Ch5.Network Layer: The Control Plane

5.0 Background

understand principles behind network control plane:

- traditional routing algorithms
- SDN controlllers
- Internet Control Message Protocol
- network management

and their instantiation, implementation in the Internet:

- OSPF
- BGP
- OpenFlow
- ODL
- ONOS
- ICMP
- SNMP

5.1 routing protocols

目的

*Routing protocol goal:*通过路由器网络，确定从发送主机到接收主机的“良好”路径(即路由)

- path: 包从给定的初始源主机到给定的最终目标主机将遍历路由器序列
- “good”:“成本”最少，“最快”，“最不拥堵”
- routing: top 10网络挑战!

算法分类

Q: 中心化还是去中心化的信息?

中心化:

- 所有路由器都有完整的拓扑结构、链路成本信息
- "link state" algorithms

去中心化:

- 路由器知道物理连接的邻居，链路到邻居的成本
- 迭代计算过程，与邻居交换信息
- "distance vector" algorithms

Q: 静态还是动态?

静态:

- 路线会随着改变地很慢

动态:

- 路线变化更快
  - 定期更新
  - 以响应链接成本的变化

link-state算法(Dijkstra)

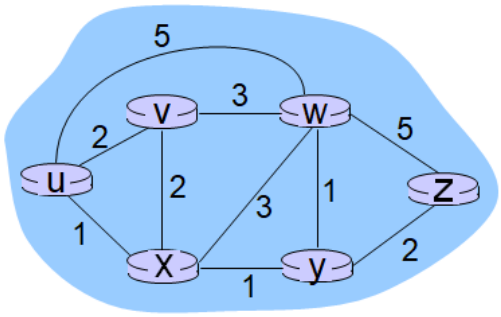
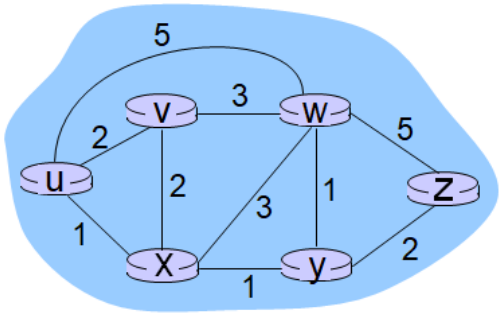


图: G = (N,E)

N =路由器的集合= { u, v, w, x, y, z }

E = 链路的集合 ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }



c(x,x') = link (x,x')的开销
例如, c(w,z) = 5

成本可以总是1，或者与带宽成反比，或者与拥塞成反比

路径的开销$(x_1, x_2, x_3, \ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

核心问题: u和z之间代价最小的路径是什么?
*routing algorithm:*找到代价最小路径的算法

Dijkstra's algorithm

- 网络拓扑结构，所有节点都知道链路开销
  - 通过“链路状态广播”实现
  - 所有节点都有相同的信息
- 计算从一个节点(“源”)到所有其他节点的最小代价路径
  - 给出该节点的*forwarding table*
- iterative: 迭代:经过k次迭代，知道到达k 的各种目的地的最小代价路径

notation:

