	Next H
A	1	A
B	1	B
C	5	B
D	1	D
E	0	E
F	2	F

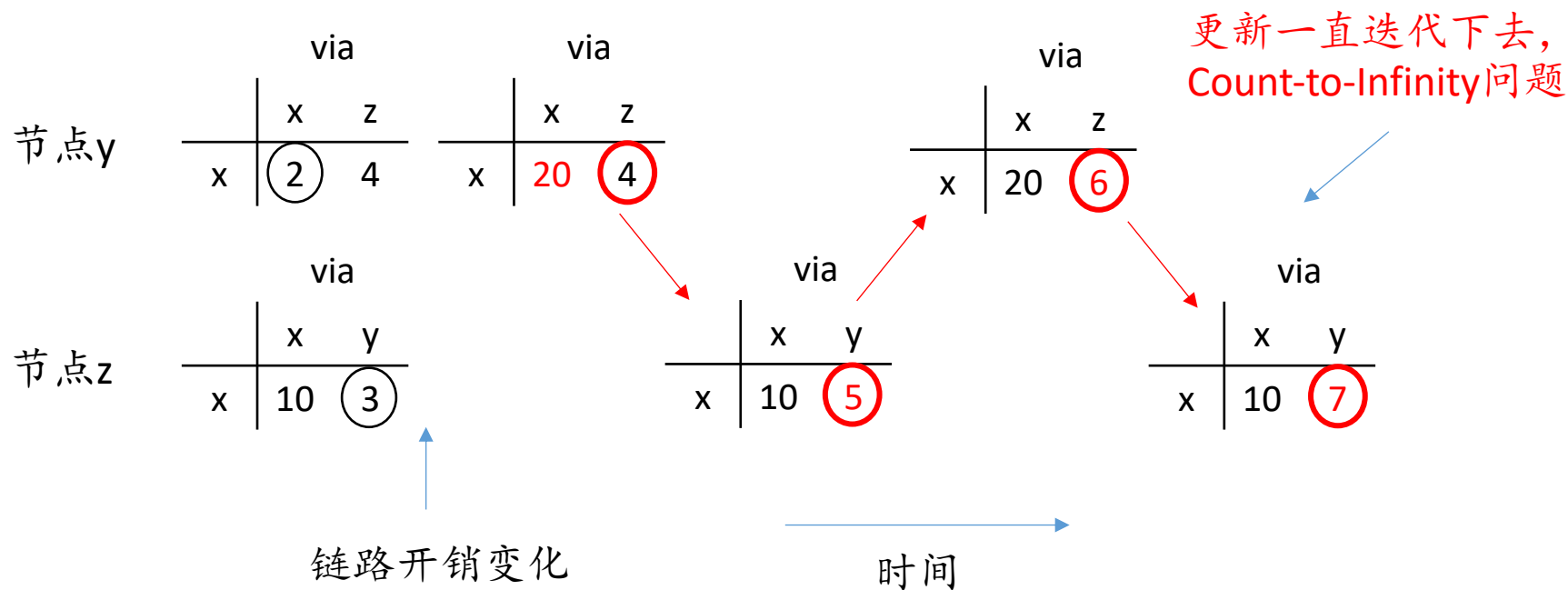
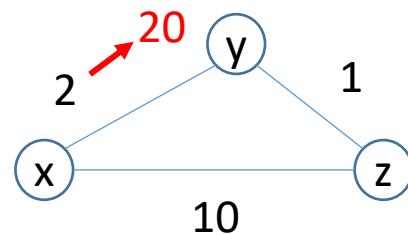
距离向量 – 网络拓扑变化

- 节点检测到相邻链路开销发生变化
 - 更新本地的距离映射关系，通知邻居节点



距离向量 – 网络拓扑变化

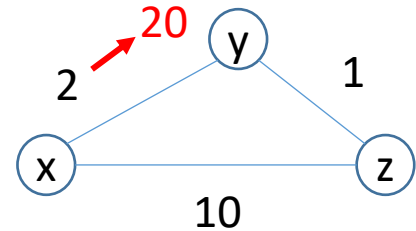
- 当链路开销增大时
 - 节点间的路由更新会相互触发
 - 更新一直迭代下去



消除Count-to-Infinity问题

- Observation:

- 节点z通过y连接到x，且z向y通告该路径
 1. 该通告对路由没有任何意义
 2. 最终产生环路

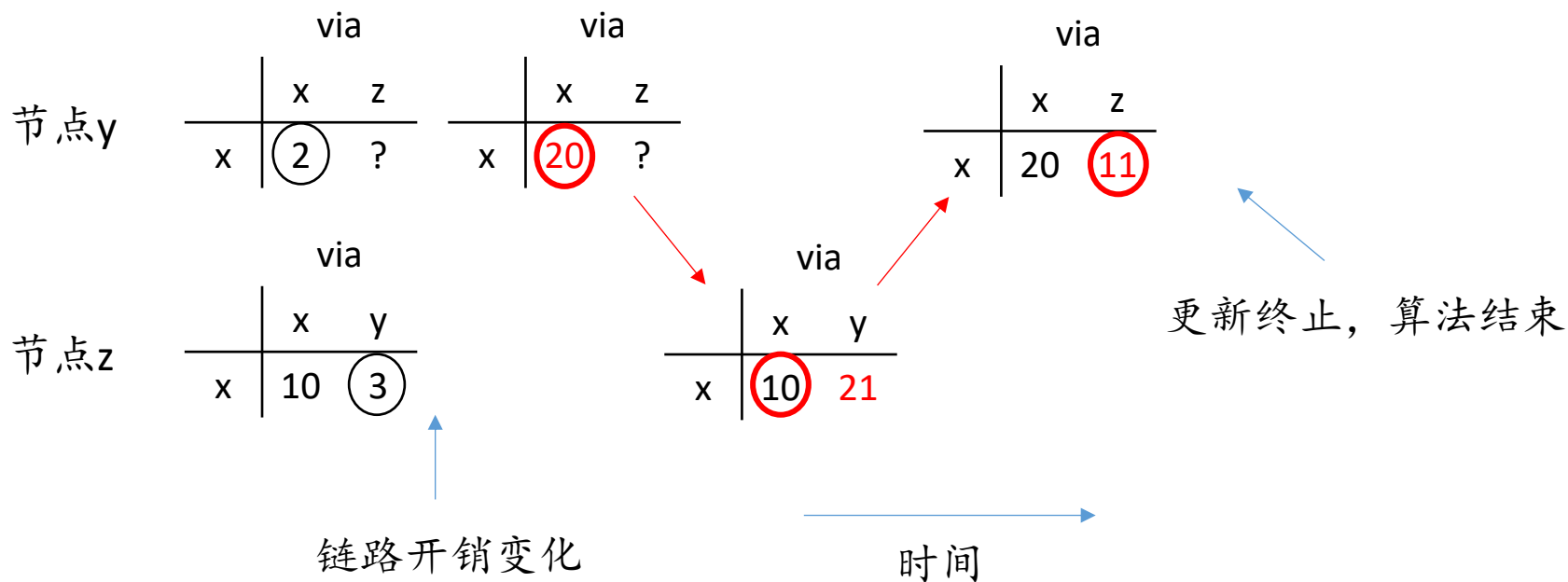
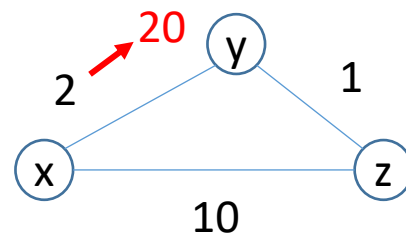


- 解决方案:

- 水平分割 (Split Horizon)
 - 节点z不向y通告该路径（经过y的路径）
- 反向抑制 (Poison Reverse)
 - 节点z通告y，z到x的距离为无穷大，这样y就不会经由z到x

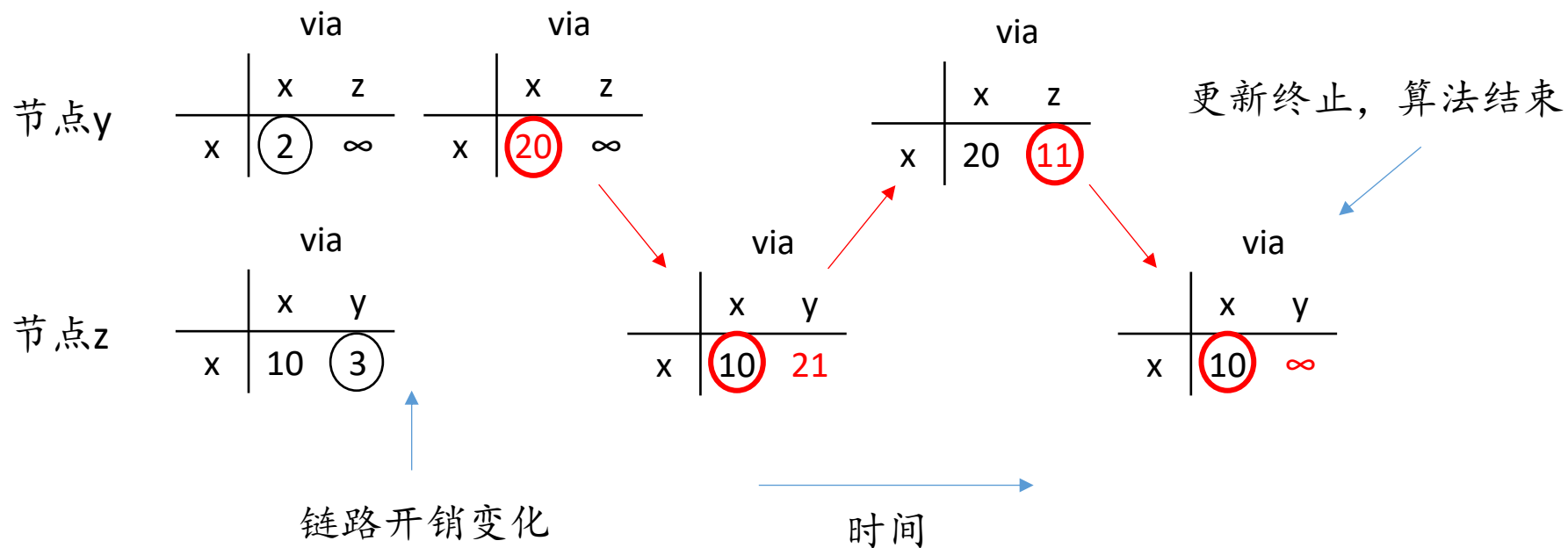
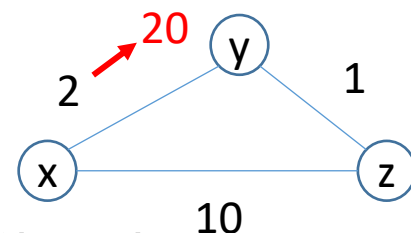
水平分割例子

