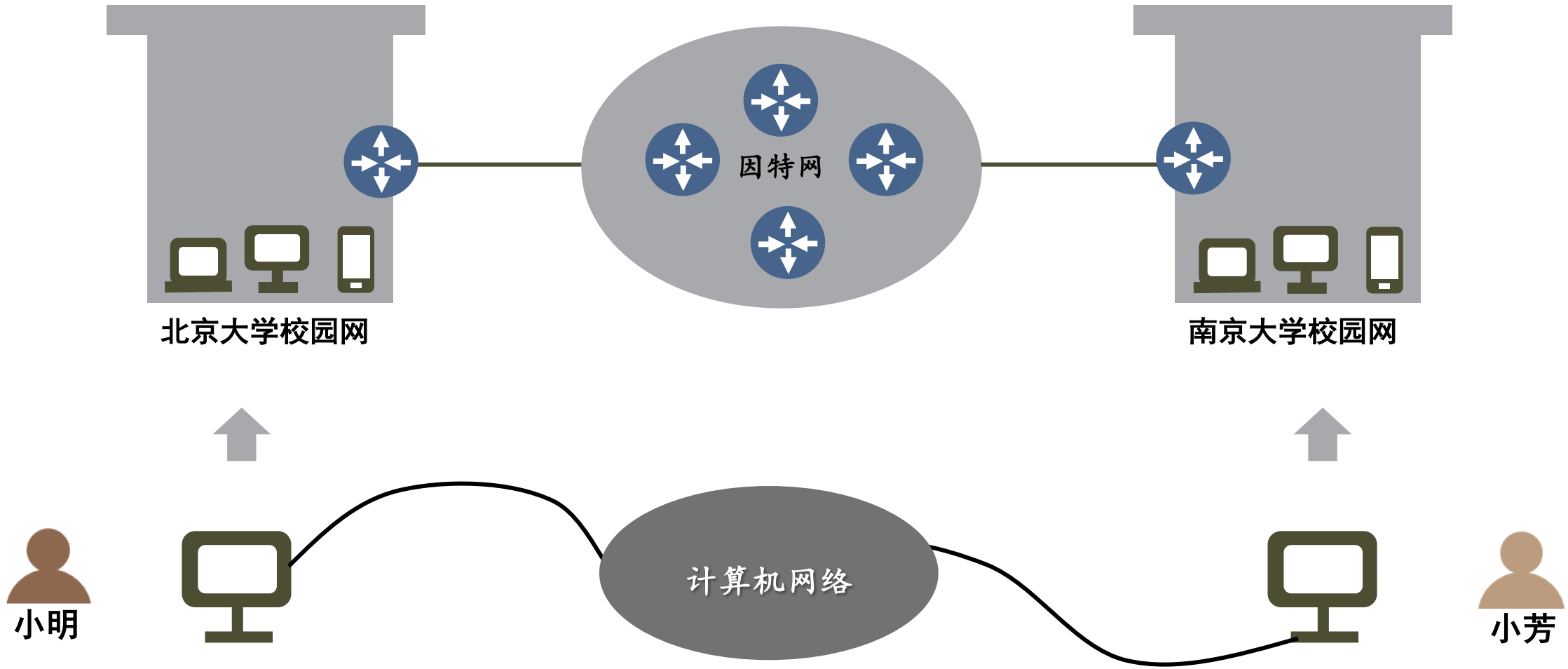
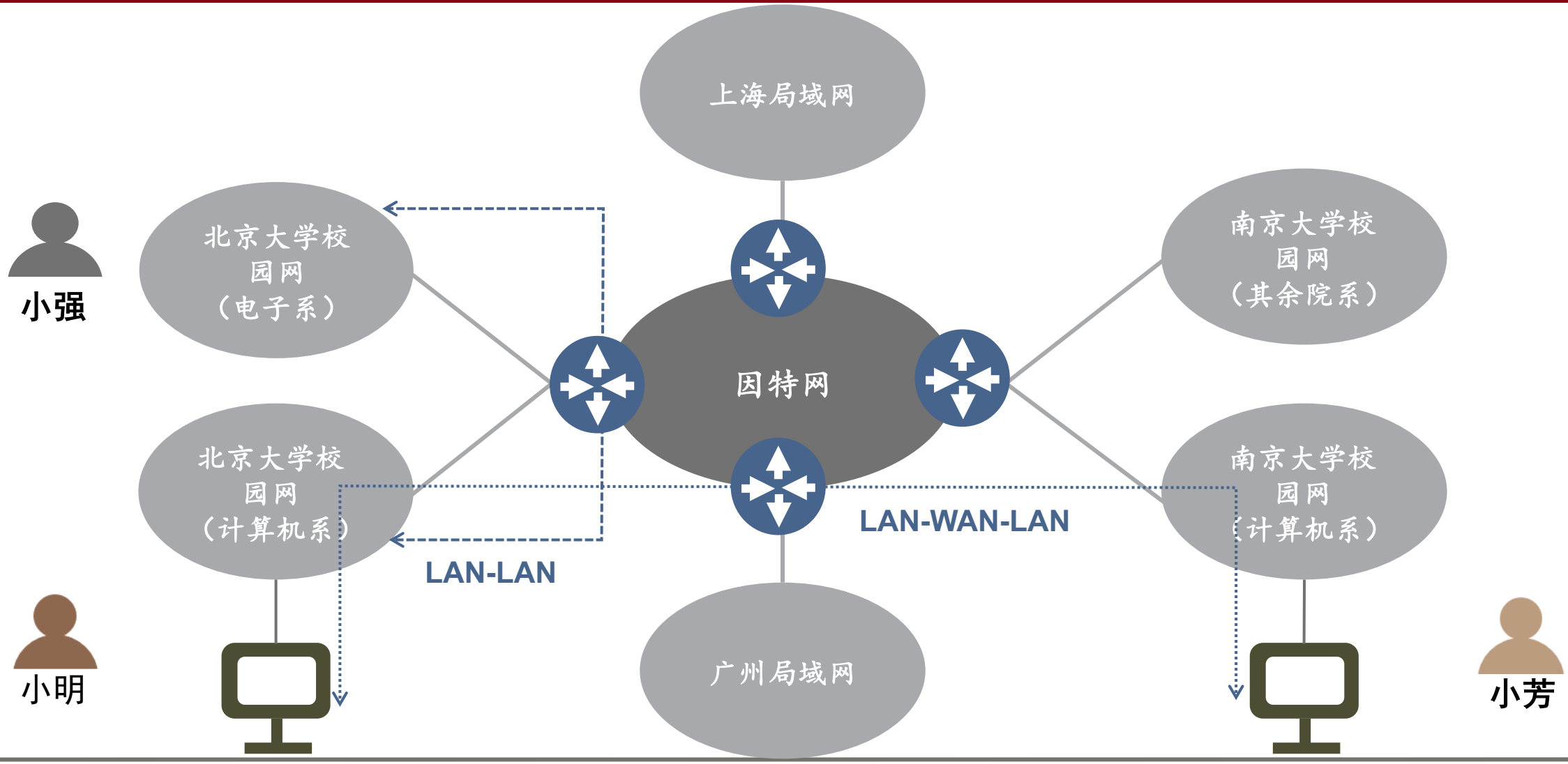
网络层互联需求分析

小明和小芳通信前提



网络一般互联形式

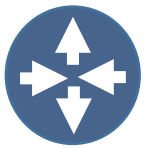

上海局域网
北京大学校园网（电子系）
南京大学校园网（其余院系）
小强
因特网
北京大学校园网（计算机系）
南京大学校园网（计算机系）
LAN-LAN
LAN-WAN-LAN
小明
广州局域网
小芳

网络常用互联设备

第一层 物理层	第二层 数据链路层	第三层 网络层	第四层 传输层	第四层以上 应用层
中继器在两段电缆之间传递单个比特	网桥在两个LAN间存储并转发数据链路帧	多协议路由器在不同网络之间转发数据包	传输网关在传输层连接字节流数据	应用网关允许互连两个采用不同协议的应用

与物理编码方式有关

- LAN通信
- 广播技术



- 远距离通信
- 点-点传输

?

用路由器连接两个网络时将面临什么困难

路由器

由于不可能在整个广域网上实现广播，因此需要进行路由选择。

路由器是一种用来连接两个运行相同/不同协议的通信子网的硬件设备，其工作在网络层。

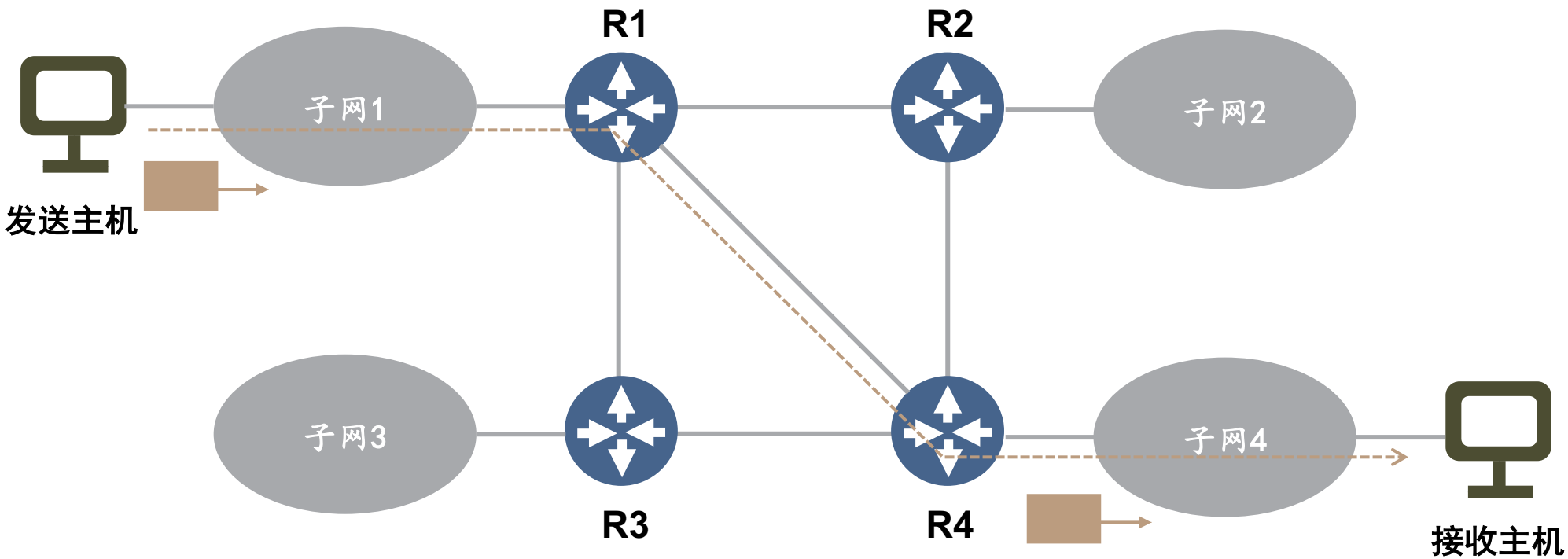
特性

- 一种用于连接两个运行相同/不同协议的中间系统
- 针对网络层地址协议（如IP地址）进行选择与判断
- 需要有二层地址与三层地址的映射能力（地址解析）
- 对相同高层协议提供多个网络的互连服务

路由器的配置

路由器（R1,R2,R3,R4）协同为任意主机之间的通信寻找一条最“好”路径。

- 路由器每个端口连接一个通信子网
- 各通信子网可采用不同协议



路由器与网桥的区别

网桥（帧）

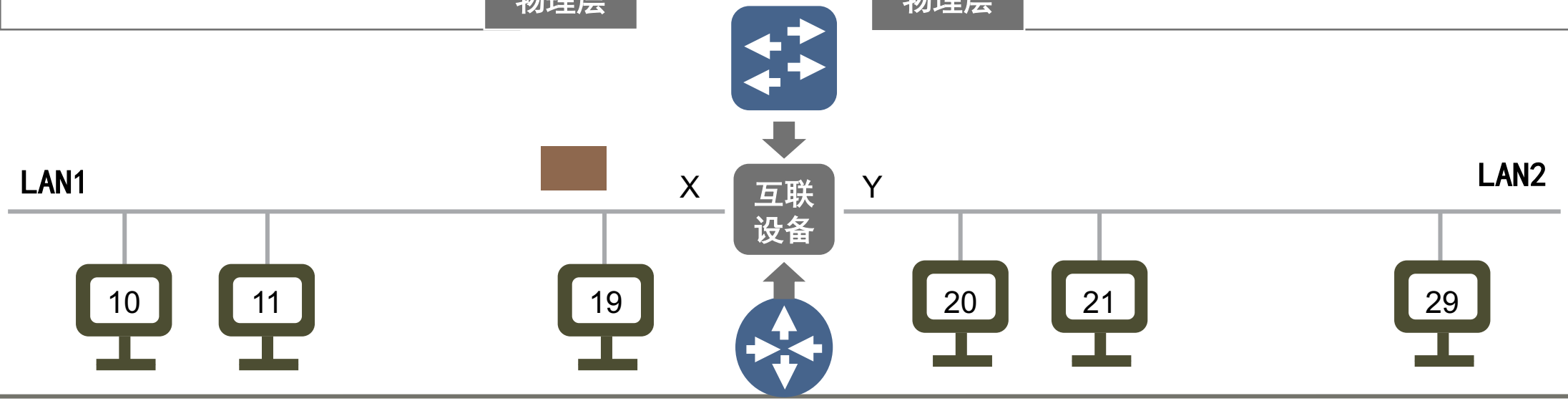
- 仅检查帧头，并不检查或修改帧包含的网络层包
- 不知道它正从某个802.x转发到802.y的帧包含什么协议的包

