



第12章 分布式系统安全

徐恪

清华大学计算机系

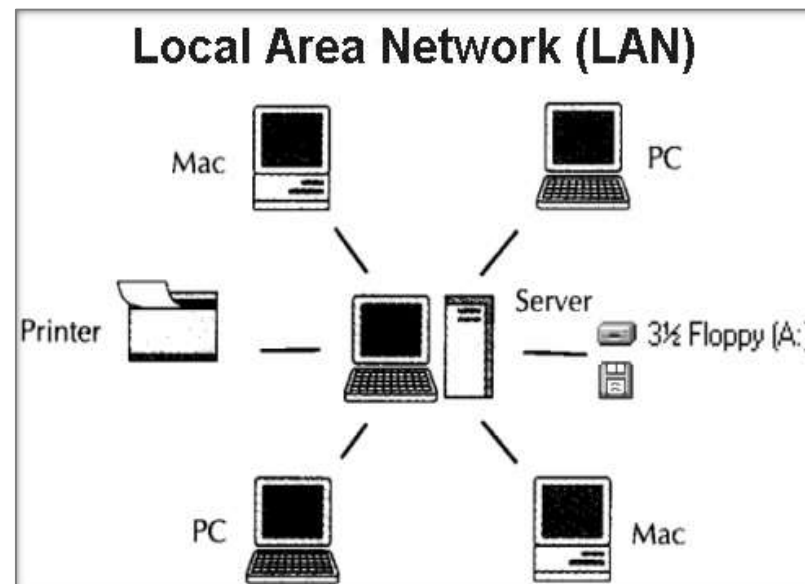
刘卓涛

清华大学网研院



分布式系统的起源

- **并行计算**：Burroughs Corporation在1962年推出了D825，这是一种四处理器计算机，可以访问多达16个共享的内存模块
- **ARPANET**：互联网的前身之一，于20世纪60年代末被推出，分布式系统开始拓展到主机这一粒度
- **LAN**：1970年，以太网等局域网技术的出现，标识着分布式系统开始广泛部署





以太网之父Bob Metcalfe



Bob Metcalfe

美国国家工程院院士、美国艺术与科学院院士，以太网之父，3Com公司创始人
德克萨斯大学奥斯汀分校电气与计算机工程名誉教授
麻省理工学院计算机科学与人工智能计算工程研究助理

2022年图灵奖获得者，曾获国家技术奖章、IEEE荣誉勋章、马可尼奖等



For the invention, standardization, and commercialization of Ethernet

以太网“发明、标准化、商业化”方面的贡献

- 1976年7月，发表具有里程碑意义的论文“Ethernet: Distributed Packet Switching for Local Computer Networks”，对以太网的设计进行了描述
- 1980年，联合Digital Equipment Corporation, Intel和Xerox联合创建了基于以太网的本地网络标准DIX，同时还创建了IEEE 802.3工作组，推动了以太网标准的发展

Computer Systems
G. Bell, S. Fuller and
D. Siewiorek, Editors
**Ethernet: Distributed
Packet Switching for
Local Computer
Networks**
Robert M. Metcalfe and David R. Boggs
Xerox Palo Alto Research Center



分布式系统飞速发展

- 随着互联网技术的不断演进与用户数量的“爆炸式”增长，导致用户需求不断增加
- 基于单一节点的C/S架构所构建的服务无法承受如今的用户规模，分布式系统规模也在不断扩张

日点击量峰值超1500亿次 12306是如何应对下来的？

2018-02-08 22:51

一年一度的春运如期而至，中国又将迎来“人类最大规模的周期性迁徙”。每年的春运，对售票系统12306也是一场艰难的大考。



2018年在线流媒体的订阅用户人数达到6.13亿，全球媒体巨头纷纷入局视频流媒体[图]

2019年09月23日 14:19:25

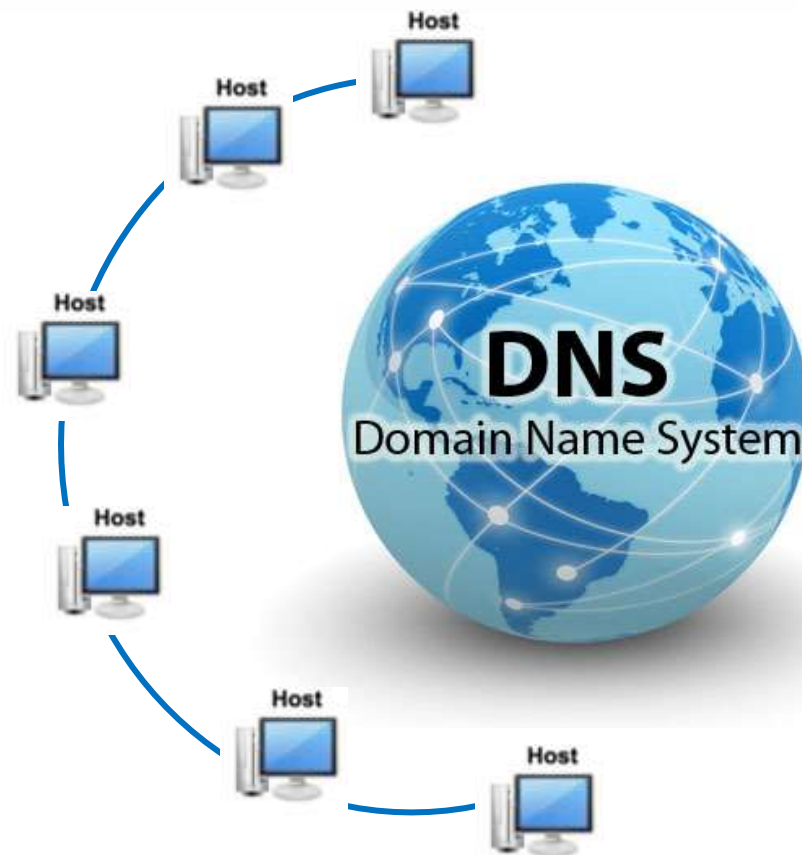
字号: T | T

根据数据，在线流媒体2018年的订阅用户数达到6.13亿，同比增长27%（增量1.31亿），成功超越有线电视5.56亿订阅用户，用户数4年翻4倍。



分布式系统无处不在

如今的我们每天都在和分布式系统打交道，例如：浏览网页用到的DNS系统，微信、搜索引擎以及淘宝、京东等购物平台，比特币、以太坊等区块链平台





讨论

畅所欲言

你认为 分布式系统安全包含哪些内容？



分布式系统是什么？

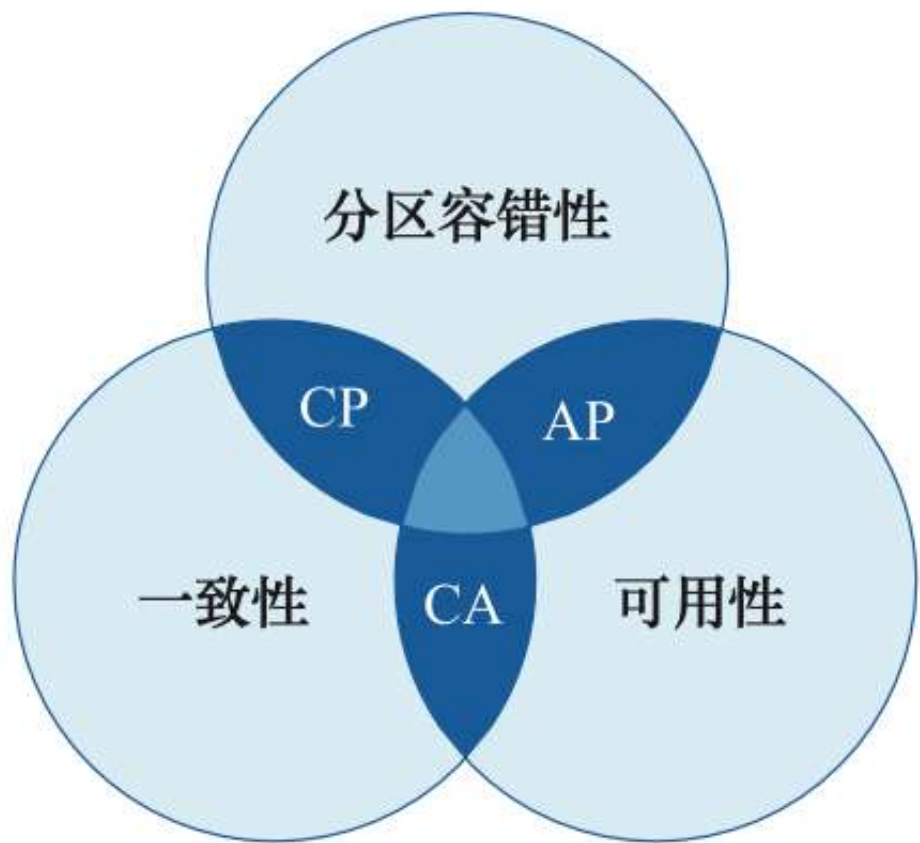
如今所说的分布式系统通常是由分布在不同地理位置的系统组件所构成，这些系统组件通过网络传递消息完成动作协调，从而相互协作完成共同的任务

