

计算机网络

网络互连

华中科技大学电信学院 2016



学习目标

- 理解分组交换网络的概念；
- 掌握数据报交换、虚电路交换的原理；
- 掌握局域网扩展的概念，理解网桥、学习型网桥的概念，了解生成树算法；
- 掌握以太网集线器、网桥、交换机设备的功能与区别；
- 理解网络互联的需求和概念；
- 理解IP服务模型，包括地址设计、分段与重组、数据报转发等；
- 掌握子网、超网的概念，以及相应的IP地址分配方式；
- 理解IP地址管理相关协议，包括地址转换ARP、动态地址分配DHCP等；
- 掌握路由与转发的概念，理解路由表与转发表的区别；
- 掌握基于距离向量和基于链路状态的路由算法的原理，理解RIP和OSPF的实现要点，具备进行简单路由计算的能力；
- 理解路由权值代价设置的问题，了解互联网链路权值设计的演变；
- 理解路由器的执行机制，理解路由器与交换机的区别；



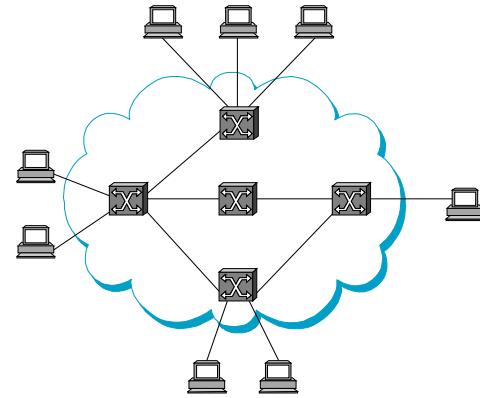
提纲



- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
- 实现和性能
- 总结

单一网络的局限

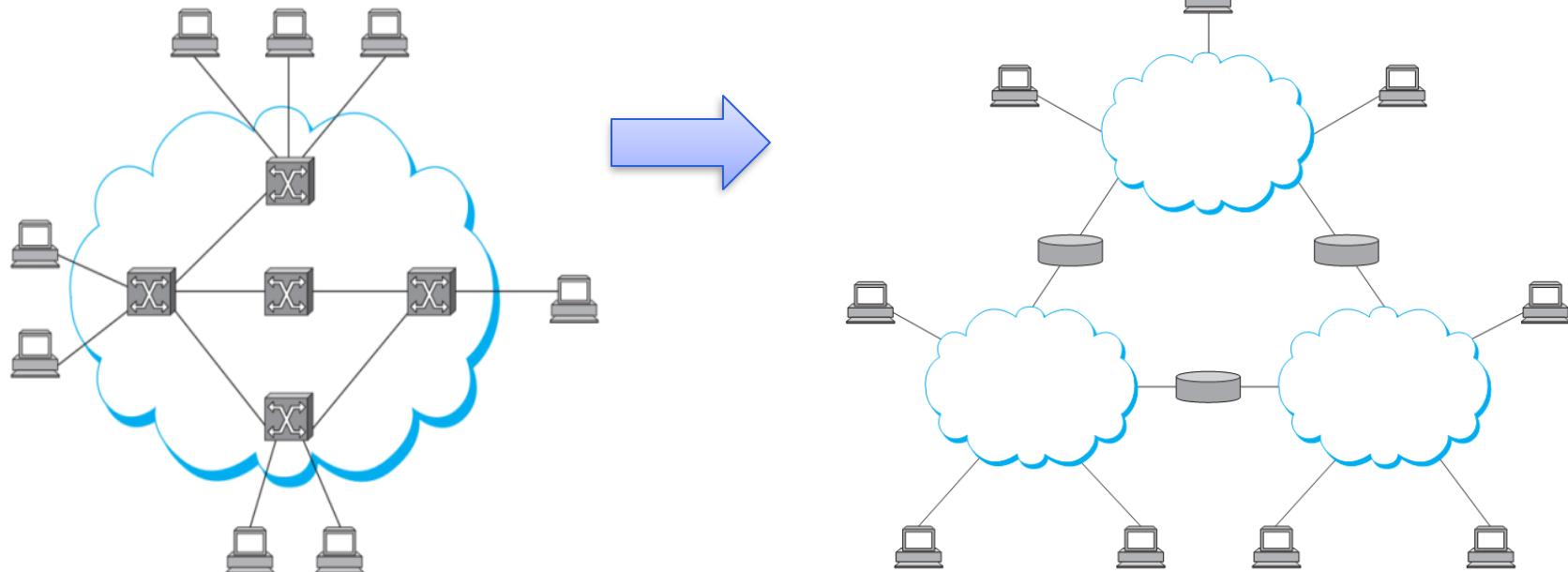
- 单一类型的网络
 - 互联的主机采用相同的寻址机制, 介质访问控制协议, 服务模型, 等等



- 局限性
 - 异构性
 - 当某一类型网络中的用户期望与另一类型网络中的用户进行通信, 如何提供主机之间的端到端服务?
 - 扩展性
 - 当网络规模增大, 如何发现有效的路径?
 - 当网络规模增大, 如何对网络节点进行有效的标识?

网络互联

- 单一网络面临的问题
 - 网络异构和扩展的局限性
- 解决方案
 - 网络互联





核心问题

并不是所有网络都是直接相连的

电路交换机(*Circuit switch*)

- 观察电话网络
 - 电话不是直接相连的
 - 每部电话连到一个交换设备：交换机
 - 交换机保证两部电话机建立单独的电路链接



PBX交换主板, 1924



人工交换主板



数字交换设备。
每个交换机通常支持
10,000-100,000+ 用户

分组交换机(*Packet switch*)

- 动机
 - 计算机连接到分组交换机
 - 分组可从一台主机传输到另外一台主机
 - 分组交换机采取存储转发将分组从输入端口交换到正确的输出端口
 - 链路不是单独占用，而是**统计复用**



IMP (Interface Message Processor)
in 1969



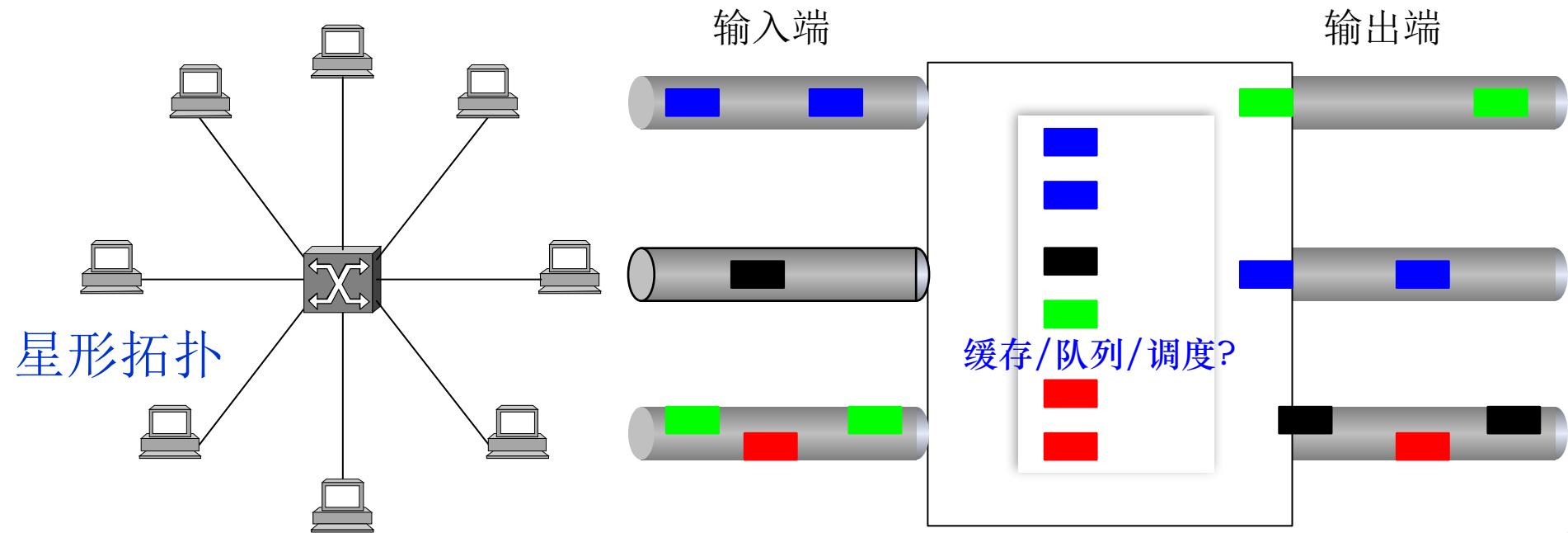
以太网交换机



高速交换机

什么是交换机?

- 有多个输入端和多个输出端的设备
- 可将分组或数据帧从一个输入端口传输到一个或多个输出端口
 - 称为交换（或转发）

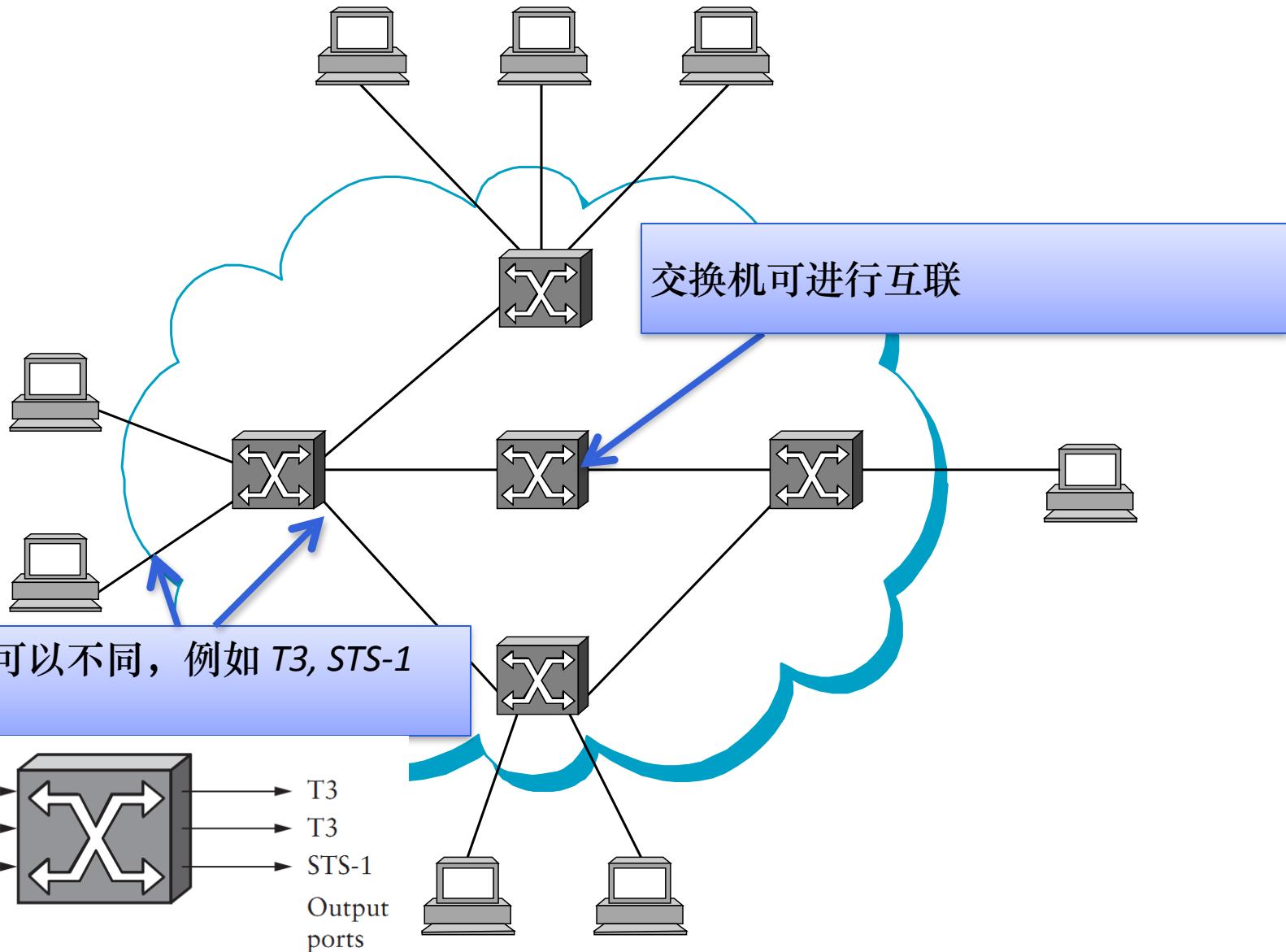




交换网络(*Switched Network*)

- 星形拓扑优点
 - 构建网络链接大量主机
 - 构建网络覆盖大片地理区域
 - 增加新主机不会影响现有主机的性能
 - 当网络基于交换机构建，交换机到每个节点的链路速度不互相影响
- 交换网络比媒介共享网络有更好的扩展性
 - 扩展性: 网络可以增大规模，连接更多节点，而性能并不明显下降

交换网络

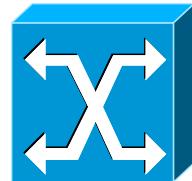


分组交换网络

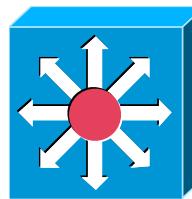
- 根据交换模式划分类别
 - 电话网络: 电路交换网络
 - 计算机网络: 分组交换网络
 - 根据转发地址, 可划分**交换机所属的层次**
 - 2层: 以太网交换机, 基于MAC地址
 - 3层: IP交换机, 基于IP地址



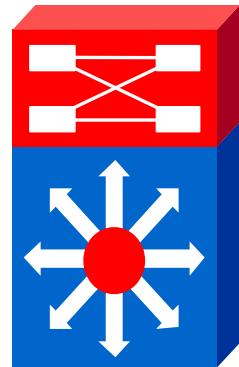
2层交换机,
e.g. Ethernet



通用交换机,
e.g. ATM, X.25



3层交换机



多层交换机

思科公司所使用的交换机图标



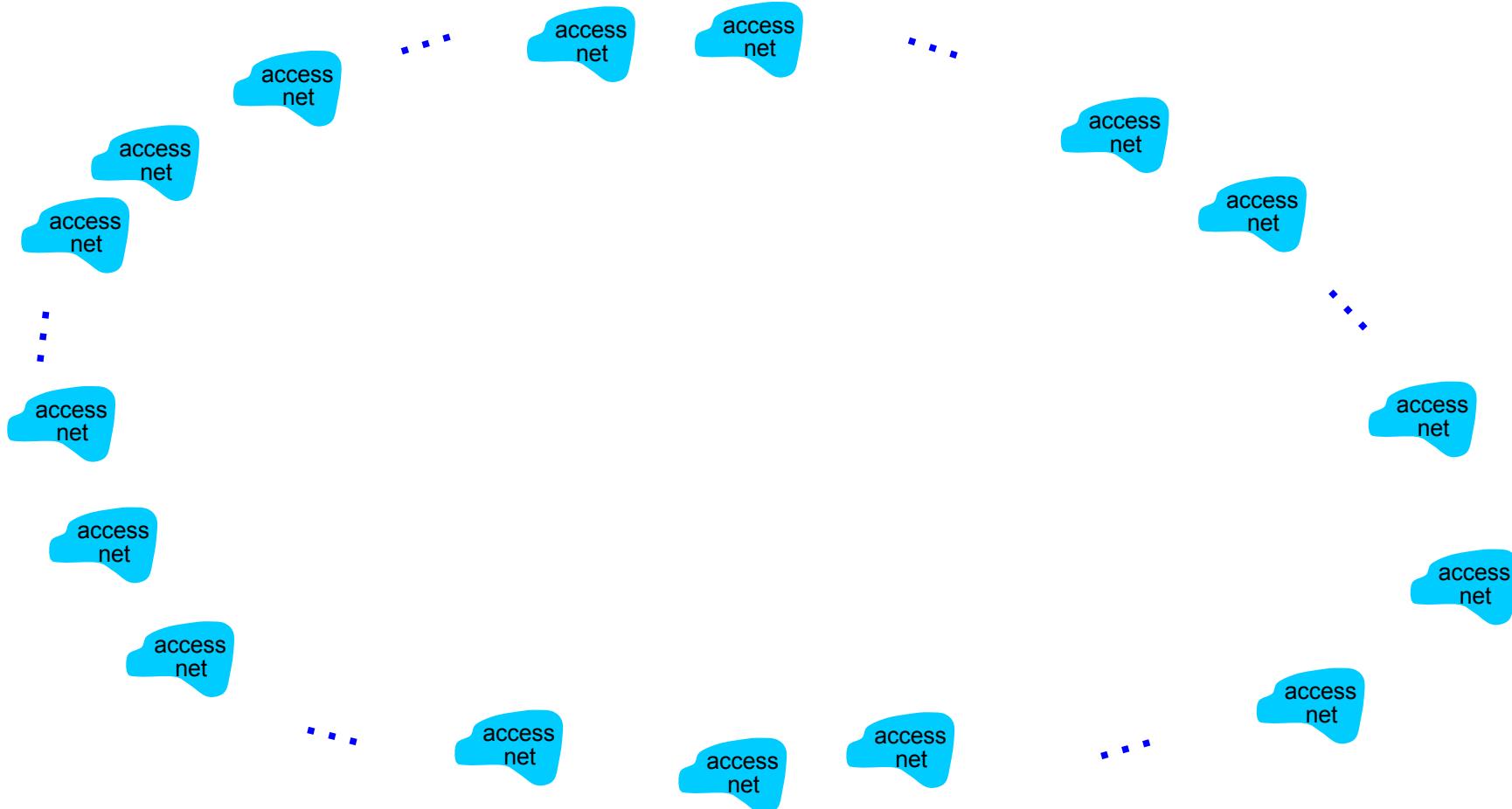
互联网架构：网络互联的网络

- ❖ 终端系统通过接入网服务商接入互联网
 - 住宅、公司和大学网络服务提供商(**ISPs**)
- ❖ 接入网服务商相应的也必须互联互通
 - ❖ 如此，任意两台主机可以互相发送分组
- ❖ 如此生成的网络互联的网络非常复杂
 - ❖ 网络的演进是由**经济**和**国家政策**所驱动



互联网架构：网络互联的网络

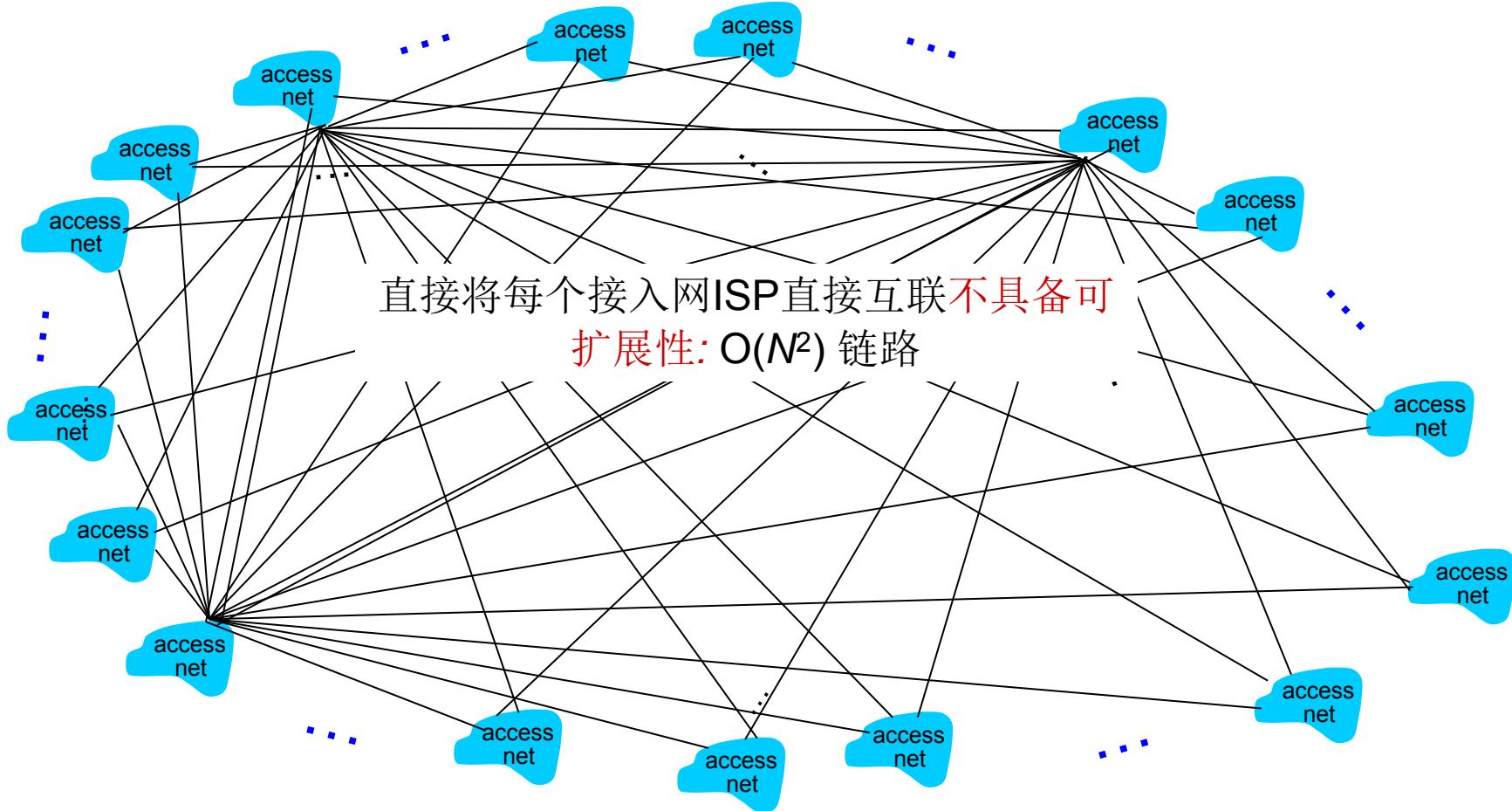
问题：给定数百万计的接入网络服务提供商，如何将其互联互通？





互联网架构：网络互联的网络

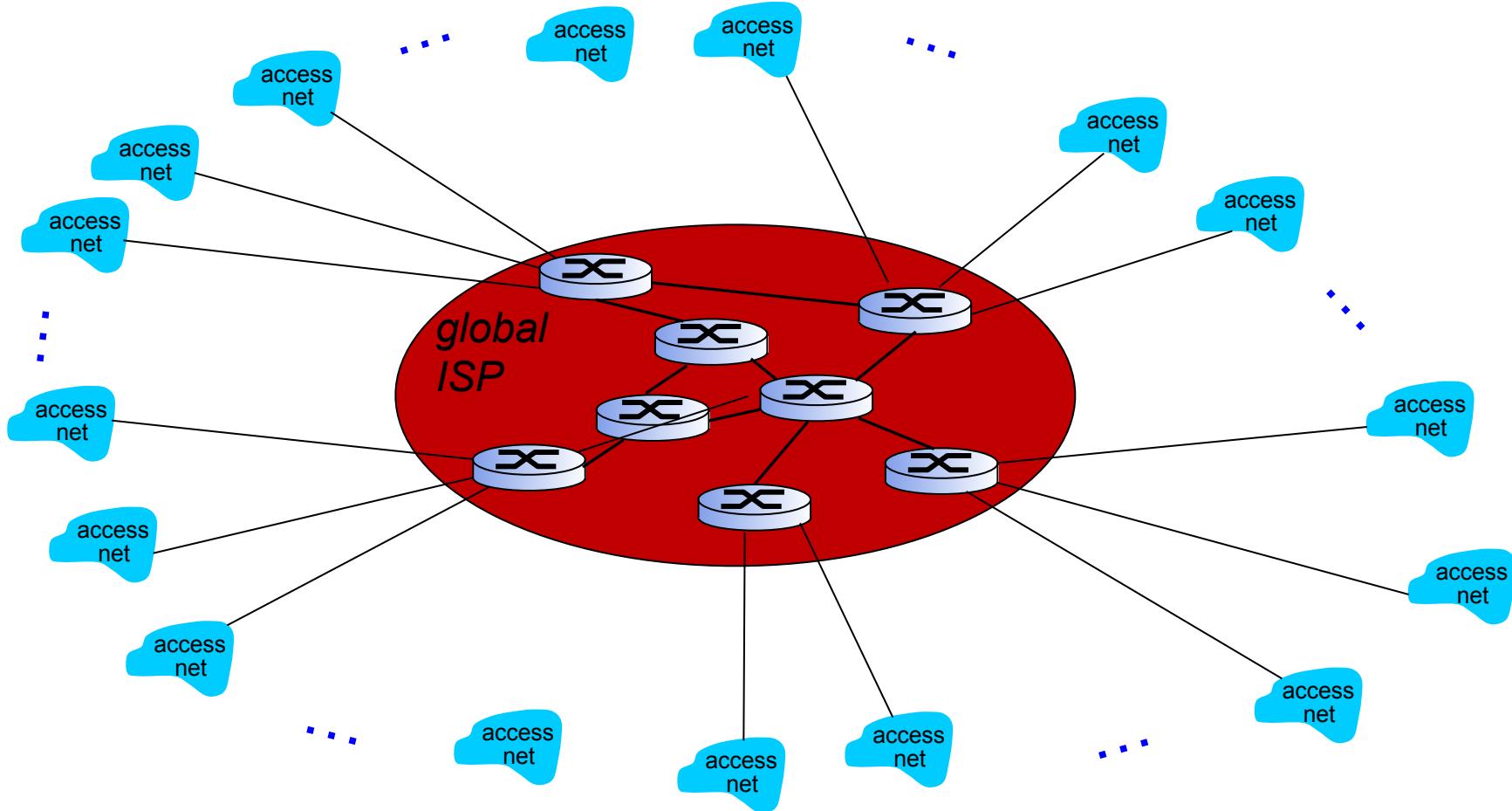
选项：将每个接入网服务提供商与其他接入网服务提供商直接相连？





互联网架构：网络互联的网络

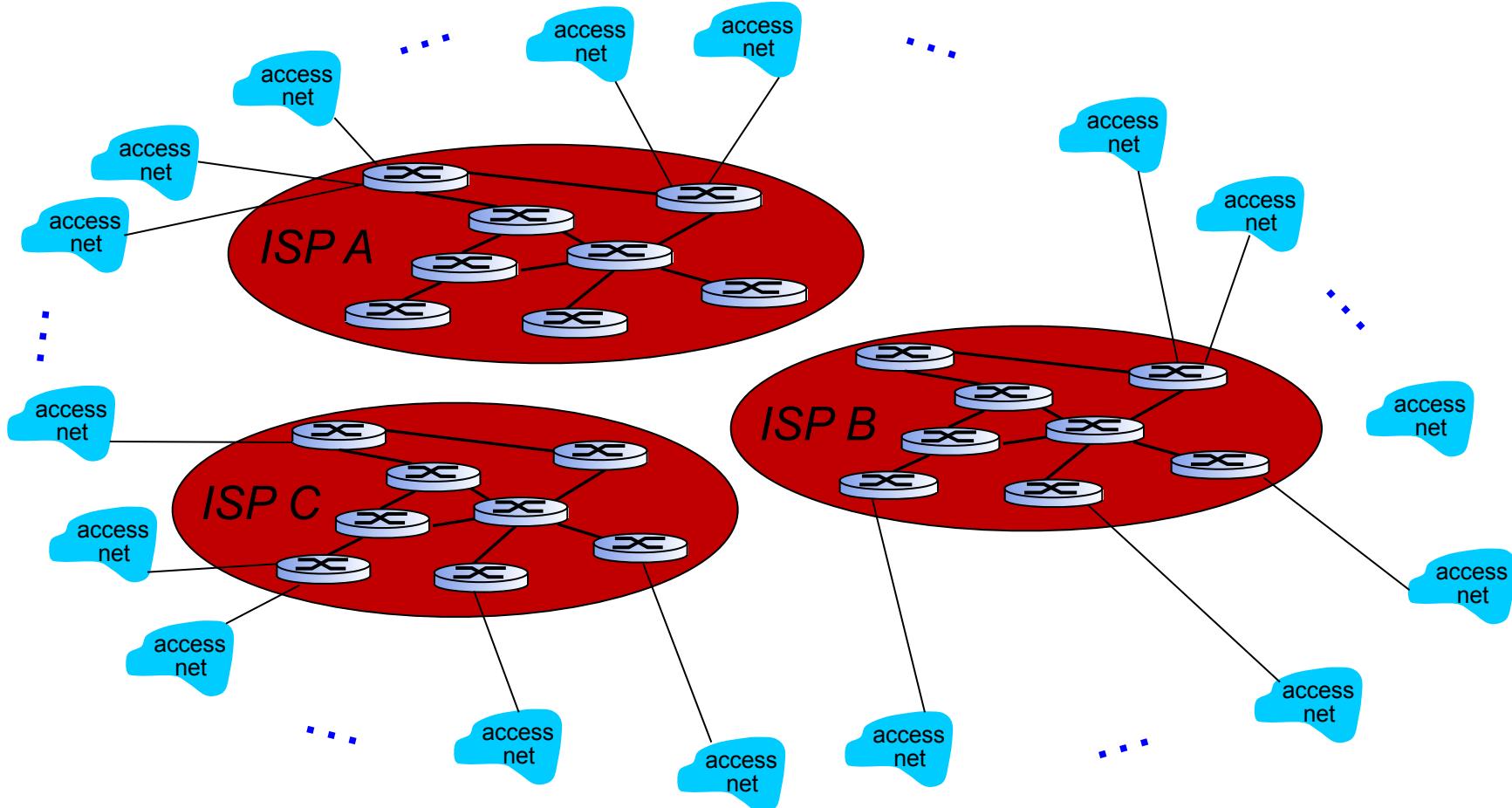
选项: 将每个接入网*ISP*通过一个全球传输*ISP*进行互联? 用户*ISP*和 服务*ISP*签订流量结算协议.





互联网架构：网络互联的网络

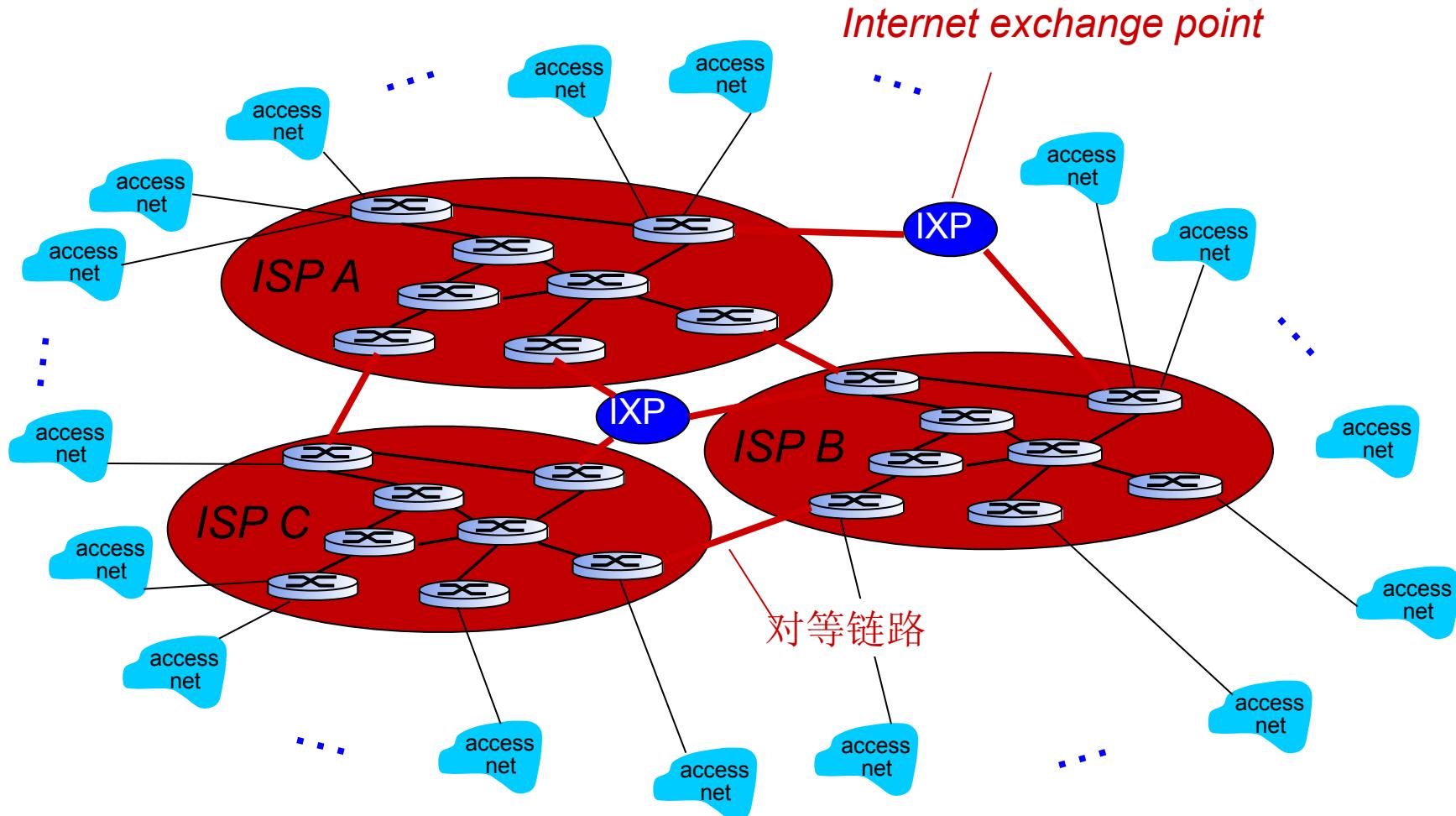
但是如果这样的全球传输ISP是重要的业务，则一定会有多家竞争者





互联网架构：网络互联的网络

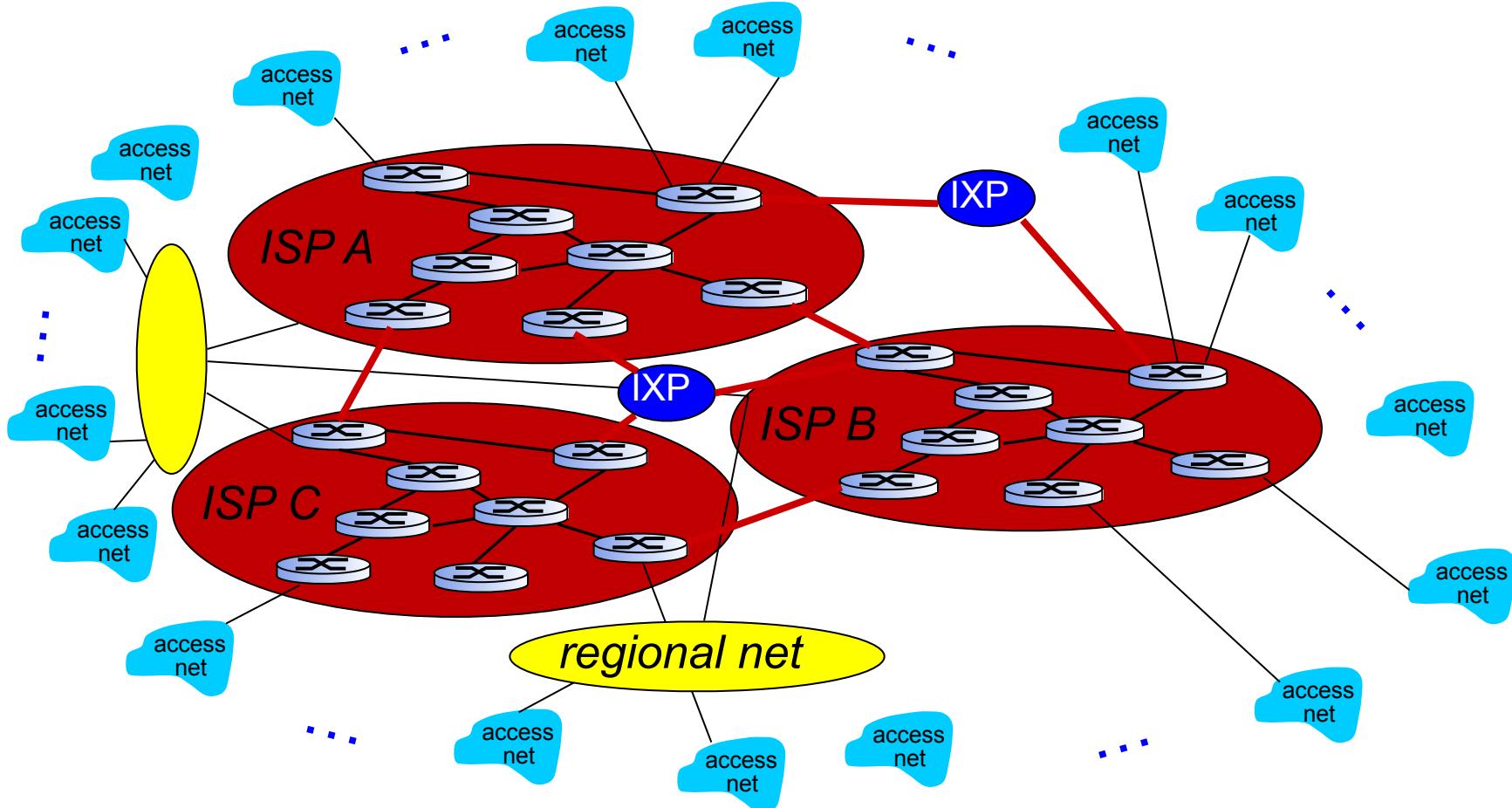
但是如果这样的全球传输ISP是重要的业务，则一定会有多家竞争者 那么这些ISP也必须互联互通





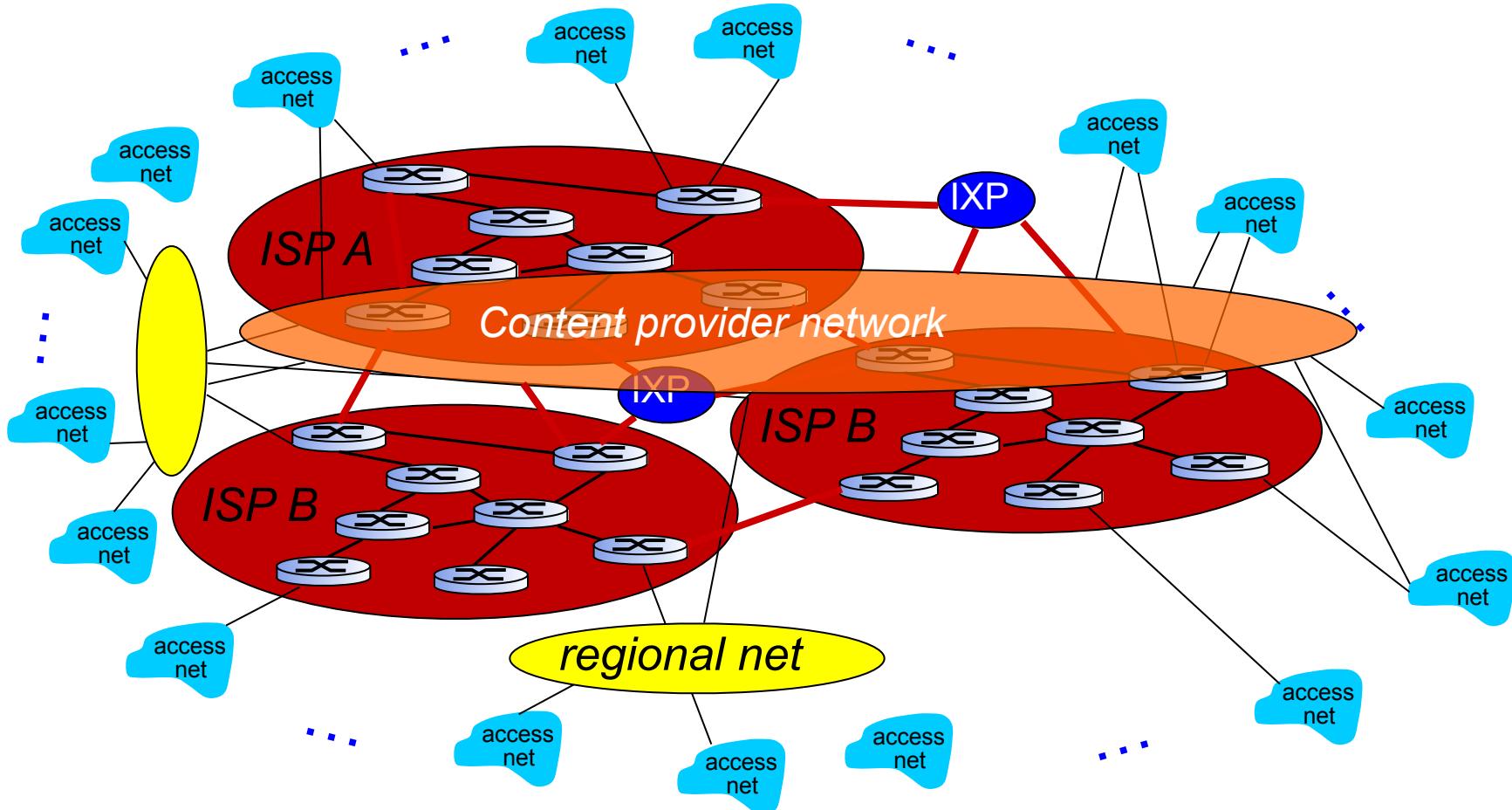
互联网架构：网络互联的网络

...而且也可能出现将接入网连入公共互联网的区域网络服务运营商

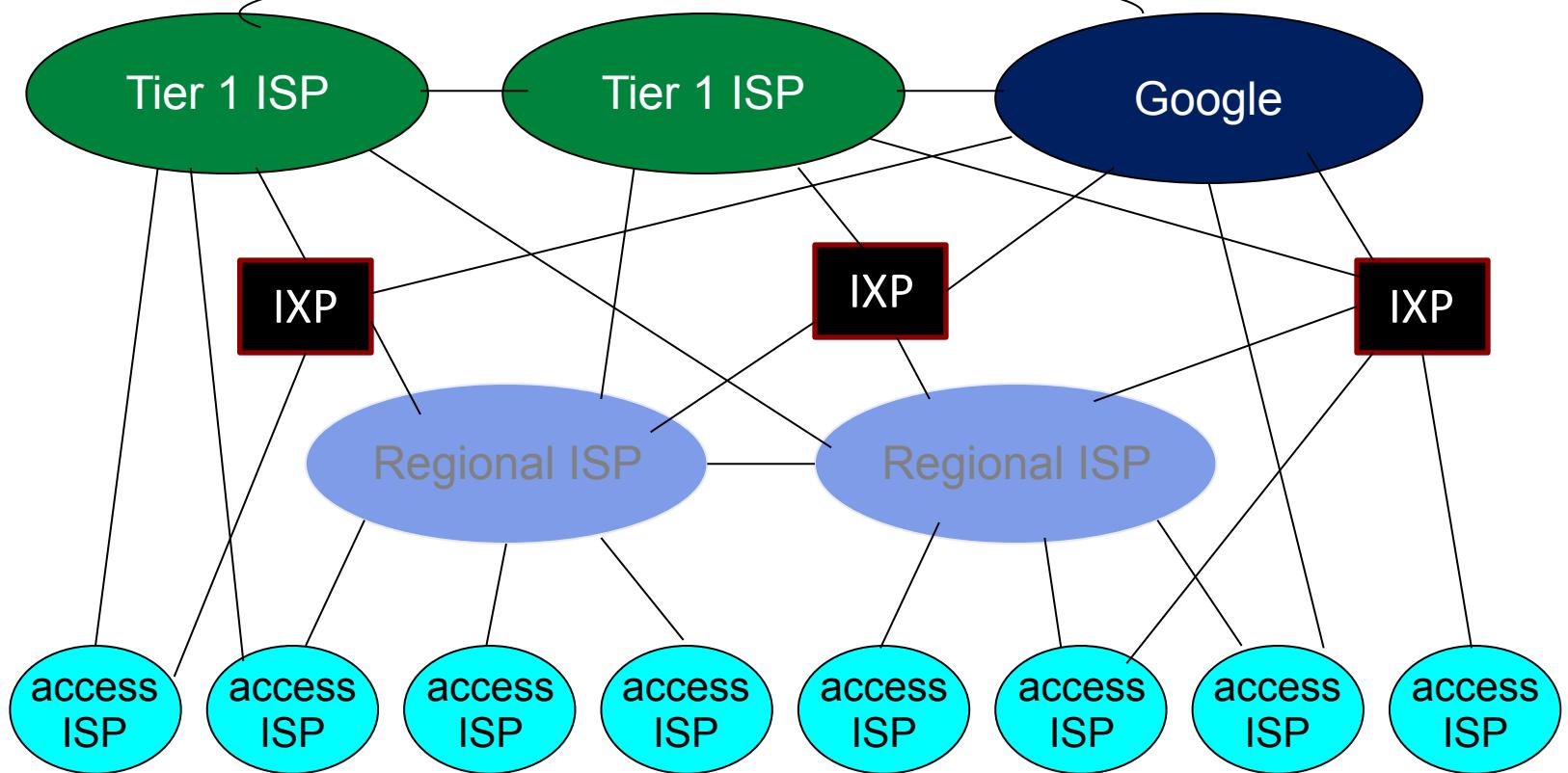


互联网架构：网络互联的网络

... 此外，内容服务商 (例如, Google, Microsoft, Akamai)也可能建设自己的网络，将服务和内容推近端用户

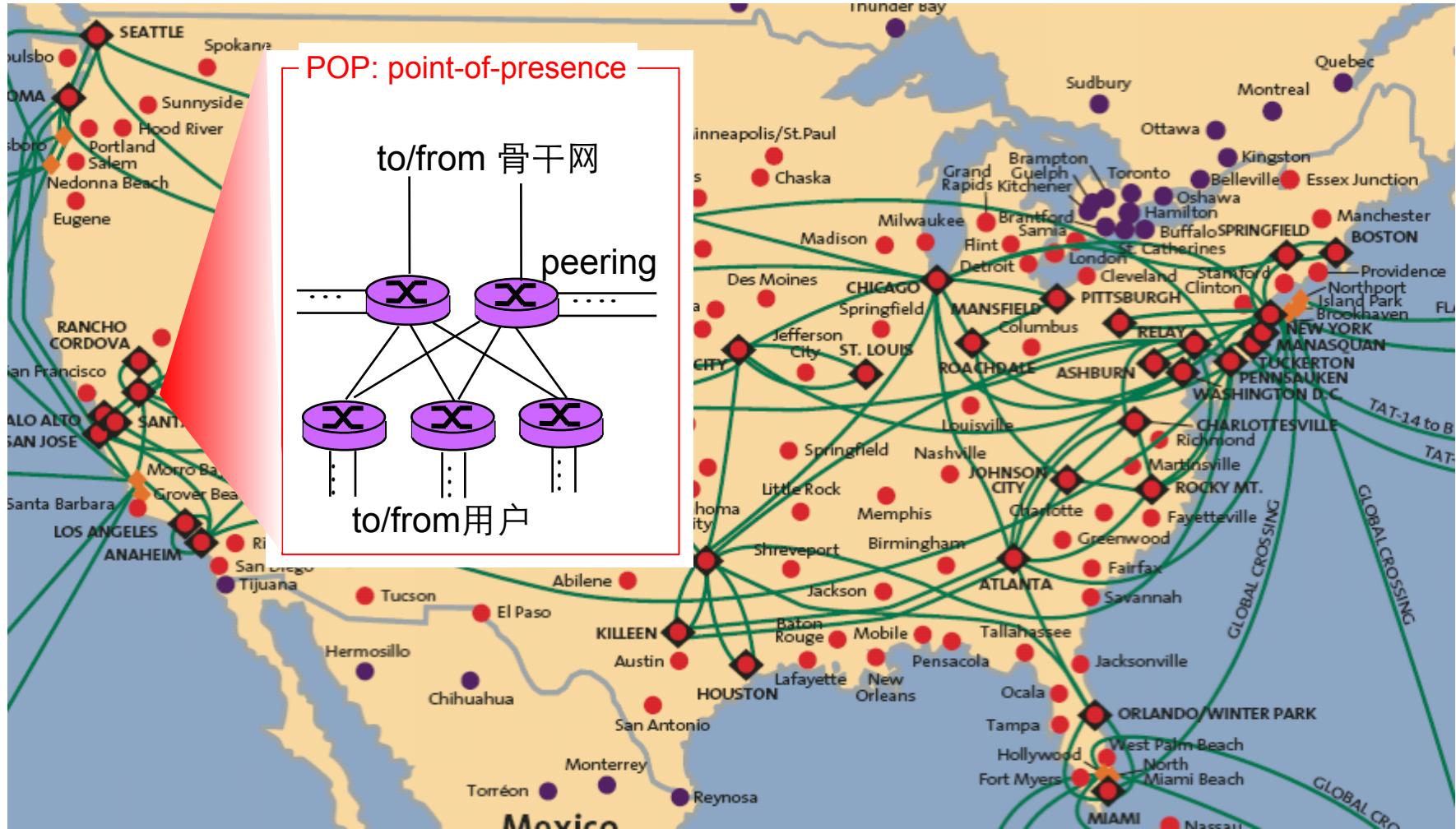


互联网架构：网络互联的网络



- 网络中心: 少量充分互联的大型网络
 - 顶级商业ISPs (e.g., Level 3, Sprint, AT&T, NTT, China Telcom, Unicom), 全国 & 全球覆盖
 - 内容服务网络 (e.g, Google): 建设私有网络将自己的数据中心直接连入互联网, 而不通过顶级或者区域网络服务提供商

顶级ISP: 例如 Sprint



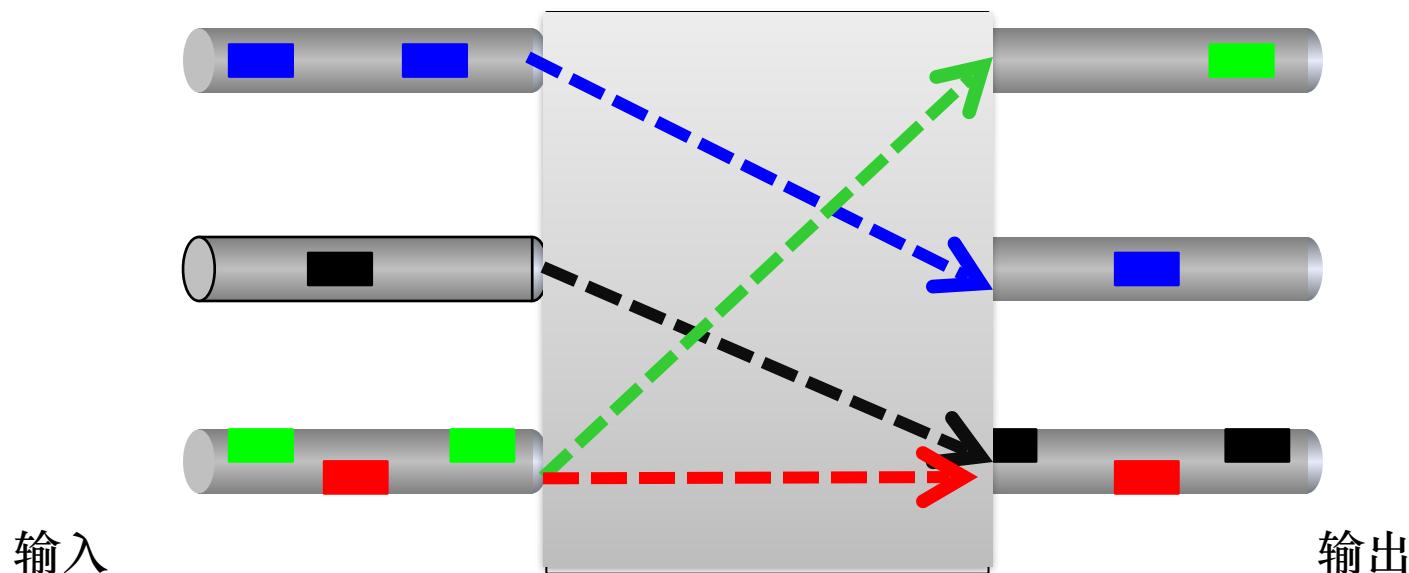


提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- → 交换和桥接
 - 数据报
 - 虚电路
 - 源路由
 - 网桥和局域网交换机
- 互联网基础
- 路由
- 实现和性能
- 总结

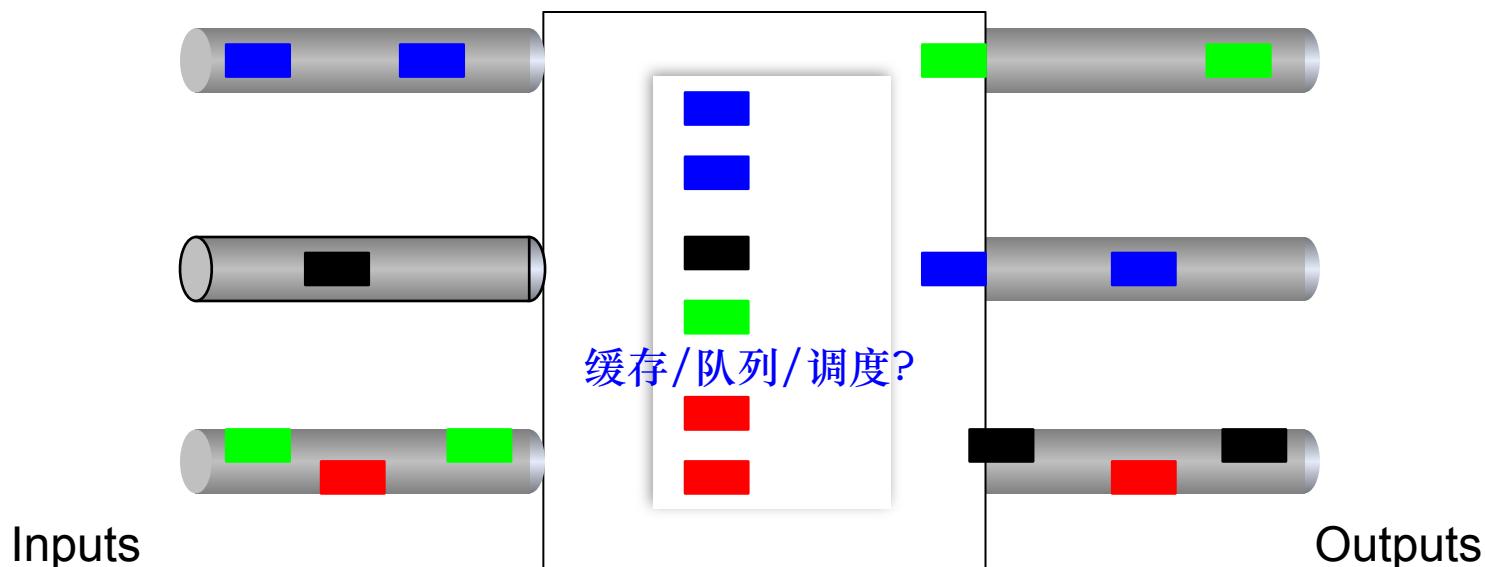
分组交换网络的问题

- 通过交换机连接形成的网络，存在一些通用问题
 - 如何确定一个分组的输出端口？



分组交换网络的问题

- 通过交换机连接形成的网络，存在一些通用问题
 - 如何确定一个分组的输出端口？
 - 如何处理有限的输出带宽？





交换/转发

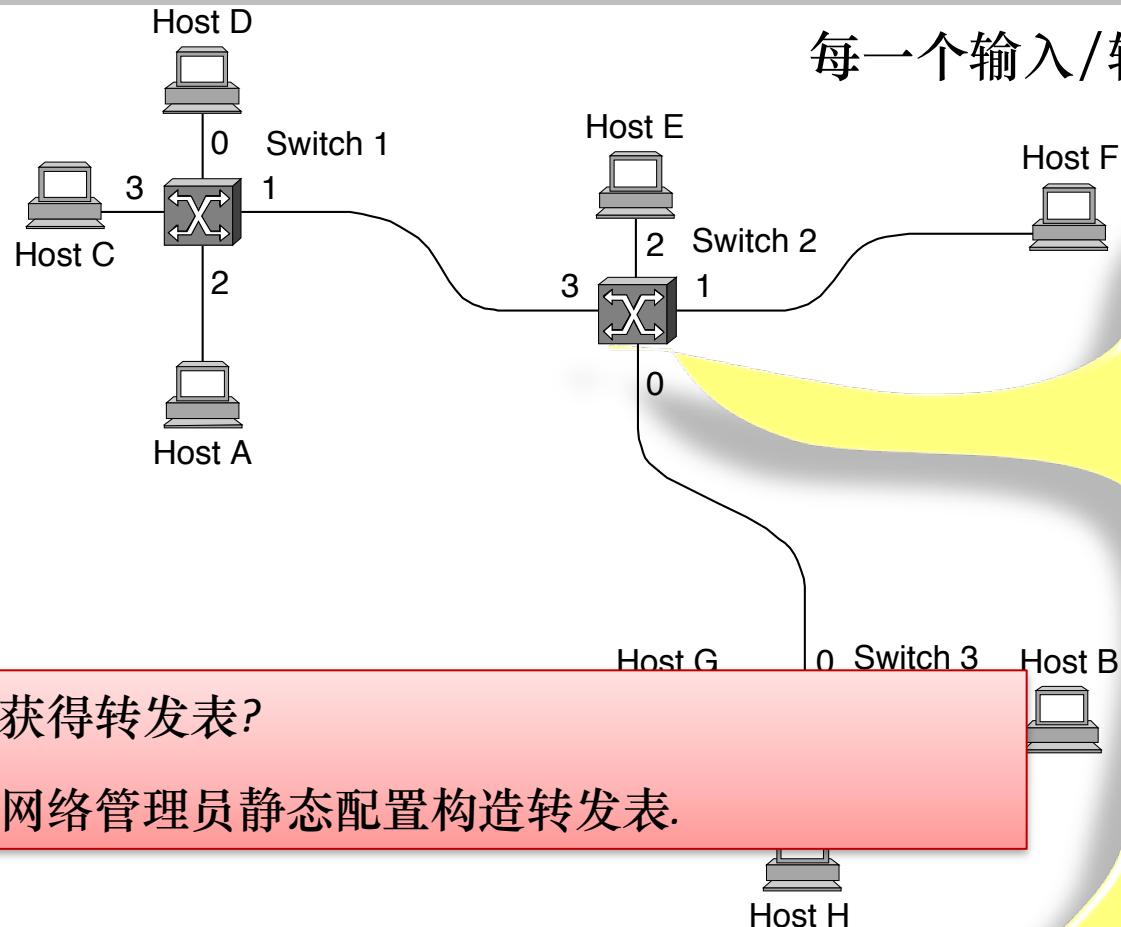
- 交换/转发
 - 将从某一个端口收到的分组/数据帧交换至一个或多个端口的行为
- 如果确定一个分组的输出端口?
 - 如何基于分组/数据帧中的地址确定其输出端口?
 - 采用一个标识, 嵌入分组的首部
 - 三种方法
 - 数据报, 无连接
 - 虚电路, 面向连接
 - 源路由



数据报

- 数据报
 - 数据报, 传输的数据块, 事实上即分组
- 数据报转发方法
 - 标识: 目的地址
 - 每一个分组中包含完整的目的地址
 - 交换机决策: 基于转发表
 - 交换机查询转发表确定分组的输出端口
 - 转发表的构造与维护: 基于收集的桥接/路由信息

数据报转发示例



Destination	Port
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0

Forwarding table in switch 2

如何获得转发表?