网桥协议
MAC
物理层

路由器（包）

路由协议
网络层
MAC
物理层

- 检查包头，并根据头包含的目标地址作出路由决策
- 当将包交给数据链路层转发时，它将该包封装在以太帧或非以太帧中。



X,Y：网桥/路由器的端口

网络层互联面临的困难

网络互联面临的主要问题

提供的服务不同

- 一个提供面向连接服务，另一个提供无连接
- 当分属两个子网的节点通信时

组播是否支持

- 组播是一对多的通信模式
- 一个网络具备把一个包发给一组节点的能力，而另一个网络没有

采用的协议不同

- 两个子网采用了不同的协议，导致两个子网的包结构以及字段内容定义解释不一样

包大小不同

- 如果两个子网的包长短不一，在转发时面临着长的包无法通过短包所在的网络

寻址方式不同

- 不同协议子网的寻址方式通常不同，地址字段的位数、网络编址方法都对包传递带来困难

服务质量不同

- 当实时数据包通过不能保证实时性的非实时网络时，实时网络的上层用户就可能得不到传输保障

网络互联面临的其他问题

流量/差错处理不同

- 差错校验方法（奇偶、CRC循环码）
- 差错处理方法（检错重发、纠错编码）
- 控制机制（停等式、滑动窗口）
- 序号空间大小、窗口大小不等

拥塞控制不同

- 不同网络采用方法不同（主机控制、网络控制、集中、分布式的。。。）

安全性不同

- 使用规则和加密算法的不同

参数设置不同

- 各种计时器设定/数据流说明

计费方式不同

- 按连接时间
- 按传输的包数
- 按传输的字节数

网络层的包长度有限

- 包的长度受到限制
  - 硬件的限制
  - 操作系统的局限
  - 协议设计

有效载荷从48~65535字节不等

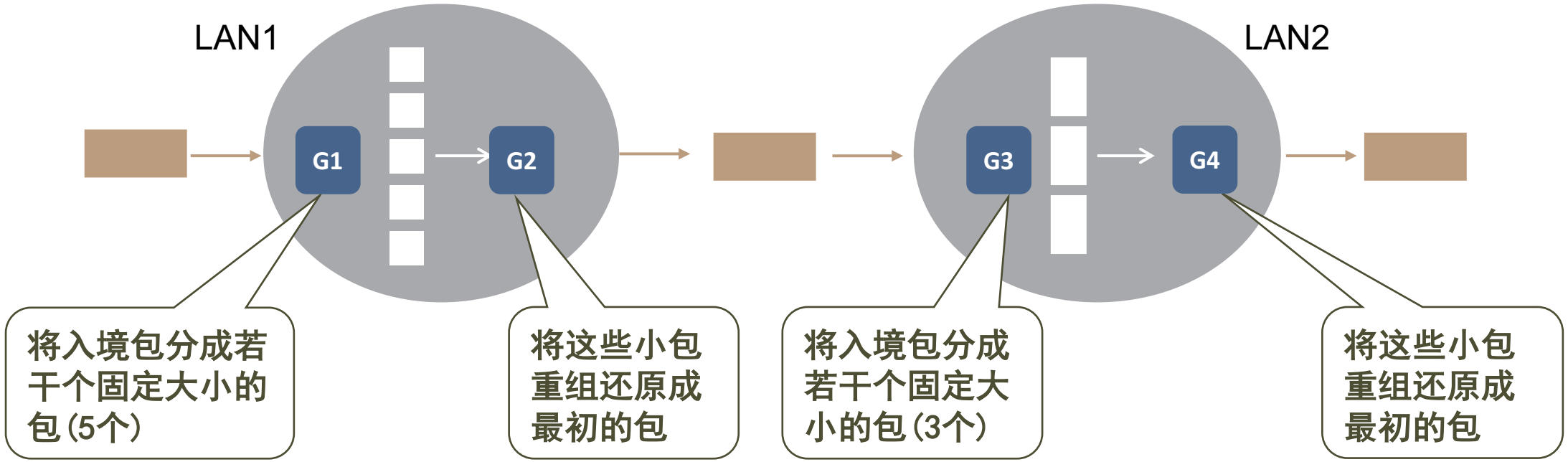
? 包长度超过网络所能传的最大长度时如何处理？

  - 减少由差错引发的重传
  - 防止某个包占用信道时间过长

分段技术：将大的包分成网络能容纳的一系列段，将每一段作为一个独立的包发送。

透明分段

透明分段：数据包进入网络时按需分段，离开网络时回复原样，使得前面的分段对后面的网络透明。



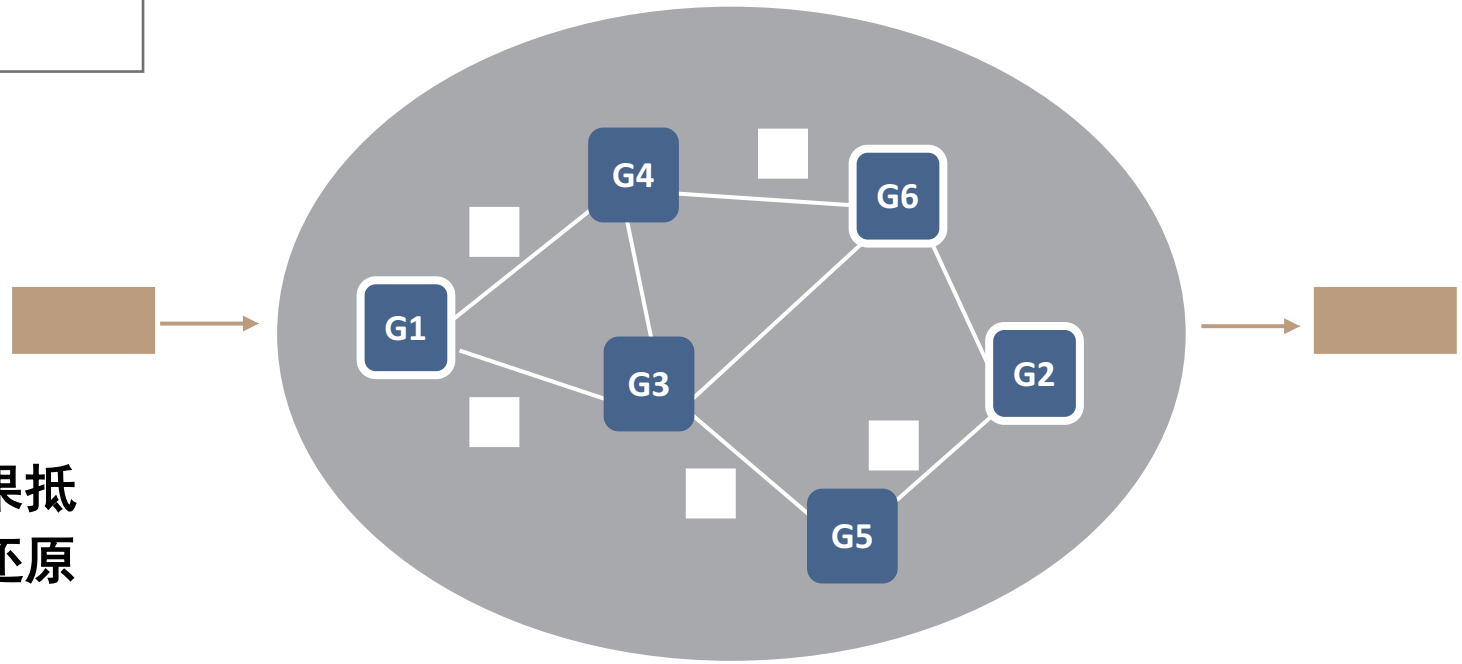
网关　分段后的段　原始包

透明分段策略

分段策略特点

- 出口网关必须确定何时收到全部小包
- 所有小包必须经同一网关离开网络
- 不断地分段与重组会增大开销

假设：G1、G2、G6都是边界网关



如果拆分的小包走了不同路径，结果抵达不同的出口网关，那将无法重组还原出原始包。

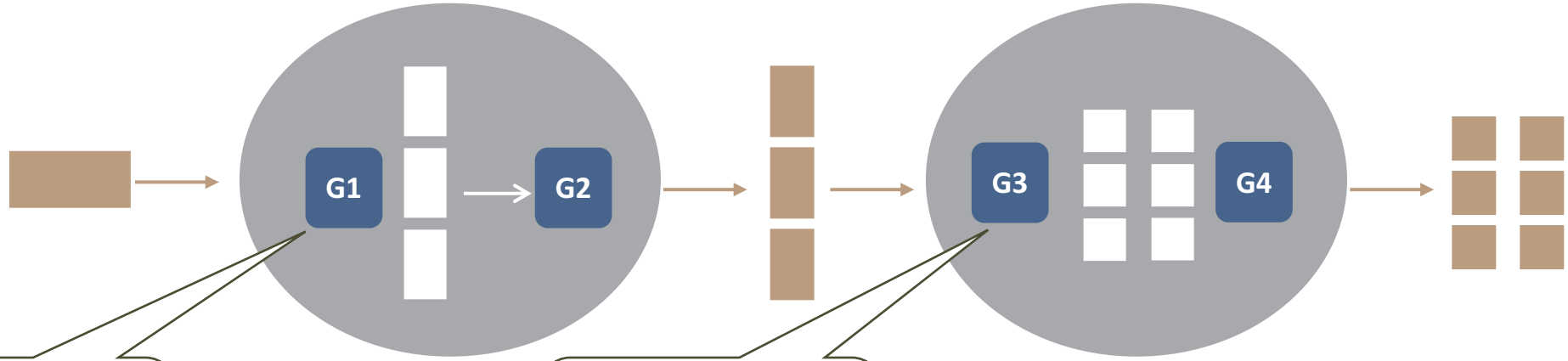
网关　分段后的段　原始包

不透明分段

不透明分段：任一中间网关都不重组，必要时只进行分段，仅在目标主机进行一次重组。

不透明分段特点

- 要求每个主机都能重组
- 总的开销增大



网关　分段后的段　原始包

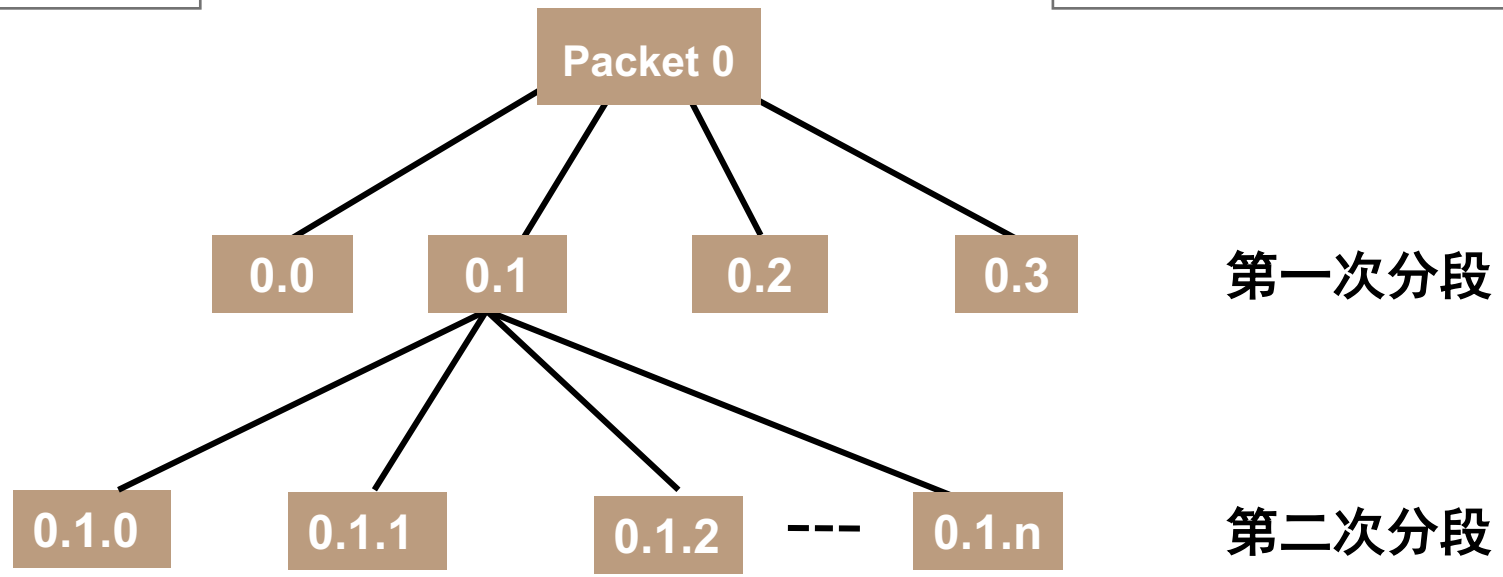
重组

重组目的

将分成若干段的小包
还原成原始的长包

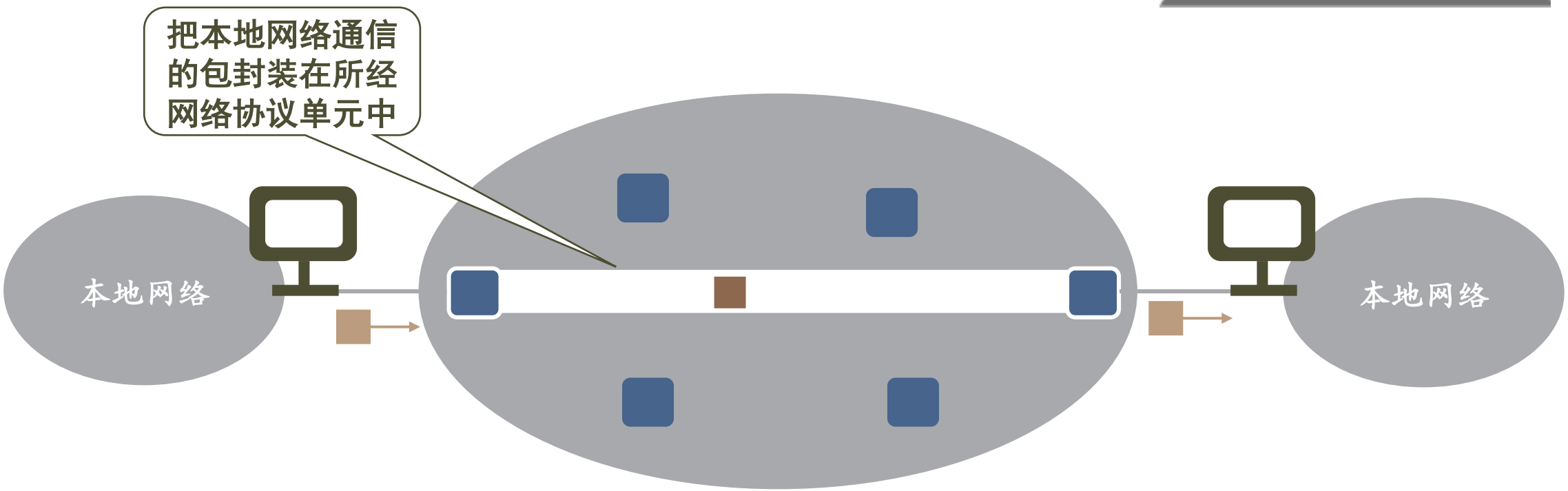
重组方法

- 一般采用树型结构编号
- 按照编号组合成原始包



隧道互联技术

隧道：在两个端点建立传输数据报的虚拟管道，使所传输的数据报不为途径的节点所知，通常采用封装技术。

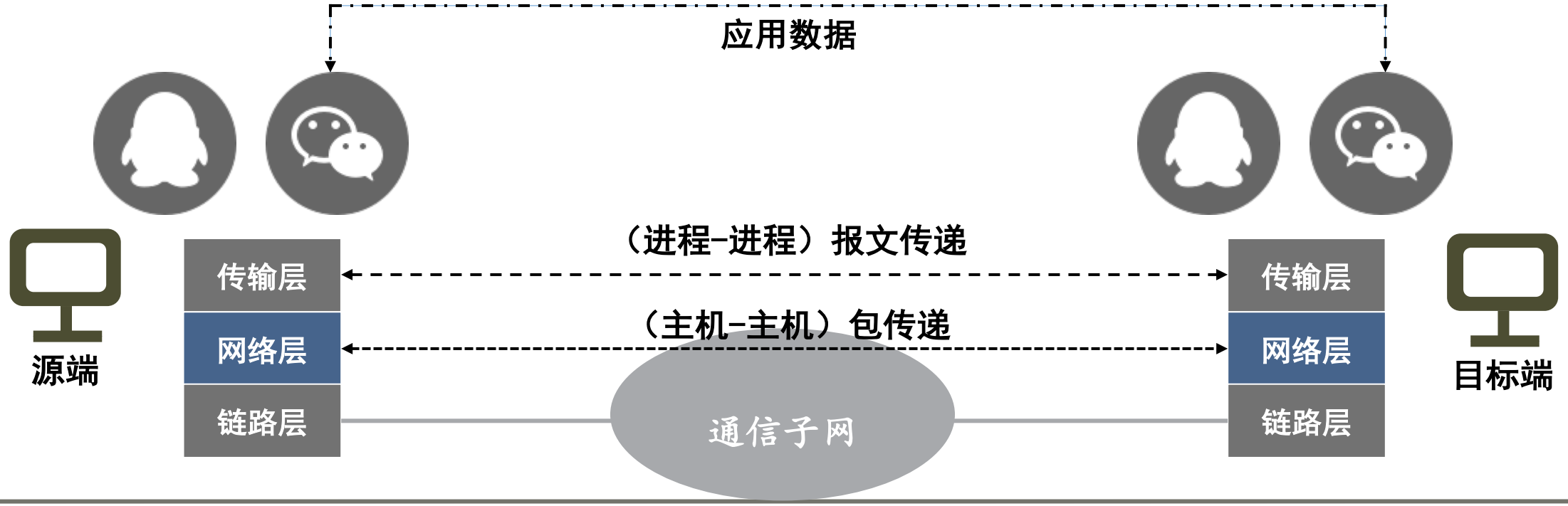


网络层功能及作用

端-端通信

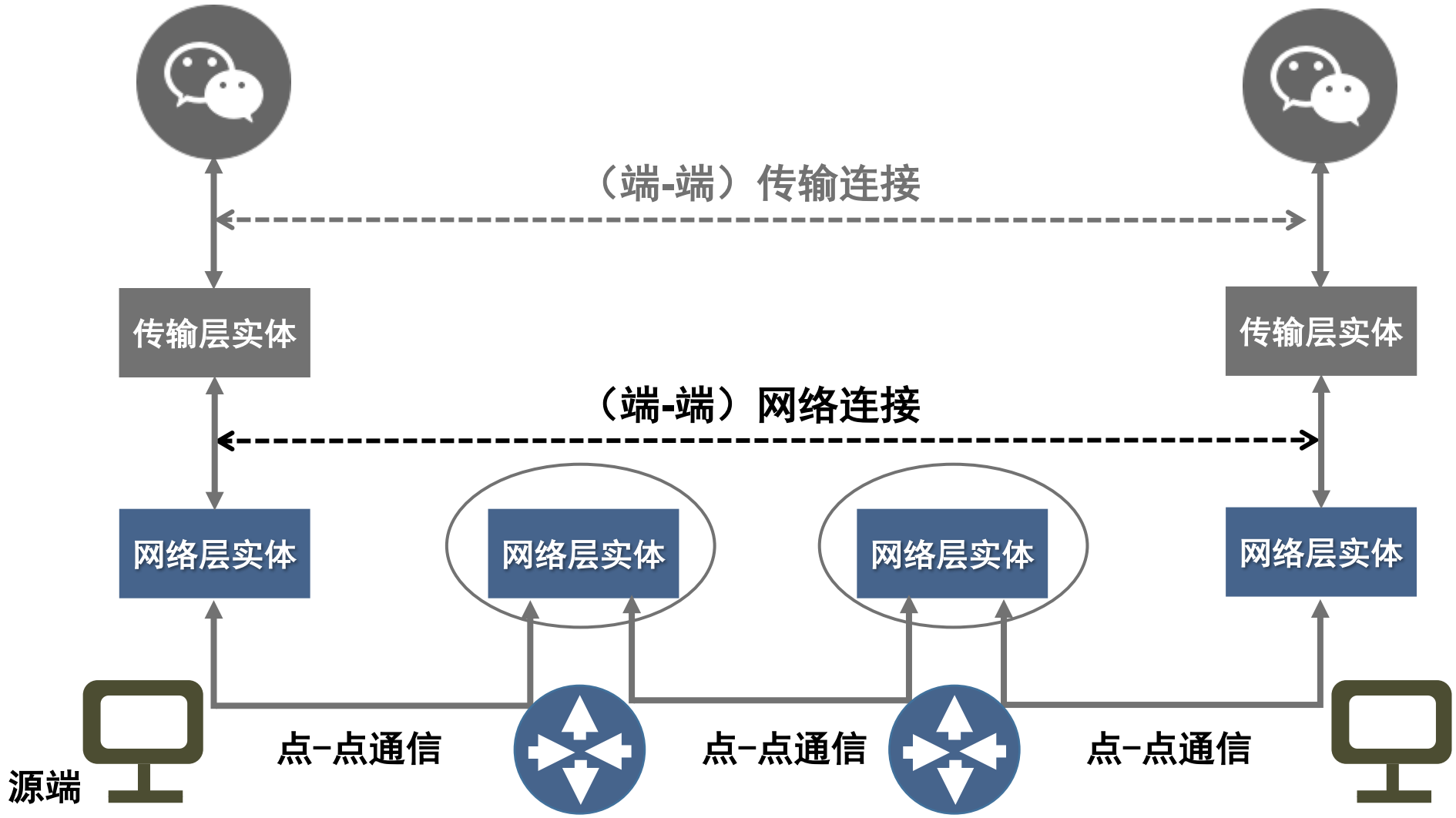
端-端通信：两个计算机系统传输实体之间的通信。

- 网络层为传输层提供服务
- 网络层是处理端-端数据传输的最底层



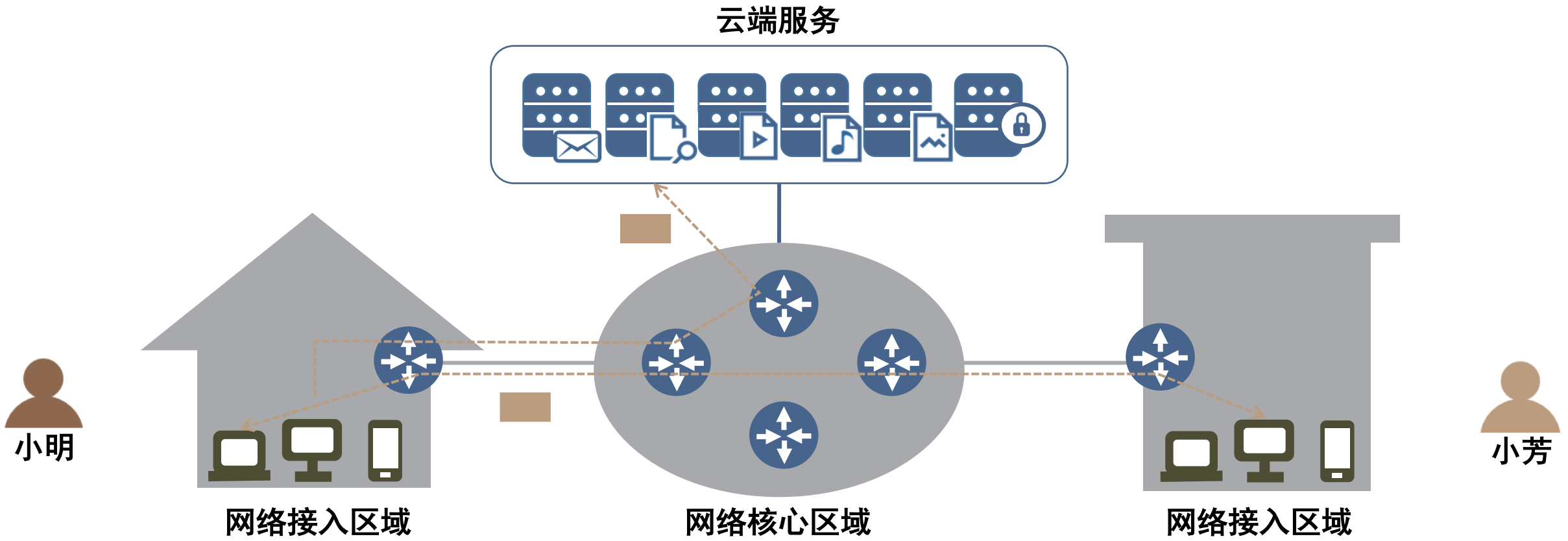
端-端通信与点-点通信

网络层连接从源端到目标端包含了若干个中间系统。



数据链路层仅将数据帧从传输介质的一边送到另一边。

远程通信与网络路径



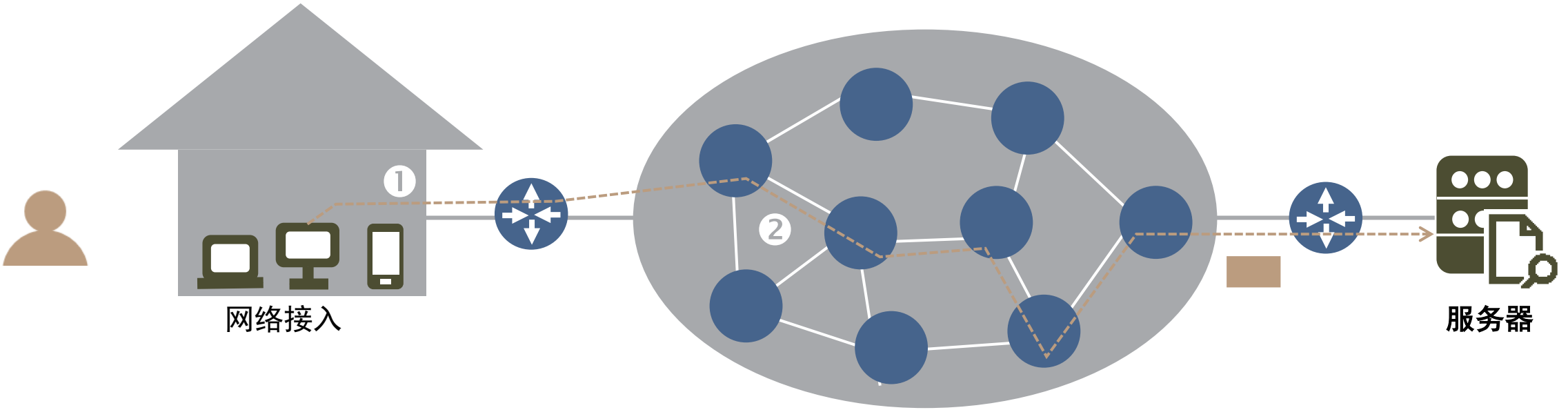
网络层的作用

① 接入网络将用户数据报发往因特网
② 网络核心完成路由选择，由链路层实现每一跳的数据传输

网络层在数据包的传递中起着决定性的作用。

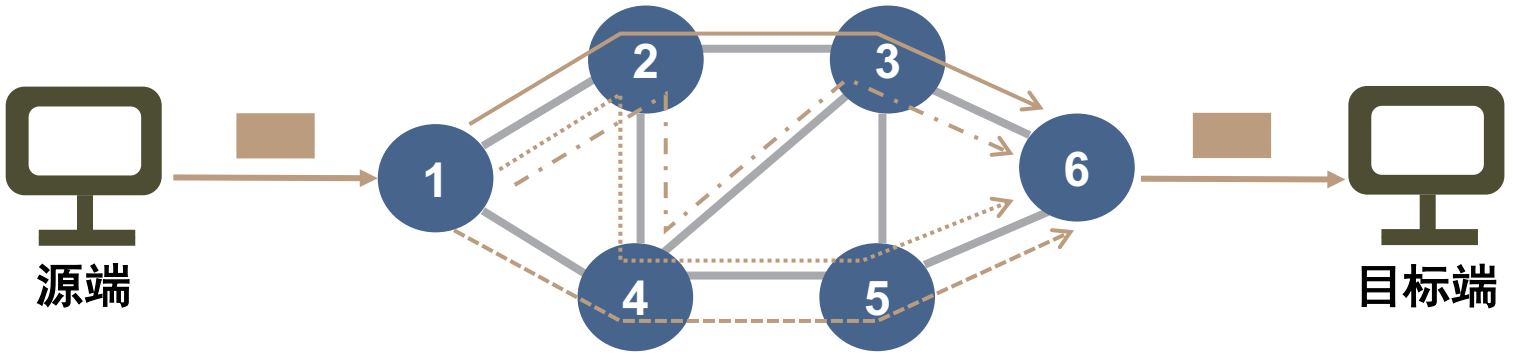
?

- 如何标识自己
- 如何找到对方
- 如何选择通往对方路径
- 路径的好坏如何定义



网络层的命名和路由机制

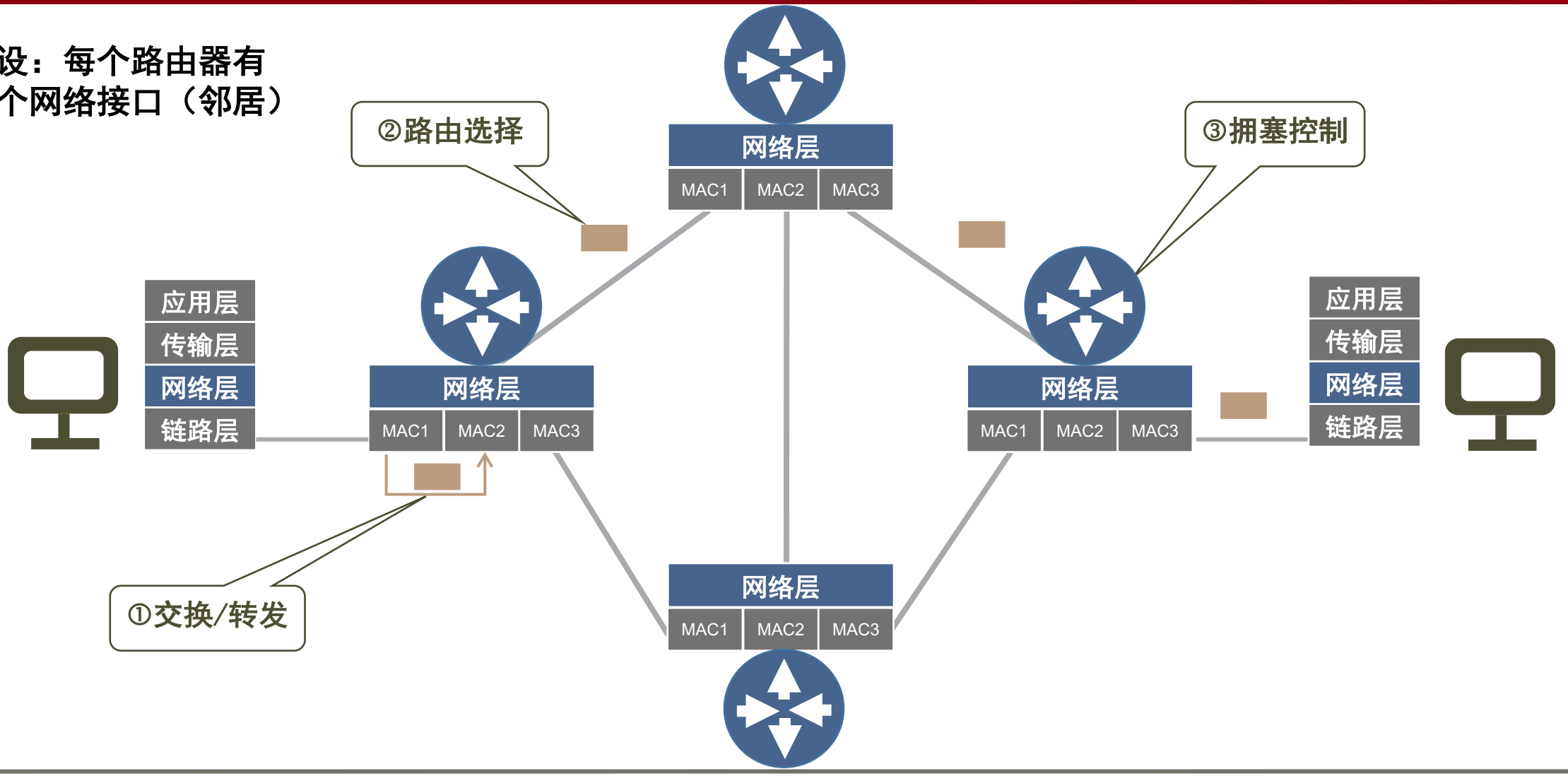
- 两个端系统（主机/服务器）通信需要一种命名和路由选择机制
- 路由器沿着通向目标端的方向转发数据包



客户机/服务器通信包

网络层的功能

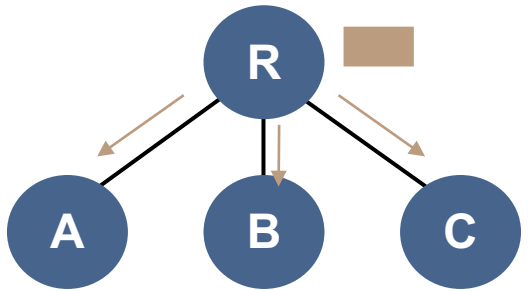
假设：每个路由器有
三个网络接口（邻居）



网络层的数据平面和控制平面

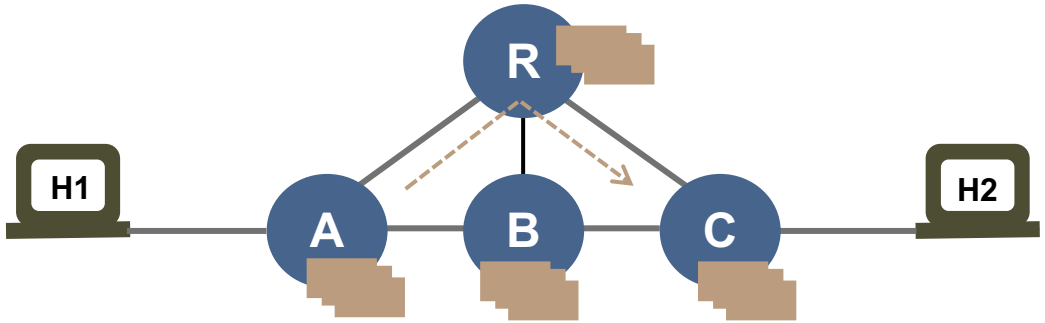
	转发(数据平面)	路由选择(控制平面)	
本地操作	当一个包到达路由器时，路由器必须将它放到适当的输出链路。	路由器必须确定包从发送主机流向接收主机所走的路由或路径。	全局决策

① 当路由器R收到从任一邻居发来的包（源地址X，目标地址Y），要将其转发给离目标地址最“近”的邻居（A,B,或C）。



② 路由器要确定从本地出发到任意目标地址Y的最好路径，即从所有邻居（A,B,C）中选择一条通往Y的最好出路。

网络层的连接建立和拥塞控制



呼叫建立(面向连接)

某些网络层需要所选路径沿途上的路由器，在真正数据交换之前握手协商状态信息(如序号、初始流控的窗口大小等控制参数)。

拥塞控制

网络交通拥塞会增加包传输延迟，从而降低吞吐量。路由器要调节数据包通过网络的流动。

例如：主机H1要给H2发送数据。

- 两个路由器必须为这次数据交换建立一条逻辑连接
- 当网络出现拥塞，路由器要想办法解决拥堵问题。

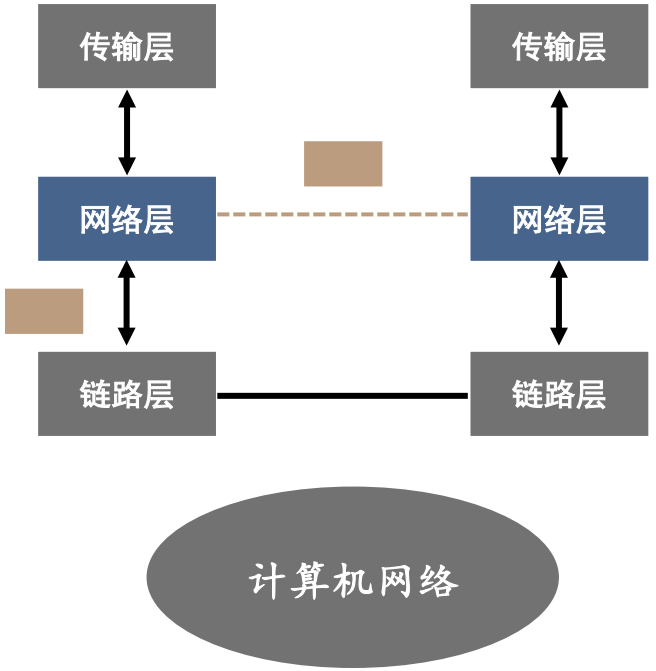
网络层服务

VS.

通信模式

网络层提供的服务

- 传输层是网络层的服务使用者
- 传输层通过与网络层的接口把数据交给网络层发送



- 网络层是传输层的服务提供者
- 网络层把上层用户数据按照本层协议封装成包
- 网络层通过与链路层的接口把包交给下层发送

面向连接

当上层用户（本层的服务用户）通过接口把数据向下交给网络层时，网络层要先建立一条与目标节点的连接，然后把封装了数据的包往下交给链路层（本层的服务提供者）发送，相当于把包发到了该连接上。

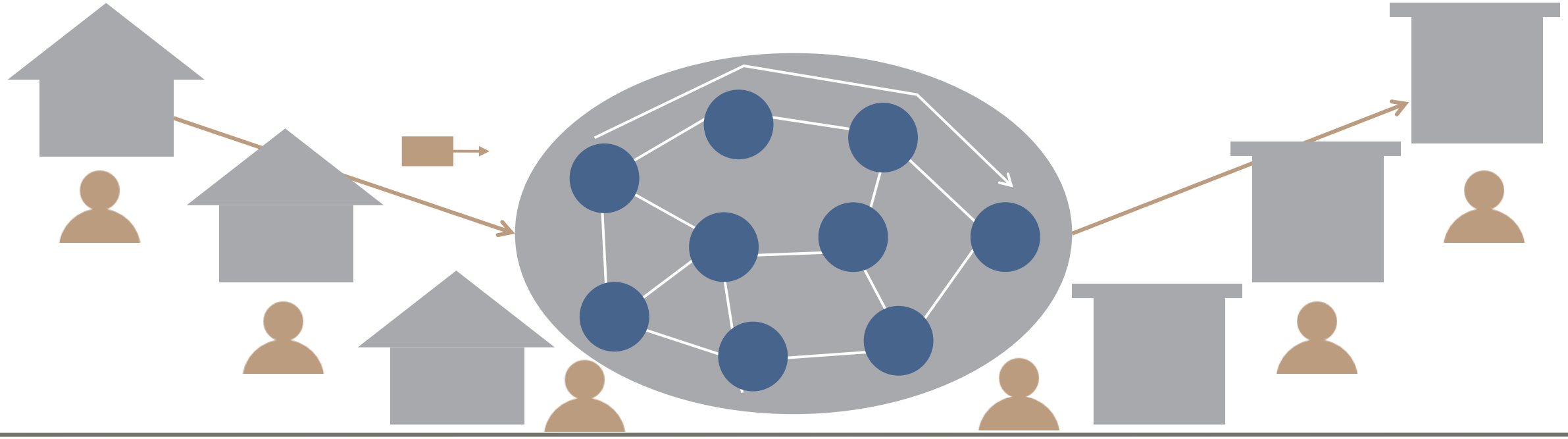
无连接

当上层用户通过接口把数据向下交给网络层时，网络层直接把数据封装成本层的包，往下交给链路层发送。

网络层的通信模式——单播

单播

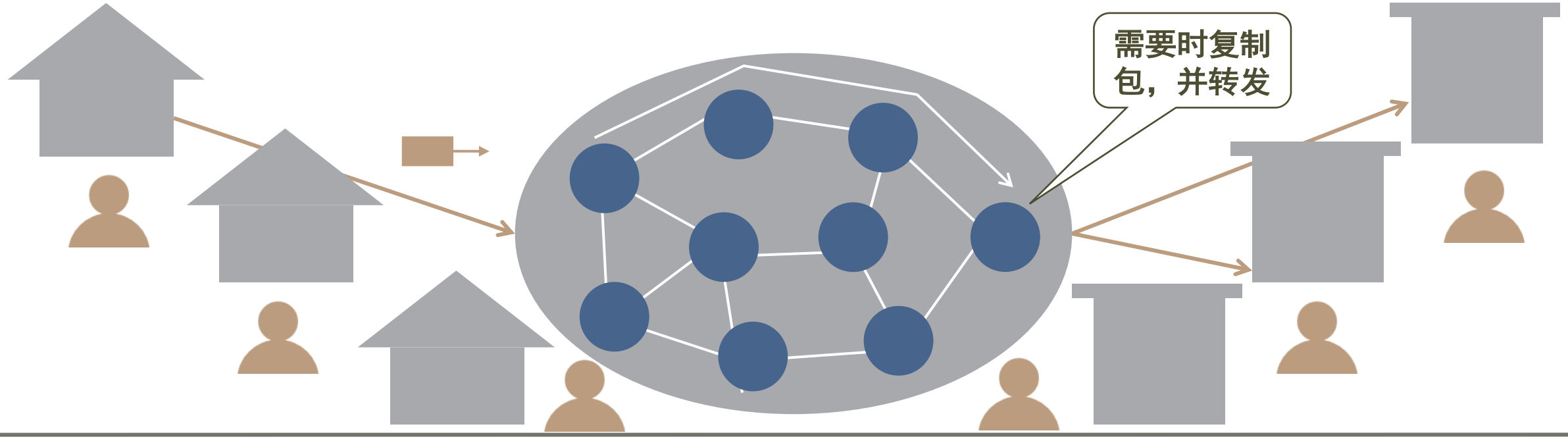
- 一对一的通信
- 必须有一套全局唯一的命名规则
- 每个网络节点有一个唯一的ID
- 路由器处理包时根据该ID进行路由决策



网络层的通信模式——组播

组播

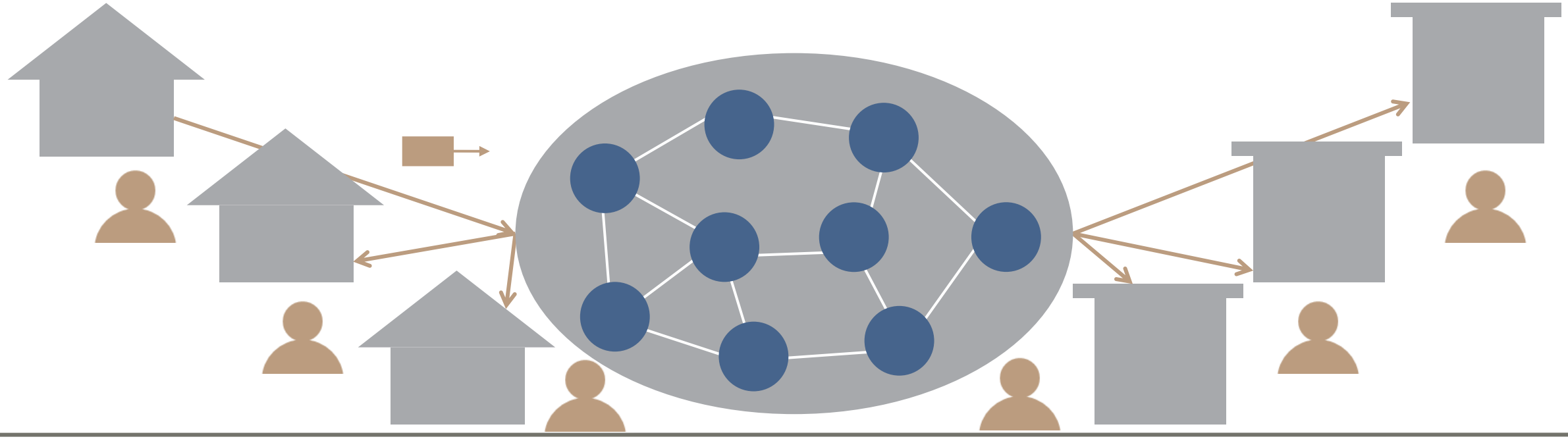
- 一对多的通信
- 必须有一套命名一组用户的规则（组播地址）
- 每个网络节点除有固定唯一的ID外，还有相应的组ID
- 路由器根据组成员分布将组播包转发到组成员所在通信子网