- 分布式系统的组成：
 - 交互网络 — 实现协作的前提，系统组件基于交互网络实现消息交互
 - 分布式算法 — 协调分布式组建之间的交互，确保系统稳定运行
 - 信任机制 — 解决不同系统组件之间的信任问题，实现可信协作





分布式系统的三个属性

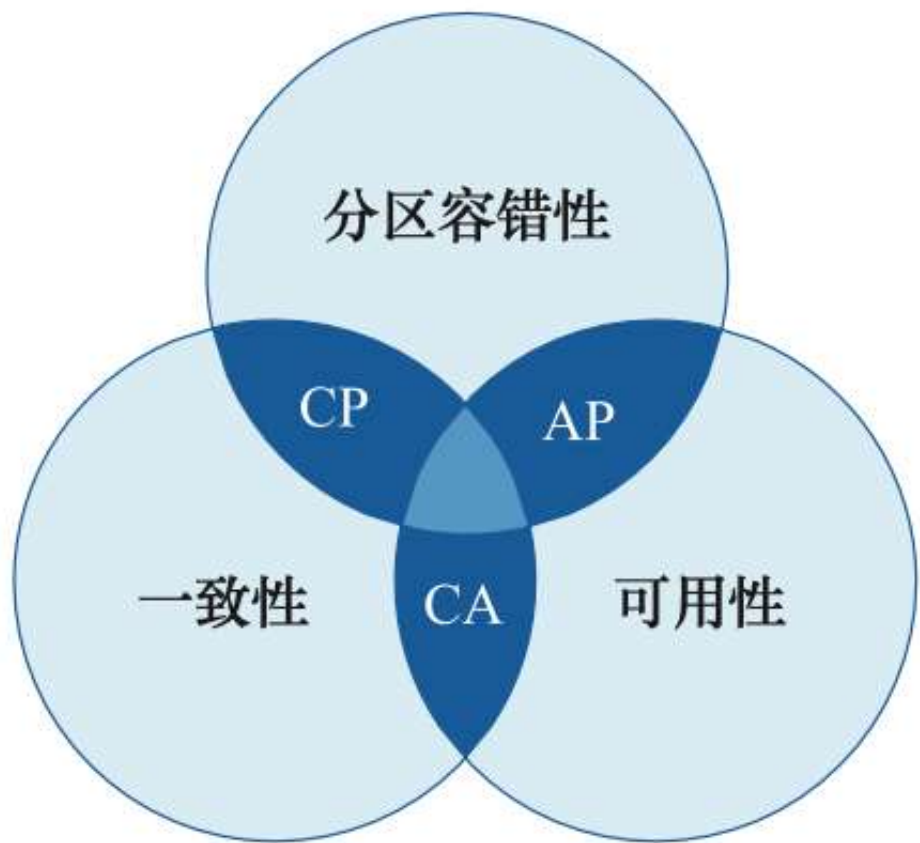


CAP定理

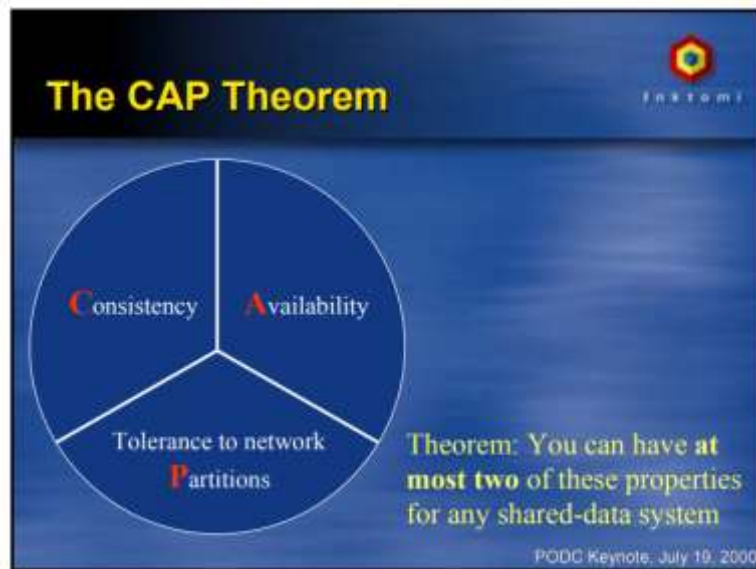
- **一致性**：分布式系统中各正确节点所保存的同一变量状态信息都是一致的
- **可用性**：分布式系统在收到客户端的请求后，必须给出回应，不能让客户端陷入无限等待过程中
- **分区容错性**：分布式系统容忍其中节点出现分区，当分区出现时，一个区域中节点发往另一个区域中节点的数据包全部丢失，即区域间无法进行通信



分布式系统的三个属性



CAP定理



Eric Brewer
UC Berkeley

在一个分布式系统中，一致性、可用性、分区容错性，三者不可得兼

P总需要满足，而C和A矛盾



分布式系统安全问题的根源

安全问题可能会造成系统分区，或是危害系统的一致性、有效性和隐私性

- 从分布式系统的组成来看，其安全问题的根源体现在交互网络、分布式算法和信任三个方面：
 - 攻击者可以攻击交互网络，监听交互过程中的隐私信息或造成网络分区
 - 协作过程中可能会出现冲突，故障等问题，从而影响一致性和可用性
 - 系统内部可能存在恶意组件，系统和客户端交互时也存在信任问题





本章的内容组织



第一节

建立安全、稳定的交互网络

- 交互网络的组成
- 交互信道安全
- 弹性覆盖网络
- 防御日蚀攻击

了解交互网络的组成以及如何构建安全的交互网络



第二节

分布式算法

- 协作过程中的安全隐患
- 时钟同步算法
- 并发问题
- 故障容错的一致性算法

挖掘分布式系统各组件之间如何进行协同



第三节

信任问题

- 访问控制
- 信任评价模型
- 拜占庭容错

了解分布式系统中的信任问题及解决方案

为分布式系统各组件之间的协同提供交互基础

为分布式系统添加信任面，实现可信协同



第1节 建立安全、稳定的交互网络

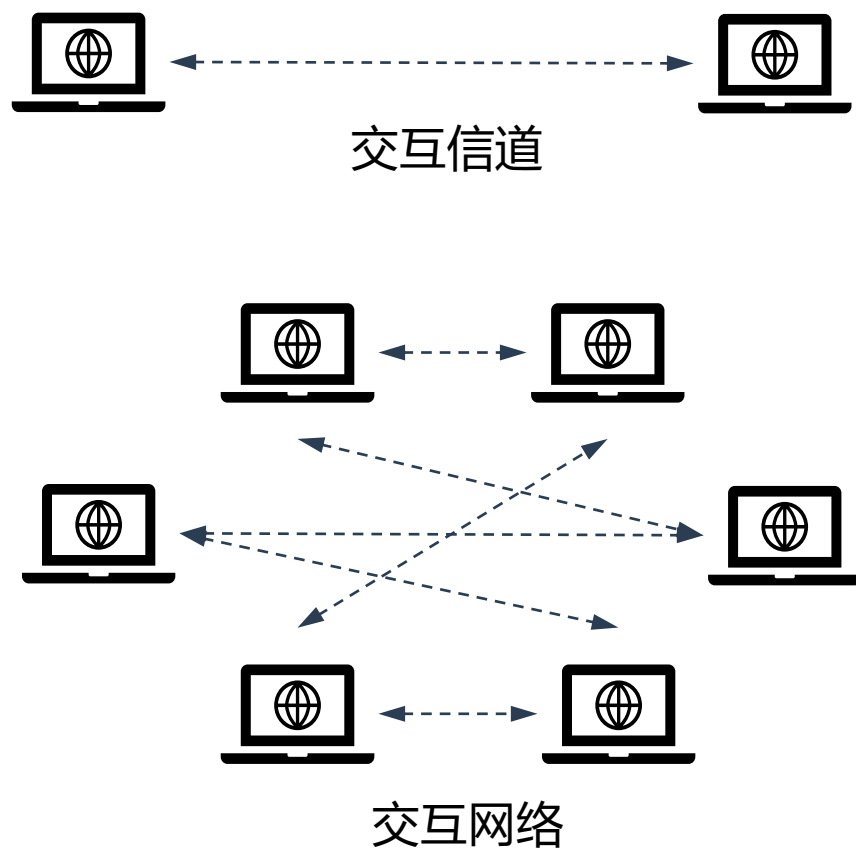
- ✓ 交互网络的组成
- ✓ 交互信道安全
- ✓ 弹性覆盖网络
- ✓ 防御日蚀攻击



交互网络的组成

在分布式系统中，节点之间可以通过交互信道实现消息传递；分布式节点选择邻居节点并建立交互信道，从而构成一个交互网络

- **交互信道**：两个主机通过交互信道实现host-to-host的通信，常见的交互信道有UDP、TCP和TLS等
- **邻居选择机制**：每个节点依靠邻居选择机制选取系统中的部分或全部节点建立交互信道，从而建立邻居关系，节点可通过邻居节点与其它非邻居节点交互