假设网络管理员静态配置构造转发表.

如果网络较为复杂, 且拓扑结构动态变化, 到达目的主机存在多条路径

新的问题, 称为 *Routing*(路由)



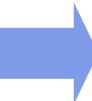
数据报网络

- 特点
 - 主机可以随时随地发送分组
 - 主机无法知道网络是否可以传送该分组或目的主机是否可以接收
 - 每个分组的转发均基于首部的目的地址, 分组之间相互独立
 - 一台交换机或一条链路出现故障时, 并不会对通信产生任何严重的影响
- 小结
 - 无连接
- 实例
 - ARPAnet, 交换型Ethernet



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
 - 数据报
 - 虚电路
 - 源路由
 - 网桥和局域网交换机
- 互联网基础
- 路由
- 实现和性能
- 总结



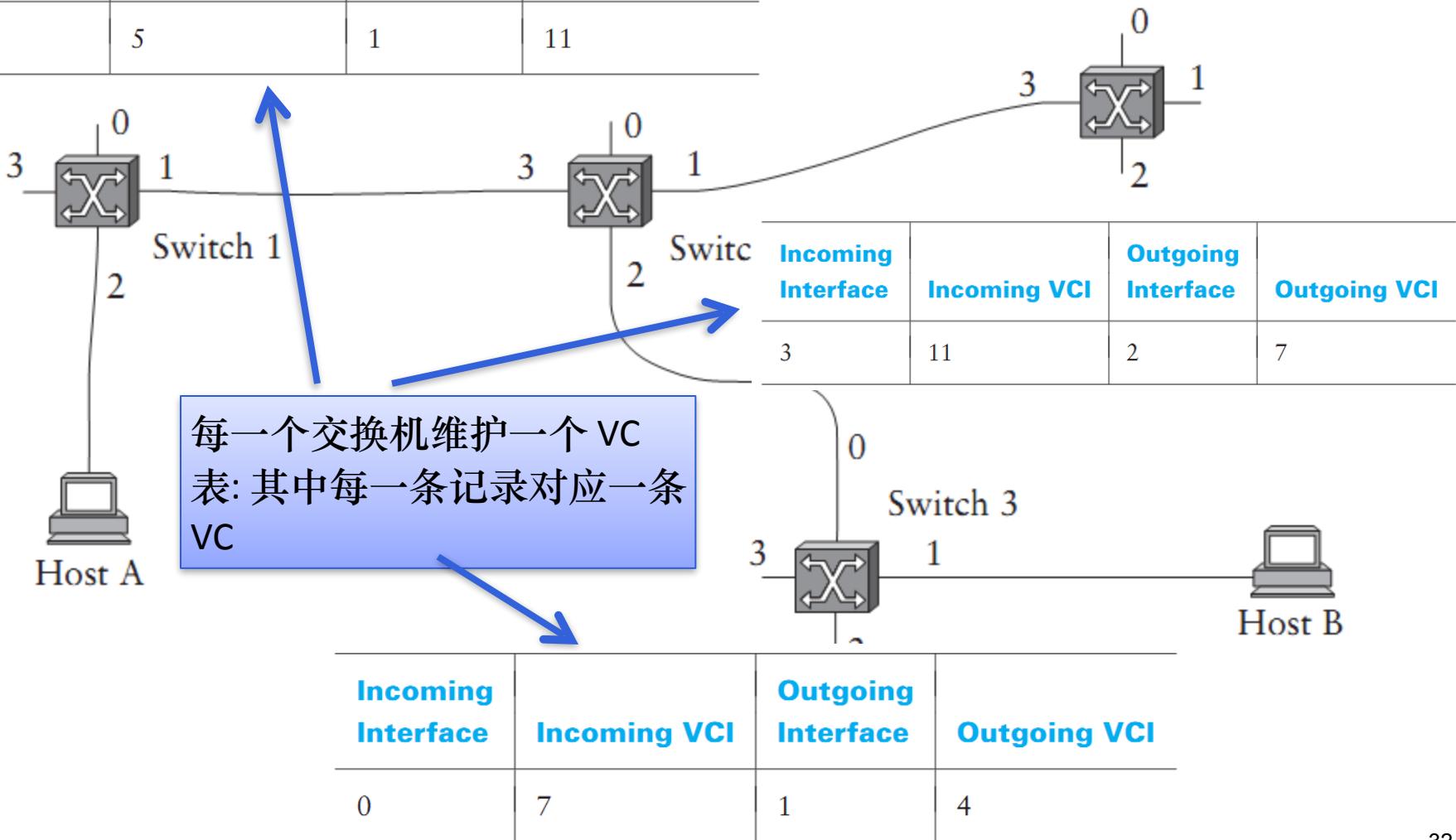


虚电路

- 虚电路(VC)
 - 借鉴传统电话网络中的电路交换思想
 - VC 是预先建立的连接
- 转发方式
 - 分组标识: 虚电路标识 (VCI)
 - 每一个分组包含一个 VCI
 - 交换机决策: 基于虚电路表
 - 交换机查询转发表确定分组的输出端口
 - 连接过程
 - 连接建立, 数据传输, 连接释放

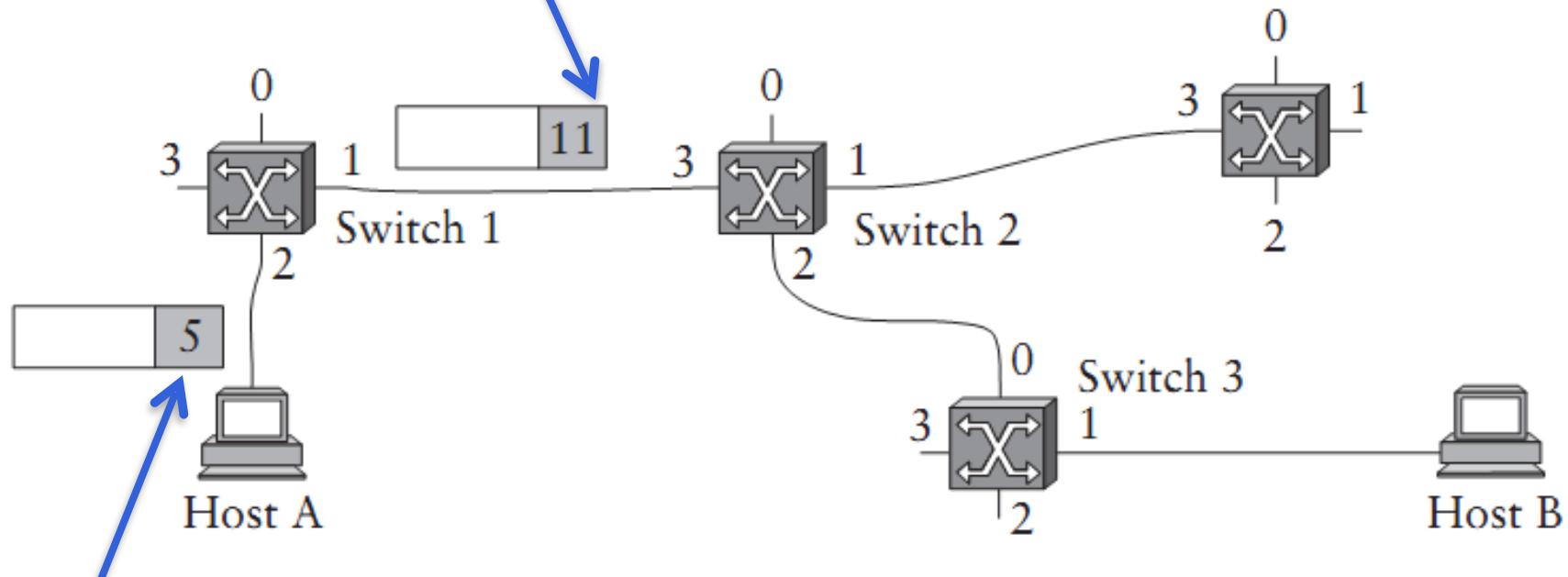
虚电路表

Incoming Interface	Incoming VCI	Outgoing Interface	Outgoing VCI
2	5	1	11



虚电路转发

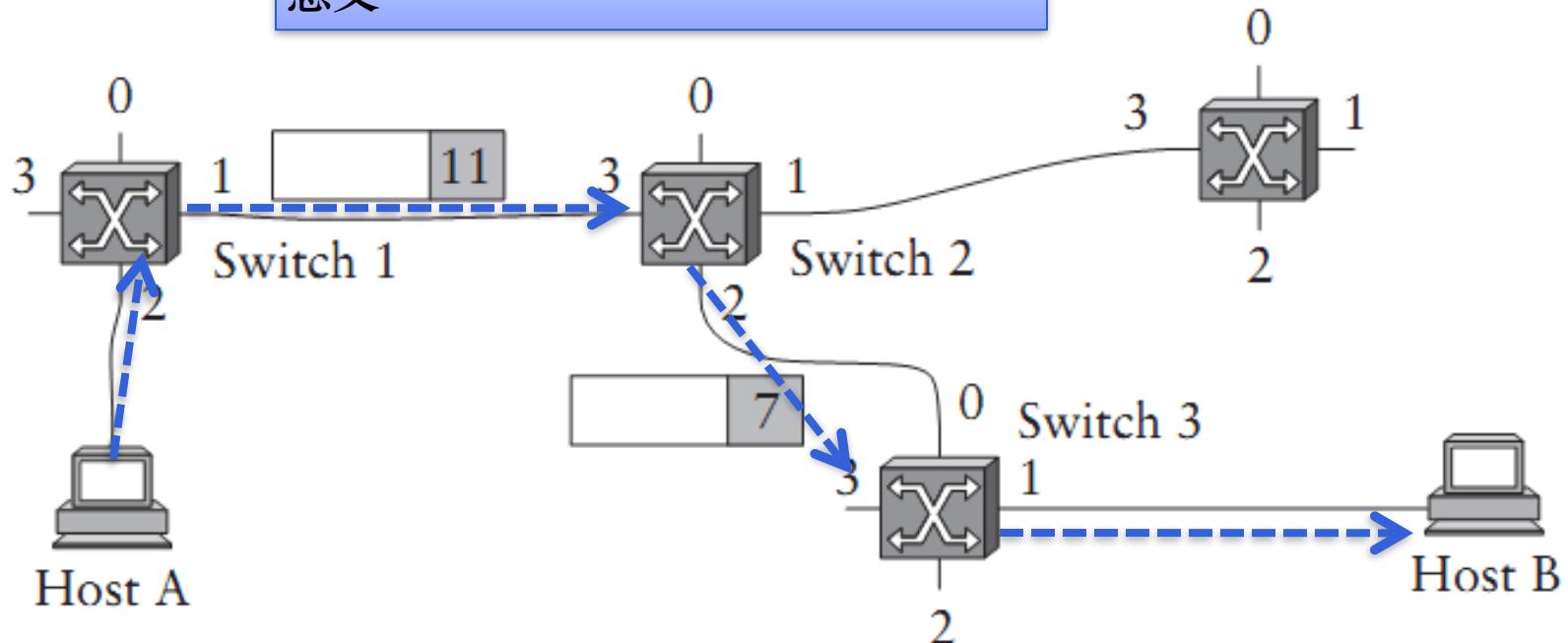
分组首部中的VCI值被查找表中对应的VCI值替换



每一个分组首部中包含一个VCI值用于标识其使用的VC

虚电路转发

VCI 具有“链路局部作用域”，而非全局意义



虚电路连接的数据传输：
分组沿着已建立的VC传输，最终按照发送方的发送顺序到达目的节点



虚电路的建立

- 两种类型的虚电路
 - 永久型虚电路(PVC)
 - 交换型虚电路(SVC)
- PVC
 - 网络管理员手工配置, 或由网络管理员产生信令探测形成
 - 长期生存的虚电路
- SVC
 - 主机动态发送信令建立连接
 - 实际网络中SVC更为普遍



SVC 信令

建立

- 源主机发送建立连接请求
 - 建立连接请求包含目的主机的完整地址
- 中间交换机
 - 记录建立连接请求消息的输入端口
 - 选择一个未使用的VCI值
 - 确定输出端口
 - 在虚电路表中插入一条记录
 - 转发建立连接请求至下一个交换机
- 目的主机
 - 选择一个可用的VCI值
 - 沿着连接建立请求消息的路径反向发送确认消息
- 中间交换机
 - 形成VC表记录

释放

- 源主机发送释放连接消息
-



虚电路网络

- 特点
 - 主机发送第一个数据分组前至少有一个RTT的时延, 用于连接建立
 - 虽然建立连接请求包含目的主机的完整地址, 但每一个数据分组中仅带有一个很小的标识, 分组首部开销较小
 - 如果一个连接上的交换机或链路出现故障, 连接就会被破坏, 需要建立新的连接
 - 连接建立过程为虚电路提供了资源预留
- 小结
 - 面向连接
- 实例
 - X.25 , Frame relay, ATM



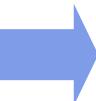
更深入的讨论

- VC 信令是一种资源预留
 - 逐跳流量控制
 - 端到端服务质量保证
- 不同的资源策略
 - 资源: 交换机内部的缓存
 - 资源问题
 - 竞争: 多个分组争用一条输出链路
 - 拥塞: 缓存被过多的分组所占用导致溢出或分组丢弃
 - 不同的策略:
 - 数据报网络: 拥塞发送后的恢复
 - 虚电路网络: 资源预留避免拥塞



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
 - 数据报
 - 虚电路
 - 源路由
 - 网桥和局域网交换机
- 互联网基础
- 路由
- 实现和性能
- 总结

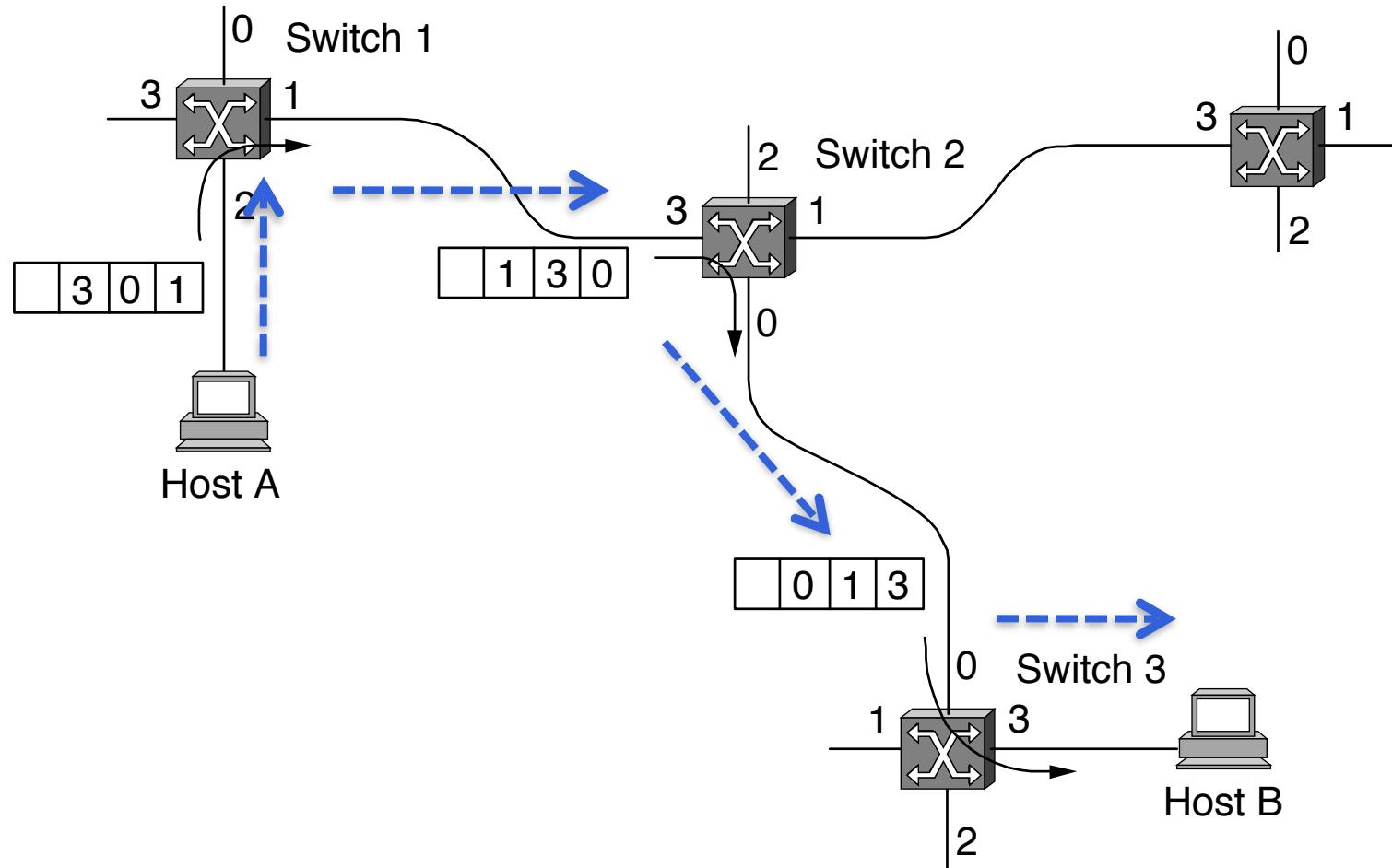




源路由

- 源路由
 - 源节点为每一个分组提供网络交换分组时所需的路由信息
- 源路由方法
 - 分组识别: 整条路径上所有交换机的端口序列
 - 分组首部包含从源节点到目的节点的整条路径的所有交换机的输出端口编号
 - 交换决策: 基于分组首部的路由
 - 交换机通过首部中的“下一个端口”指针读取下一个转发端口进行分组转发

源路由转发机制





源路由

- 特点
 - 源节点知道全部的网络拓扑结构信息, 从而构造分组首部的源路由来指导交换机的分组转发
 - 首部长度不固定, 由整条路径的跳数确定
 - 源路由选择存在一个网络的规模性问题
- 机制? 网络?
 - 源路由选择可以应用于数据报网络或虚电路网络
 - 在小规模的无基础设施无线网络中 (ad hoc网络), 源路由非常有效.



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
 - 数据报
 - 虚电路
 - 源路由
 - 网桥和局域网交换机
 - 透明网桥
 - 生成树算法
 - 广播和多播
 - 网桥的局限性
- 互联网基础
- 路由
- 实现和性能
- 总结



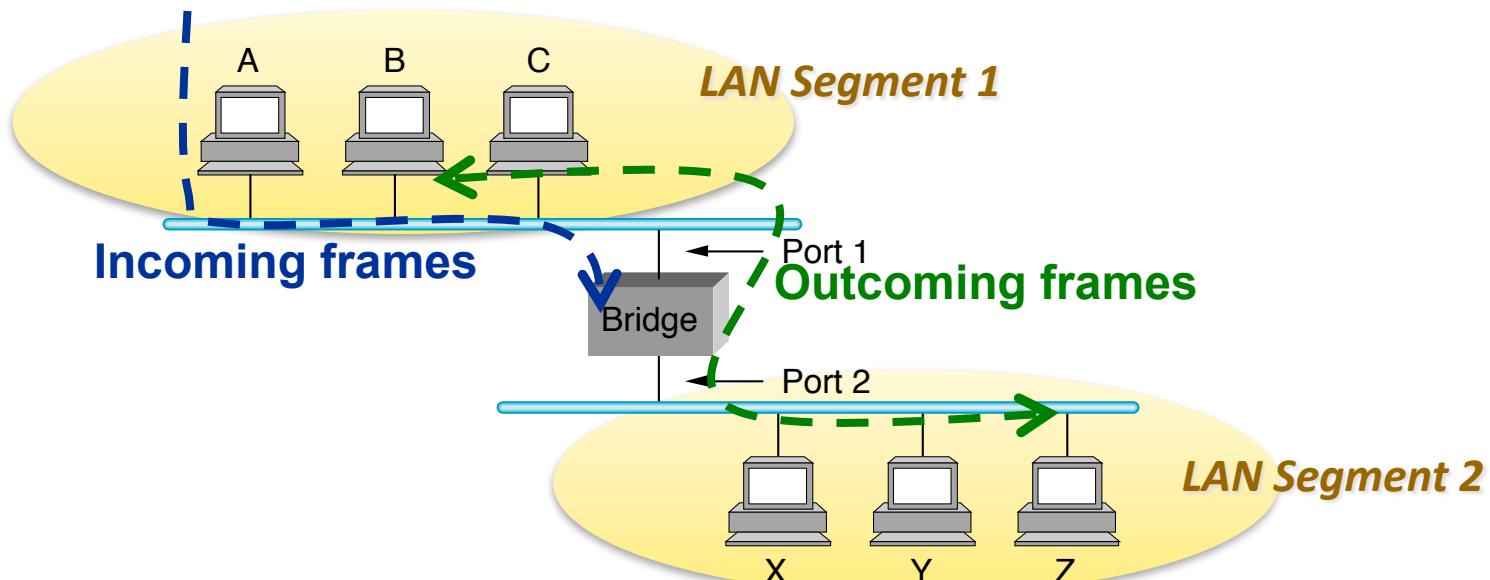


网桥和交换机

- 传统以太网的局限性
 - 最多2500m的覆盖范围
 - 最多容纳1024台主机
- 如何扩展网络?
 - 通过中继器再生放大信号
 - 设计一种新的节点实现多个以太网之间的数据帧转发
→ 网桥, 有时通称为 以太网交换机

网桥和局域网扩展

- 通过网桥扩展局域网
 - 网桥是交换机的一种实现
 - 单一局域网的总通信量为10Mbps, 连接n个网桥的吞吐量最多能够达到 $n * 10\text{Mbps}$
- 最简单的网桥
 - 将输入端口上接收的局域网的数据帧向所有的其他端口输出
 - 有必要吗?

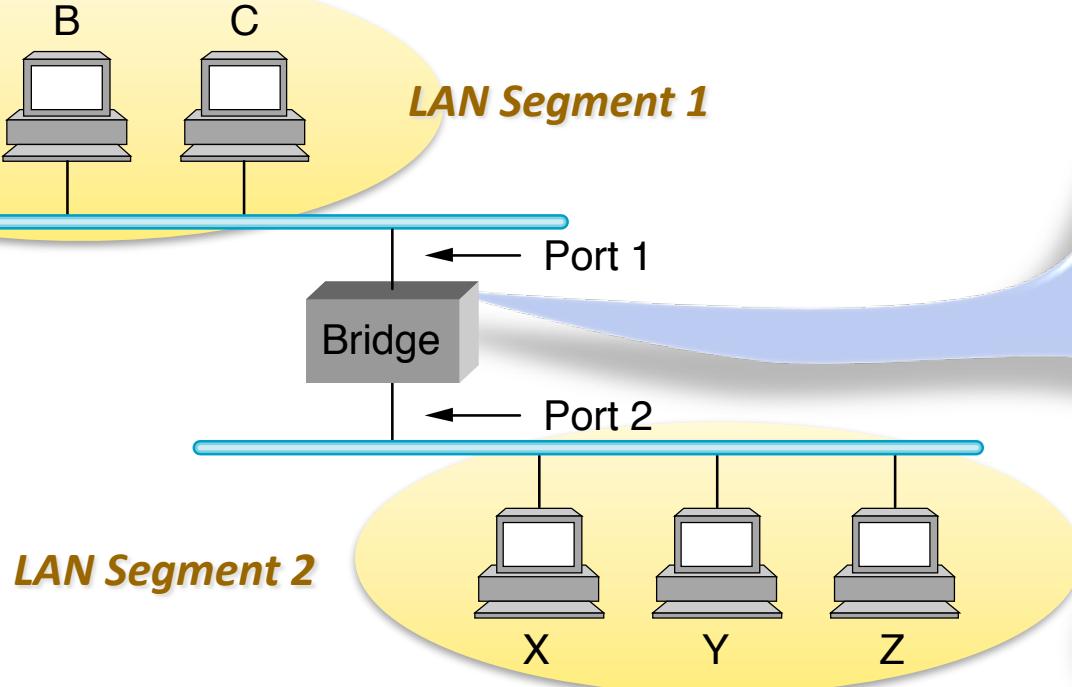




透明网桥

- 问题
 - 最简单的网桥向所有端口转发数据帧
- 动机
 - 根据需要进行转发
- 解决方案
 - 引入转发表

透明网桥



Forwarding table in bridge

Host	Port
A	1
B	1
C	1
X	2
Y	2
Z	2



透明网桥

- 新的问题
 - 如何自适应构造转发表?
- 动机
 - 根据数据帧转发过程自学习构造转发表
- 解决方案
 - 逆向学习
 - 主机与端口的映射关系学习
 - 监测接收数据帧的源地址信息

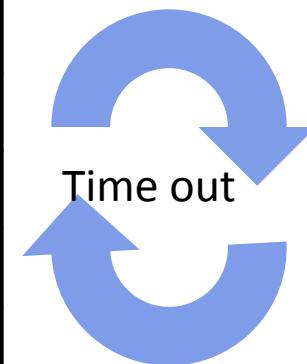
透明网桥

- 网桥启动时, 转发表为空
- 通过逆向学习法构建转发表
- 如果转发表中无对应记录, 则向所有其他端口转发数据帧

Host	Port

Booting up

Host	Port
A	1
B	1
C	1
X	2
Y	2
...	...



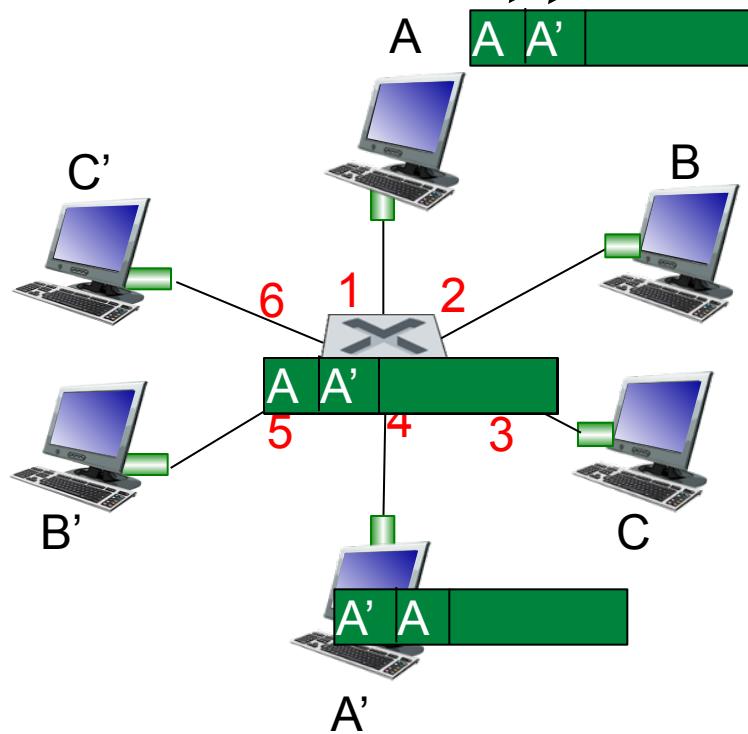
Host	Port
A	1
B	1
X	2
Y	2
...	...

网桥丢弃超时的记录(一段时间内未更新), 即, soft state

Self-learning, forwarding: example



- frame destination, A', location unknown: **flood**
- ❖ destination A location known: **selectively send on just one link**



MAC addr	interface	TTL
A	1	60
A'	4	60

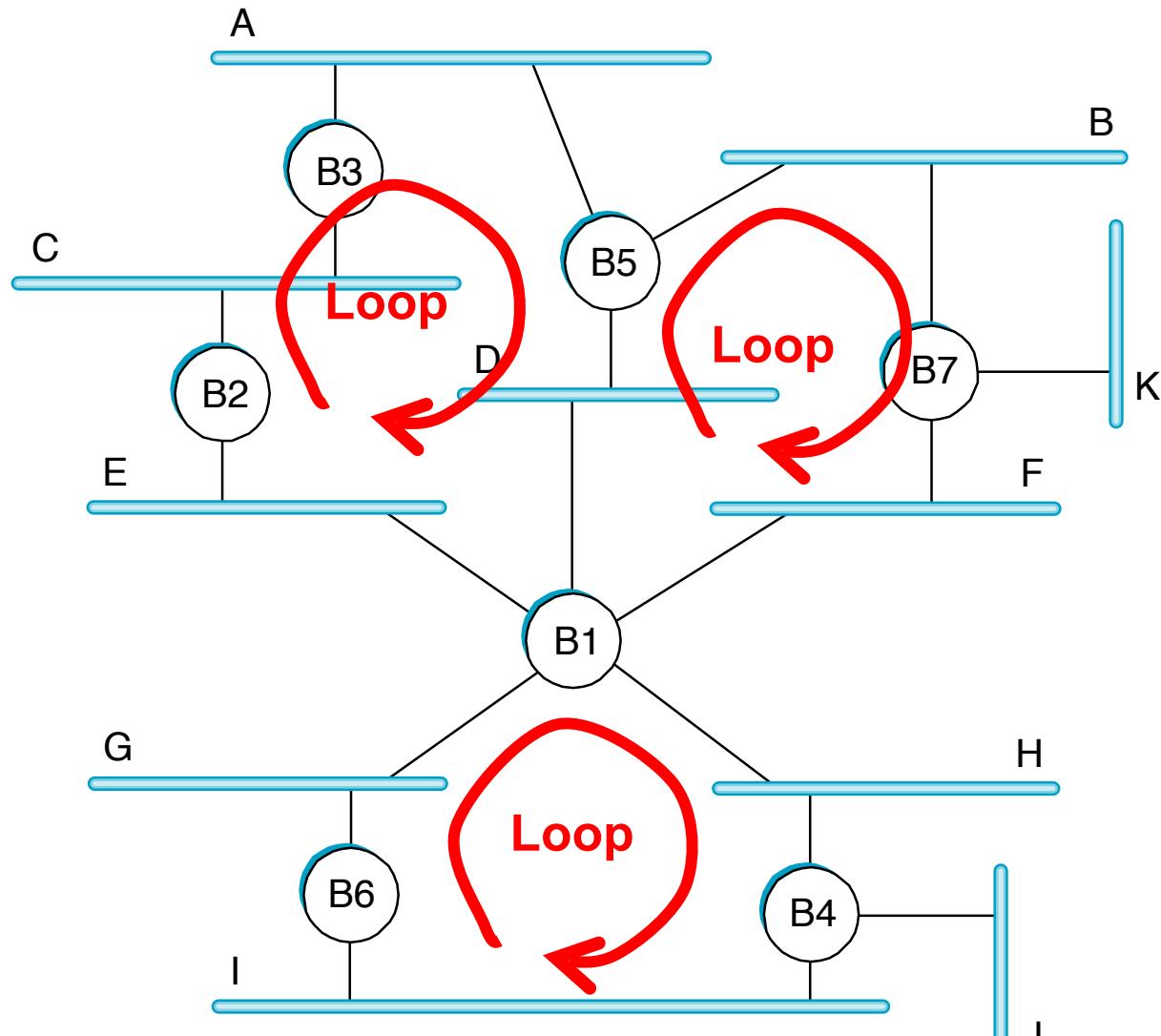
switch table
(initially empty)



生成树算法

- 新的问题
 - 网络环路
 - 广播和多播
- 动机
 - 网桥交换配置信息
- 解决方案
 - 生成树算法
 - 通过从扩展局域网的拓扑结构中去掉一些端口可能使其退化成为一颗无环树

局域网扩展产生环路

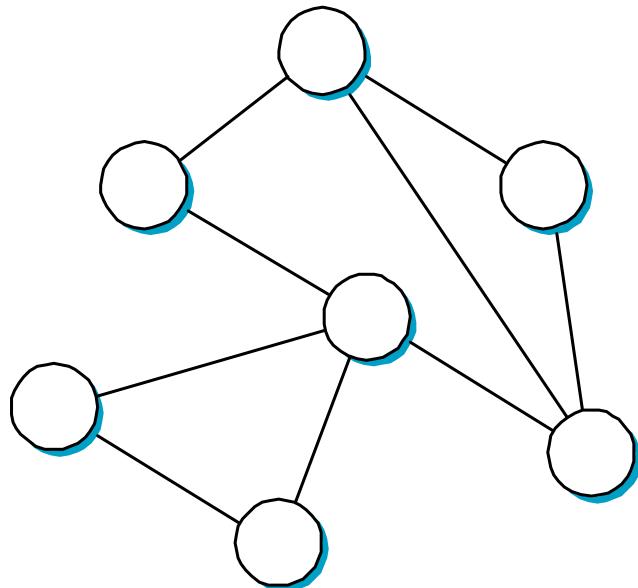


An example of Extended LAN with loops.

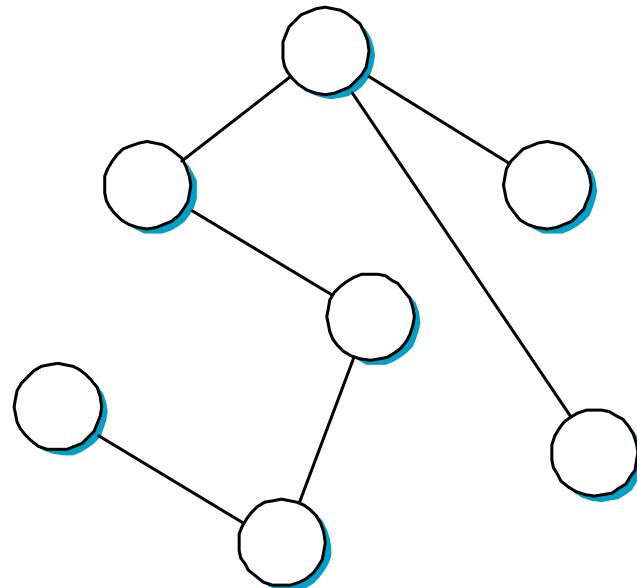
[以太网数据帧结构](#)

基于图论的生成树算法

- 生成(跨越)
 - 覆盖所有的节点
- 树
 - 无环路, 或任意两个节点之间只有一条路径



(a) 有环图



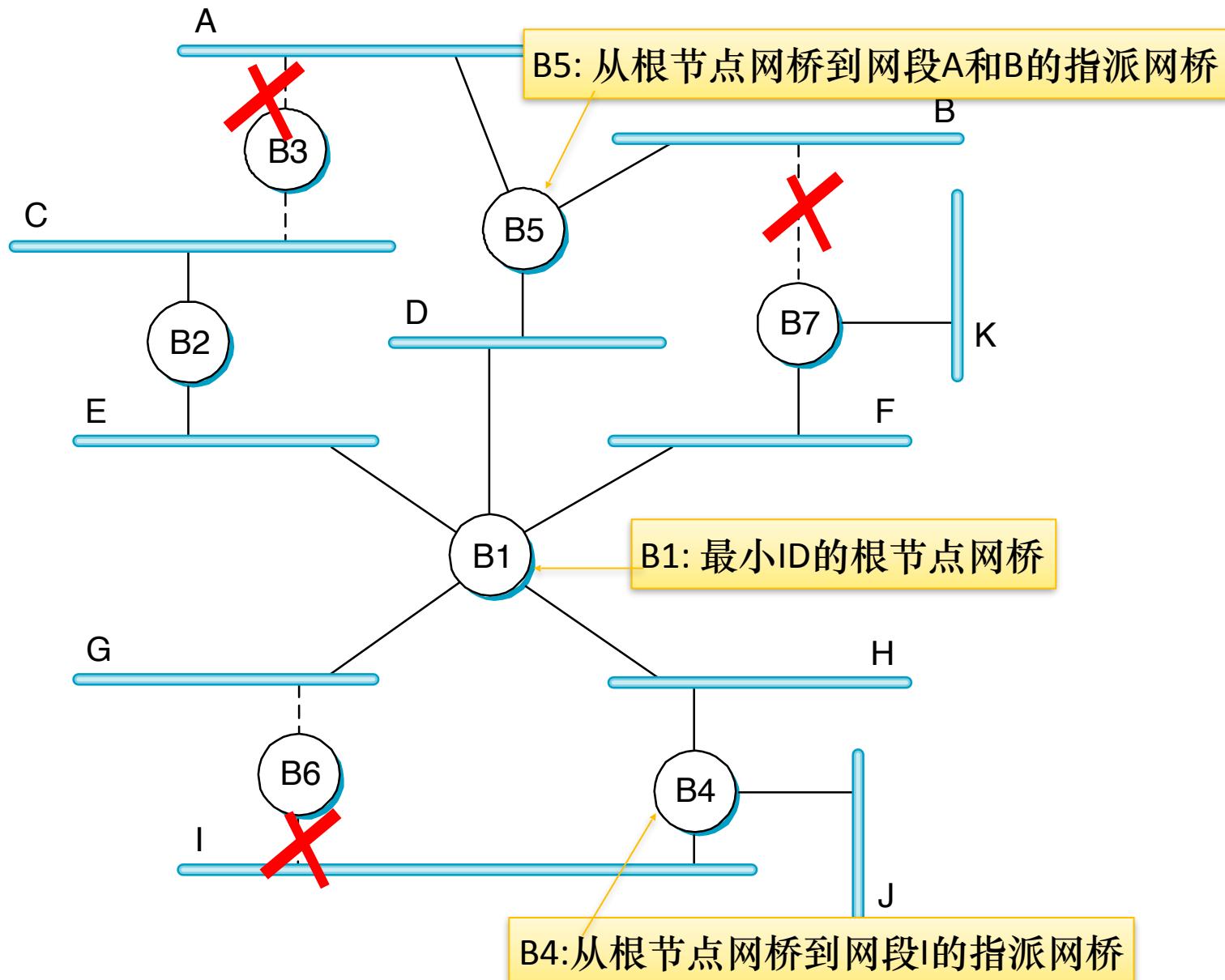
(b) 生成树



理想的生成树算法

- 假设
 - 所有交换机均知道扩展局域网的拓扑结构
 - 每个网桥有一个唯一的标识 B_x , x 为网桥 id
- 生成树节点的选择
 - 选择最小标识符的网桥为生成树的根节点
 - 选择离根最近的网桥为 LAN 的指派网桥(转发数据帧)
 - 每一个网桥根据其端口是否转发数据帧决定是否是指派网桥
- 转发机制
 - 根网桥总是向所有的端口转发数据帧
 - 其他网桥通过指派网桥对应的端口转发数据帧

局域网扩展(避免环路)





实用的生成树算法

- 去除假设
 - 所有交换机均知道扩展局域网的拓扑结构
- 动机
 - 设计交换机之间的信息交换机制
 - 实现一种动态的分布式算法
- **问题1:** 交换信息的设计, 称为**配置信息**:
 - 发送数据的源网桥的id
 - 发送网桥认定的根网桥的id
 - 从发送网桥到根网桥的距离 (跳数)



实用的生成树算法

- **问题2:** 设计 **分布式协商机制**
 - 最初每个网桥认为自己是根节点
 - 每个网桥生成配置消息并从每个端口发送出去
 - 当网桥的某个端口收到新的配置消息后, 更新每个端口的配置消息
 - 当网桥学习发现自己既不是根节点也不是指派节点时则停止发送配置信息

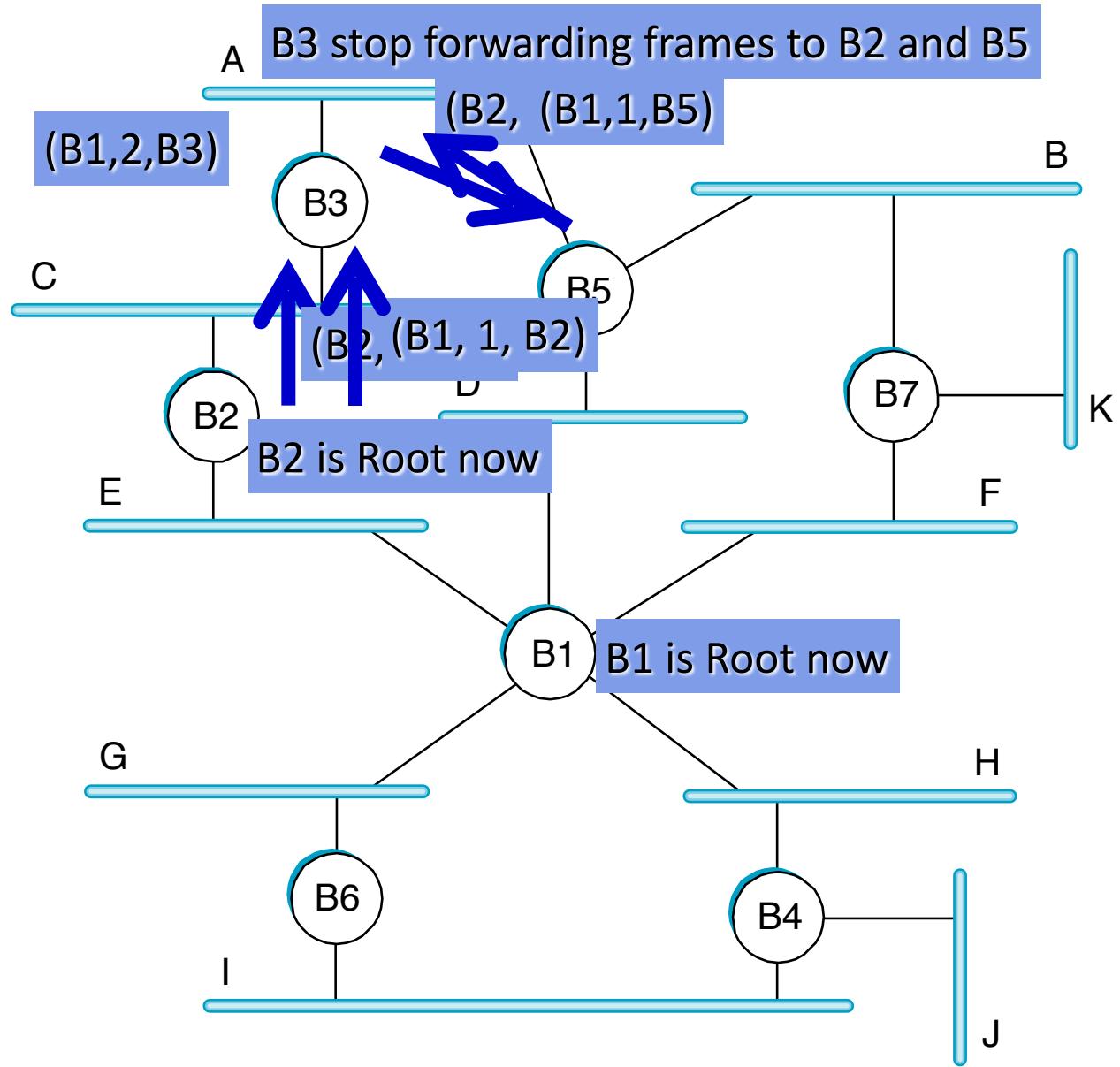


实用的生成树算法

- **问题3: 设计维护机制**

- 系统稳定时, 仅根网桥产生配置消息, 其他网桥仅在那些指派网桥的端口上转发这些配置消息
- 根节点周期性的发送配置消息
- 如果网桥在一段时间后仍未收到配置消息, 则重新宣布自己是根节点, 重新生成配置消息并转发

生成树算法示例

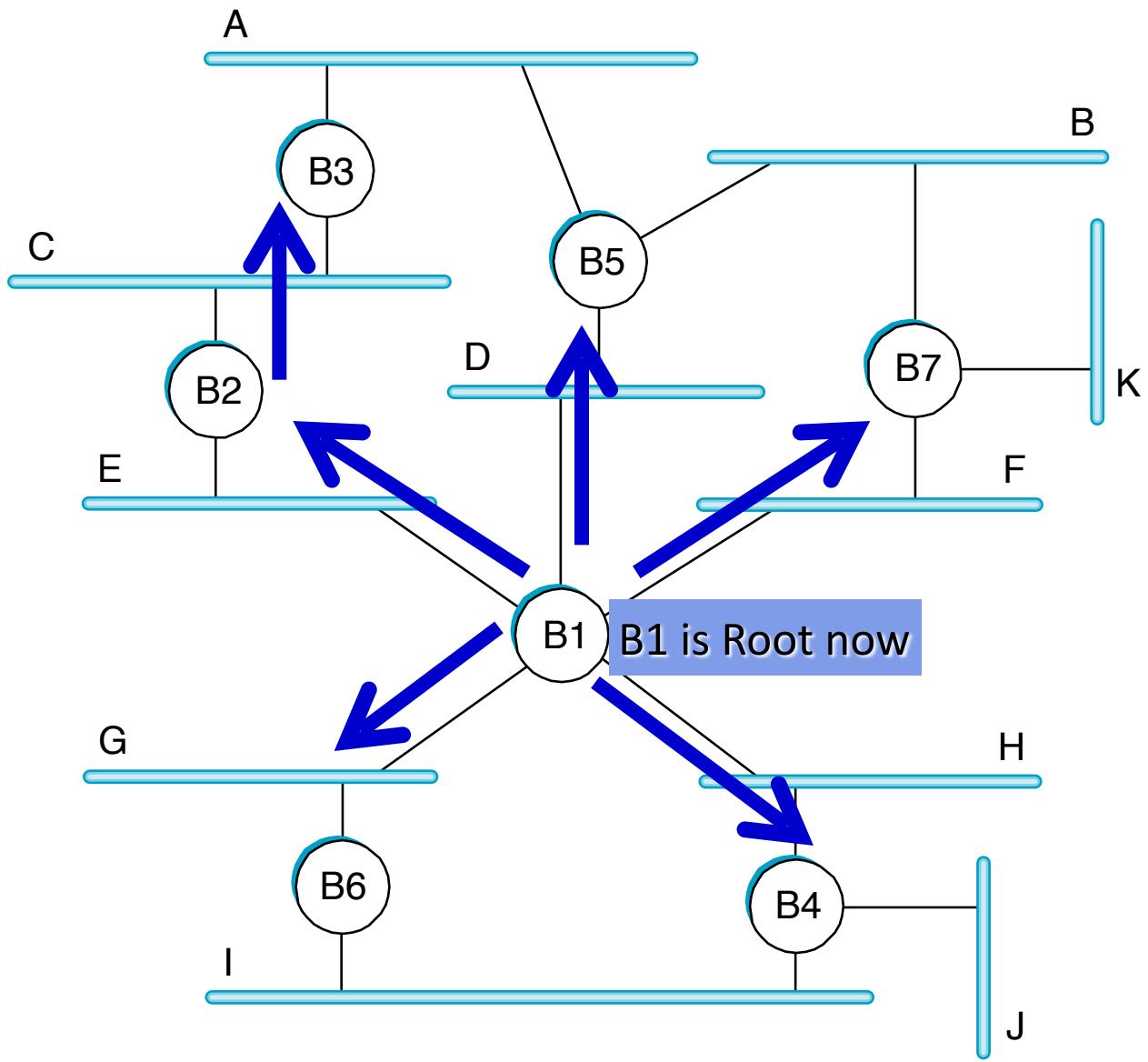


假设所有网桥同时上电

从节点X发出配置消息，声称从节点X到根节点Y的距离: (Y, d, X)

关注 B3

生成树算法示例



Assuming all bridges power on at the same time

Configuration message from node X claiming distance d from root node Y : (Y, d, X)

Focus on B3



广播和多播

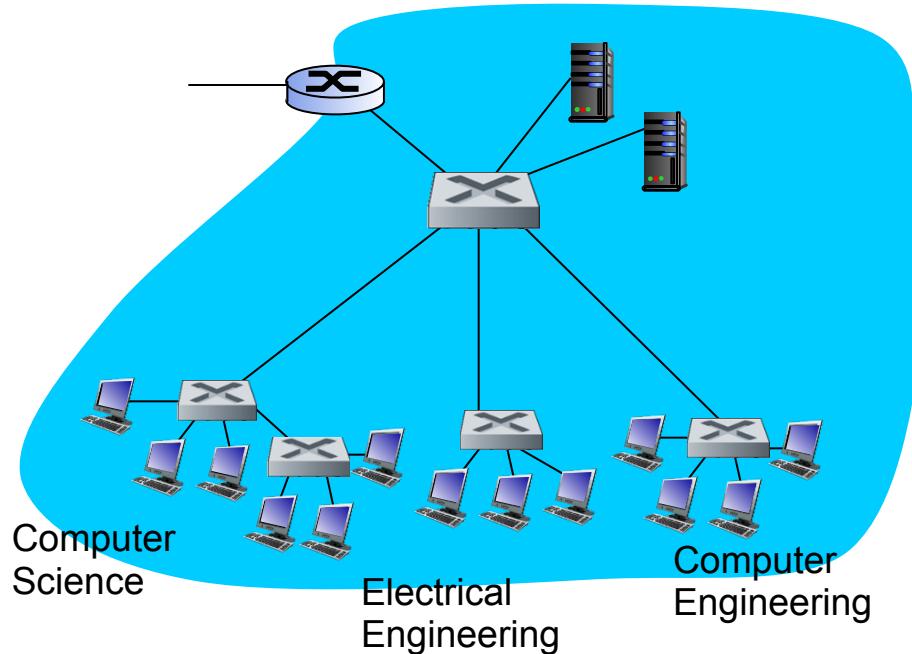
- 问题
 - 上述仅考虑了网桥的单播数据帧
 - 网桥的目标是透明的扩展局域网，必须支持广播和多播
- 广播的解决方案
 - 每个网桥将带有目标广播地址的数据帧传到除了接收它的端口以外的其他活动(选择)的端口
- 多播的解决方案
 - 扩展生成树算法用来裁减掉那些不需要转发多播帧的网络



网桥的局限性

- 网络的局限性: **Scalability(扩展性)**
 - 当连接网段的数量增加时, 生成树算法扩展性存在局限性
 - 在一个大规模的网络中, 广播帧会影响网络的性能
- 解决方案: **虚拟局域网 (VLAN)**
 - 每一个虚拟局域网分配一个标识符
 - 只有两个网段的标识符相同时, 才能完成数据帧的转发