网络层的通信模式——广播

广播

- 一对全部的通信
- 必须有一套命名全体用户的规则（广播地址）
- 每个网络节点除有固定唯一的ID外，还有相应的广播ID
- 路由器将广播包转发到一定范围内的全部通信子网

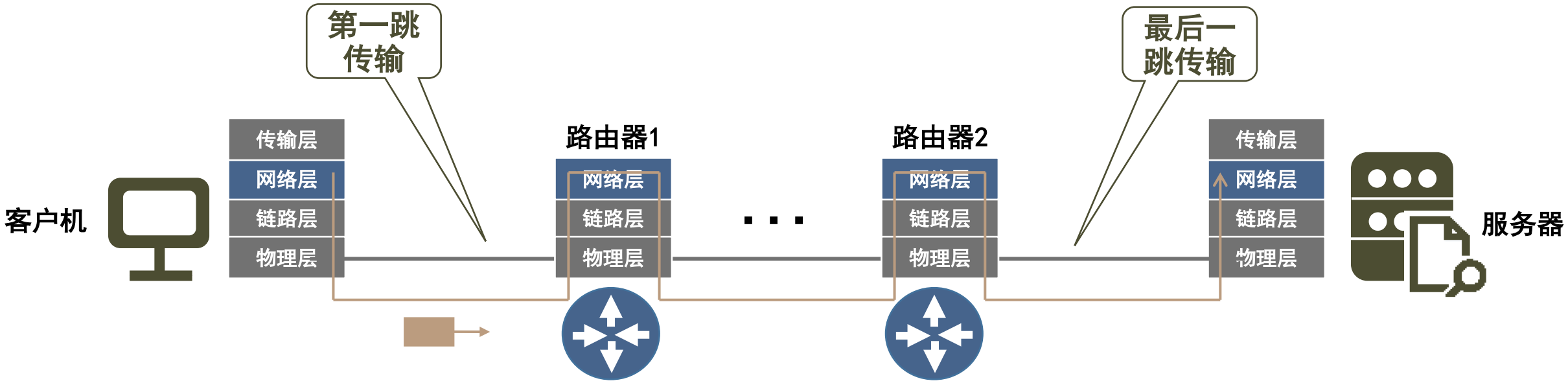


存储转发vs.路由选择

包的逐跳传递

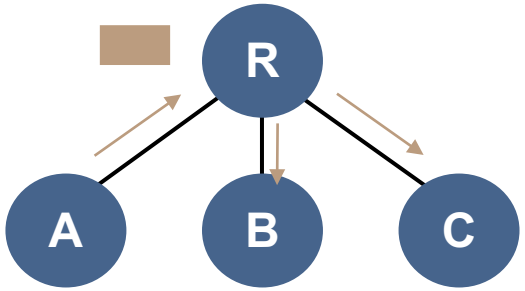
一跳(one hop)传输：指网络传输路径上的一次存储-转发。

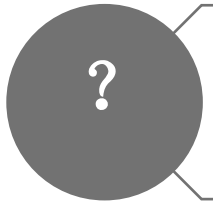
- 在包的传输路径上，每个路由器收到包后，根据包给定的目标地址把包从特定端口转发到路径上的下一站
- 网络层的包必须被封装在端口所连网络协议规定的帧中才能真正被发送



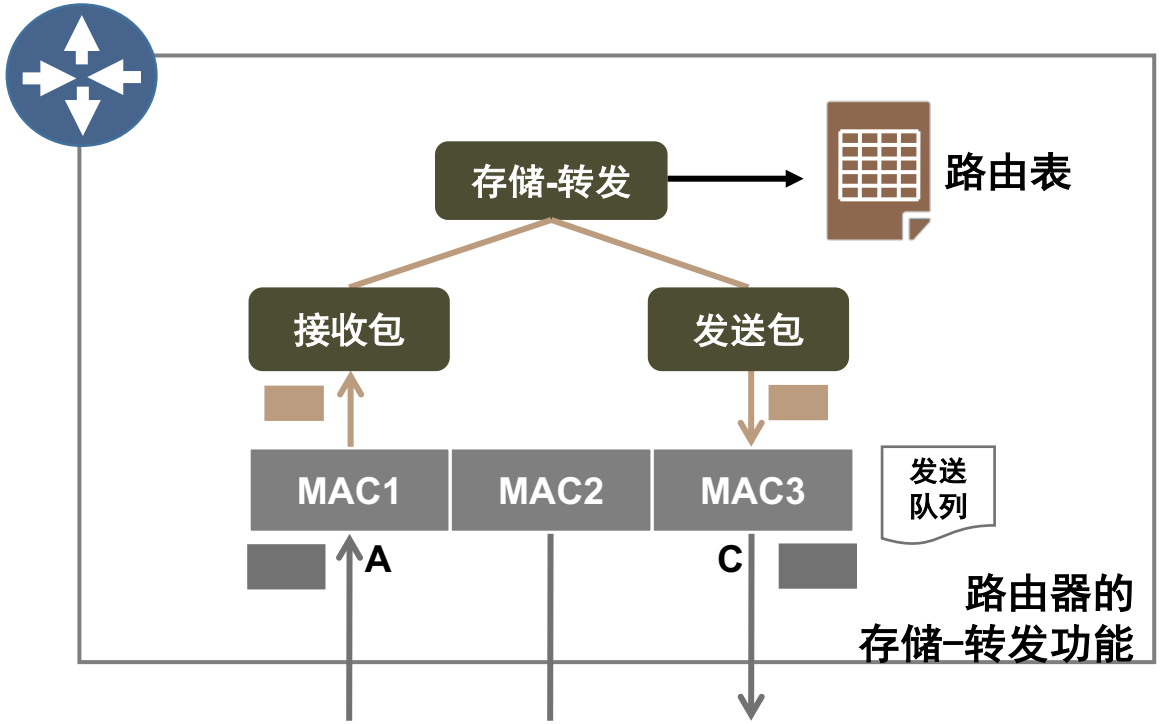
网络层包

路由器的存储-转发





路由器R如何根据包的目标地址做出路由决策（B/C？）



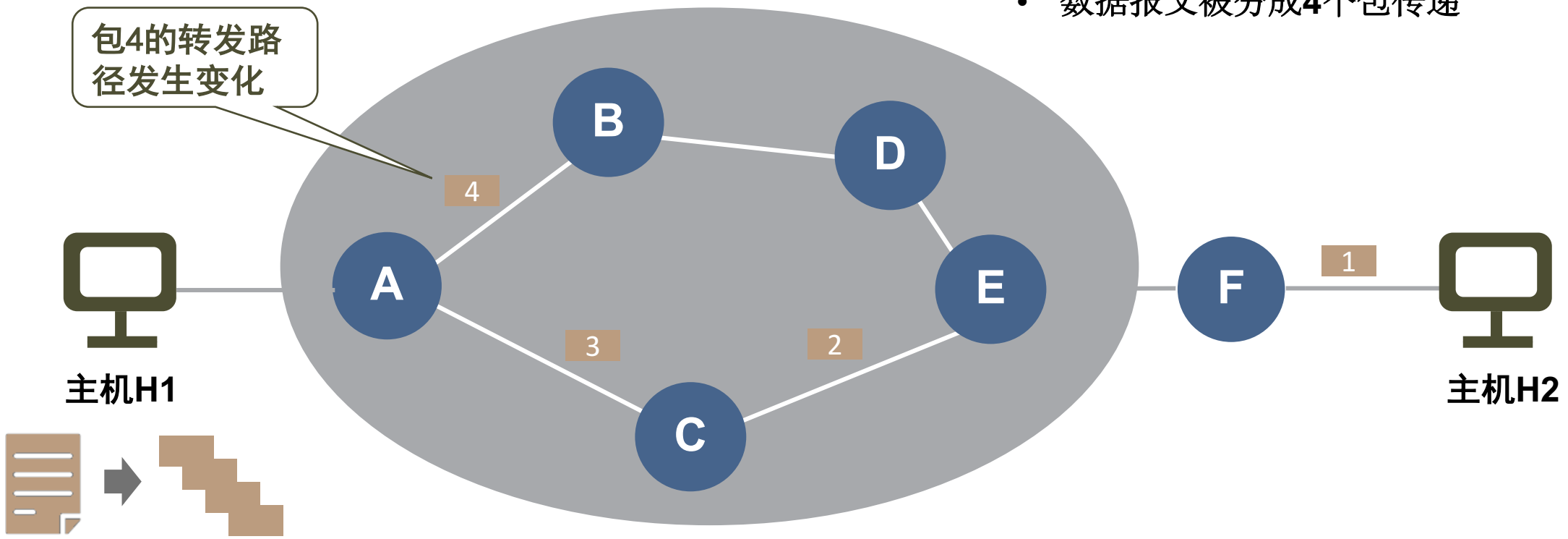
存储-转发时延：从包进入路由器R算起，到被转发到下一站路由器C所经历的全部时间。

路由表在存储-转发中的作用

同一对端系统之间的包可能走不同的路径。

假设：

- 主机H1给主机H2发送一个数据报文L
- 网络规定的包长度为P
- 数据报文被分成4个包传递

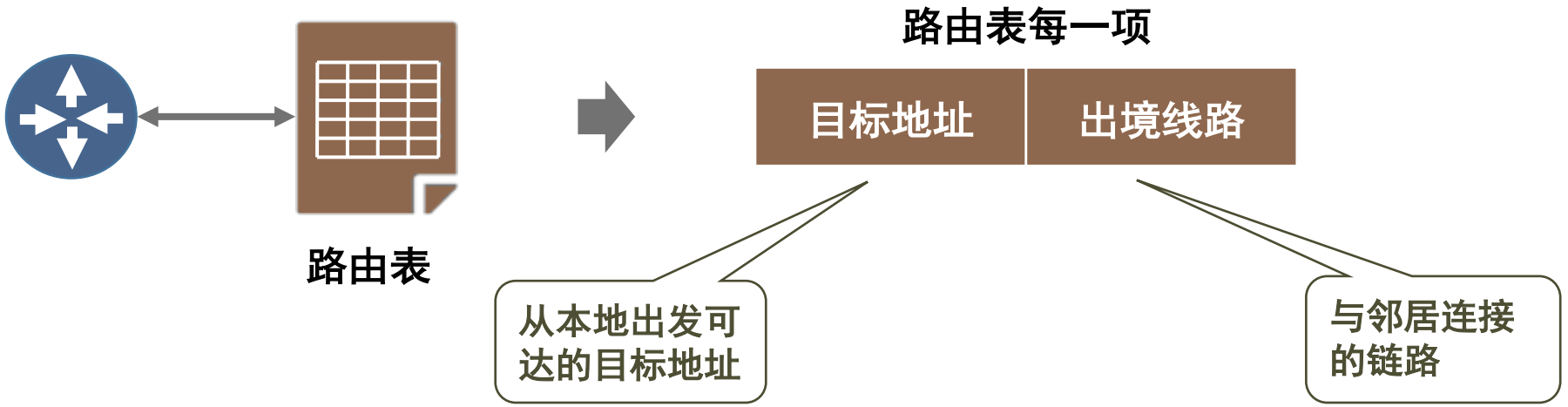


存储-转发技术的实现

存储-转发技术的实现

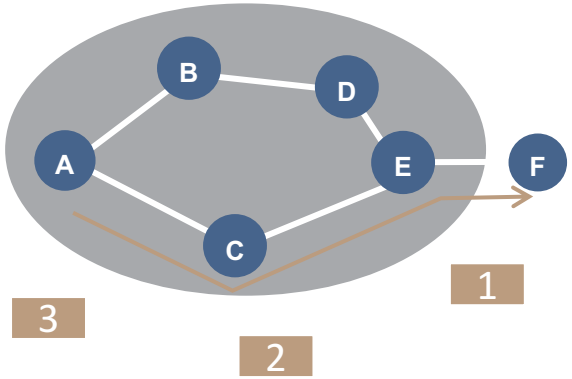
- 每个包必须包含目标主机的完整地址(如IP地址)
- 路由器用一张路由表指出通向目标端的出境线路
- 当一个包入境时，路由器查找路由表并将包沿出境线路发出，无须修改包中的任何内容。

- 路由表只表明从本地出发去往目标地址的路径
- 路由表必须及时更新反应网络的动态变化情况



路由表反应网络状态的实时变化

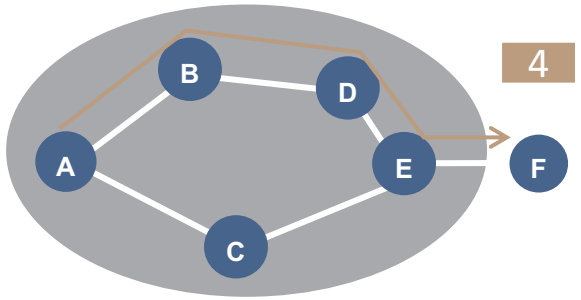
①A转发包1、包2和包3时，路由表信息显示包走A-C-E-F路径传输性能最“好”。



A的路由表

目标地址	出境线路
A	-
B	B
C	C
D	B
E	C
F	C

②A转发包4时网络状态发生变化，路由表显示走A-B-D-E-F路径传输性能比前面包走的路径更好。

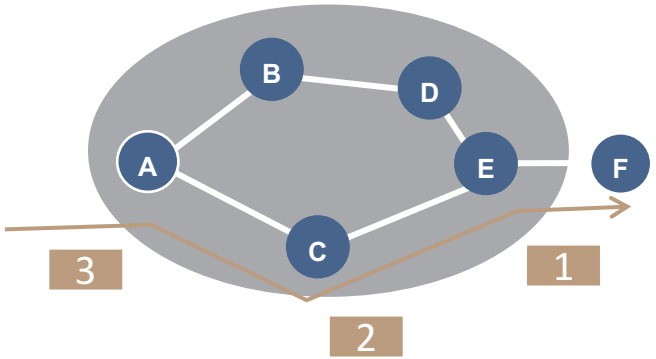


A的路由表

目标地址	出境线路
A	-
B	B
C	C
D	B
E	B
F	B

完整路由表示例

示例1：包1、包2和包3传输过程中各路由器的路由表



?

- 路由器根据什么信息以及如何生成路由表

A的路由表

目标地址	出境线路
A	-
B	B
C	C
D	B
E	C
F	C

C的路由表

目标地址	出境线路
A	A
B	A
C	-
D	E
E	E
F	E

路由器功能

- 存储-转发
- 生成路由表

E的路由表

目标地址	出境线路
A	C
B	C
C	C
D	C
E	-
F	F

F的路由表

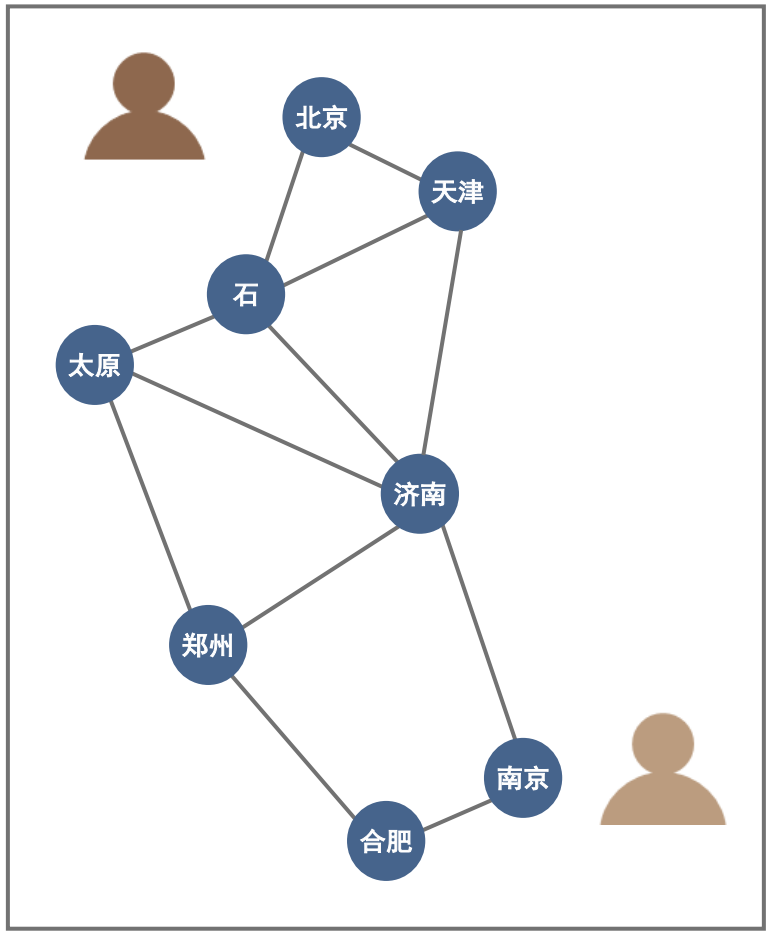
目标地址	出境线路
A	E
B	E
C	E
D	E
E	E
F	-

路由算法概述

旅行者 vs. 旅行路线

? 北京的“小明”要去南京看望“小芳”怎么走

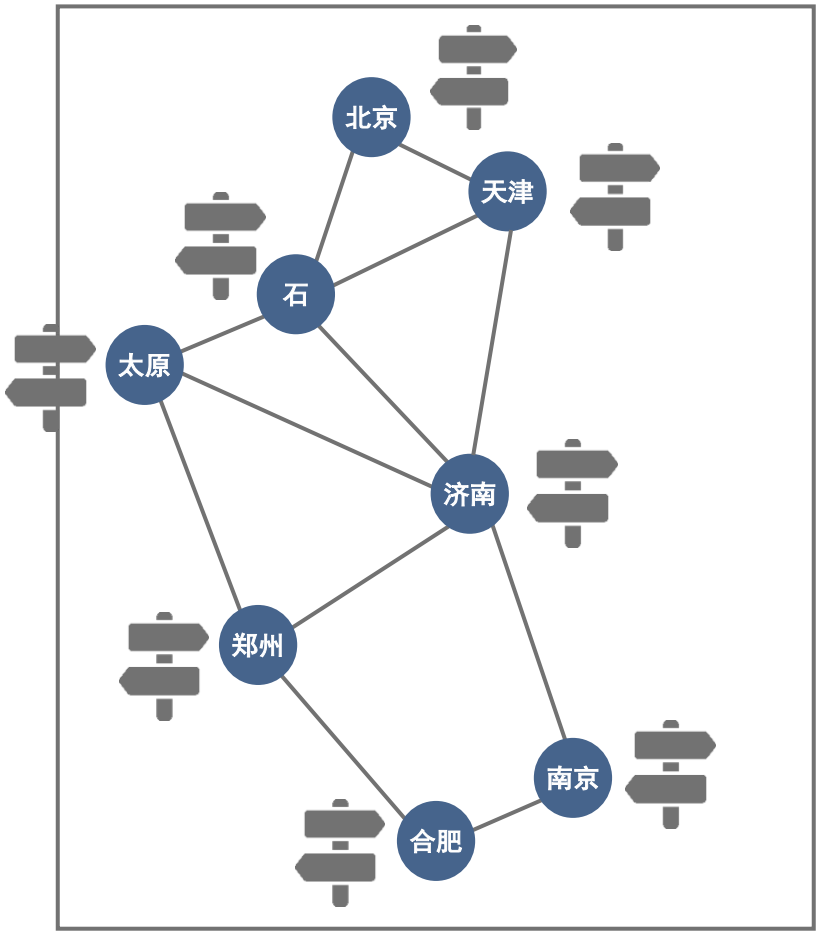
	有地图	没地图	
链路状态路由算法	自己独立计算从北京出发到目的地的旅行路线，并且知道该条线路的长短。	看自己有几个邻居，期望通过和邻居交流获得抵达目的地的所有线路，并从中选择一条最好的。	距离矢量路由算法



走一步看一步的旅行

假设

- **出发地北京，目的地南京**
- **没有同行者商量**
- **每一站都有旅行信息咨询机构（相当于在交叉路口有方向指示牌）**



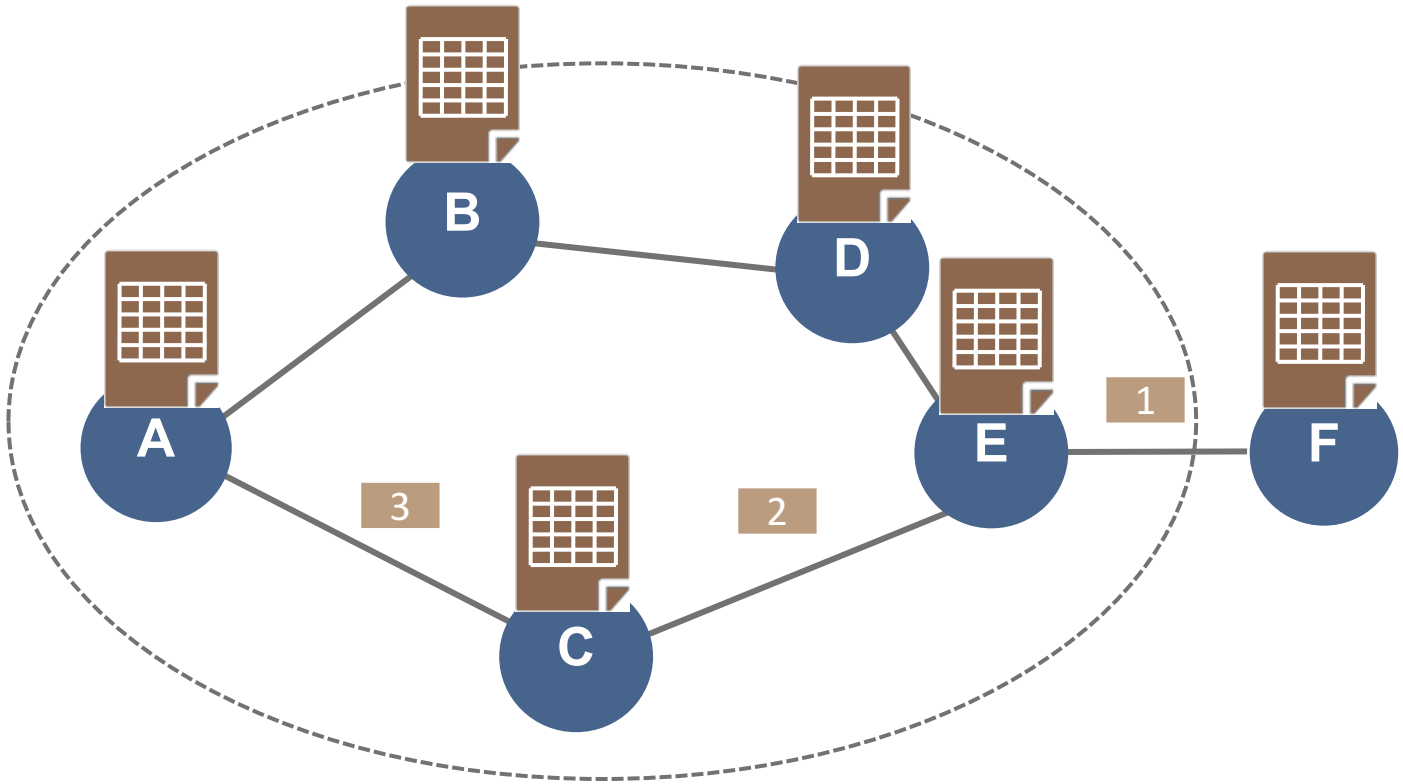
小明可能的旅游线路：

- 沿着唯一的路出发，在每个交叉路口寻找方向指示牌
- 第一个路标：“到西安太远向右，到天津向左，其余向左”，选择了省缺路线向左；
- 到天津后发现第二个路标：“到石家庄向右，到北京向后，其余向前”，选择向前
- 如此这般，在每个路口根据路标指示的方向前进
- 最终看到了通向“南京大学”的路标

包在互联网中传递

假设

- 每个包有完整的目标地址
- 每个包独立选择下一跳
- 只能依赖路由表

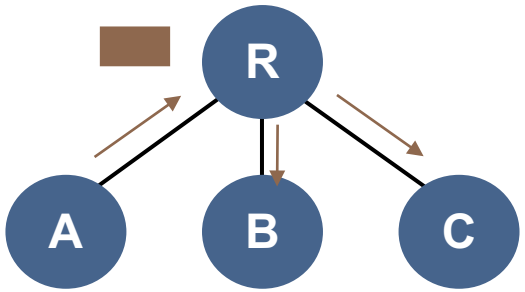


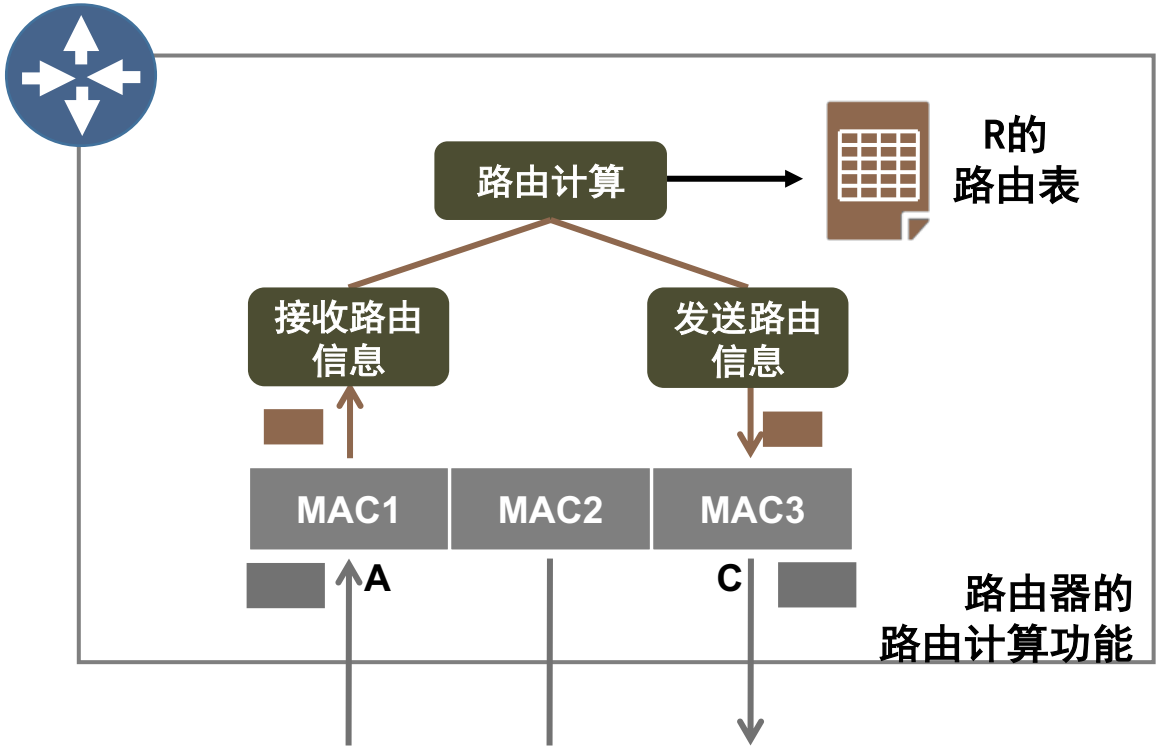
旅行者小明 = 网络层的包
旅行路线 = 包的路径
旅游咨询台 = 沿途路由器
交叉路口路标 = 路由器路由表