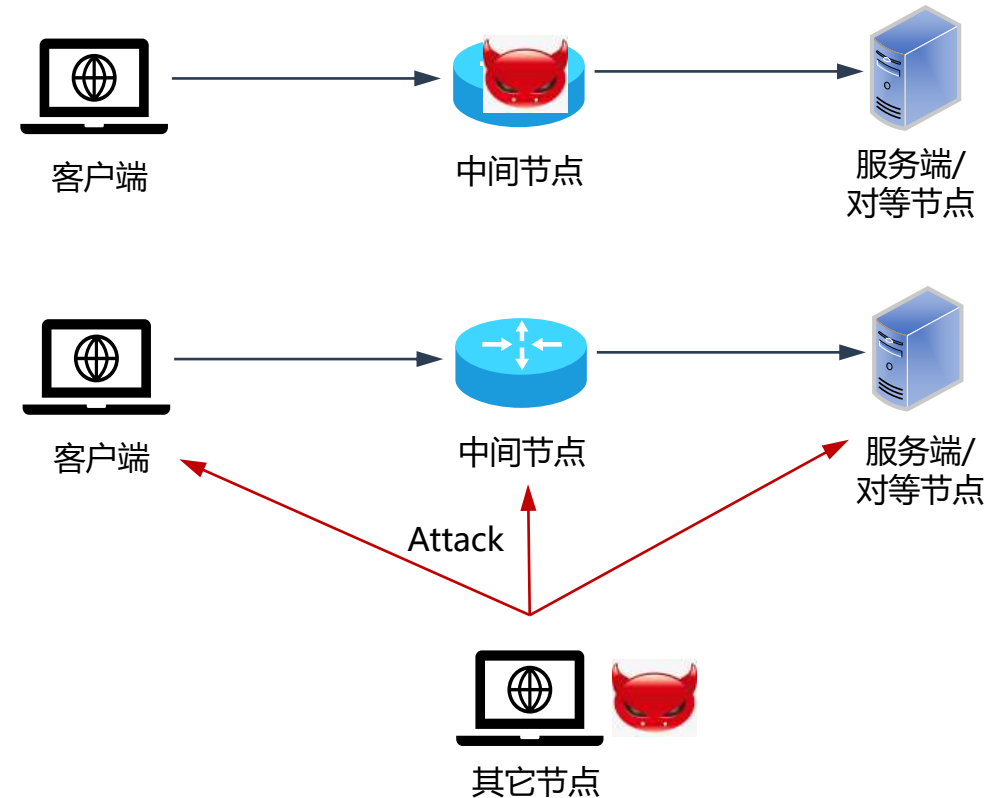




交互信道中的安全隐患

- 根据网络体系结构的不同层面进行分类
 - 网络层面：网络中传输的数据包可能出现**丢失**、**重复**和**泄漏**
 - 应用层面：节点间的交易请求可能出现**泄漏**、**重放**、**篡改**和**伪造**

- 根据攻击者的位置分类
 - **中间转发节点**：直接参与转发数据包，攻击能力强，可以监听、篡改、伪造、重放、延迟甚至丢弃转发信息
 - **其它节点**：攻击能力较弱，但能够通过DDoS、中间人等攻击影响交互信道