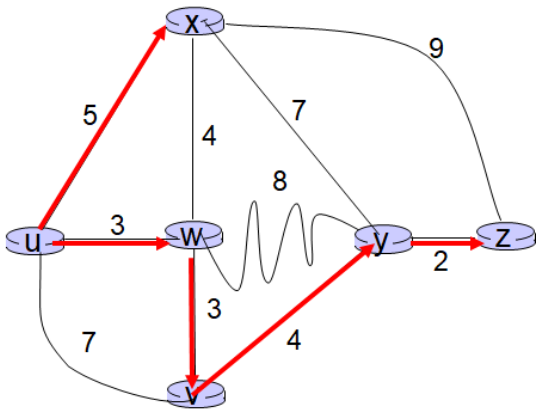
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijkstra algorithm

```
1  Initialization:
2   N' = {u}
3   for all nodes v
4     if v adjacent to u
5         then D(v) = c(u,v)
6     else D(v) = ∞
7
8  Loop
9    find w not in N' such that D(w) is a minimum
10    add w to N'
11    update D(v) for all v adjacent to w and not in N' :
12       D(v) = min( D(v), D(w) + c(w,v) )
13    /* new cost to v is either old cost to v or known
14     shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```

示例

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

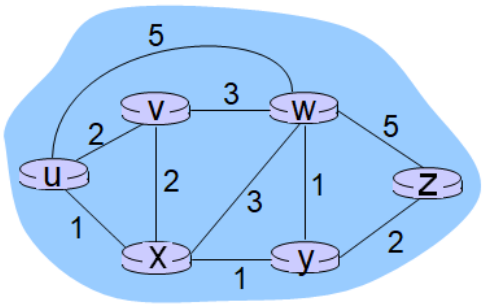


notes:

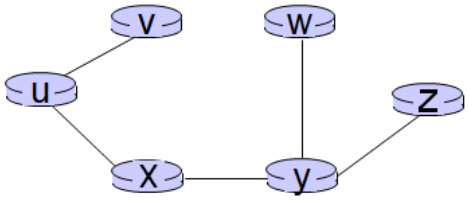
- 通过跟踪祖先节点构造最短路径树
- 关系可以存在(可以任意断开)

forwarding table的构建

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



从u得到的最短路径树



u的forwarding table:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

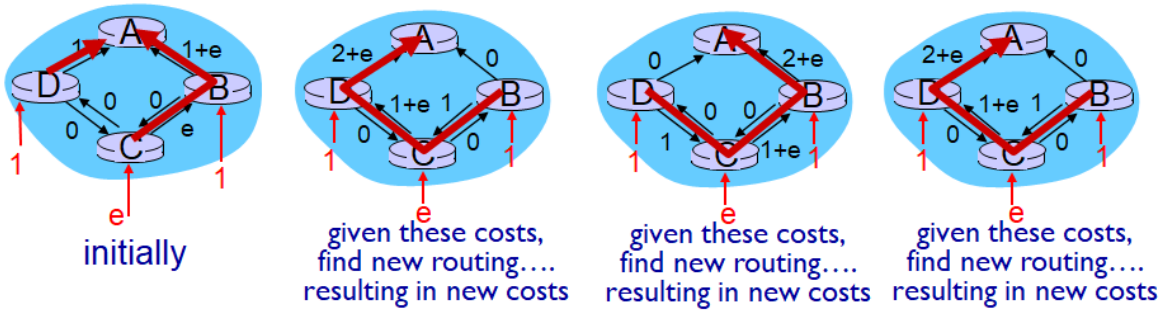
分析

algorithm complexity: n nodes

- 每次迭代:需要检查所有不在N的所有节点w
- n(n+1)/2次比较: $O(n^2)$
- 更有效的实现可能: O(nlogn)

oscillations possible:

- 例如:支持链路开销等于承载的流量:



distance-vector算法

算法

Bellman-Ford方程(动态规划)

设

$d_x(y)$:=从x到y的成本的最小成本路径

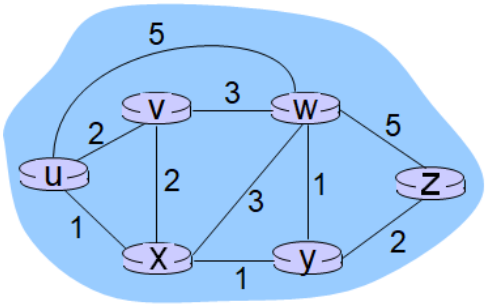
那么

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

$d_v(y)$: cost from neighbor v to destination y

$c(x,v)$: cost to neighbor v

$\min_v$: *min* taken over all neighbors v of x



明显地, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

那么:

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

最小的节点是最短路径中的下一跳，用于转发表

- $D_x(y)$ =从x到y的最小成本估计