- 节点z不向y通告其经过y的路由路径
- 节点z收到y的路由更新后，替换掉原来的值
 - 虽然旧的路由开销更小，但过时了



反向抑制例子

- 如果z经由y到达节点x
 - z通告y其到x的距离为 ∞ ，因此y就不会经由z到x
- 反向抑制会立即通告，而不需要像水平分割那样等待超时



RIP (Routing Information Protocol)

- 最早的IP路由协议 (1982, BSD)
- 特点（限制）
 - 每条链路的开销都是1（跳数）
 - 网络直径（最大跳数）小于16
- 路由器有不同类型的更新
 - 初始化：路由器启动时，向邻居节点请求它们的路由表副本
 - 周期性：每30秒，向邻居节点发送路由表副本
 - 触发性：当路由表条目发生变化时，向邻居节点通告

链路状态方法

- 基本思想

1. 每个节点获得整个网络的完整拓扑
 - 各节点广播扩散其邻接关系
 - 通常使用可靠洪泛 (Reliable Flooding) 机制
2. 每个节点单独计算到其它节点的最短路径
 - 使用单源最短路径算法 (Dijkstra)
3. 网络拓扑发生变化时，执行上述步骤
 - 连接断开/重连等

获得完整网络拓扑

- 每个节点创建链路状态LSP (Link State Packet), 包含:

- 创建该LSP的节点标识

用于路由计算

- 该节点的相邻节点列表和对应链路开销

- 序列号、生存周期

用于可靠洪泛

- 扩散链路状态

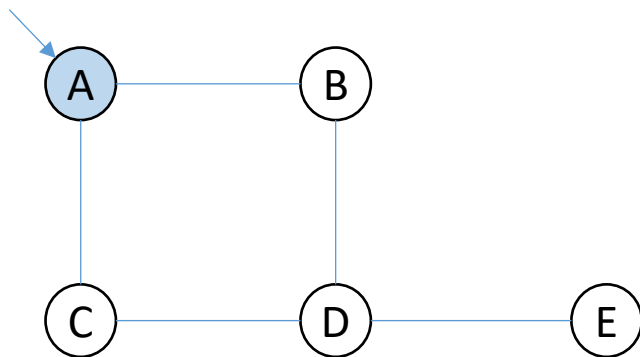
- 节点x收到来自y的LSP副本后:

- 如果之前没有保存对应ID的LSP, 则保存

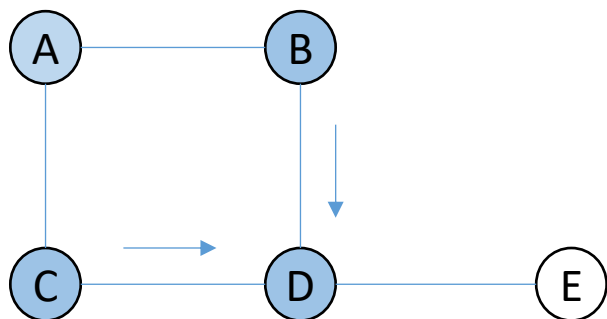
- 如果之前有保存, 新副本的序列号更大, 则更新

- 保存或更新后, 向除y以外的所有节点继续扩散

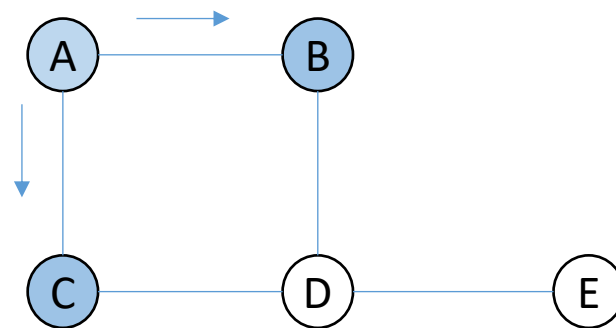
链路状态扩散的例子



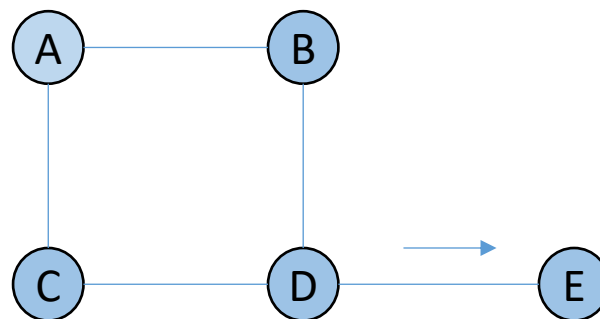
1、某LSP到达A节点



3、B和C扩散LSP到D



2、A扩散LSP到B和C



4、D扩散LSP到E

路由计算

- 使用Dijkstra算法求取源节点到其它节点的最短路径和路由表

```
visited <- [ A ]
unvisited <- nodes - visited

while unvisited not empty:
    x, y <- arg min { d(x,y) where x in visited, y in unvisited }

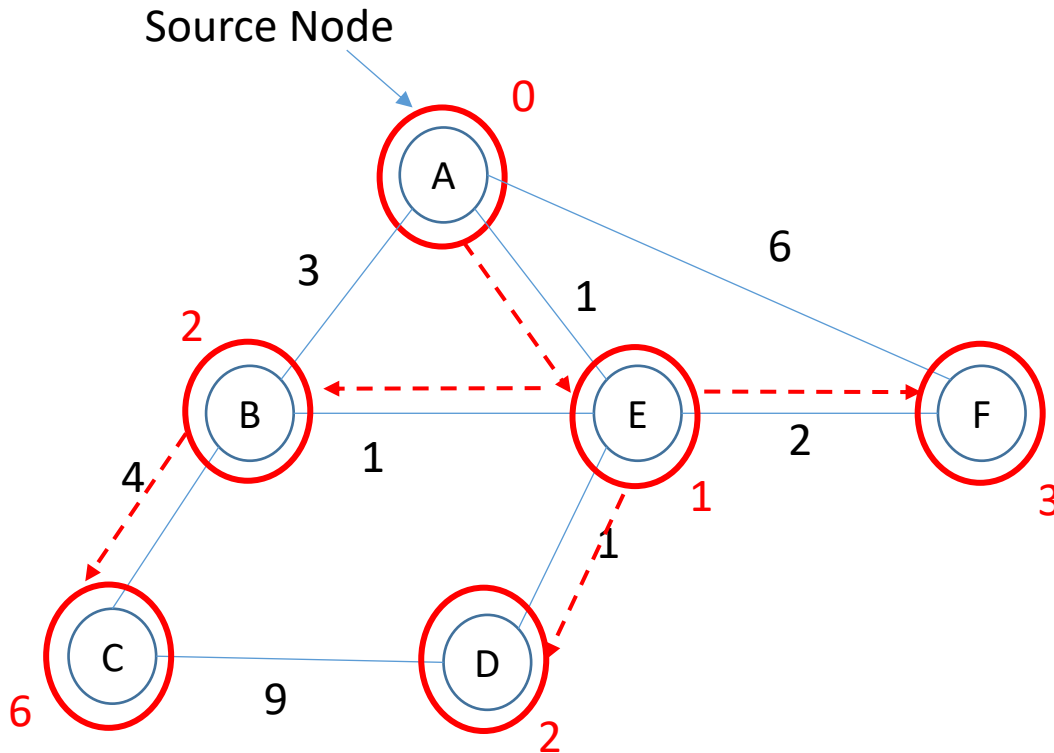
    if y is A's neighbor:
        fib(y) = y
    else:
        fib(y) = fib(x)
    d(A,y) <- d(A,x) + d(x,y)

    visited.add(y)
    unvisited.remove(y)
```

在未访问的节点中，选取离已访问节点最近的那个

生成到该节点的路由表项

路由计算的例子

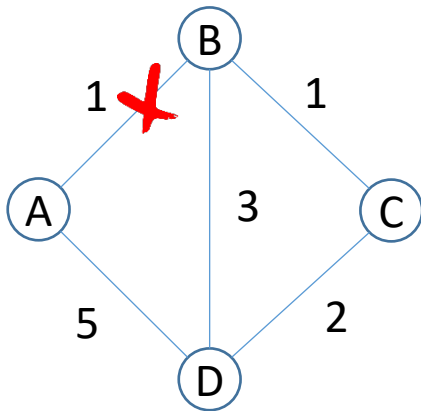


Dest	Cost	Next Hop
A	0	A
B	2	E
C	6	E
D	2	E
E	1	E
F	3	E

- 最终生成的路由路径组成一颗最短路径树(shortest path tree)

链路状态的环路

- 基于一致性链路状态数据库(Link State Data Base, LSDB)
 - 由可靠洪泛保证一致性
 - 所有节点都可以计算出无环路径
- 但可能存在瞬时环路(transient loops)
 - 当网络拓扑发生变化时



当链路A-B断开时

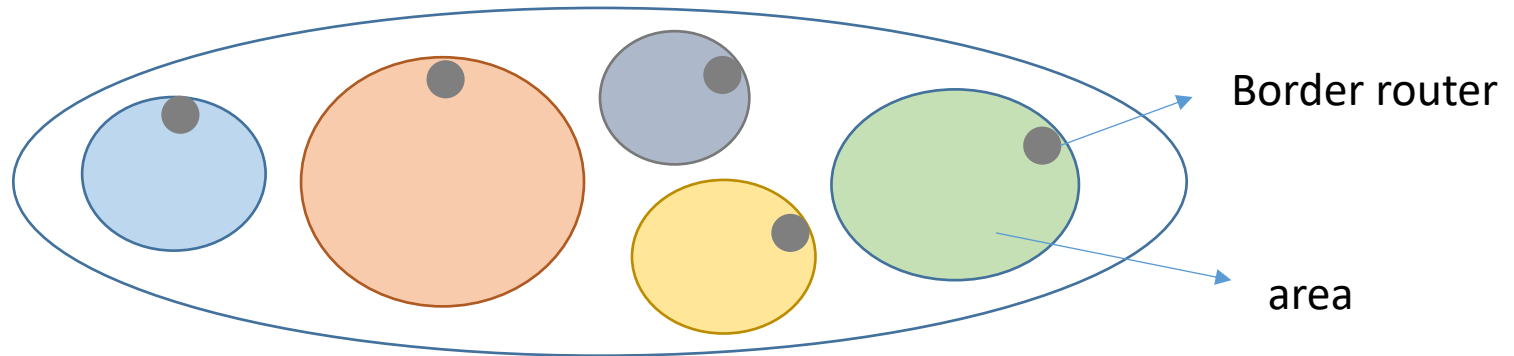
- B首先感知到，但C和D还不知道
- C到A的数据传输会形成环路：C-B-D-C

开放最短路径优先协议 (Open Shortest Path First, OSPF)

- **Open**: 由IETF制定的开放标准
 - 链路度量可自定义, 表示费用、距离、时延、带宽等
- **Shortest Path First**
 - 也就是Dijkstra算法
- **可靠洪泛 (Reliable Flooding)**
 - 向邻居节点发送链路状态通告 (Link State Advertisement, LSA)
 - 周期性发送、当链路发生变化时
- **基本上已经取代RIP**
 - 收敛速度更快
 - 使用区域(area)概念, 可支撑更大规模的网络路由