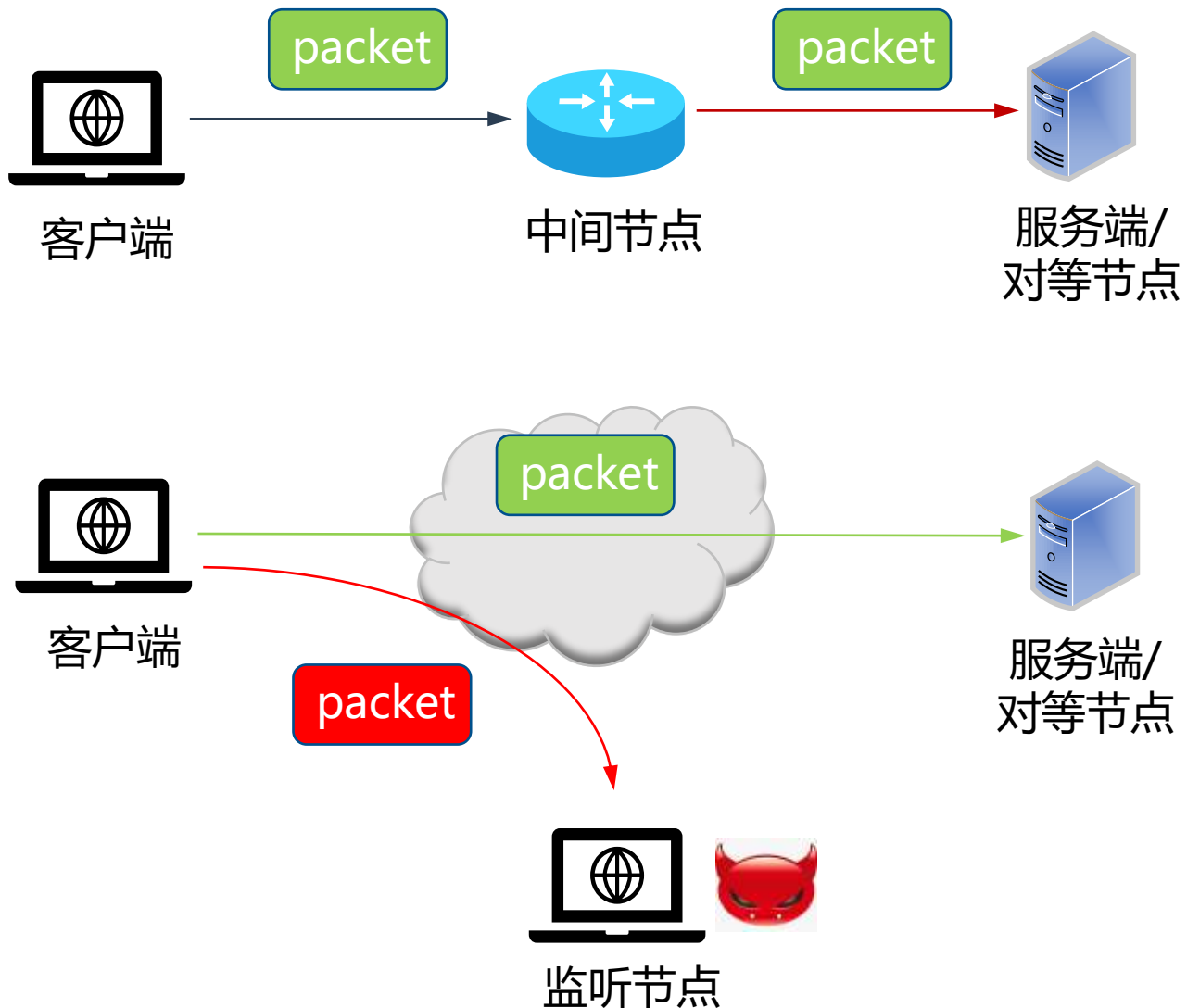




交互信道中的安全隐患

- **数据包丢失和重复：**当网络出现拥塞或被攻击时，客户端发送的数据包可能会丢失；此时客户端重传数据包可能导致服务端收到重复数据包

- **信息泄漏：**明文传输数据包时可能会发生，节点可以将监听到的数据包进行重组，从而获取应用层数据信息

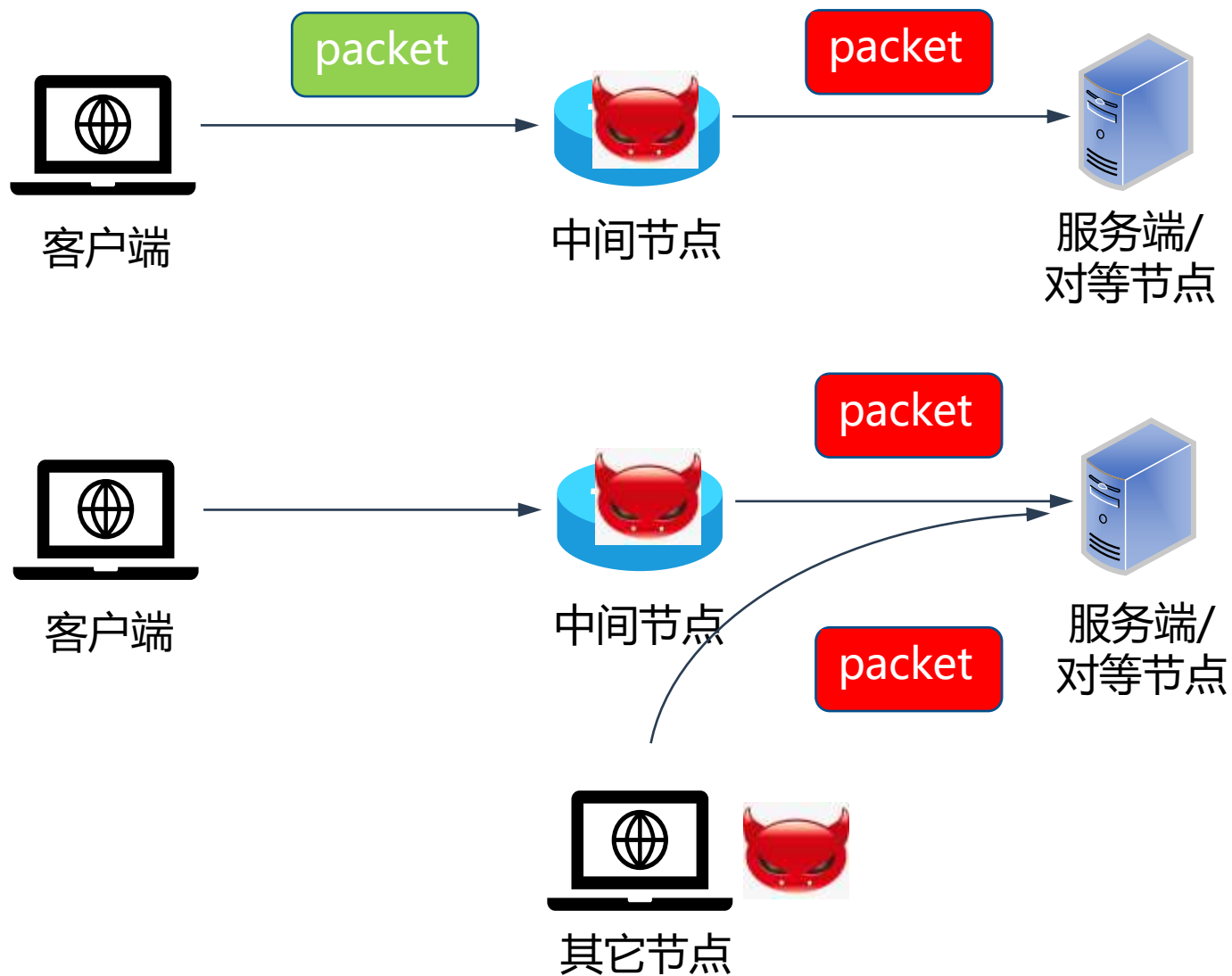




交互信道中的安全隐患

- 数据包篡改
 - 中间转发节点可能篡改经过的数据包
 - 其它节点伪造

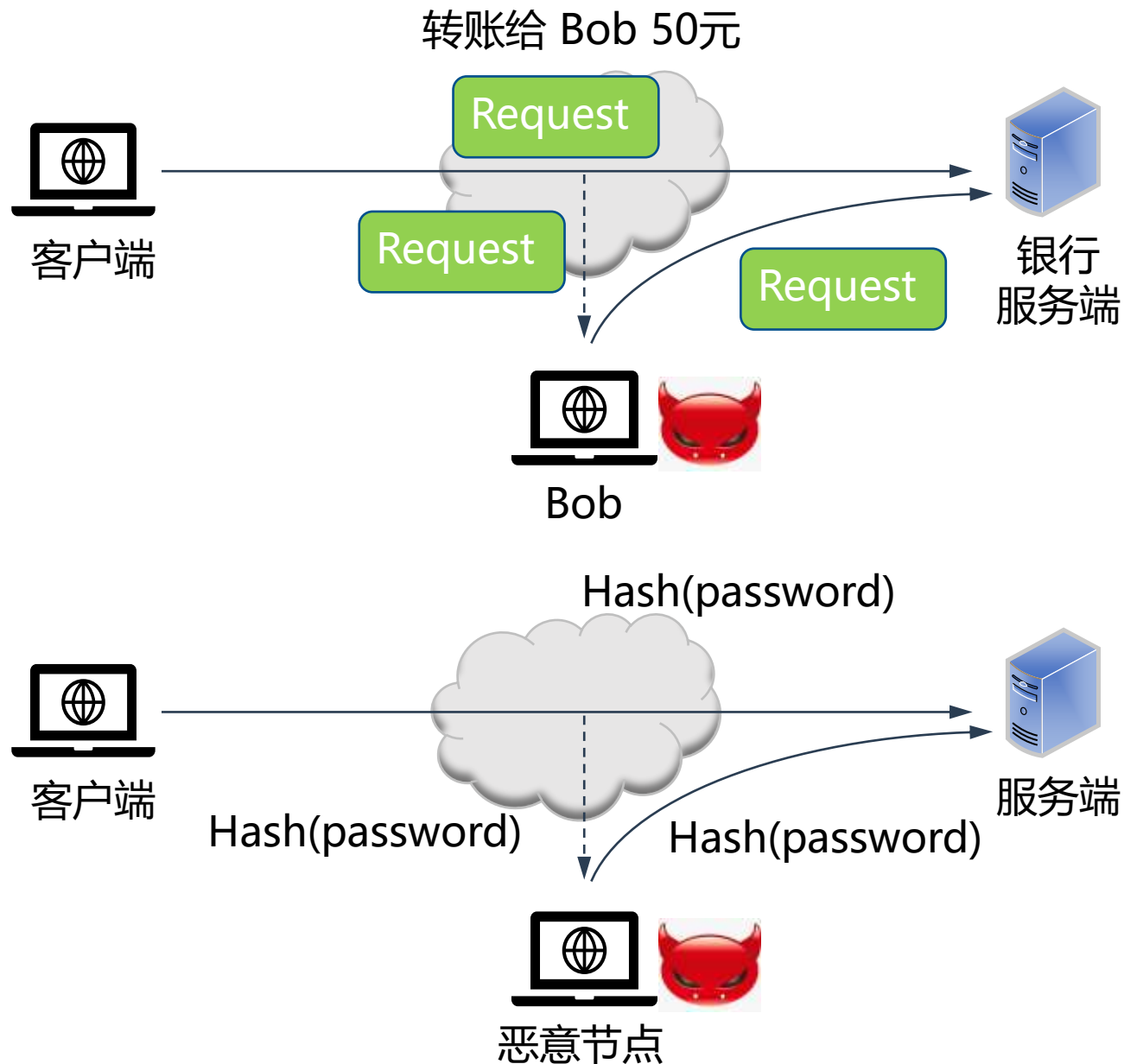
- 数据包伪造
 - 中间转发节点或其它节点伪造数据包
 - 其它节点伪造客户端发送应用层请求





交互信道中的安全隐患

- 重放攻击：恶意节点通过监听数据包构造客户端发往服务端的应用层请求，重新将该请求多次发送给服务端
- 客户端发送交易转账给Bob，Bob监听到该交易后进行重放，从而完成多次转账
- 客户端发送密码登入服务端，恶意节点监听到消息后进行重放，从而冒充客户端登入服务端