只要路由表上的信息是一致的和完备的，按照路由表走就一定能抵达目的地，而且这是一条最“好”的路径。

路由器的内部结构

假设：某个互联网中某个路由器R和邻居的连接关系如下：



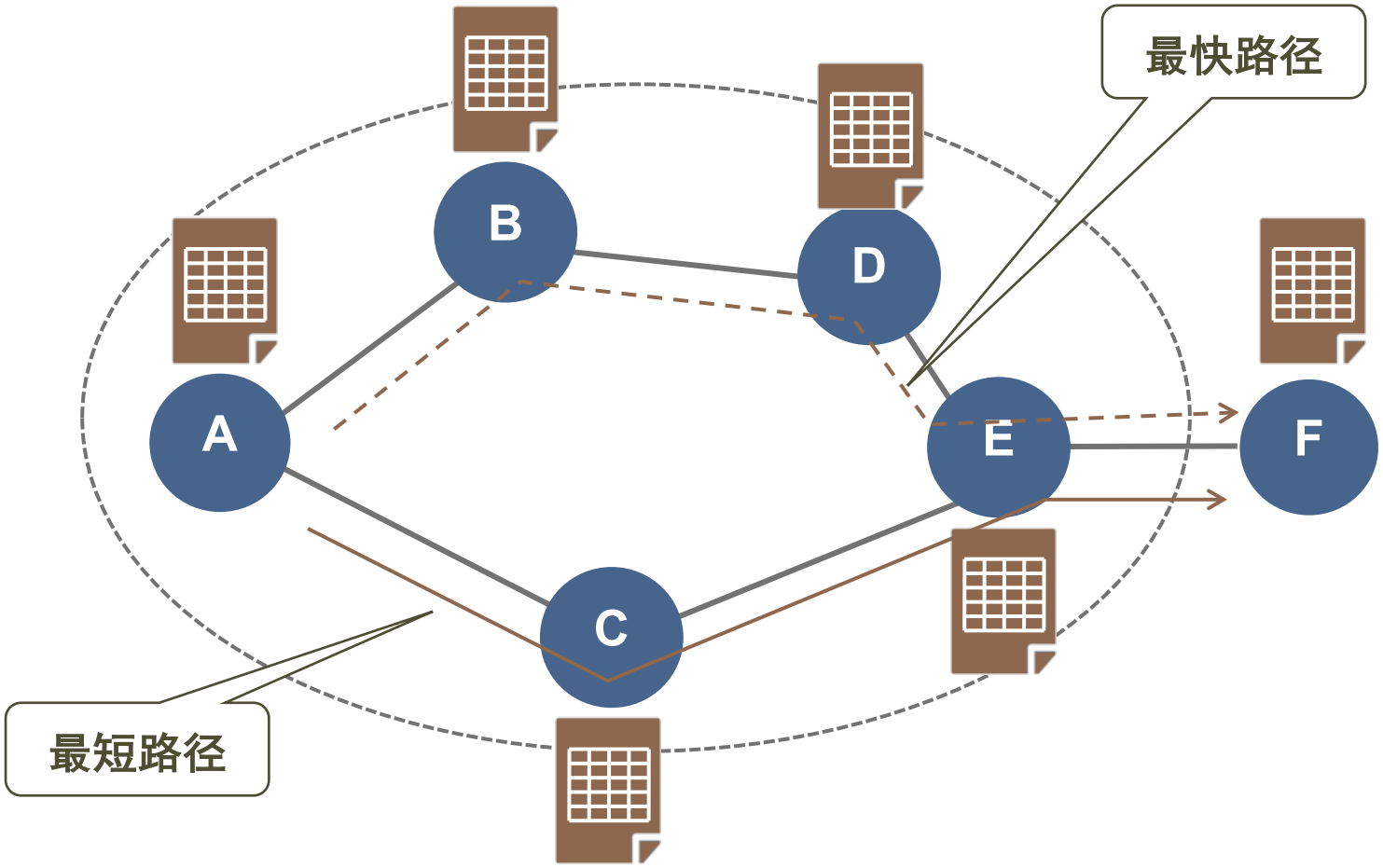


- 路由器之间只能通过消息交换获得更多的拓扑和路由信息
- 网络所采用的路由算法决定了各路由器之间交换怎样的路由信息

路由计算给出的最“好”路径

“好”的标准

- 最短路径
- 最快路径
- 最宽路径
- 最可靠路径
- 最底成本路径
- 。。。



路由算法的分类

非自适应算法——静态路由

- 不是根据实际测量或估计的网络当前流量和拓扑结构作路由决策

自适应算法——动态路由

- 根据网络拓扑结构、流量的变化来改变其路由选择

自适应路由的前提：节点间交换网络状态信息

- 信息越多，做出的路由决策越好
- 信息越多，网络负担越大性能下降越快

优点

- 可提高网络性能
- 有助于拥塞控制

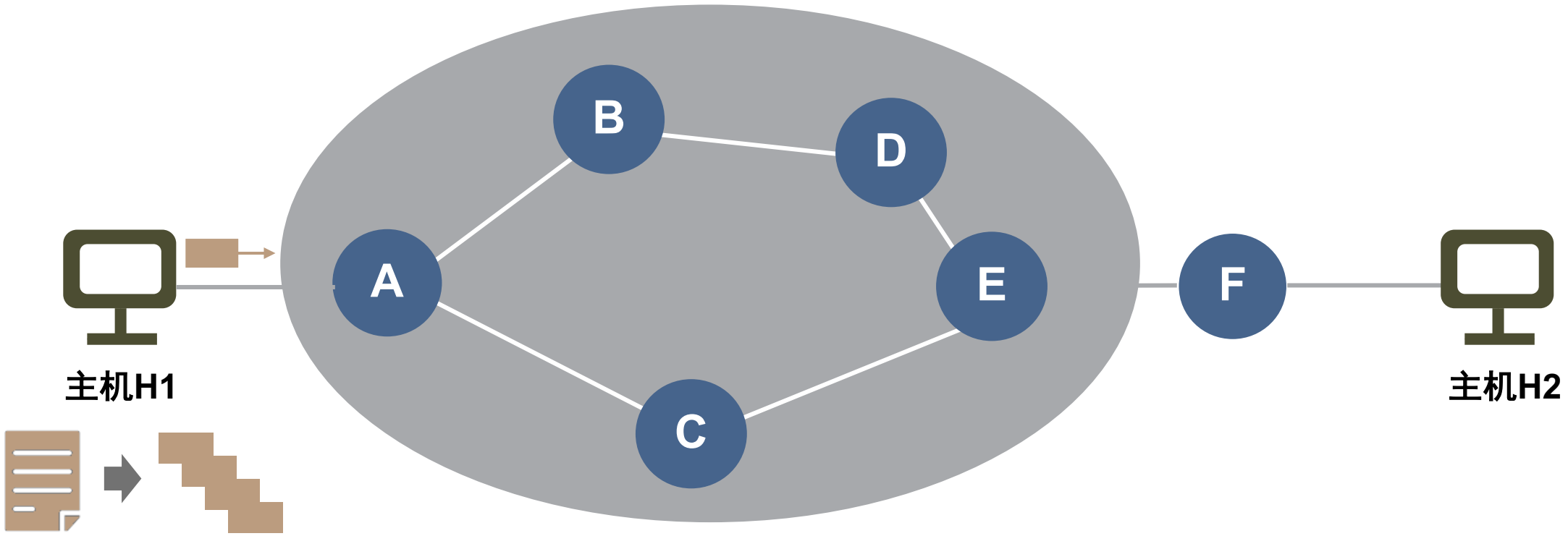
缺点

- 路由决策复杂
- 依赖于状态信息
- 不能太快和太慢

静态路由算法

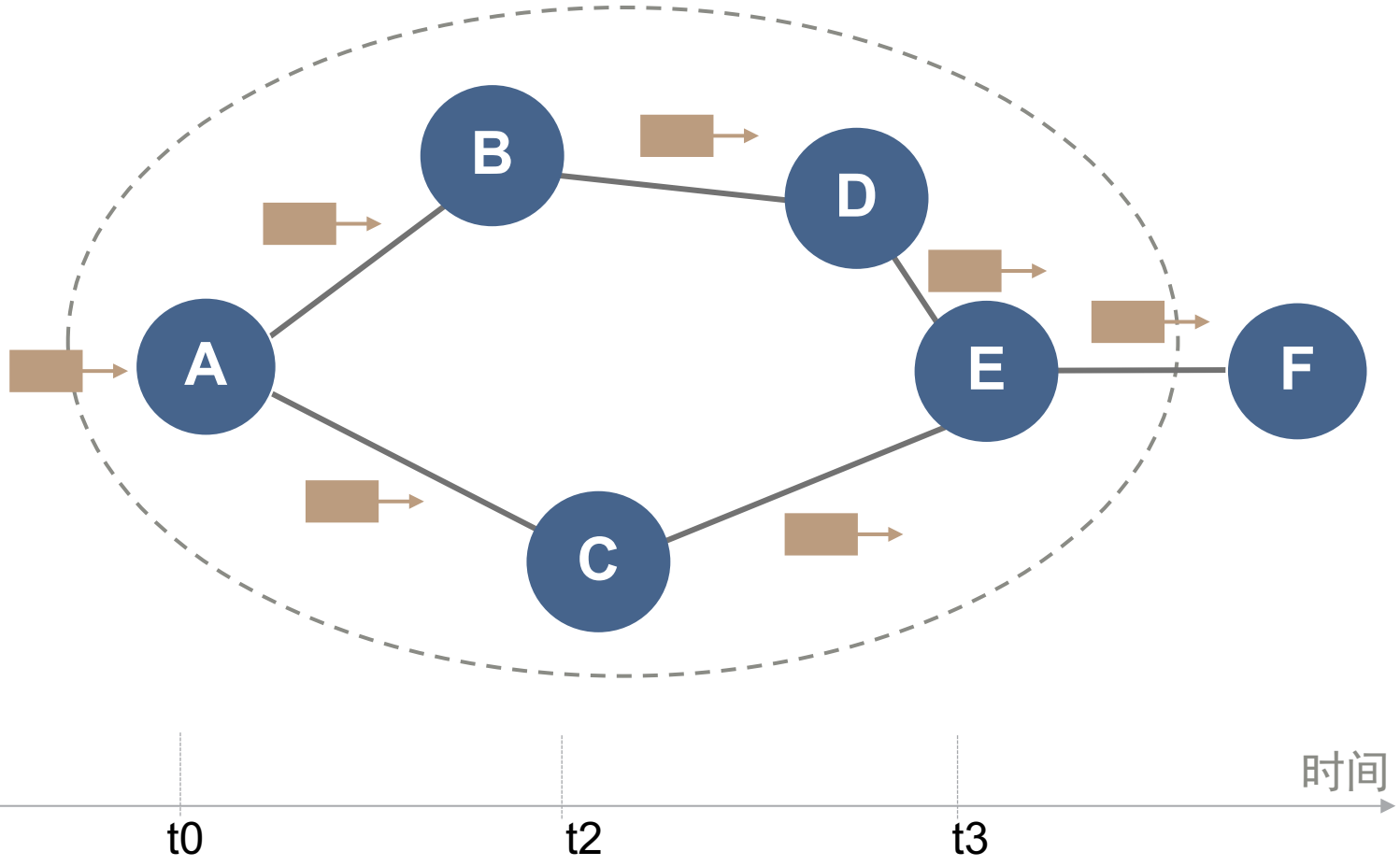
泛洪路由

泛洪法：路由器之间无需交换任何网络状态信息和路由信息，没有路由表，将收到的包转发到所有出境线路（入境线路除外）



H1发给H2的包　　Flooding：扩散法

泛洪路由过程示例



假设：每一跳的存储-转发时间相同

试问：有多少个节点会收到重复包？

t0：A收到H1发来包

t1：A将该包转发给B和C

t2：B和C分别转发包到出境线路

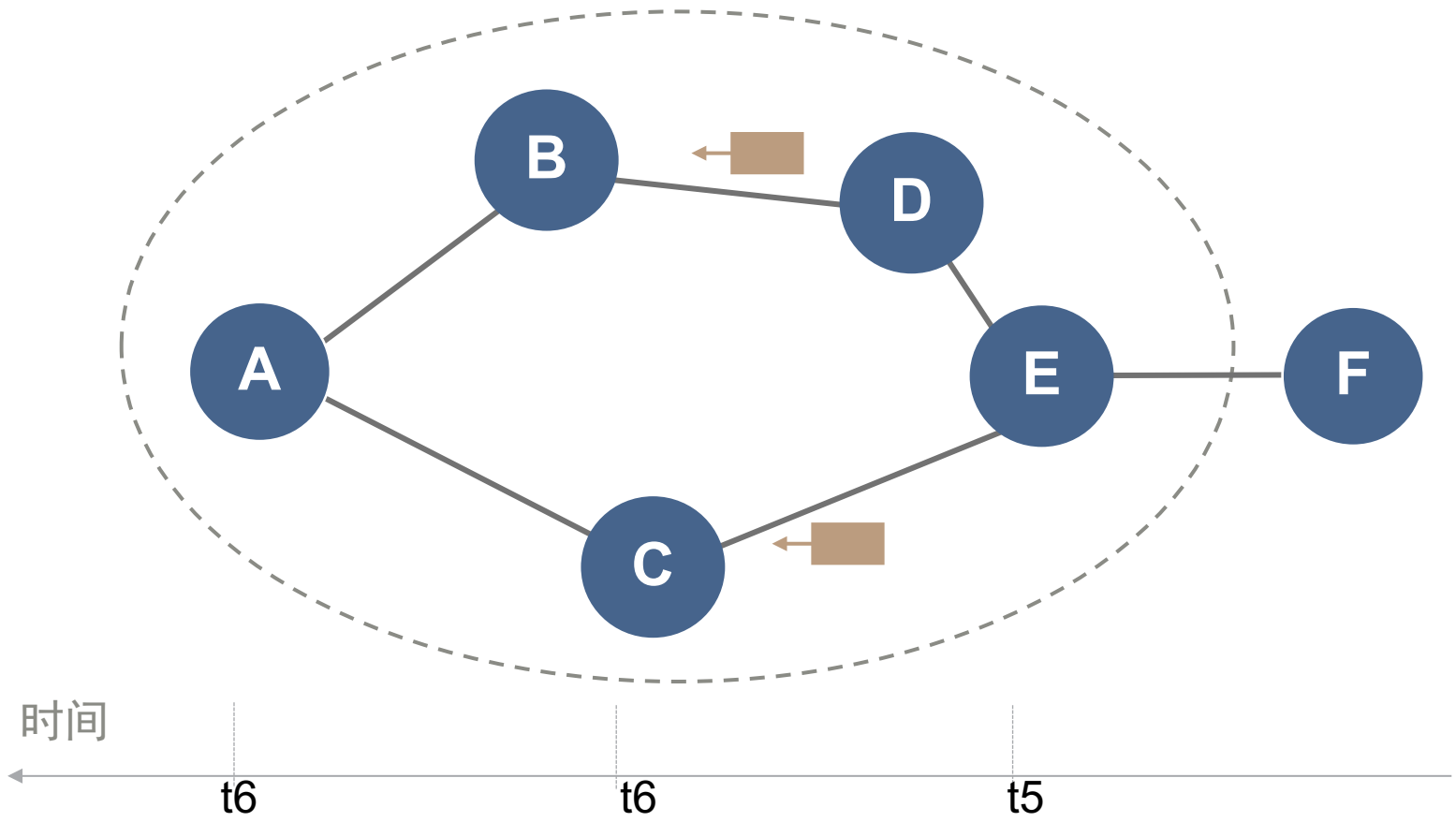
t3：D和E分别转发包

- D给E发送包
- E给D和F发送包

t4：F收到包，继而转发到本地网络H2收到包

？

泛洪路由过程中的重复包



t4：F收到E转发的包
E收到D转发的包
D收到E转发的包
t5：D和E分别转发包
t6：B和C分别转发包

只要网络拓扑结构存在环路，重复包将急剧增多，甚至成指数级的增长。

泛洪路由的特性

特性

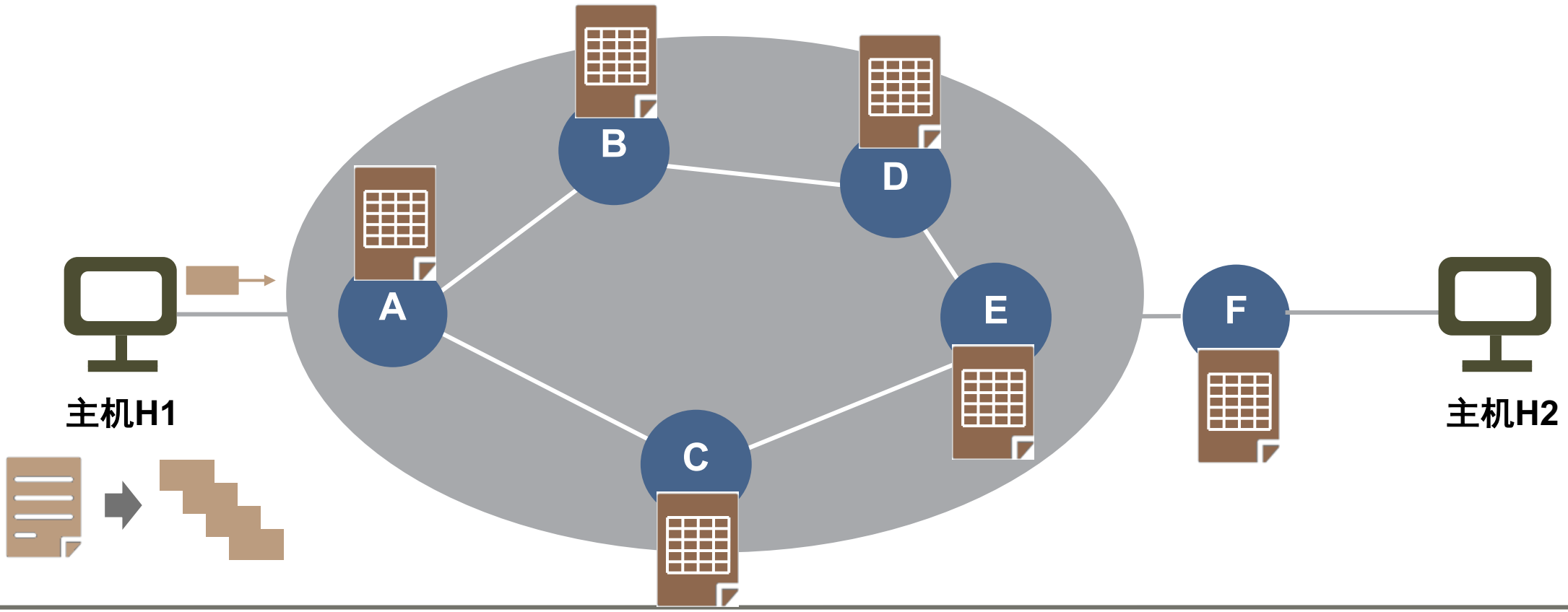
- 尝试所有可能路由
- 至少有一个包通过最小跳路由到达
- 所有与源节点连接的节点都被访问

优点

- 具备路由健壮性
- 能发现最好的路径
- 可用来广播重要信息

最短路径选择法

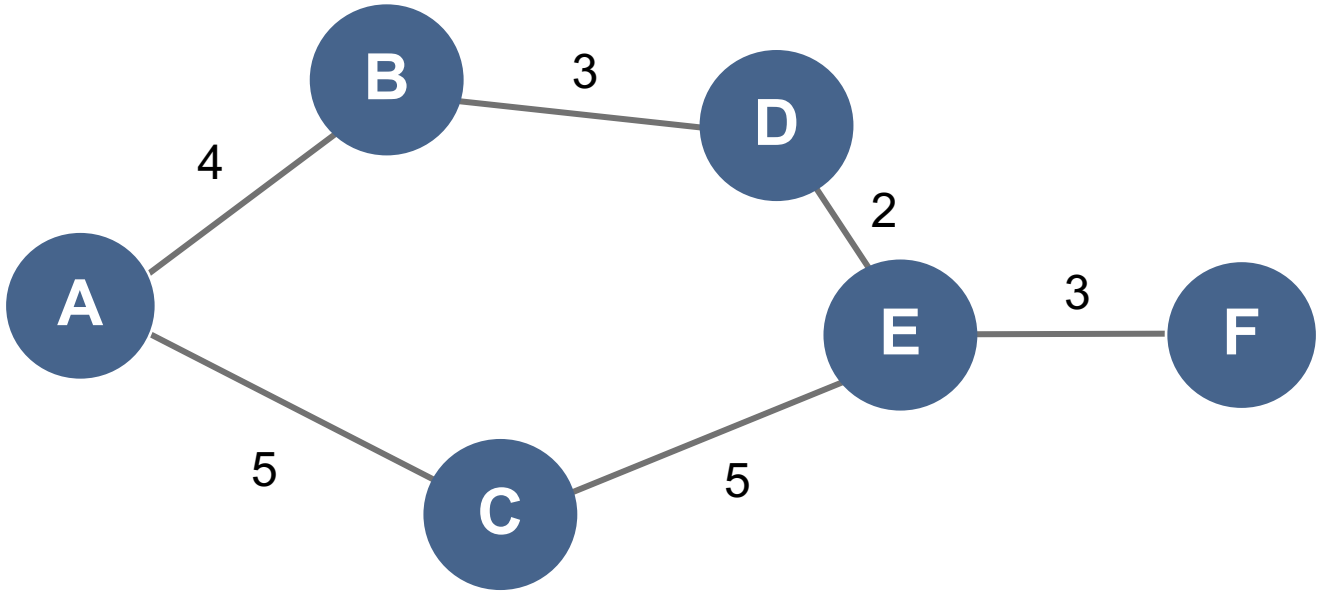
最短路径选择法：每个路由器计算从本地出发到达所有目的地的最“短”路径。根据计算结果，生成路由表。



最短路径计算过程

子网图：节点代表路由器；弧线代表两个路由器之间的一条链路；

Dijkstra算法：找出一个节点到所有其他节点的最短路径



注意：边上的标记统称为链路成本

路径长度计量

路由度量，一种用来计量路径的标准。

可以选择任何一种标准或多个标准的组合来计算从本地出发到所有其他目的地的最“短”路径。

跳计数	物理距离	信道带宽	平均时延	通信成本
包被路由器转发一次就是经过一跳，边的成本为1.	两个路由器之间的物理距离作为边的成本。	连接两个路由器的物理链路的信道带宽。	两个路由器之间信道的平均时延。	连接两个路由器的链路的通信成本

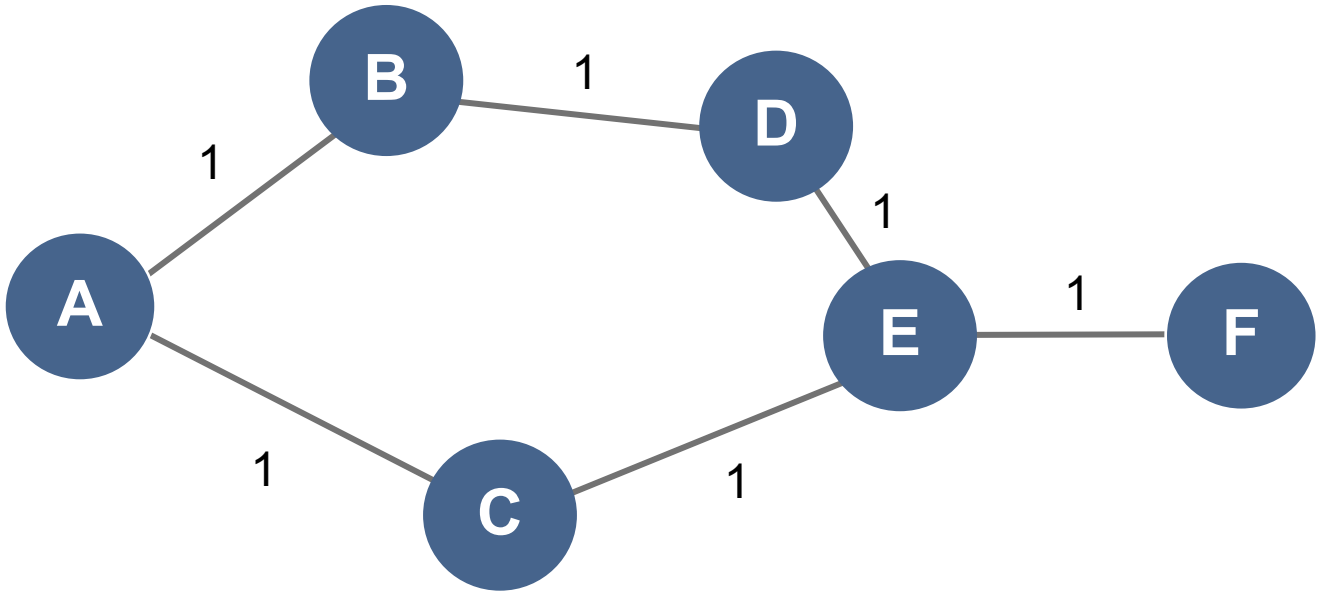
以跳计数为度量的最短路径

假设：连接两个路由器的边的成本为1

试问；路由器A的路由表内容

A的路由表

目标地址	出境线路
A	-
B	B
C	C
D	B
E	C
F	C



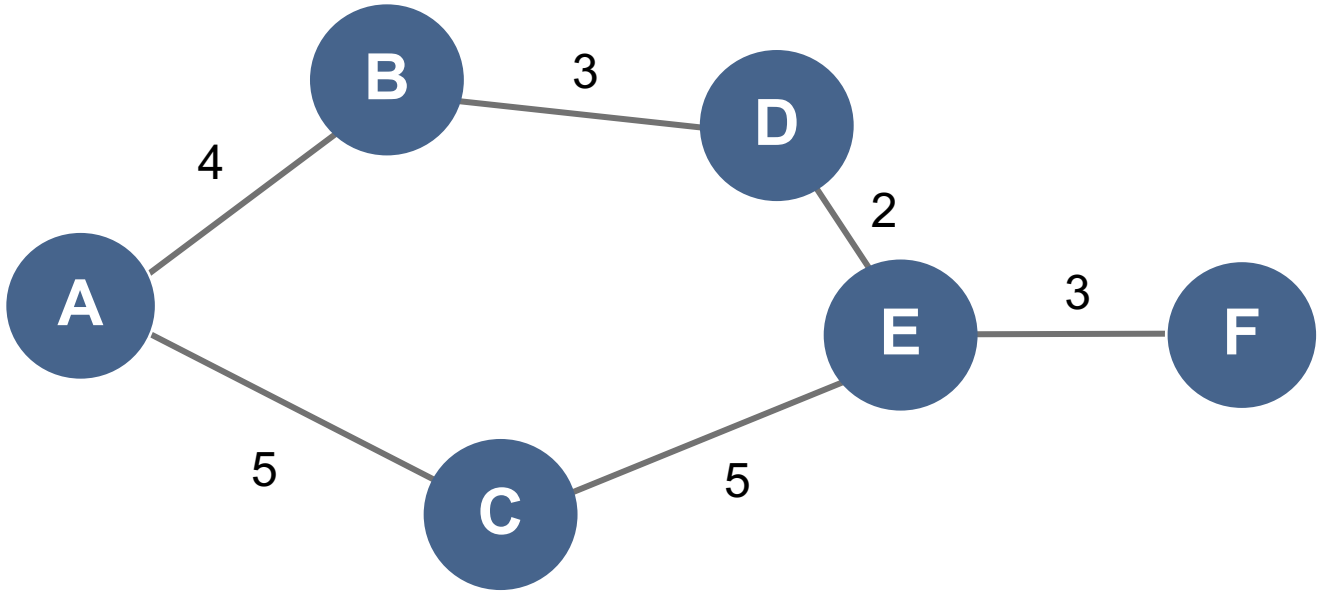
以距离为度量的最短路径

假设：连接两个路由器的边的成本以两点之间的物理距离计量

试问；路由器A的路由表内容

A的路由表

目标地址	出境线路
A	-
B	B
C	C
D	B
E	B
F	B



最短路径特性

优点	缺点
• 可以根据用户需求选择一条“最好”的路径	• 必须事先获得全局的网络拓扑信息 • 必须拥有每条连接两个路由器边的状态信息

泛洪路由vs.最短路径

所有的静态路由方法都无法根据网络状态变化做动态调整。

泛洪路由	最短路径路由
• 可以根据用户需求选择一条“最好”的路径	• 必须事先获得全局的网络拓扑信息 • 必须拥有每条连接两个路由器边的状态信息

链路状态路由算法

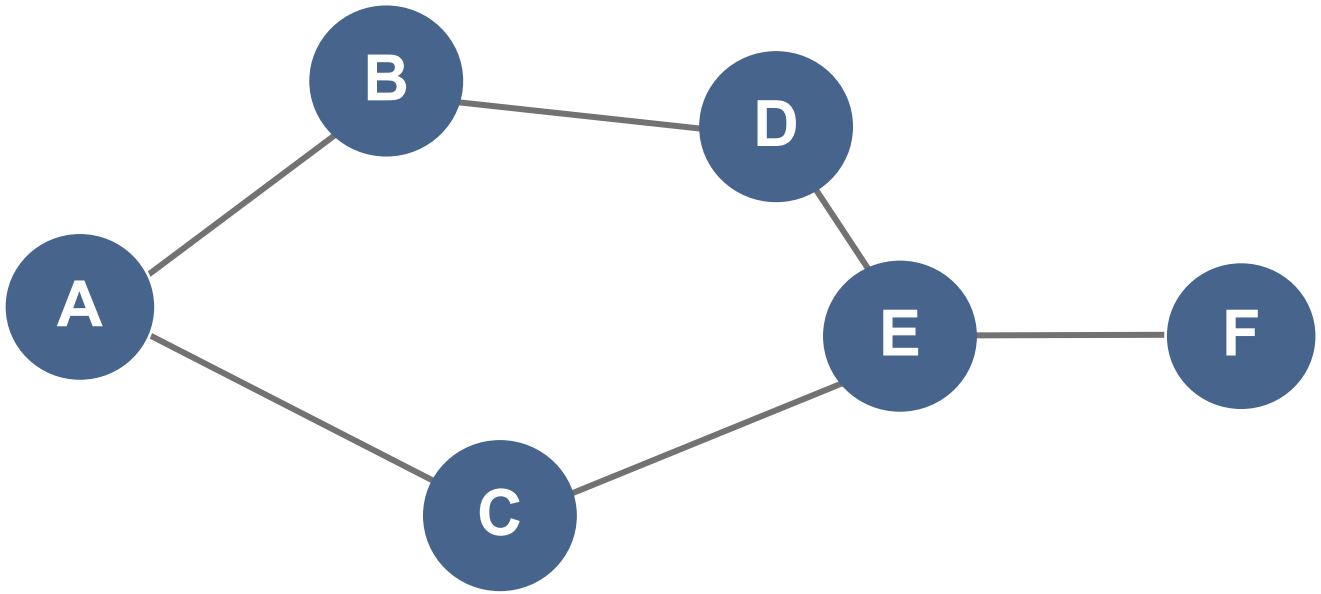
概述

链路状态路由算法

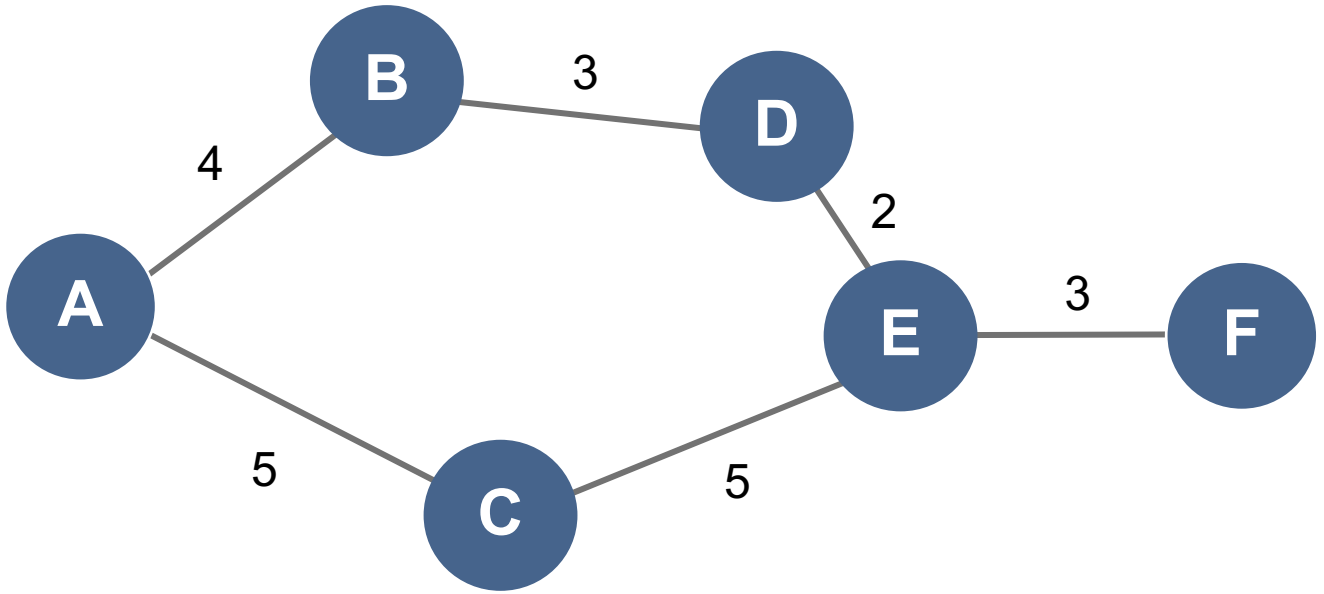
链路状态路由算法：每个节点都有网络完整拓扑图，每个节点维护到邻居的连通性与链路成本。节点向网络中所有其他节点广播自己和邻居的连接信息，每当接收到来自其他节点信息时用Dijkstra算法重新计算路由表。