- x 保持distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- node x:
  - 知道每个邻居v的开销: c(x,v)
  - 维护它的邻居的distance vector.对于每个邻居v, x维护 $\mathbf{D}_v = [D_v(y): y \in N]$

key idea:

- 不时地，每个节点向邻居发送自己的distance vector的估计
- 当x从邻居接收到新的DV估计时，它使用B-F方程更新自己的DV:

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

- 在较小的自然条件下，估计$D_x(y)$收敛于实际的最小成本$d_x(y)$

异步迭代:

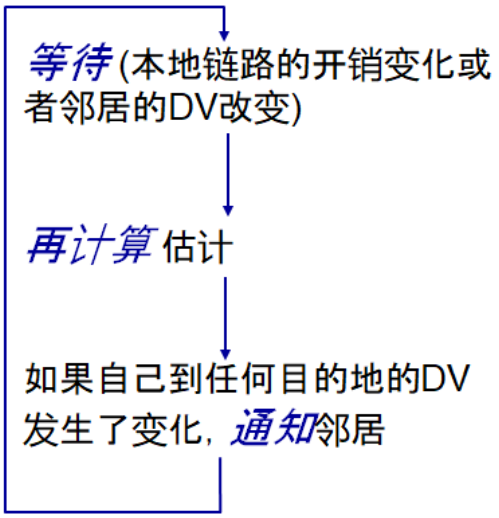
每个局部迭代由:

- 本地链路开销变化
- 邻居发来的DV更新消息

分布式的:

- 每个节点只有在它的DV发生变化时才会通知邻居
  - 邻居在必要时通知他们的邻居

每一个节点:

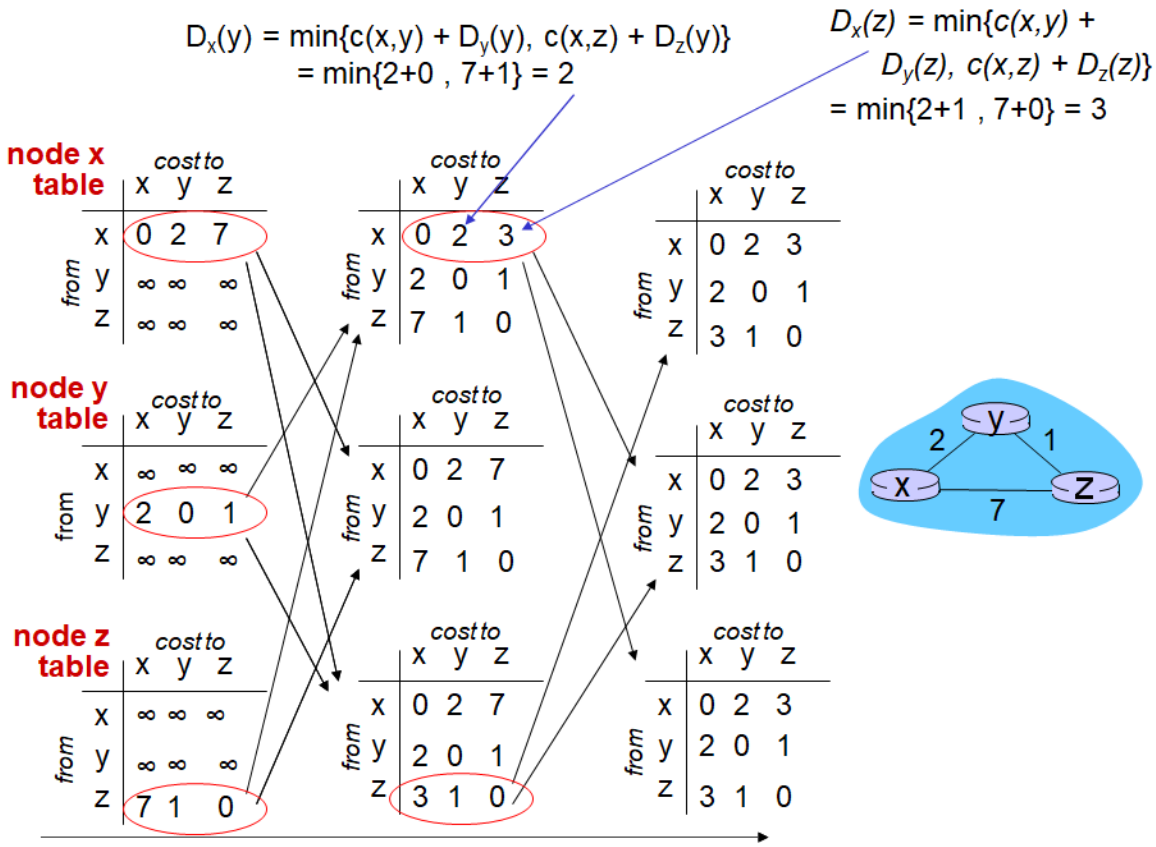


示例

计算的时候

$c(x,y)$**使用更新前的**$D_y(y)$**使用更新后的**

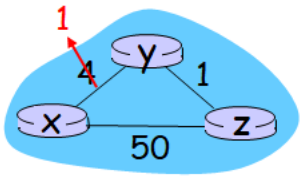
然后再把得到的值更新本行



分析

链接开销变化:

- 节点检测到本地链路开销变化
- 更新路由信息，重新计算distance vector
- 如果DV发生变化，通知邻居



"好消息传得快"

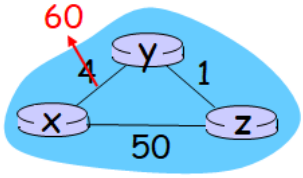
t_0: y检测链路成本变化，更新它的DV，通知它的邻居

t_1: z从y接收更新，更新它的表，计算新的最小成本x，发送它的邻居DV。

t_2: *y接收到z的更新，更新它的距离表。y的最小成本不变，所以y不会向z发送消息*

链接开销变化:

- 节点检测到本地链路开销变化
- 坏消息传得慢——"数到无穷"的问题!
- 算法稳定之前的44次迭代



毒性逆转:

- 如果Z通过Y到达X:
  - Z告诉Y它到X的距离是无限的(所以Y不会通过Z到达X)
- 这能完全解决数到无穷的问题吗?

两个算法的比较

当遇到内部的Router故障时，DV受到的影响更大

时间复杂度

- **LS:** n个节点，E个链接，O(nE)消息发送
- **DV:** 只在邻居之间交换
  - 收敛时间不同

收敛速度

- **LS:** O (n^2)算法需要O(nE)个消息
  - 可能振荡
- **DV:** 收敛时间不同
  - 可能routing loops
  - count-to-infinity问题

Robustness鲁棒性: 如果路由器发生故障会发生什么？

LS:

- 节点可能会发布错误的链路开销
- 每个节点只计算自己的表

DV:

- DV节点可以通告不正确的路径开销
- 每个节点的表被其他节点使用
  - 错误会在网络中传播

5.3 intra-AS routing in the Internet: OSPF

背景 理想化的routing table

我们的路由研究到目前为止还是理想化的

- 所有路由器相同
- 网络“平”
- …*实际上并不正确*

规模: 拥有数十亿目的地:

- 不能在路由表中存储所有目的地!
- 路由表交换会使链接失效!

行政自治

- internet = network of networks
- 每个网络管理员可能想在自己的网络中控制路由