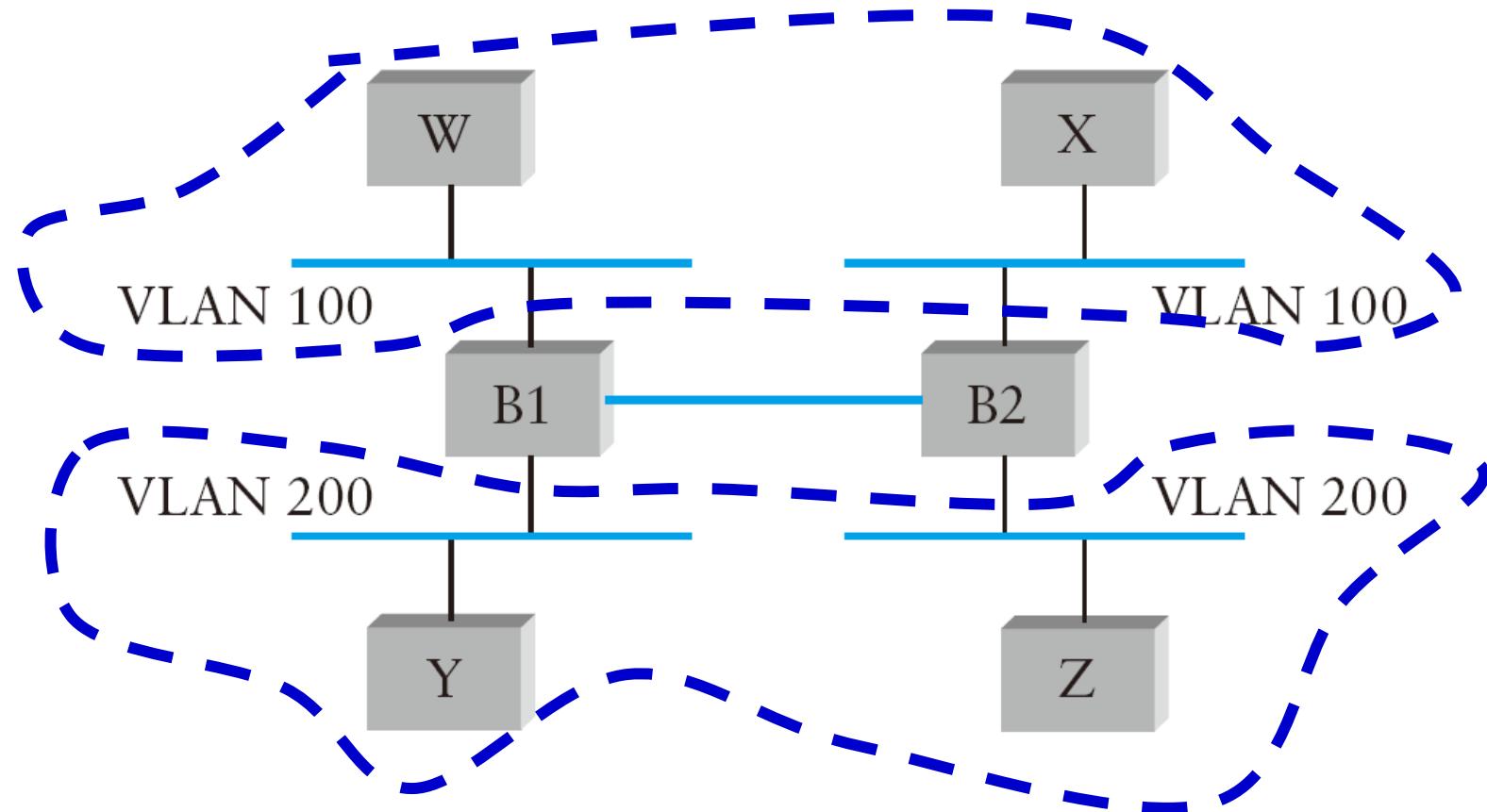
VLANs: motivation



consider:

- CS user moves office to EE, but wants connect to CS switch?
- single broadcast domain:
 - all layer-2 broadcast traffic (ARP, DHCP, unknown location of destination MAC address) must cross entire LAN
 - security/privacy, efficiency issues

虚拟局域网示例



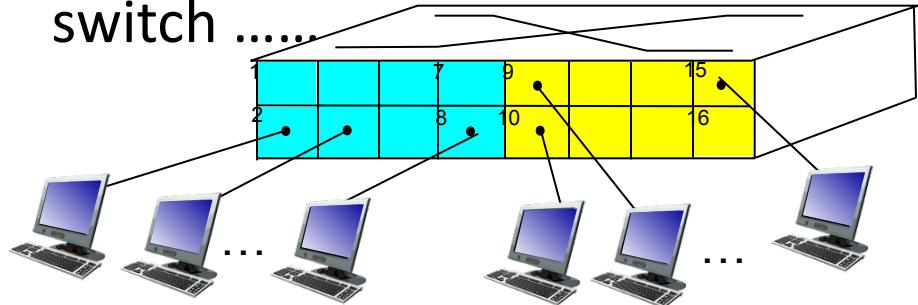
VLANs



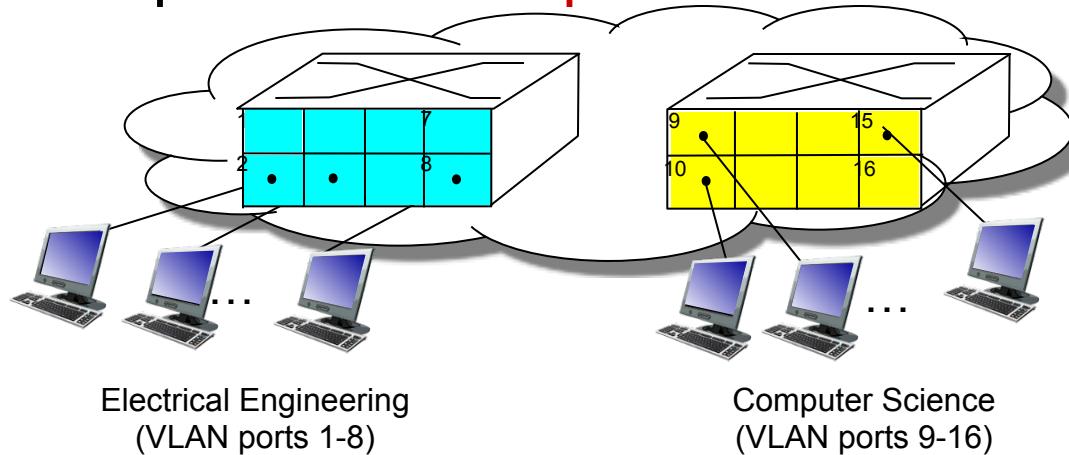
Virtual Local Area Network

switch(es) supporting VLAN capabilities can be configured to define multiple **virtual LANS** over single physical LAN infrastructure.

port-based VLAN: switch ports grouped (by switch management software) so that *single* physical switch

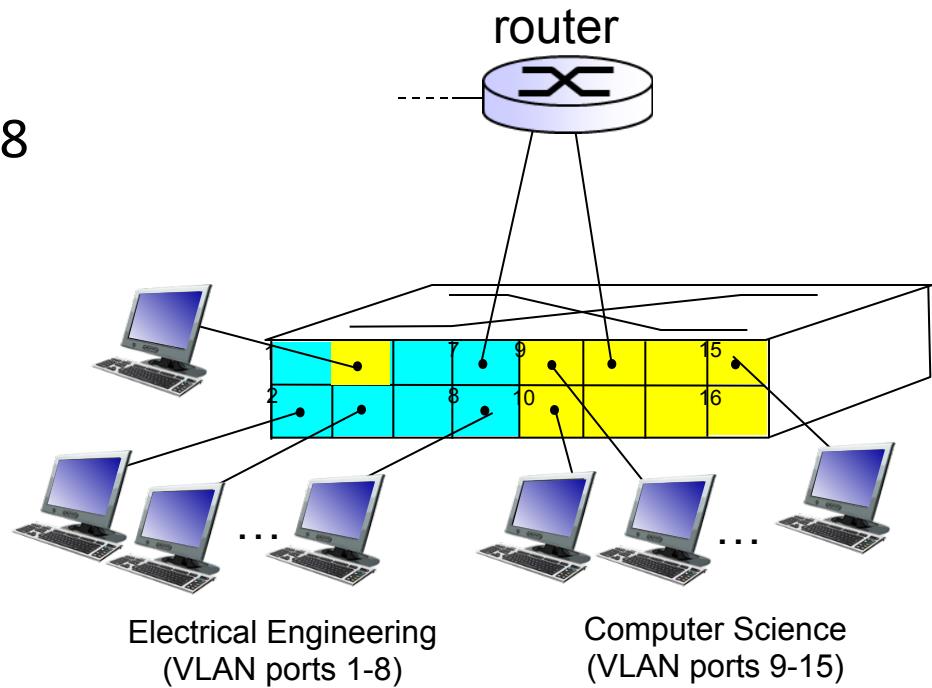


... operates as **multiple virtual switches**

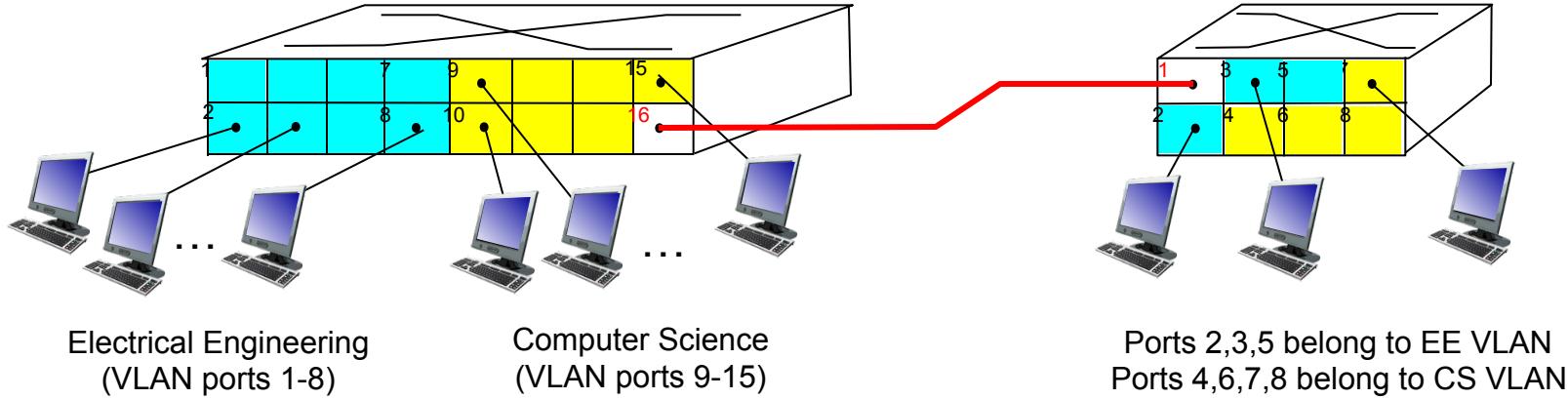


Port-based VLAN

- ❖ ***traffic isolation:*** frames to/from ports 1-8 can *only* reach ports 1-8
 - can also define VLAN based on MAC addresses of endpoints, rather than switch port
- ❖ ***dynamic membership:*** ports can be dynamically assigned among VLANs
- ❖ ***forwarding between VLANs:*** done via routing (just as with separate switches)
 - in practice vendors sell combined switches plus routers

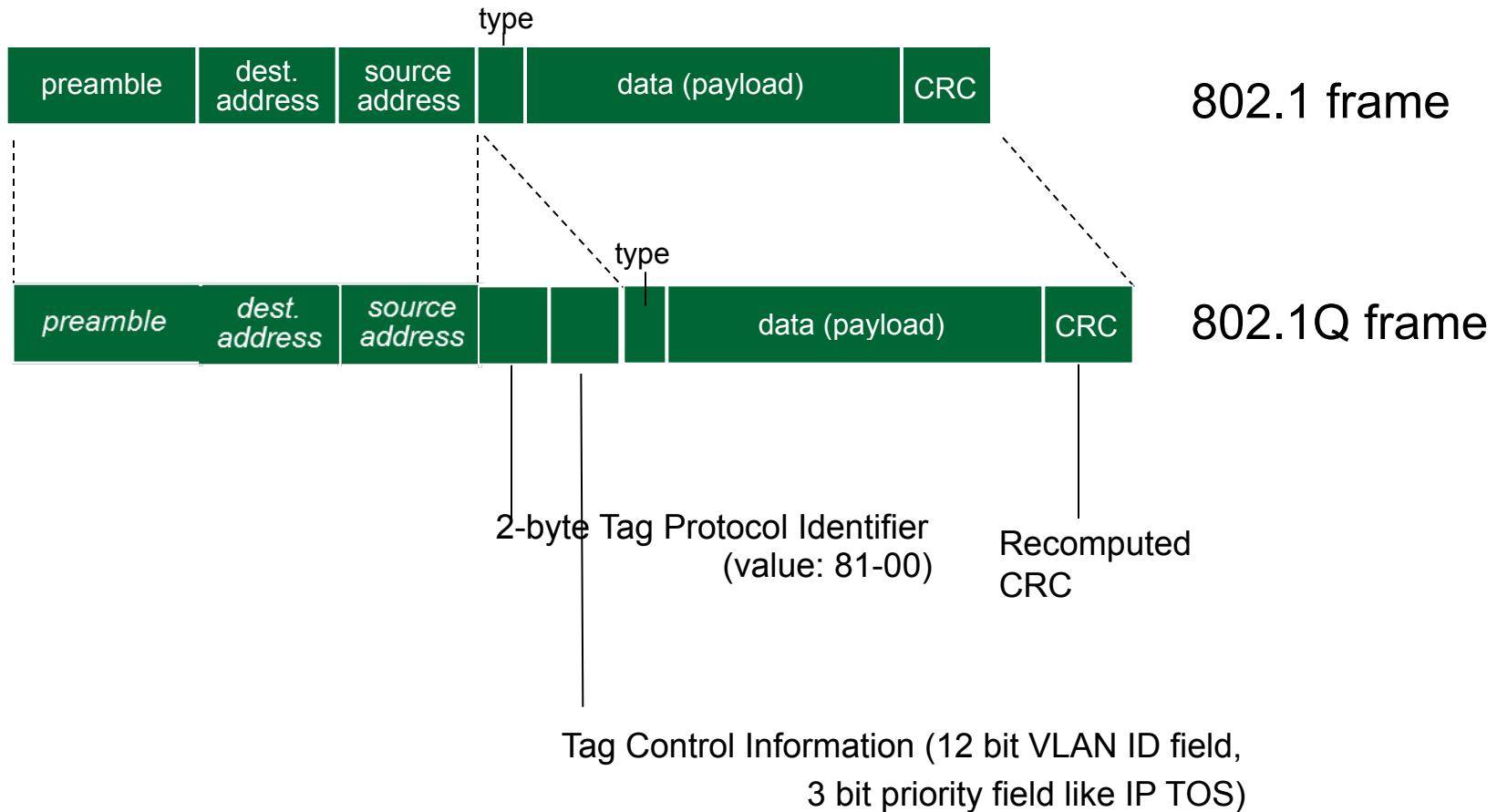


VLANs spanning multiple switches



- **trunk port:** carries frames between VLANs defined over multiple physical switches
 - frames forwarded within VLAN between switches can't be vanilla 802.1 frames (must carry VLAN ID info)
 - 802.1q protocol adds/removed additional header fields for frames forwarded between trunk ports

802.1Q VLAN frame format





网桥的局限性

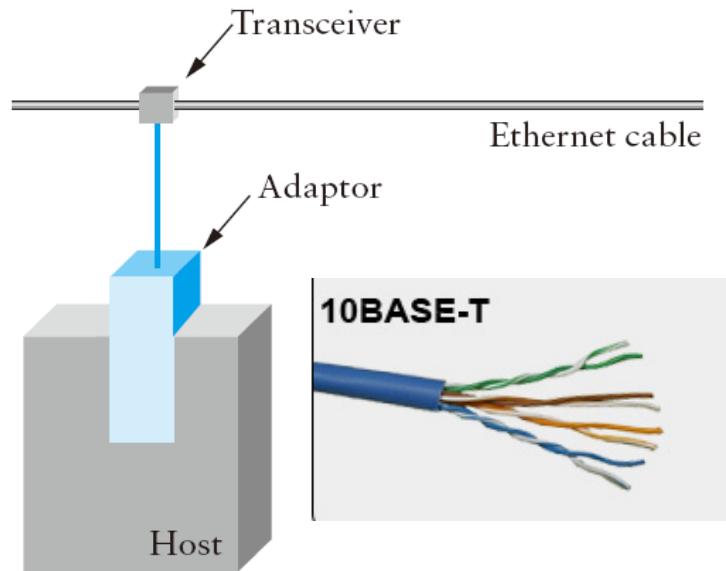
- 网络的局限性: **Heterogeneity(异构型)**
 - 网桥利用帧头部进行数据帧转发, 因此仅支持采用相同格式地址的网络, 例如 48bits
- 解决方案: ?



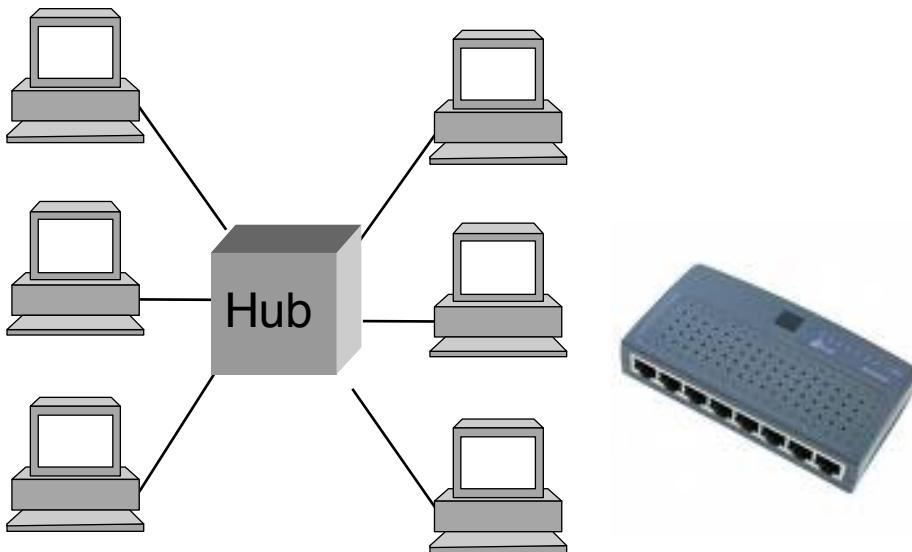
Ethernet 桥接和交换

序号	问题	解决方案
1.	Ethernet 扩展	转发表
2.	Ethernet 自适应技术	学习算法
3.	交换机回路	生成树算法
4.	扩展性和安全性	虚拟局域网

历史回顾: 以太网集线器Hub



1990, 10 BASE-T, 10Mbps,

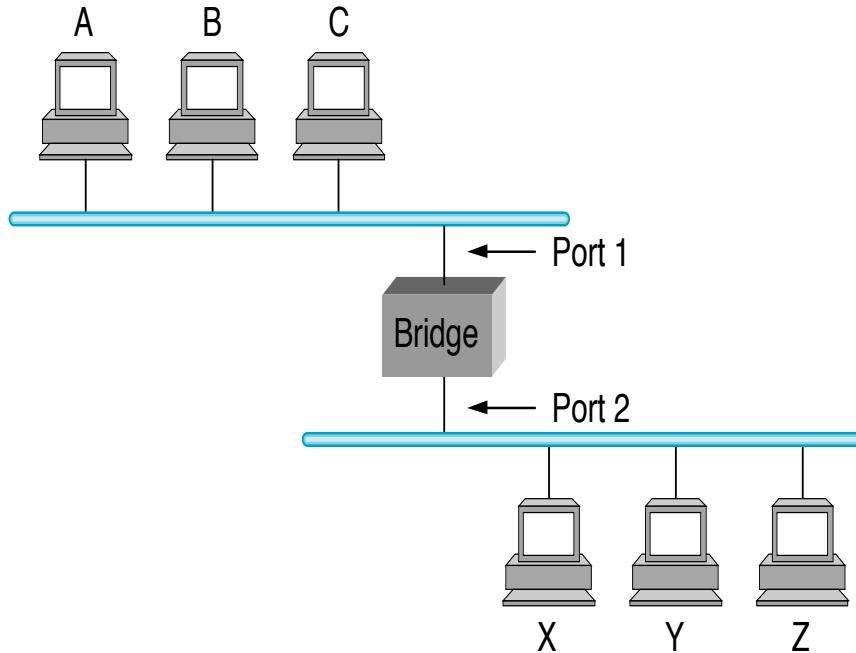


Hub(集线器): 物理层设备

- (1) 多个端口重复发送
- (2) 将一个端口收到的比特序列，以相同的速率向所有其他端口发送
- (3) 连接到同一个集线器的主机同时发送数据，会相互冲突



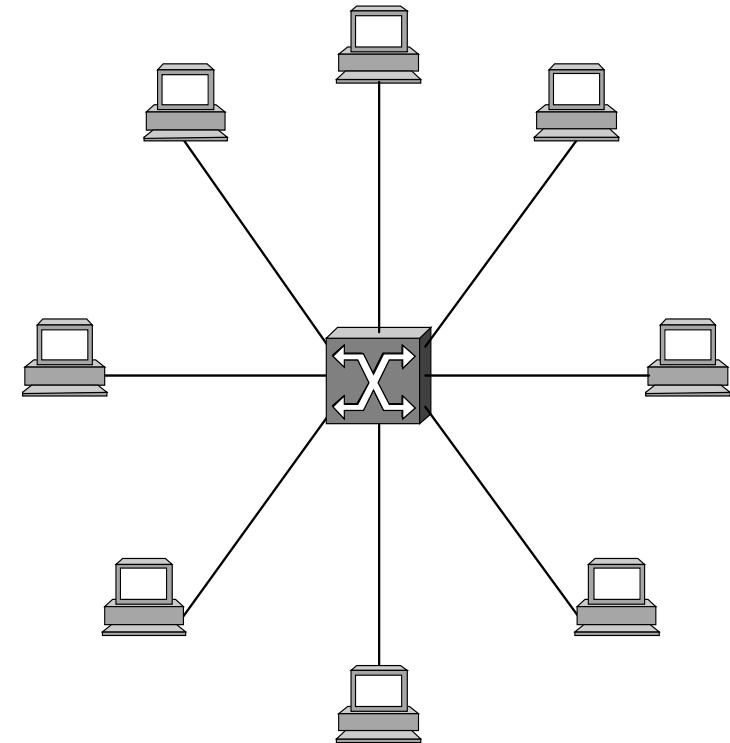
历史回顾: 以太网桥和交换机



1984, DEC推出第一个网桥

1990, IEEE 802.1D, Spanning Tree Protocol (STP)

Bridge(网桥): 数据链路层



1989, Kalpana推出第一个以太网交换机(公司后被思科收购)

Switch(交换机): 数据链路层
多端口网桥



从以太网集线器、网桥到交换机

- 比较
 - 集线器: 在共享媒介局域网的核心设备, CSMA/CD
 - 网桥: 早期版本的交换机, 用于局域网扩展, 在不同网段之间转发数据帧
 - 交换机: 每个端口是个网桥, 端口之间转发数据帧, 不同端口之间可以传输隔离
- 什么变化了?
 - 共享媒介 → 专有媒介, 不需使用 CSMA/CD
 - 共享带宽 → 独占带宽
 - 半双工 → 全双工

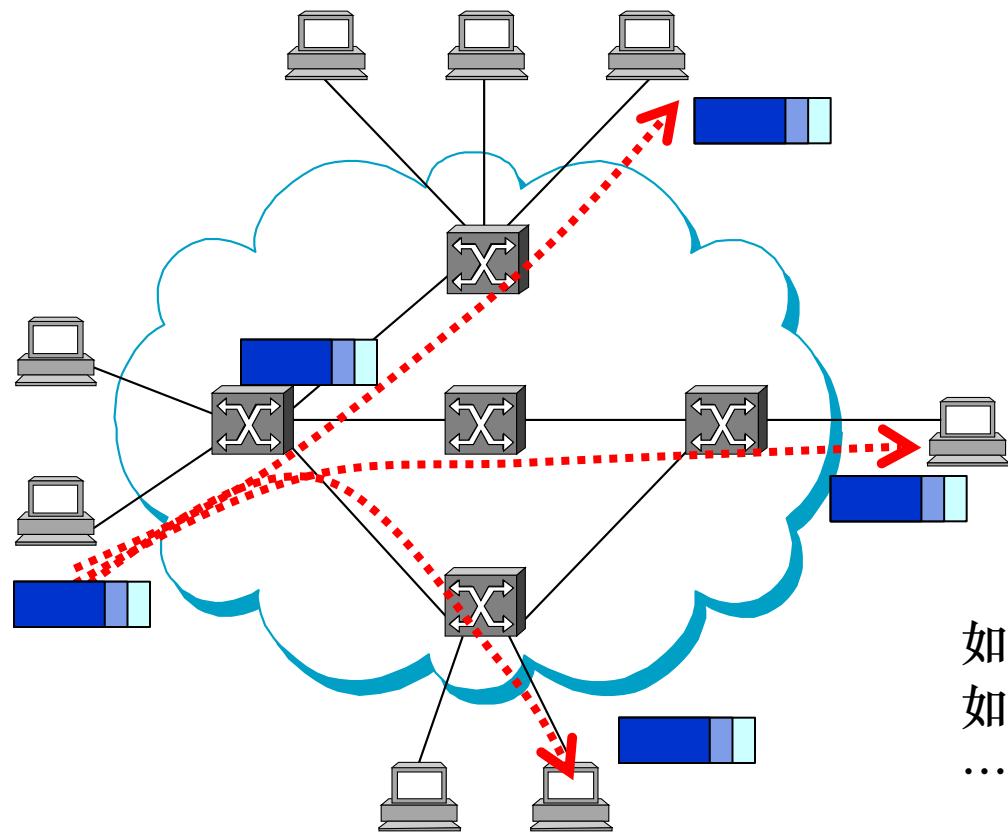


重复器, 集线器, 交换机, 网桥

	<u>Repeater</u>	<u>Hub</u>	<u>Switch</u>	<u>Bridge</u>
层次	PHY	PHY	MAC	MAC
功能	扩展以太网覆盖范围	配置星形拓扑连接更多节点	高速连接更多主机	多个异构局域网互联
机制	信号放大和整形	信号放大和整形	端口间快速转发数据帧	基于过滤器转发数据帧
端口	两个	多端口	多端口	多端口
地址	/	/	MAC地址	MAC地址
冲突域	所有节点共享媒介	所有节点共享媒介	没有冲突	每个局域网是不同的冲突域

分组交换网络中的问题

- 如何确定分组的输出端口?
 - 无连接方式, 例如 LAN 交换
 - 面向连接方式, 例如 ATM 交换



如何建立连接?
如何保持连接?

...



分组交换方法比较

	<u>数据报</u>	<u>虚电路</u>	<u>源路由</u>
起源	ARPANET, 1969	X.25, CCITT, 1976	Token Ring, 802.5, 1980s
核心思想	主机提供可靠保证	网络提供可靠保证	分组提供可靠保证
分组信息	分组首部包含源/目的主机地址	分组首部包含一个本地有效的临时VCI	分组首部包含完整的路由信息
转发机制	独立处理每一个分组	按照VC模式处理每一个分组	独立处理每一个分组
目的节点接收的分组	乱序	按序	按序



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结

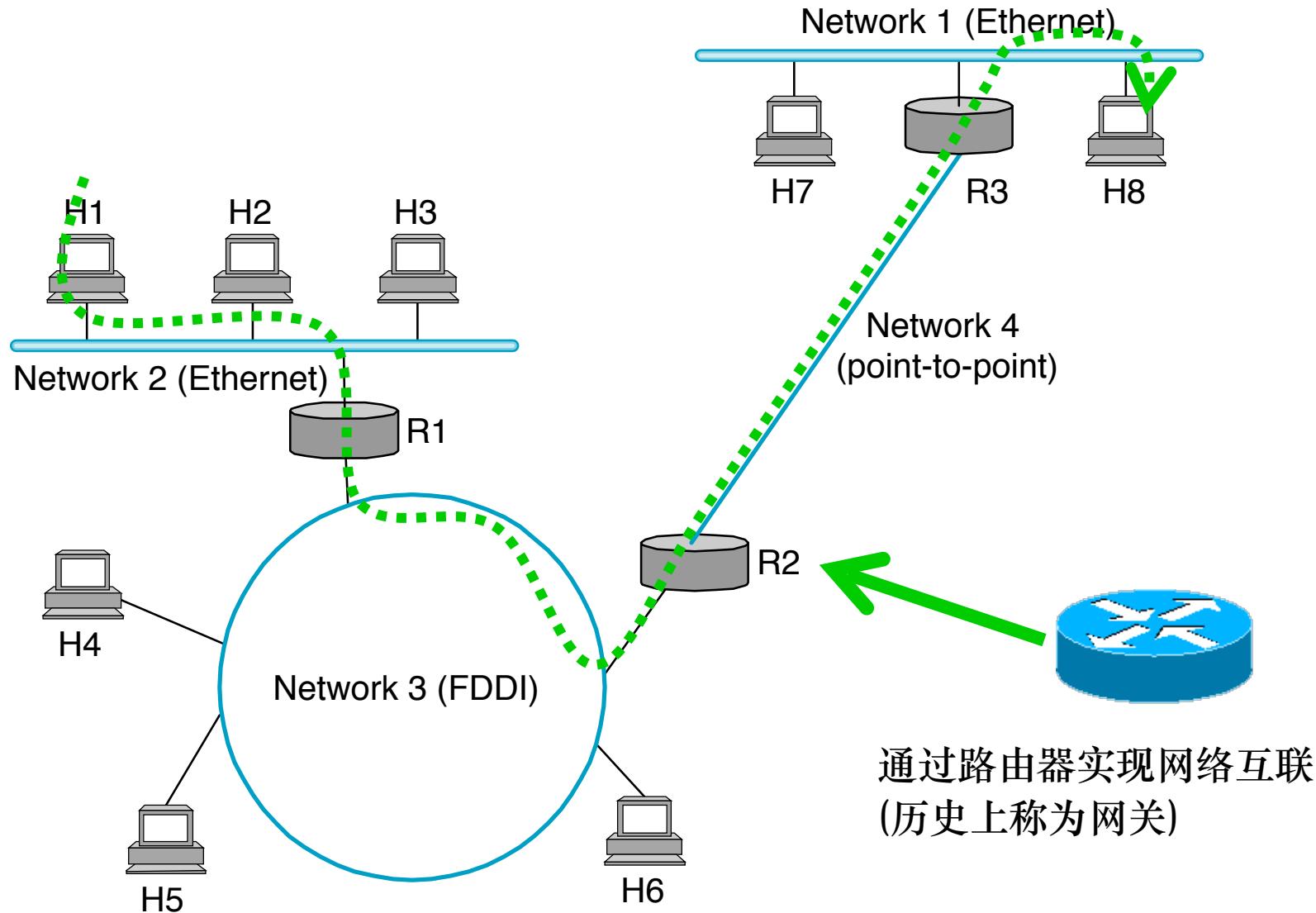




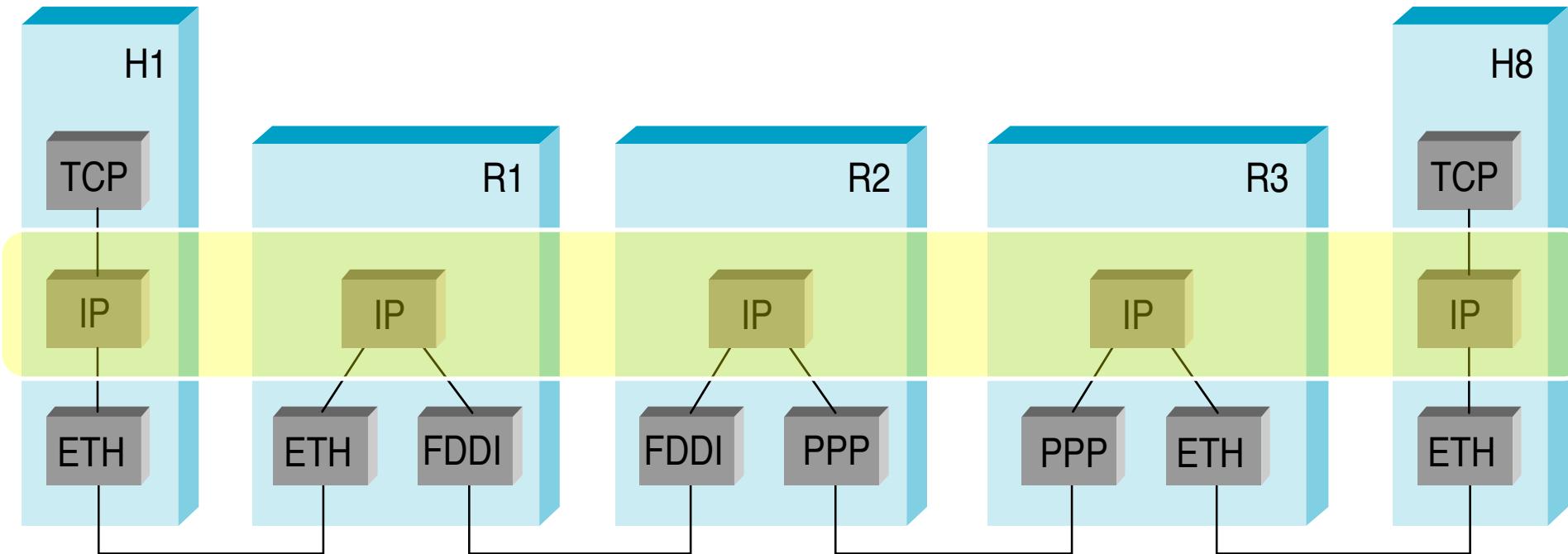
什么是互联网?

- “单一”网络
 - 采用同一种网络技术,例如点到点链路,共享介质网络以及链路层交换机组建的网络
- 互联网
 - “单一”网络的互联集合
 - 视为“网络的网络”:由许多较小的网络构成
 - 范例: Internet, 当今应用最为广泛的全球性互联网
- 另一种观点
 - 互联的底层网络视为物理网络.
 - 互联网是由物理网络集合构成的逻辑网络.

互联网: 示例

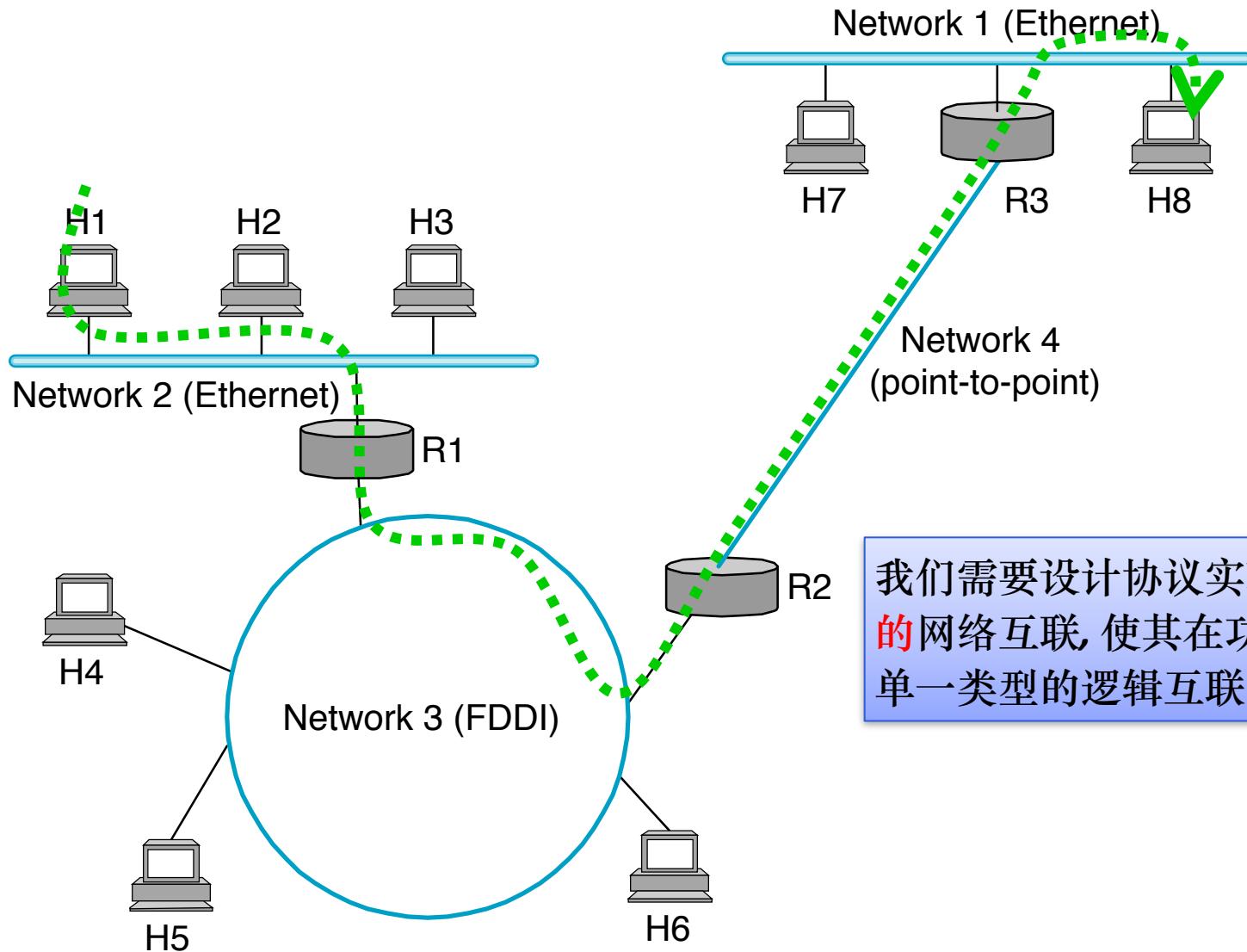


网络互联: 示例



Adding an Internetwork Layer (IP) for Interoperability

网络互联: 示例



我们需要设计协议实现**可扩展的, 异构的**网络互联, 使其在功能上等同于一个单一类型的逻辑互联网.



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结





服务模型

- 什么是服务模型?
 - 主机到主机服务
 - 可获得服务依赖于物理网络的能力
- 服务模型示例
 - 数据报的一种服务模型:
 - 有保证的传送
 - 保证传送时延不超过40毫秒
 - 数据流的一种服务模型:
 - 有序的数据报传送
 - 保证数据流的最小带宽
 - 分组间间隔变化受限



历史背景

ARPANET发展情况

- 1968-1972: 验证分组交换网络
 - 1969 第一条ARPANET消息从UCLA (Kleinrock)传输到SRI (Engelbart)
 - 1970: 第一个ARPAnet终端主机之间的通信协议，网络控制协议 [Network Control Protocol \(NCP\)](#) [RFC 001].
- 1972-1980: 解决网络互连的问题
 - 1972: ARPAnet公开演示 (Robert Kahn)
 - 1973: [Transmission Control Program \(TCP\)](#) (Robert Kahn and Vinton Cerf)

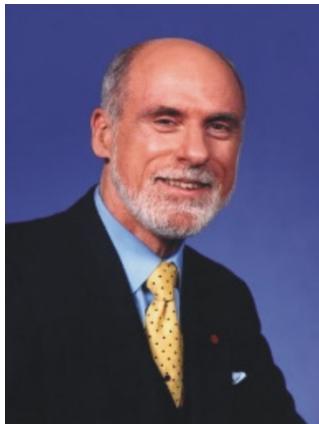


有效设计和实现了分组交换网络!



早期TCP

- 1973, ARPAnet
 - 只有一个核心协议: TCP, *Transmission Control Program* 早期传输控制“程序”
 - 第一版TCP于1973年实现, 后于1974年12月修改并正式归档
 - RFC 675, *Specification of Internet Transmission Control Program*



1973年TCP协议

A Protocol for Packet Network Intercommunication

VINTON G. CERF AND ROBERT E. KAHN,

Cerf, V. and Kahn, R.E., A Protocol for Packet Network Intercommunication, IEEE Transactions on Communications, vol. 22, no. 5, pp. 637 - 648, 1974.



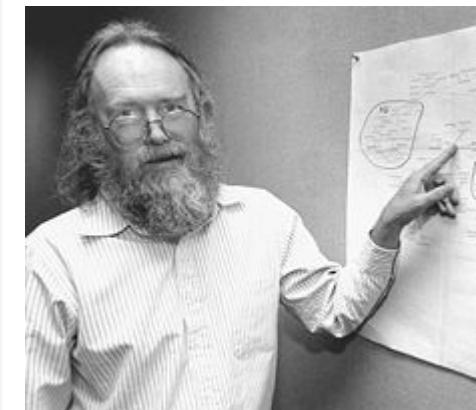
早期TCP存在的问题

- 1973年 – 1977年
 - TCP测试和开发持续了好几年
- 1977年3月, TCP第二版定稿归档.
- 1977年8月, Internet Engineering Note number 2 (IEN 2)

We are screwing up in our design of internet protocols by violating the principle of layering. Specifically we are trying to use TCP to do two things: serve as a host level end to end protocol, and to serve as an internet packaging and routing protocol. These two things should be provided in a layered and modular way. I suggest that a new distinct internetwork protocol is needed, and that TCP be used strictly as a host level end to end protocol

TCP：处理主机级别的端到端问题

IP：处理互联网分组与路由的问题



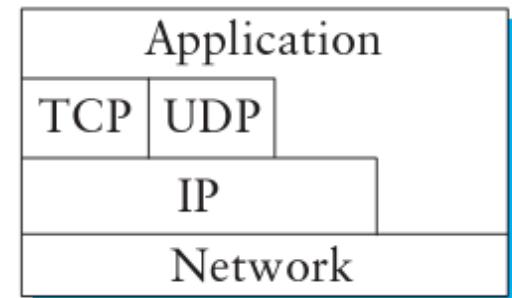
Jon Postel, 互联网和**TCP/IP**协议的重要先驱者之一

他是互联网协议(**Request for Comment (RFC)**)文档编辑，并主管互联网号码分配局，直到他去世。



发明TCP/IP体系架构

- TCP/IP的发展过程
 - 单一的TCP协议 → 分割为传输层的TCP协议和网络层的IP，形成TCP/IP协议族
 - 1978年，从TCP第3版开始，TCP被分为两个部分
 - 1980年，正式提出现代网络使用的第一个独立的IP协议和TCP协议(第4版)
- 1980年代
 - 1983年，TCP/IP协议族正式被采纳，代替NCP协议成为ARPAnet的标准主机协议
 - 越来越多的主机和网络使用TCP/IP协议连入到发展中的ARPAnet；基于TCP/IP的互联网由此诞生。





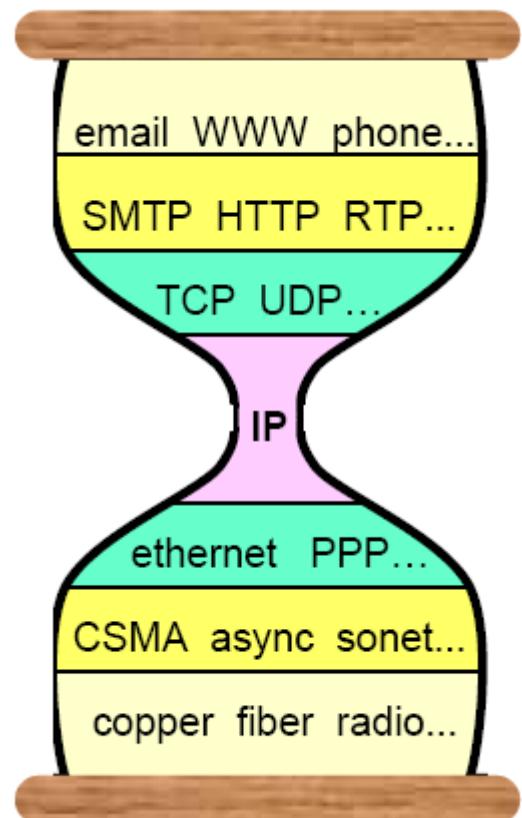
互联网架构设计原则

- R. Kahn总结了开放式网络架构
 - ***minimalism, autonomy*** 最小化的设计: a network should be able to operate on its own, with no internal changes required for it to be internetworked with other networks;
 - ***best effort service*** 尽力交付的服务: internetworked networks would provide best effort, end-to-end service. If reliable communication was required, this could be accomplished by retransmitting lost messages from the sending host;
 - ***stateless routers*** 无状态的路由器: the routers in the internetworked networks would not maintain any per-flow state about any ongoing connection
 - ***decentralized control*** 分布式的控制: there would be no global control over the internetworked networks.



IP服务模型

- Internet 协议(IP)
 - 连接不同类型网络的网络层协议
 - 最成功的网络互联协议
- 设计理念: 提供最基本的服务(最小服务集)
 - 可以支持任意类型的网络技术(现行的, 未来兴起的)





IP服务模型

- IP服务模型
 - 数据报传送
 - 尽最大努力交付, 在任何技术上运行
 - 分组格式, 分片与重组
 - 层次化寻址(参阅4.1.3节)
 - 数据报转发(参阅4.1.4节)

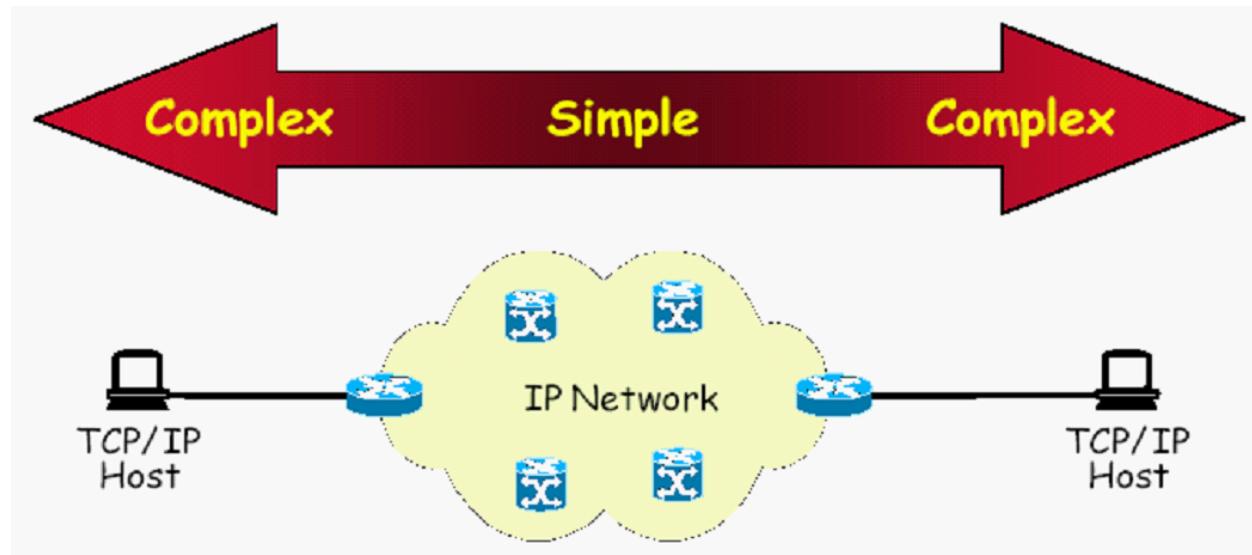


IP 服务模型: 数据报传送

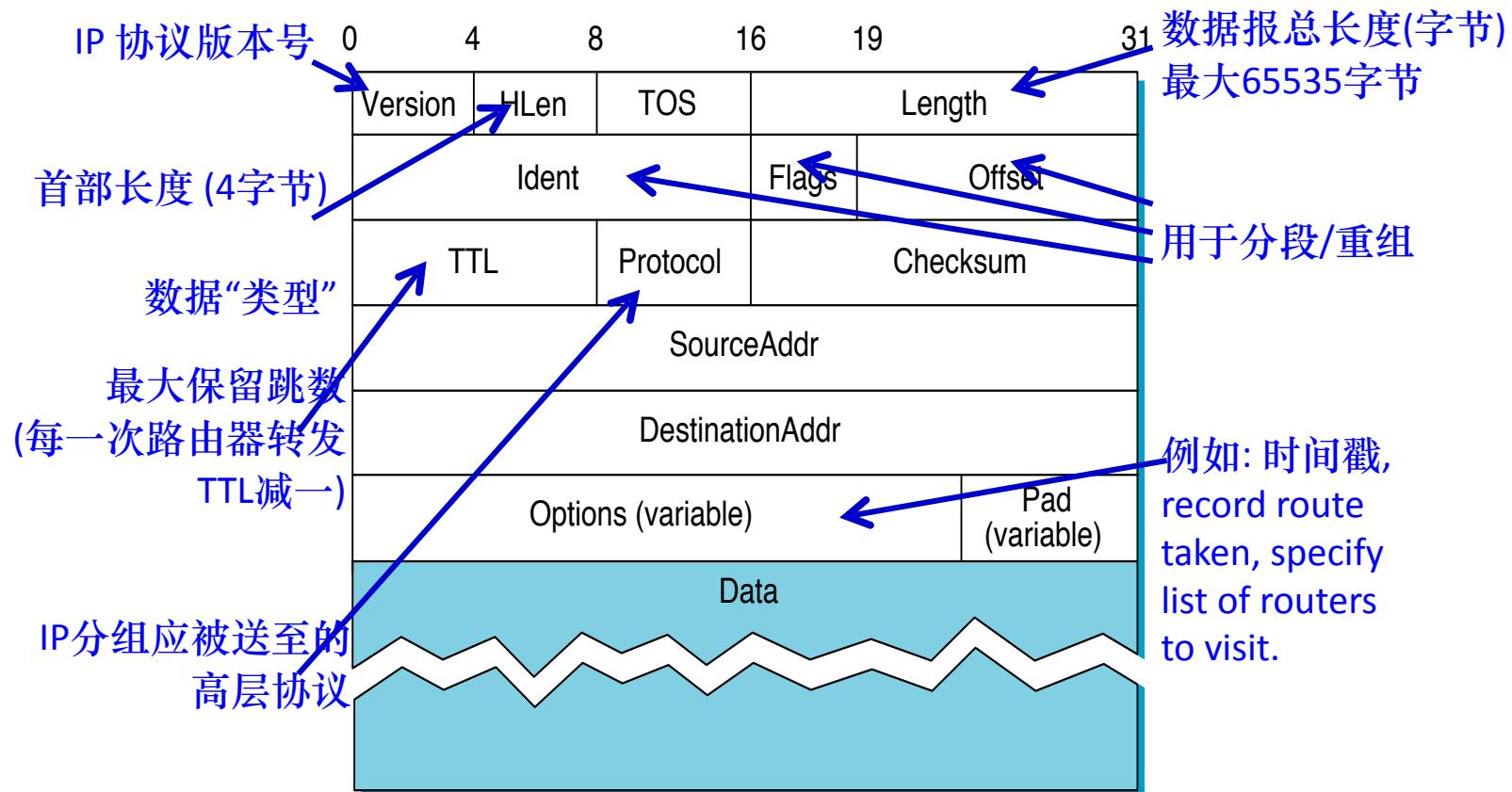
- 数据报传送
 - 无连接方式
 - 每一个分组携带转发所需的信息
- 尽最大努力交付: **不可靠服务**
 - 尽最大努力进行数据报传送, 但不对传送过程进行任何保证
 - “不保证”不仅仅意味着分组丢失
 - 分组集合可能乱序到达
 - 同一个分组可能多次传送
 - 运行于IP之上的高层协议或应用必须能够意识到这些可能的错误模式.