Dijkstra 算法

- 计算从一个节点(源，假设为A)到网络中所有其他节点的最小成本路径。
- 算法迭代进行
- 经第k次迭代后，就能找出到k个目标节点的最小成本路径。



链路状态与链路成本



链路成本：反应路由度量的函数

E = f(距离/时延/成本)

例如，链路成本与链路带宽成反比。带宽越高成本越低→路由选择高容量的路径

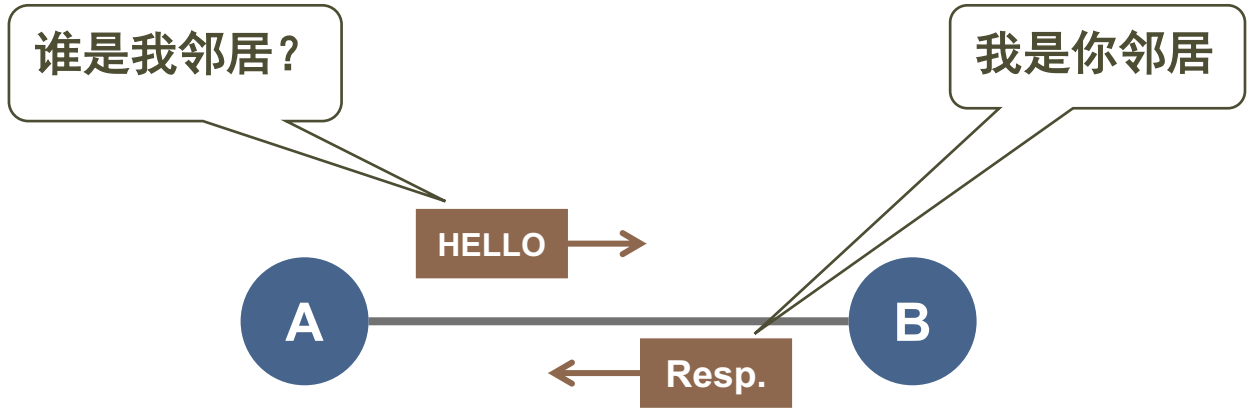
链路时延测量

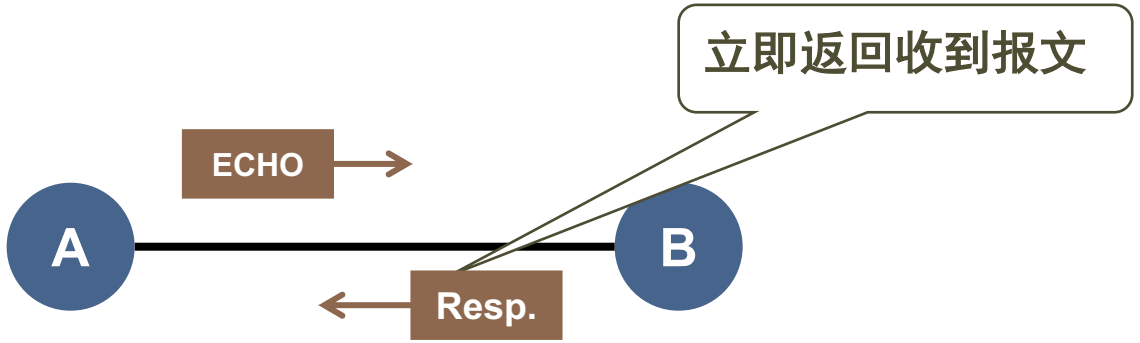
- 给邻居节点发送一个盖上时间戳的包，记录其离开时间
- 收到邻居返回的确认时，计算该包的来回时延

链路状态的获取

①发现邻接节点

在一跳范围内广播一个HELLO消息，邻居的Response消息把自己的地址带了过来。





②测量链路成本

收到ECHO消息后立即返回，用来测试链路的往返时间（时延）

Resp. 响应消息

链路状态的获取（续）

③封装链路状态包

把本地链路（与邻居相连）成本以及邻居ID封装在一个报文中

④广播链路状态信息

发送链路状态包时机

- 定期发送
- 出现重大事件时

- 本机ID：指出链路状态包的发送方
- 序号：标识了源节点发出的链路状态包次序
- 生存期：标识了本链路状态包信息的有效期
- 邻居/成本：表示本机与邻居的链路成本

防止广播通信中的重复包：接收节点可根据(源节点ID，序号)来判定此次入境包是否含有最新链路状态信息。

链路状态路由计算

计算新路由

- 根据接收到的所有节点链路状态包构建表示网络拓扑的子网
- 采用Dijkstra算法计算从本地到所有其他节点的最短路径

A发送的本地链路状态包

A
100
300s
B | 4
C | 5

A收到的来自其他路由器的链路状态包

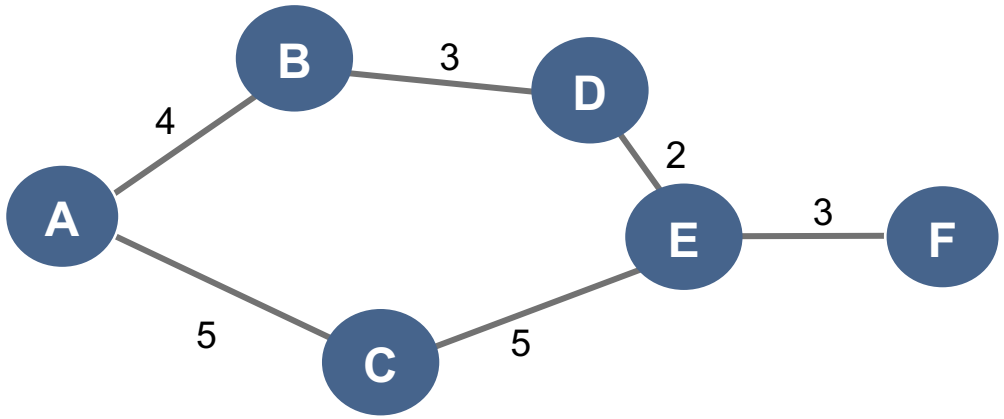
B
100
300s
A | 4
D | 3

C
100
300s
A | 5
E | 5

D
100
300s
B | 3
E | 2

E
100
300s
C | 5
D | 2
F | 3

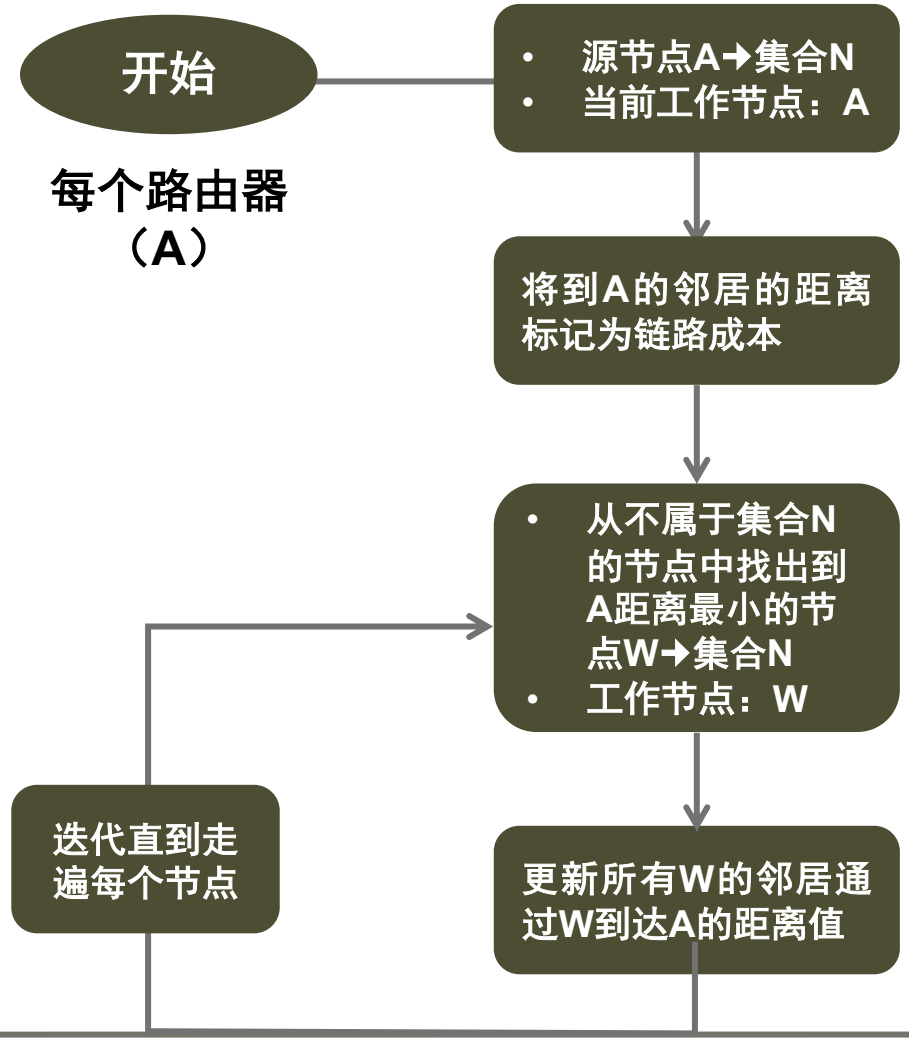
F
100
300s
E | 3



链路状态路由算法

Dijkstra算法

Dijkstra算法流程



C(i，j)：从节点i到j的成本。如果i到j没有直接链路，则为∞
D(v)： 从源节点到目标节点v的当前最小成本
P(v)： 从源节点到目标节点v的当前最小成本路径上前一节点
N： 从源点沿已定义最短路径能到达的节点的集合

第一步初始化工作。在子网中标出所有节点到初始节点A的最短距离，只有与A邻接的节点才知道到A距离。

第二步循坏迭代，直到找出所有节点到A的最短距离：

$$D(v) = \min\{ D(v), D(w)+c(w,v) \}$$

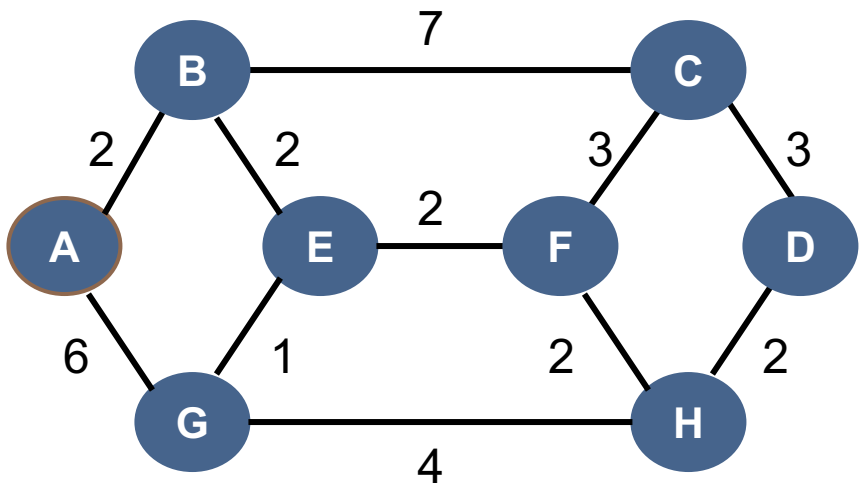
已知到A的距离

经过w到A的距离

Dijkstra算法——初始化

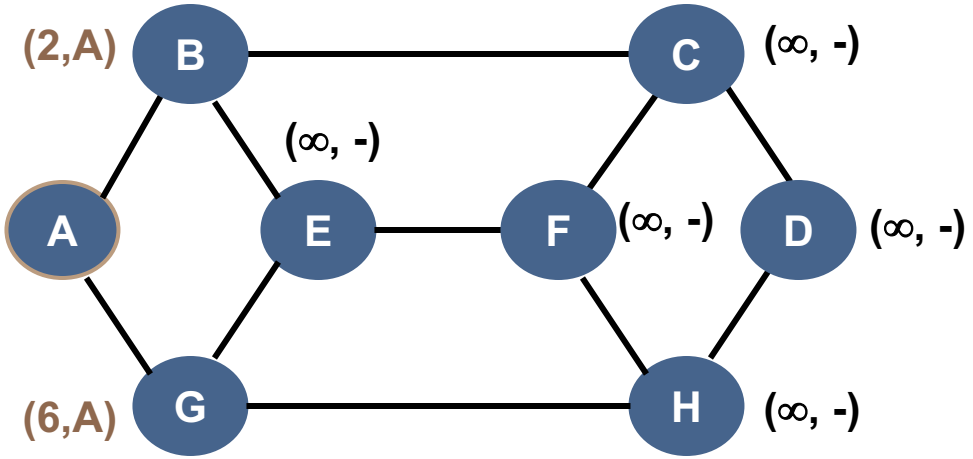
假设：根据各节点广播的路由状态包播，源节点A构造出表示网络拓扑结构的子网图

试问：**Dijkstra**算法过程(以A为例)



- **初始化**
  - **将A加入集合N**
  - **将A邻居到A的距离标为链路成本**

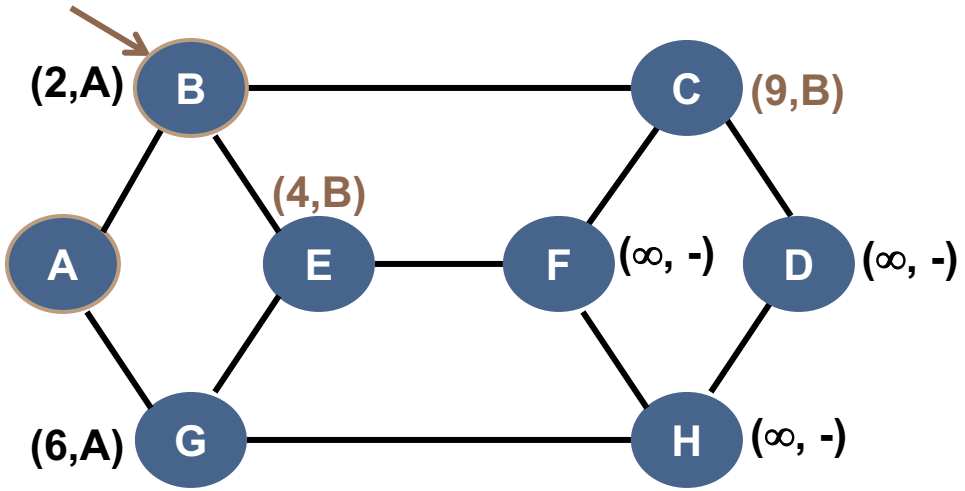
与A邻接的节点B和G到A的最短距离就是链路成本



Dijkstra算法——迭代过程（4-1）

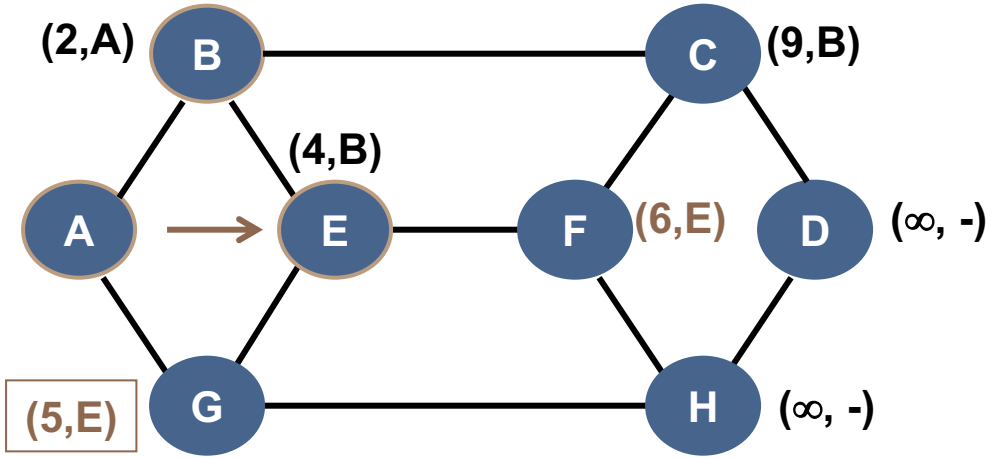
- 循环迭代1
  - 选择到A距离最短的节点B加入N
  - 标值其他节点经B到A的距离

D(C) = min{ D(C), D(B)+c(B,C) }
= min{ ∞, 2+7}



- 循环迭代2
  - 选择到A距离最短的节点E加入N
  - 标值其他节点经E到A的距离

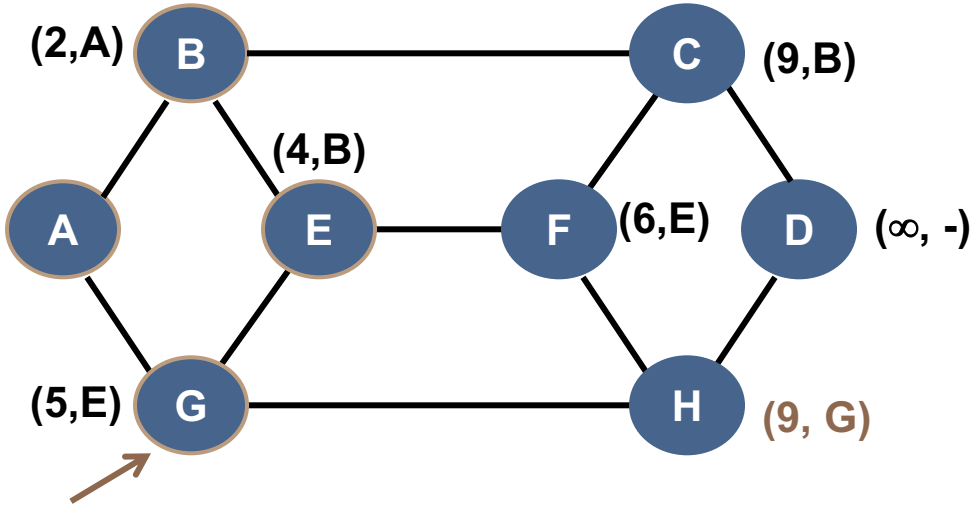
D(G) = min{ D(G), D(E)+c(E,G) }
= min{ 6, 4+1}



Dijkstra算法——迭代（4-2）

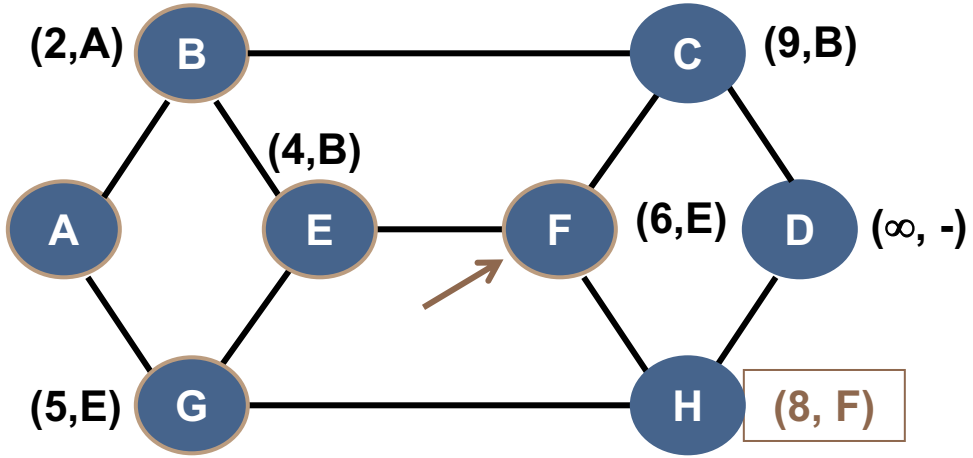
- 循环迭代3
  - 选择到A距离最短的节点G加入N
  - 标值其他节点经G到A的距离

D(H) = min{ D(H), D(G)+c(G,H) }
= min{ ∞, 5+4}



- 循环迭代4
  - 选择到A距离最短的节点F加入N
  - 标值其他节点经F到A的距离

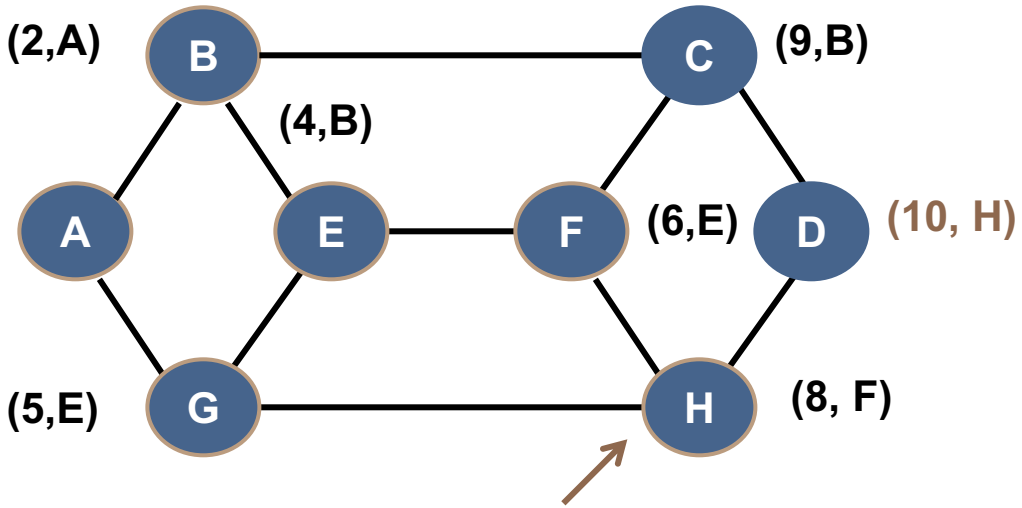
D(H) = min{ D(H), D(F)+c(F,H) }
= min{ 9, 6+2}



Dijkstra算法——迭代（4-3）

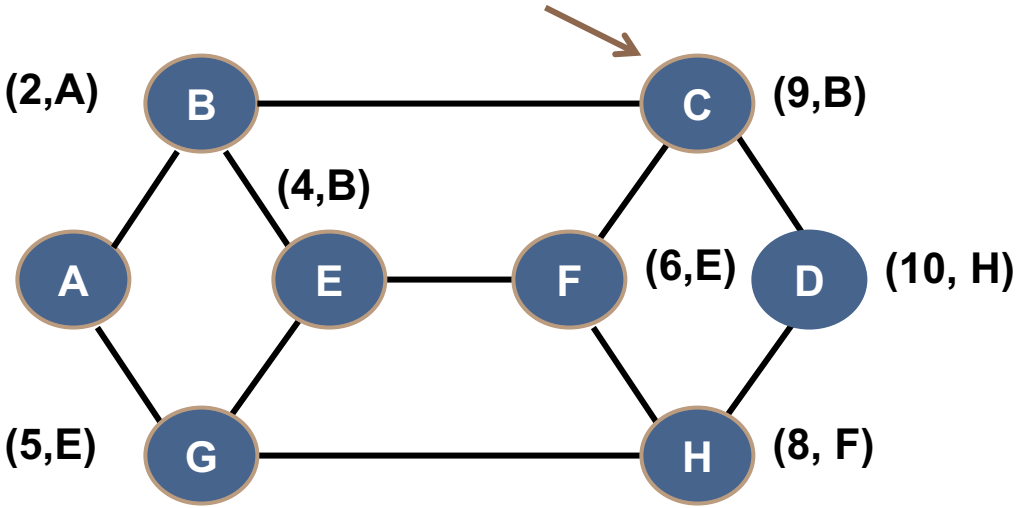
- 循环迭代5
  - 选择到A距离最短的节点H加入N
  - 标值其他节点经H到A的距离