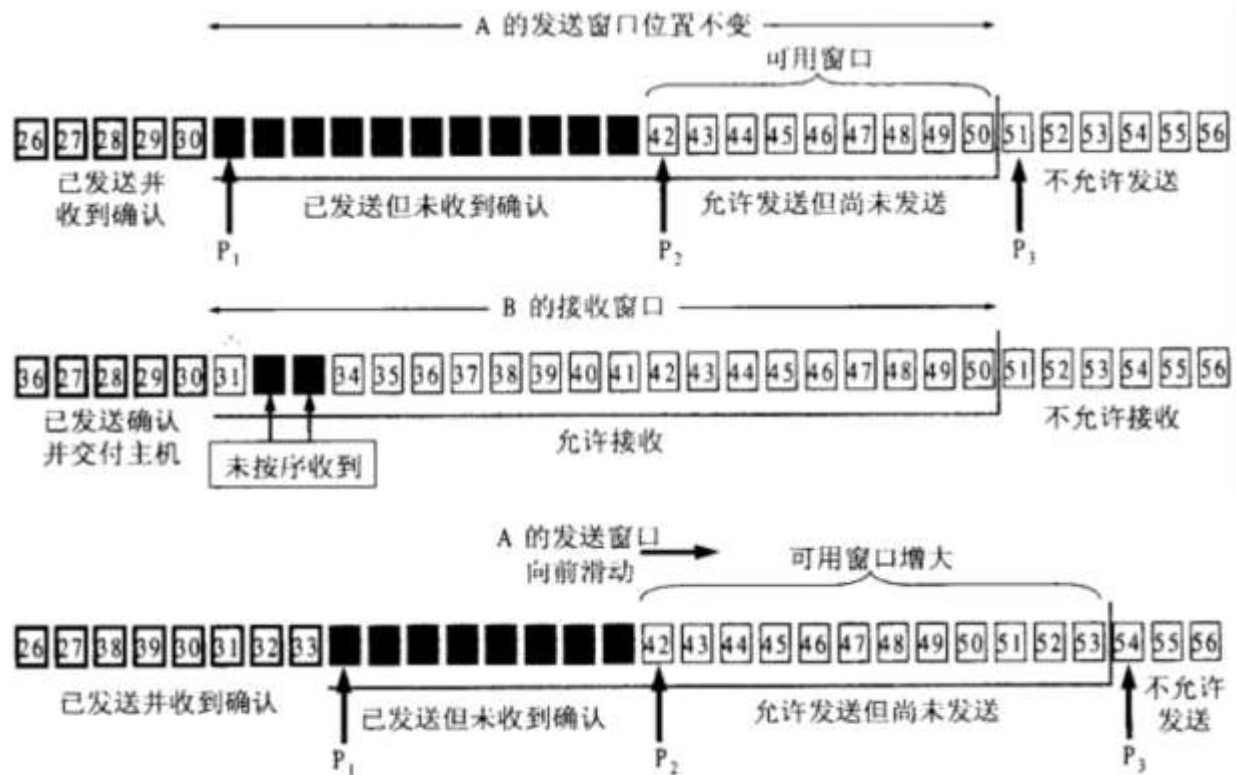




构建TCP交互信道

TCP协议采用滑动窗口协议确保数据包有序提交，并解决重复问题；采用超时重传机制解决链路拥塞导致的数据包丢失问题

- 当未按序接收时，等待前面的数据包完成接收后再将数据包序列提交到上层应用
- 当长时间未收到数据包时，触发超时空重传机制，请求源端超时重传数据包



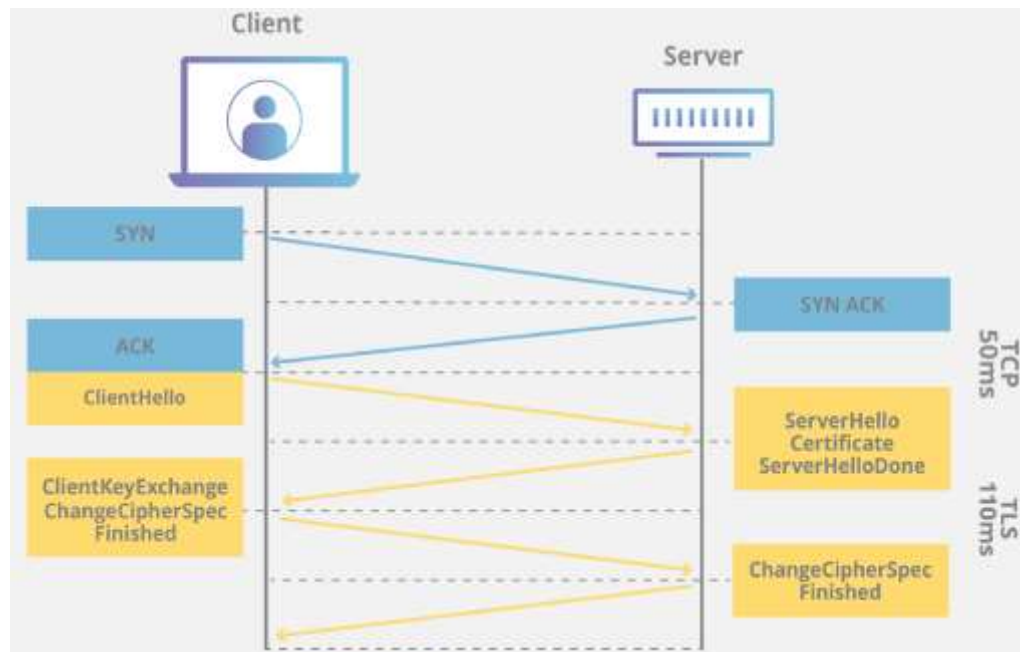


构建TLS交互信道

TLS 协议建立在TCP协议之上运行，提供以下三个安全功能：

- **数据完整性：**验证数据是否伪造而来或未遭篡改过
- **身份认证：**获取通信方的身份，为访问控制机制作支撑
- **数据加密：**协商会话密钥用于数据加密，防止第三方窃听传输数据

- 基于PKI获取对方公钥证书
- 基于公钥非对称加密协商随机数
- 基于协商的随机数生成会话密钥，从而建立安全信道
- 基于会话密钥对传输数据加密，并生成消息认证码对消息完整性进行验证





TLS防重放攻击

- TLS交互信道带来的优势
 - 基于TCP协议确保传输层面数据包不会出现重复
 - 通过数据加密确保恶意节点无法通过监听获取应用层消息

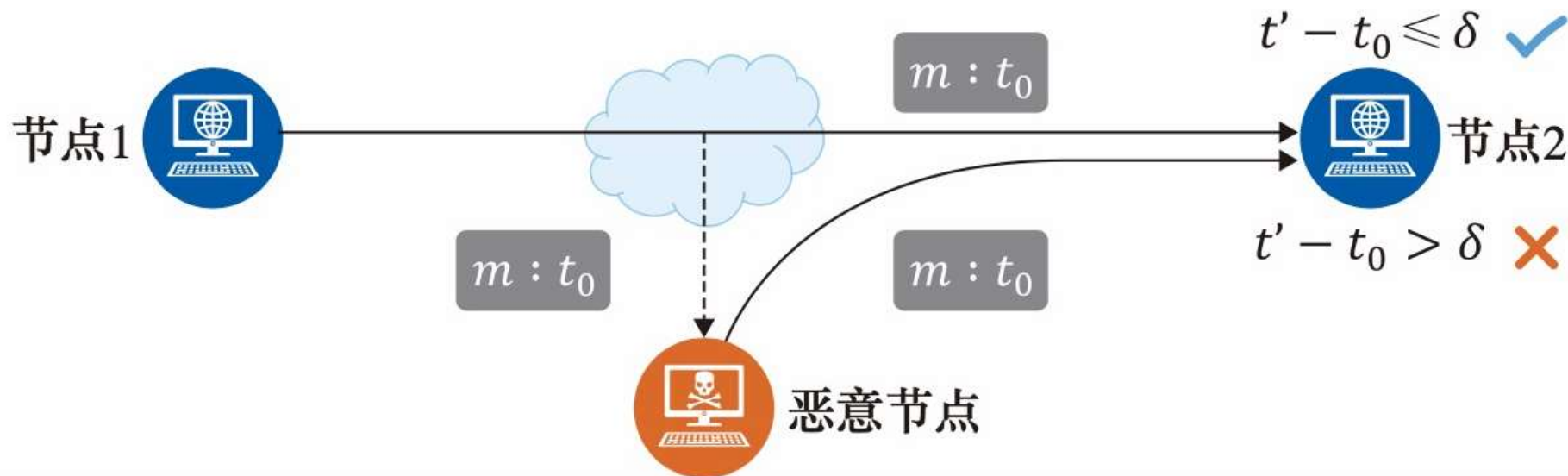
问题：TLS是否可以完全解决重放攻击？

- 以下消息依然可能遭受重放攻击
 - 客户端发起的TLS请求建立信息ClientHello
 - TLS1.3为了加速握手过程而引入的0-RTT信息



基于时间戳防御重放攻击

- 交互双方完成精确时钟同步
- 节点1在发送的消息上附带本地时间戳 t_0
- 节点2收到消息后，用本地时间减去消息中所附带的时间戳得到一个时间差
- 若时间差小于允许值 δ ，则接受该消息；否则丢弃



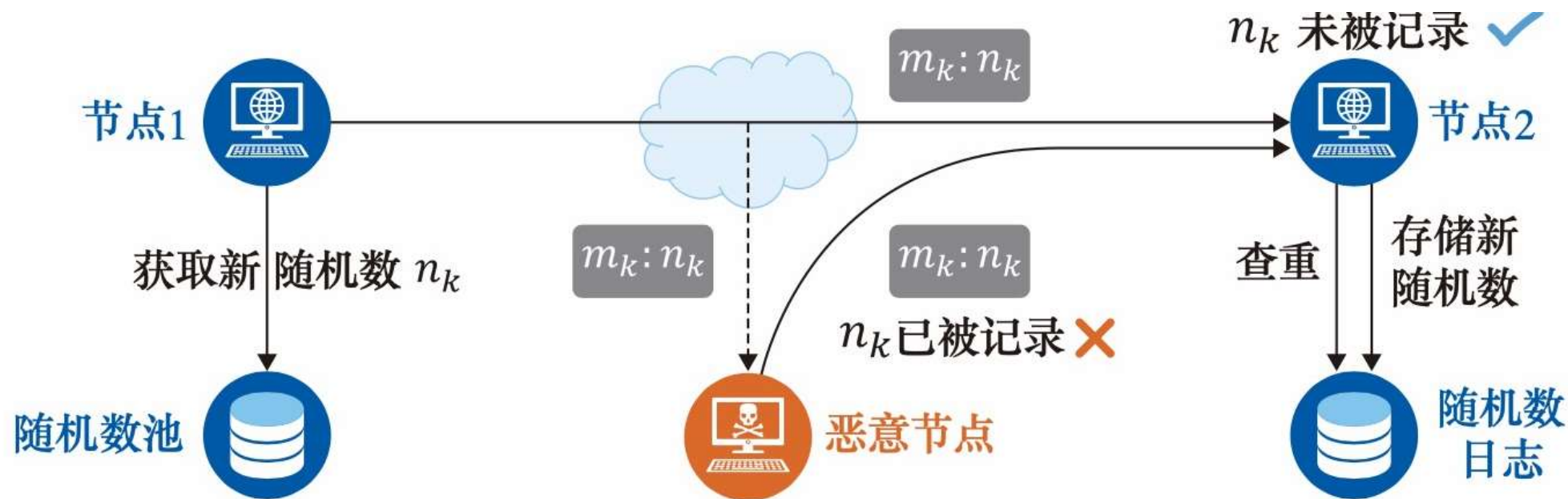
优点：容易实现

缺点：要求交互双方进行精确的时钟同步；存在一个可以被攻击的时间窗口



基于随机数防御重放攻击

- 节点1维护随机数池，和节点2交互的每条消息都附带新随机数
- 节点2收到消息后，判断该消息所附带的随机数是否已在日志R中
若否，则接受该消息，并将新随机数存储在本地图志；若是，则丢弃该消息