IP服务模型: 数据报传送

- 优点
 - IP可以“在任何技术上运行”
 - 需要强调的是现今IP赖以运行的很多网络技术在IP被发明时还不存在.
- 隐含的设计选择
 - 将复杂的事情(服务保证)交给主机完成
 - 扩展: 简单的核心和复杂的边界



IP服务模型: 分组格式





IP服务模型: 分组格式

- IP首部的不同字段
 - 自身的描述
 - 版本(4b), 首部长度(4b), 分组长度(16b), 校验和(16b)
 - 寻址
 - 源地址(32b), 目的地地址(32b)
- 底层的传送能力
 - 标识(16b), 标志(3b), 偏移量(5b)
- 向上层提供的服务
 - 服务类型(8b), 生存期(8b), 协议(8b)



IP 服务模型: 分段与重组

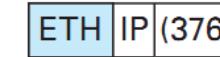
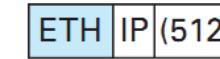
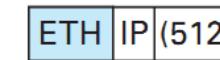
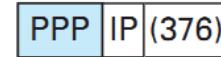
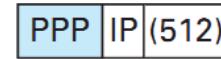
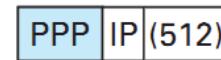
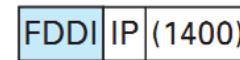
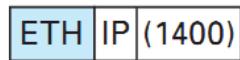
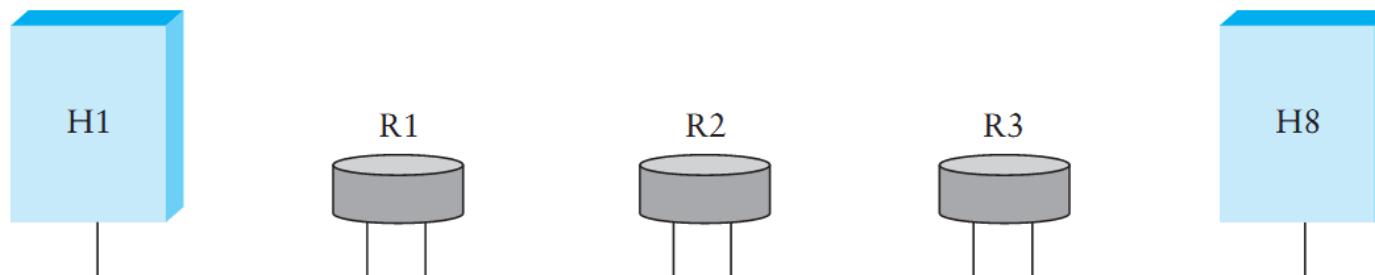
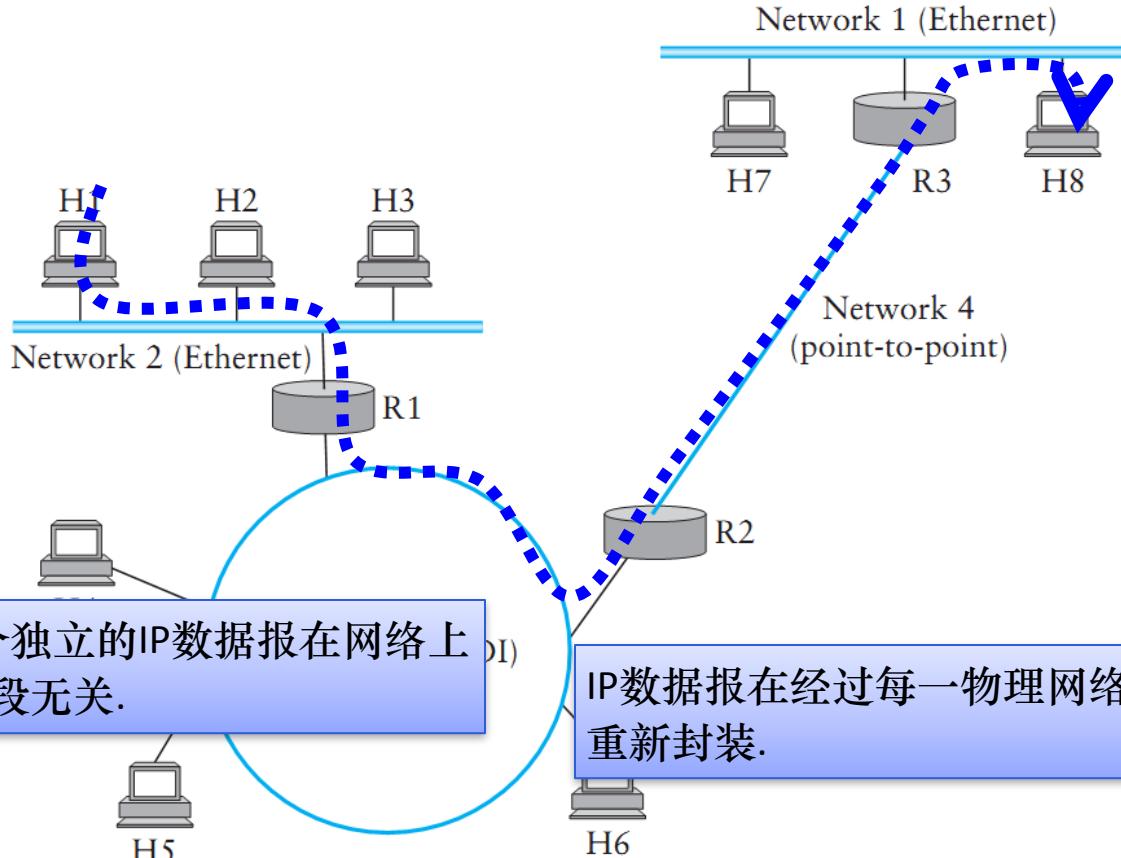
- 最大传输单元(MTU)
 - 网络能够承载的最大的分组/数据帧
 - 示例: 以太网 (1500 字节), PPP (532 字节)
- 原因
 - 不同的网络存在不同的MTUs
- 所有不同类型的网络为什么不采用的最小的MTUs?
 - 问题
 - 新的网络技术可能出现
 - 效率低: 分组首部越长则协议开销越大, 且会产生更多的待处理分组



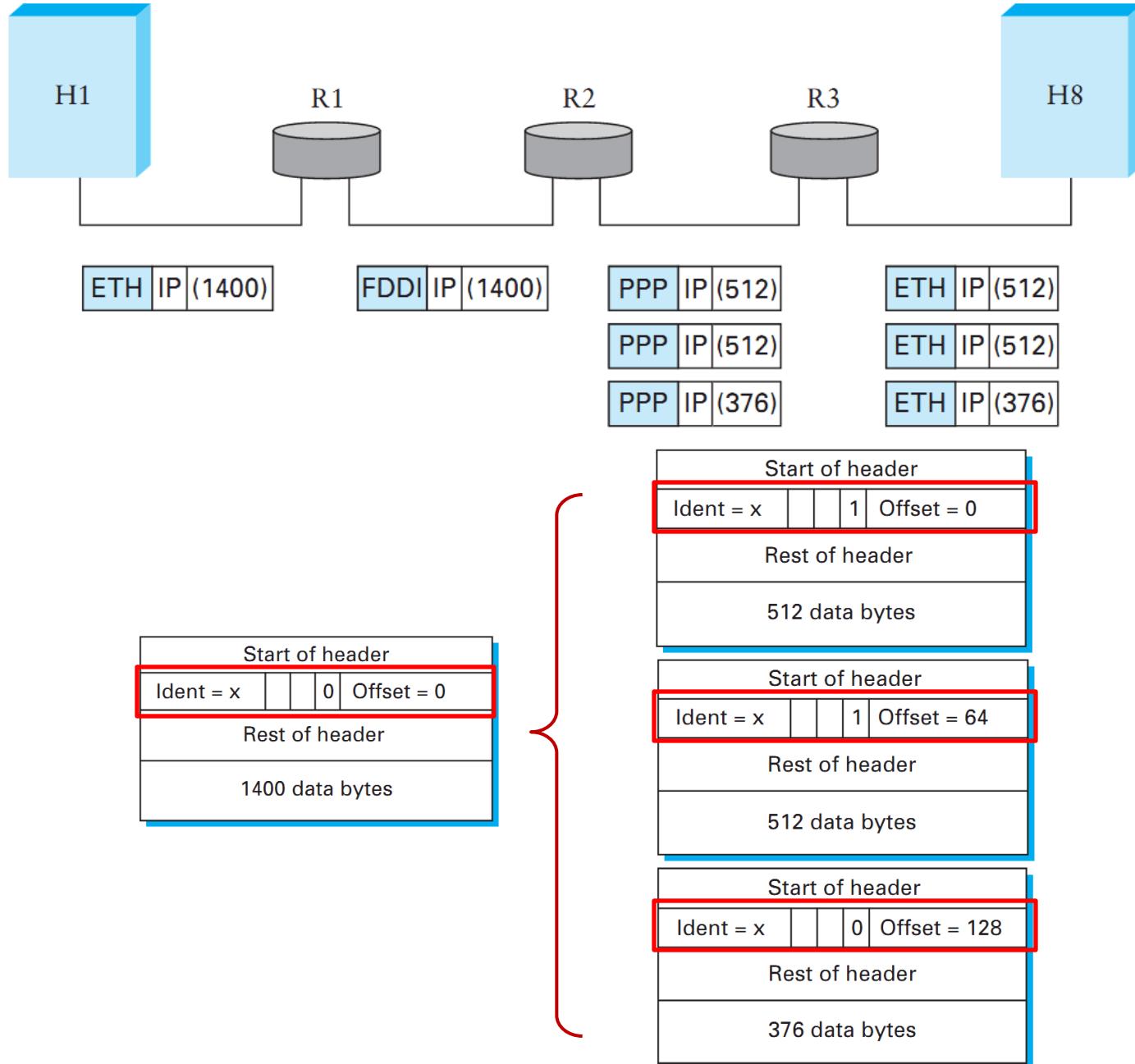
IP 服务模型: 分段与重组

- 策略
 - 主机按照与之直接相连的网络的MTU发送分组
 - 需要时进行分组分段($MTU <$ 分组长度)
 - 分段产生独立的数据报
- **分段重组由目的主机完成**
 - 分段丢失无法恢复
 - 迭代分段可能发生
- 问题
 - 如果某一个分段丢失, 目的主机将放弃重组进程并丢弃已收到的分段

示例:



示例:





提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结



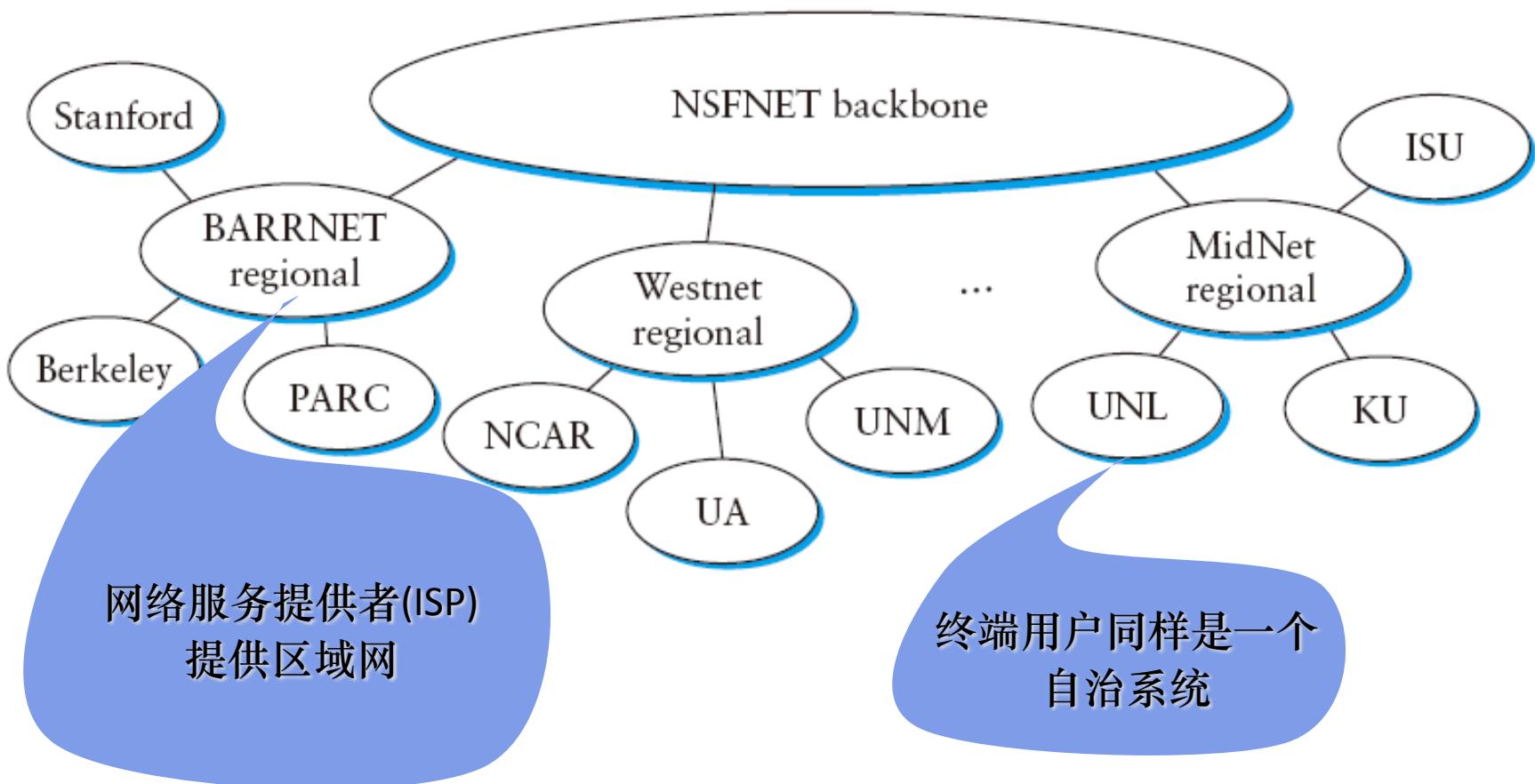


全球互联网

- 然而，简单的网络互联并不足以满足扩展性的要求
- IP地址的简单分层仅能实现“一定程度”的扩展
 - 路由器有必要知道连在互联网上的所有网络，这在现实中完全不可能达到
- 存在一些大幅度提高可扩展性的技术使得互联网发展到了当前的全球范围

全球Internet的实际情况

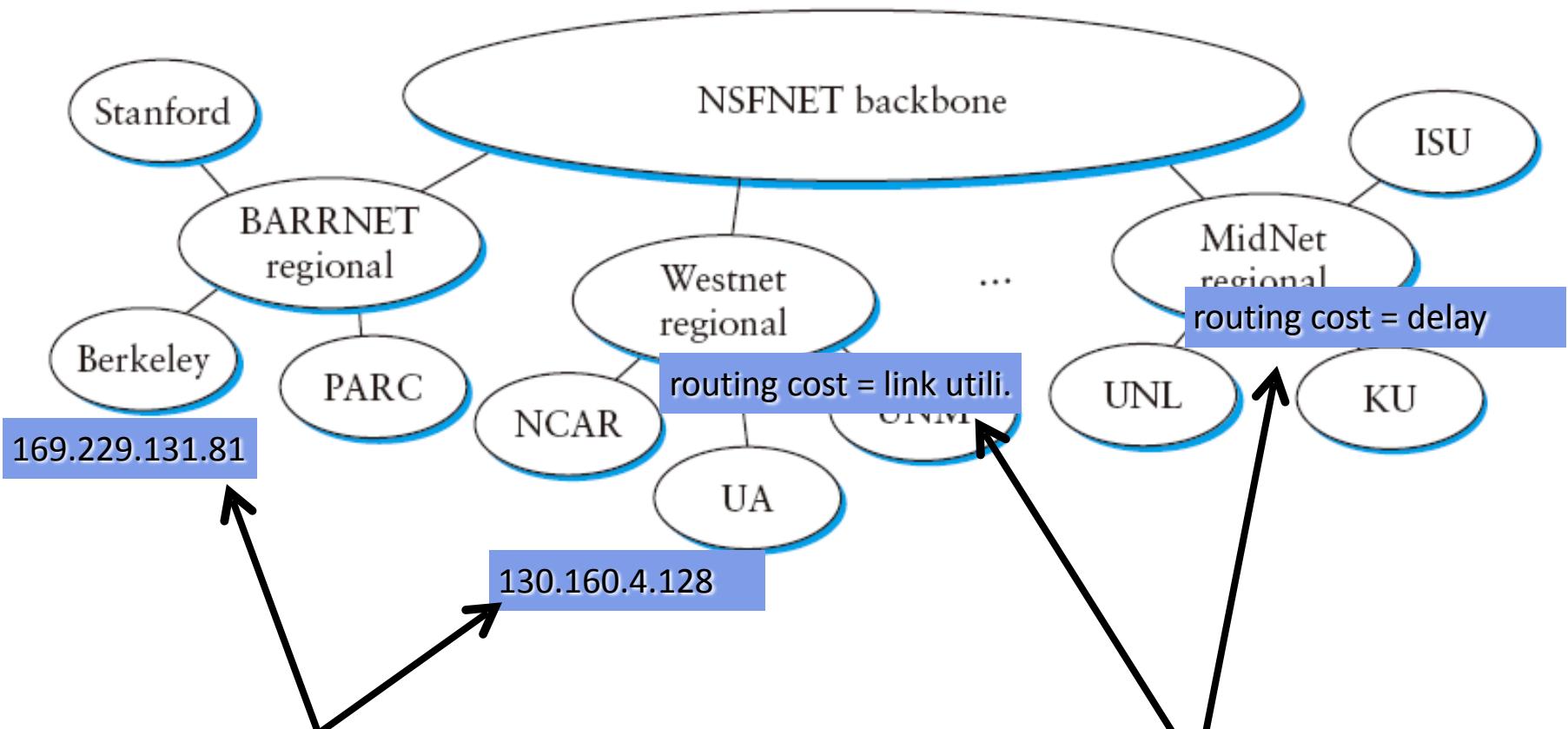
1990年时Internet的树形结构



全球Internet的实际情况



1990年时Internet的树形结构



大学可以为内部用户免费分配 IP 地址
=> 地址空间的利用率

ISPs 对网络中所使用的最佳路由协议以及如何定义链路评价指标都有不同的看法
=> 自治系统



编址方案

- Flat 扁平化编址方案
 - 每个主机以48位MAC地址标志
 - 路由器需要为每台主机准备转发表项目
 - 太大(虽然技术上可能实现)
 - 非常难以维护，主机上线离线
- Hierarchy 层次化编址方案
 - 地址分段分配
 - 先按照一般地区，再按照特定地址确定路由
 - 当网络动态变化，路由表内容可只更新增量

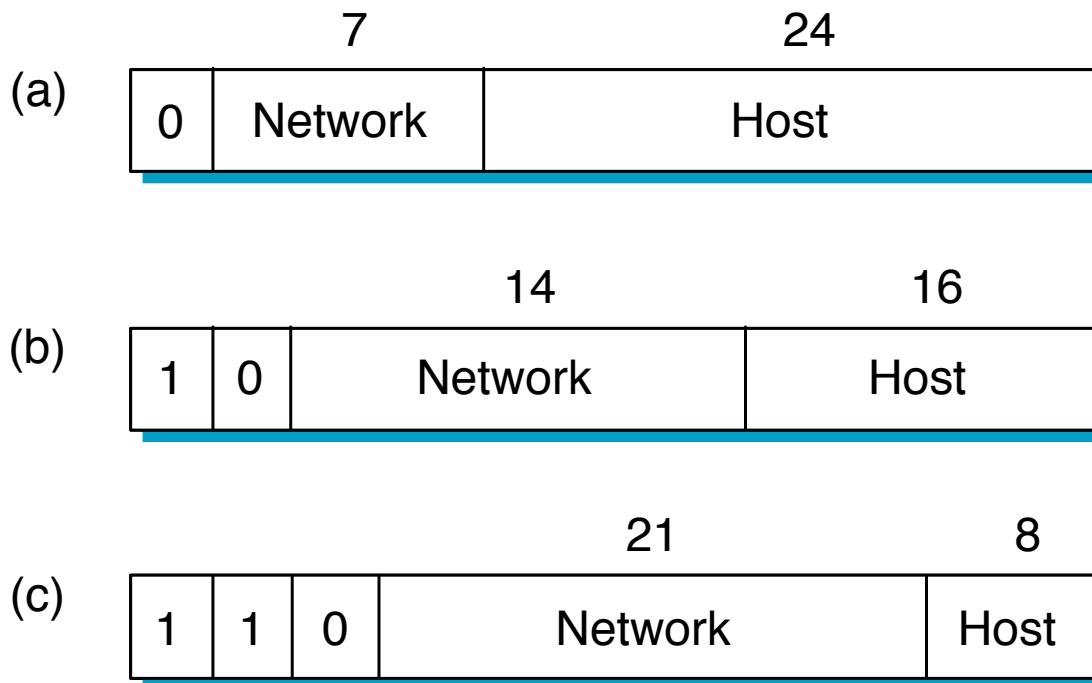


全局地址

- 网络互联的寻址问题
 - 首先,每一个主机拥有唯一的地址
 - 其次,在一个大规模网络中需要层次化IP地址
- IP的解决方案
 - 每一个地址全局唯一
 - 层次化寻址
 - 两级地址空间, {网络号:主机号}

层次化地址空间

- 每一个IP地址包括2个部分
 - 网络号和主机号
- IP地址分类





网络地址

- IP地址的网络号唯一标识一个网络
- 连接在同一个网络内的主机以及路由器接口的IP地址中包含相同的网络号
- 给定任意IP地址，路由器可以识别该IP地址所属的网络
- 如何将分组转送至目的网络？



分类寻址

- 分类寻址
 - 假设: Internet包含
 - 少量的广域网 (A类网络)
 - 相当数量的站点- (校园-) 规模的网络 (B类网络)
 - 大量的LANs (C类网络).
- 然而, 这种分类方式可能存在一些不公平的且未充分利用的地址分配
 - Internet中采用“无分类寻址”方式 (参阅4.3节)



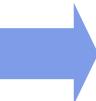
IP地址的表示

- 为了便于, IP地址采用”点分十进制”方式表示
 - IP 地址记为四个十进制整数, 用点分离.
 - 其中每一个整数表示为地址中1个字节对应的十进制数值.
 - 例如, 本机的IP地址为 171.69.210.245.
- 另一种节点命名方式: [Internet域名](#)
 - 域名标识为由“.”隔开的ASCII字符串, 例如`www.edu.cn`,
`www.hust.edu.cn`, `itec.hust.edu.cn` .
 - 应用层的DNS (域名解析服务) 完成域名和IP地址之间的映射转换.

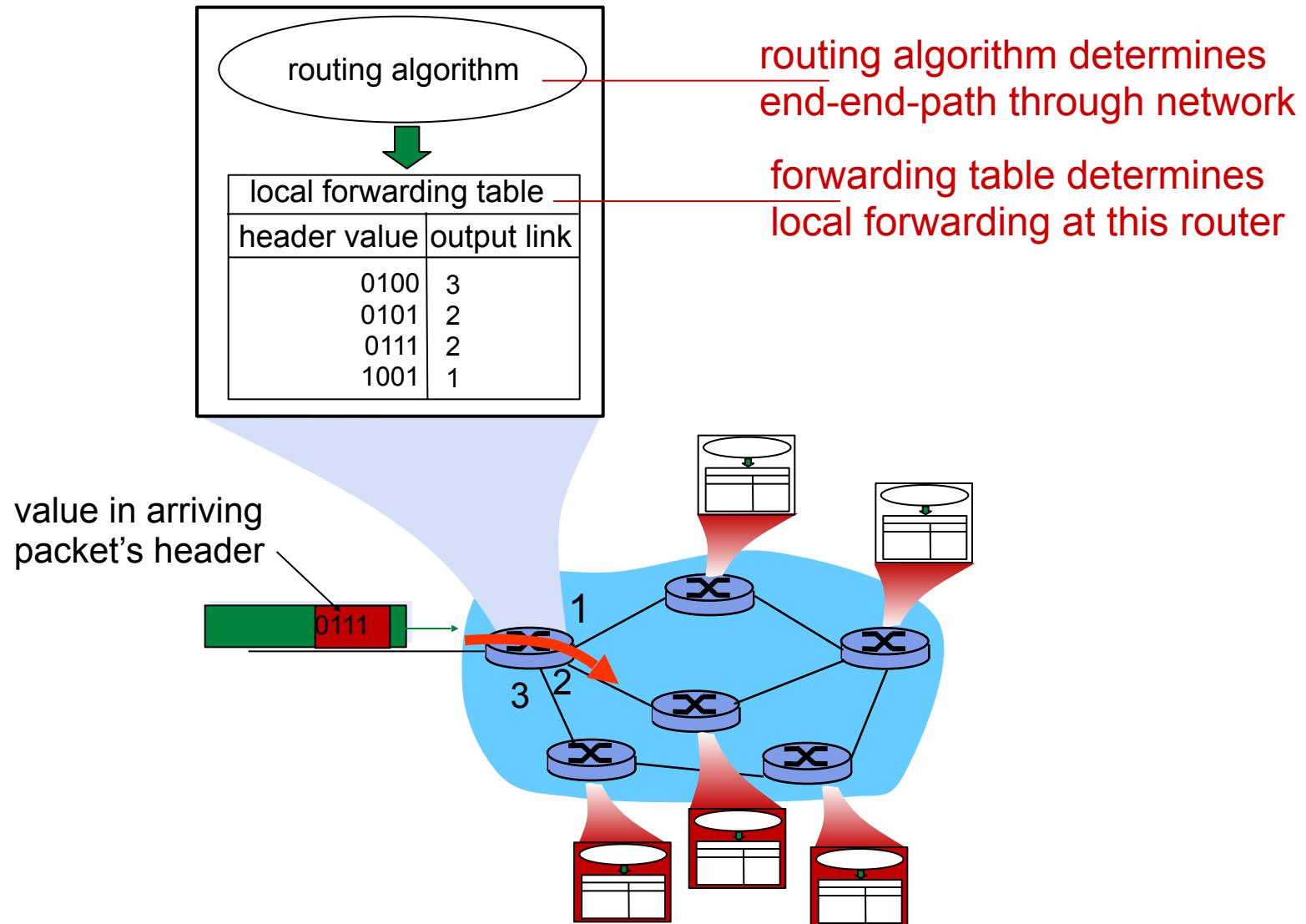


提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结



Interplay between routing and forwarding





IP中的数据报转发

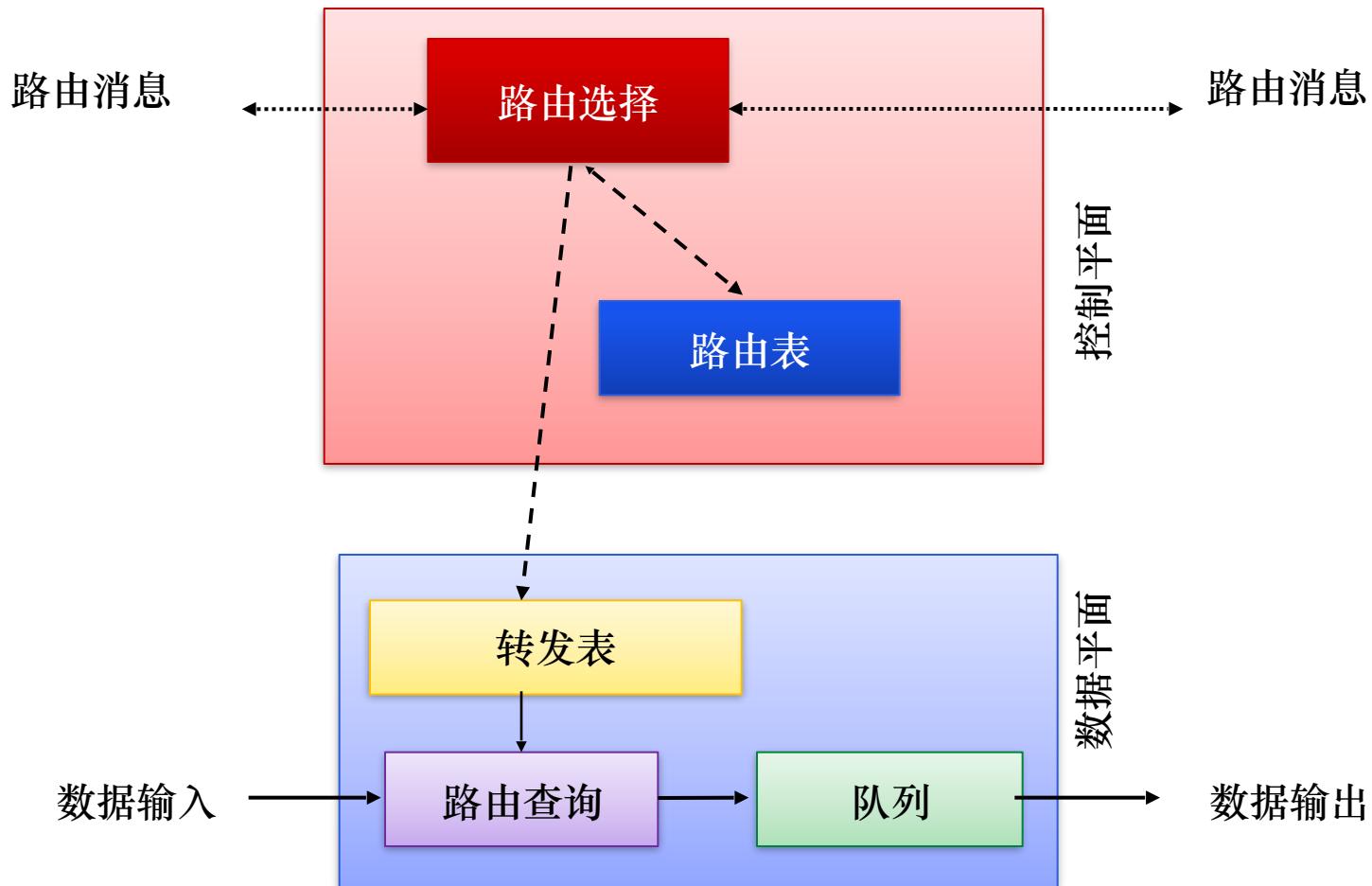
- 网络层的两大核心功能
 - 转发: 将路由器输入端口收到的分组从正确的输出端口发送出去
 - 路由选择: 决定分组从源节点到达目的节点的路径
 - 路由选择算法
- 可以类比到实际生活中:
 - 路由: 策划一次从家出发的旅行
 - 转发: 旅行途中每个中转站的换乘过程



转发vs.路由

- 转发(数据平面)
 - 过程: 获得一个分组, 查看其目的地址, 查询本地转发表, 将分组从输出端口发送出去
 - 节点(路由器/交换机)本地执行
- 路由选择(控制平面)
 - 转发表的构建过程
 - 通常由基于复杂分布式算法的路由协议完成

转发vs.路由





IP分组转发

- IP 分组转发
 - 路由器将某一输入端口收到的数据从一个或多个接口输出的过程
 - 基于转发表
- 转发表
 - $\langle \text{NetNum}, \text{NextHop} \rangle$
 - 如果不采用层次化寻址, 转发表形式为 $\langle \text{IPAddr}, \text{NextHop} \rangle$
- NextHop 信息通过路由算法获得



IP分组转发

- 路由器节点

if (目的节点的NetNum == 我的一个接口的NetNum)

 经过那个接口传送分组到目的节点

else

if (目的节点的NetNum在我的转发表中)

 传送分组至NextHop路由器

else

 传送分组至缺省路由器

- 主机节点

if (目的节点NetNum == 我的NetNum)

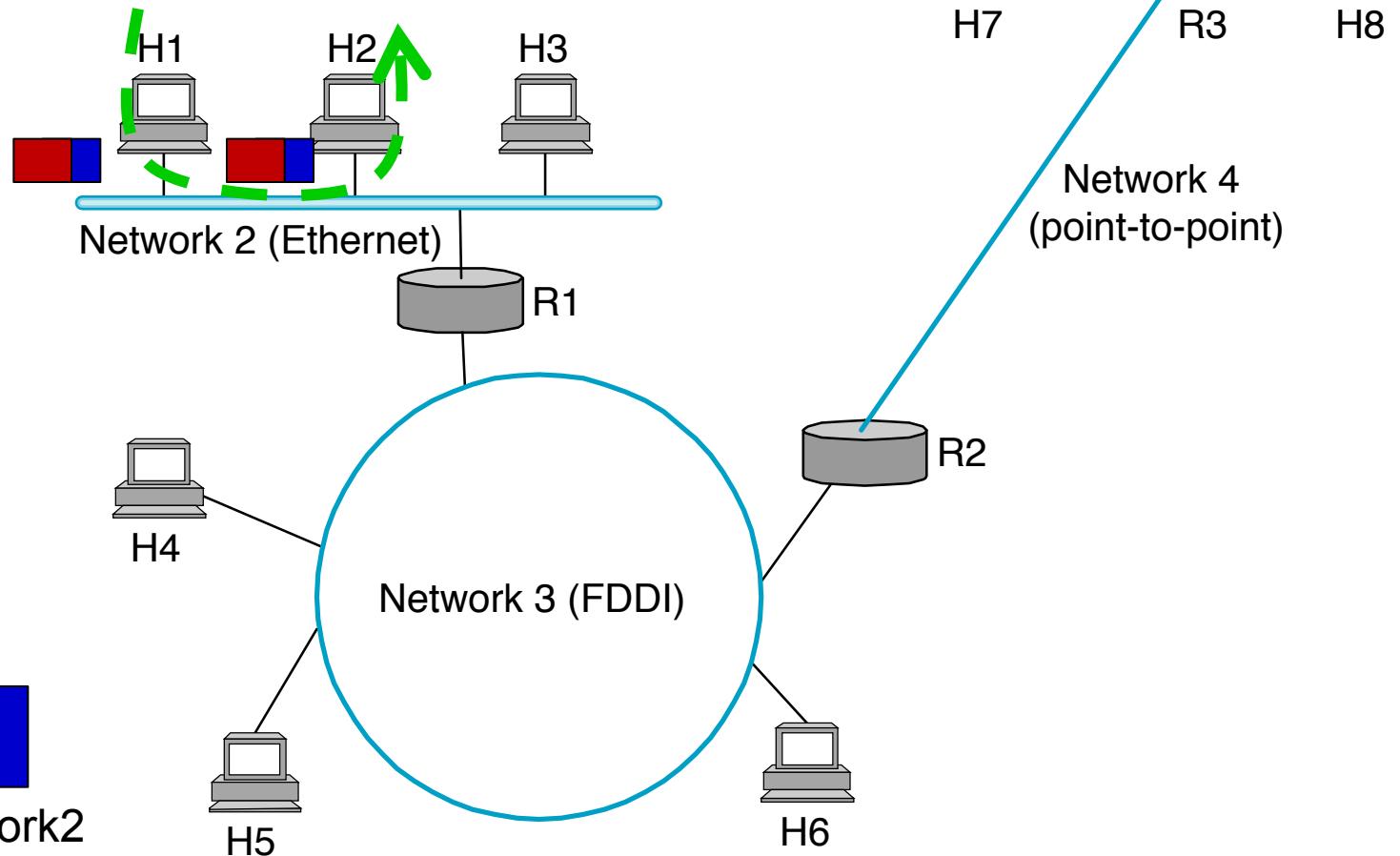
 直接传送分组至目的节点

else

 传送分组至缺省路由器

IP分组转发示例

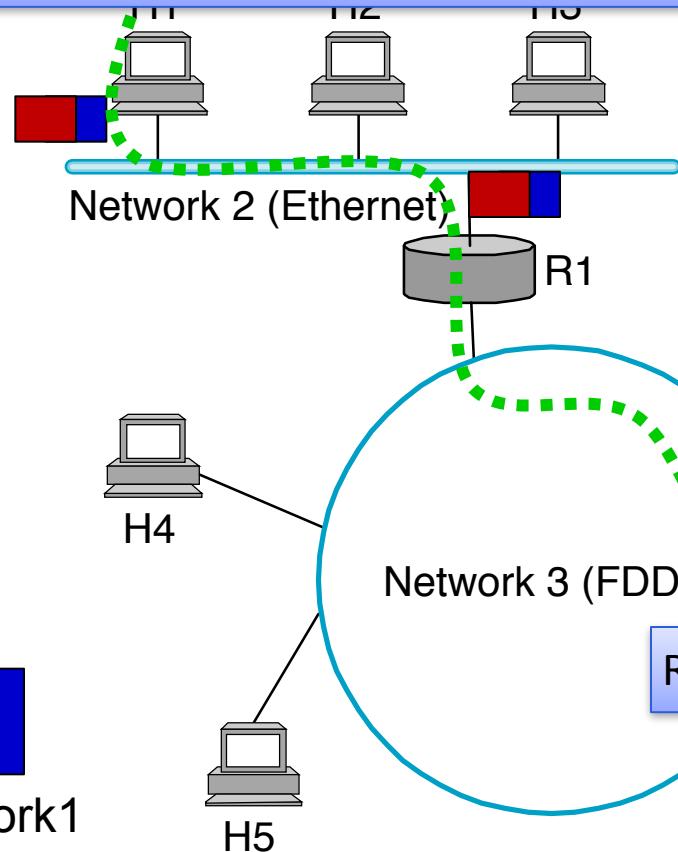
H1 和 H2 的 IP 地址中包含相同的网络号。因此，H1 可以知道数据报可以直接通过以太网传送至 H2。



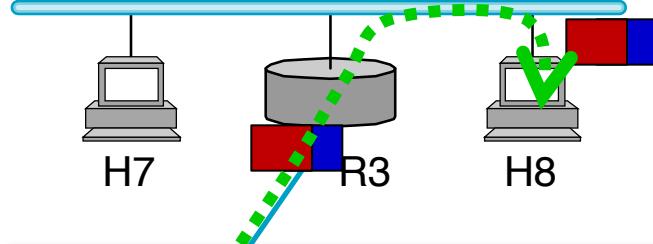
IP分组转发示例

H1 可以知道数据报需要通过路由器转发，则向 R1 传送

假设 R1 的默认路由为 R2; R1 则通过令牌环网将数据报发送至 R2.



Network 1 (Ethernet)



R3, 因为与 H8 在同一网络内, 直接发送数据报至 H8.

NetworkNum	NextHop
1	R3
2	R1

R2 查询转发表. 下一跳为 R3.



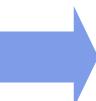
IP分组转发示深入讨论

- 路由器内的转发表以网络号为索引，而不是网络中的节点地址。
 - 层次化寻址的优点
- 构建可扩展网络的重要原则之一
 - 为了满足扩展性要求，尽可能减少节点内存储的信息以及节点之间交换的信息。
 - 最常见的方法是分层聚合。
 - IP采用两级寻址方式



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结





子网划分

- 问题
 - IP 地址耗尽问题
- 动机
 - 引入新的层次
- 解决方案
 - 寻址: 子网掩码
 - IP 转发



IP 地址耗尽问题

- IPv4 包含32bit的地址空间: 总共包含40亿个合法地址
 - 并非所有的地址均用于主机或接口标识
 - 部分地址用于多播或保留使用
- 为什么IP地址会耗尽?
 - Internet的快速发展
 - 低效的地址分配

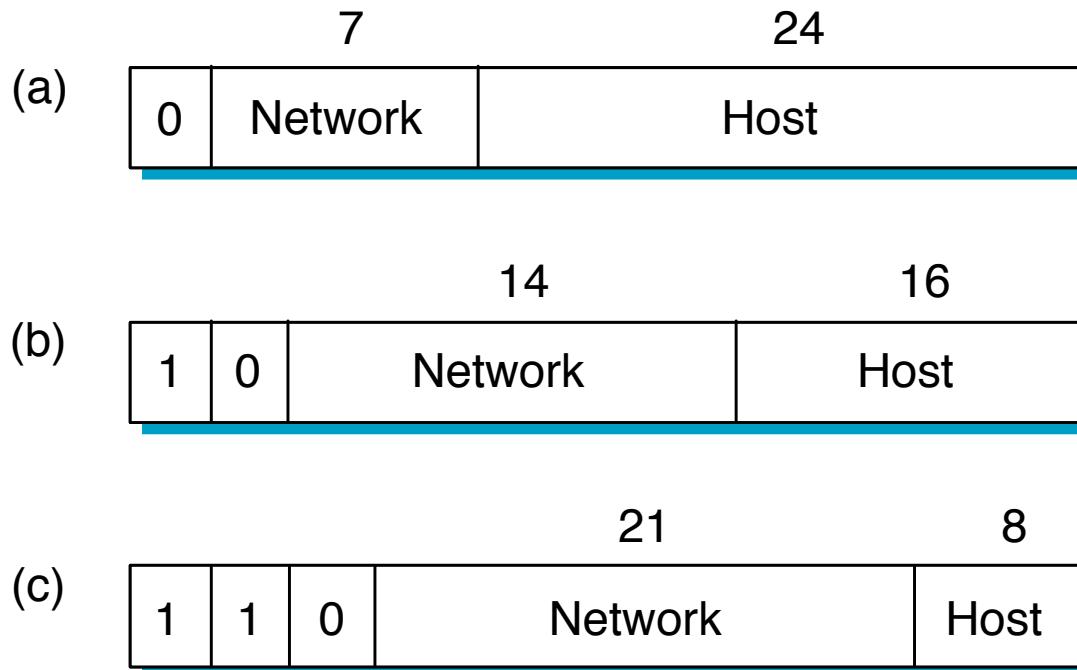


子网划分的动机

- 子网在IP地址中 引入了新的层次
- 子网划分将拥有同一个网络号的IP地址分配给若干个物理网络(子网)
 - 子网彼此离得很近, 拥有同一个网络号
 - 路由器只能选择一条路由到达任何子网
 - 子网可以减少对网络号的需求

IP地址回顾

- IP 地址最初采取分类方式管理
 - 网络被分为A, B, C三类

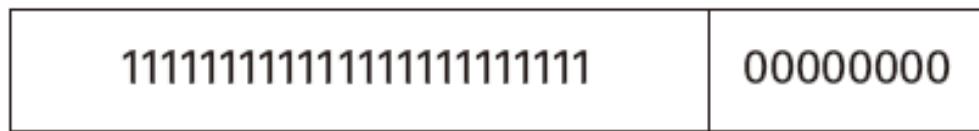




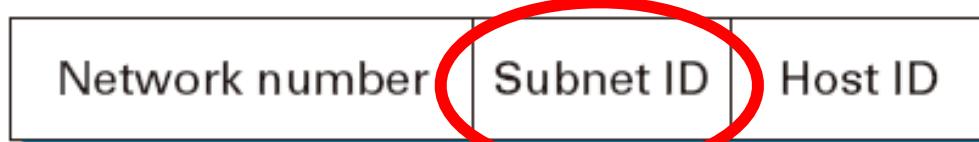
子网



Class B address



Subnet mask (255.255.255.0)



Subnetted address

□ 注意

- 主机号部分为全0 表示这个网络
- 主机号部分为全1 表示该网络中的所有主机, 即广播地址



低效的地址分配

- 示例1: 一个包含10台主机的网络
 - 需要一个C类网络号
 - 效率: $10/256$ 约为 4%
- 示例2: 一个拥有300台主机的网络
 - 需要一个B类网络号
 - 效率: $300/65536$ 约为 0.5%



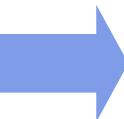
子网划分的效率

- 示例: 3 个网络, 其中2个包含50台主机, 另1个包含 100台主机
 - 传统分配
 - 需要3个C类网络号
 - 效率: $200/(3 \times 256)$ 约为 26%
 - 子网划分
 - 仅需要1个C类网络号
 - 划分3个子网: 其中2个分配64的地址, 另1个分配128个地址
 - 效率: $200/256$ 约为 78%



子网划分

- 问题
 - IP 地址耗尽问题
- 动机
 - 引入新的层次
- 解决方案
 - 寻址: 子网掩码
 - IP 转发

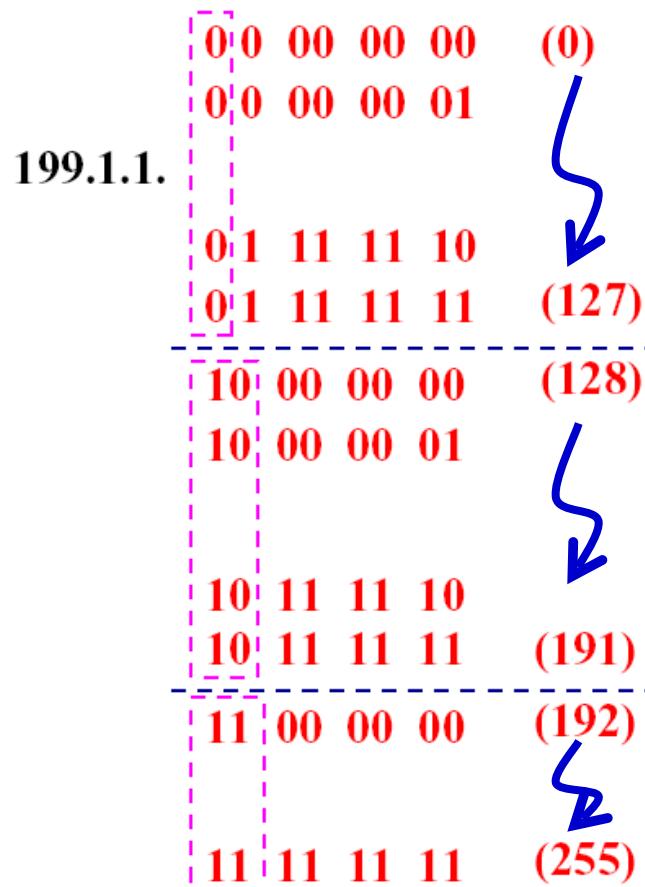




如何划分子网

- 示例: 给定一个C类网络号199.1.1.0, 划分为3个子网, 其中2个分配64个地址, 另1个128个地址
 - 对于包含128个地址的子网, 地址中的主机号部分占有7bits
 - 子网掩码为255.255.255.128
 - 对于包含64个地址的子网, 地址中的主机号部分占有6bits
 - 子网掩码为255.255.255.192

如何划分子网(续)



subnet number is given by
the first 25 bits

$$128 = 1000\ 0000$$

subnet mask: 255.255.255.128

subnet number: 199.1.1.0

subnet number is given by
the first 26 bits

$$192 = 1100\ 0000$$

subnet mask: 255.255.255.192

subnet number: 199.1.1.128

子网号	子网掩码	IP地址
192.1.1.0	255.255.255.128	192.1.1.0 – 192.1.1.127
192.1.1.128	255.255.255.192	192.1.1.128 – 192.1.1.191
192.1.1.192	255.255.255.192	192.1.1.192 – 192.1.1.255

如何划分子网(续)

199.1.1.	00 00 00 00	(0)
	00 11 11 11	(63)
	01 00 00 00	(64)
	01 00 00 01	(65)
	10 11 11 10	(190)
	10 11 11 11	(191)
	11 00 00 00	(192)
	11 11 11 11	(255)

subnet mask: 255.255.255.192

subnet number: 199.1.1.128

subnet mask: 255.255.255.0

subnet number: 199.1.1.0

优先划分较大的子网 !!



IP寻址示例

- 网络地址: 网络号和子网号部分全部为0
- 广播地址: 主机号部分全部为1
- 如果一台主机的IP地址为202.112.14.137, 子网掩码为255.255.255.224, 则网络地址和广播地址计算如下:
 - $137 = 1000\ 1001$
 - $224 = 1110\ 0000$
 - 网络地址的最后8bit为 $1000\ 0000 = 128$
 - 网络地址为202.112.14.128
 - 广播地址的最后8bit为 $1001\ 1111 = 159$
 - 广播地址为202.112.14.159



IP寻址示例

- 在子网中, 网络地址 (全0) 和广播地址 (全1) 占用了两个额外的地址
 - 在实际的以太网中, 选择其他地址作为主机或网关的IP地址
- 对于包含10台主机的子网, 计算其子网掩码:
 - 所有地址 $10 + 1 + 1 = 12$, $2^3 < 12 < 2^4$
 - 因此主机号部分占有4bits, 掩码的最后8bit为 $1111\ 0000 = 240$
 - 子网掩码为255.255.255.240



子网划分

- 问题
 - IP 地址耗尽问题
- 动机
 - 引入新的层次
- 解决方案
 - 寻址: 子网掩码
 - IP 转发





转发过程的变化

- 引入子网概念后, 分组的传送目标是子网而非(A,B, C)类网络, 转发表记录形式变为 <子网号, 子网掩码, 下一跳>
- 转发算法

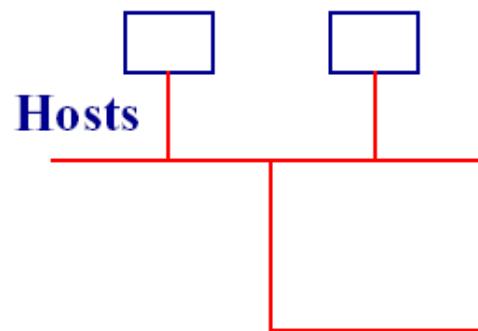
D = dst. IP addr.

```
for each entry in forwarding table
    D1 = SubNetMask & D
    if D1 = SubNetNum
        if NextHop is an interface
            deliver datagram directly to dst.
        else
            deliver datagram to NextHop (router)
```

子网划分转发表示例

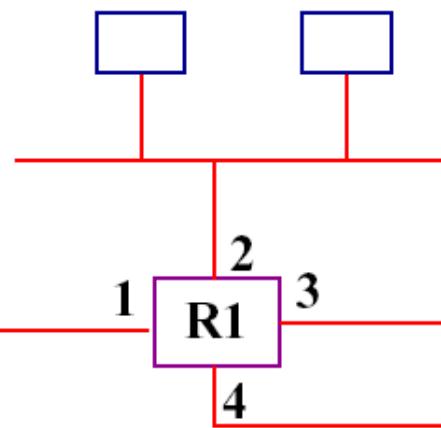
Subnet 199.1.1.0

Mask 255.255.255.128



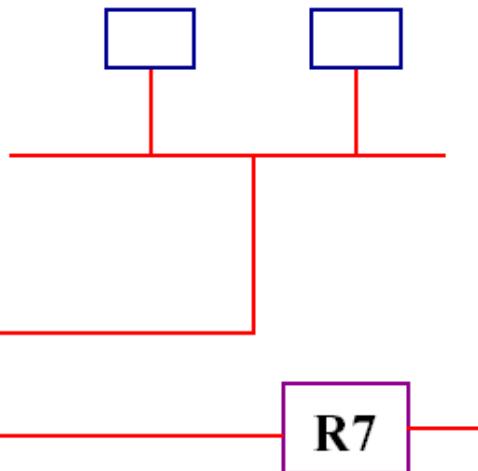
Subnet 199.1.1.128

Mask 255.255.255.192



Subnet 199.1.1.192

Mask 255.255.255.192





小结: 子网划分的优点

- 子网划分通过两种方式解决了网络扩展问题
- 第一, 提高了地址分配效率
 - 加入一个新的物理网络时不需要用光B类或C类网络的所有地址.
- 第二, 有利于信息聚合.
 - 可以减少路由器为了转发数据报到这些网络所需存储的信息量.



无分类路由选择(CIDR)

- 问题
 - IP地址固定结构的低效性
-
- 动机
 - IP地址不采用固定的分类结构
- 解决方案
 - CIDR
 - 汇聚路由



固定分类地址的问题

- 示例: 某ISP组建一个包含1000台主机的网络
 - 如果采用分类地址结构
 - 需要一个B类网络地址
 - 效率: $1000/65536$ 约为0.15%
- 实际问题
 - 网络大小的粒度划分非常粗略
 - 网络大小只能为256, 65k, 17M
- 即使采用子网划分技术也无法避免该问题



固定分类地址的问题

- 一种可能的解决方案
 - 以256个地址(C类网络地址)为单位分配空间
 - 然而, 这个方案会引起一个严重的问题: 对路由器超量存储的需求.
- 需要在两者之间达到平衡
 - 减少路由器的路由记录数 vs. 有效分配地址
 - 汇聚路由: 仅使用转发表中的一条记录来知道如何到达多个不同的网络.