D(D) = min{ D(D), D(H)+c(H,D) }
= min{ ∞, 8+2}



- 循环迭代6
  - 选择到A距离最短的节点C加入N
  - 标值其他节点经C到A的距离

D(D) = min{ D(H), D(C)+c(C,D) }
= min{ 10, 9+3}



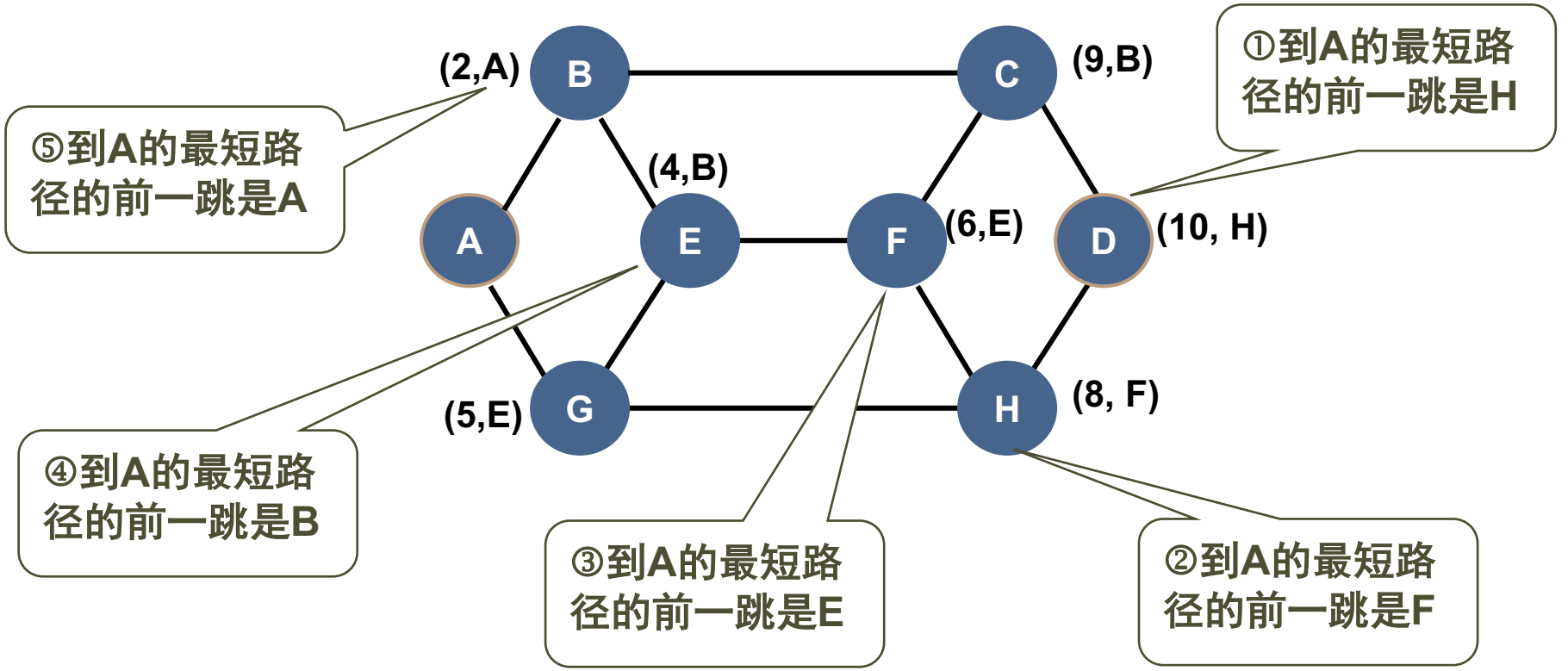
Dijkstra算法——最短路径的确定

计算最短路径：从任意一个目标节点倒着往前推即可获得从源节点A到该节点的一条最短路径。

从A出发到达D的最短路径：A-B-E-F-H-D

A的路由表

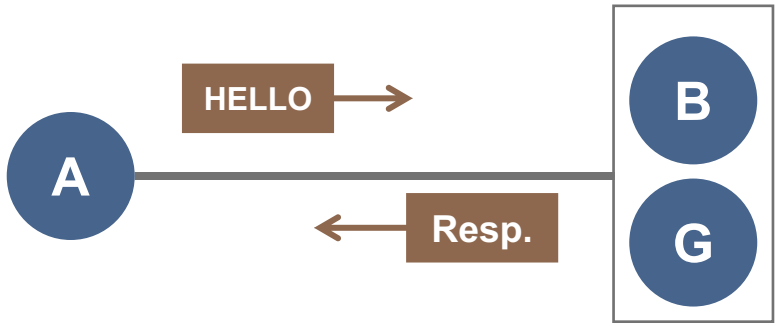
目标地址	出境线路	路径长度
A	-	-
B	B	2
C	B	9
D	B	10
E	B	4
F	B	6
G	B	5
H	B	10



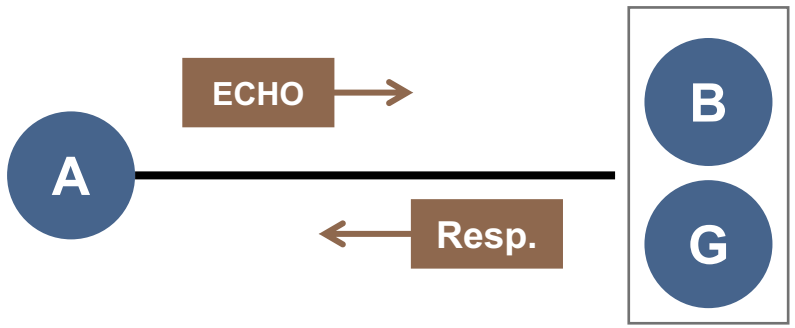
链路状态路由算法示例

发现邻居并测量链路成本

①发现邻接节点

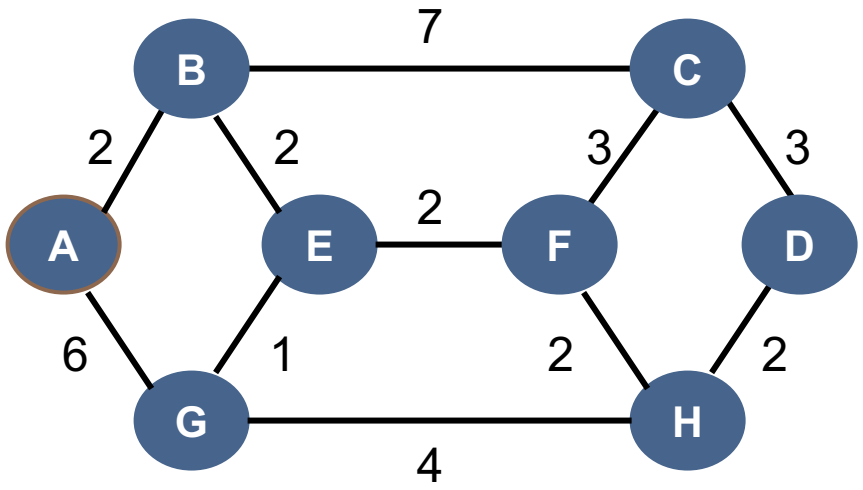


②测量链路成本

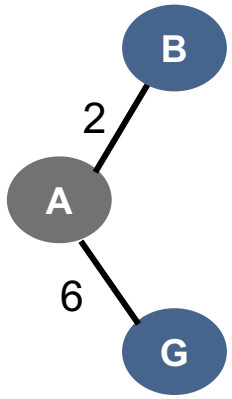


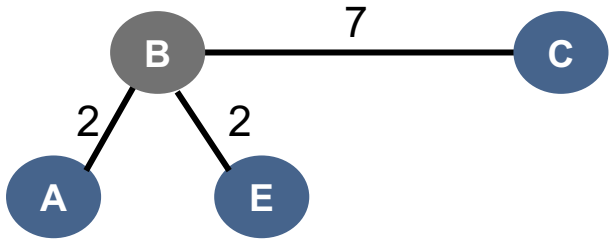
假设：以链路时延作为链路成本的度量值
（链路成本=链路时延）

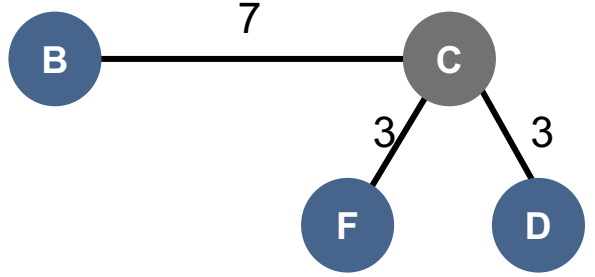
- 链路时延越大，成本越大
- 最短路径为最小时延的路径

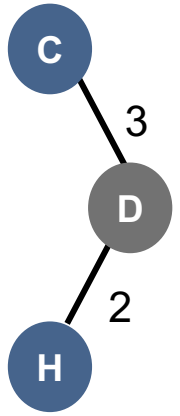


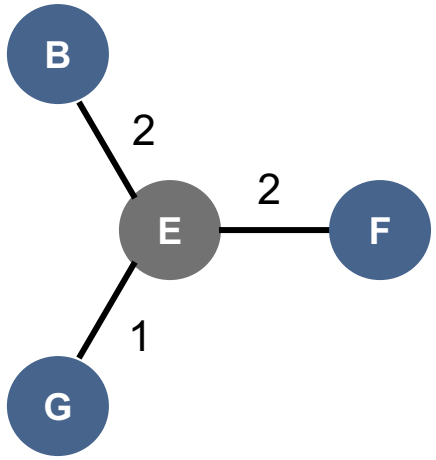
测量与邻居相连的链路状态

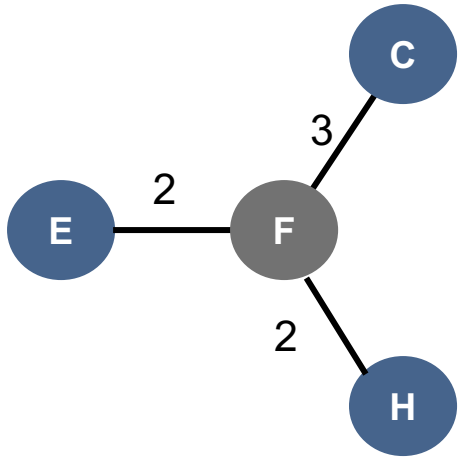


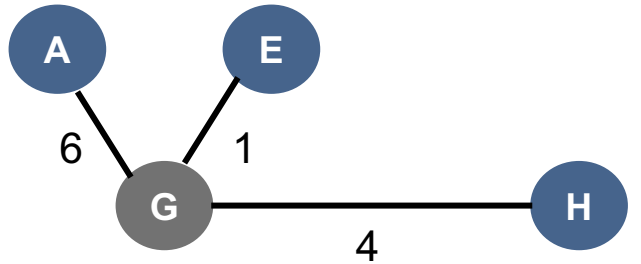


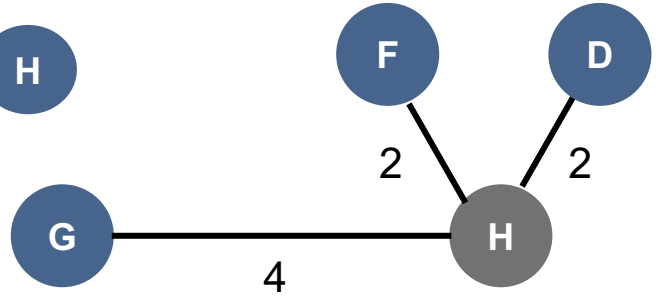












生成并发送链路状态包

③封装链路状态包

- 每个节点根据测量结果生成链路状态包

A
100
300s
B | 2
G | 6

B
100
300s
A | 2
C | 7
E | 2

C
100
300s
B | 7
D | 3
F | 3

D
100
300s
C | 3
H | 2

④广播链路状态信息

- 每个节点根据测量结果生成链路状态包

E
100
300s
B | 2
F | 2
G | 1

F
100
300s
C | 3
E | 2
H | 2

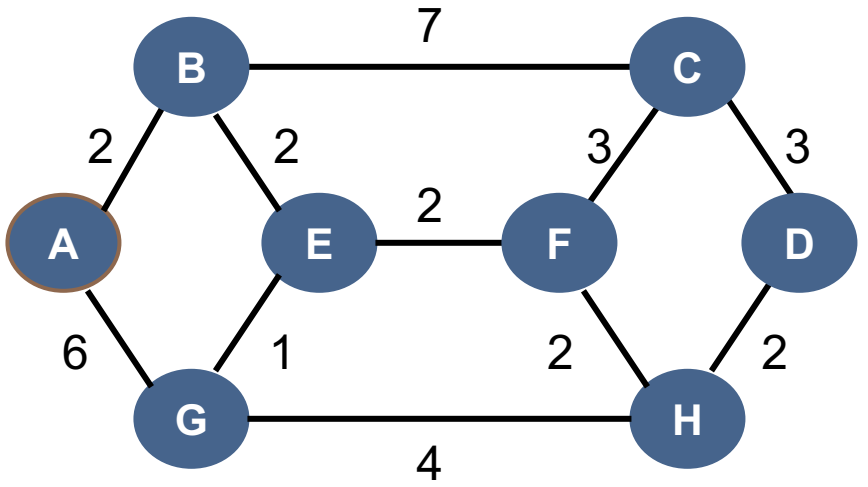
G
100
300s
A | 6
E | 1
H | 4

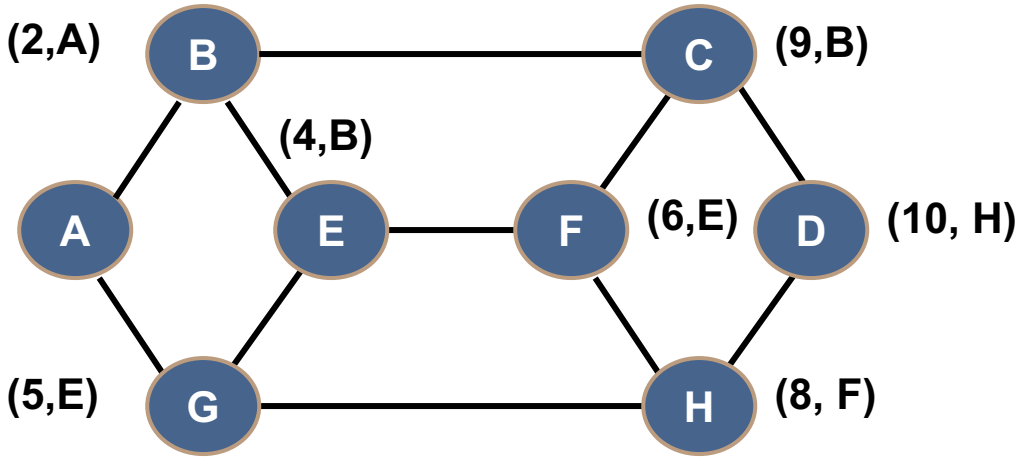
H
100
300s
D | 2
F | 2
G | 4

计算最短路径

⑤计算最短路径

- 根据收到的链路状态包构造出网络拓扑子网图
- 运行Dijkstra算法计算最短路径





从A出发到所有目标节点的最短路径

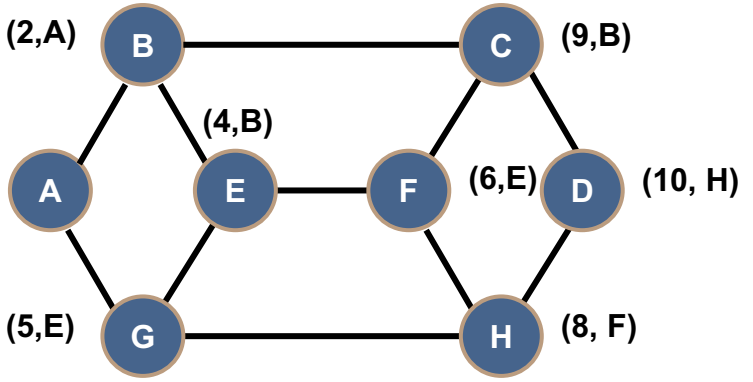
- A-B, 2
- A-B-C, 9
- A-B-E-F-G-D, 10
- A-B-E, 4,
- A-B-E-F, 6
- A-B-E-G, 5
- A-B-E-F-H, 8

生成路由表

A的路由表

目标地址	出境线路	路径距离
A	-	-
B	B	2
C	B	9
D	B	10
E	B	4
F	B	6
G	B	5
H	B	8

A → B → E → F → G → D



从A出发到所有目标节点的最短路径

- A-B, 2
- A-B-C, 9
- A-B-E-F-G-D, 10
- A-B-E, 4,
- A-B-E-F, 6
- A-B-E-G, 5
- A-B-E-F-H, 8

逐跳路由：每个路由器只有从本地出发前往目的地的下一站路由，没有完整的路由表。

链路状态路由算法

VS.

路由震荡

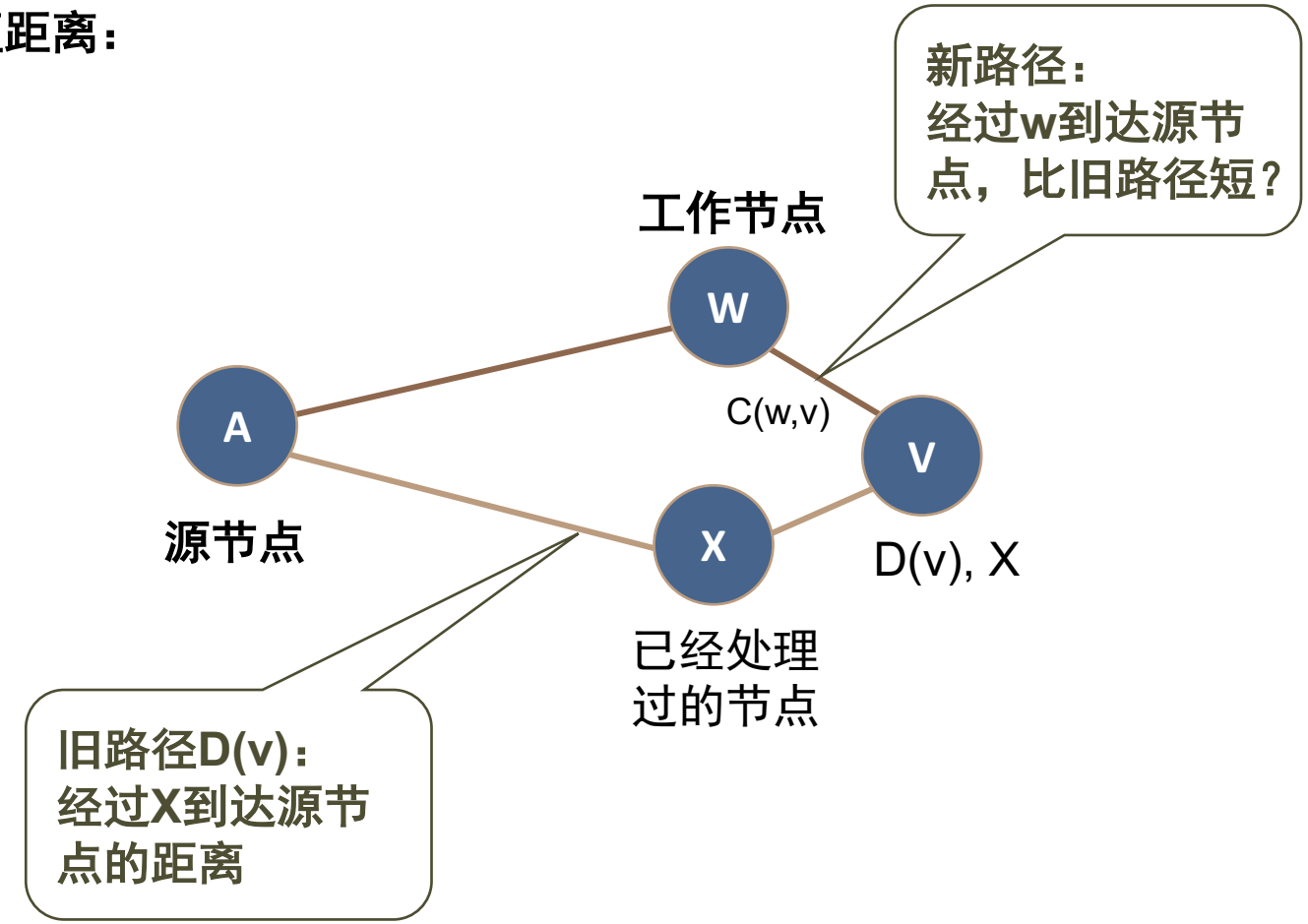
Dijkstra算法的迭代

在循环迭代中，算法试图找出所有节点到A的最短距离：

- 当前工作节点w
- 检查所有w的邻居节点，考察经过w到源节点的距离是否需要更新

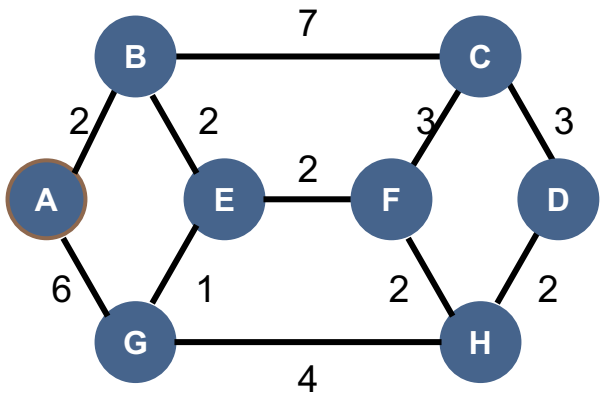
$$D(v) = \min\{ D(v), D(w)+c(w,v) \}$$

- D(v)：w的邻居通过其他节点到达源节点的距离
- C(w, v) ： w和v的链路成本



通过迭代发现更好的路径

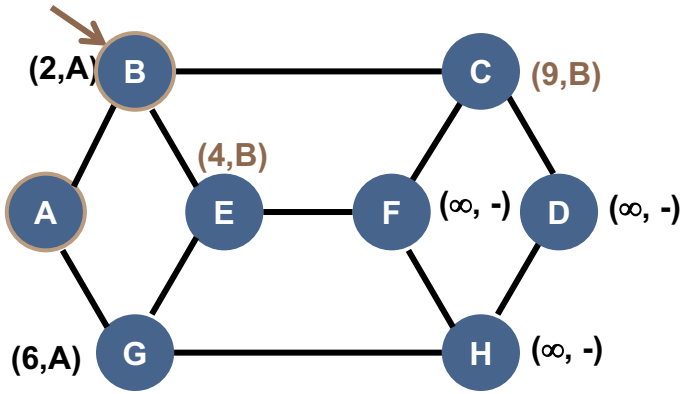
算法运行的基础：完整的网络拓扑信息



子网图

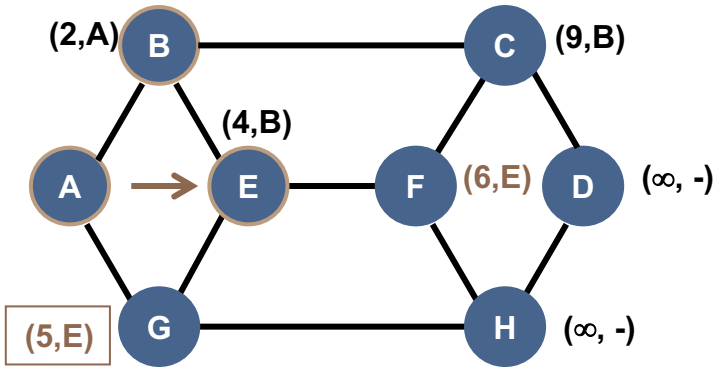
t1时刻

- 节点G到源节点A的最短距离是6
- 从A出发到目标节点G的路径是：A-G



t2时刻

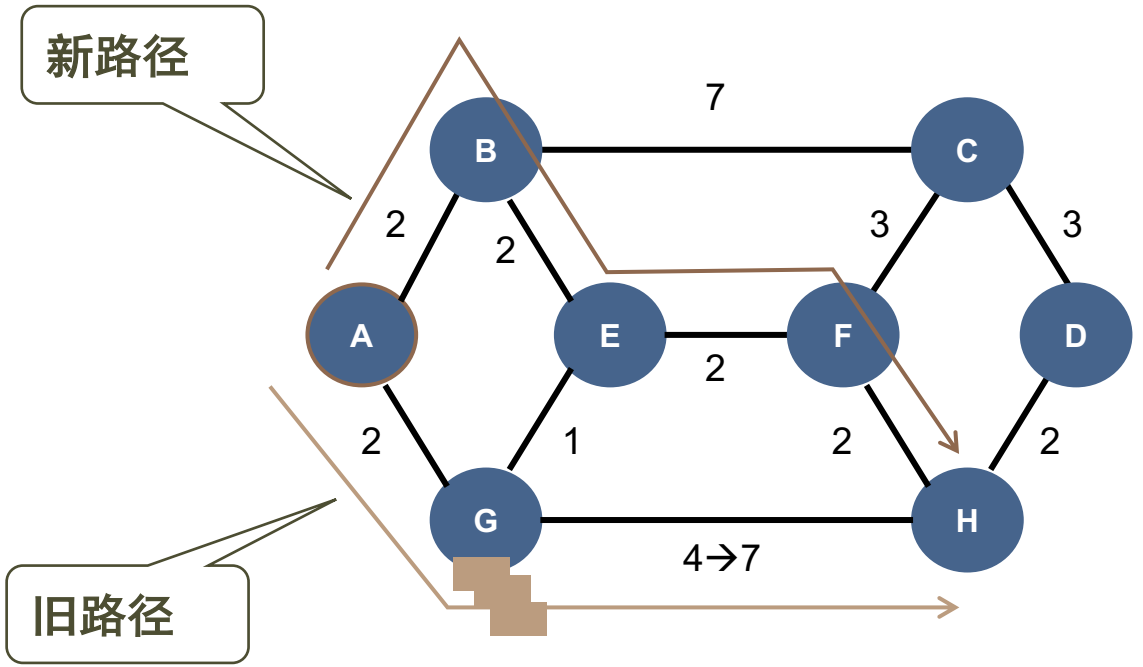
- 旧路径：A-G，长度6
- 新路径：G通过邻居E到A的距离为E到A的距离加上G到E的链路成本，长度为5
- 更新从A出发到目标节点G的路径：A-B-E-G



以链路时延作为链路成本

如果链路成本表示流量负载，那么在通过ECHO消息测量链路成本时就考虑了负载因素。

优点	缺点
在相同链路条件下，选择轻负载的路径可减少排队时延	有可能造成路由来回震荡



- t0：A到H的最短路径选择了A-G-H
- t1：A到H的最短路径更新为A-B-E-F-H

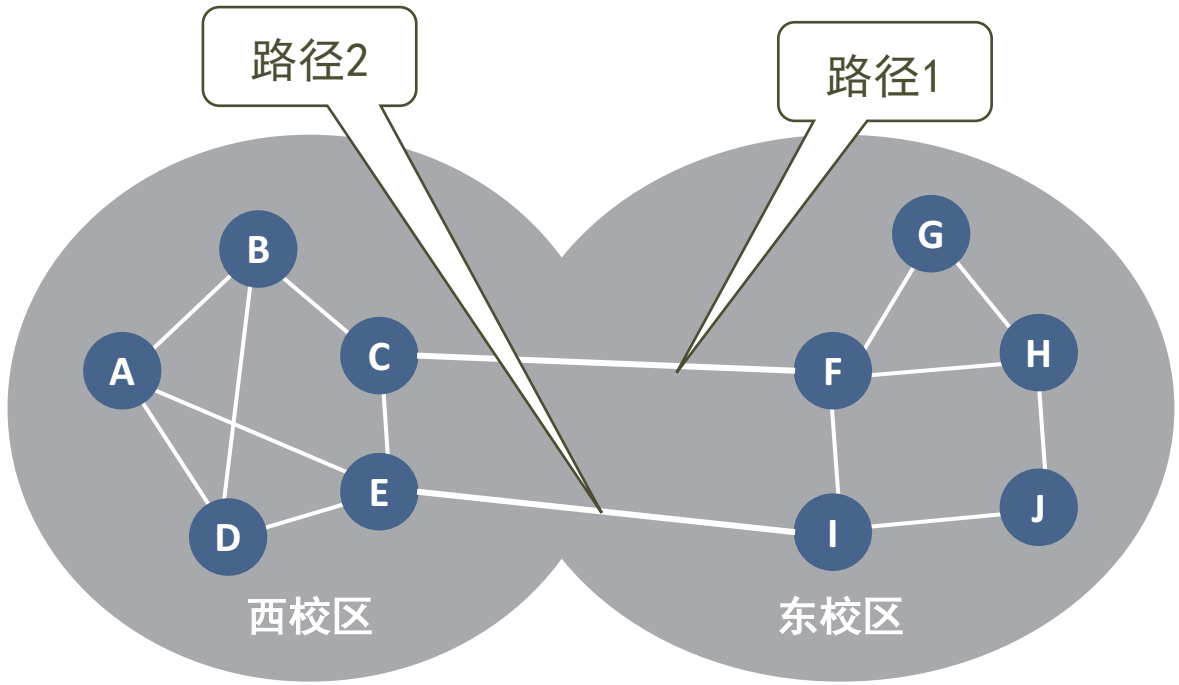
路由来回震荡问题

路由震荡：路由随着链路成本的变化来回切换。

- **初始时东西园区之间的包选择CF（或EI），所有包都通过C/F转发➔C/F上的排队时延增加**
- **路由更新时把路径切换至EI（或CF），所有的包都通过E/I转发➔E/I上的排队时延增加**
- **。。。。。。**
- **两个校区之间的路由在路径1和路径2之间来回切换，带来了路由的不稳定**