更大规模网络的路由

- 将网络分割成若干独立的区域 (areas)
 - 区域内：每个节点计算到其它所有节点的路由路径
 - 区域外：节点只知道到其他区域的路由路径
 - 跨区域的数据包首先会路由到最近的边界路由器节点(border router)
 - 区域可以进一步划分成若干子区域
 - 两点之间的路由路径可能不再是最短路径

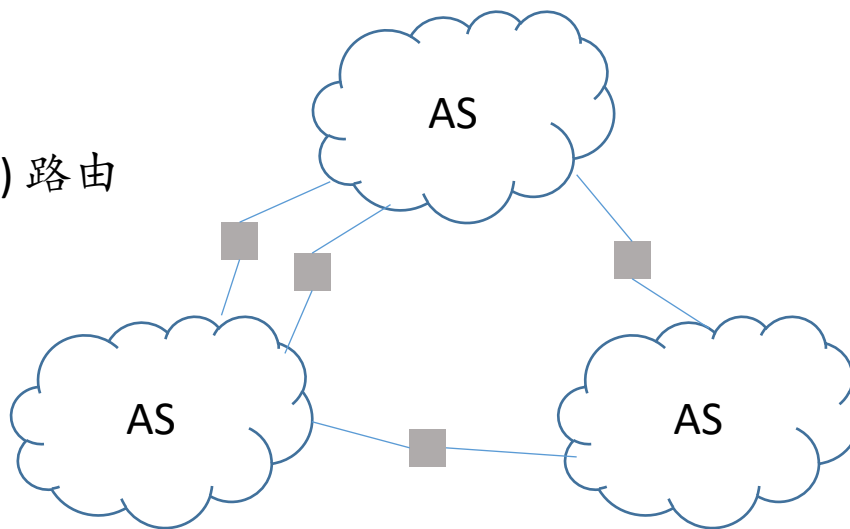


距离向量方法与链路状态方法比较

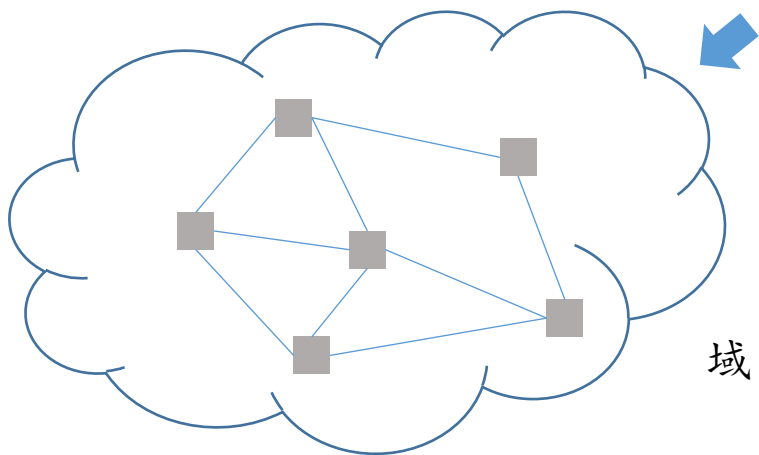
	距离向量方法	链路状态方法
消息数目	相邻节点间交换信息	所有节点间都需要交换信息
空间开销	只保存邻居距离信息	需要保存全网拓扑信息
收敛速度	较慢	较快
可扩展性	较差	较好
路由环路	可能会出现环路，Count-to-Infinity问题	可能会出现短暂环路
路由协议	RIP、BGP	OSPF、IS-IS

域内路由与域间路由

域间(Inter-Domain) 路由



域内(Intra-Domain) 路由

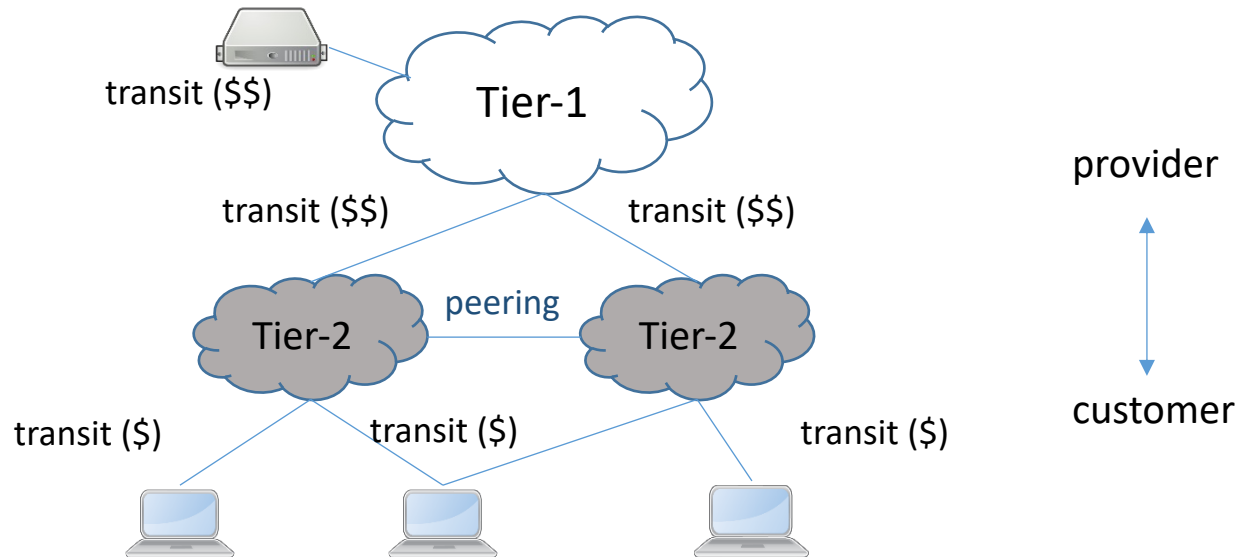


什么是自治系统 (Autonomous System, AS)?

- 自治系统：
 - 由单一实体管辖的路由器集合组成
 - 使用内部网关协议(Interior Gateway Protocol, IGP)和统一的度量标准在AS内路由数据包
 - 使用外部网关协议(Exterior Gateway Protocol, EGP)将数据包路由到其它AS
- 每个AS都有唯一的标识
 - 范围：1-65535 （现在已扩展到32位）
 - 例如， MIT: 3, China Telecom: 4835, Baidu: 55967

域间路由协议

- 能不能将域内路由协议直接应用到域间路由？
 - 不能
 - 域内路由是性能目标导向的，全网有统一目标
 - 域间路由是策略和经济目标导向的，每个AS有自己的策略



域间路由的挑战

- 域的自治特性
 - 每个AS有自己的路径度量指标
 - 不可能得到全局最优的路由路径
- 可扩展性问题
 - 路由表中应包含每个合法的IP前缀，当前大约100万条
- 域间的信任问题
 - 一个AS的错误路由配置可能导致下游AS的传输故障

域间路由设计选择

- 距离向量 or 链路状态？
 - 没有全网统一的度量指标，每个AS有自己的路由策略(Policy)
- 使用距离向量方法？
 - Bellman-Ford算法收敛速度较慢
 - Count-to-Infinity问题
- 使用链路状态方法？
 - 每个系统使用的度量指标不一致 – 可能会导致环路
 - 链路状态数据库太大
 - 会暴露自己的路由策略给其它AS

BGP (Border Gateway Protocol, 边界网关协议)

- 路径向量（带路径的距离向量）

- 每个路由更新中携带整条路径信息

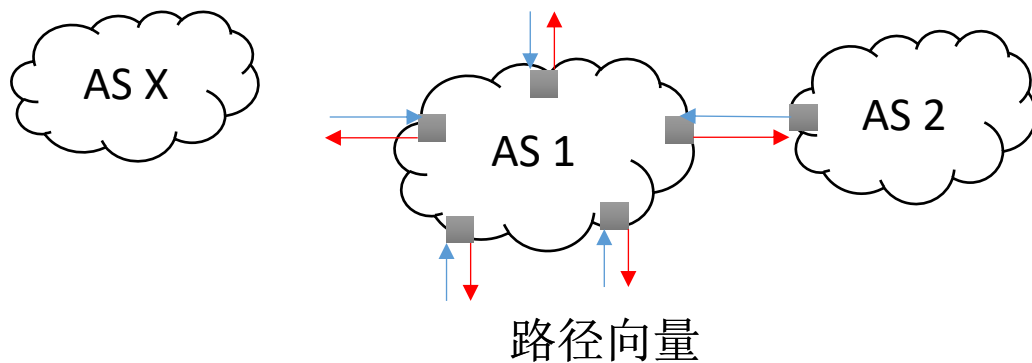
- 检查环路：

- 当AS收到路由更新消息后，检查其是否在对路径中
 - 如果在，存在环路，丢弃该更新消息
 - 如果不在，将自己添加至路径中，通告该路由更新

- 优点：

- 度量指标不需要全局统一
 - AS选择路径，协议检查避免可能的环路

路由路径通告



1. AS 1收到至AS X的路由通告消息后

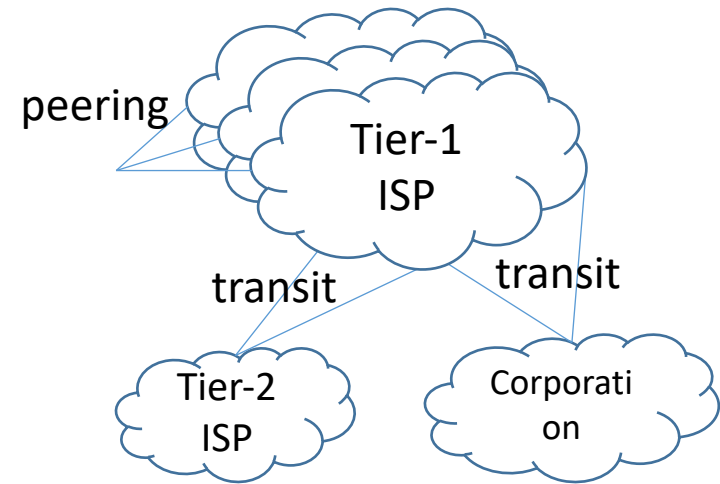
- 根据本地(AS 1的)策略, 选择到X的路径

2. 选择性的通告至AS X的路由

- 根据本地策略, 决定向哪些AS通告该路由更新

BGP策略的例子

- 一个多连接AS拒绝传送中转(transit)数据
 - 限制路径通告，例如大公司
- 一个多连接AS可以为一些AS传送中转数据
 - 只将路径通告给部分AS，例如二级运营商
- 一个AS可以显式的允许或禁止某AS的数据中转经过
 - 通过允许或禁止某些路径通告