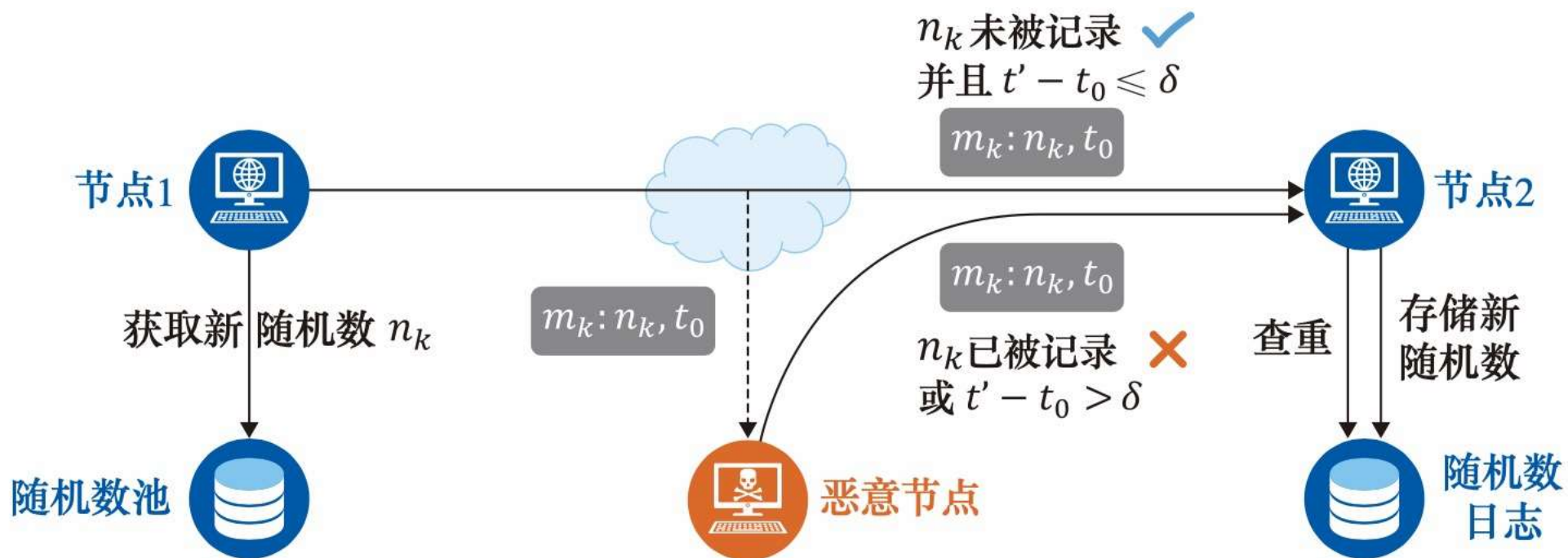
优点: 无需时钟同步；不存在可被攻击的时间窗口

缺点: 复用之前的数时，可能遭受重放攻击；需要记录大量已用过的随机数



时间戳和随机数相结合

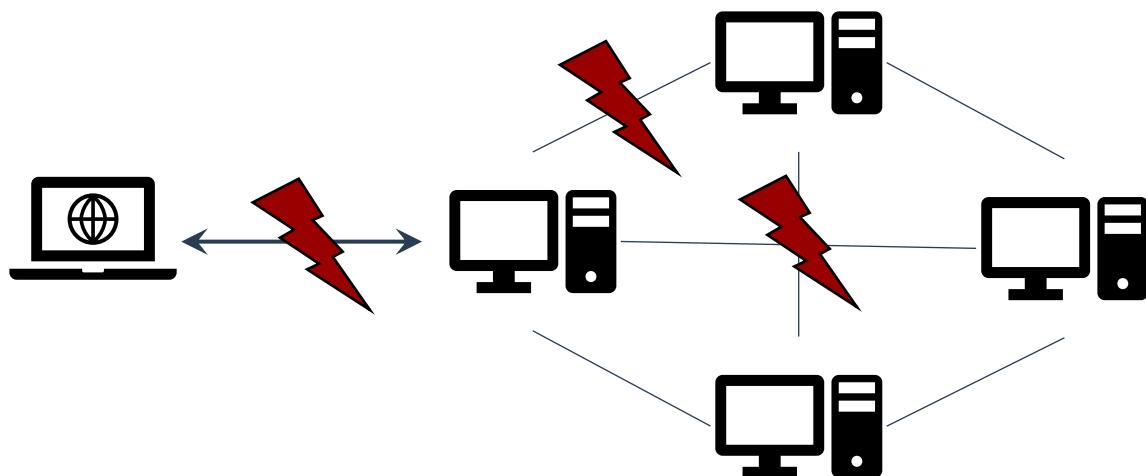
- 节点1和节点2时钟同步，无需非常精确
- 节点1维护随机数池，随机数的量确保可以在短时间内不重复
- 节点2只存储短时间内的随机数
- 对消息的判断需同时判断时间戳和随机数





当网络链路出现故障

- 链路故障类型
 - **链路崩溃:** 路由配置、BGP劫持等问题, 使主机无法和部分其它域的主机通信
 - **性能问题:** 网络攻击导致主机和部分其它域主机通信的延迟显著上升



- 链路故障的特征
 - 无法通过网络层路由解决
 - 通常只会影响节点与部分节点间的通信信道, 不会造成网络分区



造成网络故障的主要原因

Internet的不稳定性，分布式系统所依赖的底层网络随时都可能出现物理链路故障、路由故障等因素

Paxson 95-97	<ul style="list-style-type: none">• 3.3% 的路由存在严重问题
Labovitz 97-00	<ul style="list-style-type: none">• 10% 的路由有效时间不超过 95%• 65% 的路由有效时间不超过 99.9%• 路由故障的检测与恢复至少耗费3分钟，平均花费15分钟• 40% 的链路崩溃需要至少30分钟才能修复
Chandra 01	<ul style="list-style-type: none">• 5% 的链路错误持续2.75小时以上