AS的定义与分类

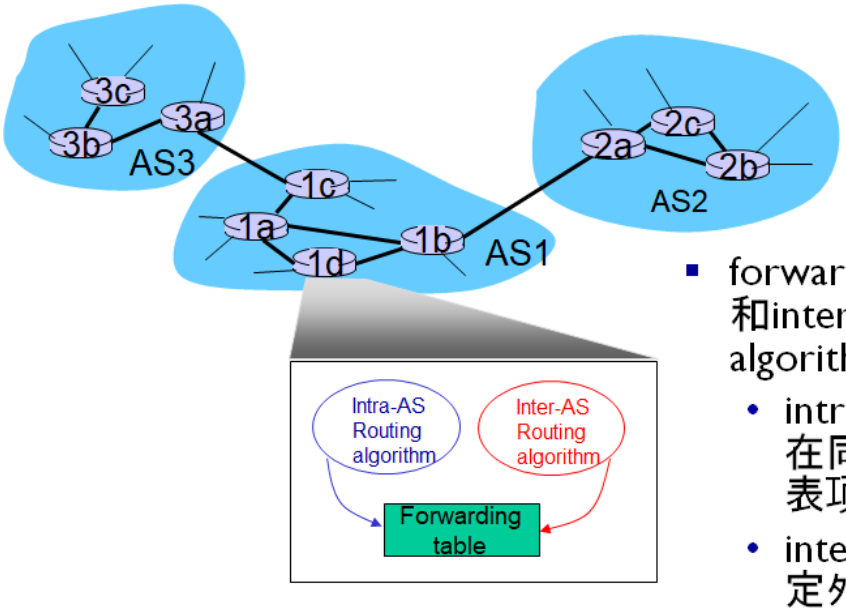
将路由器聚合成称为“自治系统”(AS)的区域(又名“domain”)

intra-AS routing

- 相同的AS间的主机、路由器之间的routing
- AS中的所有路由器必须运行相同的内部域协议
- 不同自治系统中的路由器可以运行不同的域内路由协议
- 网关路由器:在它自己的AS的“边缘”，有连接到其他AS的路由器

inter-AS routing

- 在AS之间的routing
-)网关执行inter-domain routing(以及intra-domain routing)



- forwarding table 由intra-和inter-AS routing algorithm共同配置
  - intra-AS routing确定在同一个AS下的各个表项
  - inter-AS & intra-AS确定外部目的地的表项

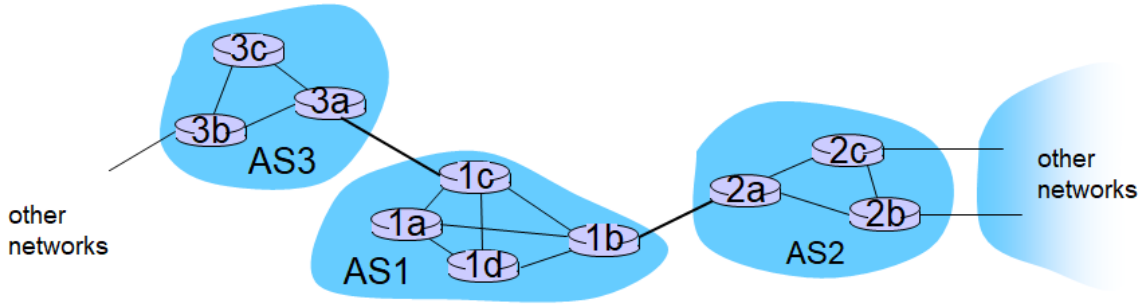
Inter-AS（BGP)

- 假设AS1中的路由器收到了目的地在AS1之外的数据报:
  - 路由器应将包转发给网关路由器，但是是哪一个?

AS1 必须:

1. 了解哪些目的地可以通过AS2到达，哪些可以通过AS3到达
2. 将此可达性信息传播到AS1中的所有路由器

job of inter-AS routing!



Intra-AS（IGP)

- 也称为内部网关协议*interior gateway protocols (IGP)*
- 最常见的intra-AS路由协议:
  - RIP: Routing Information Protocol(distance-vector)
  - OSPF: Open Shortest Path First (link state)
  - IS-IS: Intermediate System-to-Intermediate System(link state)
  - IGRP: Interior Gateway Routing Protocol(distance-vector)

OSPF(Open Shortest Path First)介绍

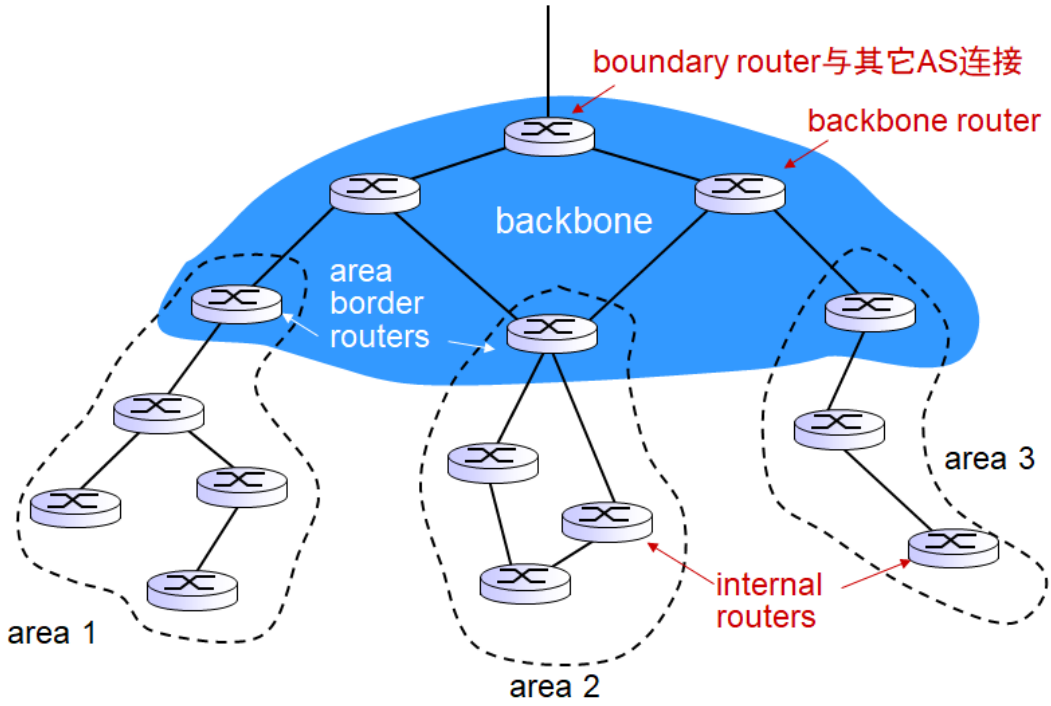
- “open”:公开的
- 使用link-state algorithm
- 路由器将OSPF的link-state通告传递给整个AS的所有其他路由器
  - OSPF消息直接在IP(而不是TCP或UDP)中携带
  - 链接状态:每个连上的链接
- *IS-IS routing* protocol:几乎与OSPF相同

OSPF高级功能

- *security*: 所有的OSPF消息经过验证(防止恶意入侵)
- 允许多条相同开销的路径(在RIP中只允许一条)
- （卫星链路因为低可信度设置低TOS，因为实时性设置高TOS）
- integrated uni- and multi-cast support:
  - 组播OSPF使用与OSPF相同的拓扑数据库
- 在大domain下使用hierarchical OSPF

分层OSPF

注意它们仍然都在同一个AS下面

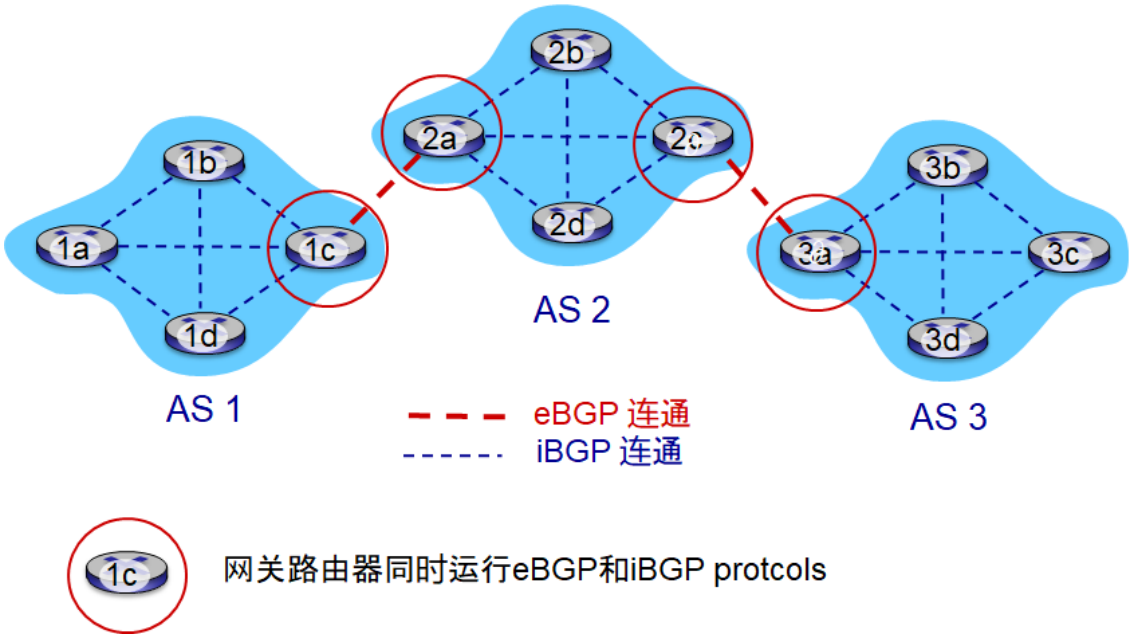


- *两层结构*: 当地的支柱
  - 仅在area广播link-state
  - 每个节点都有详细的区域拓扑;只知道到其他area的方向(最短路径)。