假设：某个大学由两个校区组成(如图校园网)。

- 校园网的路由器采用了基于链路状态路由算法的路由协议
- 链路成本代表了链路时延



距离矢量路由算法

概述

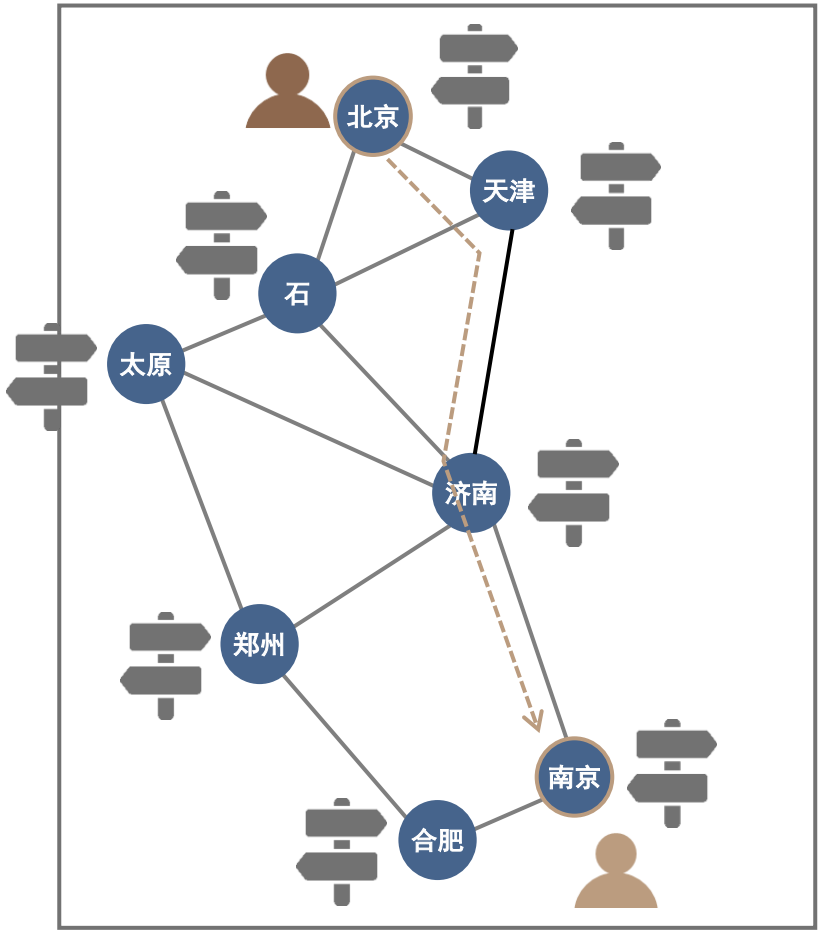
没有地图如何旅行

假设

- 出发地北京，目的地南京
- 没有同行者商量
- 每个交叉路口有方向指示牌

只要路标上的信息是一致的和完备的，小明一定能够从北京出发到达目的地南京，而且他走的是一条最好的路径。

旅行者小明 = 网络层的包
旅行路线 = 包的路径
旅游咨询台 = 沿途路由器
交叉路口路标 = 路由器路由表

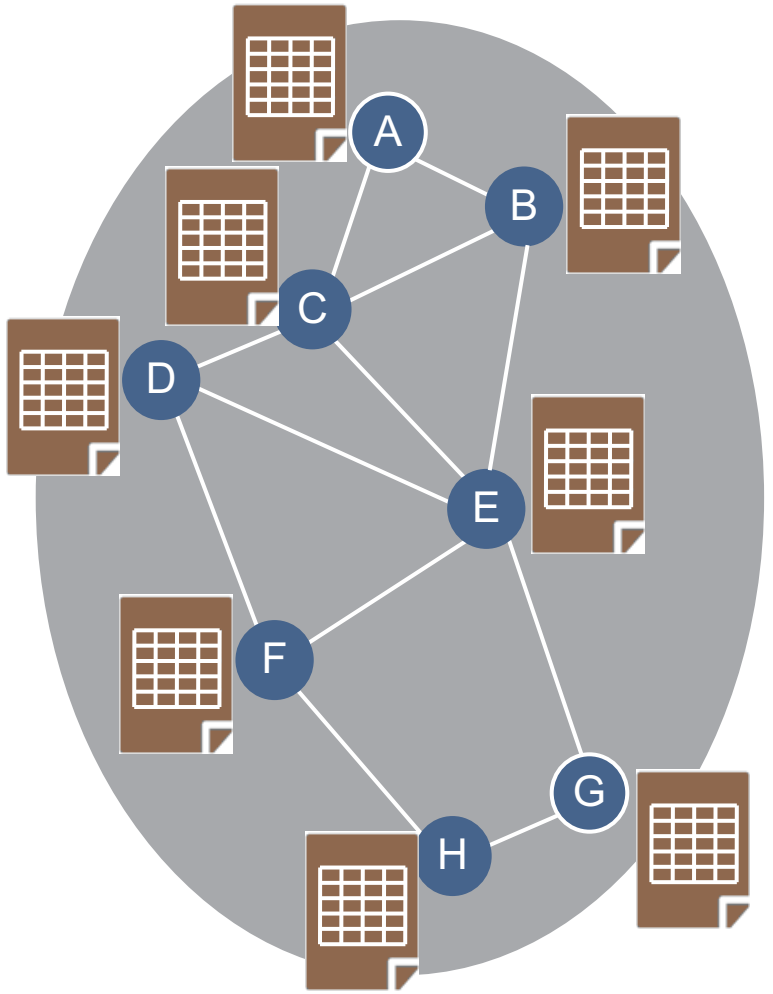


没有网络完整拓扑信息下传递包

假设

- 每个包有完整的目标地址
- 每个包独立选择下一跳
- 只能依赖路由表

? 没有完整的网络拓扑信息，如何计算路由

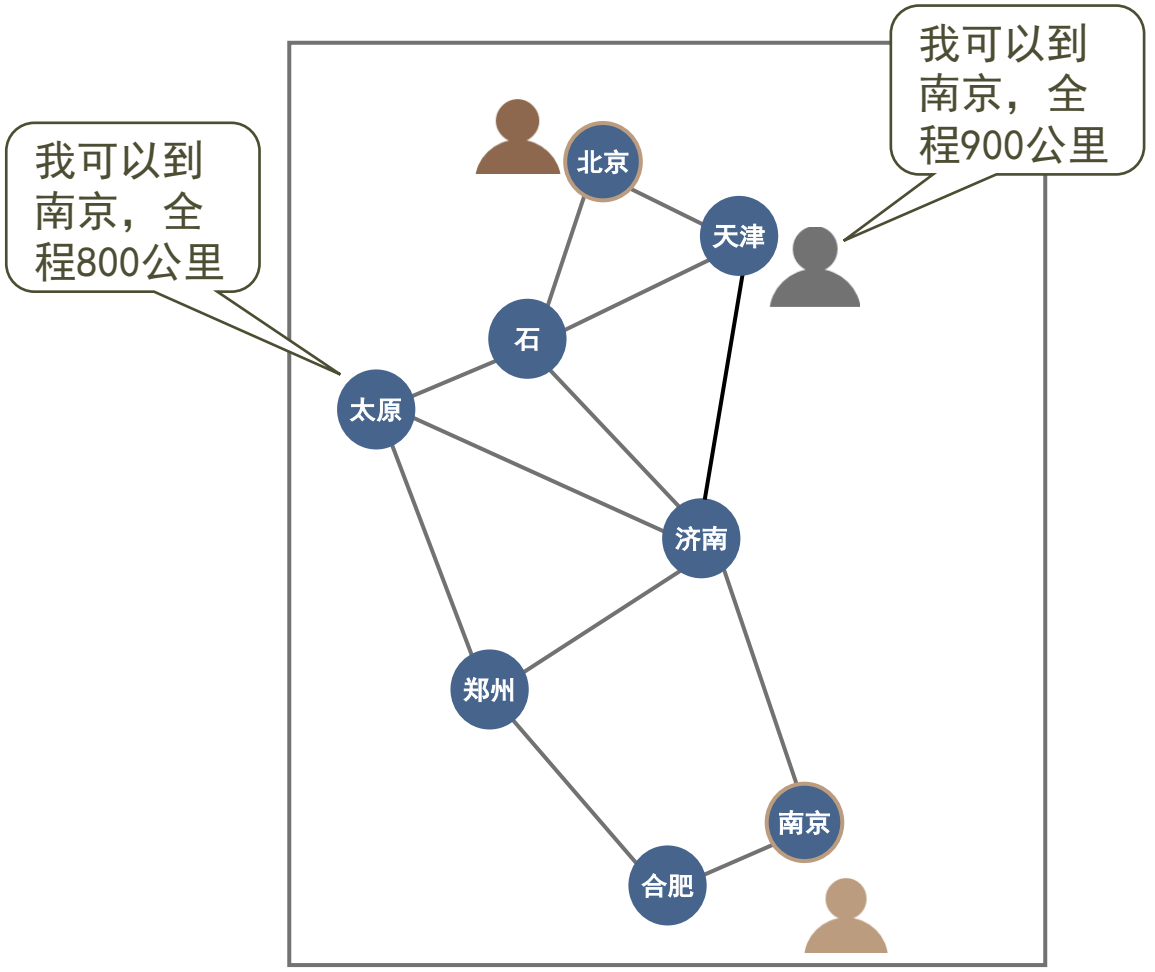


只要路由表的信息是一致的和完备的，按照路由表转发包就一定能将包传递到目的地，而且这是一条最好的路径。

距离矢量路由算法（DV）

基本思想

只要一个节点不是孤立点，就可以通过和邻居交换路由信息来获得网络全部的路由信息。



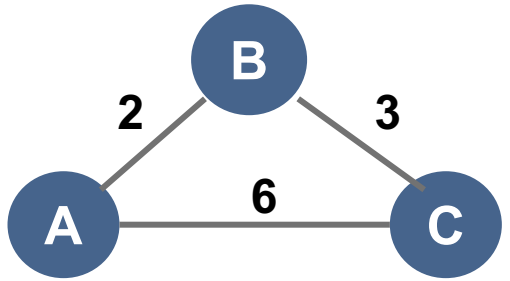
假设：小明要去南京看小芳，没有地图，不知道怎么去。

- 小明有个同学小强在天津
- 小强说“从我这里出发可以到南京，900公里”
- 石家庄同学说“从我这里出发可以到南京，800公里”

试问：小明怎么做？

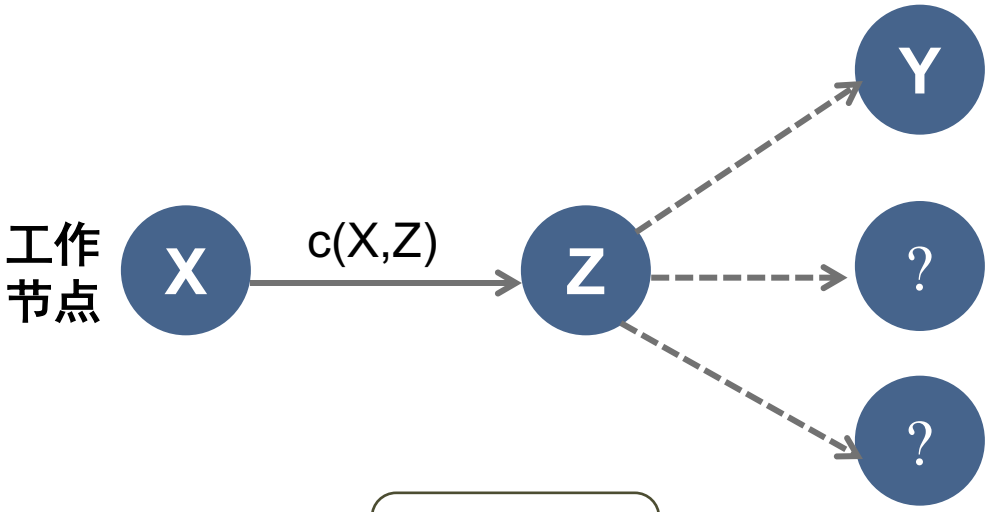
距离矢量算法特点

分布的	迭代的	异步的
••每个节点接收来自与其直接邻接节点的路由信息，并执行路由计算 ••将计算结果回传给直接邻接的节点	••计算过程循环进行 ••直到相邻节点没有可交换的路由信息为止	••并不要求所有节点相互锁步操作
通过和邻居节点交换路由信息来计算全网的路由	当各节点的路由稳定后，算法收敛。	A，B和C各自独立运行路由算法



? A有两个邻居，A到C应该走哪条路？

距离矢量算法——距离



假设：X和Z是邻居，连接两点的边成本为c(X, Z)

试问：X经邻居Z到达Y的距离，表示为

$D^x(Y, Z)$

X本地测量获得

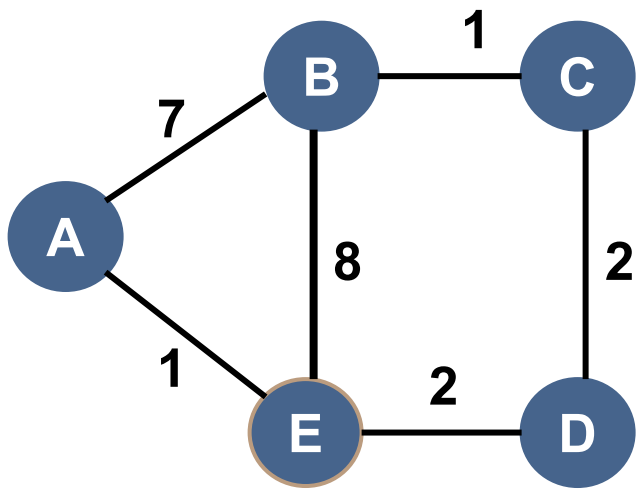
从Z广播的路由报文中获得

$$D^x(Y，Z) = c(X，Z) + \min_w \{D^z(Y，w)\}$$

节点X经过Z到达Y的距离 = X到邻居Z的距离 + Z到Y的最短距离

w为Z的所有直接邻居(包括X)

距离矢量算法——距离矩阵



E经过D到达A的距离：

$D^E(A,D) = c(E,D) + D^D(A, w)$

$= 2+3 = 5$

节点E的距离表
（从本地到达所有目的地距离）

目标节点	经过邻居节点的成本 A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

所有抵达A的路径

所有抵达B的路径

所有抵达C的路径

所有抵达D的路径

距离矢量算法——生成路由表

节点E的距离表（从本地到达所有目的地距离）			
目标节点	经过邻居节点的成本		
	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

获得最小距离的那个邻居即是路由表中最短路径的下一跳

节点E的路由表

目标地址	出境线路	路径距离
A	A	1
B	D	5
C	D	4
D	D	2

距离矢量路由算法

VS.

算法过程

70/100/140

167/169/172

142/104/78

139/0/18

148/7/9

138/139/133

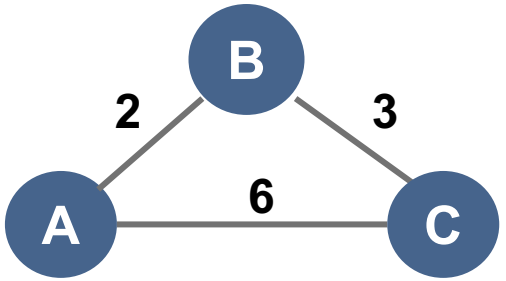
187/156/127

76/77/50

114/114/114

距离矢量算法的数据结构

主要数据结构	节点信息资源	距离矢量算法
••每个节点维护一张经过所有邻居到全部目的地的距离表（距离矩阵）。	••与其直接相连的链路(本地链路)的成本 ••来自邻接节点的路由信息	••用估算延迟作为性能指标 ••基于Bellman-Ford算法



① 计算距离矩阵

计算从本地出发到达所有目的地的全部可达路径，需要本地链路信息和邻居的路由信息。

② 生成路由表

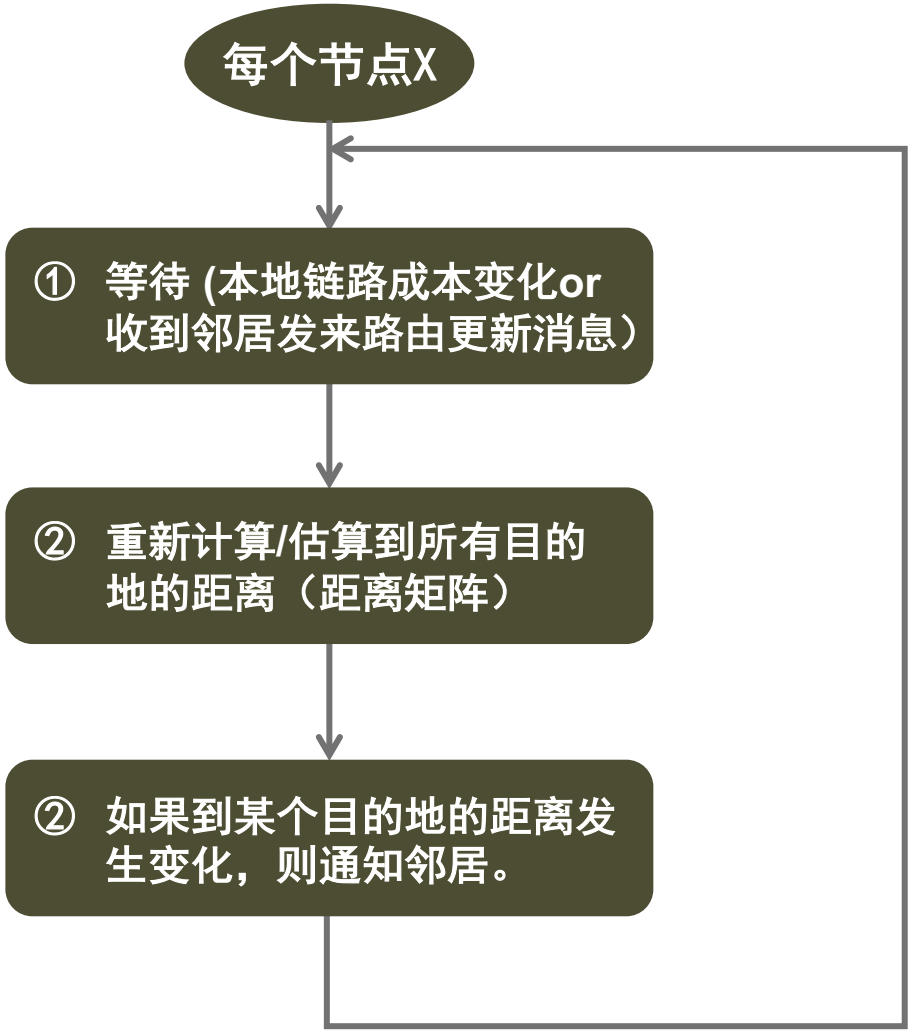
距离矢量算法过程

迭代的条件

- 本地链路成本发生变化
- 收到邻居发来的距离更新消息

迭代更新路由

- 每个节点仅当自己的距离发生变化时才通知邻居
- 如果需要其邻居再通知邻居的邻居



距离矢量算法示例

假设：如图所示的网络中路由器A, B, C运行基于距离矢量路由算法的路由协议

考察：A, B, C路由表的变化

- A, B, C各自独立运行路由协议，一旦满足迭代条件就重新计算路由。
- 注意：路由表是根据距离矩阵计算获得的

- **t0：路由稳定后的各节点路由表**

A路由表

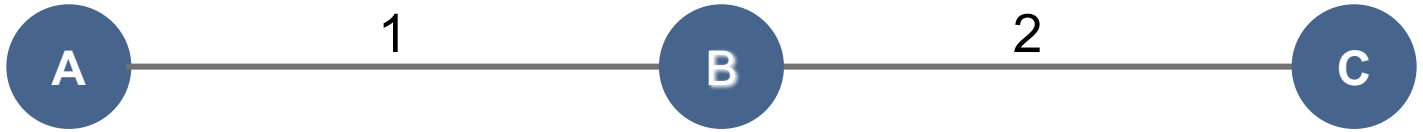
目的地.	下一跳	度量
A	A	0
B	B	1
C	B	3

B路由表

目的地.	下一跳	度量
A	A	1
B	B	0
C	C	2

C路由表

目的地.	下一跳	度量
A	A	3
B	B	2
C	C	0

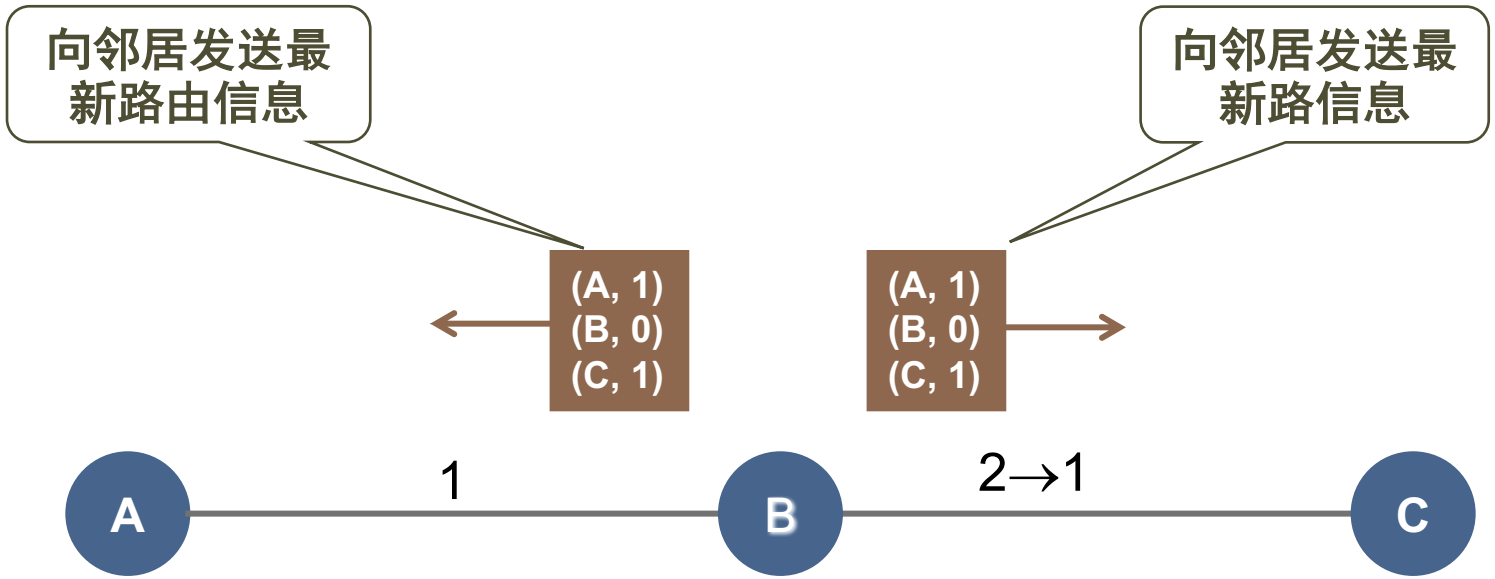


注：From Tobias Schwaegli

向邻居发送最新路由信息

- t1：B-C的链路成本发生变化

  B重新计算路由，并将计算结果发送给自己的邻居（A,C）。

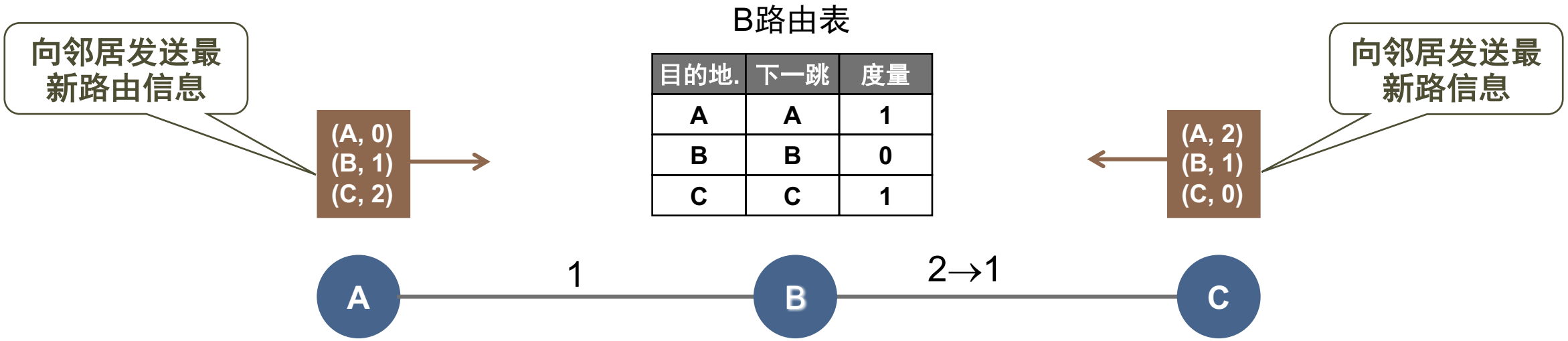


注：From Tobias Schwaegli

迭代时重新计算路由

- A和C收到来自B的路由信息后，发现B到目的地C/A的路径长度发生了变化，于是
  ① 重新计算路由
  ② 发现到B的路由有了变化，把最新路由发给邻居（B）

- B收到来自A和C的路由信息后，检查经过邻居到达B和C的路径是否比已有的更好
  ① A到C的距离是2，加上B-A链路成本1，经过A到达C的路径长度为3，比已有的差，不更改到C的路由
  ② 同理，B放弃经过C抵达A的路由，维持到A的原路由



注：From Tobias Schwaegli

算法收敛的条件

- t2：路由稳定后的各节点路由表

当所有节点（A, B, C）把最新路由表发送给邻居后，没有导致邻居更新路由，至此路由达到稳定，算法收敛。

A路由表

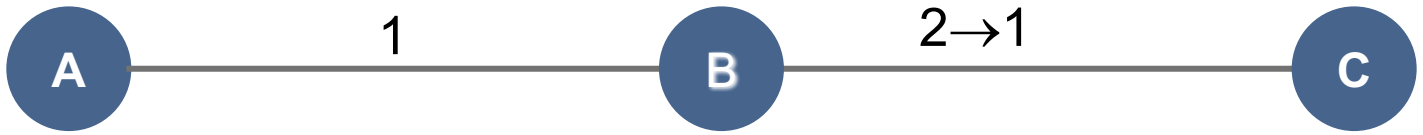
目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2

B路由表

目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1

C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0



注：From Tobias Schwaegli

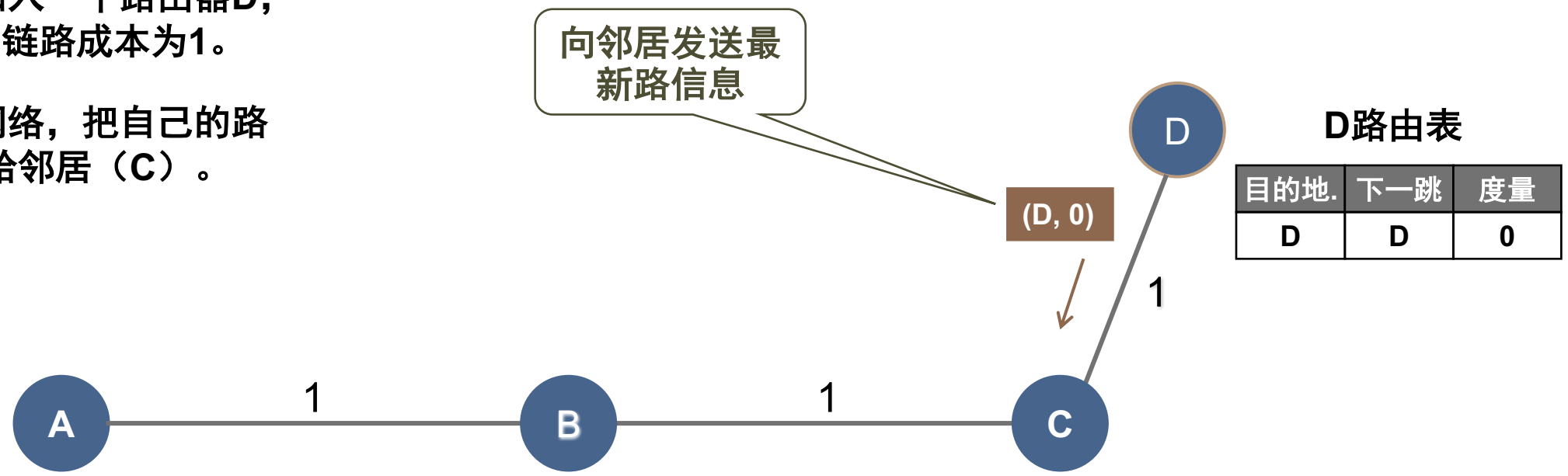
距离矢量路由算法

VS.