无分类路由选择(CIDR)

- 问题
 - IP地址固定结构的低效性
- 动机
 - IP地址不采用固定的分类结构
- 解决方案
 - CIDR
 - 汇聚路由



动机

- 寻址
 - 更加合理的进行网路大小的粒度划分
 - 网络可以为“任意大小”: 256, 512, 1024, 2048, ...
- 路由选择
 - 分发多个具有相同前缀的连续C类地址块
 - 地址块包含数目为2的幂次方的C类网络



无分类路由选择(CIDR)

- 问题
 - IP地址固定结构的低效性
- 动机
 - IP地址不采用固定的分类结构



解决方案

- CIDR
- 汇聚路由



解决方案

- CIDR: Classless InterDomain Routing 无分类域间路由选择
 - RFC 1518和RFC 1519, 定义了新的IP地址块分配方式以及IP分组路由选择的新方法
 - 基于变长的子网掩码 (VLSM) 允许分配任意长度前缀



示例

- 示例:某ISP组建一个包含1000台主机的网络
 - 成功获得了4个连续的C类网络地址块
 - 假设C类地址网络号为 199.199.0-199.199.3
 - 这些C类地址网络号部分的高22bit是相同的
 - 基于CIDR, 可以认为这是一个网络号占用22位的网络, 记为199.199.0.0/22
 - 这种标识表示网络号为 199.199.0.0的高22位



CIDR 和转发表

- 转发表

网络/掩码长度	下一跳
199.199.199.0/22	Router A
199.199.192.0/17	Router B
200.200.16.0/20	Router C
200.200.21.0/24	Router D

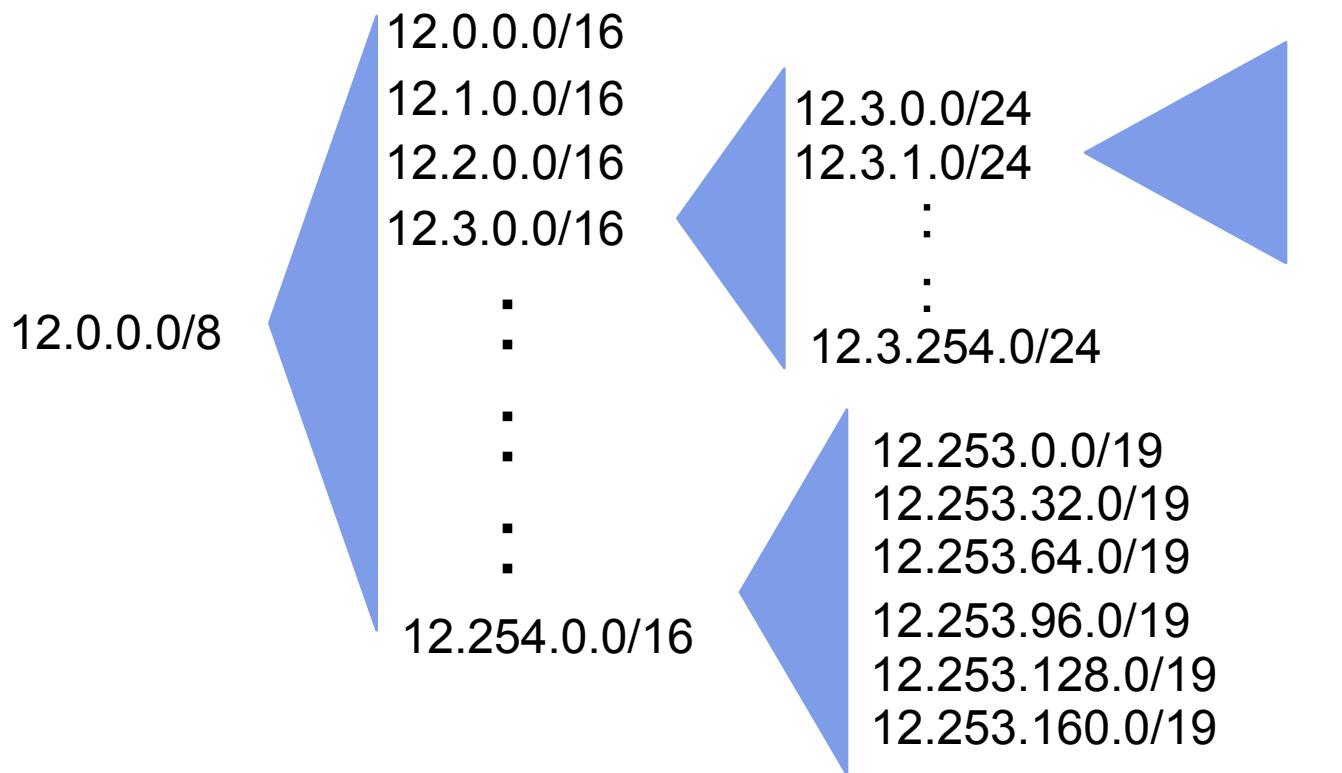
- 当路由器收到一个分组
 - 查询其目的IP地址
 - 遍历转发表
 - 匹配: 掩码长度为n, 目的IP地址ide高n位 = 网络号的高n位



CIDR: 灵活的层次化地址分配

- 采用前缀进行Internet扩展

- 对持续的块进行地址分配(前缀)
- 基于前缀进行路由选择和分组转发
- 至今, 路由表包含约200,000个前缀(vs. 4B)





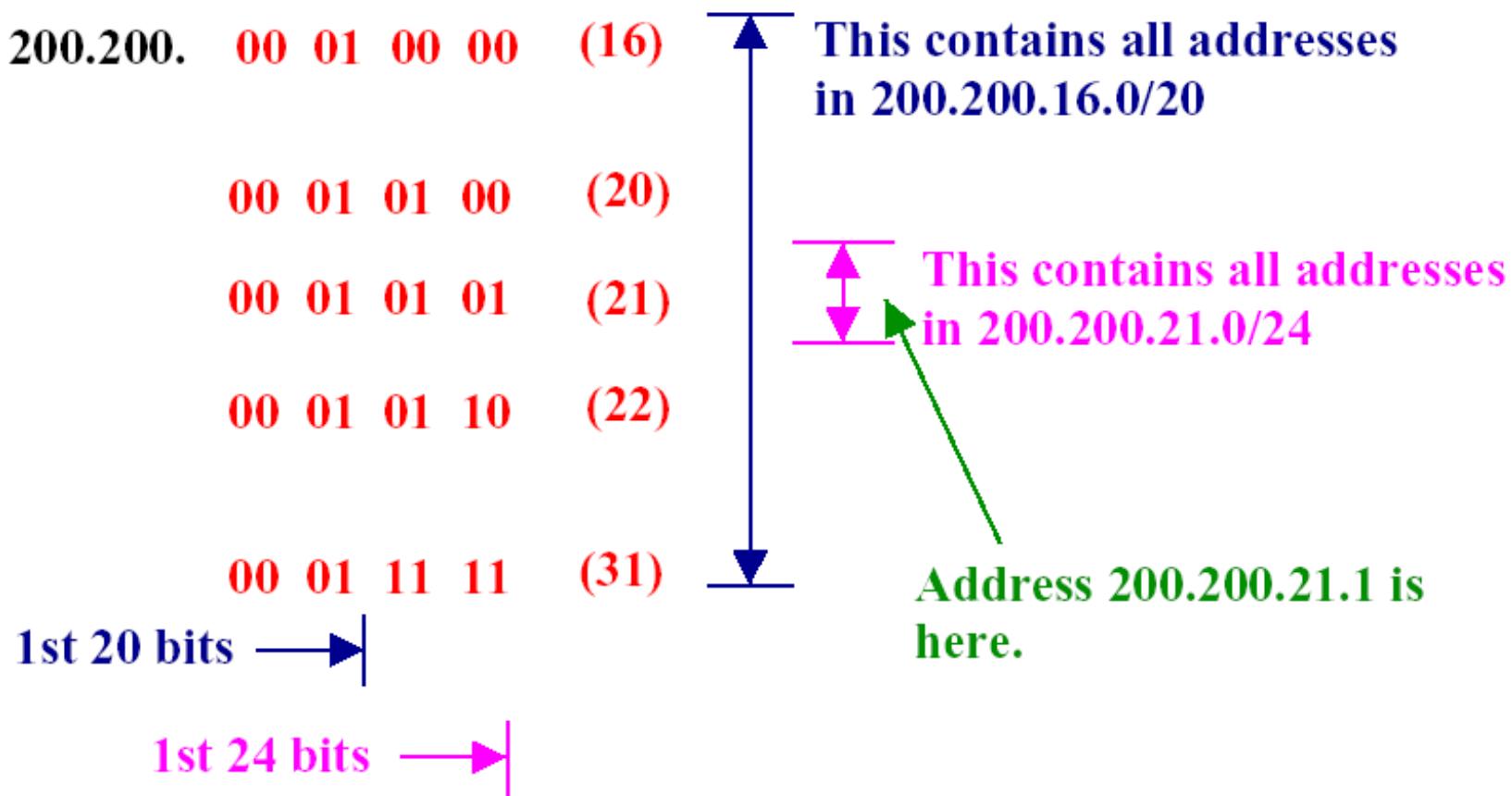
最长前缀匹配

- 示例: 目的IP地址为200.200.21.1
 - 倒数第2条记录: 200.200.16.0/20
 - 最后一条记录: 200.200.21.0/24
 - 均与目的IP地址相匹配
- 原则: 分组与最长的前缀匹配
 - 本例选择结果: 200.200.21.0/24

最长前缀匹配: 图示说明

Dst: 200.200.21.1

We ignore the last 8 bits





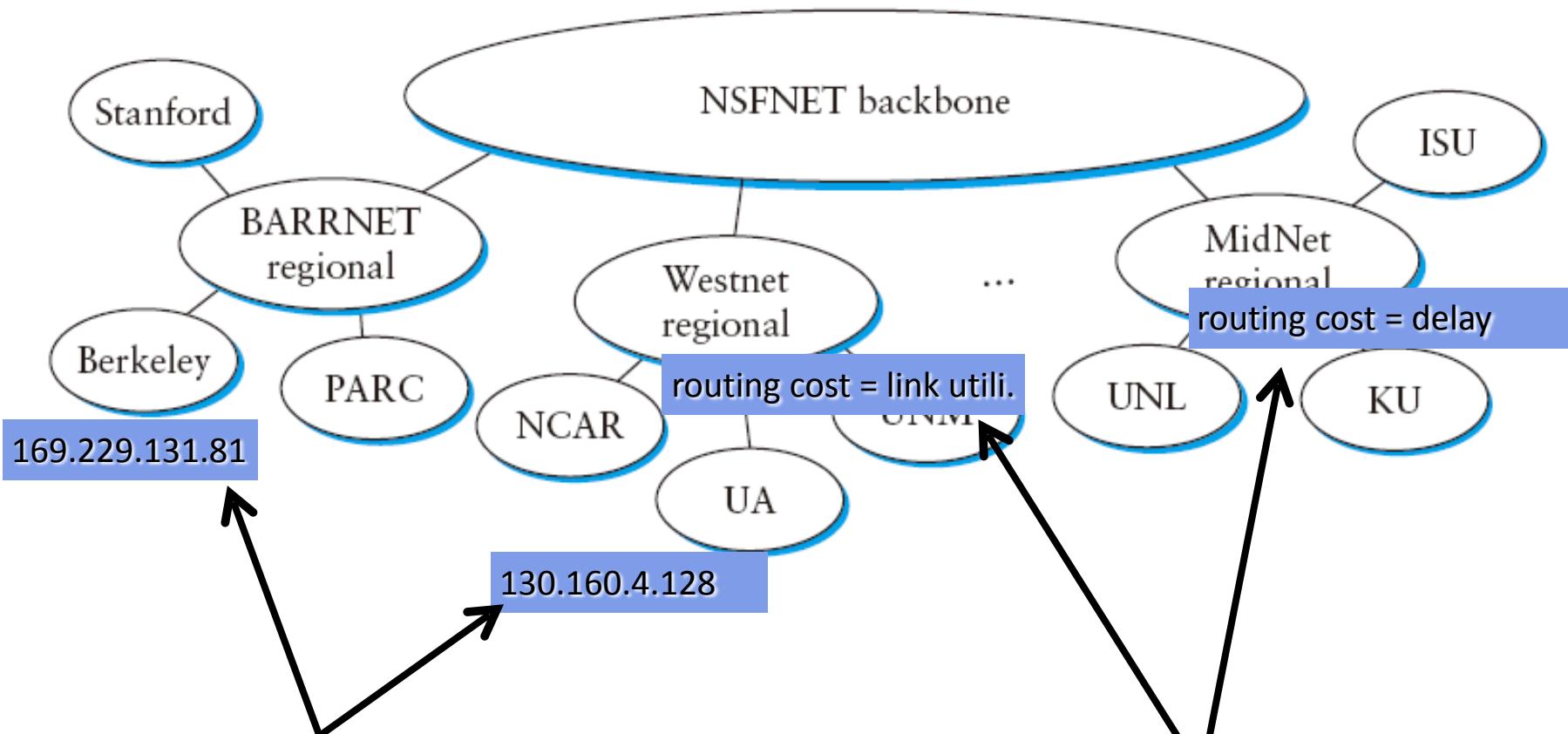
小结

- IP的扩展
 - 问题 1: 低效的地址空间使用
 - 问题 2: 路由表和转发表的记录数量
- 两种改进方法
 - 子网划分: 仅能解决问题1
 - CIDR: 同时有效解决问题1和2

全球Internet的实际情况



1990年时Internet的树形结构



大学可以为内部用户免费分配 IP 地址
=> 地址空间的利用率

ISPs 对网络中所使用的最佳路由协议以及如何定义链路评价指标都有不同的看法
=> 自治系统

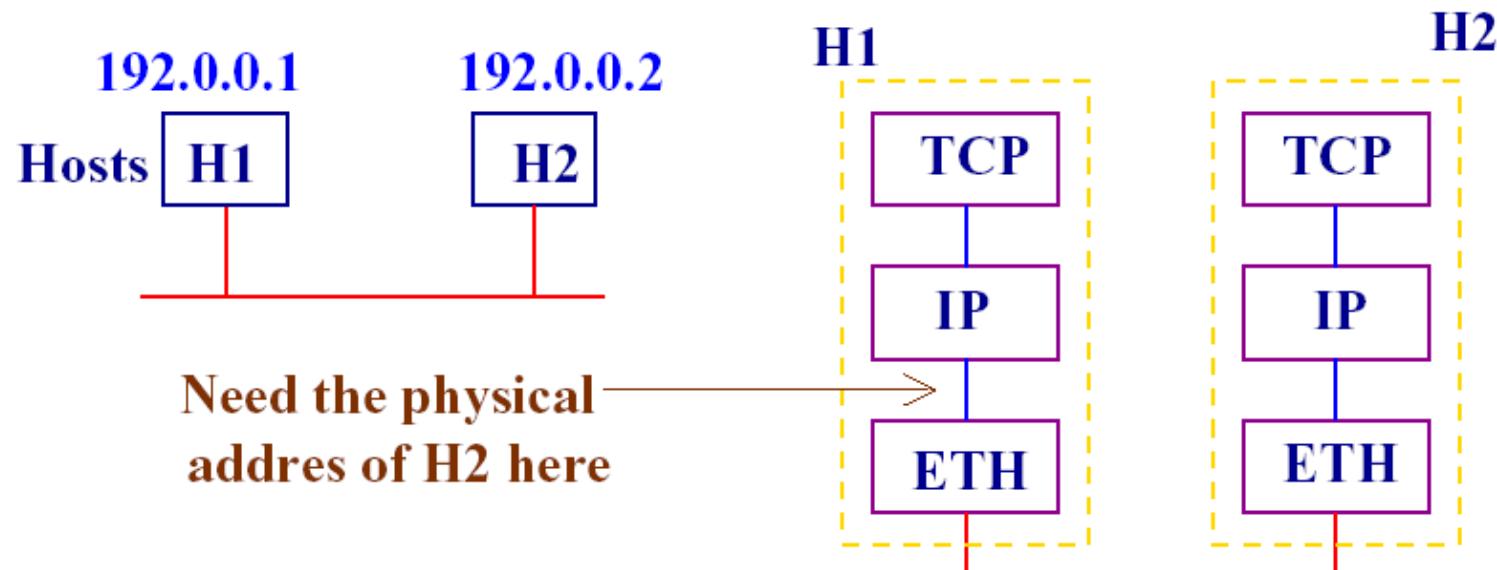
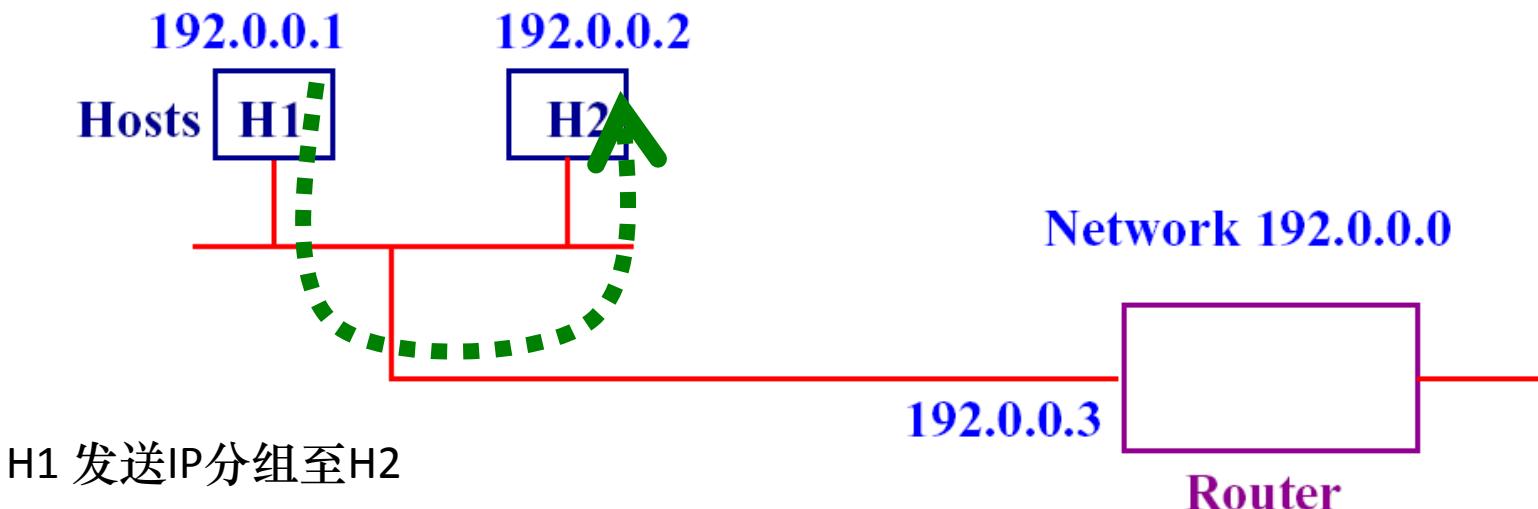


提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结



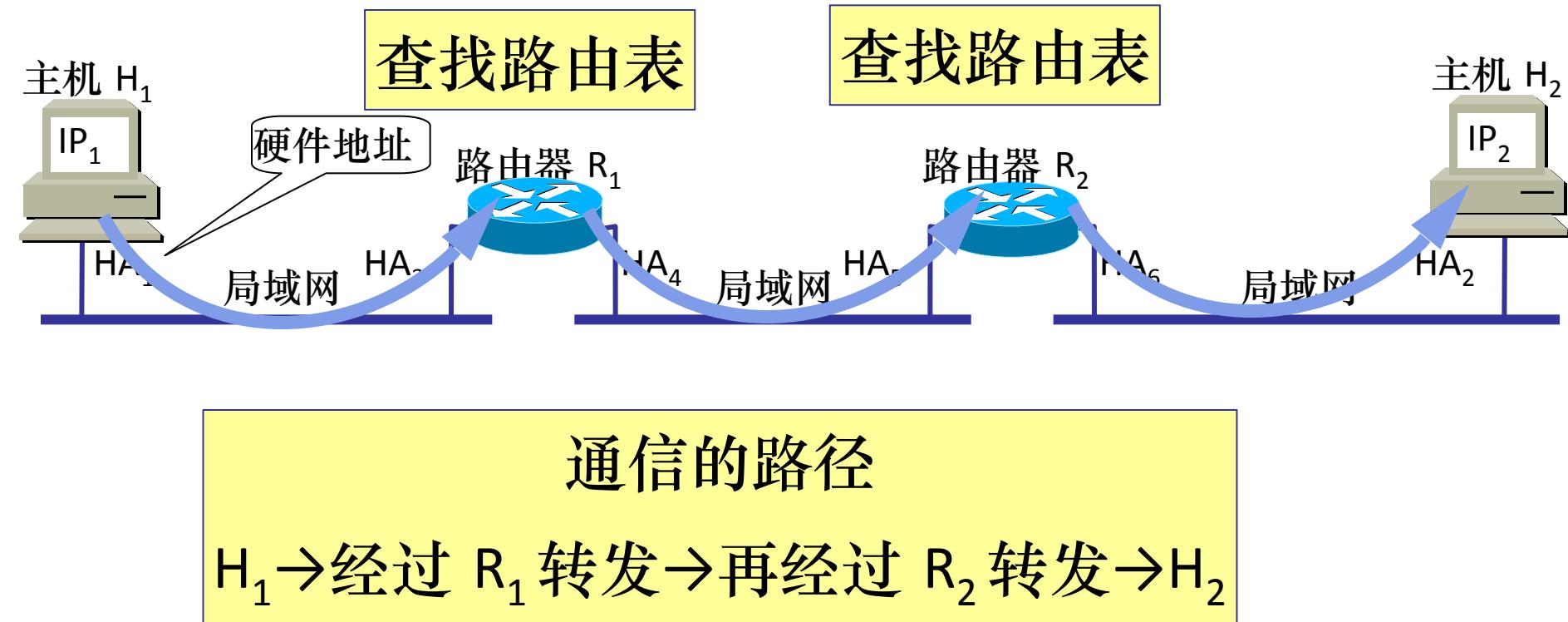
问题：网络内部的IP发送



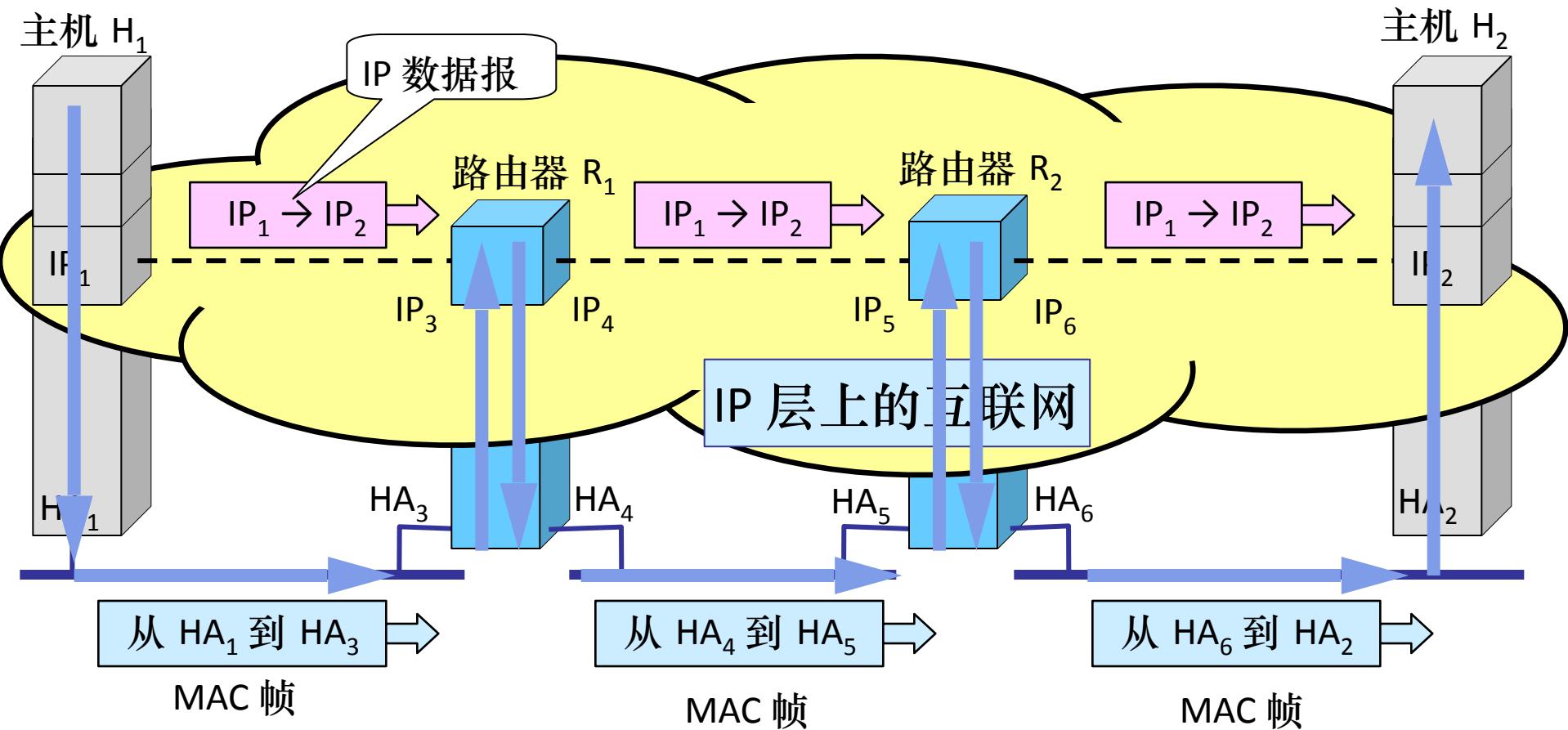
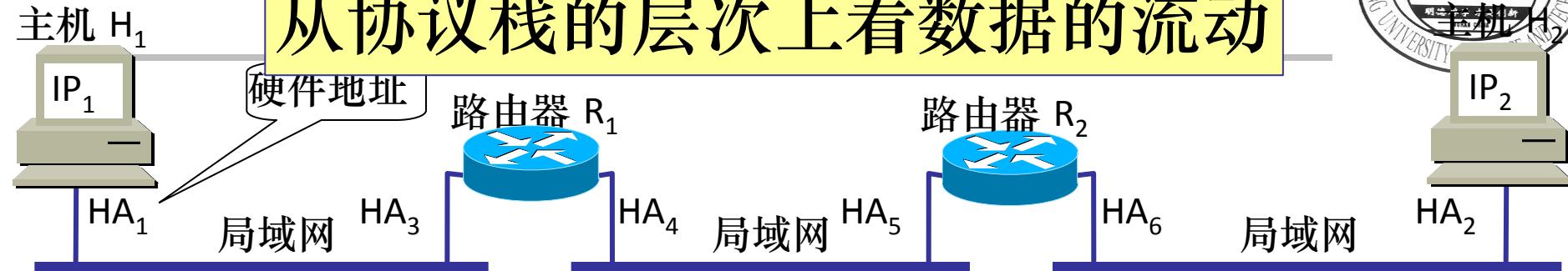
需要将IP地址转换为数据链路层地址



IP地址与数据链路层地址

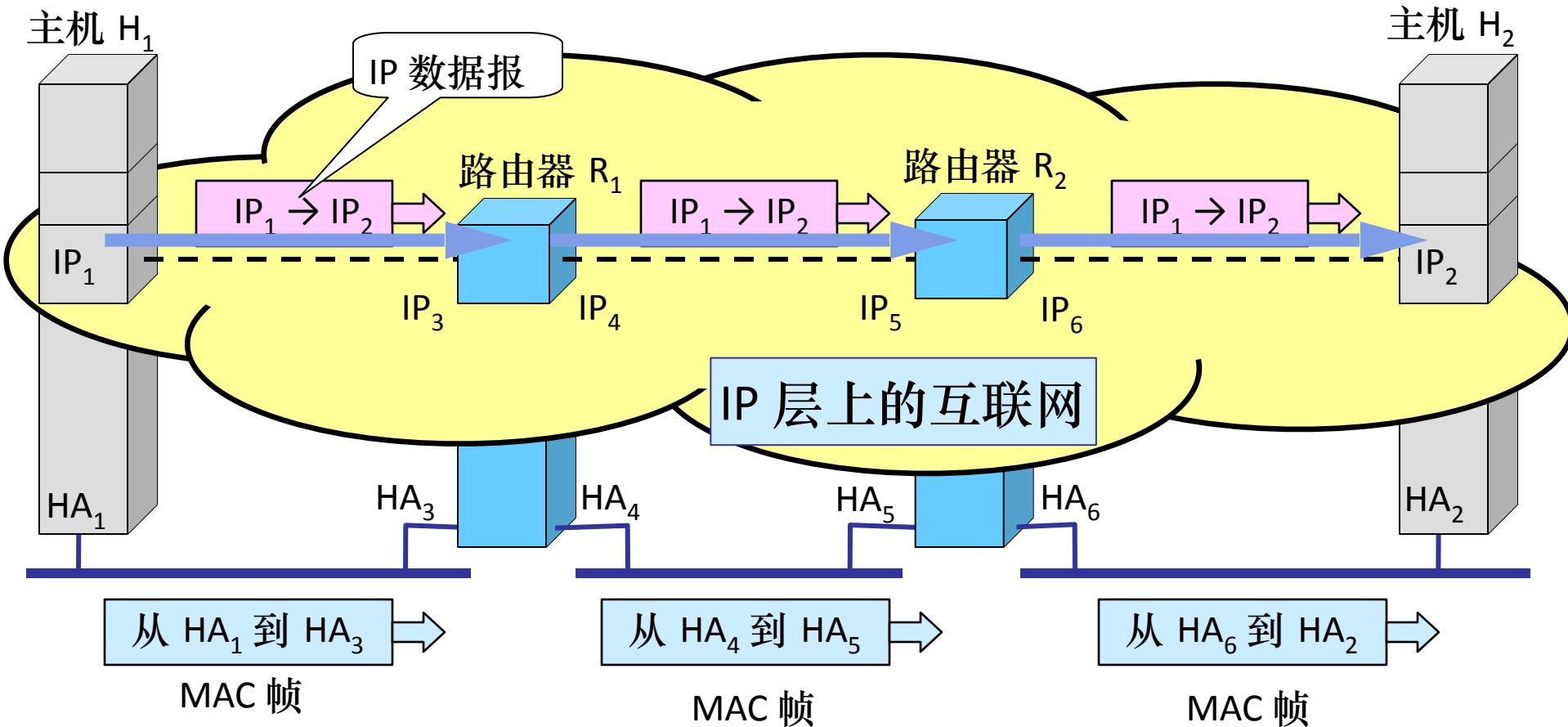
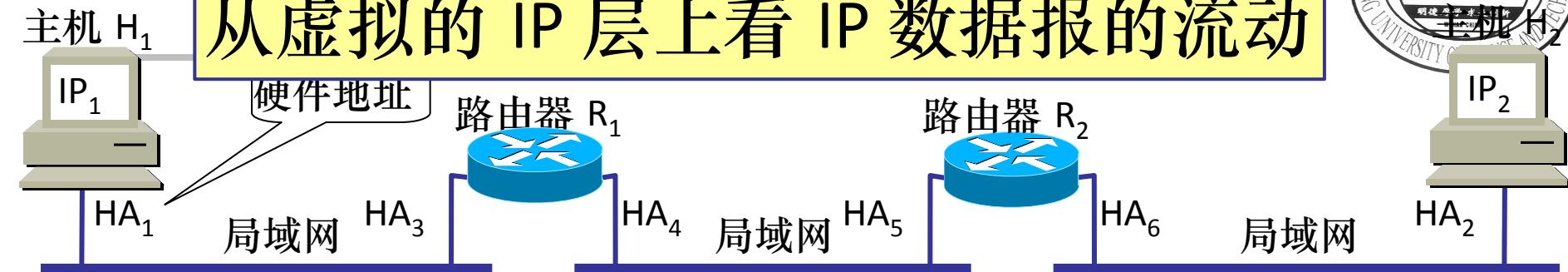


从协议栈的层次上看数据的流动



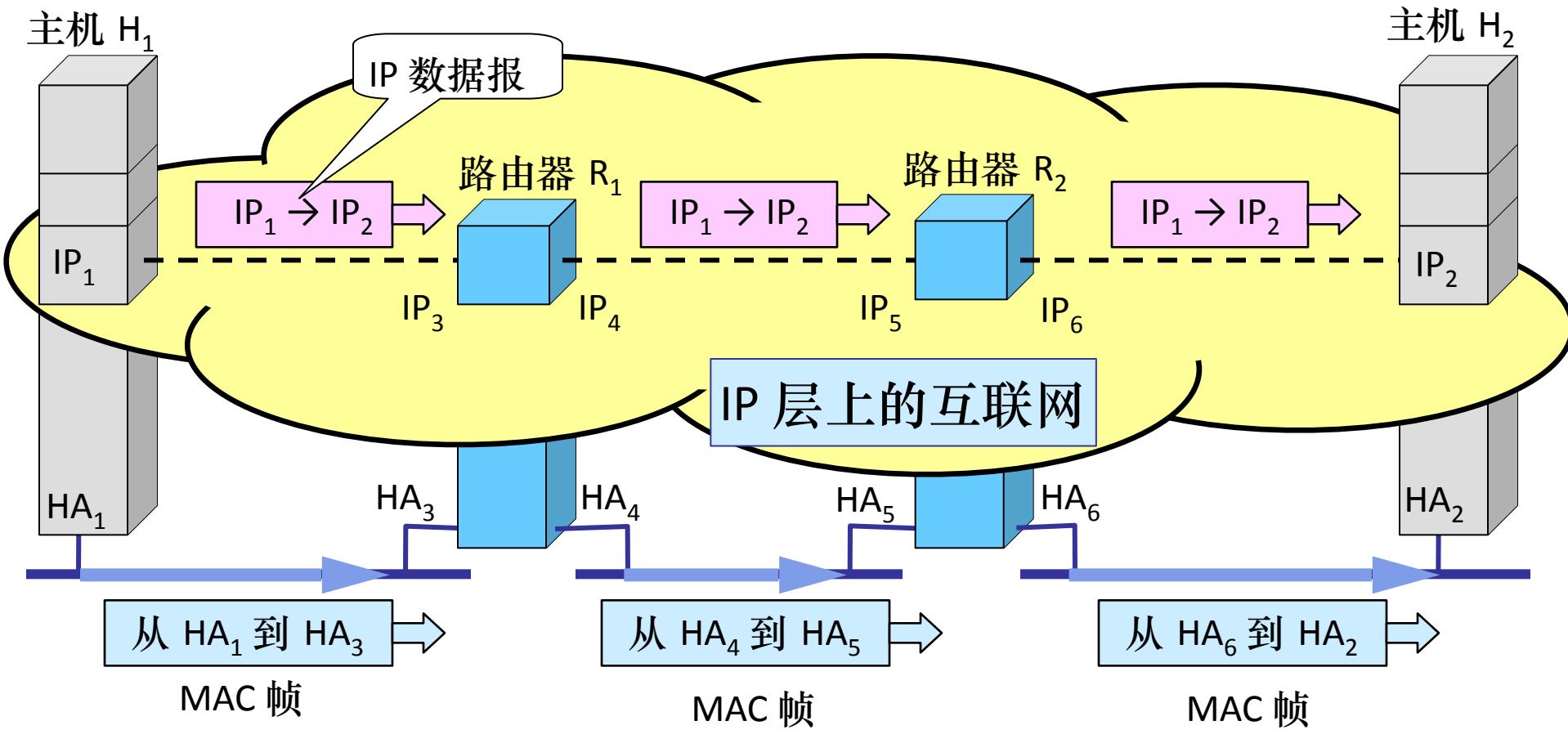


从虚拟的 IP 层上看 IP 数据报的流动

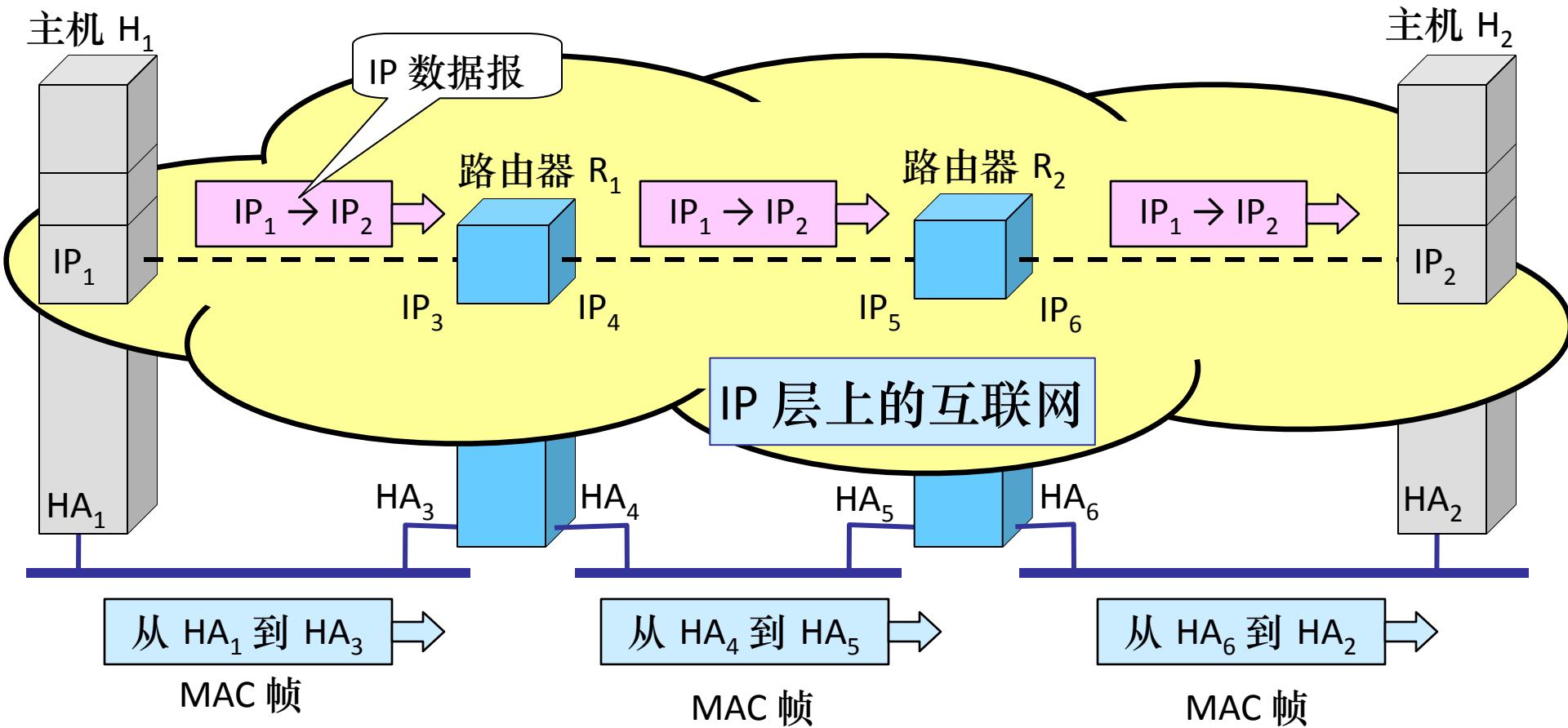




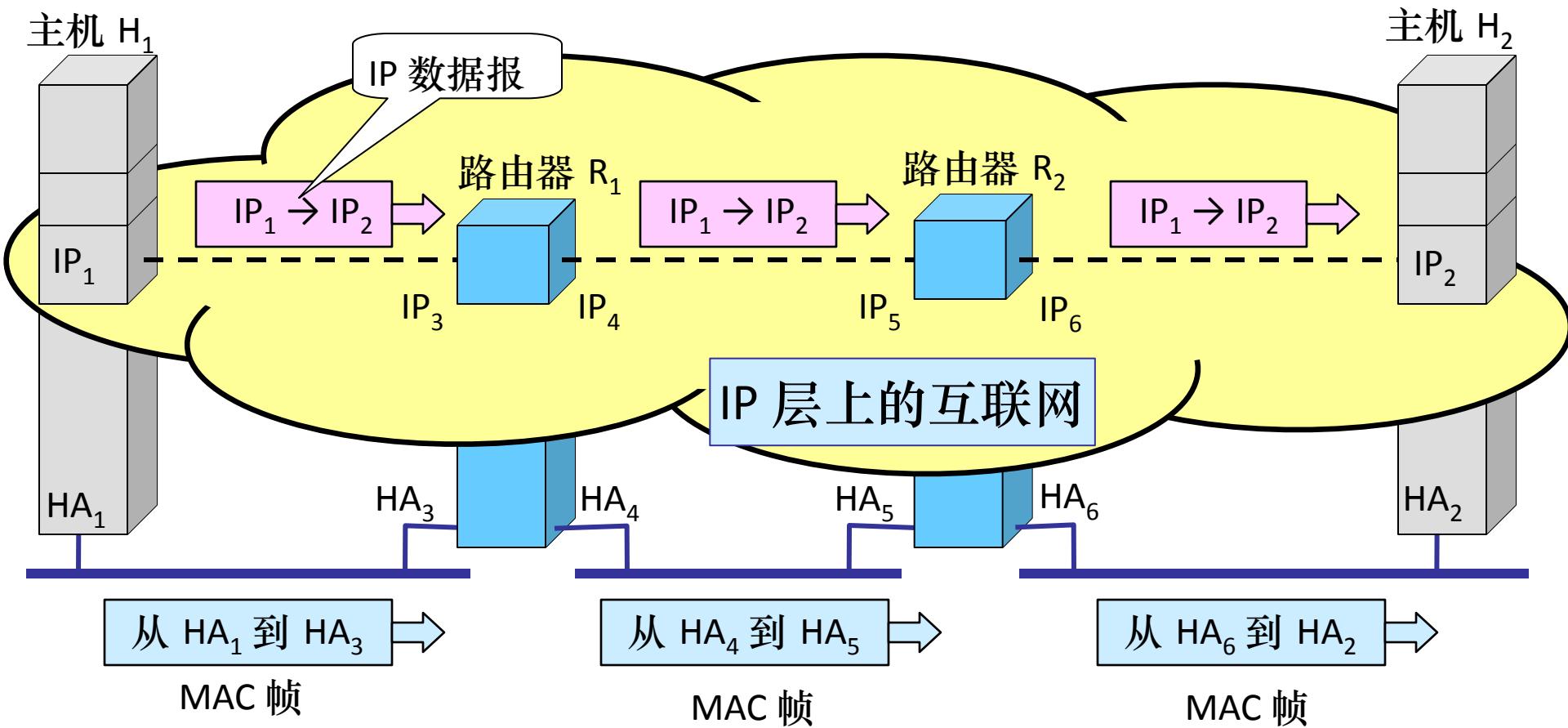
在链路上看 MAC 帧的流动



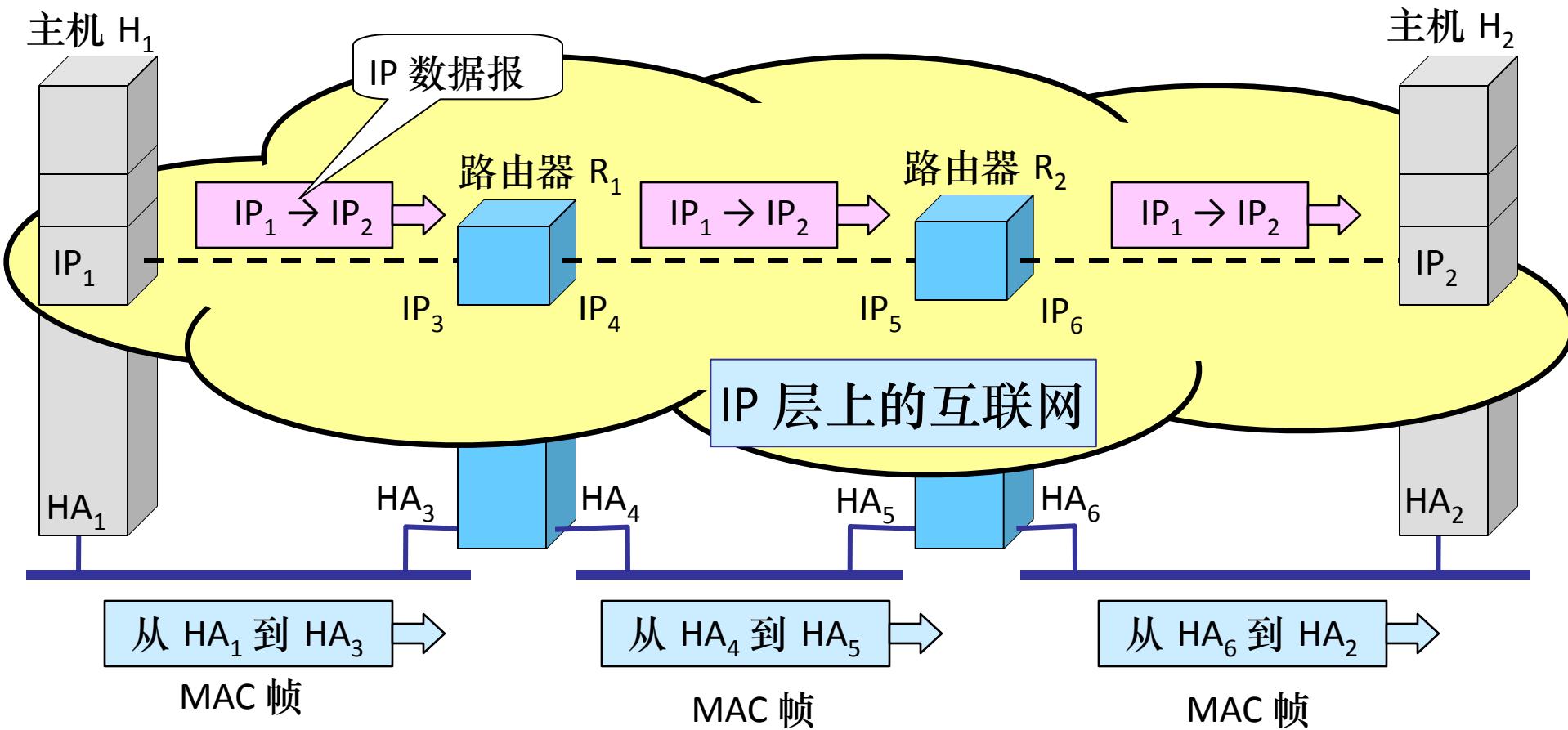
在 IP 层抽象的互联网上只能看到 IP 数据报
图中的 $IP_1 \rightarrow IP_2$ 表示从源地址 IP_1 到目的地址 IP_2
两个路由器的 IP 地址并不出现在 IP 数据报的头部中



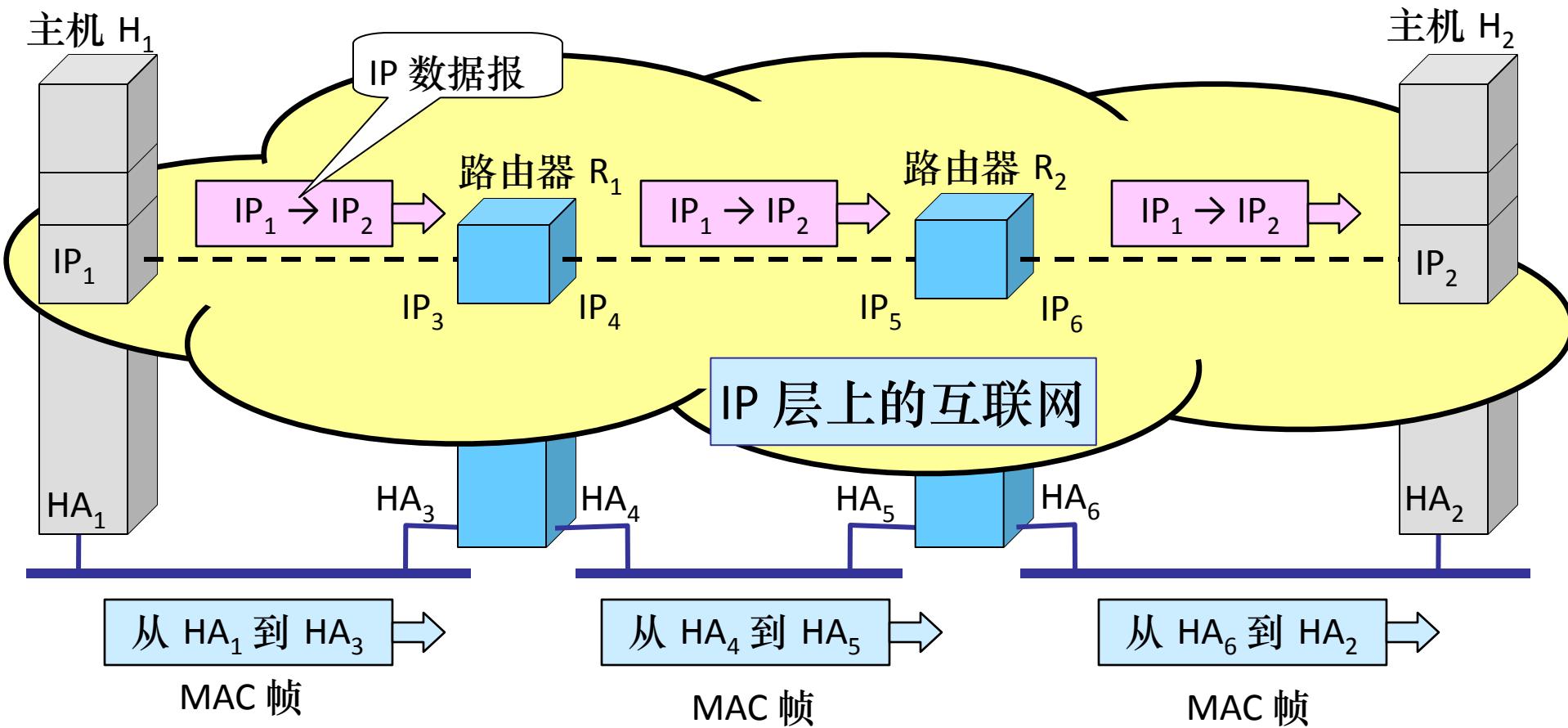
路由器只根据目的站的 IP 地址的网络号进行路由选择



在具体的物理网络的链路层
只能看见 MAC 帧而看不见 IP 数据报



IP层抽象的互联网屏蔽了下层很复杂的细节
在抽象的网络层上讨论问题，就能够使用
统一的、抽象的 IP 地址
研究主机和主机或主机和路由器之间的通信





地址转换

- 物理地址
 - 数据链路层地址 == 某一物理网络定义的硬件地址
- 地址转换的解决方案
 - 方案1: 直接编码
 - 将物理地址编码为IP地址的主机号部分
 - Ethernet 48bit 地址?
 - 方案2: 表
 - 网络管理员在每一个主机内部维护一个IP地址—物理地址的映射表
 - 如何将映射表分发给每一个节点?



地址转换协议(ARP)

- 给定目的主机的IP地址, ARP可以找到同属一个物理网络内部的目的主机的物理地址
- ARP采用广播机制
- 每一个主机维护一个ARP表 : <IP 地址, 物理地址>
 - 15分钟记录未更新则超时失效
- 当ARP表中无目的IP地址的对应记录, 则ARP协议开始启动

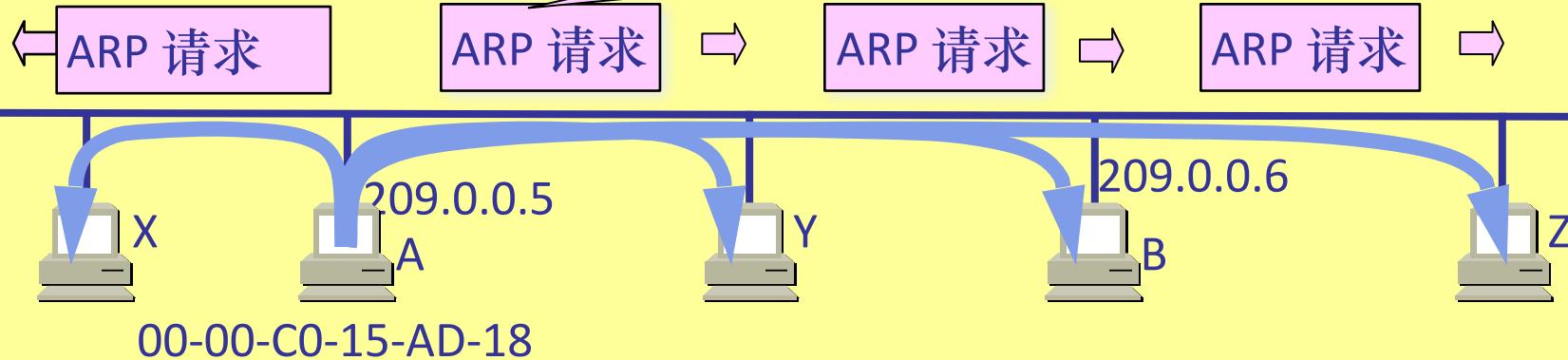


ARP工作原理

- 主机X广播一个 ARP请求报文来探寻未知的节点D的IP地址
- 节点D 向源节点反馈其物理地址
 - D节点 在本地ARP表中增加或更新节点X对应的记录
- 节点X将收到的信息写入其ARP表
- 同一网络内的其他主机
 - 如果本地的ARP表中已存在节点X的IP地址对应的记录，则更新记录 *<IP address of X, physical address of X>*
 - 否则, do nothing

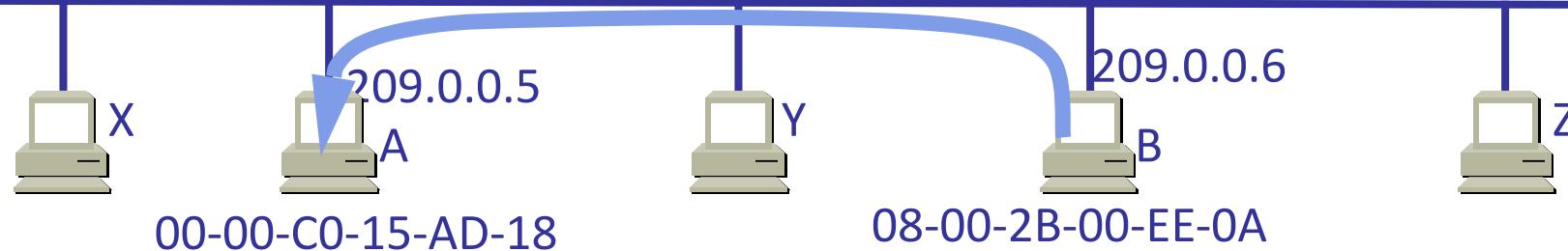
主机 A 广播发送
ARP 请求分组

我是 209.0.0.5，硬件地址是 00-00-C0-15-AD-18
我想知道主机 209.0.0.6 的硬件地址



主机 B 向 A 发送
ARP 响应分组

我是 209.0.0.6
硬件地址是 08-00-2B-00-EE-0A





ARP 分组格式

0	8	16	31
Hardware type = 1		ProtocolType = 0x0800	
HLen = 48	PLen = 32	Operation	
SourceHardwareAddr (bytes 0–3)			
SourceHardwareAddr (bytes 4–5)		SourceProtocolAddr (bytes 0–1)	
SourceProtocolAddr (bytes 2–3)		TargetHardwareAddr (bytes 0–1)	
TargetHardwareAddr (bytes 2–5)			
TargetProtocolAddr (bytes 0–3)			

将**IP**地址映射为以太网地址的**ARP**分组格式



ARP的深入讨论

- ARP 机制不同的数据链路地址
 - IP 地址 → 以太网MAC地址
 - IP地址→ 令牌环网MAC地址
 - IP地址→ ATM地址(LANE) , …
- RARP
 - 反向地址解析协议
 - 数据链路层地址→ IP地址
- ARP表和 ARP广播风暴
- Windows 命令: arp



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 子网划分和无类地址
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结





问题: 动态IP设置

- 对于网络而言, 主机节点必须拥有
 - IP地址, 网络掩码以获得网络地址
 - 默认路由器和DNS服务器的IP地址
- 问题1: IP地址需要重新配置.
 - IP地址不仅要求在给定的互联网中唯一, 而且必须能够反映互联网的结构.
- 问题2: IP配置需要自动化.
 - 网络管理员很难手工完成所有节点的地址
 - Zero Networking Configuration的需求
- 解决方案: 动态主机配置协议(Dynamic Host Configuration Protocol, DHCP)



动态主机配置协议 (DHCP)

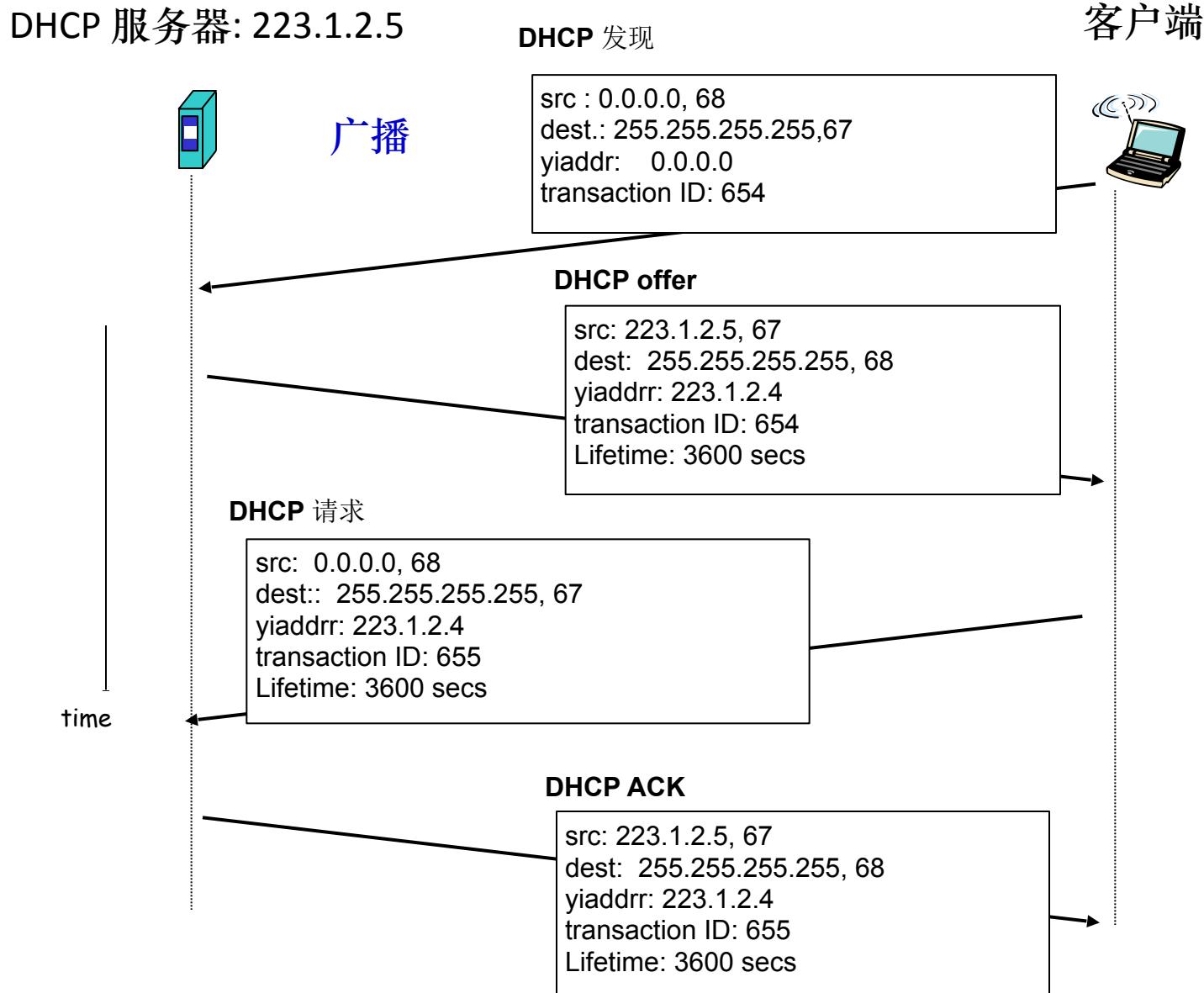
- 目标: 允许主机加入网络时自动从网络服务器获取IP地址
 - 需要使用时重新租用地址
 - 允许地址重复使用 (当节点工作时始终占用地址)
 - 支持节点随时加入网络
- 降低大型网络的管理负担



DHCP 工作原理

- 主机与DHCP服务器通信的四种标准报文
 - DHCPDISCOVER (从主机到DHCP服务器)
 - 主机广播该报文
 - DHCPOFFER (从DHCP服务器到主机)
 - DHCP服务器给出包含配置参数的提议
 - DHCPREQUEST (从主机到DHCP服务器)
 - 主机选择并接受某一提议
 - DHCPPACK (从DHCP服务器到主机)
 - DHCP服务器确认配置

示例: DHCP 客户端与服务器协商过程





DHCP深入讨论

- DHCP 时间
 - DHCP服务器: 租用时间
 - 4字节, 1秒 – 136 年
 - DHCP客户端: 超时触发器
 - 0.5T 和 0.875T
- Windows 命令:
 - ipconfig /release
 - ipconfig /renew



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
 - 什么是互联网?
 - 服务模型
 - 全局地址
 - IP的数据转发
 - 地址转换(ARP)
 - 主机配置(DHCP)
 - 差错报告 (ICMP)
- 路由
- 实现和性能
- 总结





问题: 如何报告错误

- Internet如何处理错误?
 - 当IP在数据报传送受阻而要将其丢弃时
 - 不能默默地归于失败
- Internet 控制消息协议(ICMP)
 - IP协议的附属协议
 - 当路由器或主机无法成功处理IP数据报时向源主机发送错误报文(定义了错误报文集合)

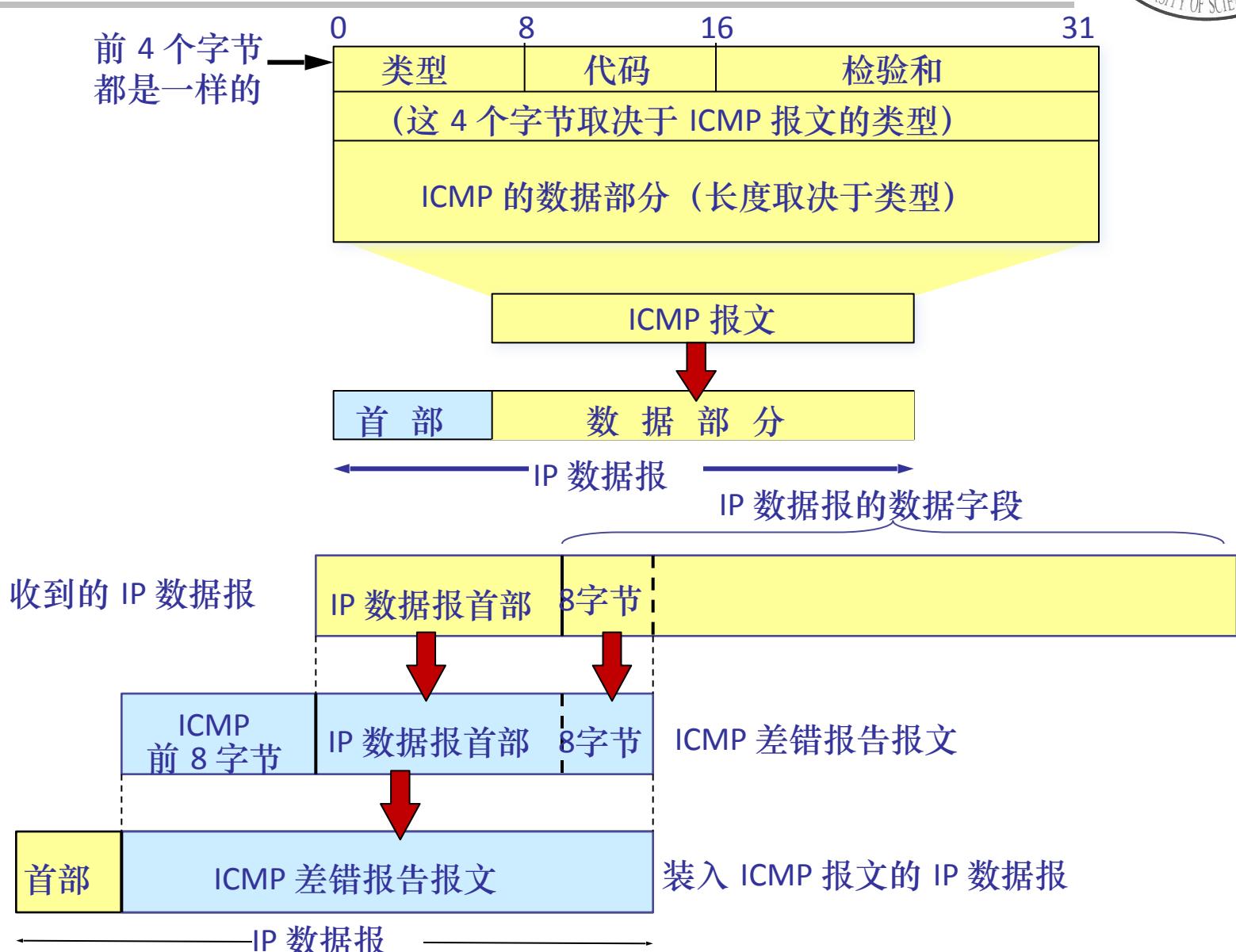


Internet 控制消息协议

- 主机和路由之间进行网络层信息的通信
 - 错误报告: 无法到达的主机, 网络, 端口, 协议
 - echo request/reply (ping命令)
- 运行于IP之上的网络层协议:
 - ICMP 消息封装于IP数据报中
- ICMP 报文: 错误原因(类型, 编码)

类型	编码	描述
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

ICMP分组格式





ICMP的应用: traceroute

- 源节点发送若干个UDP数据报至目的节点
 - 第一个数据报的TTL = 1
 - 第二个数据报的TTL=2, 依次类推.
- 当第n个数据报到达第n个路由器:
 - 路由器丢失数据报
 - 向源节点发送一个ICMP报文 (type 11, code 0)
 - 报文中包含路由器的名字及IP地址
- 当ICMP报文到达源节点, 源节点计算RTT
- Traceroute重复执行上述过程3次

停止原则

- UDP 数据报最终到达目的主机
- 目的节点发送ICMP “host unreachable” 分组 (type 3, code 3)
- 当源节点收到该ICMP报文, 则停止发送.