- *area border routers*: “汇总”到本区域网的距离，向其他区域边界路由器通告。
- *backbone routers*: 只在backbone中执行OSPF
- *boundary routers*: 连接到其他的AS

5.4 Internet inter-AS routing: BGP (among ISP's)

BGP介绍

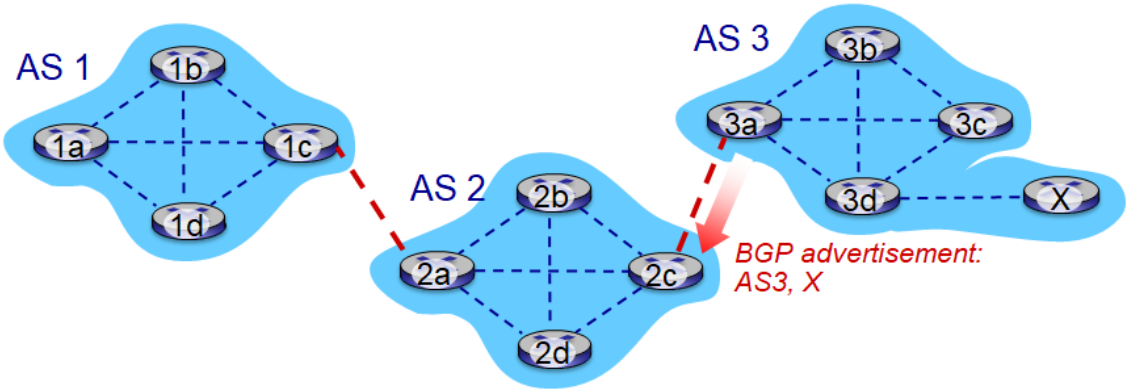
- BGP (Border Gateway Protocol): 事实上的inter-domain routing protocol
  - “将互联网凝聚在一起的粘合剂”
- BGP为每一个AS提供了一种方法:
  - eBGP:从相邻AS中获取子网可达性信息
  - iBGP: 向所有AS内部路由器传播可达性信息
  - 根据可达性信息和政策确定到其他网络的“良好”路由
- 允许子网向互联网的其他部分宣传其存在:“我在这里”



网关路由器同时运行eBGP和iBGP protcols

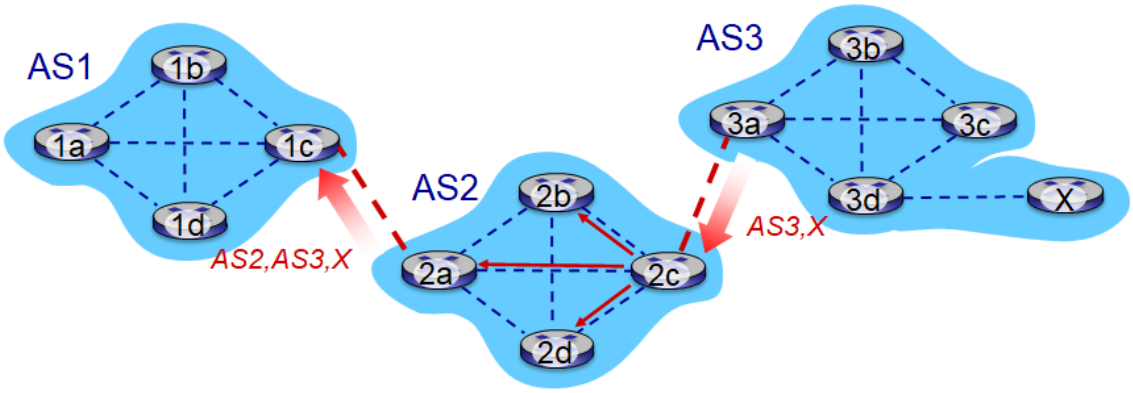
BGP基础

- BGP session: 两台BGP路由器(“对等体”)通过半永久TCP连接交换BGP消息:
  - 向不同的目标网络前缀发布路径(BGP是一种“path vector”协议)
- 当AS3网关路由器3a向AS2网关路由器2c发布路径AS3,X时:
  - AS3向AS2承诺将datagram转发到X

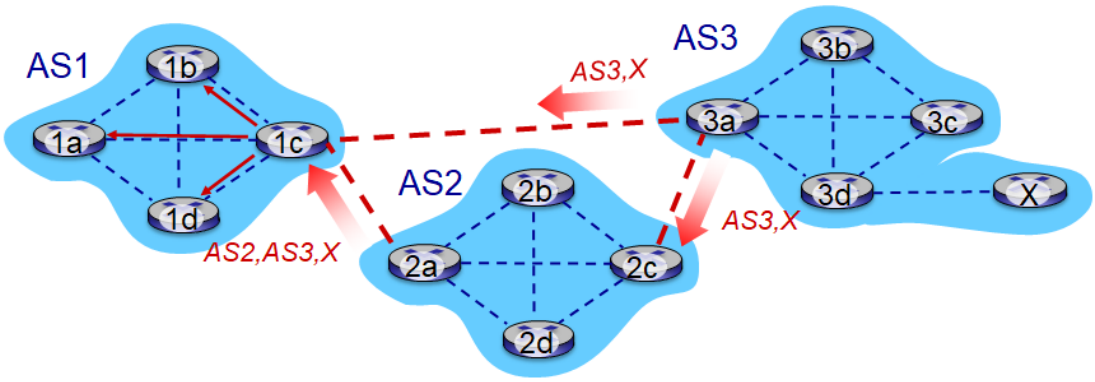


- 发布的前缀包括BGP属性
  - prefix + attributes = “route”
- 两个重要的属性:
  - AS-PATH: prefix发布的经过的一系列AS
  - NEXT-HOP: 指出一个特别的internal-AS可以满足下一个AS的一跳
- *Policy-based routing:*
  - 网关接收路由通告使用输入政策来接受/拒绝路径(例如,从不通过AS Y路由)
  - AS策略还决定是否向其他相邻的AS告知路径

BGP路径通告



- AS2路由器2c从AS3路由器3a接收路径通告AS3,X(通过eBGP)
- 基于AS2策略，AS2路由器2c接受路径AS3,X，(通过iBGP)向所有AS2路由器传播
- 根据AS2策略，AS2路由器2a(通过eBGP)向AS1路由器1c发布路径AS2、AS3、X



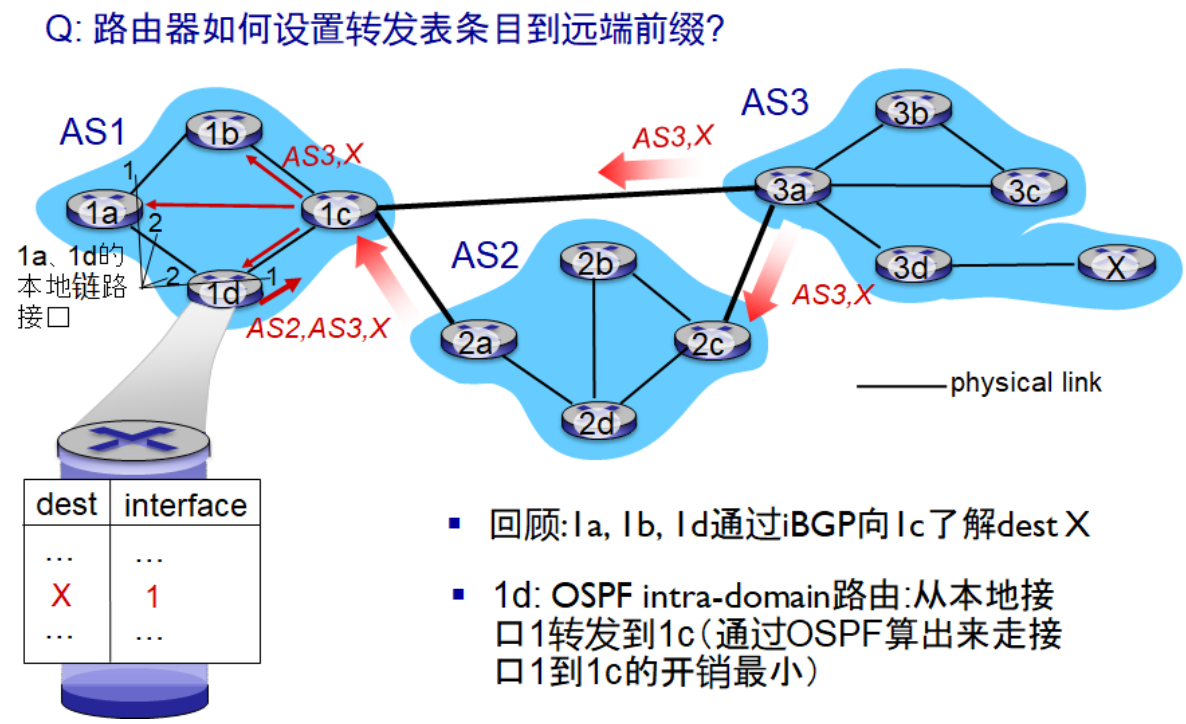
网关路由器可能了解到目的地的多条路径:

- AS1网关路由器1c从2a学习路径AS2、AS3、X
- AS1网关路由器1c从3a学习路径AS3,X
- 根据策略，AS1网关路由器1c选择路径AS3、X，(通过iBGP)发布向AS1内部发送路径

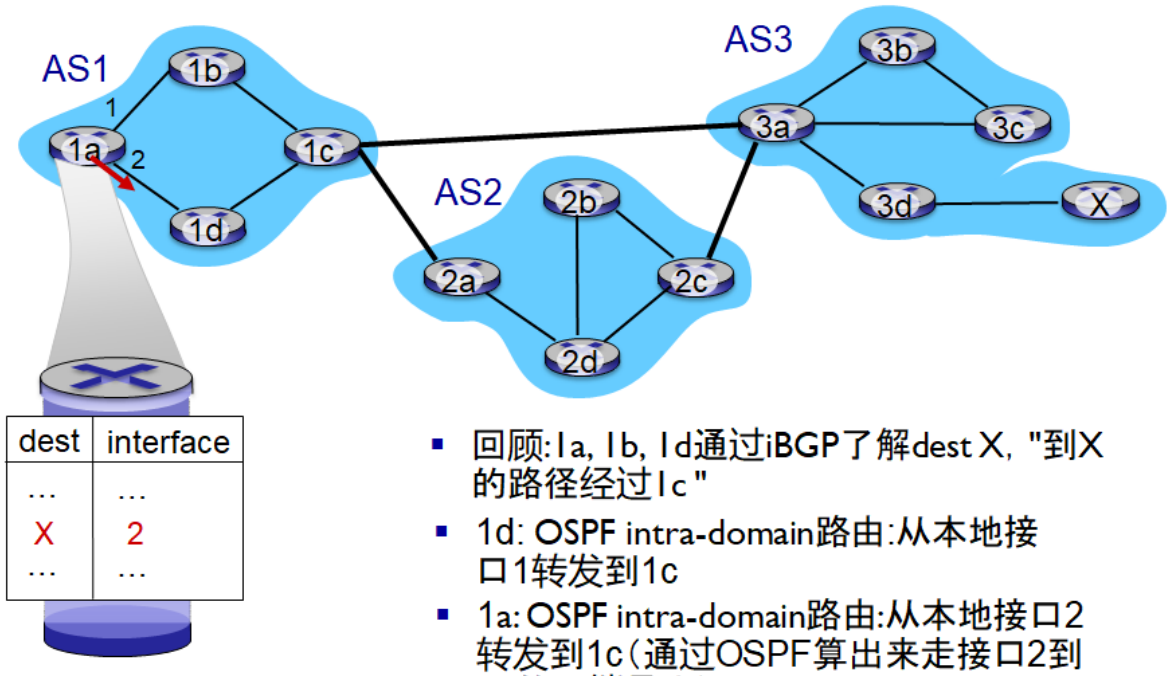
BGP信息

- 对等体之间通过TCP连接交换BGP消息
- BGP 信息:
  - OPEN: 建立与远端BGP对等体的TCP连接，并验证发送BGP对等体的身份
  - UPDATE: 通知新路径(或撤回旧路径)
  - KEEPALIVE: 在没有更新的情况下保持连接处于活动状态;也打开请求
  - NOTIFICATION: 报告错误先前的错误信息;也用于关闭连接

AS内部forwarding table表项设置



- 回顾:1a, 1b, 1d通过iBGP向1c了解dest X