算法特点

距离矢量算法优点

- t3：网络新加入一个路由器D，与C是邻居，链路成本为1。
  - D刚加入网络，把自己的路由信息发给邻居（C）。



D路由表

目的地.	下一跳	度量
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2

B路由表

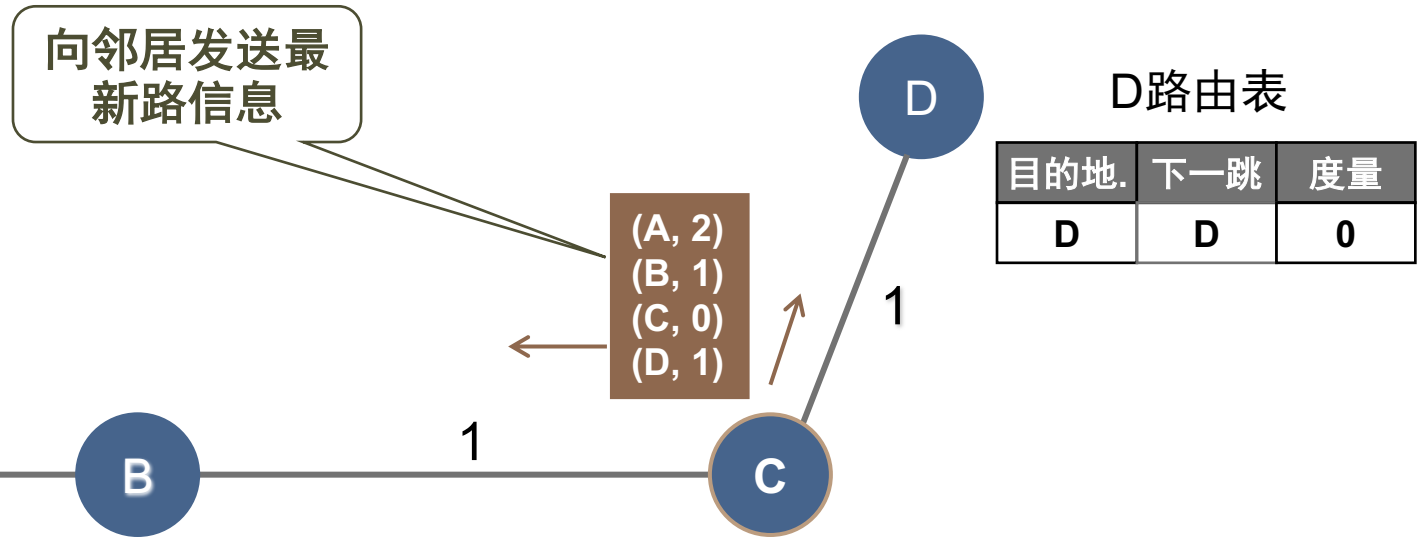
目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1

C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0

距离矢量算法优点

- t4：C收到D的路由信息
  - C发现多了一条抵达D的新路由
  - C在路由表中添加这条新路由
  - C把新的路由表发给邻居(B,D)



D路由表

目的地.	下一跳	度量
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2

B路由表

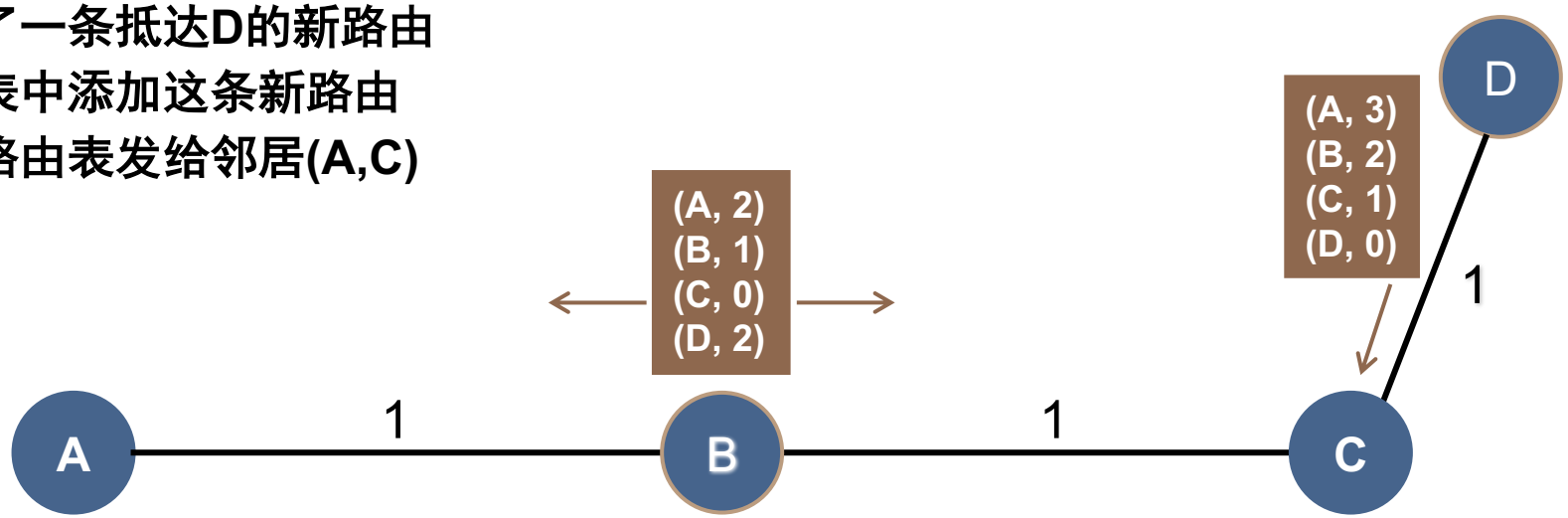
目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1

C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0
D	D	1

距离矢量算法优点

- t5：B和D收到C的路由信息后
  - B发现多了一条抵达D的新路由
  - B在路由表中添加这条新路由
  - B把新的路由表发给邻居(A,C)



D路由表

目的地.	下一跳	度量
A	A	3
B	B	2
C	C	1
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2

B路由表

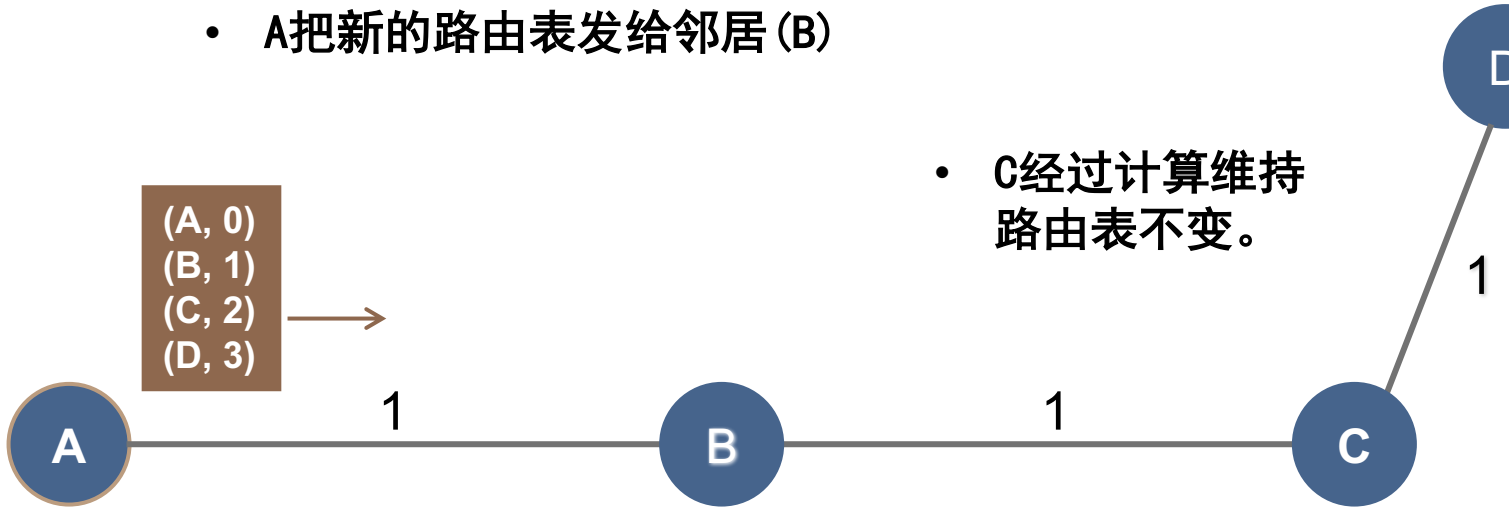
目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1
D	D	2

C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0
D	D	1

距离矢量算法优点

- t6：A和C分别收到B和D的路由信息后
  - A发现多了一条抵达D的新路由
  - A在路由表中添加这条新路由
  - A把新的路由表发给邻居(B)
  - C经过计算维持路由表不变。



D路由表

目的地.	下一跳	度量
A	A	3
B	B	2
C	C	1
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2
D	D	3

B路由表

目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1
D	D	2

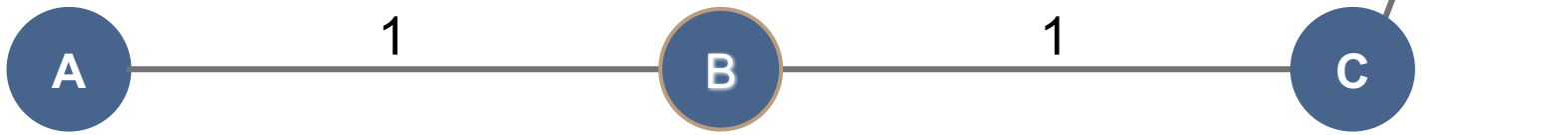
C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0
D	D	1

距离矢量算法优点

- t7：B收到来自A的路由信息后

- B发现经过A到达所有目的地的路由都不如原来的路由，维持路由表不变。



D路由表

目的地.	下一跳	度量
A	A	3
B	B	2
C	C	1
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2
D	D	1

B路由表

目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1
D	D	2

C路由表

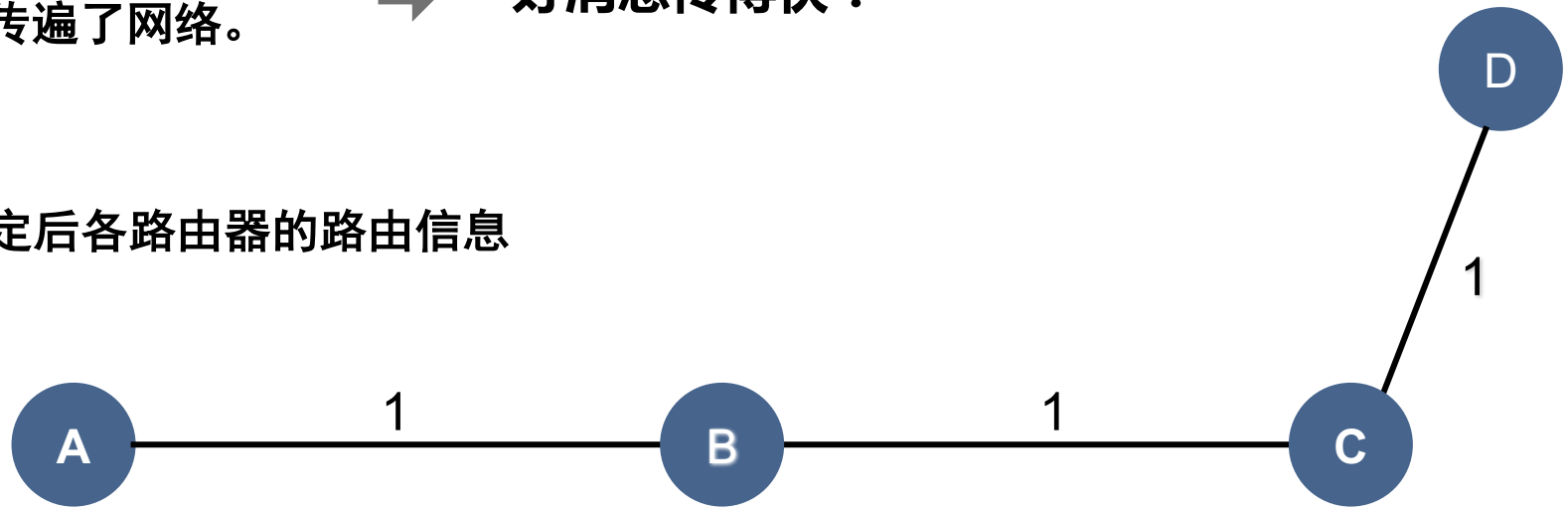
目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0
D	D	1

距离矢量算法优点

经过有限的几次传播，新节点和新路由便传遍了网络。

➡ 好消息传得快！

- t8：路由稳定后各路由器的路由信息



D路由表

目的地.	下一跳	度量
A	A	3
B	B	2
C	C	1
D	D	0

A路由表

目的地.	下一跳	度量
A	A	0
B	B	1
C	B	2
D	D	1

B路由表

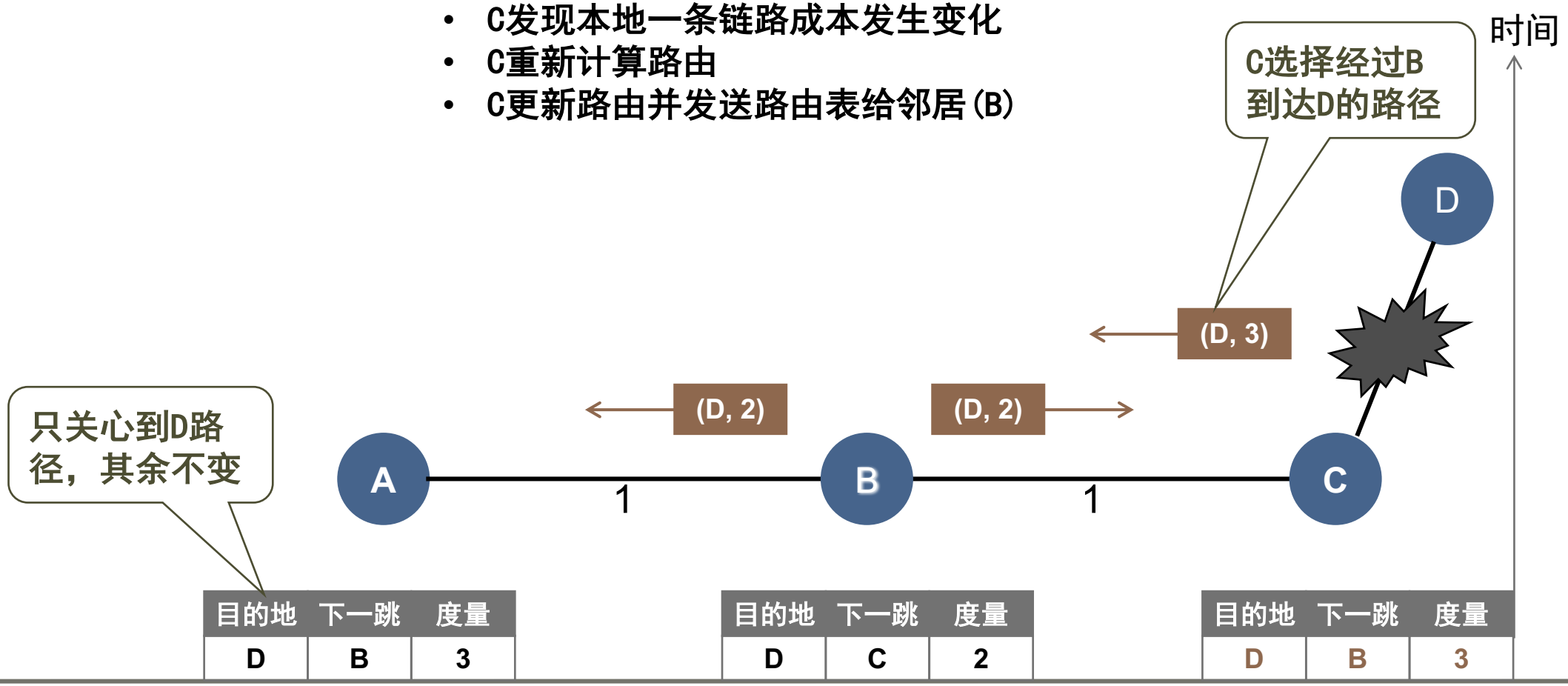
目的地.	下一跳	度量
A	A	1
B	B	0
C	C	1
D	D	2

C路由表

目的地.	下一跳	度量
A	A	2
B	B	1
C	C	0
D	D	1

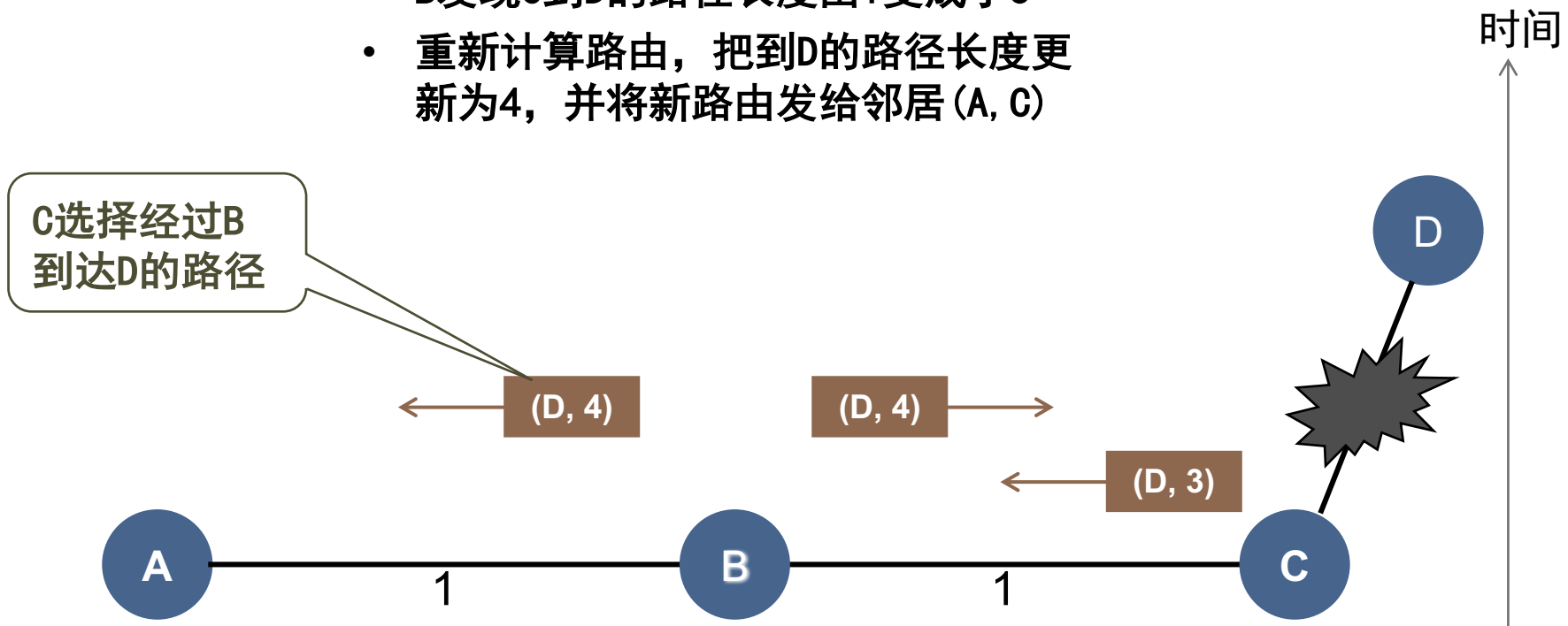
距离矢量算法缺点

- t8：C到D的链路故障
  - C发现本地一条链路成本发生变化
  - C重新计算路由
  - C更新路由并发送路由表给邻居(B)



距离矢量算法缺点

- t9：B收到C的路由包
  - B发现C到D的路径长度由1变成了3
  - 重新计算路由，把到D的路径长度更新为4，并将新路由发给邻居(A, C)



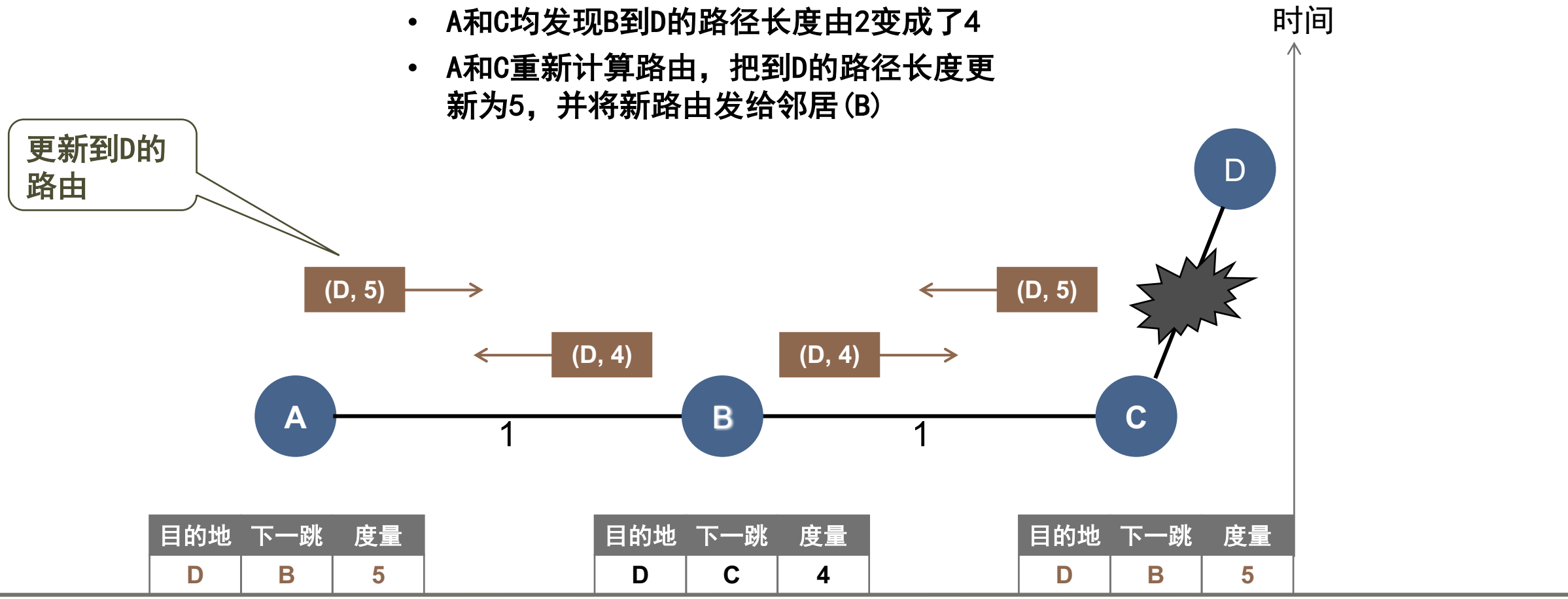
目的地	下一跳	度量
D	B	3

目的地	下一跳	度量
D	C	4

目的地	下一跳	度量
D	B	3

距离矢量算法缺点

- t10：A和C收到B的路由包
  - A和C均发现B到D的路径长度由2变成了4
  - A和C重新计算路由，把到D的路径长度更新为5，并将新路由发给邻居(B)

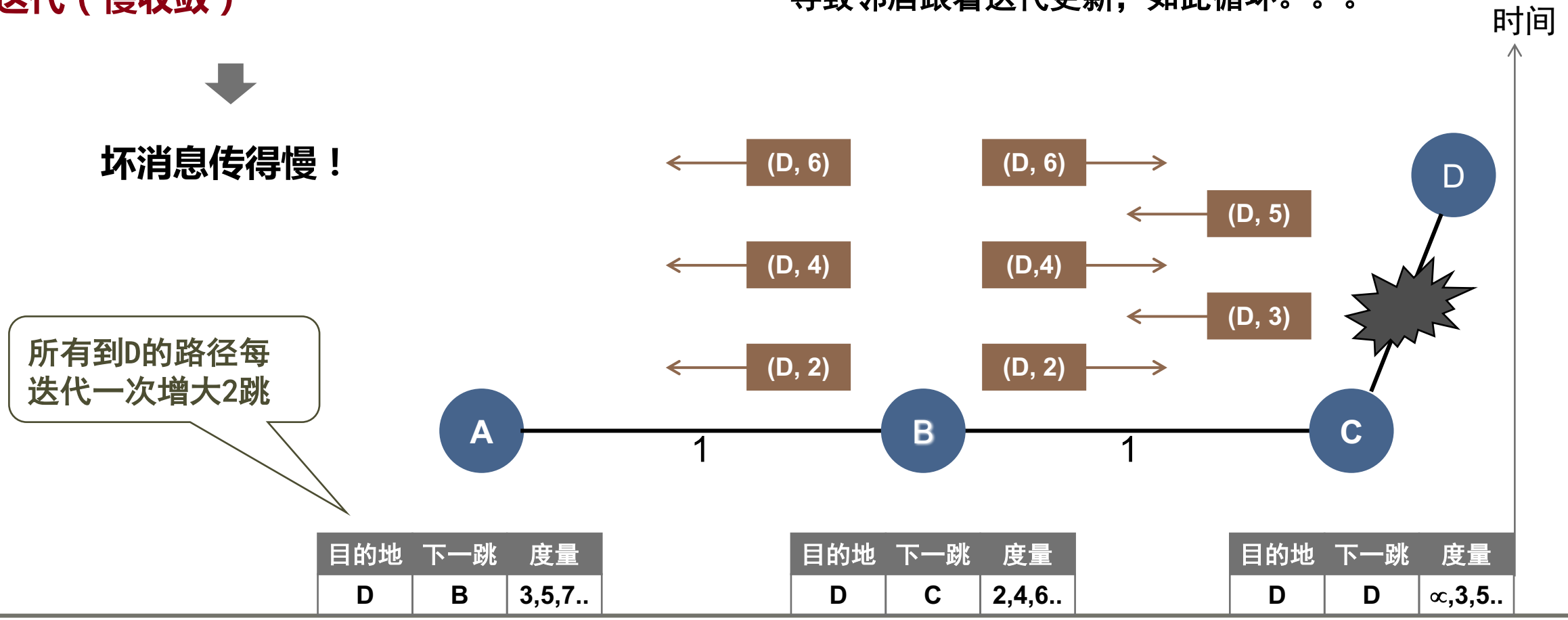


距离矢量算法缺点

无穷计算问题：算法陷入无限迭代（慢收敛）

算法每次迭代都修改了到D的路由长度，导致邻居跟着迭代更新，如此循环。。。

坏消息传得慢！



目的地	下一跳	度量
D	B	3,5,7..

目的地	下一跳	度量
D	C	2,4,6..

目的地	下一跳	度量
D	D	∞,3,5..

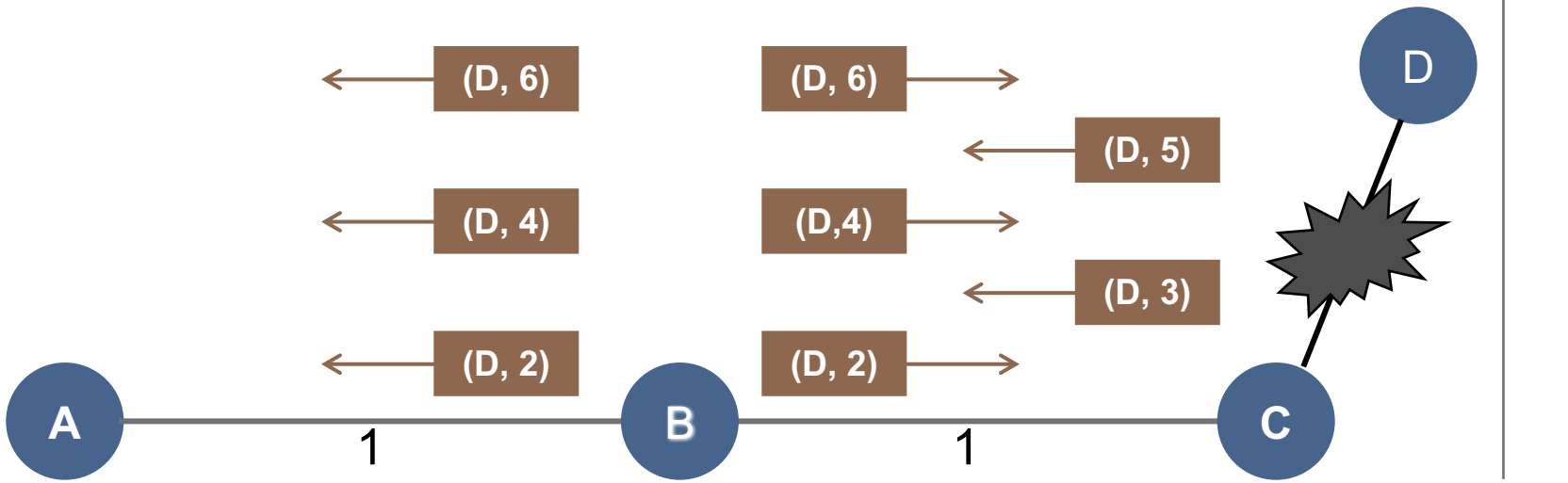
无穷计算问题的本质

? 为什么会出现这种无穷迭代情况？

交换的路由信息包括 → ① 可达目的地 ② 相应路径长度

解决方案

- 包括完整的路径信息
- 不发布下一跳是对方的路由



距离矢量路由算法

LS与DV路由算法的比较

链路状态算法复杂性

- 发送的链路状态包量级O(nE)
- 只要一条链路成本发生变化，新链路成本必须通知所有节点

无论周期性地广播链路成本还是链路成本发生变化时发布更新的链路成本，对网络带宽都是很大的消耗。

距离矢量算法复杂性

- 每次迭代时相邻节点交换信息（目的地、路径长度）
- 只有在新链路成本影响到该链路上节点的最小成本时才会重新计算和分发路由

只在路由计算结果发生变化时才将新路由通知给邻居，这种路由包仅在一跳范围内扩散，网络带宽消耗小。

LS与DV路由算法的比较

链路状态路由算法收敛速度

••O(n2)需要交换(nE)报文

为了可靠性，一般采用广播方式

- 每条链路成本被两边的节点各发送一次
- 每条链路成本被所有的节点要转发一次

距离矢量路由算法收敛速度

••收敛速度慢

好消息传得快坏消息传得慢

- 好路径的出现通过逐跳转发很快扩散到全网
- 坏路径的出现不仅传播慢而且有可能形成无穷计算问题

LS与DV路由算法的比较

链路状态算法的健壮性

- 有一定程度的健壮性

出错的情况包括广播的链路成本有错或者本地计算错，但这种错误只影响局部。

距离矢量算法的健壮性

- 计算错误可能扰乱整个网络

节点通过和邻居交换路由信息间接地了解整个网络状态（有哪些目的节点），一个节点的计算错误有可能波及一大片。