Windows 命令: tracert

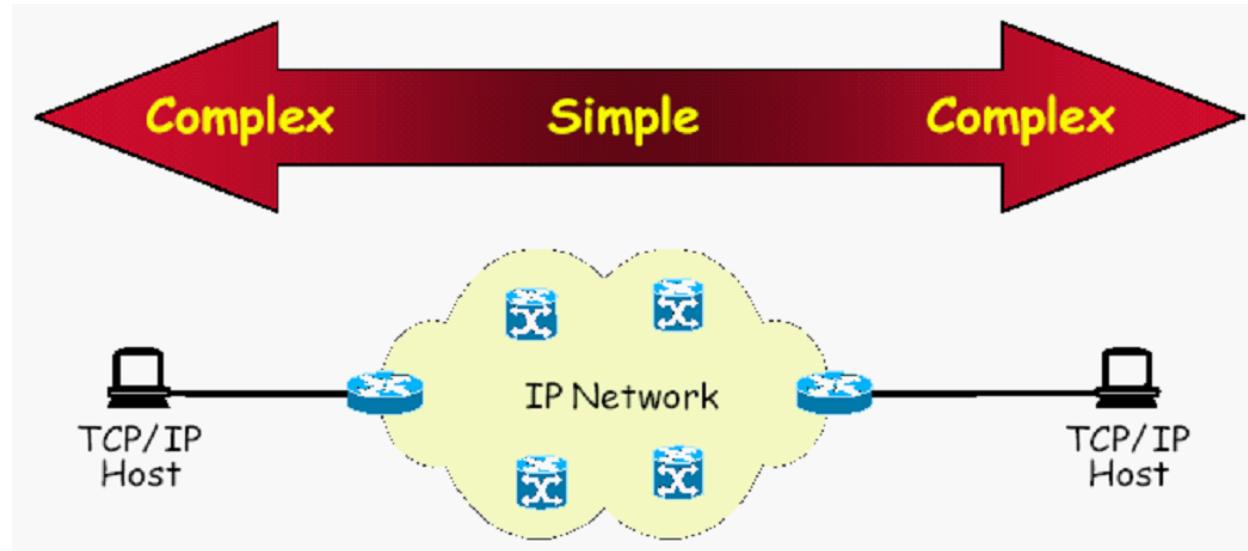
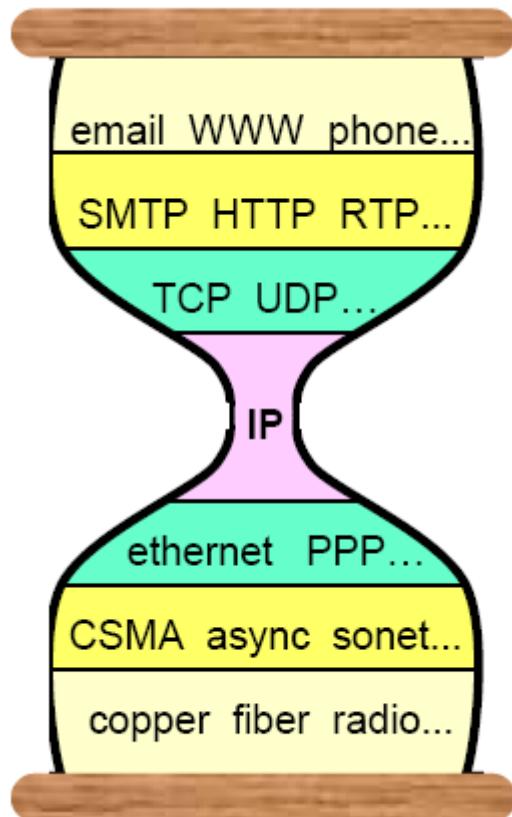


小结

- IP机制解决了异构性问题
 - 定义了尽最大努力交付服务模型, 提供不可靠的数据报传送
 - 提供一种通用的分组格式, 分段和重组机制使分组能够适应具有不同MTU的网络
 - 提供识别所有主机的全局地址空间, ARP机制使其能够在具有不同物理地址编制方案的网络上运行
- IP机制解决了扩展性问题
 - 地址采用分层聚合, 地址被划分为网络地址和主机地址

小结

- IP 服务模型





思考

- 网络服务是否必须基于主机地址?
- 网络的分布式管理存在什么缺点?

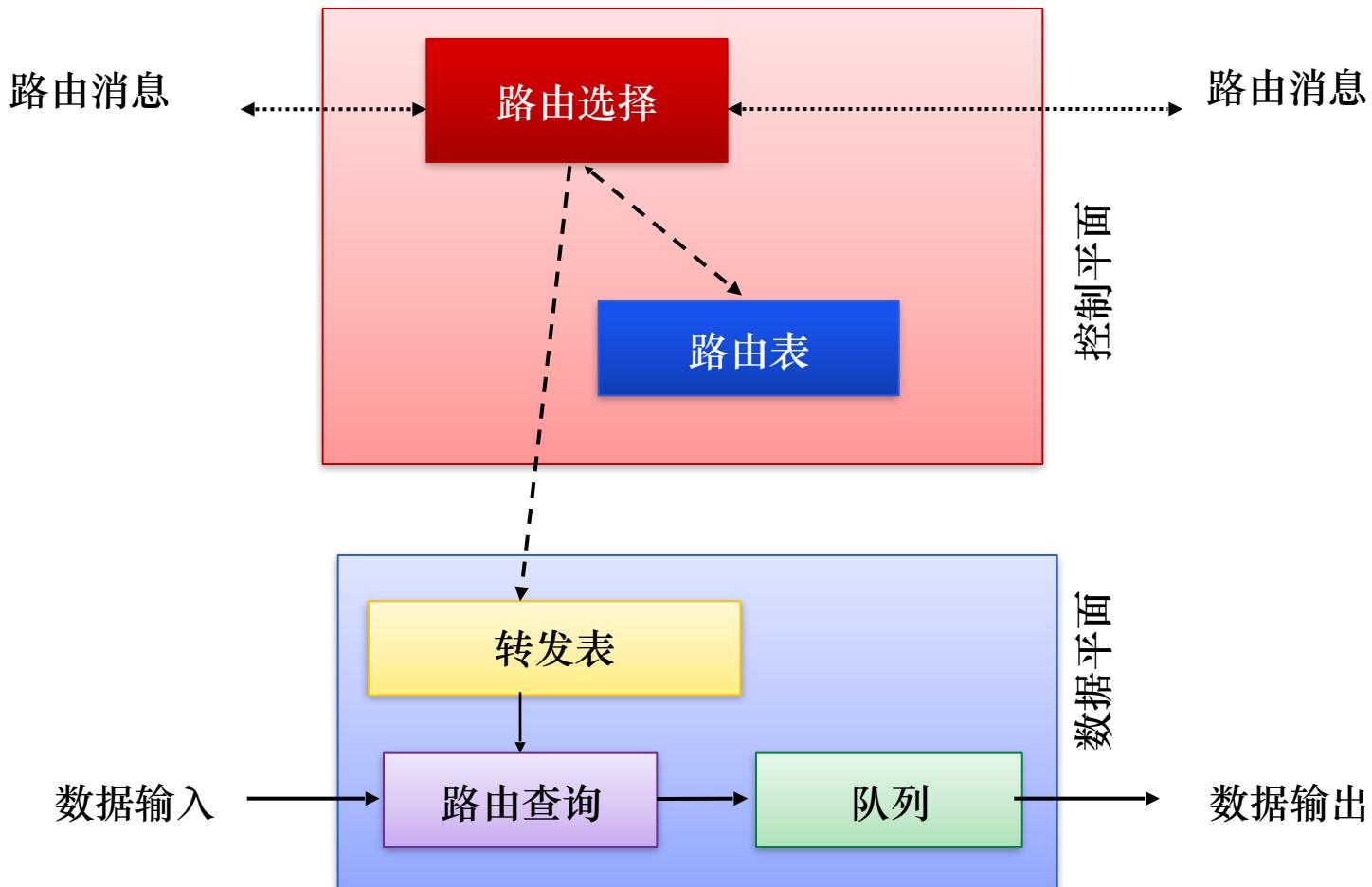


提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- **路由**
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结



转发vs.路由





路由表和转发表

- 路由表

- 由路由算法建立, 进而生成转发表
- 包含从网络号到下一跳的映射

Network Number	NextHop
10	171.69.245.10

- 转发表

- 分组转发时使用转发表, 表中需要包含足够的信息完成转发功能
- 包括从网络号到输出接口的映射以及一些MAC信息, 例如下一跳的以太网地址

Network Number	Interface	MAC Address
10	if0	8:0:2b:e4:b:1:2



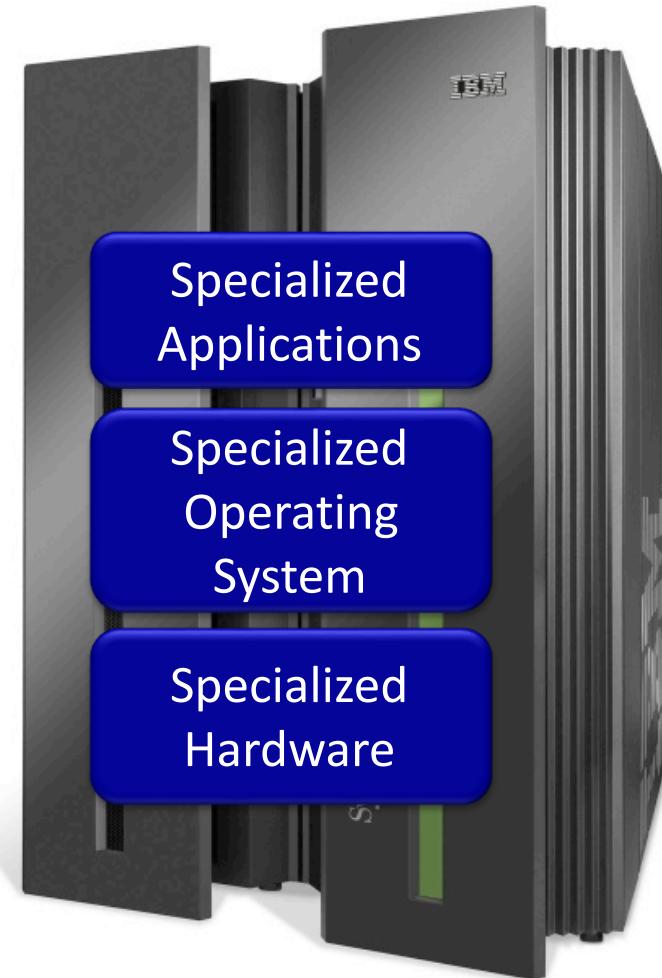
本节说明

- 本节讨论**小到中型网络**的路由选择问题, 而非整个 Internet的.
- 本机讨论的路由选择
 - 解决方案可扩展? NO
 - 为中等规模的网络而设计(少于100个节点)
 - 本节讨论的协议通常称为域内路由协议, 或内部网关协议 (IGPs)

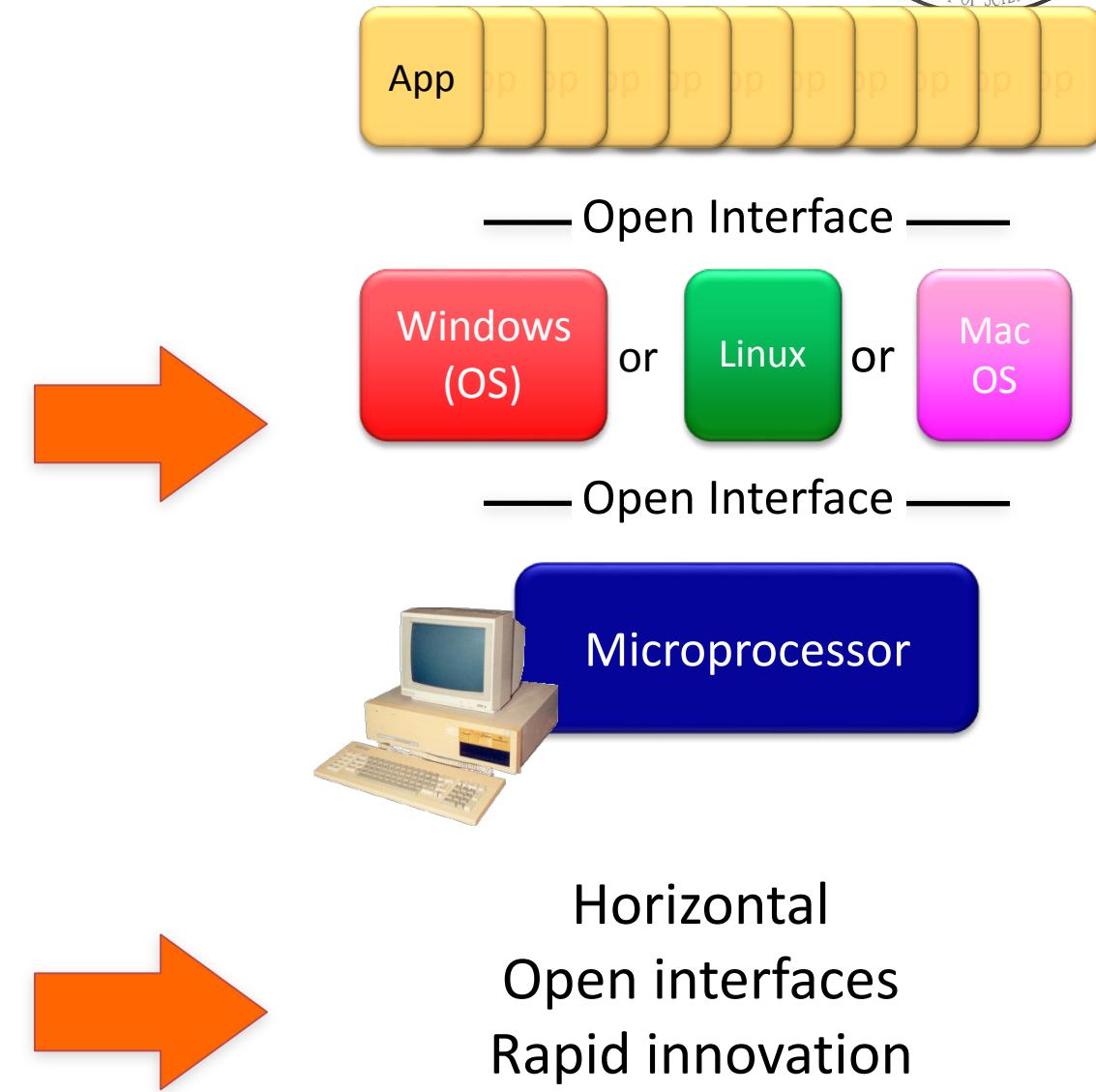
未来互联网？



智慧移动互联网

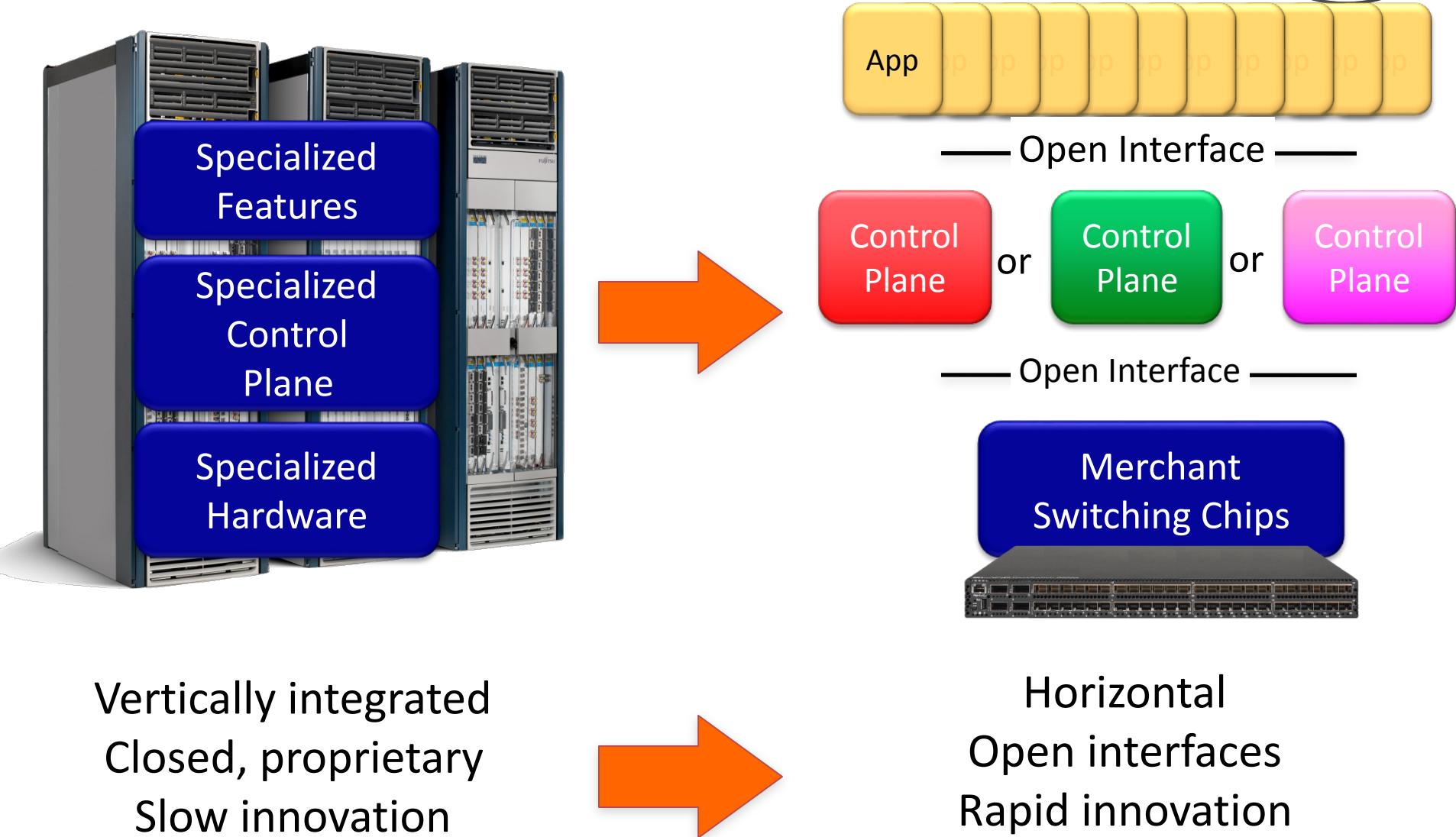


Vertically integrated
Closed, proprietary
Slow innovation
Small industry



Horizontal
Open interfaces
Rapid innovation
Huge industry

Software Defined Networking





Questional Assumptions

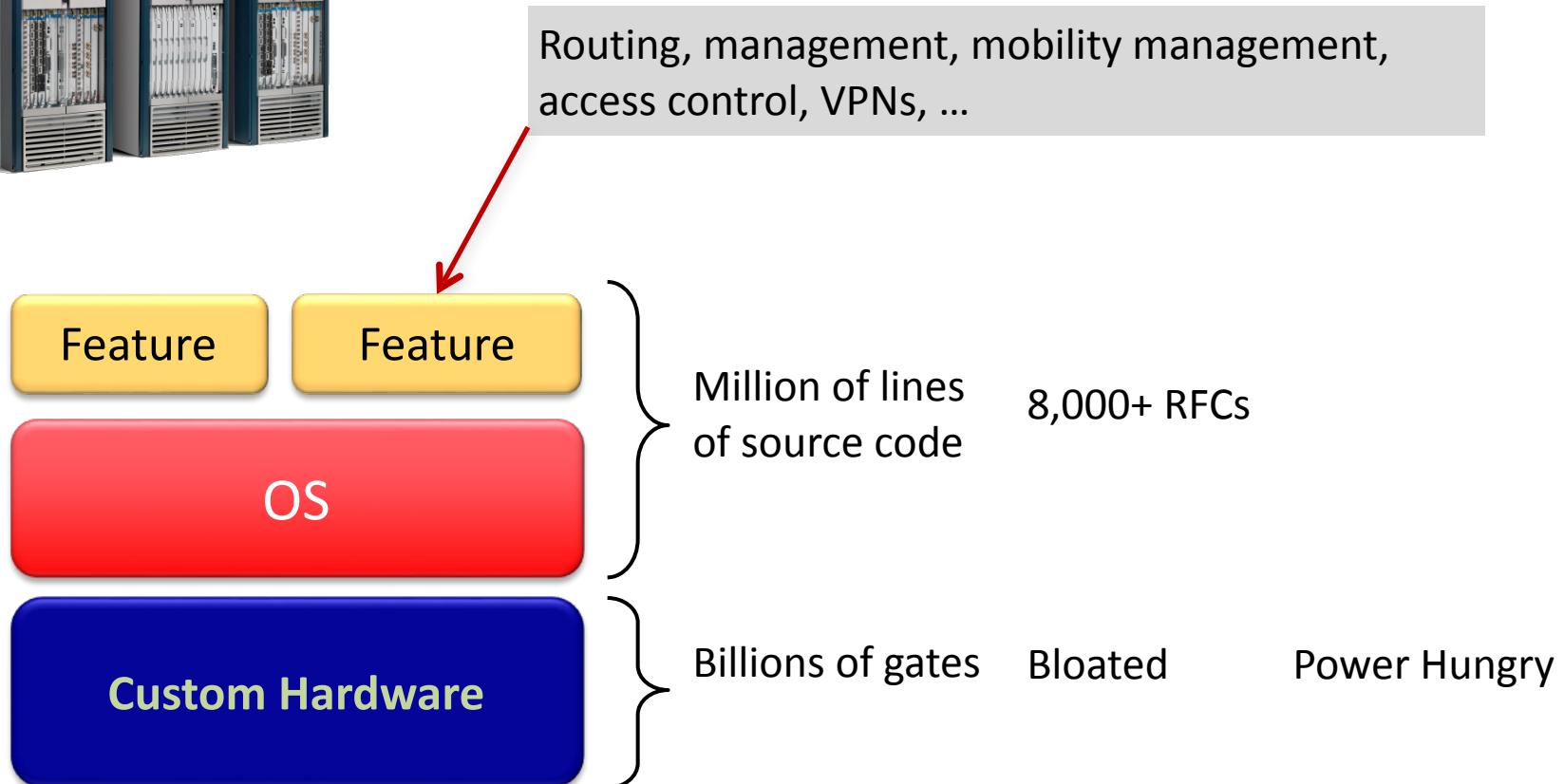
Myth 1: It is hard for switch/router hardware to maintain lots of queues.

- Since 1995, seen 10-15 ASICs do it easily
- Recently: 64x10Gb/s switch, 128k flows, 8% overhead.

Myth 2: You can't build a large flow table

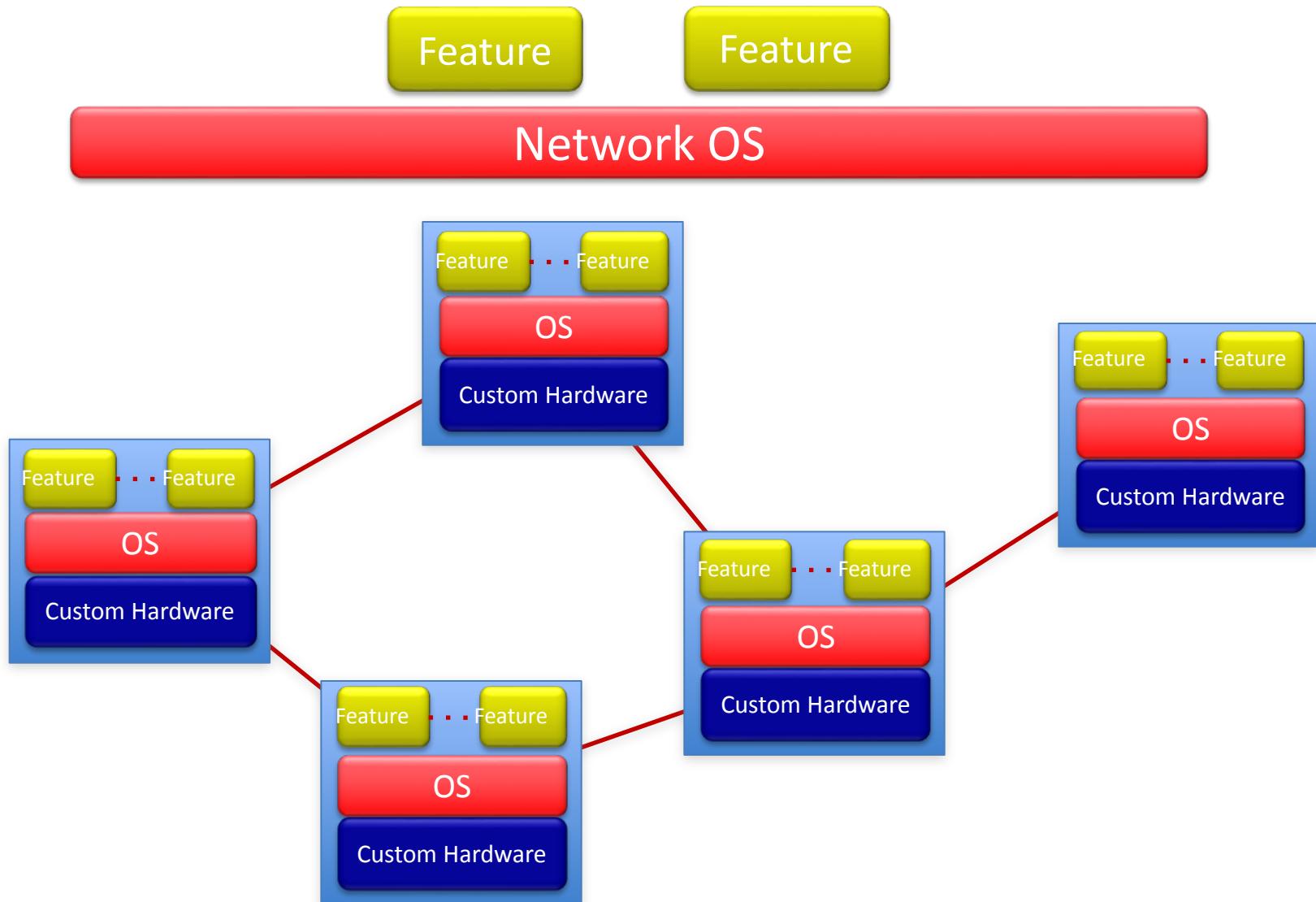
- Next couple of years: Over 1Tb/s, 100,000s entries, 100s bits wide, several tables.

It doesn't mean you have to; just says you can.



- Vertically integrated, complex, closed, proprietary
- Networking industry with “mainframe” mind-set

Network restructuring



Software Defined Networks



3. Well-defined open API

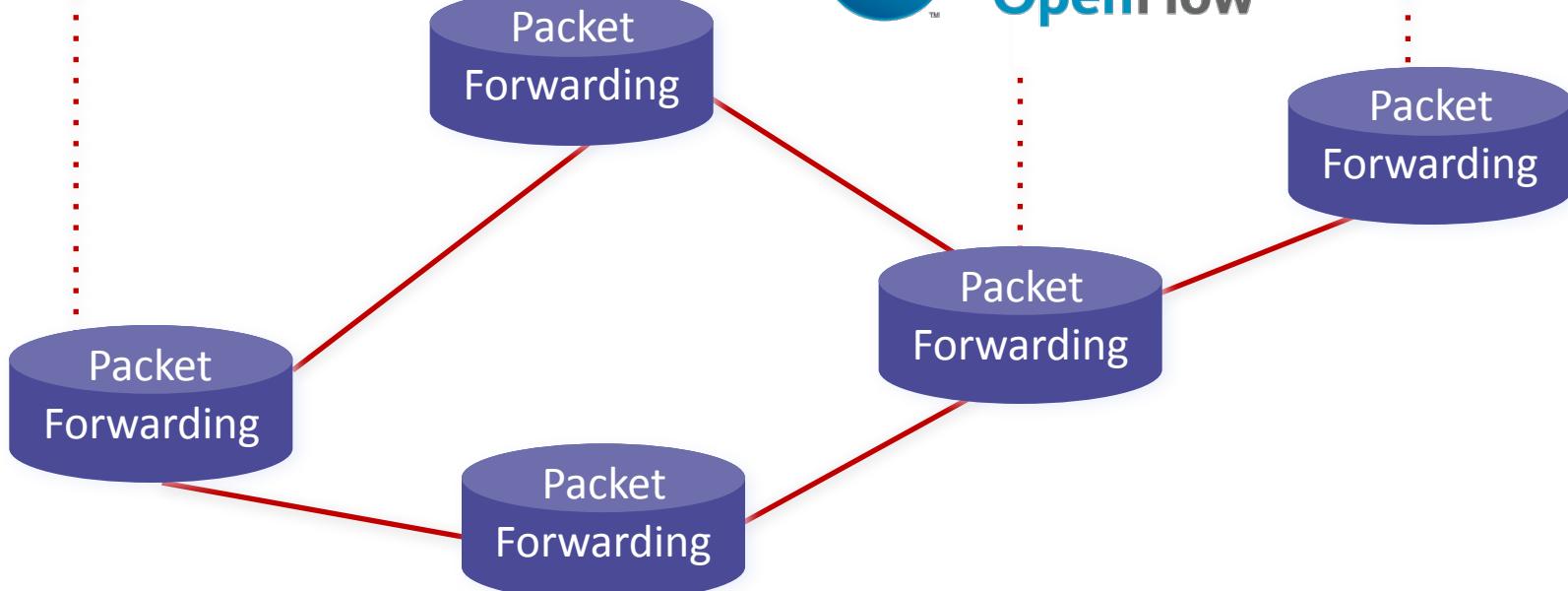
Feature

Feature

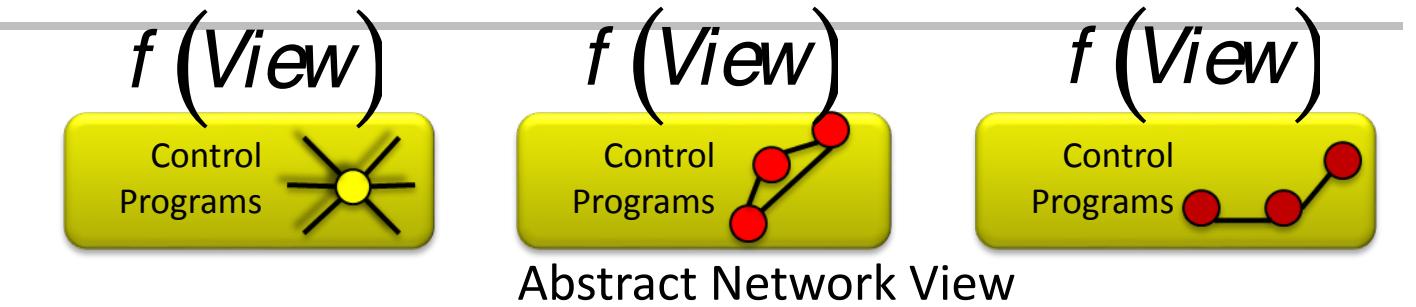


2. At least one Network OS
probably many.
Open- and closed-source

1. Open interface to packet forwarding



Software Defined Network (SDN)

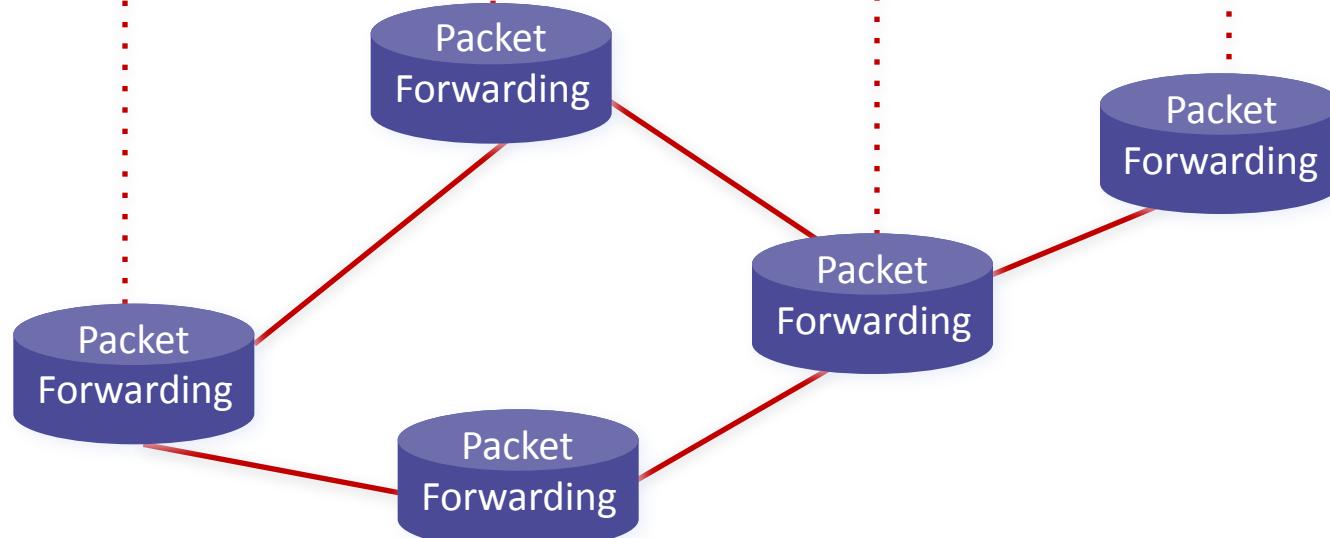


Network Virtualization

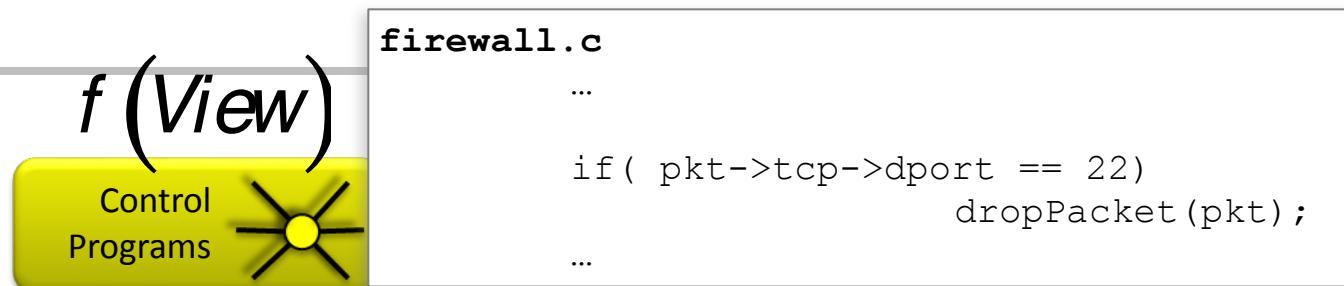
Global Network View



Network OS

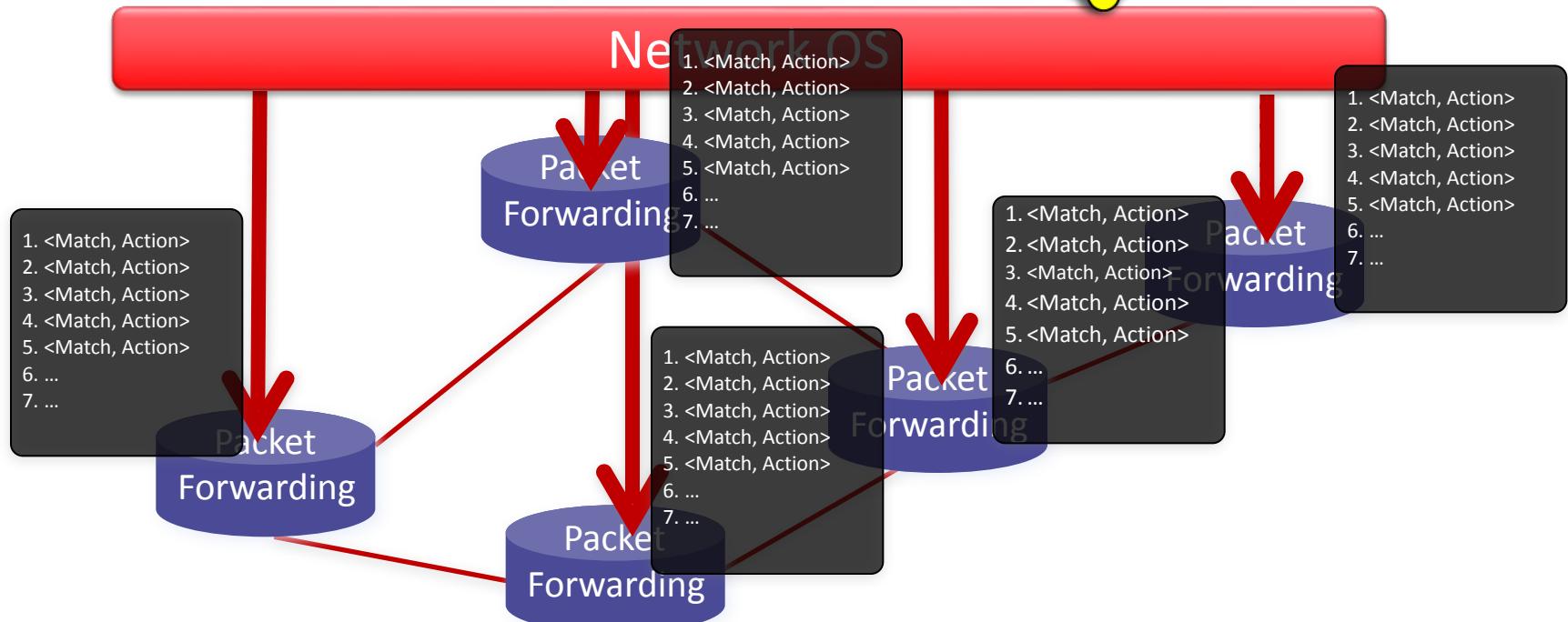


Software Defined Network (SDN)

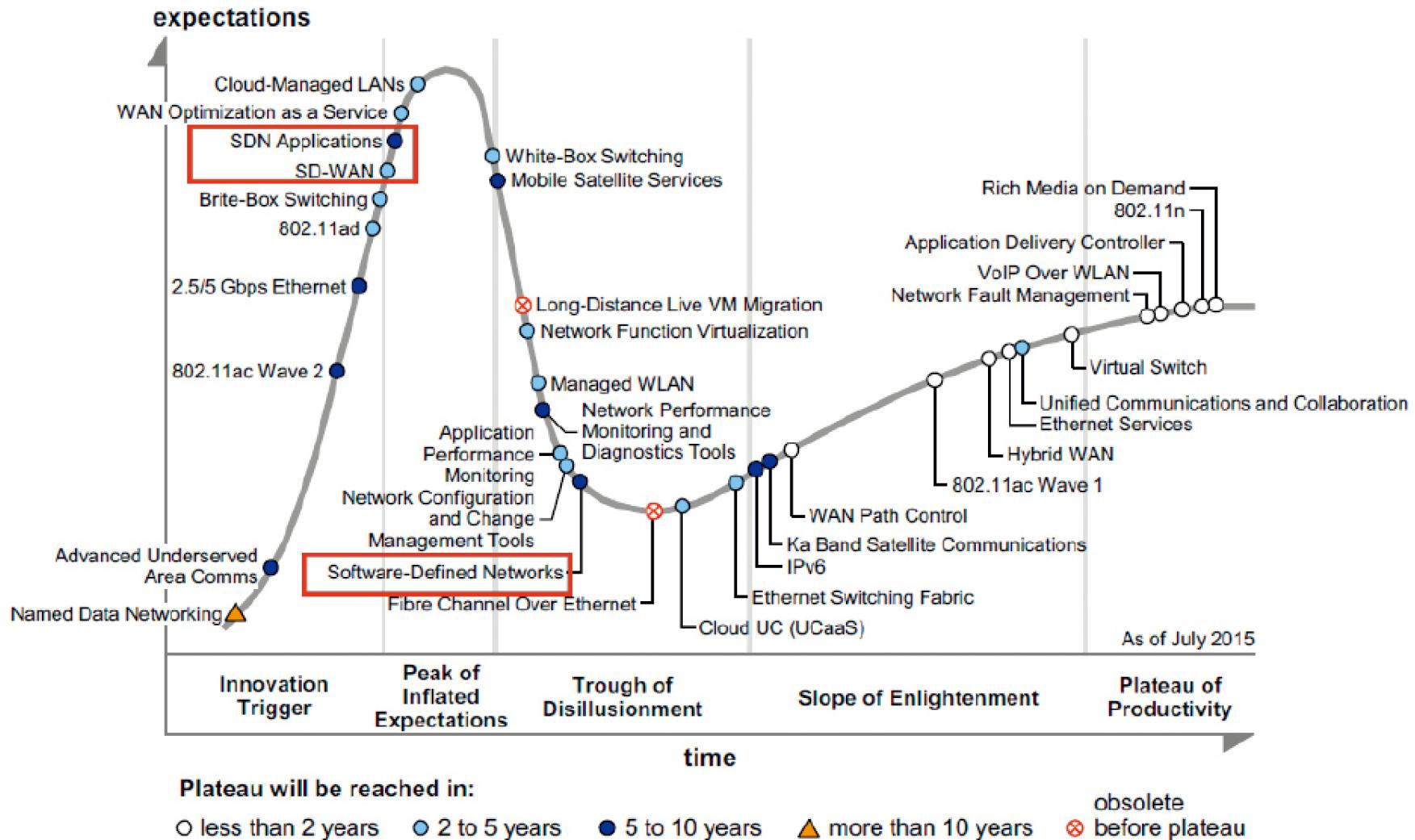


Network Virtualization

Global Network View



当前SDN的发展阶段



Source: Gartner (July 2015)



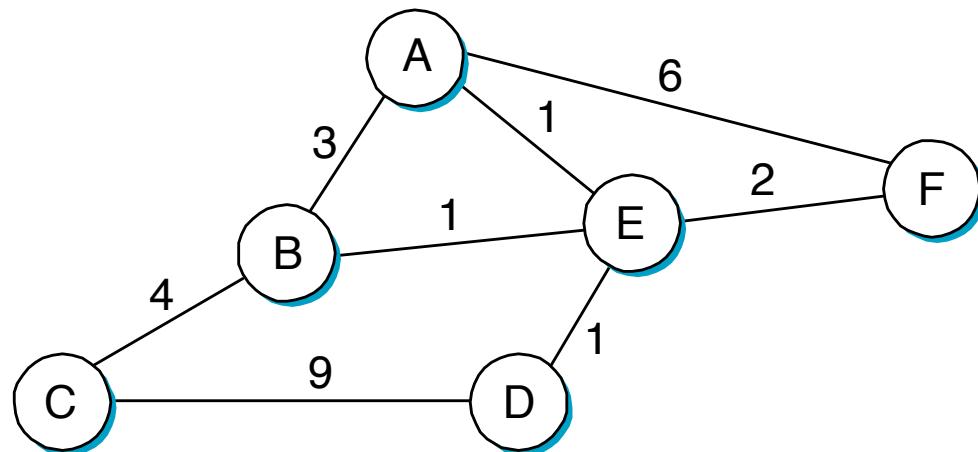
提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 图论
 - 最短路径算法
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结



用图表示网络

- 网络可以用图表示
 - 节点: 路由器
 - 边: 链路
 - 边的代价: 成本, 时延, ...



路由选择本质上是图论中寻找两个节点之间较低成本的路径的问题

图的抽象

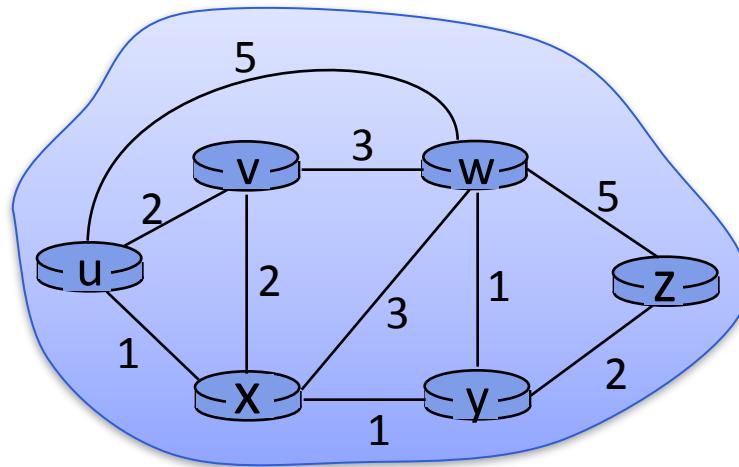
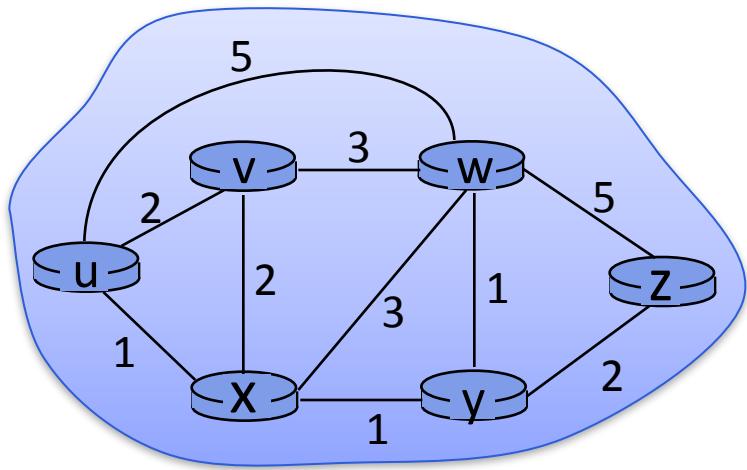


图: $G = (N, E)$

N = 路由器的集合= { u, v, w, x, y, z }

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

图的抽象：代价



- $c(x, x') = \text{链路的代价}(x, x')$
 - 例如, $c(w, z) = 5$
- 代价通常取值为1, 或者与带宽成反比, 或者与拥塞成正比

$$\text{路由的代价}(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$$

问题: 哪一条路径是节点u和z之间的最小代价路径?

解决方案: 路由选择算法, 寻找代价最小的路径



最短路由

- 用广义的距离概念代表边的权值
- 图 $G = (V, E)$, 其权值函数为 $W: E \rightarrow R$ (为每一条边指定一个实数值)
- 路径的权值 $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ is
$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$$
- 最短路由= 拥有最小权值的路径
- 应用
 - 静态/动态网络路由选择
 - 机器人动作规划
 - 交通地图/路径生成



最短路径问题

- **最短路径问题**
 - **Single-source (single-destination).** 寻找从给定源(顶点s)到所有其他顶点的最短路径.
 - **Single-pair.** 寻找两个给定两个顶点之间的最短路径.
single-source问题的解决方案同样适用于single-pair问题的求解.
 - **All-pairs.** 寻找任意两个顶点之间的最短路径. 动态规划算法.