- 1d: OSPF intra-domain路由:从本地接口1转发到1c(通过OSPF算出来走接口1到1c的开销最小)

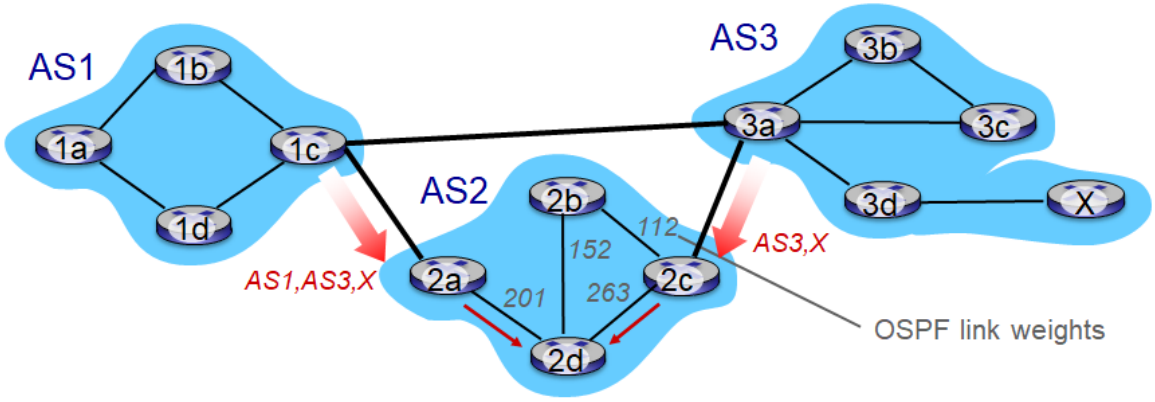


- 回顾:1a, 1b, 1d通过iBGP了解dest X, "到X的路径经过1c"
- 1d: OSPF intra-domain路由:从本地接口1转发到1c
- 1a: OSPF intra-domain路由:从本地接口2转发到1c(通过OSPF算出来走接口2到1c的开销最小)

BGP路径选择

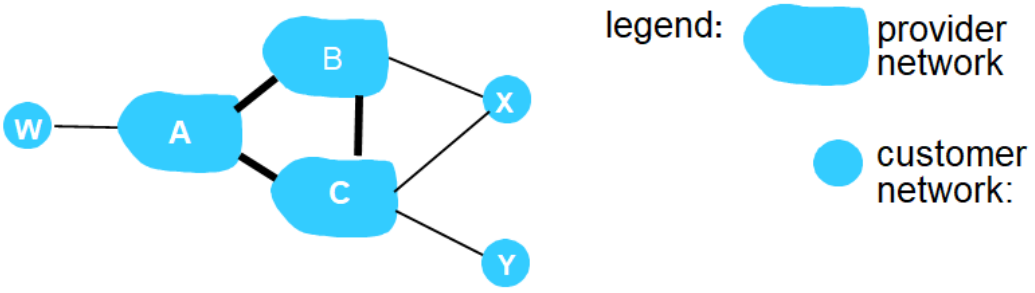
- 路由器可能会了解到多个到达目的地AS的路由，选择哪一条将基于：
  1. 局部偏好值属性:策略决策
  2. 最短AS-PATH
  3. 最近NEXT-HOP router: hot potato routing
  4. 附加标准

Hot Potato Routing



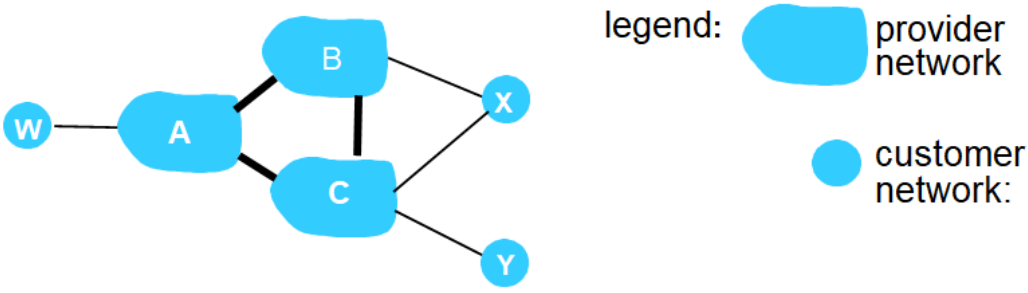
- 2d (通过iBGP)学习它可以通过2a或2c路由到X
- *hot potato routing*: 选择intra-domain成本最小的本地网关(例如，2d选择2a，即使更多的跳转到X):不要担心inter-domain成本!

BGP通告政策



假设一个ISP只希望将流量路由到或从它的客户网络(不希望在其他ISP之间传输流量)

- A向B和C通告路径Aw
- B选择不向C通告BAw:
  - B没有从路由CBAw中获得“收益”，因为C、A、w都不是B的客户
  - C不了解CBAw路径
- C将使用路线CAw(不使用B)到达w



假设一个ISP只希望将流量路由到或从它的客户网络(不希望在其他ISP之间传输流量)

- A、B、C是供应商网络
- X,W,Y为(供应商网络的)客户
- X是双归属的:连接到两个网络
- *policy to enforce:* X不希望B经过X到C
  - ..所以X不会向B通告有一条路径从它到C

Inter-AS和Intra-AS的对比

政策:

- inter-AS: : 管理员想要控制它的流量如何路由，谁通过它的网络路由。
- intra-AS: 单一管理，因此不需要政策决定

规模:

- 分层路由节省表大小，减少更新流量

效果:

- intra-AS:可以专注于性能
- inter-AS:策略可能会影响性能

5.5 SDN (Software Define Network)

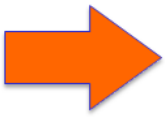
背景 功能分离

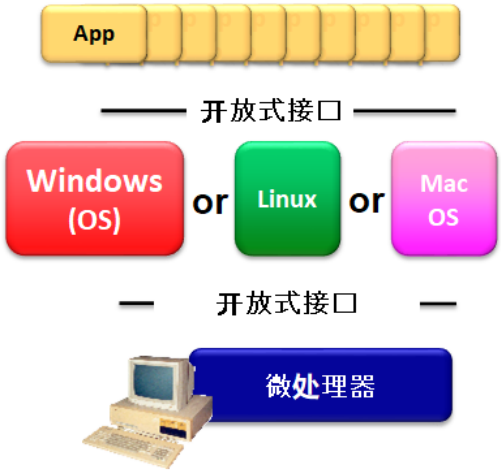
- Internet网络层:过去一直通过分布式、单一路由器的方法实现
  - *monolithic* 路由器包含交换硬件，在路由器操作系统(如Cisco IOS)中运行Internet标准协议(IP、RIP、IS-IS、OSPF、BGP)的专有实现。
  - 不同的“中间盒”用于不同的网络层功能:防火墙、负载平衡器、NAT盒…
- ~2005:重新有兴趣重新思考网络控制平面

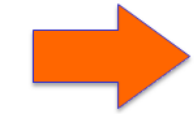
为什么要逻辑上集中的控制平面?

- 更方便的网络管理:避免路由器错误配置，更灵活的流量流
- 基于表的转发(回想一下OpenFlow API)允许“编程”路由器
  - 集中“编程”更容易:计算表集中和分布
  - 分布式编程更困难:计算表作为分布式算法(协议)在每个路由器实现的结果
- 控制平面的开放(非专有)实现