路由消息交换

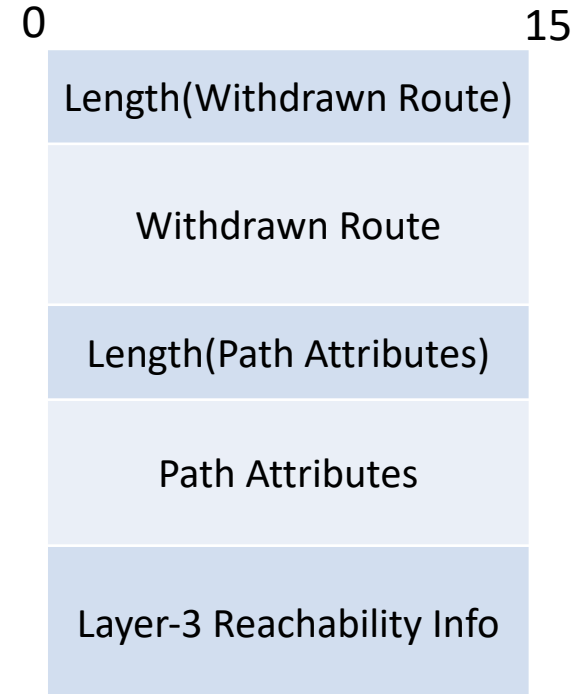
- 域间的对等点使用TCP连接
- 优点：
 - 简化路由协议设计 – 不需要等待对方的消息确认
 - 不需要周期性更新
 - 在撤销或连接断开之前，路由条目一直有效
 - 可以批量更新
- 缺点：
 - 当链路负载很高时，路由协议性能会受到影响

BGP消息

- Open
 - 宣告AS ID
 - 确定定时器时间（keep-alive与update之间的时间间隔）
- Keep-alive
 - 向对等点周期性的发送，保证连接不断开
- Notification
 - 错误消息提示，收到提示后关闭TCP连接
- Update
 - 路由更新消息

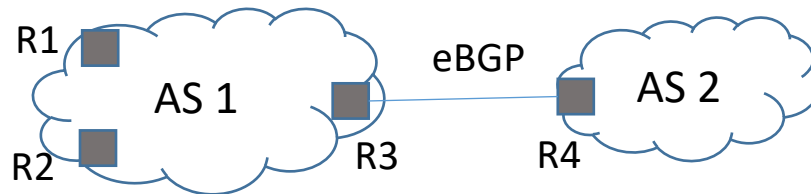
BGP路由更新消息

- 撤销路由(withdrawn route)
 - 当链路出现故障或者策略改变时
- 路径属性
 - 源
 - 路径
 - 度量值，用于路径选择策略
- 网络层可达信息
 - 可达IP前缀列表
- 在同一消息中通告的所有IP前缀拥有相同的路径属性
 - 来自同一AS



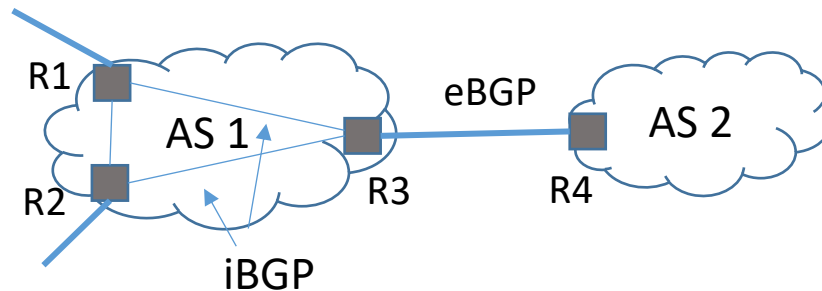
内部BGP(internal BGP, iBGP)

- 图中的节点R3和R4可以通过BGP(eBGP)学习路由
- 节点R1和R2如何学习路由？
- 边界网关路由器R1与R2也需要运行内部路由协议
 - 与AS内其他边界路由器建立连接，学习路由
- iBGP：使用与eBGP相同的消息格式



iBGP路由通告

- iBGP进行路由通告时：
 - 从eBGP中学习得到的IP前缀可以向邻居节点通告
 - 从iBGP邻居节点学习到的IP前缀不能再通告给其他节点
- 原因：
 - iBGP的AS路径中只包含该AS，可能会引起路由环路



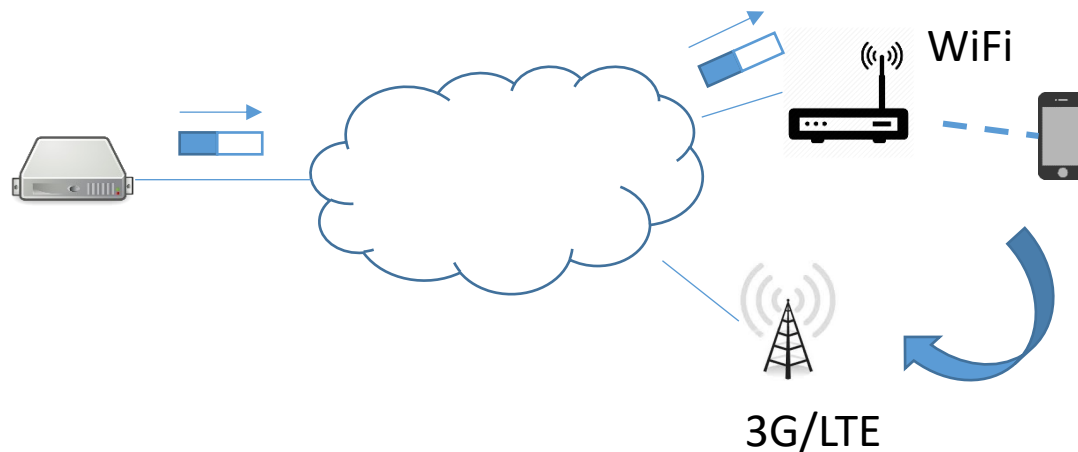
BGP总结

- 广域互联网的路由受经济因素影响
 - 提供商、客户、对等关系
- BGP可以提供：
 - 分层的路由机制，有一定可扩展性
 - 策略决定路由
- BGP的路径向量机制 (带路径的距离向量)
 - 可扩展性好
 - 避免暴露自己的路由策略给其它AS
 - 快速检测环路

路由到移动主机

- 移动网络的挑战

- 主机从一个接入网络移动至另一个接入网络
- 移动主机获得新的IP地址
- 连接另一端不知道主机的移动，仍将数据发送到旧的IP地址



IP路由与主机移动

- IP协议假设主机位于固定的网络位置
 - 当主机移动后，会获得相应网络下新的IP地址
- IP地址确保IP路由算法将数据包传送到正确的网络位置
 - IP地址分为网络部分和主机部分
 - 即使是DHCP，也假设IP地址位于固定网络位置
- 移动主机用户的需求
 - 不需要感知在不同网络间切换
 - 在数据传输过程中不希望更换IP地址

尝试解决移动主机问题

- 通告新IP地址

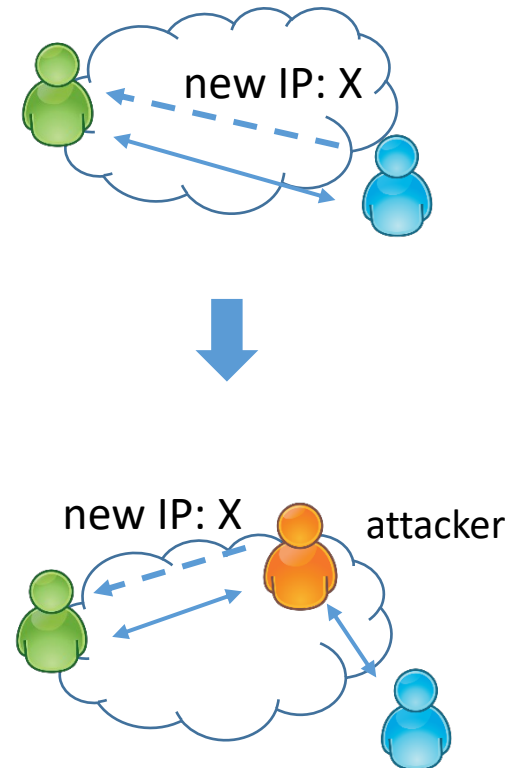
- 主机移动后，将新的IP地址通告给对端
- 对端将数据发往新的IP地址

- 安全性问题

- 攻击者可能会冒充移动主机发送移动通告
- 进行中间人攻击

- 反思

- IP地址的两个职责
 - 标识符(identifier)和定位符(locator)

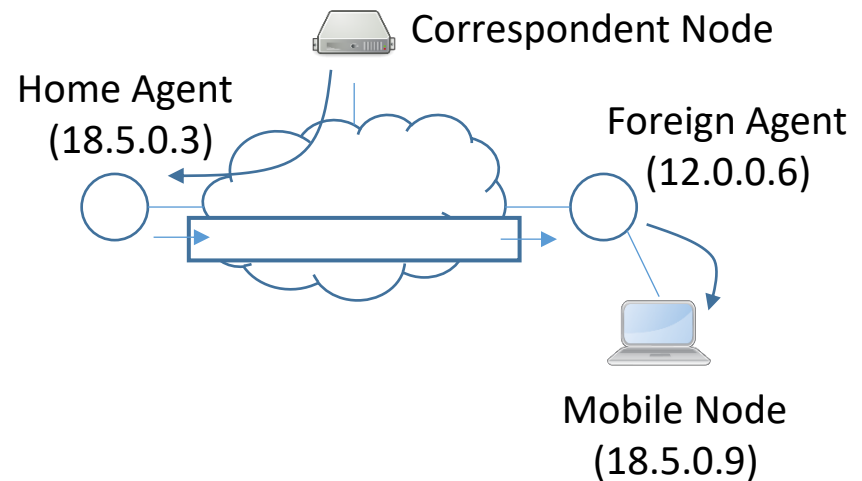


Mobile IP技术思路

- 假设移动主机有一个永久的IP地址
 - 称为本地地址(home address), 作为identifier
 - 与移动前的网络拥有相同前缀
- 主机移动到新的网络时
 - 获得新的IP地址, 作为locator
 - 两个地址可以共存
- locator负责接收数据, identifier负责解复用数据

Mobile IP方案

- 移动主机 (Mobile Node)
 - 分配一个永久的IP地址，本地地址
- 本地代理 (Home Agent)
 - 位于移动主机的本地网络
 - 维护本地地址到转交地址的映射
- 外地代理 (Foreign Agent)
 - 在本地代理与移动主机之间转发数据
 - 周期性的向外通告，提供转交地址
- 转交地址 (Care-of-address)
 - 标识移动主机的位置，通常是外地代理的地址
 - 当移动主机加入时，由外地代理提供给本地代理
- 对端主机 (Correspondent Node)