最短路径问题

- 图论中的最短路径问题
 - 双向, 用实数值作为权值
 - 更加普遍的问题
- 两种算法
 - Dijkstra算法
 - 全局交换信息
 - Bellman-Ford算法
 - 顶点对之间交换信息
- 实际上, 网络互联中使用的协议更加简单



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 图论
 - 最短路径算法
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结





路由协议

- 运行于路由器
 - 基于路由选择算法
 - 挑战: 分布式, 动态, ...
- 包括两个部分
 - 拓扑信息分发
 - 路由计算
- 类型
 - 距离向量
 - 链路状态
 - 路径向量



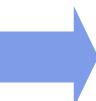
设计选项

- 如何获得拓扑信息
 - 当一个节点启动或与其他节点通过一条新的链路互联时，节点通告其连接信息
 - 节点发现其邻节点或链路
 - 主动：通过控制分组持续探测
 - 被动：意识到最近的若干个更新周期内均未收到预期的定期更新拓扑信息
- 什么时候分发拓扑信息
 - 周期性分发
 - 频率
 - 信息的数量
 - 触发更新
 - 网络拓扑和路由发生变化引起，例如，节点/链路失效，路由变化



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 距离向量路由选择
 - RIP协议
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结





距离向量路由选择

- 基本思想
 - 每个节点构造一个包含到所有其他节点的“距离”(代价)的一维数组(一个向量)
 - 并将该向量分发给其邻节点
 - 节点根据接收到的距离向量计算到达其他所有节点的最短路径
 - 经过距离向量的迭代交换和计算过程最终构造完整的路由表



距离向量路由选择

- 步骤1: 最初节点将向量中可直接到达的邻节点的代价设置为**1**, 到所有其他节点的代销赋值为 **∞**
- 步骤2: 节点将距离向量发送至直接可达的邻节点
- 步骤3: 根据从邻节点**Y**收到的距离向量, 节点**X**
 - 计算本节点到达所有其他节点()的距离: 将来自**Y**的向量中到达所有其他节点的距离(例如**Z**)加上本节点到**Y**的距离(即**1**)
 - 比较计算结果与本地向量中对应结果的大小: 如果计算结果更小, 则用该值替换本地距离向量的对应结果, 并记录**NextHop** 为**Y**

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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$



**node x
table**

		x	y	z
from	x	0	2	7
	y	∞	∞	∞
z	∞	∞	∞	

**node y
table**

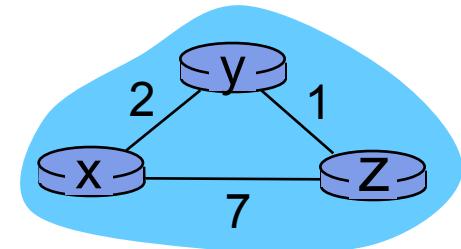
		x	y	z
from	x	∞	∞	∞
	y	2	0	1
z	∞	∞	∞	

**node z
table**

		x	y	z
from	x	∞	∞	∞
	y	∞	∞	∞
z	7	1	0	

cost to

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



time

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

$$= \min\{2+0, 7+1\} = 2$$

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

$$= \min\{2+1, 7+0\} = 3$$

node x table

	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

node y table

	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

cost to

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

cost to

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

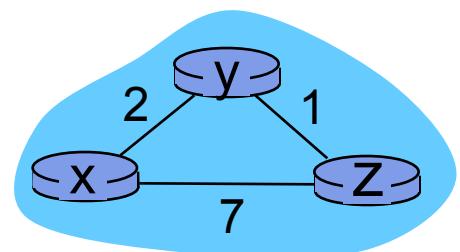
cost to

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

cost to

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

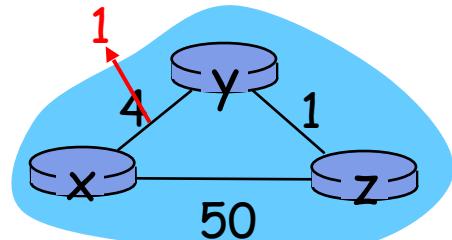
time



Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ updates routing info, recalculates distance vector
- ❖ if DV changes, notify neighbors



“good
news
travels
fast”

t_0 : y detects link-cost change, updates its DV, informs its neighbors.

t_1 : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

t_2 : y receives z's update, updates its distance table. y's least costs do *not* change, so y does *not* send a message to z.

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

$$= \min\{1+0, 50+1\} = 1$$

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

$$= \min\{1+1, 50+0\} = 2$$



**node x
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

**node y
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

**node z
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

	x	y	z
x	0	1	2
y	4	0	1
z	5	1	0

	x	y	z
x	0	4	5
y	1	0	1
z	5	1	0

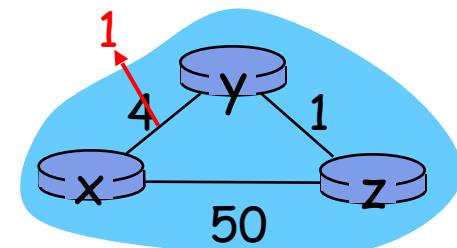
	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

	x	y	z
x	0	1	2
y	1	0	1
z	5	1	0

	x	y	z
x	0	1	2
y	1	0	1
z	5	1	0

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

time



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

$$= \min\{1+0, 50+1\} = 1$$

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

$$= \min\{1+1, 50+0\} = 2$$



**node x
table**

	x	y	z
x	0	1	2
y	1	0	1
z	5	1	0

**node y
table**

	x	y	z
x	0	1	2
y	1	0	1
z	5	1	0

**node z
table**

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

converged

cost to

x y z

x y z

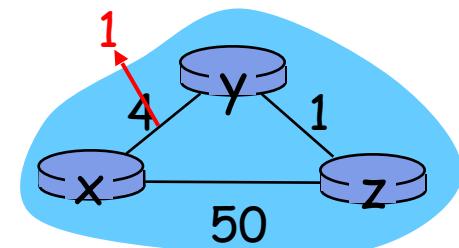
x y z

cost to

x y z

x y z

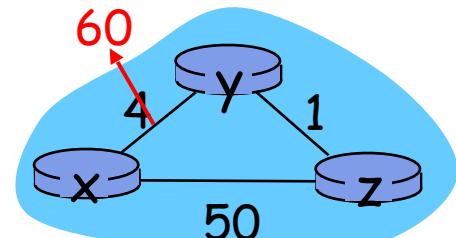
time



Distance vector: link cost changes

link cost changes:

- ❖ node detects local link cost change
- ❖ *bad news travels slow* - “count to infinity” problem!
- ❖ 44 iterations before algorithm stabilizes: see text



poisoned reverse:

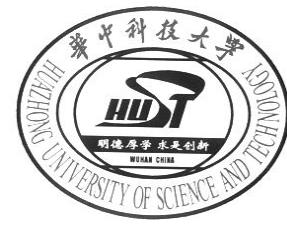
- ❖ If Z routes through Y to get to X :
 - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- ❖ will this completely solve count to infinity problem?

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

$$= \min\{60+0, 50+1\} = 51$$

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

$$= \min\{60+1, 50+0\} = 50$$



**node x
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

**node y
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

**node z
table**

	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

	x	y	z
x	0	51	50
y	4	0	1
z	5	1	0

	x	y	z
x	0	4	5
y	6	0	1
z	5	1	0

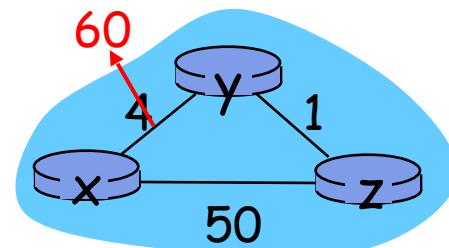
	x	y	z
x	0	4	5
y	4	0	1
z	5	1	0

	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

time

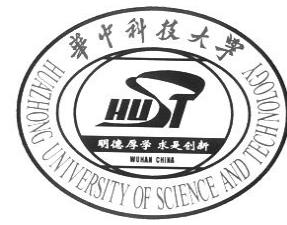


$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$

$$= \min\{60+0, 50+1\} = 51$$

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

$$= \min\{60+1, 50+0\} = 50$$



**node x
table**

	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

**node y
table**

	x	y	z
x	0	51	50
y	6	0	1
z	5	1	0

**node z
table**

	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

	x	y	z
x	0	51	50
y	8	0	1
z	7	1	0

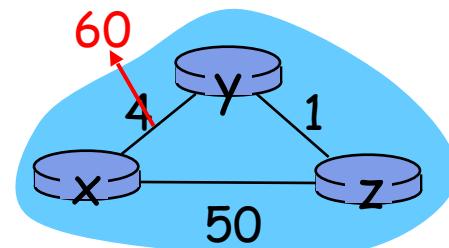
	x	y	z
x	0	51	50
y	6	0	1
z	7	1	0

	x	y	z
x	0	51	50
y	8	0	1
z	7	1	0

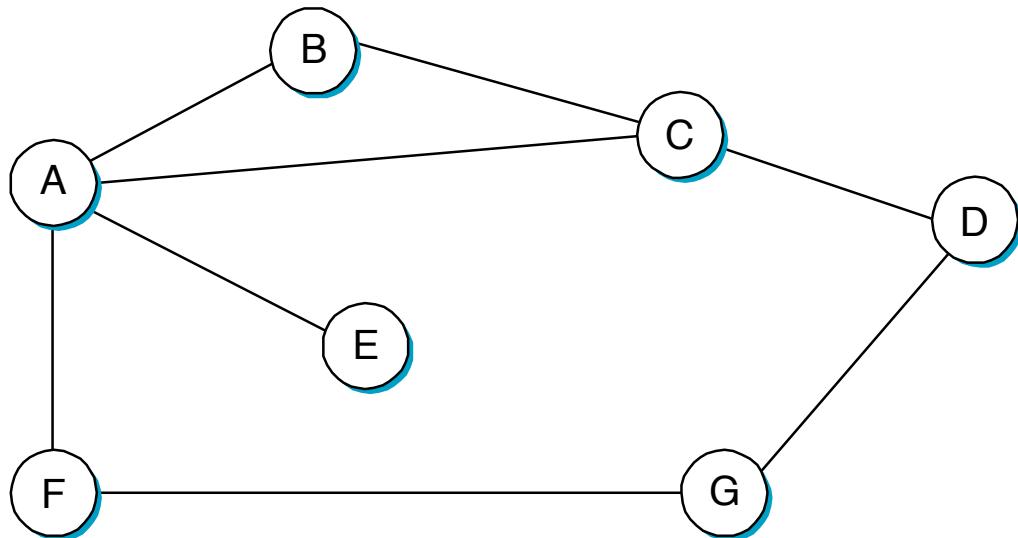
	x	y	z
x	0	51	50
y	8	0	1
z	7	1	0

	x	y	z
x	0	51	50
y	8	0	1
z	9	1	0

time



距离向量路由选择: 示例



dst.	cost	NextHop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

节点A中的初始路由表

- 初始距离向量

A: $(0, 1, 1, \infty, 1, 1, \infty)$

C: $(1, 1, 0, 1, \infty, \infty, \infty)$

F: $(1, \infty, \infty, \infty, \infty, 0, 1)$

B: $(1, 0, 1, \infty, \infty, \infty, \infty)$

E: $(1, \infty, \infty, \infty, 0, \infty, \infty)$



距离向量 - 本地视图

Dest	Cost	Next Hop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

Dest	Cost	Next Hop
A	1	A
C	1	C
D	∞	-
E	∞	-
F	∞	-
G	∞	-

Dest	Cost	Next Hop
A	1	A
B	1	B
D	1	D
E	∞	-
F	∞	-
G	∞	-

Dest	Cost	Next Hop
A	∞	-
B	∞	-
C	1	C
E	∞	-
F	∞	-
G	1	G

Dest	Cost	Next Hop
A	1	A
B	∞	-
C	∞	-
D	∞	-
F	∞	-
G	∞	-

Dest	Cost	Next Hop
A	1	A
B	∞	-
C	∞	-
D	∞	-
E	∞	-
G	1	G

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



距离向量-第一次更新

Dest	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Dest	Cost	Next Hop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	∞	-

Dest	Cost	Next Hop
A	1	A
B	1	B
D	1	D
E	2	A
F	2	A
G	2	G

Dest	Cost	Next Hop
A	2	C
B	2	C
C	1	C
E	∞	-
F	2	G
G	1	G

Dest	Cost	Next Hop
A	1	A
B	2	A
C	2	A
D	∞	-
F	2	A
G	∞	-

Dest	Cost	Next Hop
A	1	A
B	2	A
C	2	A
D	2	G
E	2	E
G	1	G

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



距离向量-第二次更新

Dest	Cost	Next Hop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

Dest	Cost	Next Hop
A	1	A
C	1	C
D	2	C
E	2	A
F	2	A
G	3	C

Dest	Cost	Next Hop
A	1	A
B	1	B
D	1	D
E	2	A
F	2	A
G	2	G

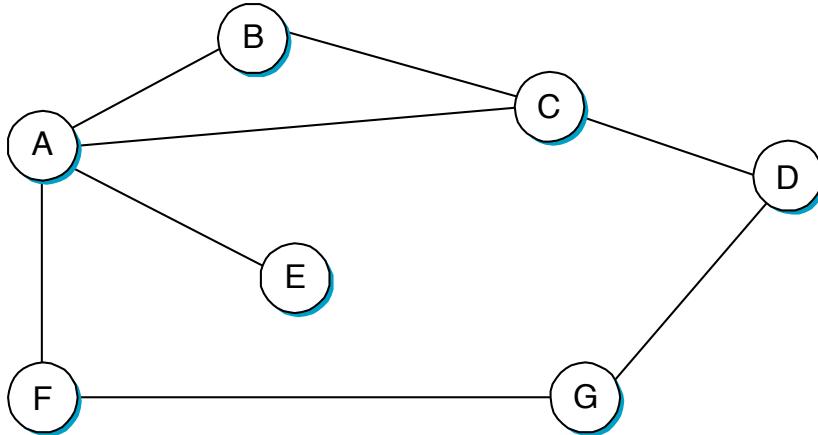
Dest	Cost	Next Hop
A	2	C
B	2	C
C	1	C
E	3	C
F	2	G
G	1	G

Dest	Cost	Next Hop
A	1	A
B	2	A
C	2	A
D	3	A
F	2	A
G	3	A

Dest	Cost	Next Hop
A	1	A
B	2	A
C	2	A
D	2	G
E	2	E
G	1	G

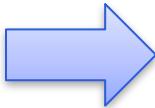
Dest	Cost	Next Hop
A	2	F
B	3	D
C	2	D
D	1	D
E	3	F
F	1	F

示例 (续)



dst.	cost	NextHop
B	1	B
C	1	C
D	∞	-
E	1	E
F	1	F
G	∞	-

节点A中的初始路由表



dst.	cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

节点A中最终的路由表



实际问题

- 如何分发路由更新信息?
- 解决方案1: 周期性更新
 - 及时没有任何变化, 每一个节点总是定期自动发送更新报文.
 - 这使得其他节点知道它仍在正常运行.
 - 可以确保当现有路由不可用时, 他们仍然可以一直得到所需的信息.
 - 更新频率: 几秒到几分钟.

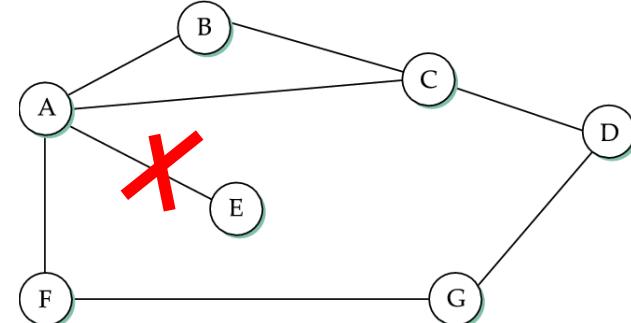


实际问题

- 如何分发路由更新信息?
- 解决方案2: 触发更新
 - 每当一个节点从它的邻节点收到导致更新的报文时
 - 当一个节点的路由表发生变化时将向邻节点发送一条更新报文
 - 可能引起邻节点的路由表变化, 从而使得这些邻节点又向他们的邻节点发送更新报文.

无穷计算问题

- 节点A向邻节点通告其到达节点E的距离为infty
- 节点C通告其通过节点A到达节点E的距离为2
- 节点B通告其通过节点C到达节点E的距离为3
- 节点A更新本地向量，并通告其通过节点B到达E的距离为4
- 节点C更新本地向量，并通告其通过节点A到达E的距离为5...



A	B	C	D	E	
∞	∞	∞	∞	∞	Initially
1	∞	∞	∞	∞	After 1 exchange
1	2	∞	∞	∞	After 2 exchanges
1	2	3	∞	∞	After 3 exchanges
1	2	3	4	∞	After 4 exchanges

(a)

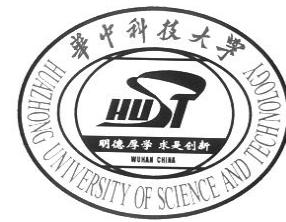
A	B	C	D	E	
1	2	3	4	4	Initially
3	2	3	4	4	After 1 exchange
3	4	3	4	4	After 2 exchanges
5	4	5	4	4	After 3 exchanges
5	6	5	6	6	After 4 exchanges
7	6	7	6	6	After 5 exchanges
7	8	7	8	8	After 6 exchanges
\vdots					
∞	∞	∞	∞	∞	

(b)



无穷计算问题的解决方案

- 方案A: 采用最大跳数取代无穷大
- 方案B: 水平分割
 - 不向提供路由的邻节点发送更新报文.
 - 例如: 节点B不向节点A发送有关节点E的更新信息, 因为节点B到达节点的用路由是从节点学习而来的
- 方案C: 带反向抑制的水平分割
 - 节点B通告到达节点E的代价为无穷大, 以确保节点A最终不会使用节点B来到达节点E
- 上述解决方案在路由循环超过3节点的网络环境中网络会失效



小结：距离向量路由

- 距离向量算法
 - 距离向量: 每个节点仅与直接相连的节点通信但是 包含到达所有节点的距离
 - 基于 Bellman-Ford 算法计算路由结果



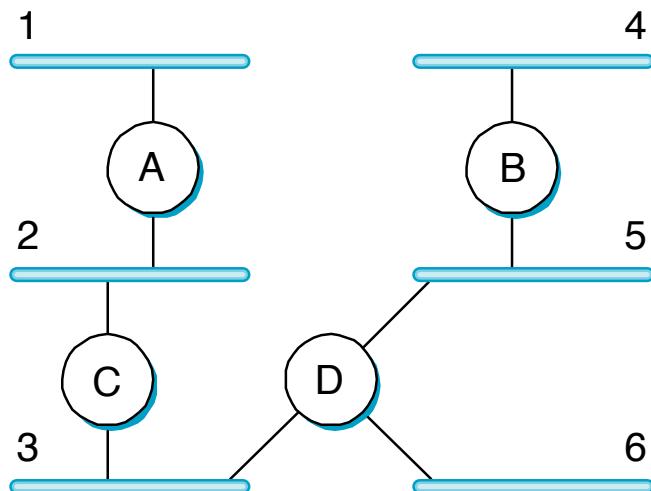
提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 距离向量路由选择
 - RIP协议
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结



路由选择信息协议(RIP)

- 基于距离向量算法的路由选择协议
 - 距离向量算法
 - 1982年跟随BSD-UNIX发布版本一同发布
 - 距离评价指标: # 跳数(最大值 = 15 跳)



路由器通告到达网络的代价, 而不是到达其他路由器的代价



RIP协议报文格式

0	8	16	31
Command	Version	Must be zero	
Family of net 1		Address of net 1	
	Address of net 1		
	Distance to net 1		
Family of net 2		Address of net 2	
	Address of net 2		
	Distance to net 2		



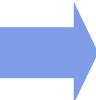
RIP协议的特点

- 版本: 1和2
- 支持多协议族 (不仅仅是IP)
- 拓扑信息分发
 - 每隔30秒
 - 无论何时收到来自其他路由器的引起路由表改变的信息
- 链路的代价(评价指标)
 - 所有链路的代价为1
 - 有效距离取值范围[1, 15], 16意味着无穷大



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 可靠洪泛
 - 路由计算
 - OSPF 协议
 - 路由评价指标
- 实现和性能
- 总结



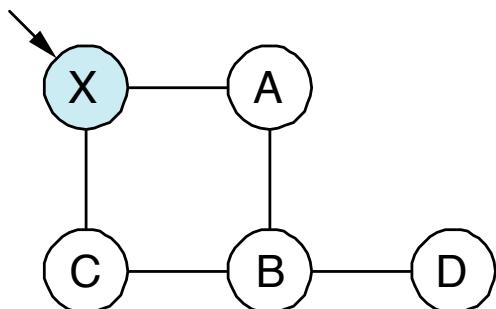


链路状态路由选择

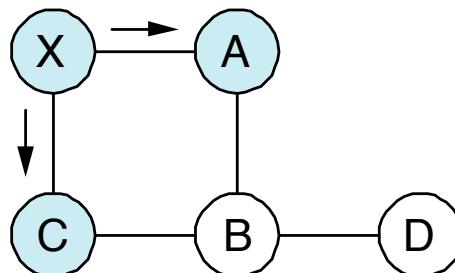
- 基本思想
 - 每个节点都知道怎样到达它的邻节点(link state), 并向通告给其他所有节点, 因此每个节点都可以获得完整的网络信息来建立其路由表
- 依赖于两个机制
 - 链路状态的可靠洪泛(reliable flooding)
 - 根据积累的链路状态知识进行路由计算(route calculation)

洪泛

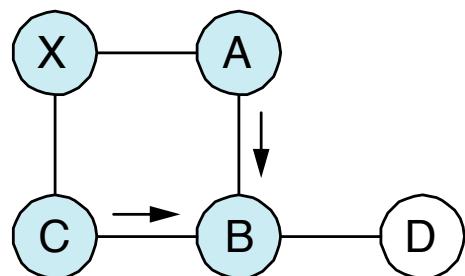
- 洪泛: 通告报文达到所有节点



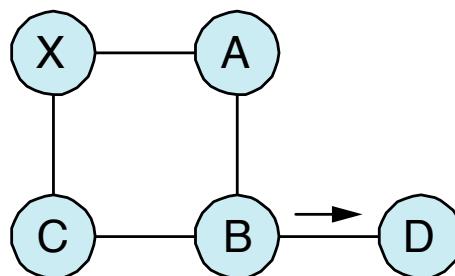
(a)



(b)



(c)



(d)

B可以识别该报文，避免多次转发



可靠洪泛

- 链路状态报文(Link State Message)
 - 链路状态分组, LSP (Link State Packet)
 - 创建LSP的节点ID
 - 与该节点直接相邻的节点信息列表 <AdjacentNode, Cost>, 其中包括到这些邻节点的链路代价
 - 序号 Seq
 - LSP的生命周期
- 如何保证链路状态分组的洪泛
 - 采用确认和重传机制
 - 通过序号
 - 不发回发送LSP的节点



可靠洪泛(续)

- 新的链路状态分组快速洪泛, 旧的分组快速被删除
 - 采用序列号和TTL
 - 长序列号
 - 逐跳递减TTL
- 最小化网络洪泛过程中的链路状态分组的数量
 - 使用很长时间(通常在几个小时)的定时器周期性地生成
 - 按需触发



链路状态洪泛的触发器

- 与RIP协议类似，两种情况下产生新的LSP
 - 周期性计时器超时
 - 拓扑结构变化
- 拓扑变化检测
 - 新的节点/链路：通过链路状态报文通告
 - 节点/链路失效
 - 通过链路层协议
 - 周期性交换“hello”报文



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 可靠洪泛
 - 路由计算
 - OSPF 协议
 - 路由评价指标
- 实现和性能
- 总结





路由计算

- 基于Dijkstra 算法

- 标识

s : 执行算法的节点

N : 网络中的节点集合

$I(i, j)$: 节点 $i, j \in N$ 之间链路的代价

M : 当前参与算法运算的节点集合

~~$C(n)$: 从 s 到 n 的收敛代价~~

$M = \{s\}$

for each n in $N - \{s\}$

$C(n) = I(s, n)$

while ($M \neq N$)

find out w such that $C(w)$ is the minimum for all nodes $w \in (N - M)$

$M = M \cup \{w\}$

for each $n \in (N - M)$

$C(n) = MIN (C(n), C(w) + I(w, n))$



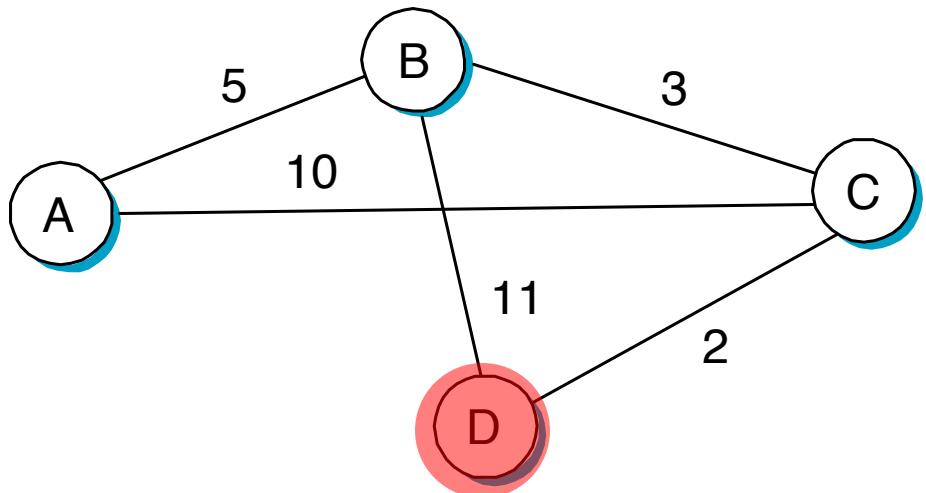
Dijkstra 算法的实现: 前向搜索

- 每个路由器维护两个列表, 称为, 试探表(Tentative) 和证实表(Confirmed)
- 每张表中有多条记录, 记录包含(Destination, Cost, NextHop)
- 最终证实表即期望的结果 - 形成节点的路由表
- 步骤(详见下一页)

1. 用记录(**Myself**, 0, -)初始化证实表
2. 前一步加入证实表的节点称为**Next节点**, 选择其LSP
3. **For** (**Next节点**的每一个**邻节点**)
到达该**邻节点**的代价 = 从自身到**Next节点**的代价+ 从**Next节点**到**邻节点**的代价
 - **If** **邻节点**当前既不在证实表中也不在试探表中, **then** 将记录(**Neighbor**, **Cost**, **NextHop**) 加入试探表中, 其中 **NextHop** 为自身到达**Next节点**所经的节点
 - **If** **邻节点**在试探表中, 且**代价**小于当前表中记录的代价, **then** 用记录(**Neighbor**, **Cost**, **NextHop**)替换当前记录, 其中 **NextHop** 为自身到达**Next节点**所经的节点
4. **If** 试探表为空, 则**停止**; **otherwise**, 从试探表中挑选最小代价的记录, 将其移入确认表, 并**转回执行步骤2**

Dijkstra 算法: 示例(1)

路由器D

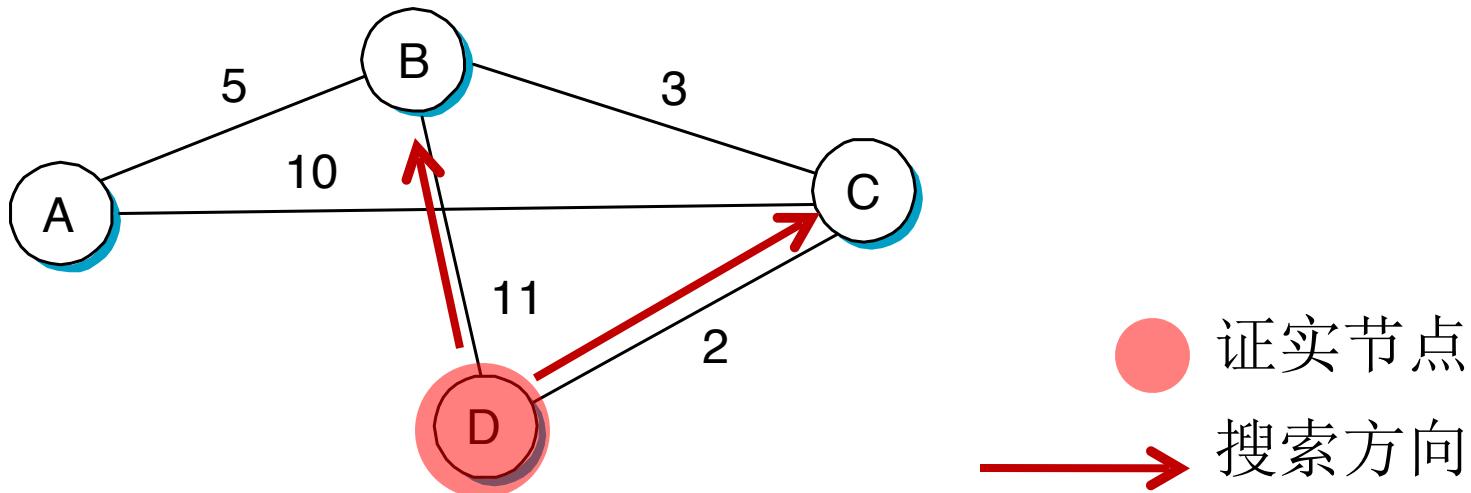


证实节点
搜索方向

<i>Step</i>	1	2	3	4	5	6	7
证实表	(D, 0, -)						
试探表							

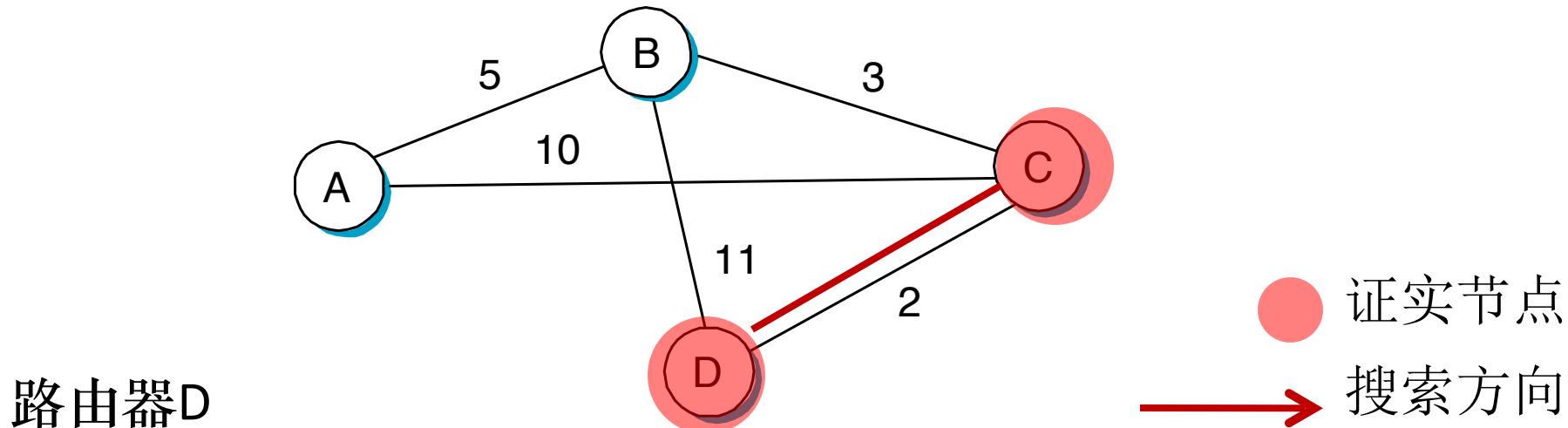
Dijkstra 算法: 示例(1)

路由器D



<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$					
试探表		$(B, 11, B)$ $(C, 2, C)$					

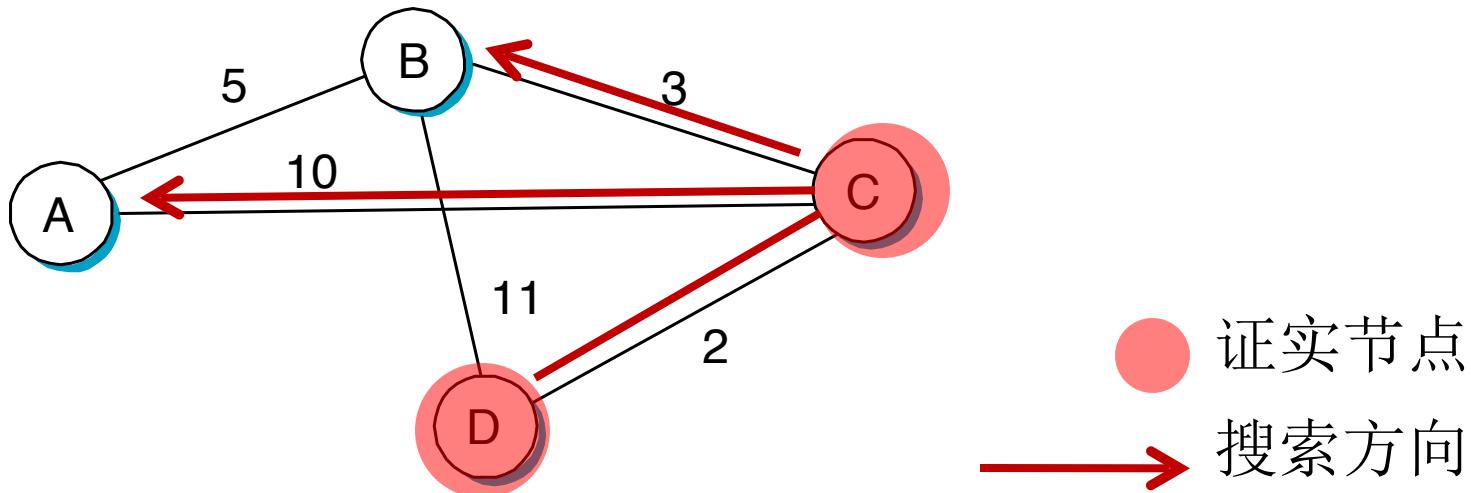
Dijkstra 算法实现：示例



<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$	$(D, 0, -)$ <u>$(C, 2, C)$</u>				
试探表		$(B, 11, B)$ <u>$(C, 2, C)$</u>	$(B, 11, B)$				

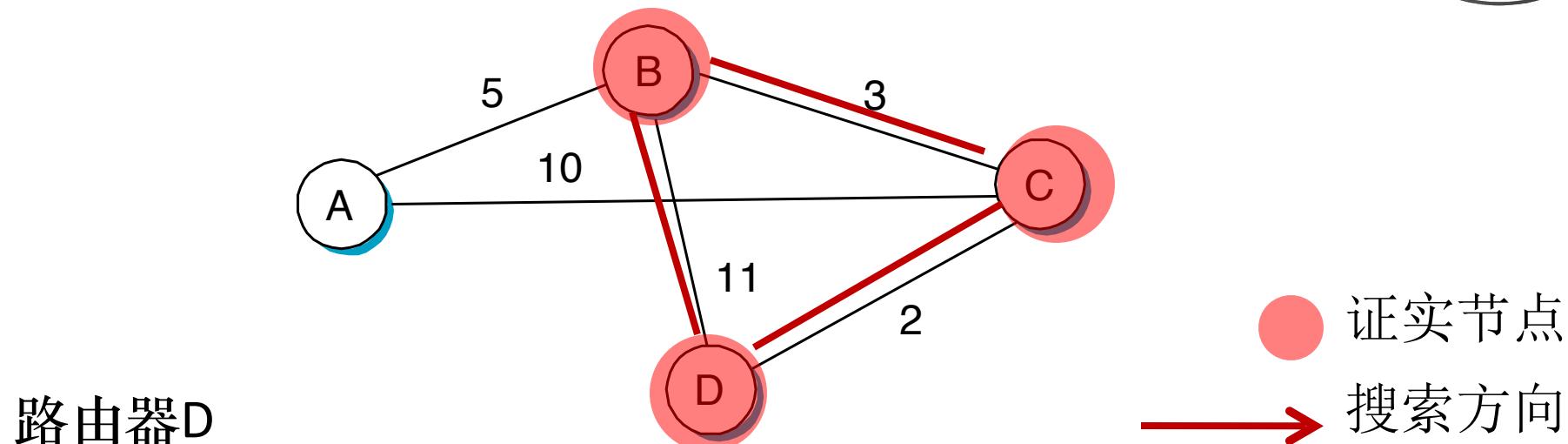
Dijkstra 算法: 示例(1)

路由器D



<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$	$(D, 0, -)$ <u>$(C, 2, C)$</u>	$(D, 0, -)$ $(C, 2, C)$			
试探表		$(B, 11, B)$ $(C, 2, C)$	$(B, 11, B)$	$(B, 11, B)$ $(B, 5, C)$ $(A, 12, C)$			

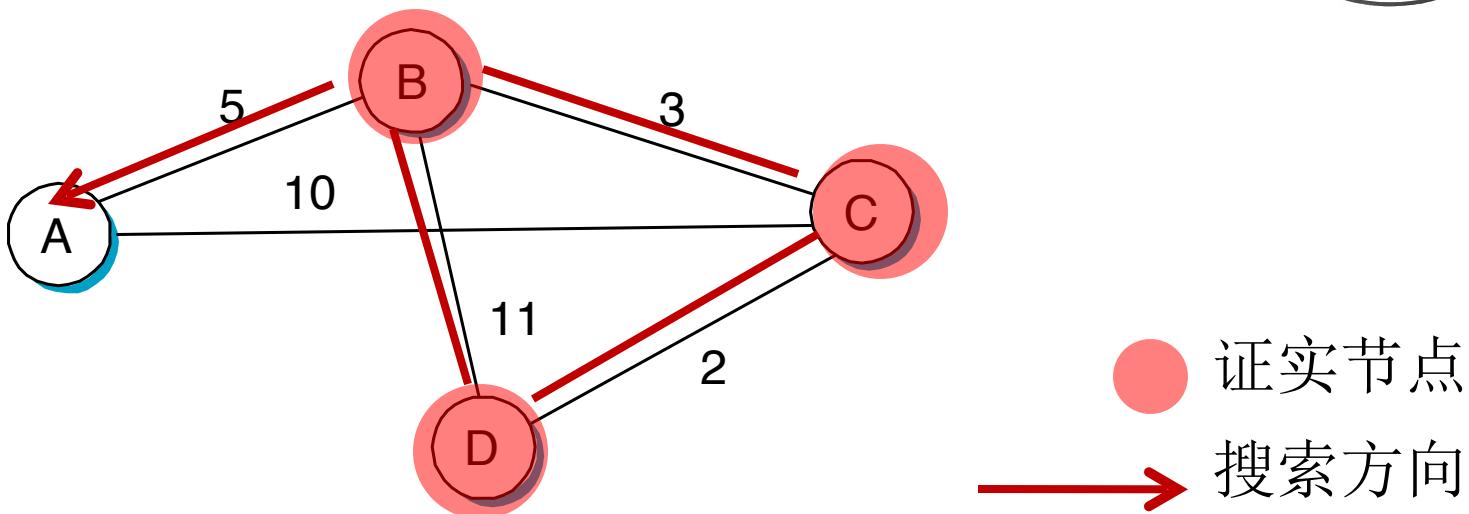
Dijkstra 算法实现：示例



<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$	$(D, 0, -)$ <u>$(C, 2, C)$</u>	$(D, 0, -)$ $(C, 2, C)$	$(D, 0, -)$ $(C, 2, C)$ <u>$(B, 5, C)$</u>		
试探表		$(B, 11, B)$ $(C, 2, C)$	$(B, 11, B)$	$(B, 5, C)$ $(A, 12, C)$	$(A, 12, C)$		

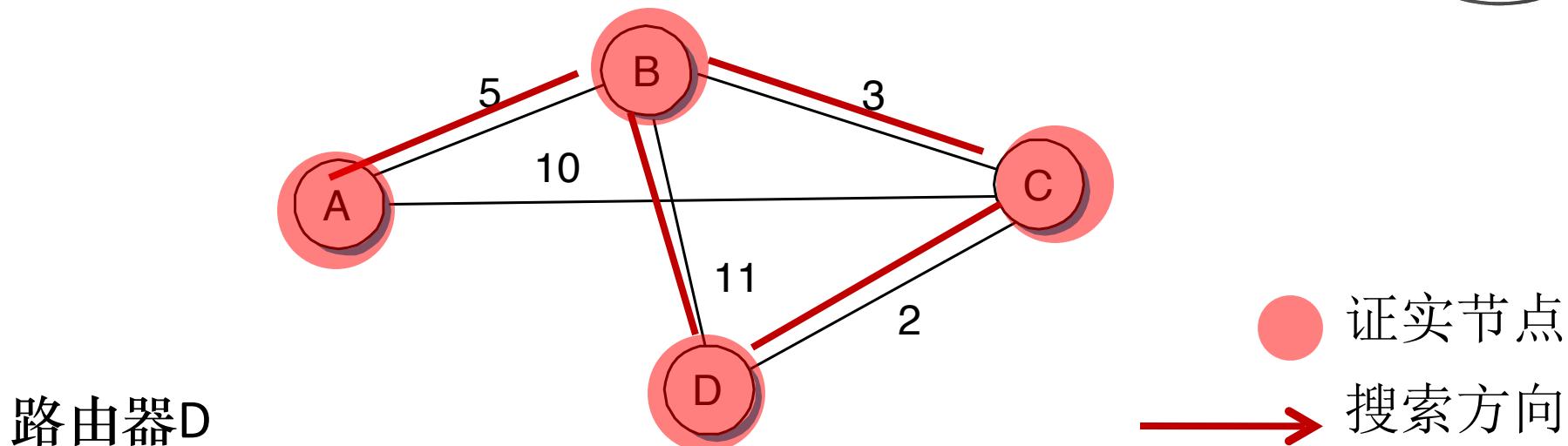
Dijkstra 算法: 示例(1)

路由器D



<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$	$(D, 0, -)$ <u>$(C, 2, C)$</u>	$(D, 0, -)$ $(C, 2, C)$	$(D, 0, -)$ $(C, 2, C)$ <u>$(B, 5, C)$</u>	$(D, 0, -)$ $(C, 2, C)$ $(B, 5, C)$	
试探表		$(B, 11, B)$ $(C, 2, C)$	$(B, 11, B)$	$(B, 5, C)$ $(A, 12, C)$	$(A, 12, C)$	$(A, 12, C)$ $(A, 10, C)$	

Dijkstra 算法: 示例(1)



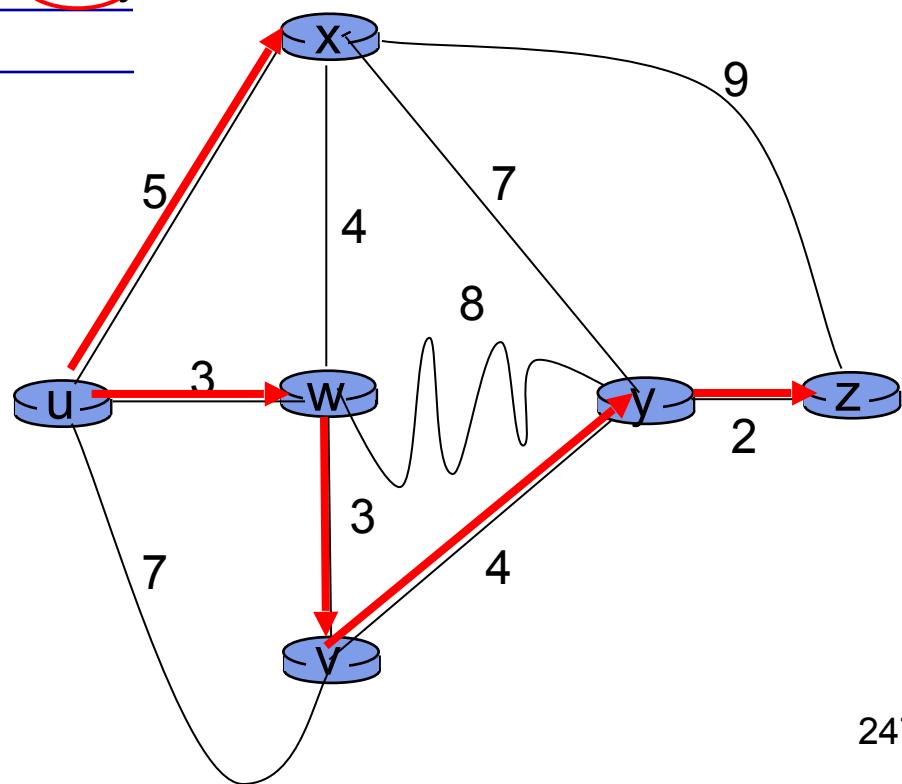
<i>Step</i>	1	2	3	4	5	6	7
证实表	$(D, 0, -)$	$(D, 0, -)$	$(D, 0, -)$ <u>$(C, 2, C)$</u>	$(D, 0, -)$ $(C, 2, C)$	$(D, 0, -)$ $(C, 2, C)$ <u>$(B, 5, C)$</u>	$(D, 0, -)$ $(C, 2, C)$ $(B, 5, C)$	$(D, 0, -)$ $(C, 2, C)$ $(B, 5, C)$ <u>$(A, 10, C)$</u>
试探表			$(B, 11, B)$ $(C, 2, C)$	$(B, 11, B)$	$(B, 5, C)$ $(A, 12, C)$	$(A, 12, C)$	$(A, 10, C)$ Done!

Dijkstra 算法: 示例(2)

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					

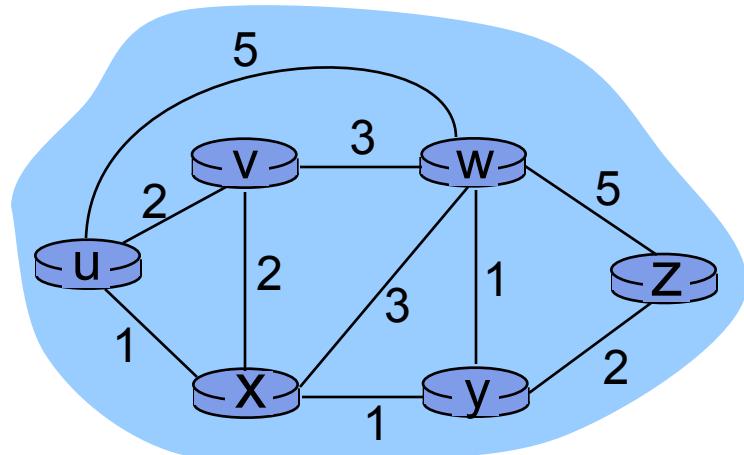
notes:

- ❖ construct shortest path tree by tracing predecessor nodes
- ❖ ties can exist (can be broken arbitrarily)



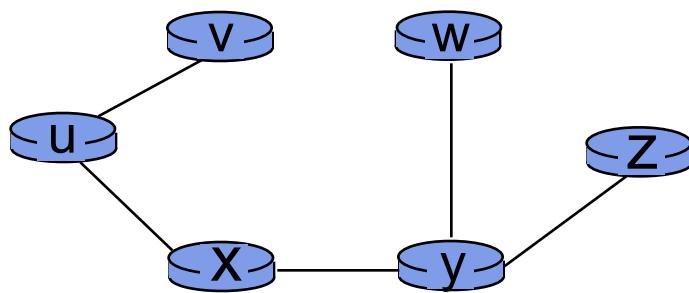
Dijkstra 算法: 示例(3)

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					



Dijkstra 算法: 示例(4)

resulting shortest-path tree from u:



resulting forwarding table in u:

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

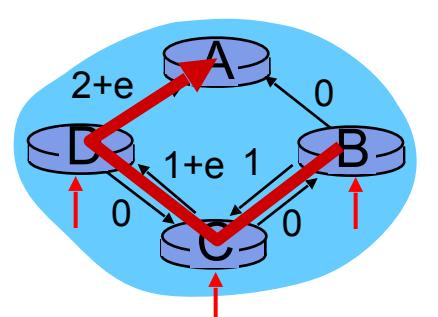
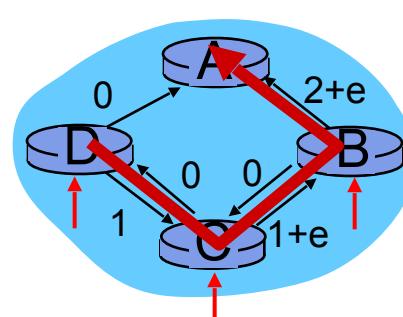
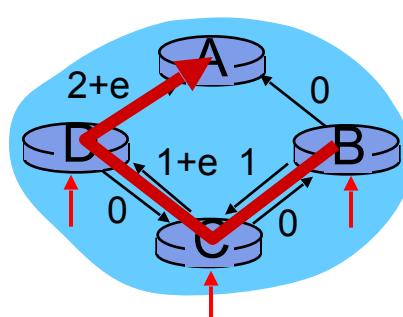
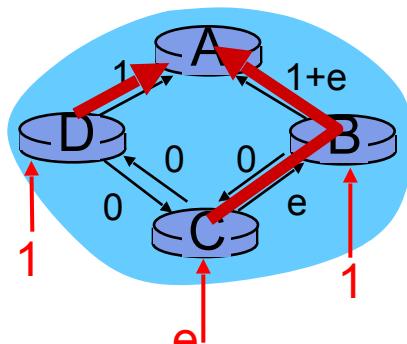
Dijkstra 算法: 讨论

algorithm complexity: n nodes

- ❖ each iteration: need to check all nodes, w, not in N
- ❖ $n(n+1)/2$ comparisons: $O(n^2)$
- ❖ more efficient implementations possible: $O(n \log n)$

oscillations possible:

- ❖ e.g., support link cost equals amount of carried traffic:





链路状态路由选择小结

- 优点
 - 快速稳定
 - 不会产生过多的通信量
 - 快速响应拓扑变化
- 缺点
 - 节点存储的信息量大(所有其他节点的LSP)
- 链路状态算法与距离向量算法的区别
 - 链路状态: 每个节点仅告诉所有其他节点与之直接相连的链路状态
 - 距离向量: 每个节点仅与直接相连的节点通信但是 包含到达所有节点的距离



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 可靠洪泛
 - 路由计算
 - OSPF 协议
 - 路由评价指标
- 实现和性能
- 总结





OSPF (开放最短路径优先)

- “开放”: 公开的, 非专有的, 由IETF主持创建
- 采用链路状态算法
 - LS分发
 - 每个节点构建网络拓扑
 - 采用Dijkstra算法进行路由计算
- 在OSPF通告中, 每一个邻接路由器对应一条记录



OSPF

- 报文类型
 - hello, request, send, 以及 acknowledge
- 链路状态通告 (LSA)
 - 链路状态报文的基本构件
 - 一个链路状态报文可能包含多个LSAs

0	8	16	31
Version	Type	Message length	
SourceAddr			
AreaId			
Checksum	Authentication type		
Authentication			

OSPF首部格式

LS Age	Options	Type=1
Link-state ID		
Advertising router		
LS sequence number		
LS checksum	Length	
0	Flags	0
Number of links		
Link ID		
Link data		
Link type	Num_TOS	Metric
Optional TOS information		
More links		

OSPF 链路状态通告.



OSPF协议的“高级”特征(RIP协议不具备的)

- **安全性**: 所有OSPF报文需要认证 (避免恶意入侵)
- 允许多条相同代价的路径存在 (RIP中仅允许一条路径)
- 对一条链路而言, 可以依据不同的**TOS**分配不同的代价值(例如, 卫星链路从实时性考虑代价较大, 但从尽最大努力交付能力来说代价较小)
- 支持单播和**多播**:
 - 多播OSPF (MOSPF)使用与OSPF相同的拓扑数据
- 在一个较大的域内采用**层次化**OSPF.



小结：链路状态路由

- 链路状态路由
 - 链路状态: 每个节点仅告诉所有其他节点与之直接相连的链路状态
 - 距离向量: 每个节点仅与直接相连的节点通信但是 包含到达所有节点的距离



距离向量路由选择

- 如何构造本地拓扑信息?
 - 构造距离向量, 如 $(0, 1, 1, \infty, 1, 1, \infty)$
 - 实例: RIP路由报文格式, 最大值为16
- 如何全局交换本地信息?
 - 通过邻节点递归交换向量
 - 好消息传递快, 坏消息传递慢
- 如何计算最短路径?
 - 计算到达所有其它节点的最短路径
 - Bellman-Ford 算法
- 如何更新拓扑信息变化?
 - 周期性更新和触发更新
 - 实例: RIP协议每隔30 sec更新一次



链路状态路由选择

- 如何构造本地拓扑信息?
 - 构造关于邻节点的链路状态报文
 - 实例: OSPF LSA 中记录 $\langle \text{AdjacentNode}, \text{Cost} \rangle$ 列表
- 如何全局交换本地信息?
 - 可靠洪泛报文至所有节点
 - 可靠保证: ACK, 重传, 序列号, TTL, ...
 - 采用Seq和TTL字段, 尽可能快速最新的LS报文
- 如何计算最短路径?
 - 计算到达所有其它节点的最短路径
 - Dijkstra 算法
- 如何更新拓扑信息变化?
 - 周期性更新和触发更新



小结: 路由选择

全局信息还是本地信息?

全局:

- 所有路由器拥有完整的拓扑和链路代价信息
- “链路状态” 算法

本地:

- 路由器知道物理上连接的邻节点以及到达邻节点的链路代价
- 与邻节点交换信息, 进行迭代计算
- “距离向量”算法

静态还是动态?

静态:

- 路由变化缓慢

动态:

- 路由快速变化
 - 周期性更新
 - 链路代价发生改变

Comparison of LS and DV algorithms

message complexity

- **LS:** with n nodes, E links, $O(nE)$ msgs sent
- **DV:** exchange between neighbors only
 - convergence time varies

speed of convergence

- **LS:** $O(n^2)$ algorithm requires $O(nE)$ msgs
 - may have oscillations
- **DV:** convergence time varies
 - may be routing loops
 - count-to-infinity problem

robustness: what happens if router malfunctions?