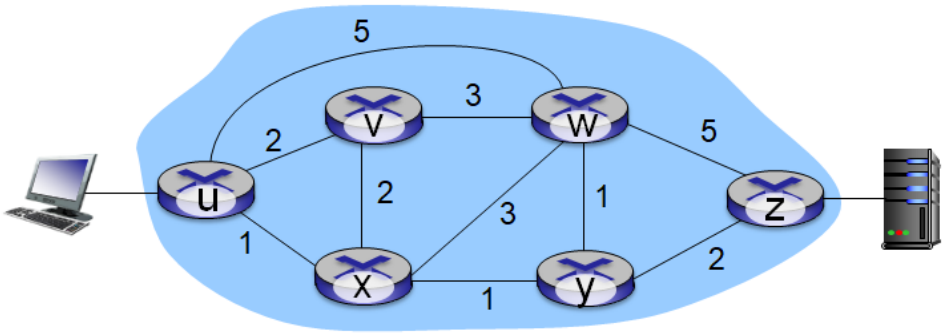
垂直整合
关闭,专有
缓慢的创新
小的行业







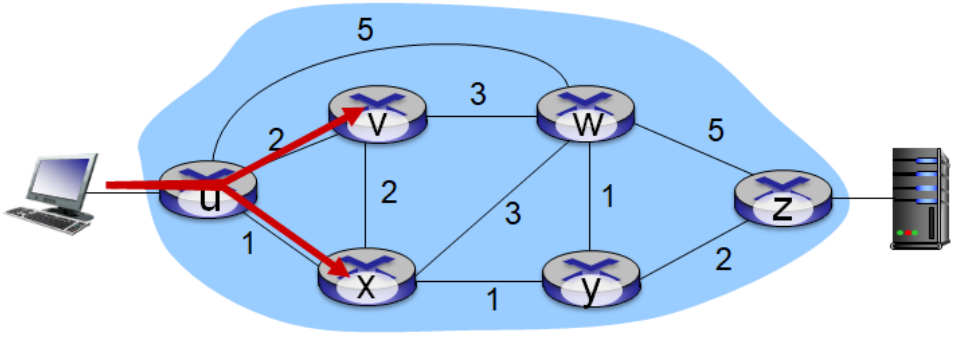
不方便性



Q: 如果网络运营商希望u到z的流量沿着uvwz流动，x到z的流量沿着xwyz流动呢？

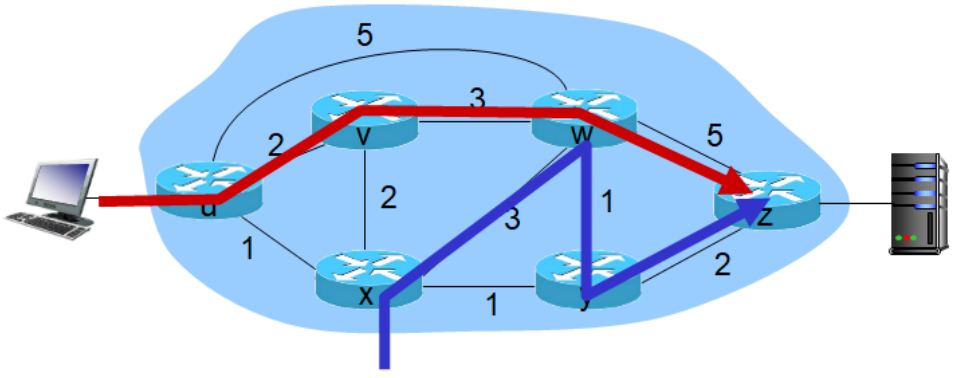
A: 需要定义链路权重，使流量路由算法相应计算路由(或需要新的路由算法)!

链路权重只是控制“旋钮”:错误!



Q: 如果网络运营商想要沿着uvwz和uxyz(负载均衡)分割u-to-z流量怎么办?

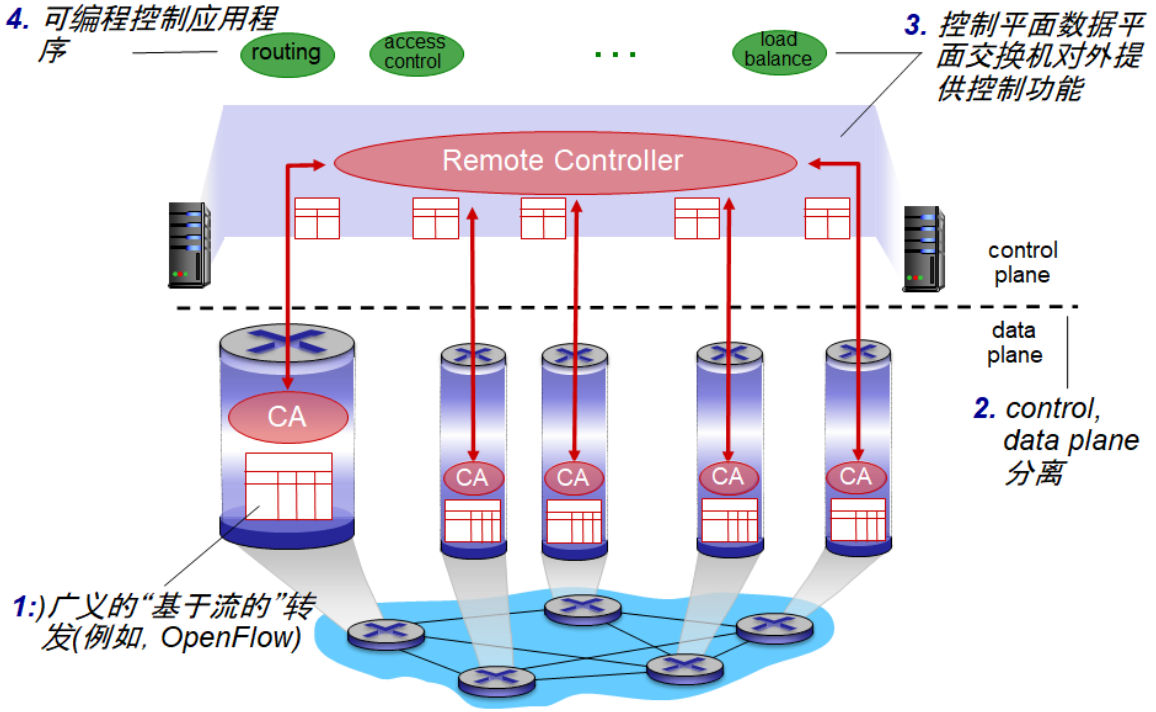
A: 不能这样做(或者需要一个新的路由算法)



Q: 如果w想让蓝色和红色的流量不同呢?

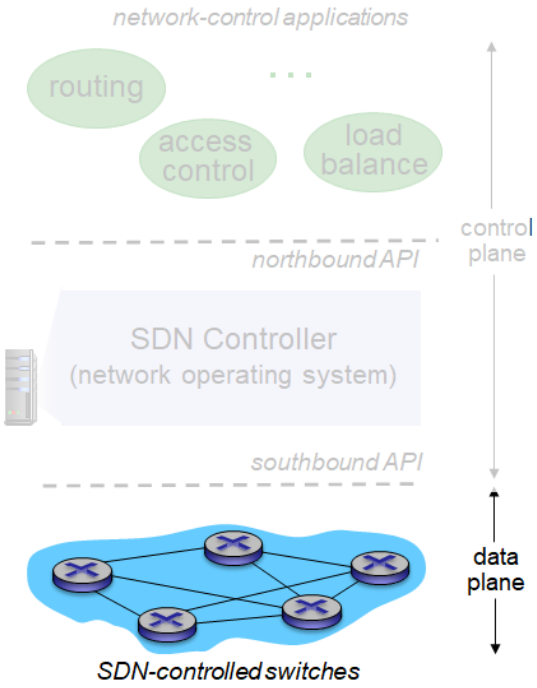
A: 不能这样做(基于目的地的转发，和LS，DV路由)

SDN概览



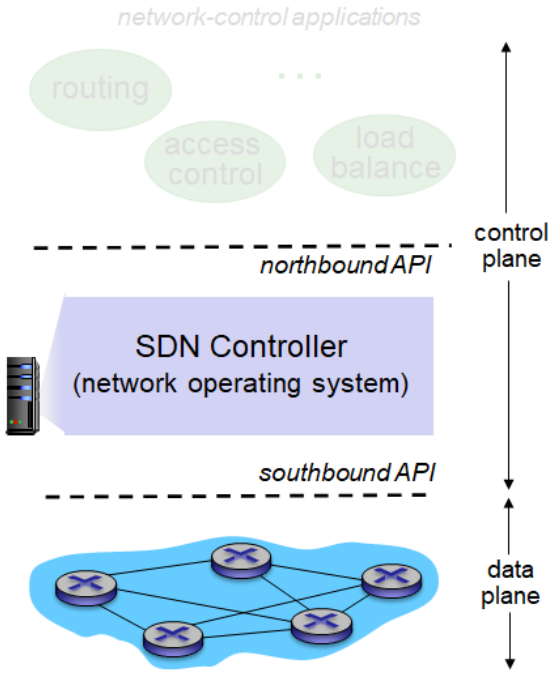
Data plane 交换机

- 在硬件上实现广义data plane的转发的快速、简单、通用交换机
- flow table表由controller计算、安装
- 基于表的交换机控制的API(例如，OpenFlow)
  - 定义什么是可以控制的，什么是不可控制的
- 与controller通信的协议(例如，OpenFlow)



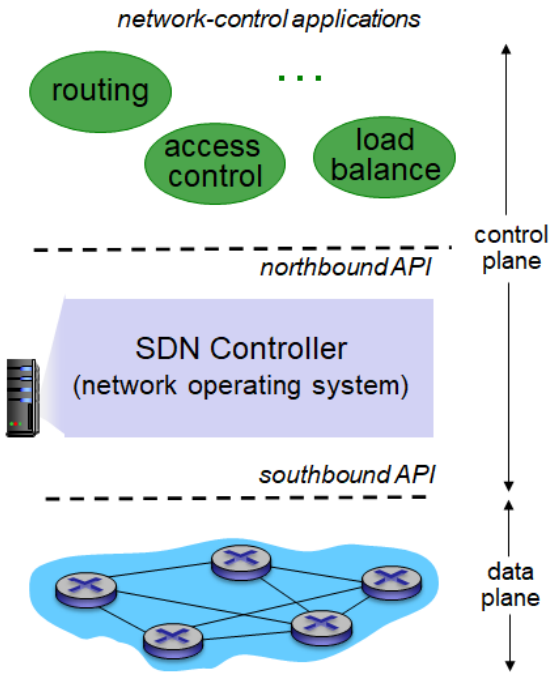
SDN 控制器 (network OS):

- 维护网络状态信息
- 通过北向API与“上面”的网络控制应用程序交互
- 通过南向API与“下面”的网络交换机交互
- 实现为分布式系统的性能，可伸缩性，容错，健壮性

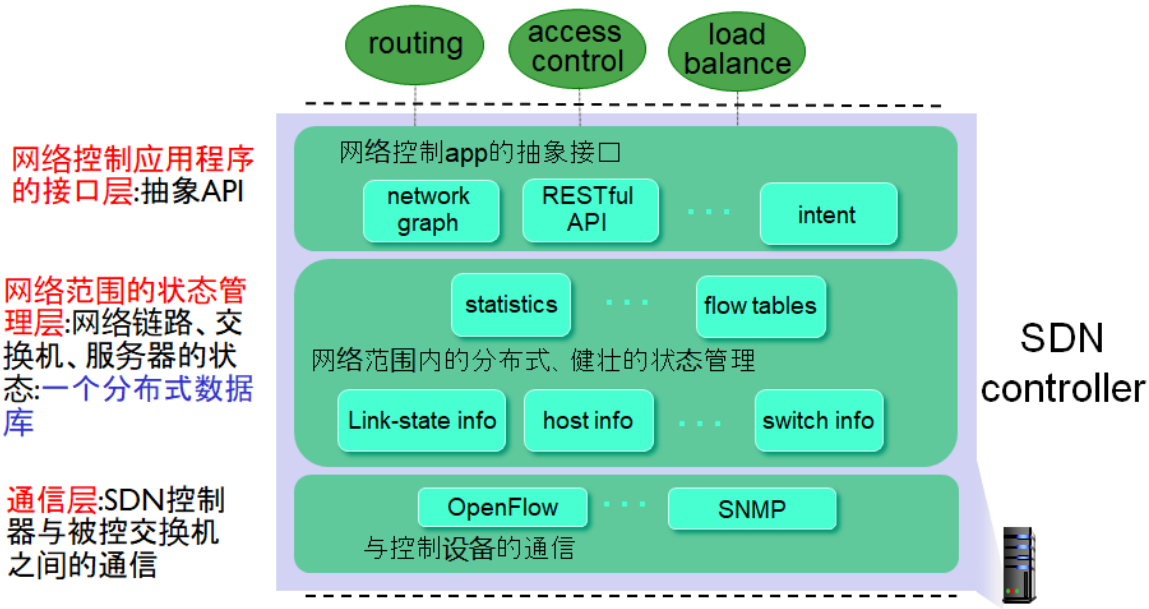


网络控制程序:

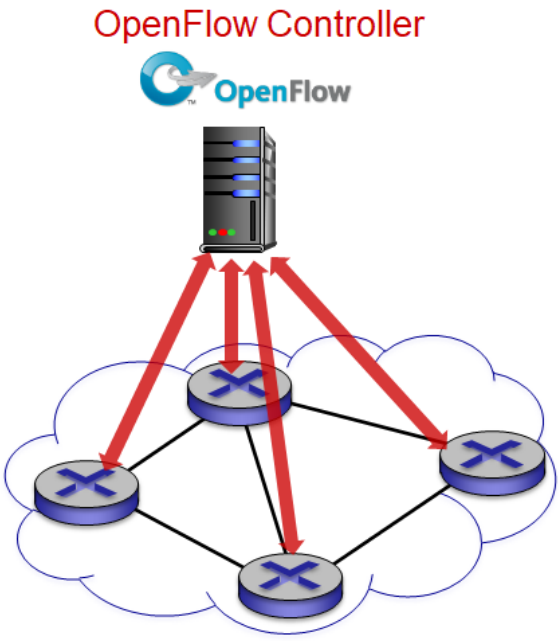
- 控制的“大脑”:通过SND控制器提供的底层服务API来实现控制功能
- 非绑定的:可以由第三方提供:与路由供应商或SDN控制器不同



SDN controller的具体组成



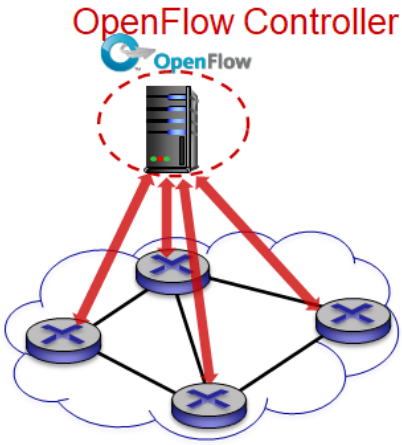
Openflow 协议



- 在controller、switch之间工作
- 用TCP交换消息
  - optional encryption可选加密
- 三种OpenFlow消息:
  - controller-to-switch
  - 异步 (switch to controller)
  - 对称(misc)

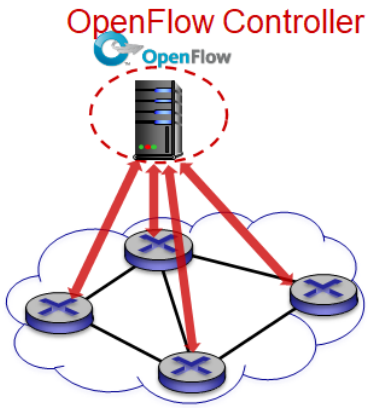
核心的controller-to-switch消息

- *features:* controller查询switch特性，switch回复
- *configure:* controller 查询/设置switch配置参数
- *modify-state:* 增加、删除、修改OpenFlow表中的flow项
- *packet-out:* controller可以将此数据包从特定的switch端口发送出去



核心的switch-to-controller消息

- *packet-in:* 传送数据包(及其控制)到controller。查看来自controller的packet-out消息