解决思路

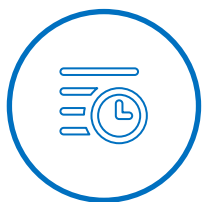
构建一个弹性覆盖网络，依靠其它主机实现消息的转发



覆盖网络

Overlay Network

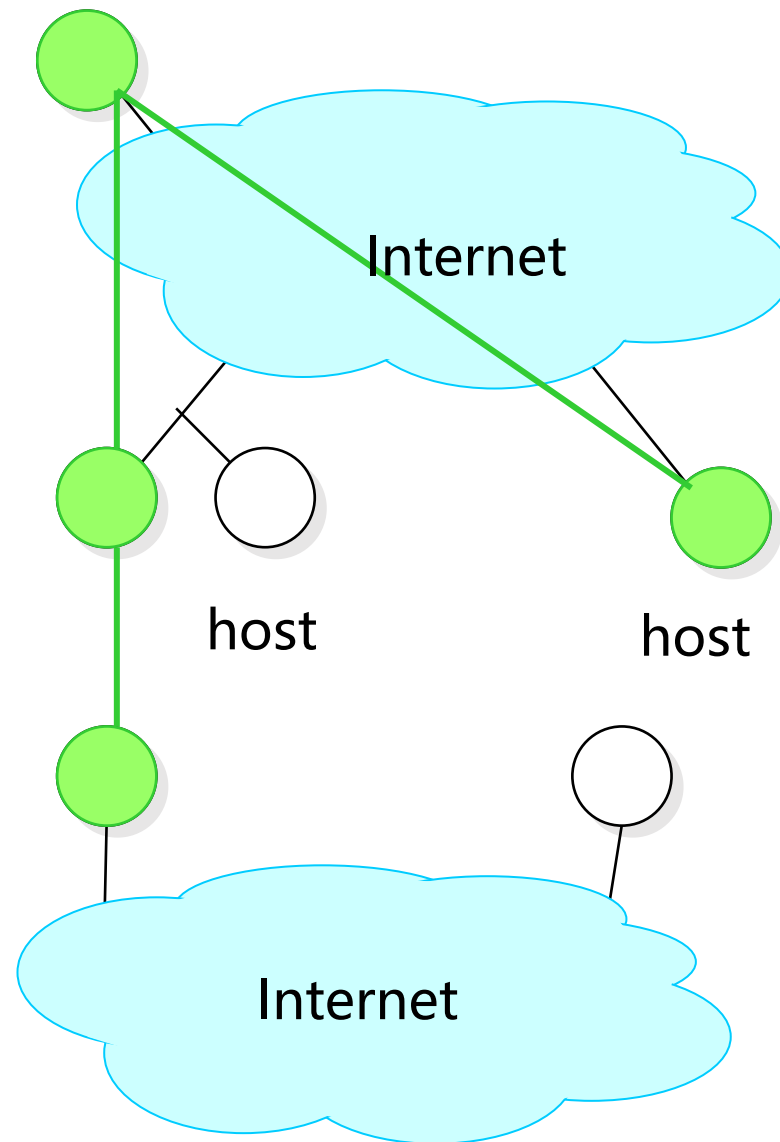
现有Internet网络上构建一个完全位于应用层的网络系统



拓扑发现，路由等功能完全由应用层完成，不依赖网络层



基于Internet网络的大规模分布式应用

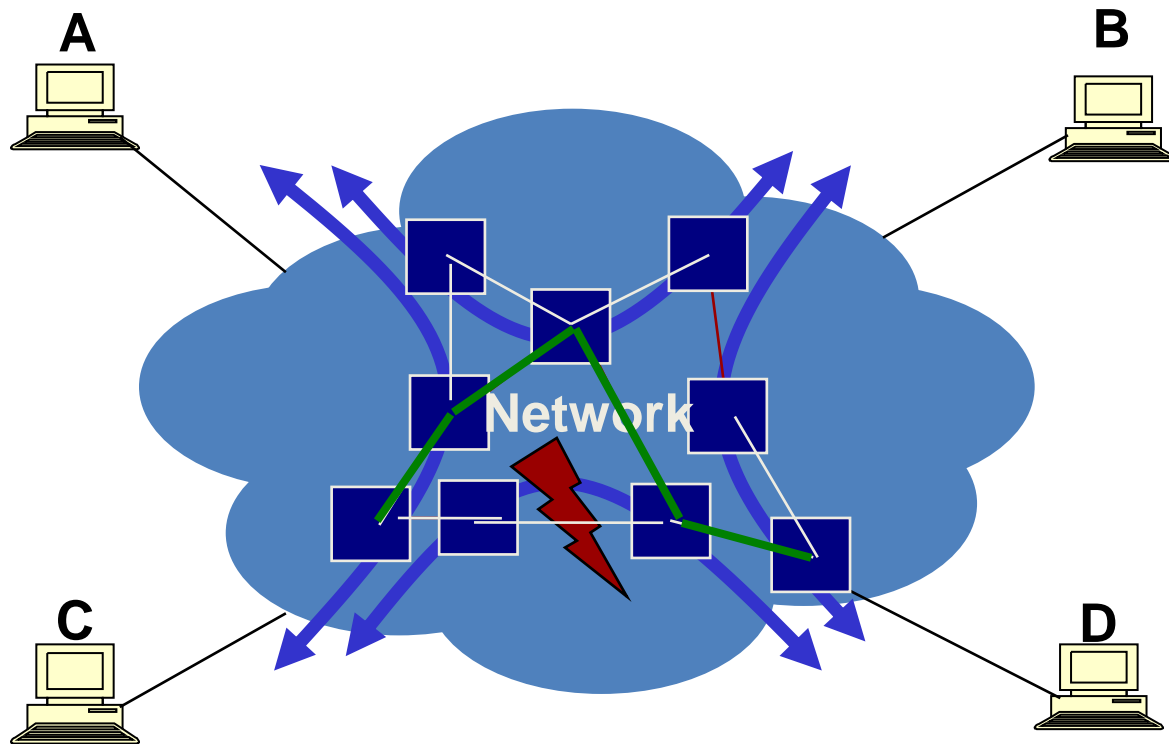




弹性覆盖网络RON

- 弹性覆盖网络是一种分布式覆盖网络体系结构，它可以使分布式应用检测到**路径的失效和周期性的性能降低现象**并能够迅速恢复

- 恢复时间少于20秒
- 对比：**在目前的Internet中，BGP恢复时间为几分钟



Packet switching and route around failures

- RON节点监控它们之间的路径的质量并根据这些信息决定
 - 直接利用Internet转发分组
 - 通过其他RON节点转发分组



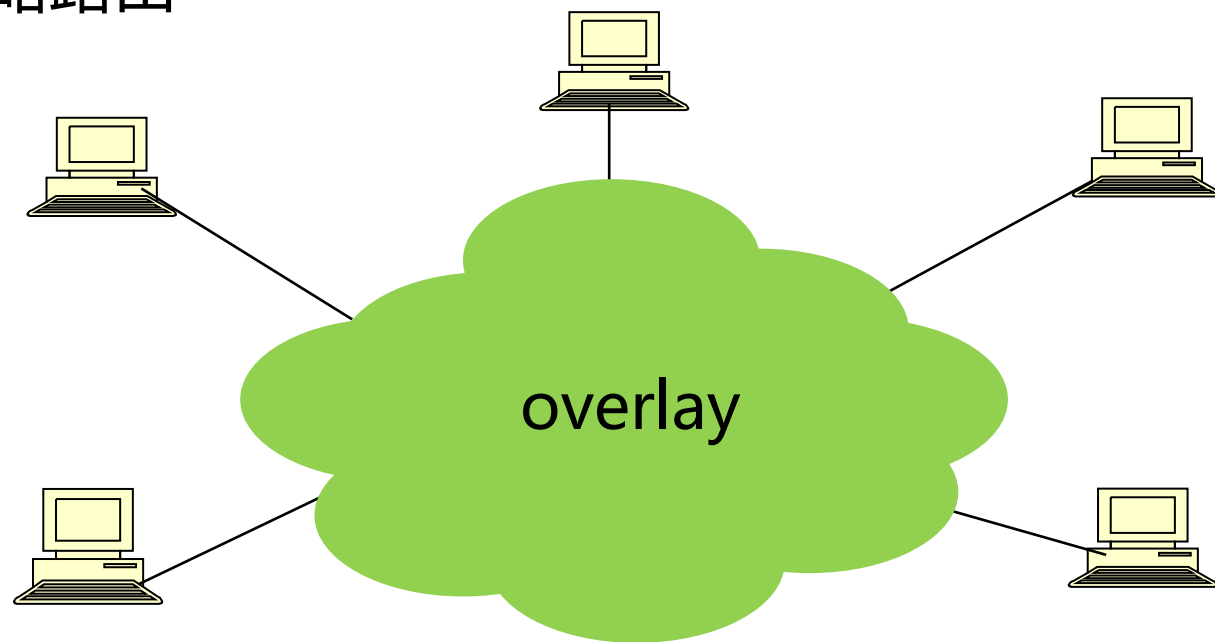
RON的目标

- **设计目标**

- 在20秒内完成故障检测和恢复，将通信的可靠性提升至少十倍
- 将路由和路径选择与应用更紧密地结合起来
- 在应用层实现表达能力丰富的策略路由

- **应用场景**

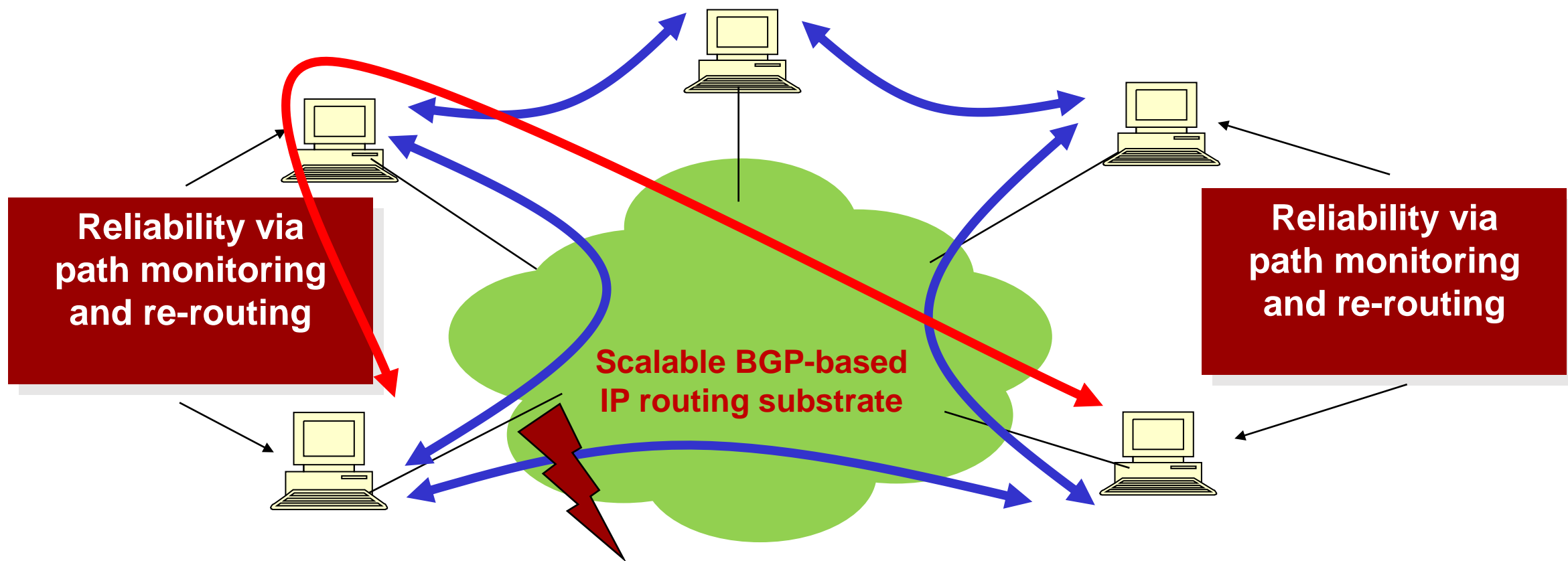
- 协同服务和视频会议
- 构建跨越互联网的VPN
- 可靠的overlay网络服务