LS:

- node can advertise incorrect *link* cost
- each node computes only its own table

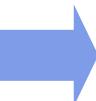
DV:

- DV node can advertise incorrect *path* cost
- each node's table used by others
 - error propagate thru network



提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- 总结





路由评价指标

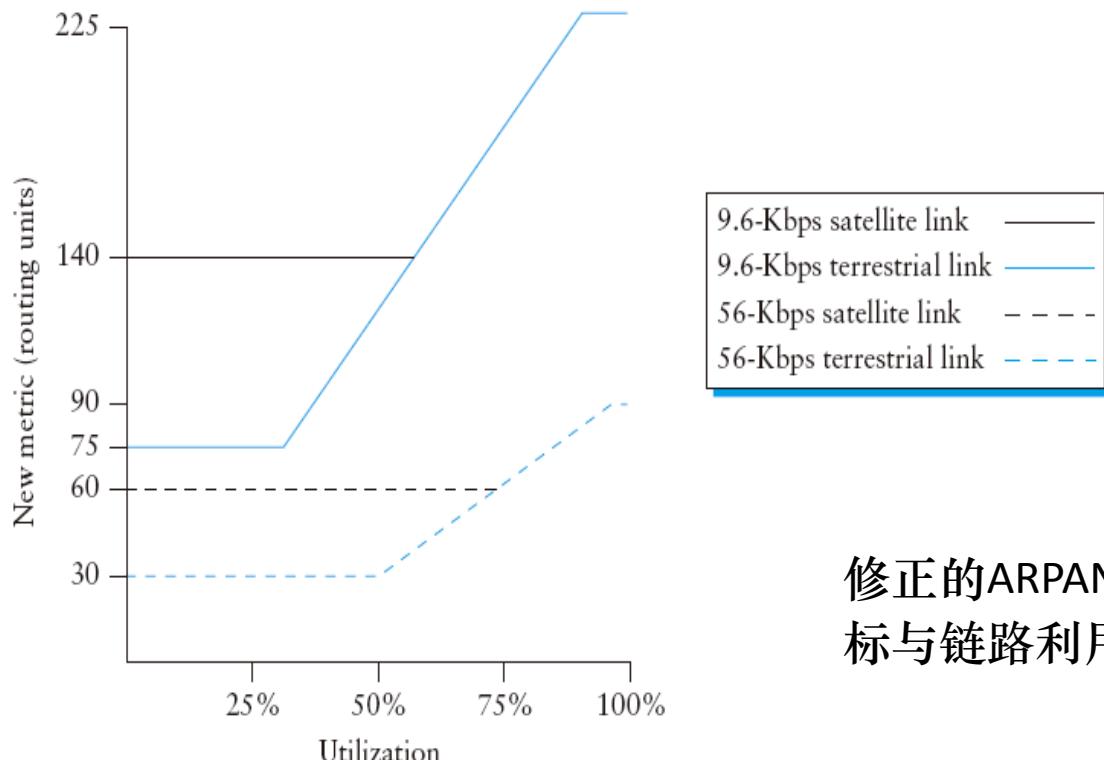
- 代价指标
 - 最早的ARPANET路由评价指标
 - 每条链路上排队等待发送的分组数量
 - 无法反映链路带宽和路径时延
 - 然而, 出现了不同类型的链路
 - 56kbps → 230kbps → 1.544Mbps
- 解决方案
 - 版本2: 新的路由选择机制, 时延

$$\text{Delay} = (\text{DepartTime} - \text{ArrivalTime}) + \text{TransmissionTime} + \text{Latency}$$

- 产生多跳路由, 浪费网络资源

路由评价指标

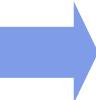
- 解决方案
 - 版本3: 修正的ARPANET路由评价指标, 链路利用率
 - 缩减了评价指标的动态范围, 说明链路类型, 并且减缓了评价指标随时间的变化





提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
 - 交换基础
 - 端口
 - 交换结构
 - 路由器实现
- 总结



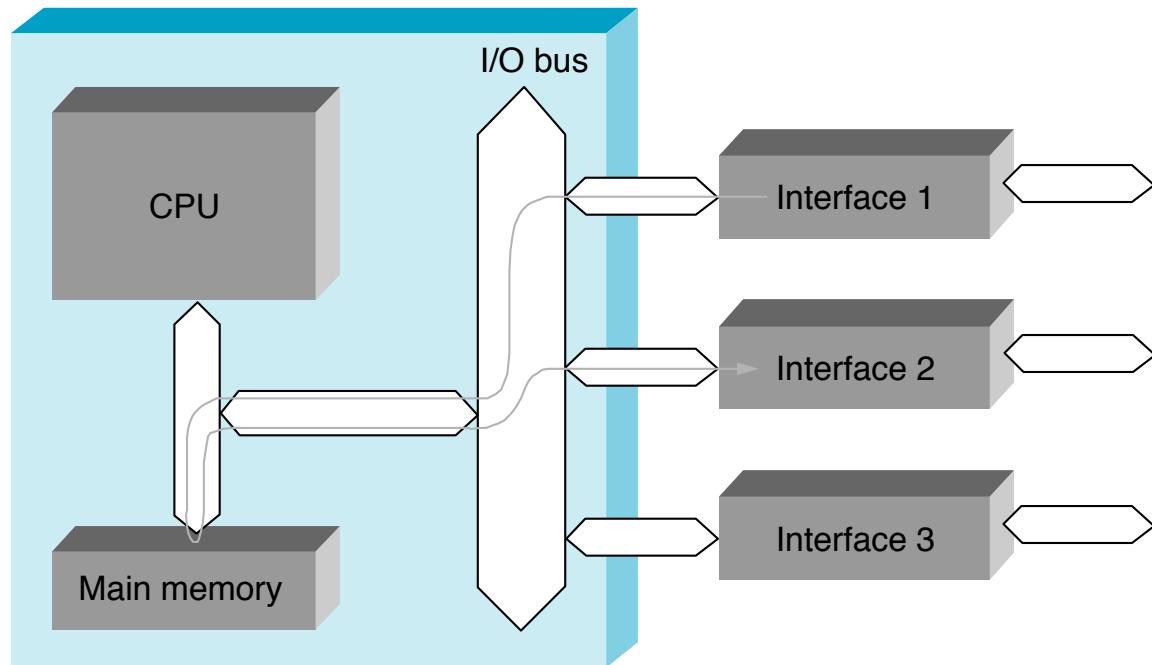


分组交换网的交换机

- 交换机
 - 多输入多输出设备
 - 主要任务: 在接口之间进行分组转发
- 性能上的考虑
 - 尽可能快速转发分组
 - 隐含着协议的设计
- 设计方案
 - 基于通用的计算机
 - 基于专用硬件

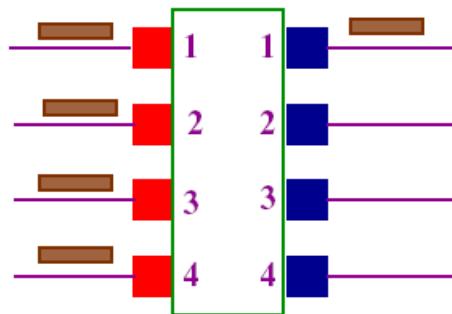
交换机设计:基于通用计算机

- 局限性:
 - 每一个分组必须经过I/O总线两次
 - 适配器到内存(写入)
 - 内存到适配器(读取)

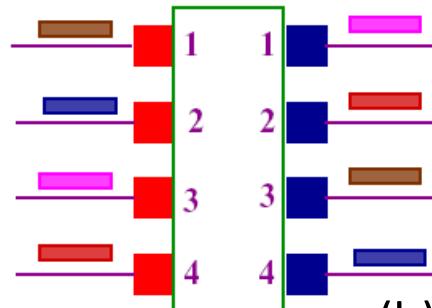


吞吐量

- 定义
 - 交换机每秒可以转发的分组/比特数量
- 最优吞吐量
 - 所有输出链路带宽之和
- 影响吞吐量的因素
 - 通信量模式
 - 分组大小



(a) 示例1



(b) 示例 2

- 注意: 通信量模式很难预测, 目前没有一个非常精准的通信量模型



吞吐量: 现行的定义

- 定义1
 - 单位时间内交换机转发分组数量
 - 单位: 每秒的分组数 (packet per second, pps)
- 定义2
 - 所有接口的输出链路带宽之和
 - 测量每秒的比特数 (bit per second, bps)
- 单位
 - pps
 - bps



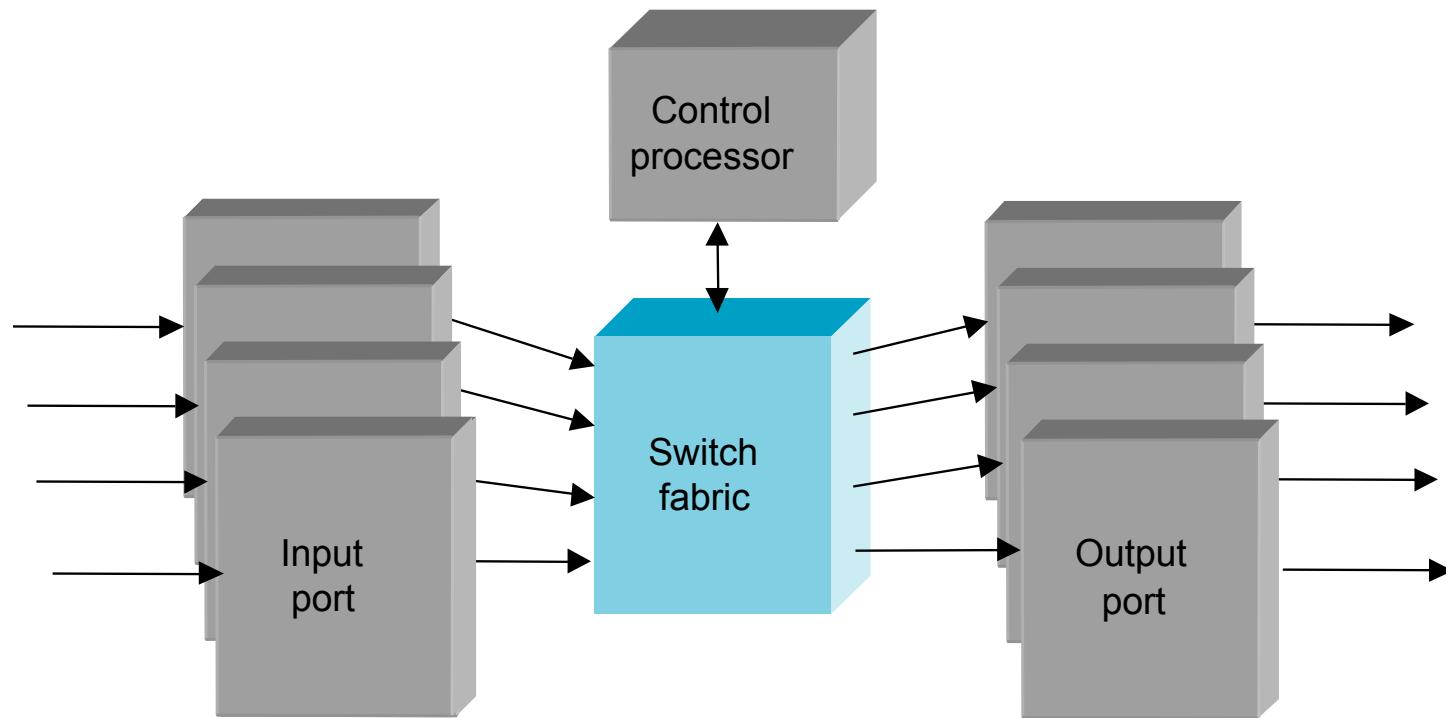
提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
 - 交换基础
 - 端口
 - 交换结构
 - 路由器实现
- 总结

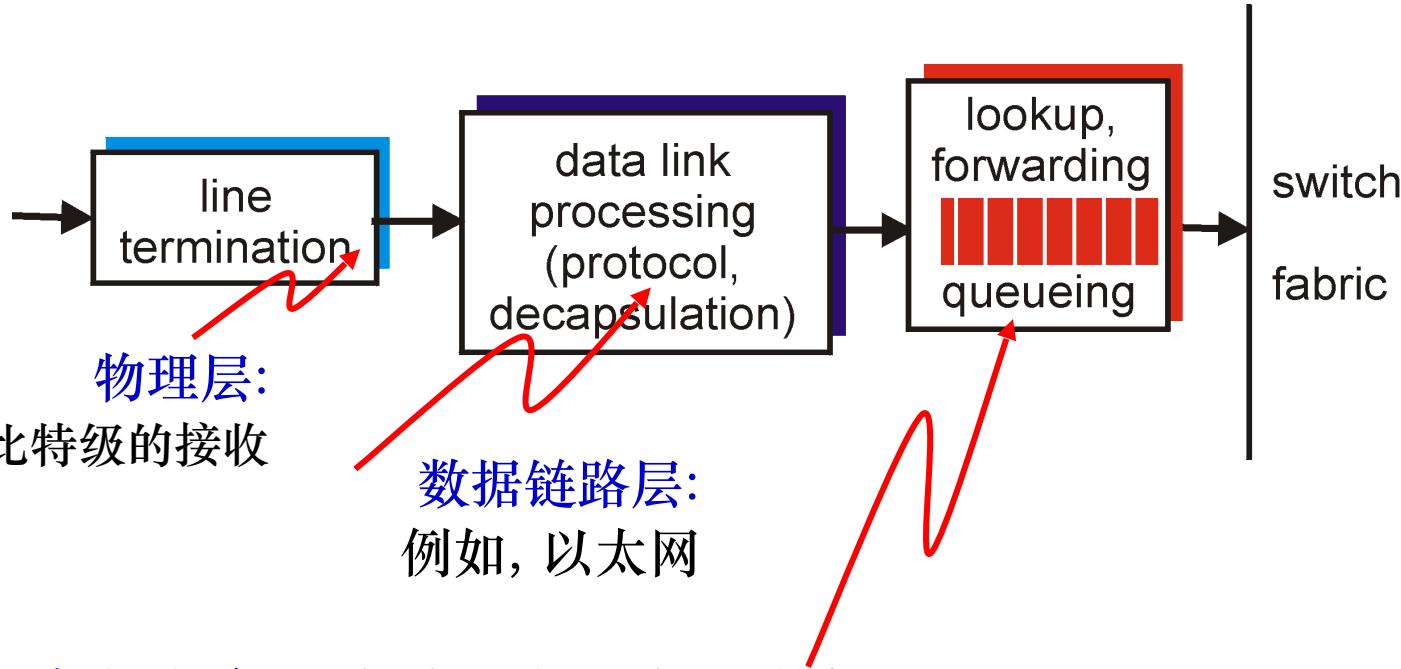


交换机设计:基于专用硬件

- 从概念上讲, 交换机包含
 - 若干输入端口
 - 若干输出端口
 - 交换结构
 - 中央处理器: 协议报文处理, 接口控制



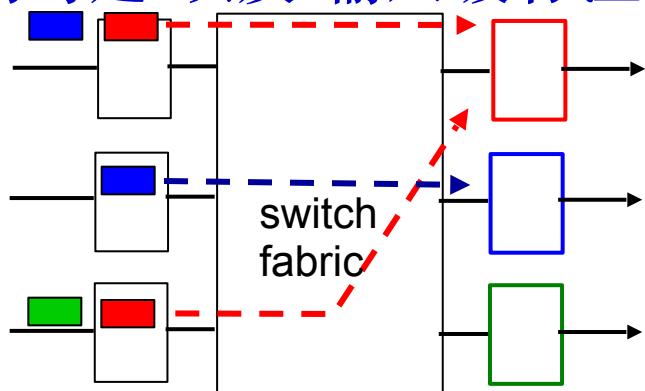
输入端口



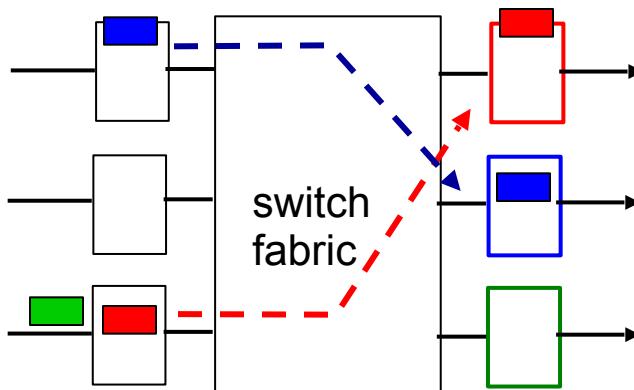
- 分布式交换(每一个端口有一个线路卡):
 - 给定数据报的目的地址, 查询输入端口缓存的转发表中对应的输出端口
 - 目标: ‘线速’完成输入端口处理
 - 队列: 如果报文到达速率快于交换结构的转发速率

输入端口队列

- 交换结构处理能力慢于所有输入端口的分组到达速度 ->缓存输入端口的分组
- **队列头 (HOL) 阻塞**: 滞留在输入缓冲区前面的分组阻碍缓冲区后面的分组的传送过程
- **队列时延 以及 输入缓存溢出导致分组丢失!**

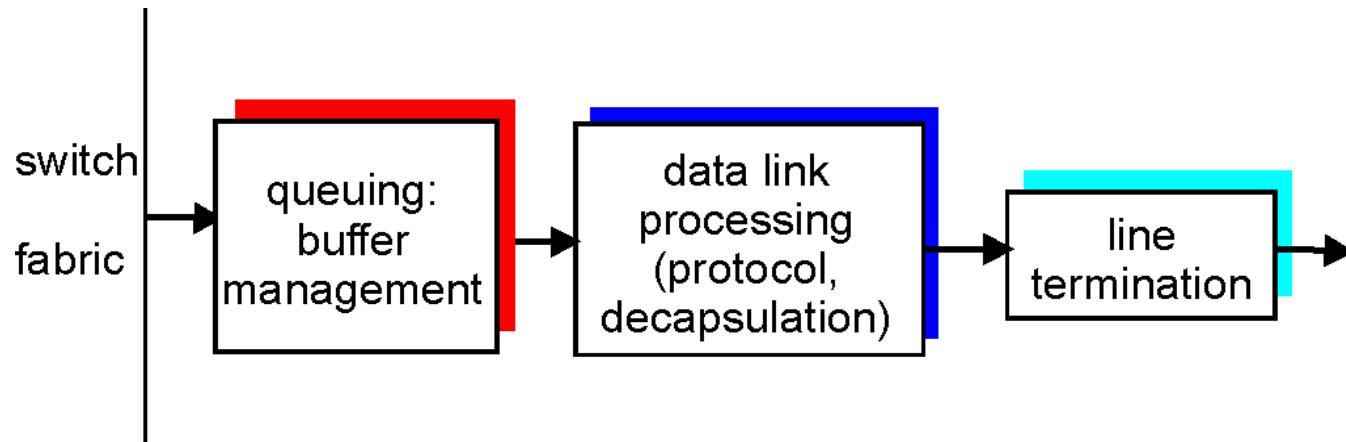


output port contention:
only one red datagram can be
transferred.
lower red packet is blocked



one packet time later:
green packet
experiences HOL
blocking

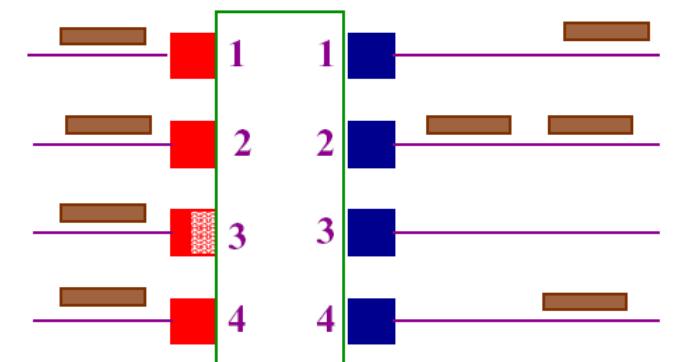
输出端口



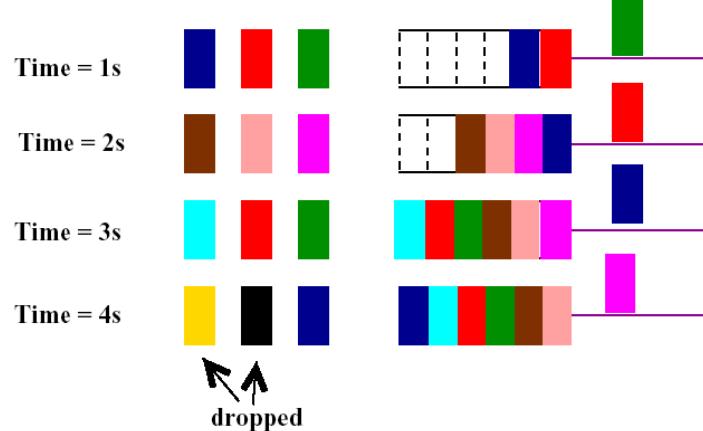
- 当报文到达速率(来自交换结构)快于传输速率, 则需要对报文进行缓存
- 调度策略 选择队列中的数据报进行传输

输出缓存

- 当多个分组同时到达某一输出端口，则进入缓冲区队列
- 当分组到达时缓冲区已满，则丢弃分组
 - 称为分组丢失



(a) 到达速率>输出速率:
输出缓冲区队列



(b) 输出缓冲区已满:
分组丢失

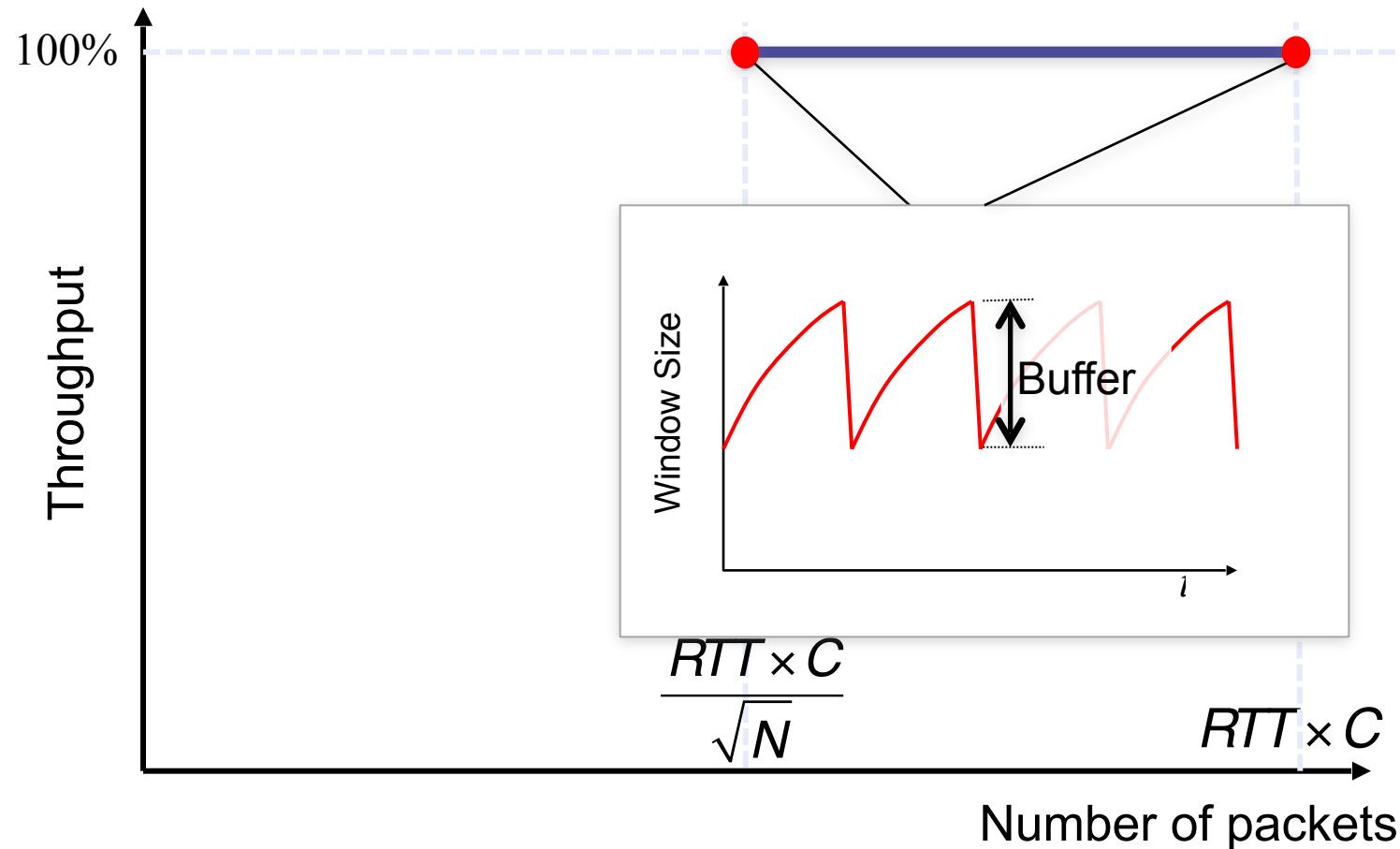


How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
 - e.g., $C = 10 \text{ Gpbs}$ link: 2.5 Gbit buffer
- recent recommendation: with N flows, buffering equal to

$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$

Buffer Size

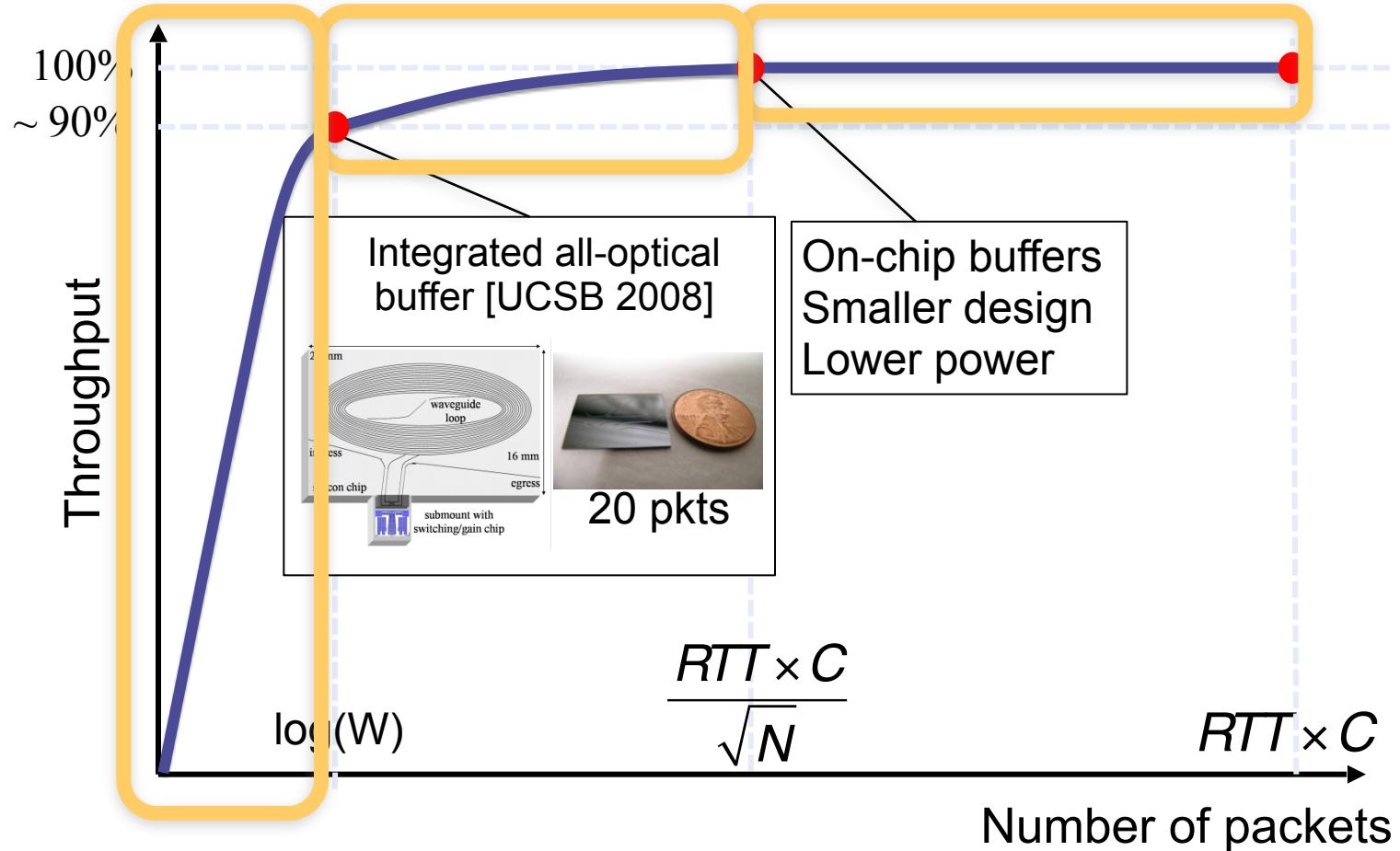


10Gb/s WAN

25,000

2,500,000

Buffer Size



10Gb/s WAN ~50

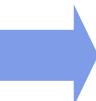
25,000

2,500,000

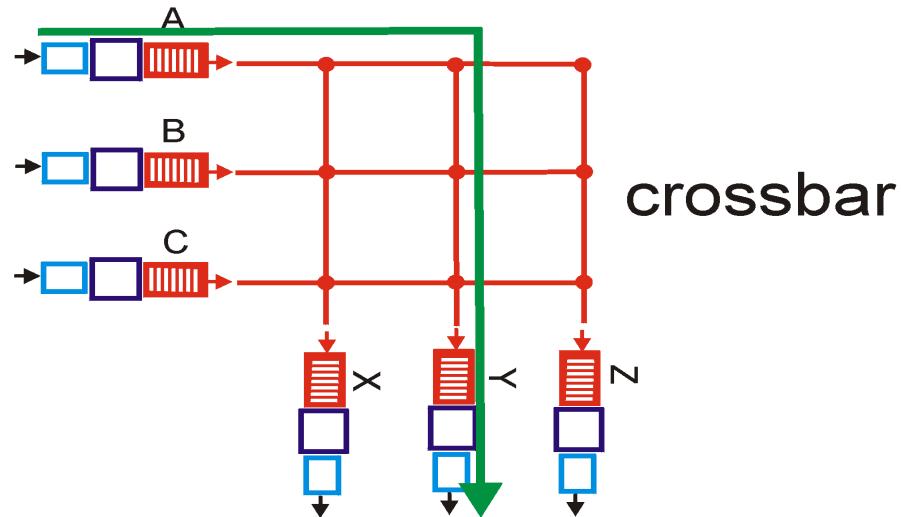
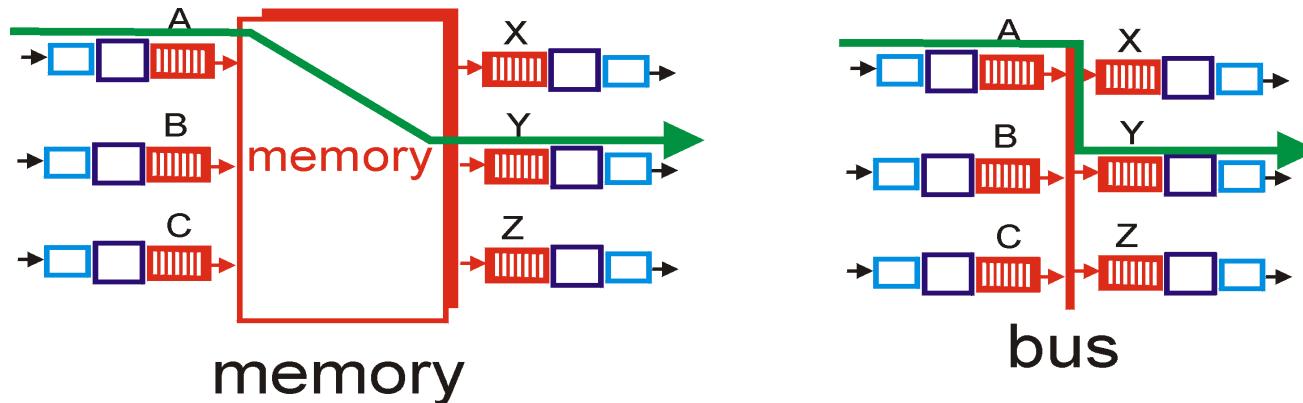


提纲

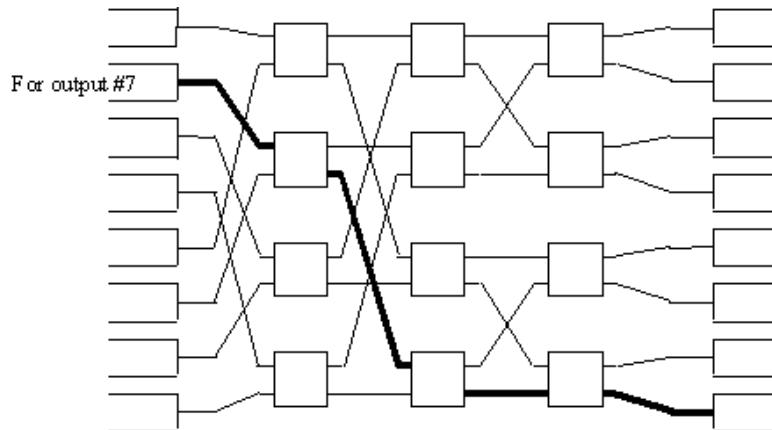
- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
 - 交换基础
 - 端口
 - 交换结构
 - 路由器实现
- 总结



交换结构的三种类型



交换结构: 互联网络架构



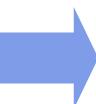
- 克服带宽限制
- 榕树网络, 最初设计应用于多处理器之间的连接网络
- 设计改进: 将数据报分段成为固定长度的单元, 通过交换结构进行交换.
- Cisco 12000: 可达到60 Gbps的交换能力





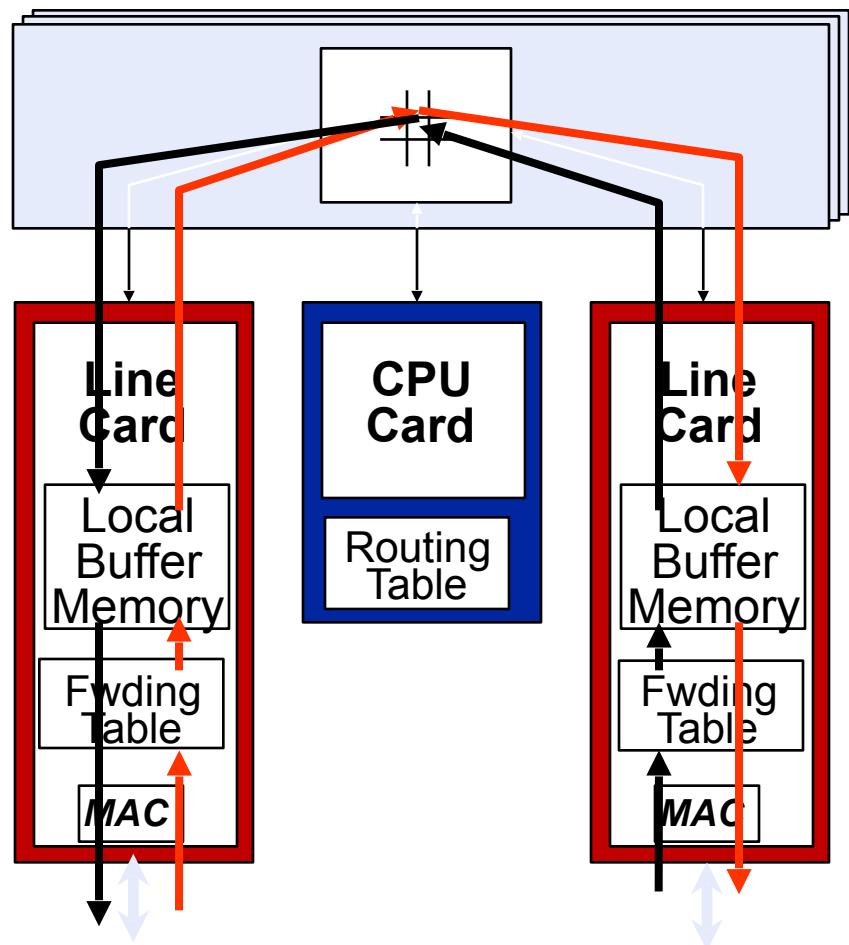
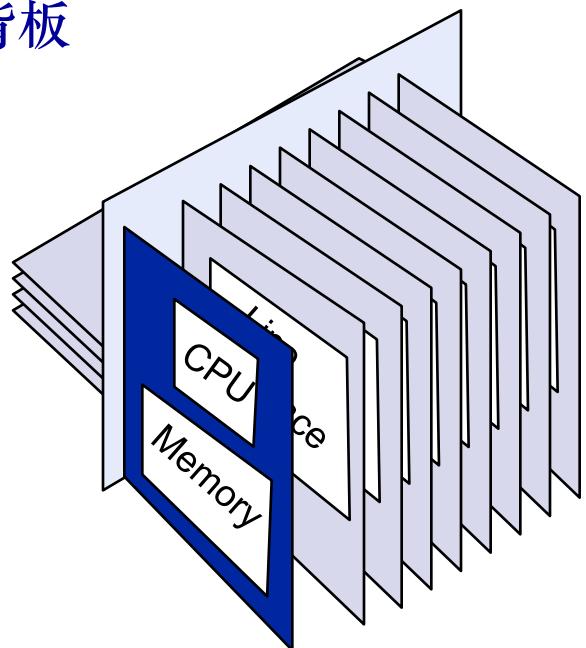
提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
 - 交换基础
 - 端口
 - 交换结构
 - 路由器实现
- 总结

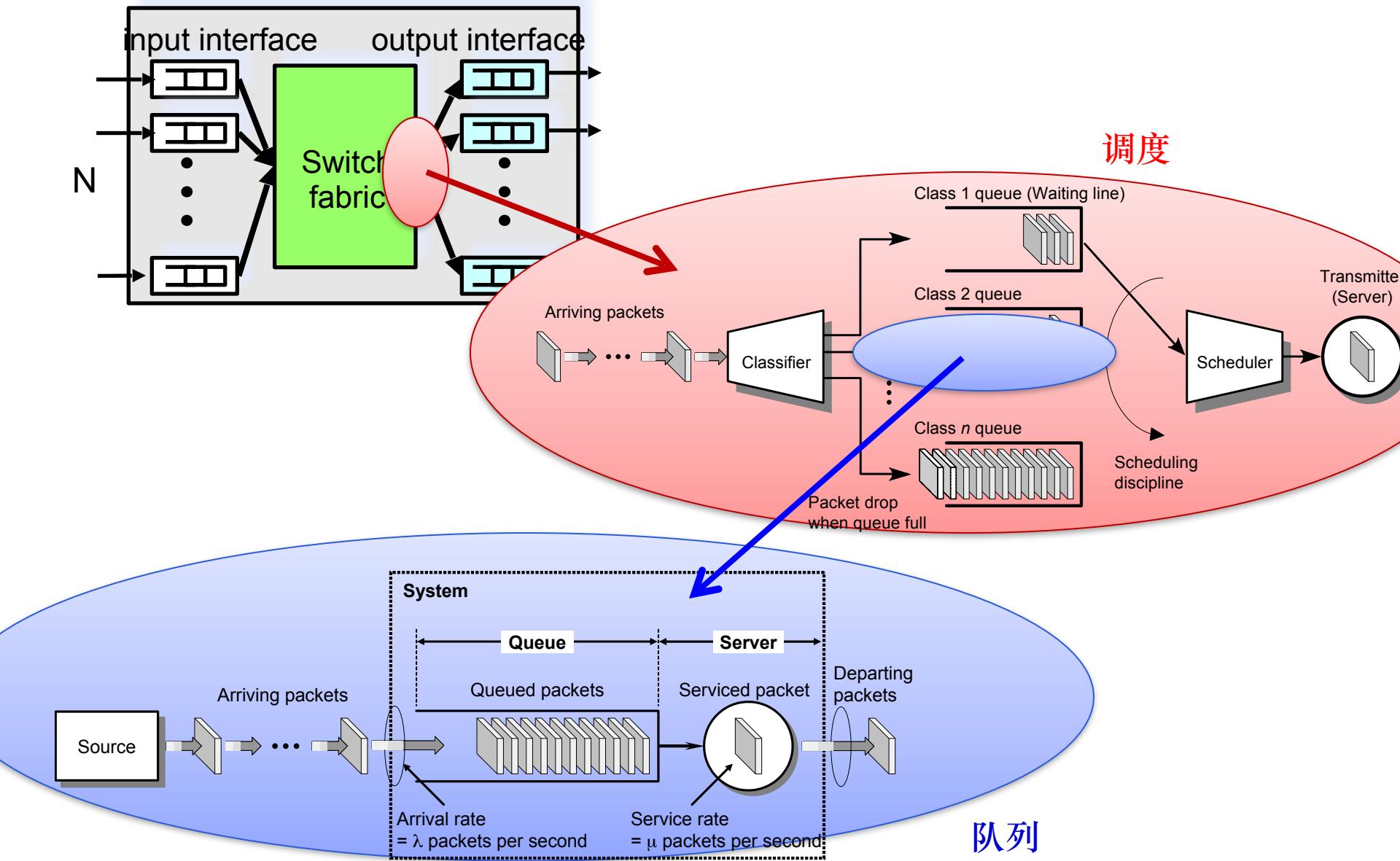


高性能路由器和交换机

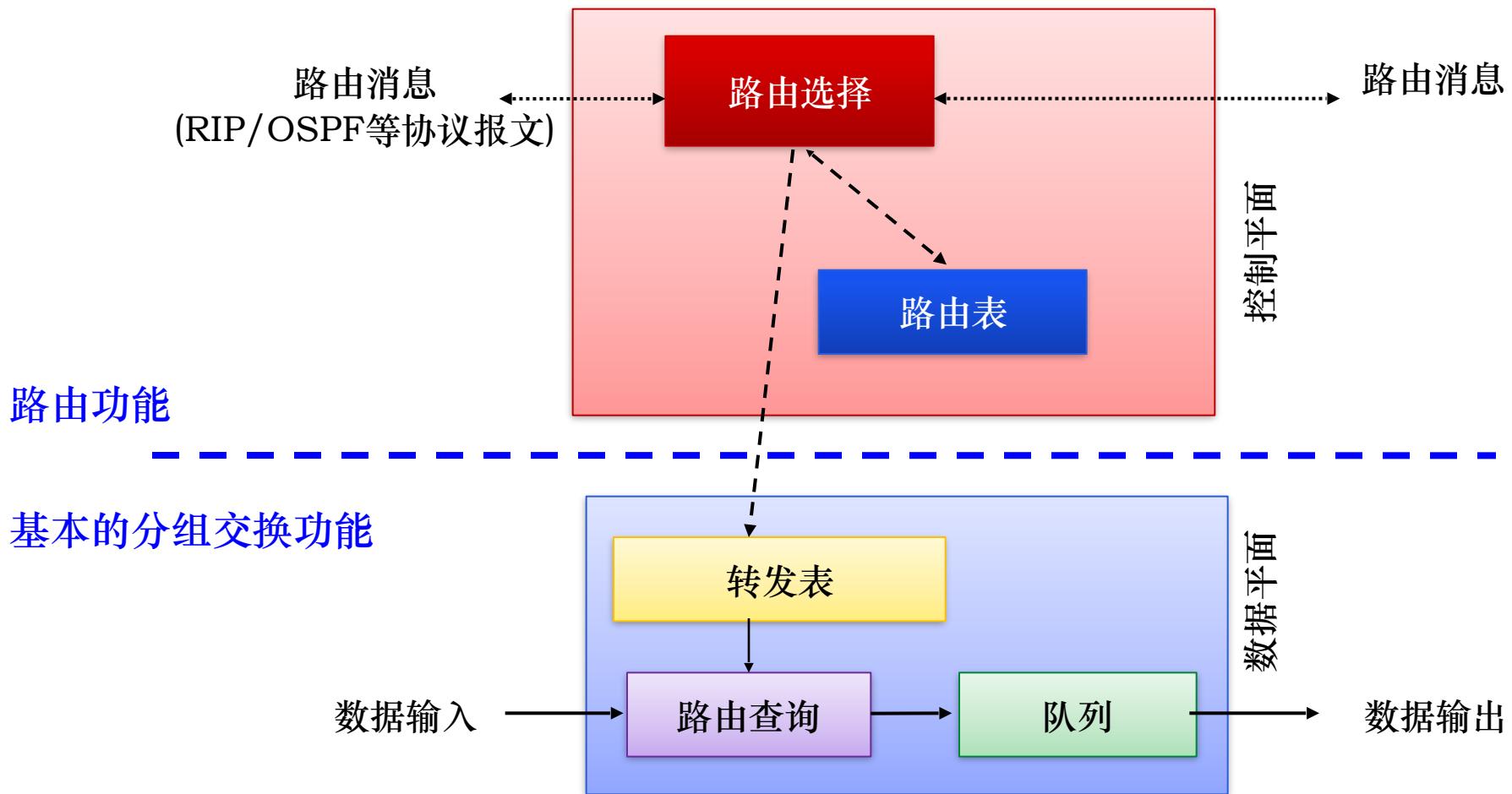
交换背板



路由器性能模型



路由器除交换以外的功能



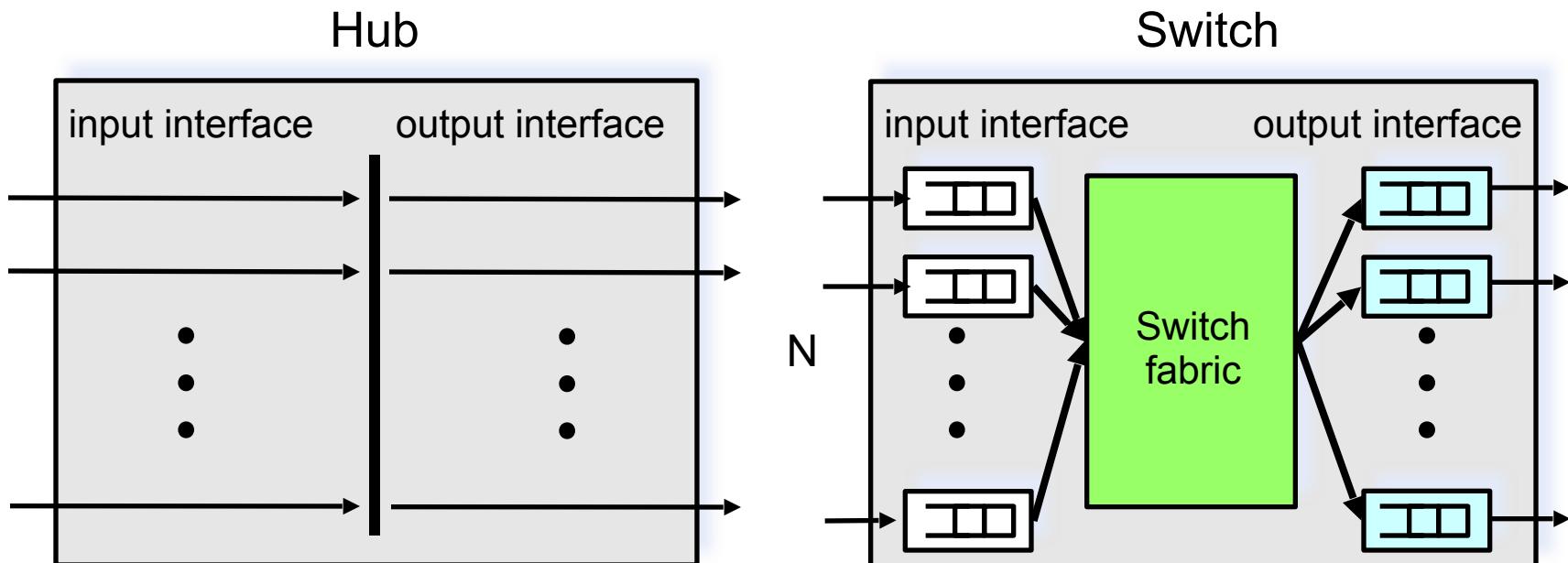
Conceptual model of a router



路由器除交换以外的功能

- 路由器必须能够处理变长分组
 - 端口可以将变长的分组转换为信元，并再将信元转换为分组。
 - 从路由器的性能角度考虑，设计上必须确定在线速率下所能支持的平均分组长度
 - Internet主干网的评价标准建议平均分组长度大约为300字节

集线器vs. 交换机



- 集线器实现共享介质网络；交换机实现分组交换网络
- 集线器连接的M个节点共享N带宽；交换机将到达的分组转发至正确的输出端口，最多为M/2个节点对提供N独享带宽，总吞吐量可达MN/2



网桥, 交换机和路由器

- 从层次的角度:
 - 网桥是链路层的节点
 - 通过不同链路之间的数据帧转发实现LAN的扩展
 - 交换机是网络层的节点
 - 通过不同链路之间的分组转发实现分组交换网
 - 路由器是互联网级别的节点
 - 通过不同网络之间的分组交换实现互联网

然而, 这只是人为的区分, 无法解释很多市场上的实际网络设备

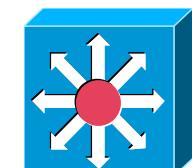
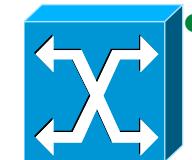
网桥，交换机和路由器

- 交换机是网络层的节点?
- 在市场上, 多接口网桥通常称为以太网交换机或局域网交换机.
- 因此网桥和交换机的区别被混淆.
- 正因如此, 网桥和交换机通常被统一归为“第2层的设备”
 - 第2层意味着“物理层之上, 网际层之下”.



网桥, 交换机和路由器

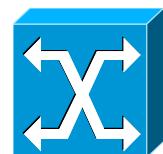
- 交换机是网络层的节点?
- 从层次的观点看待交换机的功能差异:
 - LAN 交换机和网桥依赖于生成树算法
 - 生成树方法不易扩展
 - WAN 交换机通常运行路由选择协议, 学习整个网络的拓扑结构
 - ATM 交换机, 帧中继以及 X.25 交换机
 - 了解整个网络的拓扑结构有助于交换机区分不同的路由





网桥, 交换机和路由器

- 交换机 == 路由器 ?
- 从点到点的链路 (一个合理的网络)来看
 - 区别已不明显.
 - 路由器可以连接多条这种类型的链路.
 - 因此, 路由器视为交换机, 采用数据报模型和IP路由协议进行分组的转发.
- 网络的同构和异构
 - 存在明显差异
 - ATM网络由交换机组建, 网络中仅包含同构的链路
 - Internet 通过路由器组建而成, Internet存在异构性



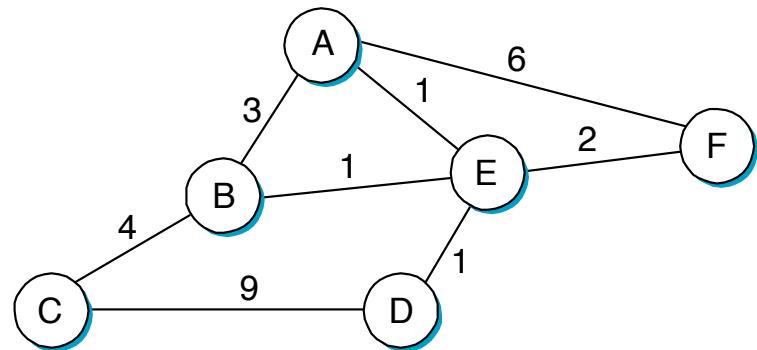


提纲

- 引言
 - 核心问题: 并不是所有网络都是直接相连的
- 交换和桥接
- 互联网基础
- 路由
 - 用图表示网络
 - 距离向量 (RIP)
 - 链路状态(OSPF)
 - 路由评价指标
- 实现和性能
- → 总结

小结: 用图表示网络

- 采用图论表示网络
 - 节点: 路由器
 - 边: 链路
- 最短路径问题
 - Bellman-Ford 算法
 - 在节点对之间交换信息
 - 迭代, 异步, 分布式计算
 - 被距离向量路由选择算法采用
 - Dijkstra's 算法
 - 全局交换信息
 - 递归计算
 - 被链路状态路由选择算法采用





小结: 设计路由协议的考虑 (1)

拓扑信息: 全局信息还是本地信息?

全局:

- 所有路由器拥有完整的拓扑和链路代价信息
- “链路状态”算法

本地:

- 路由器知道物理上连接的邻节点以及到达邻节点的链路代价
- 与邻节点交换信息, 进行迭代计算
- “距离向量”算法



小结: 设计路由协议的考虑 (2)

路由更新: 静态还是动态?

静态:

- 路由变化缓慢

动态:

- 路由快速变化
 - 周期性更新(定时器)
 - 链路代价发生改变(触发器)

谢谢!



华中科技大学
电子信息与通信学院
Email: itec@hust.edu.cn
网址: <http://itec.hust.edu.cn>



参考资料

- *Chapter 3 in L. L. Peterson and B. S. Davie, Computer Networking: A System Approach (5th edition), Morgan Kaufmann, 2012*
- *Chapter 4 in James F. Kurose and Keith W. Ross, Computer Networking: A Top-Down Approach (6th edition), Pearson Education Inc., 2012*
- 吴功宜, 计算机网络 (第3版), 清华大学出版社, 2011

附录



Ethernet frame structure

sending adapter encapsulates IP datagram (or other network layer protocol packet) in **Ethernet frame**



preamble:

- 7 bytes with pattern 10101010 followed by one byte with pattern 10101011
- used to synchronize receiver, sender clock rates



Ethernet frame structure (more)

- ❖ **addresses:** 6 byte source, destination MAC addresses
 - if adapter receives frame with matching destination address, or with broadcast address (e.g. ARP packet), it passes data in frame to network layer protocol
 - otherwise, adapter discards frame
- ❖ **type:** indicates higher layer protocol (mostly IP but others possible, e.g., Novell IPX, AppleTalk)
- ❖ **CRC:** cyclic redundancy check at receiver
 - error detected: frame is dropped





Following Topics in packet switching

- Two examples of the major two kinds of packet-switched network
- Datagram
 - The first packet-switched computer network is ARPANet, and the first datagram switch is IMP in 1969
 - The most successful packet switch today is **Ethernet switch**
- Virtual Circuit (*NOT be mentioned much in our course*)
 - The first public packet switched network standard based on telephone line is X.25, developed by CCITT in 1976
 - The highest performance switch in history is **ATM switch**



History background about spanning tree algorithm

An Algorithm for Distributed
Computation of a Spanning Tree
in an Extended LAN

Radia Perlman

Digital Equipment Corporation
1925 Andover St., Tewksbury MA 01876

1985 ACM Sigcomm

digital

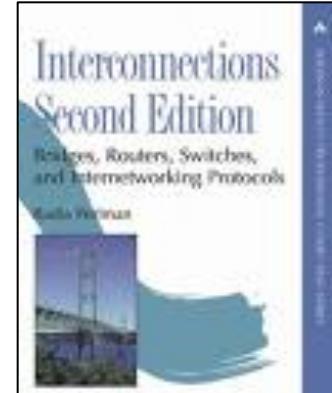
The Spanning Tree Protocol (STP), is defined in
the IEEE Standard 802.1D, in 1990

IEEE 802.1



Her invention of the algorithm behind the
Spanning Tree Protocol solved a
challenging information routing problem and
earned her the moniker "**Mother of the
Internet.**"

"Interconnections: Bridges and Routers", Addison-Wesley,





学术大师+创业明星



I place all of my research work in the public domain. I believe university research should serve society at large, and should be freely available for anyone to use and learn from. I have nothing against patents in principle: Industry invests huge amounts of time and money in R&D and needs to protect new ideas to give them the confidence to continue to invest. As academic researchers, I favor staying ahead by running faster, rather than protecting our backs. It keeps us on our toes and makes us more innovative. And it makes it much easier to work with our colleagues in industry.

I am very lucky that my employer gives me the discretion - as a Principal Investigators - to decide which of my ideas are placed in the public domain. Since 1999, I have placed all my research in the public domain.

Page generated 2014-06-11 16:06:43 EEST, by [jemdoc](#).

<http://yuba.stanford.edu/~nickm/>

Nick McKeown (PhD/MS UC Berkeley '95/'92; B.E Univ. of Leeds, '86) is the Kleiner Perkins, Mayfield and Sequoia Professor of Electrical Engineering and Computer Science at Stanford University, and Faculty Director of the Open Networking Research Center. From 1986-1989 he worked for Hewlett-Packard Labs in Bristol, England. In 1995, he helped architect Cisco's GSR 12000 router. In 1997 Nick co-founded Abrizio Inc. (acquired by PMC-Sierra), where he was CTO. He was co-founder and CEO of Nemo ("Network Memory"), which is now part of Cisco. In 2007 he co-founded Nicira (acquired by VMware) with Martin Casado and Scott Shenker. In 2011, he co-founded the Open Networking Foundation (ONF) with Scott Shenker.

Nick is a member of the US National Academy of Engineering (NAE), a Fellow of the Royal Academy of Engineering (UK), Fellow of the IEEE and the ACM. In 2005, he was awarded the British Computer Society Lovelace Medal, in 2009 the IEEE Kobayashi Computer and Communications Award and in 2012 the ACM Sigcomm Lifetime Achievement Award. He received the IEEE Rice Award for the best paper in communications theory. Nick's current research interests include software defined networks (SDN), network verification, video streaming, how to enable more rapid improvements to the Internet infrastructure, and tools and platforms for networking research and teaching.