如何将数据发送到移动主机

1. 本地代理截取目标为移动主机的数据

- 数据肯定会到达本地代理所在的网络
- 本地代理通过ARP Proxy技术模拟移动主机收取数据

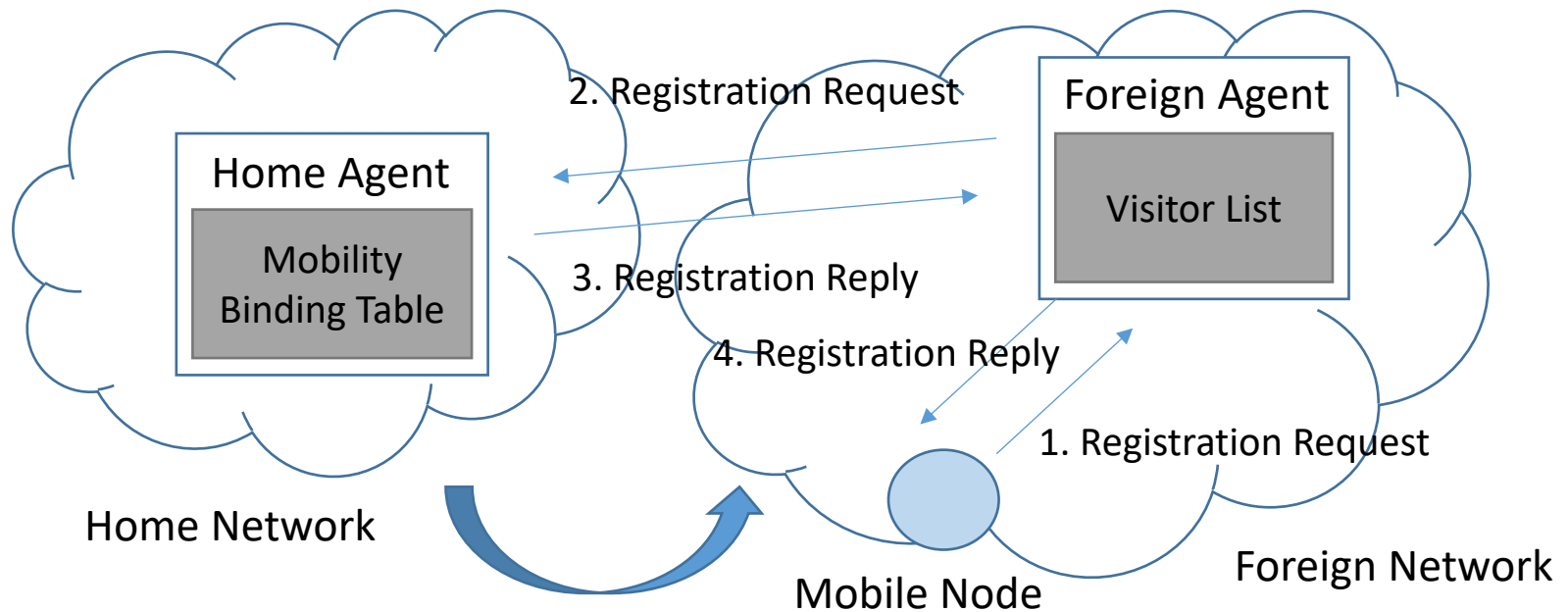
2. 本地代理将数据发送到外地代理

- 本地代理使用转交地址将数据封装，发送给外地代理(IP-in-IP tunnel)
- 外地代理剥离封装的IP头部，得到原始数据

3. 外地代理将数据传送到移动主机

- 外地代理查询原始数据的IP地址，通过Ethernet地址发送给移动主机

移动主机注册过程



移动主机注册信息

- Mobility Binding Table

Home Address	Care-of-address	Lifetime (s)
18.5.0.9	12.0.0.6	120

- 由本地代理维护
- 保存本地地址到转交地址的映射关系

- Visitor List

Home Address	Home Agent	Hardware Address	Lifetime (s)
18.5.0.9	18.5.0.3	00-60-08-95-66-E1	120

- 由外地代理维护
- 保存移动主机的地址信息
 - 本地地址、本地代理地址、Ethernet地址

Mobile IP存在的问题

- 三角路由路径
 - 绕路问题。极端情况下，MN离CN很近，离HA很远，怎么办？
 - 解决方案：让CN知道MN的转交地址
 - 由HA告知CN其MN对应的转交地址
 - CN与FA之间直接IP-in-IP隧道通信
- 单HA节点失效问题
 - 可能的解决方案：多HA机制
- 主机一直移动，频繁向HA注册
 - 可能的解决方案：多个FA协同，形成更大的FA组
- 安全问题等。。。

基于拓扑编址的路由机制

- 基于拓扑编制的路由机制具有较好的可扩展性
 - 编址必须服从网络拓扑结构
- 但是，拓扑不是静止不变的，网络规模在增大，节点可能会移动，因此地址不能永久的标识主机身份
- 问题：
 - IP地址混合两种含义：网络地址+主机标识，当主机移动后，IP地址发生变化
 - 但是，网络应用在连接过程中IP地址不能发生变化，因此产生了移动性问题

扁平化的标识

- 扁平化的名字 (Flat Names)

- 使用DHT（分布式Hash表）替代DNS
 - 例如： www.ict.ac.cn -> ict195610
 - 但是仍需要使用基于层次化地址的路由

- 扁平化的地址 (Flat Address)

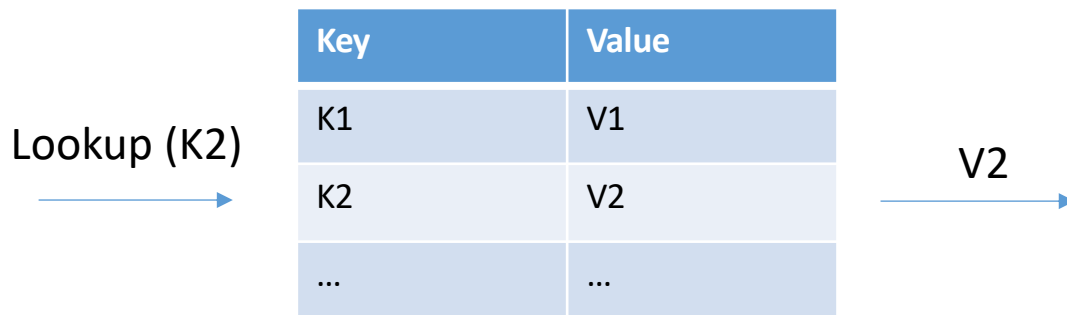
- 避免了将名字翻译成地址
- 基于扁平化地址的路由 (ROFL, Routing on Flat Label)
 - 核心问题： [如何达到可扩展性？](#)

ROFL设计思想

- ROFL设计目标：
 - 基于扁平化标识的可扩展路由
- 设计思路：
 - 机制：基于DHT/Chord路由，维护到后继节点、指取节点的源路由，使得不需要聚合就可以提供可扩展的网络路由
- 本质上：
 - ROFL就是一个全局的扁平标识到节点的映射和查找

Hash表与Hash函数

- 名字 -> 值映射对 (Key-Value Pair)
 - 例如, www.ict.ac.cn/index.html (Key) 与对应网页内容 (Value)
- Hash表
 - 将Key与Value关联起来的一种数据结构



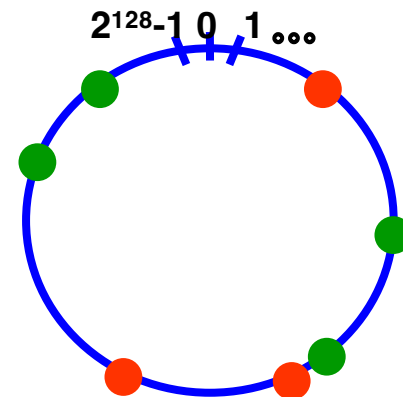
- Hash函数
 - 挑战: Hash碰撞、映射对的加入和离开

一致性Hash (Consistent Hashing)

- 一致性Hash：将一组文件存储到N个节点上，考虑节点的离开和新节点的加入
- 大型稀疏的标识空间 (Identifier Space，例如128位)，构成环形
 - 将文件按名字(Name)均匀的Hash到标识空间，为object_id
 - 将节点(Node)也均匀的Hash到标识空间，为node_id
 - 对于object_id，顺时针遍历，遇到第一个node_id，将文件存储到该节点
 - 节点加入时，后一个节点的相应内容前移；节点离开时，该节点的内容后移