- *flow-removed:* 在switch上删除flow table
- *port status:* 通知controller端口的变化



幸运的是，网络运营商不会通过直接创建/发送OpenFlow消息来“编程”交换机。相反，在controller使用更高级的抽象

SDN的优点

- control plane加固:可靠、可靠、性能可扩展、安全的分布式系统
  - 对故障的鲁棒性:利用强大的可靠分布式系统理论进行控制
  - 可靠性、安全性:从第一天起就“融入”了吗?
- 网络，满足特定任务要求的协议
  - e.g., 实时、非常稳定、非常安全
- 网络扩展

5.6 ICMP: The Internet Control Message Protocol

ICMP报文

- 用于主机和路由器通信网络层的信息
  - 错误报告:不可达的主机，网络，端口，协议
  - echo request/reply (used by ping)
- 网络层"高于"IP:
  - IP数据报中携带的ICMP消息
- ICMP message: 类型，代码加上导致错误的IP数据报的前8个byte

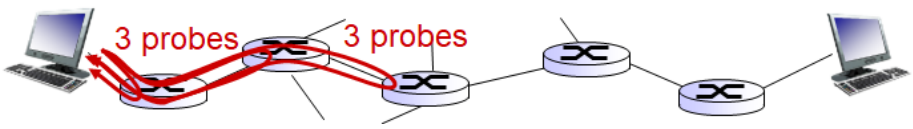
Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Traceroute流程

- source向destination发送一系列的UDP段
  - 首先设置TTL =1
  - 然后设置TTL=2, 以此类推.
- 当第n组datagram到达第n个路由器时:
  - 路由器丢弃数据报并发送sourceICMP消息(类型11，代码0)(TTL expired)
  - ICMP报文包括路由器名称和IP地址
- 当ICMP报文到达时，source记录RTT

停止判据:

- UDP段最终到达目标主机
- 目的地返回ICMP"端口不可达"消息(类型3，代码3)(dest port unreachable)
- source停止

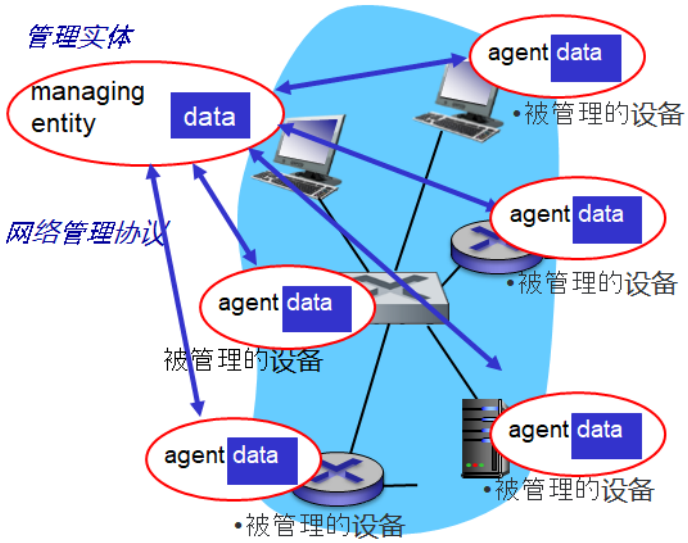


5.7 Network management and SNMP

Network management

Network management 包括部署、集成和协调硬件、软件和人力要素，以监控、测试、投票、配置、分析、评估和控制网络和要素资源，以以合理的成本满足实时、运行性能和服务质量需求。

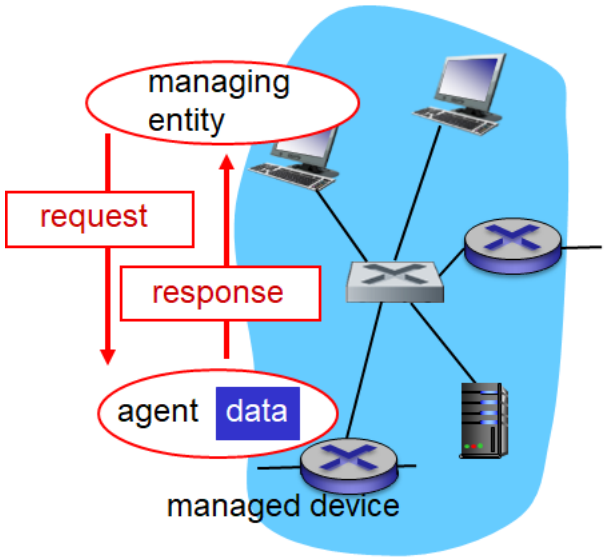
定义:



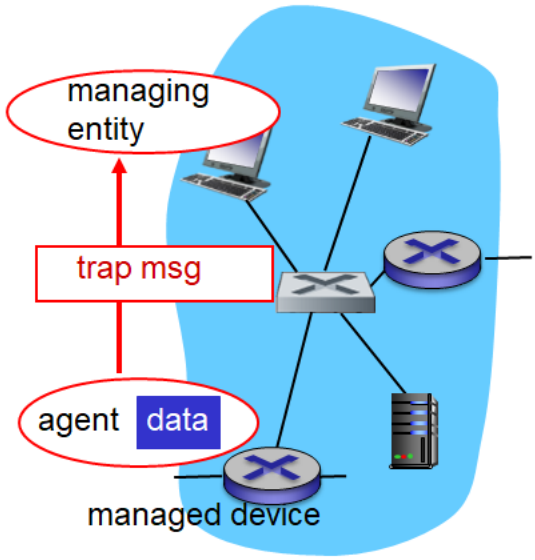
managed devices contain *managed objects* whose data is gathered into a *Management Information Base (MIB)*

SNMP协议

传递MIB的信息和命令的两种方式:



请求/响应模式



自主上传

消息类型	功能
GetRequest GetNextRequest GetBulkRequest	manager-to-agent:“给我数据” (数据实例，列表中的下一个数据，数据块)
InformRequest	manager-to-manager:这里的MIB值
SetRequest	manager-to-agent: 设置MIB值
Response	Agent-to-manager: 对请求的回应
Trap	Agent-to-manager:通知异常事件