Hash(Name) → object_id

Hash(Node) → node_id



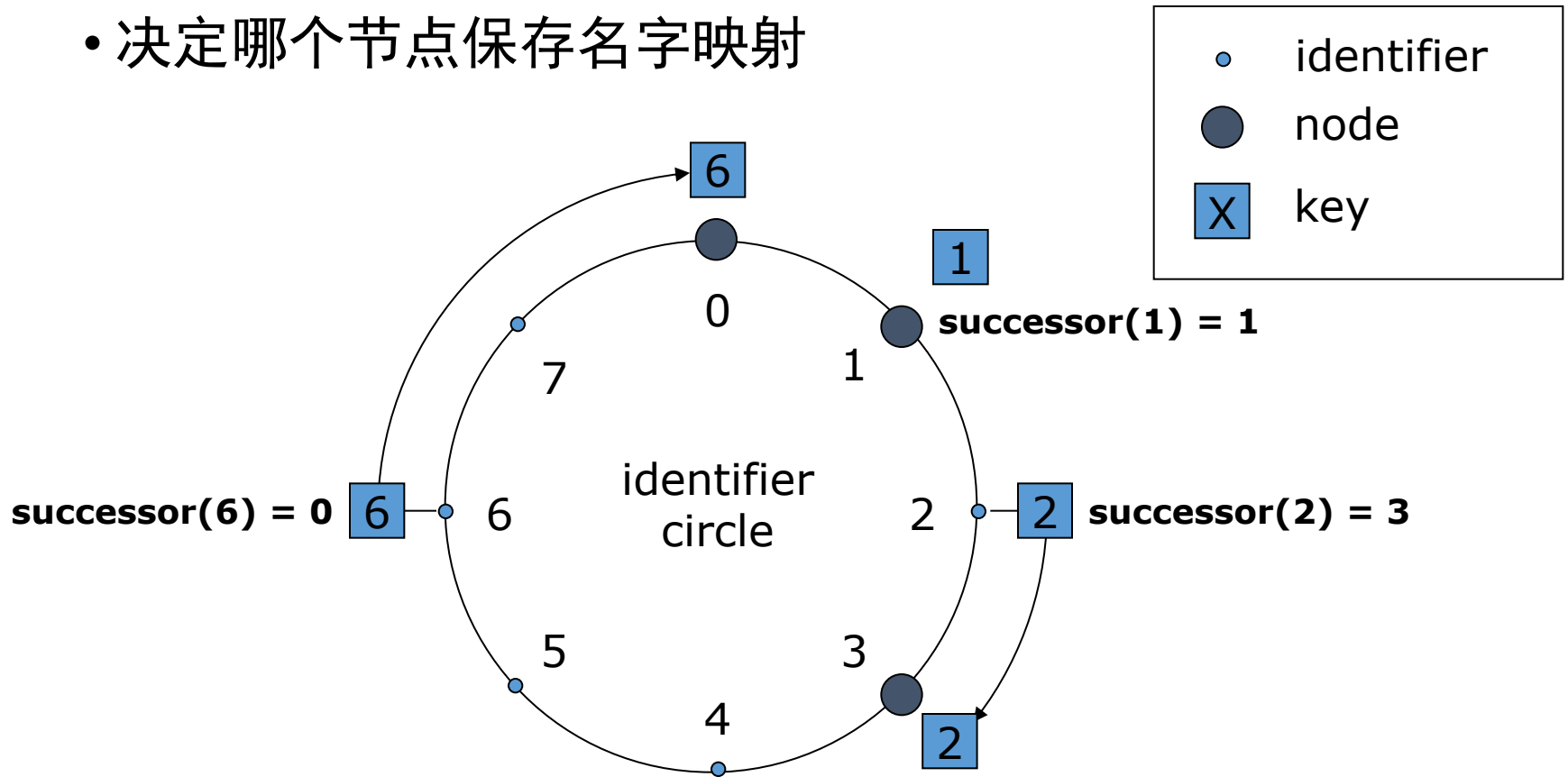
Chord

- Chord 只支持一种操作：给定一个名字，查询对应的存储节点

- 是一种分布式的、去中心化的名字到节点的查询服务
- 考虑到节点的频繁加入和离开，能够快速名字对应的节点
- 应用场景：扁平化路由、P2P文件存储位置定位

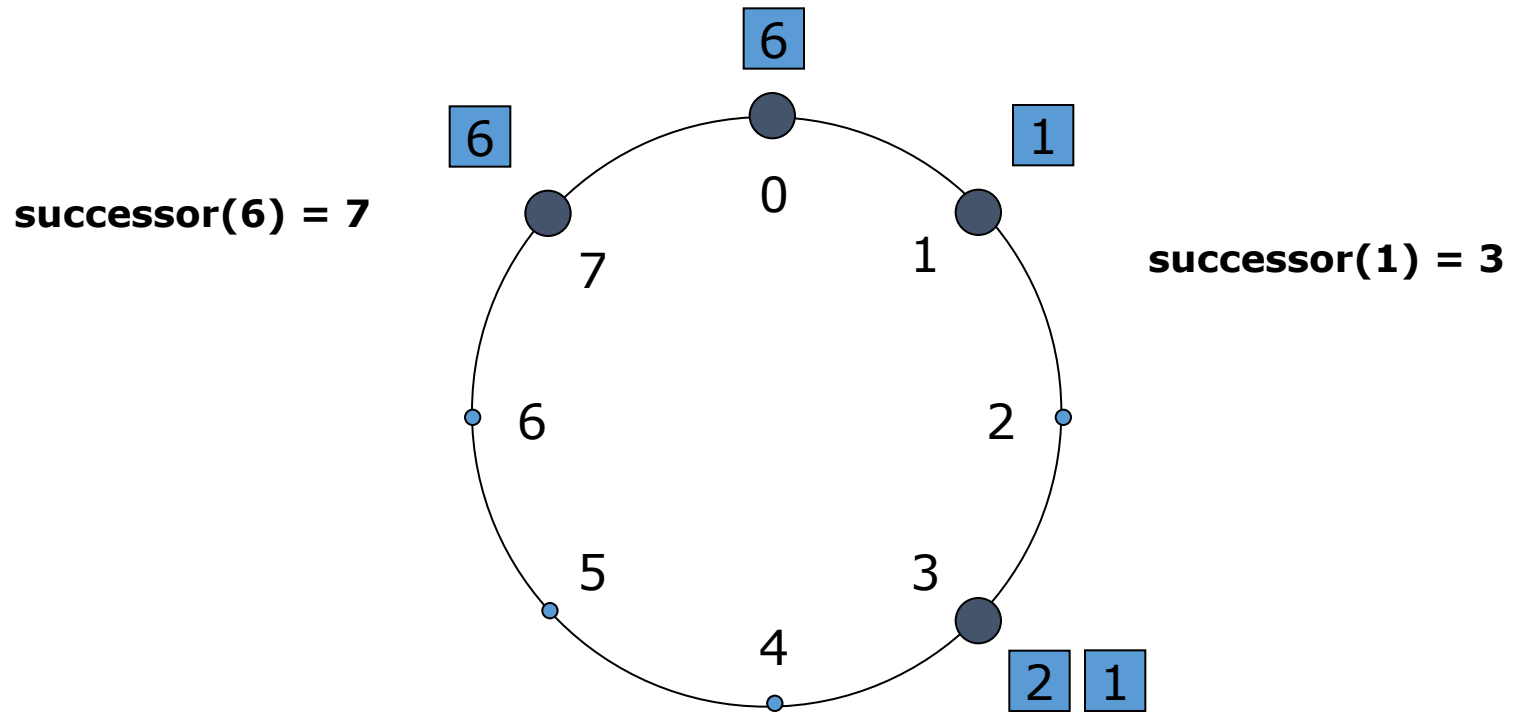
Chord: 后继节点

- 决定哪个节点保存名字映射



Chord: 节点加入和离开

- 节点加入和离开时，系统更新路由状态



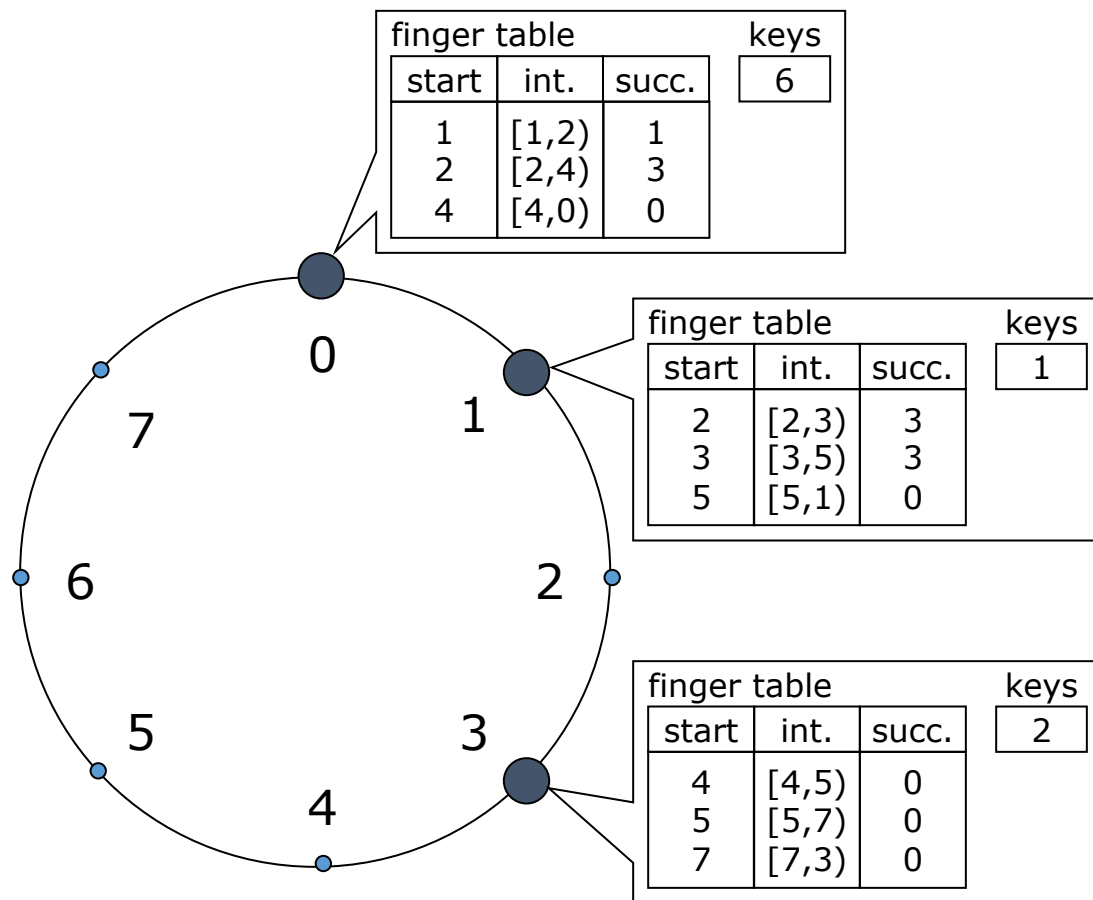
根据Key值查找相应节点

- 所有节点通过后继节点指针串成环形单向链表
- 给定Key值，从任一节点开始依次遍历，查找节点是否存储该值
- 很少量的路由信息就足以在分布式环境下实现一致性Hash
- 这种查找方式只能保证正确，但是无法保证效率
 - 最坏情况下需要遍历所有N个节点

加速查找

- 通过添加额外路由信息来加速查找
- 每个节点维护一个指取表 (Finger Table), 其路由表条目数不超过 m ($m = \log_2(N)$)
- 对于节点 n , 其指取表的第 i 个条目节点 s 的标识, s 为从 n 经由 $2^{(i+1)}$ 次以上后继遍历操作到达的第一个节点
 - $s = \text{successor}(n + 2^{(i+1)})$
 - s 称为节点 n 的第 i 个取指, 记作 $n.\text{finger}(i).\text{node}$

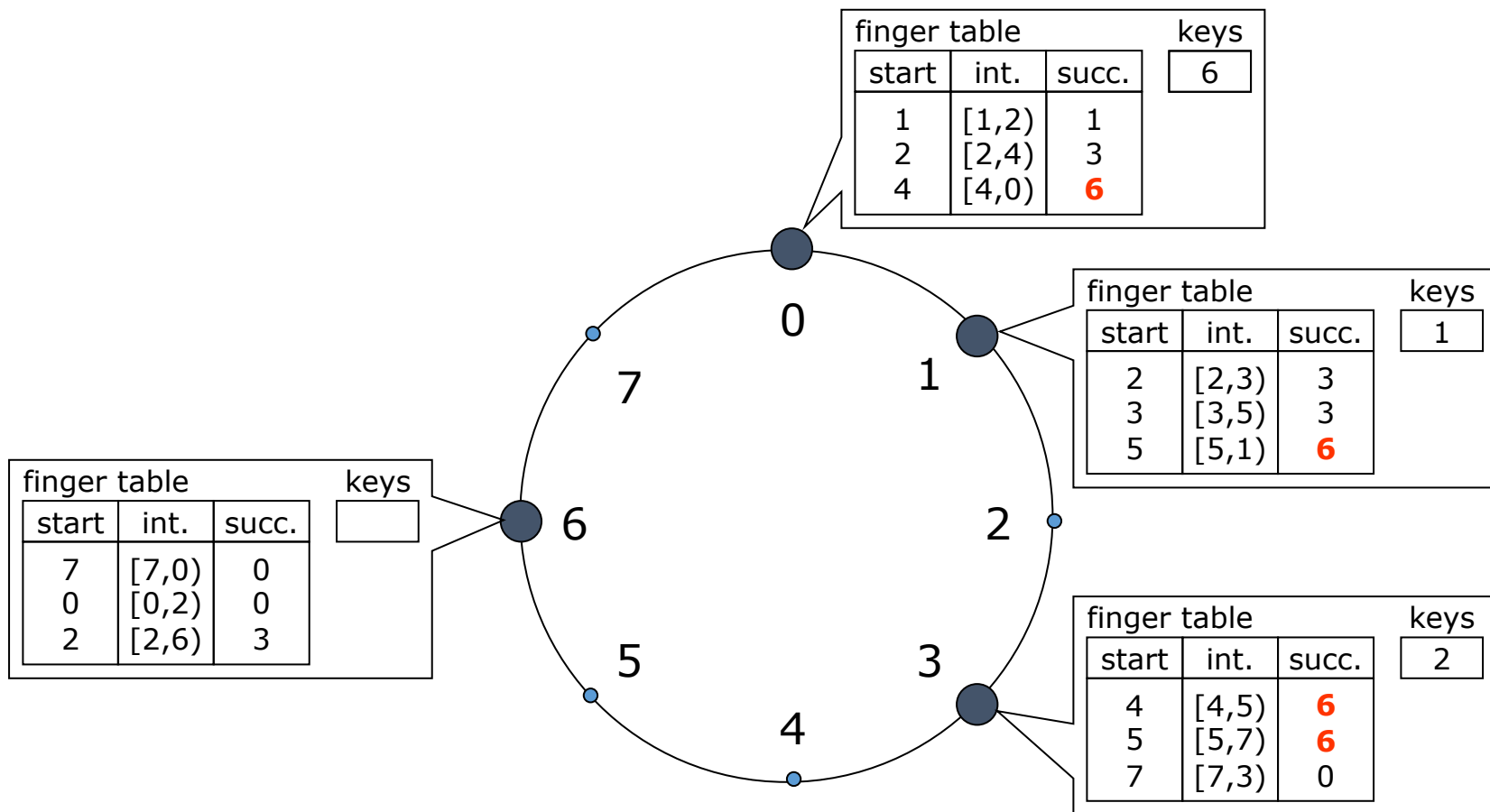
指取表



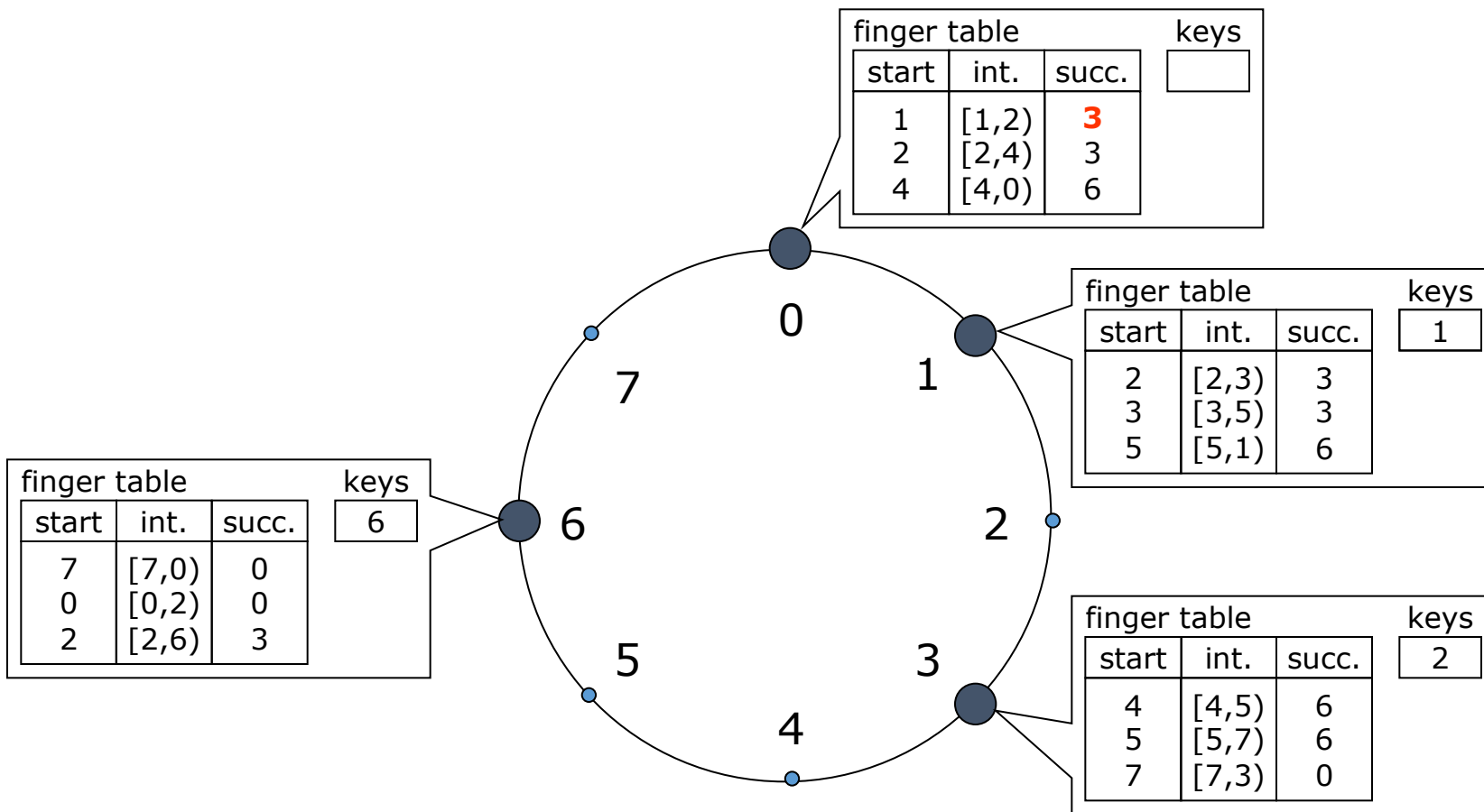
给定Key值查找对应存储节点

- 从任一节点开始遍历，如果该节点中存储相应Key值，查找结束
- 否则，在其指取表中，找到Interval包含对应object_id的区间，跳转到相应后继节点继续查找

节点加入时的指取表



节点离开时的指取表

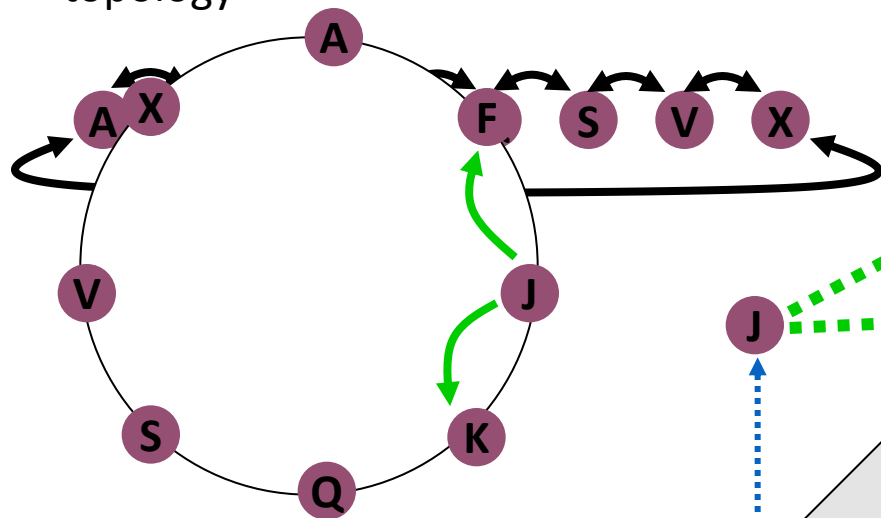


Chord特性

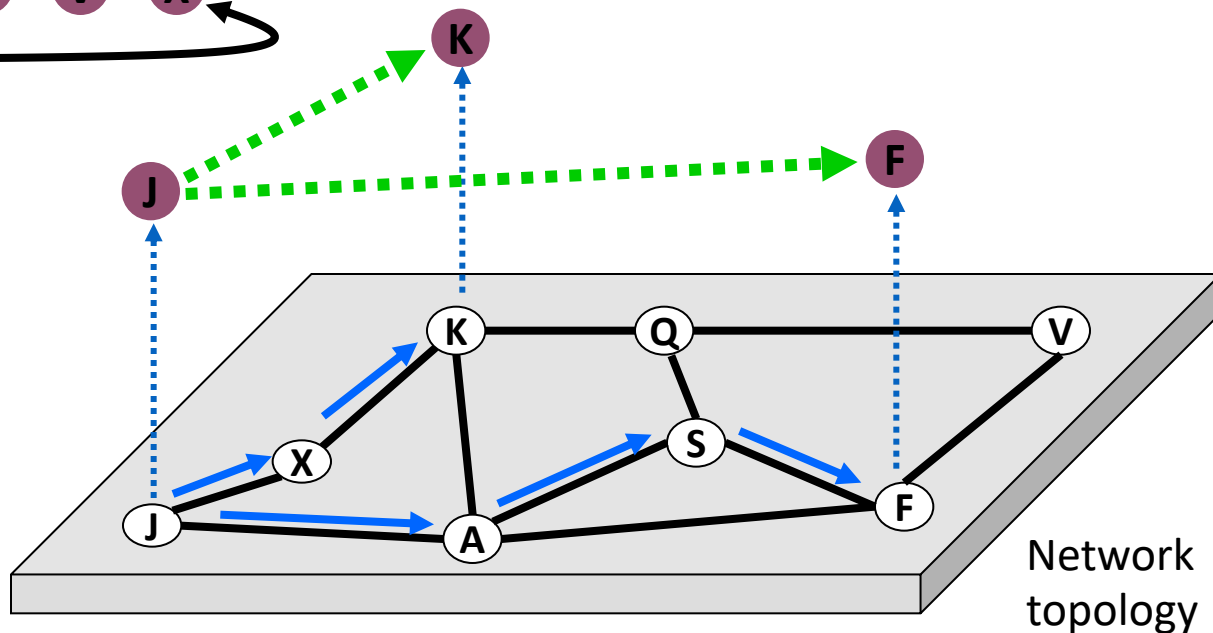
- 简洁性：可证明的**正确性和性能** ($\log_2(n)$)
- 每个节点只需要维护到少数几个节点的路由信息
- 通过向其它节点发送消息迭代/递归的解析查找
- 当节点加入或离开系统时更新路由信息

ROFL: 邻居发现

Virtual topology



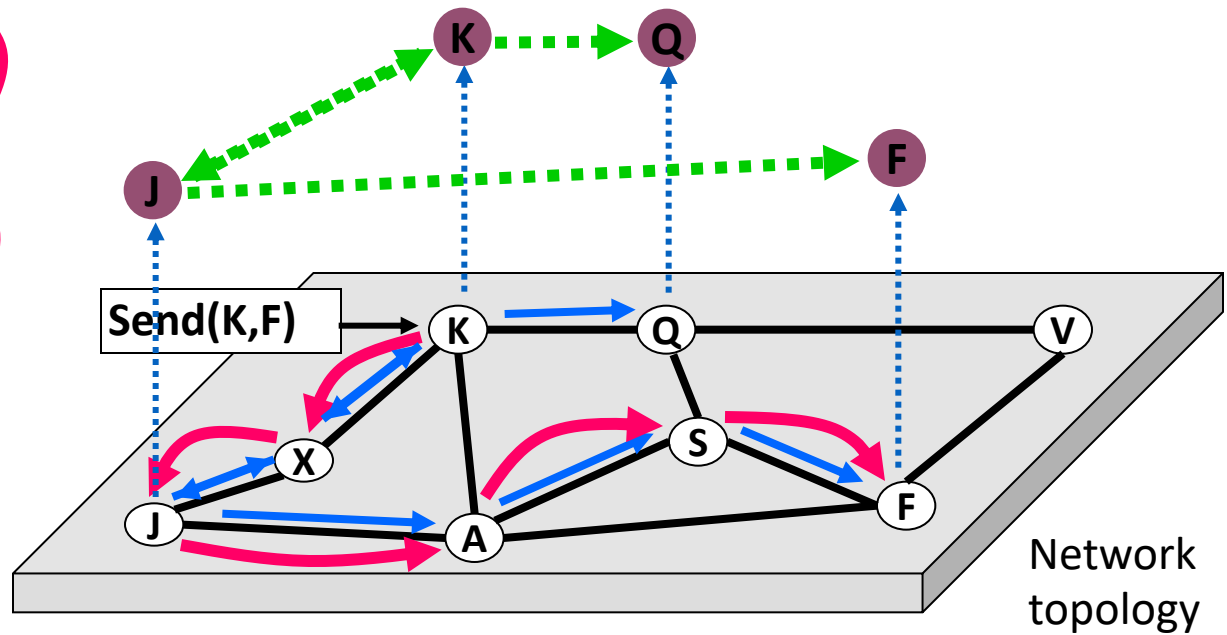
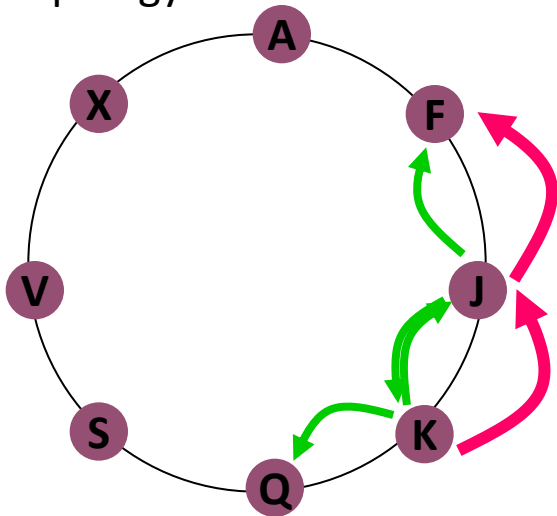
将节点按ID排序，并对相邻节点构建路径



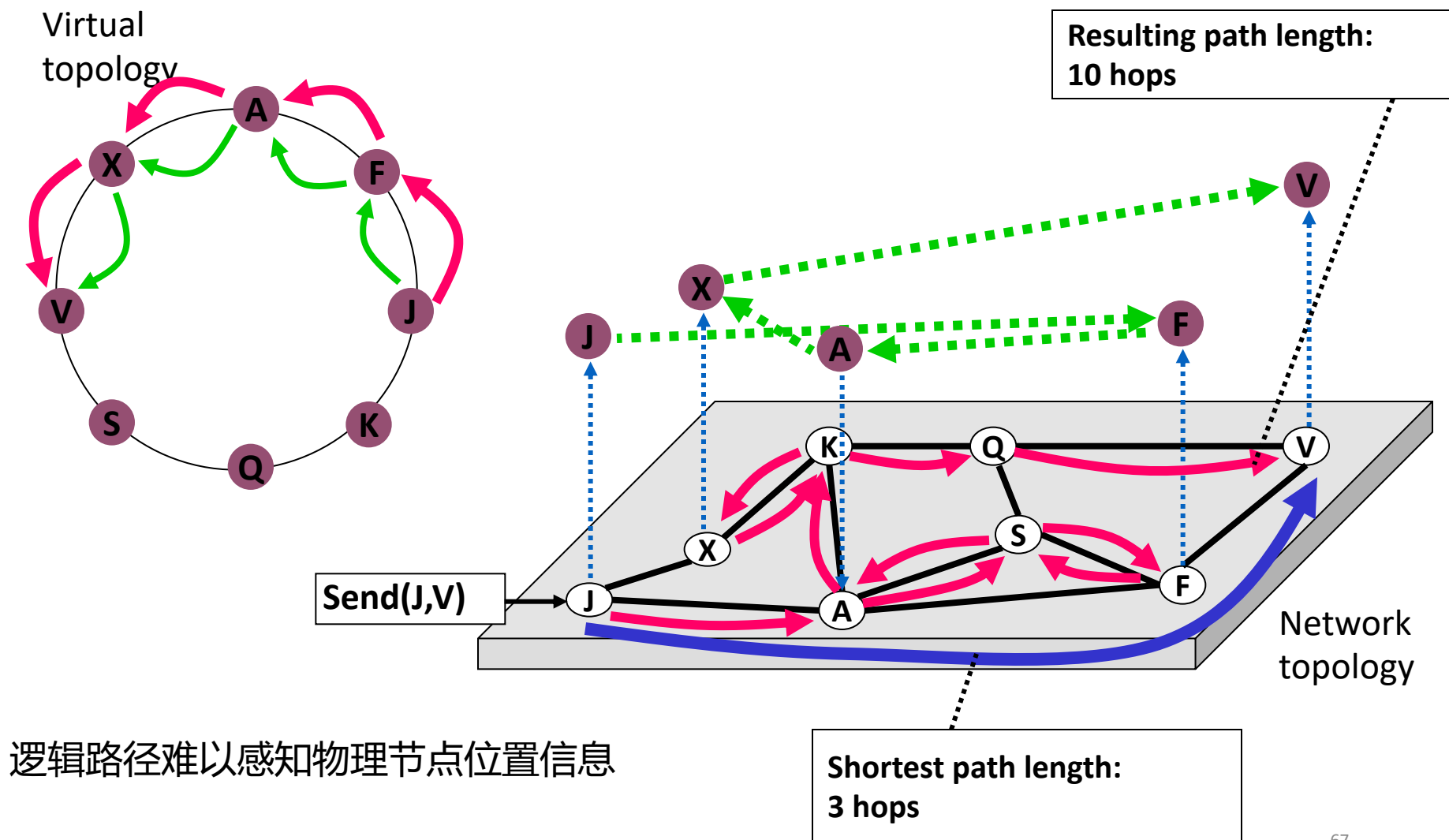
Network topology

ROFL: 数据包转发

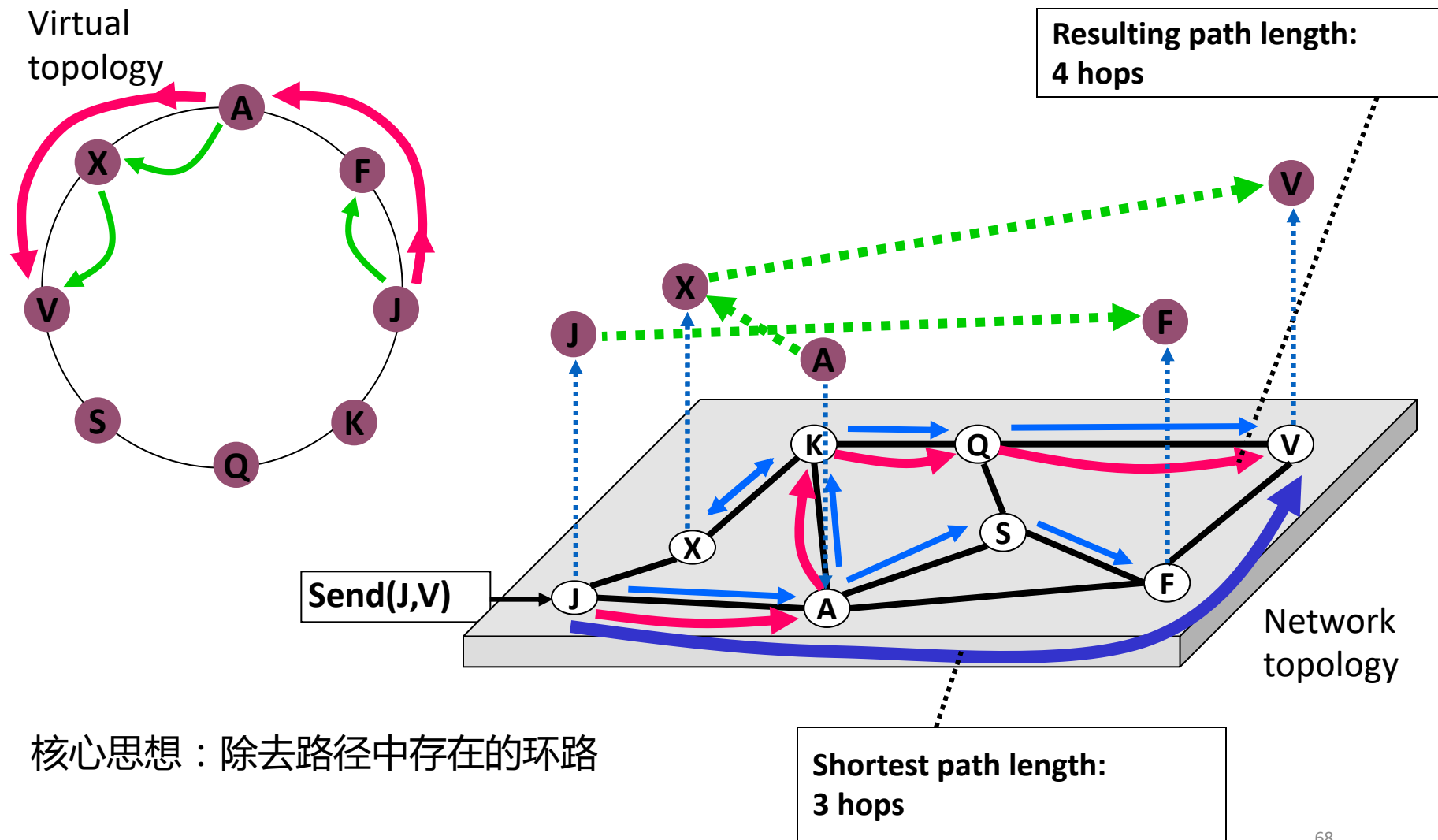
Virtual topology



ROFL: 路径延展问题



ROFL: 寻找捷径



ROFL小结

- ROFL路由基于扁平化的标识，而不是层次化的地址
- ROFL通过利用Chord等技术，实现了可扩展性较好的路由机制
- ROFL在新型网络传输范式中有较大优势
 - 多播、主机移动传输、多宿主传输等
- ROFL需要实现新的标识空间，难以直接替换现有IP架构
 - 一般应用于P2P等Overlay网络中

课后阅读

- 《计算机网络 – 系统方法》
 - 第3.3、4.1、4.4节
- BGP路由配置
 - Ratul Mahajan et al. Understanding BGP Misconfiguration. ACM SIGCOMM 2002
- 主机移动性
 - RFC6830: The Locator/ID Separation Protocol (LISP)
<https://tools.ietf.org/html/rfc6830.html>
- 扁平化路由
 - Ion Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM 2001

Any
Questions?

谢谢！