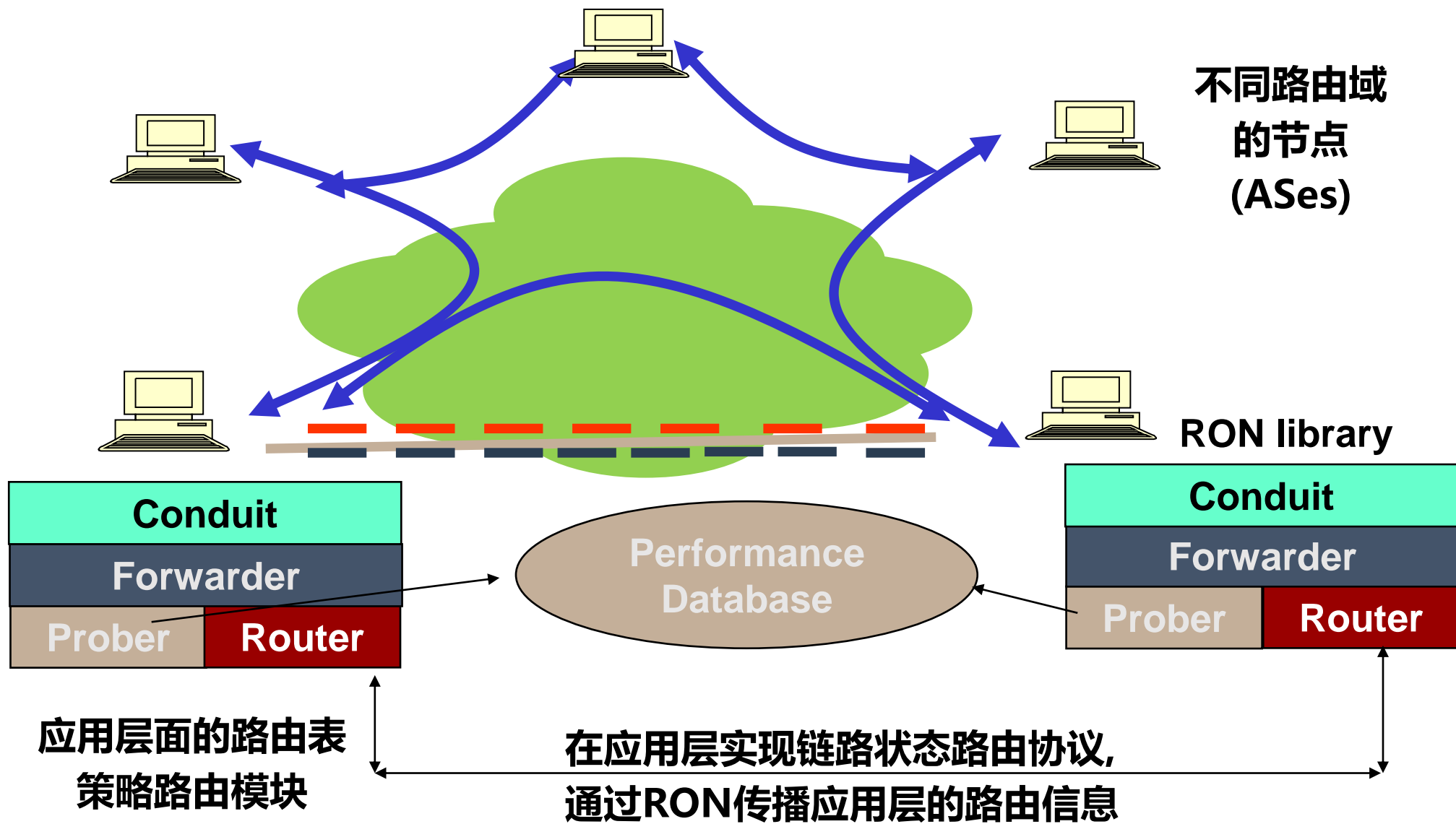
RON的设计

主要思路：不同路由域的终端合作完成信息的传递，这些终端在广域网之上实现主机与主机之间的路由，从而实现更强的稳定性





RON的设计



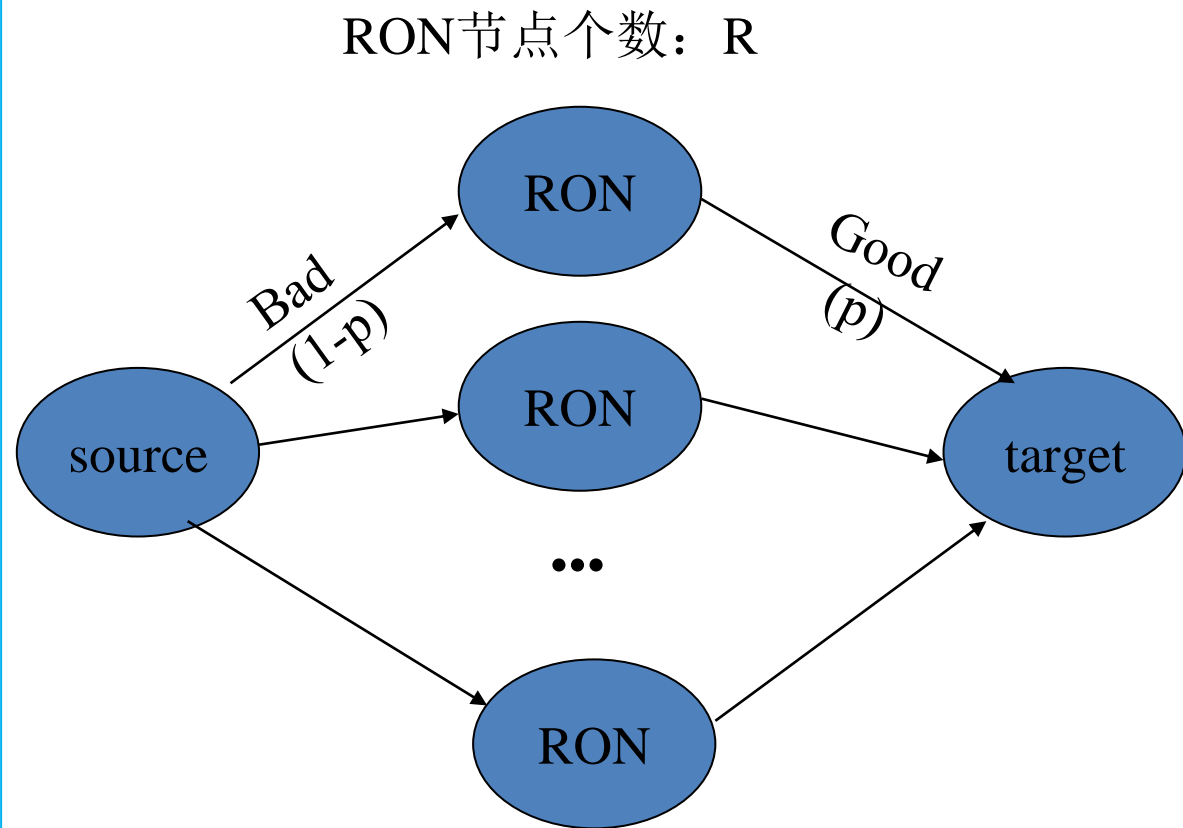


RON的稳定性分析

- 两个主机间无法正常通信的概率为 $(1-p)$
- 两个主机间能正常通信的概率为 p
- 只考虑一跳路由时，存在正确路径的概率为：

$$P(\text{good path}) = 1 - (1 - p^2)^R$$

- 在RON的测试环境中：
 - $P(\text{两个主机间能正常通信}) = 48.8\%$
 - $P(\text{两个主机能通过一跳路由间接通信}) = 51\%$
 - 无法找到正确路径的概率只有 0.2%

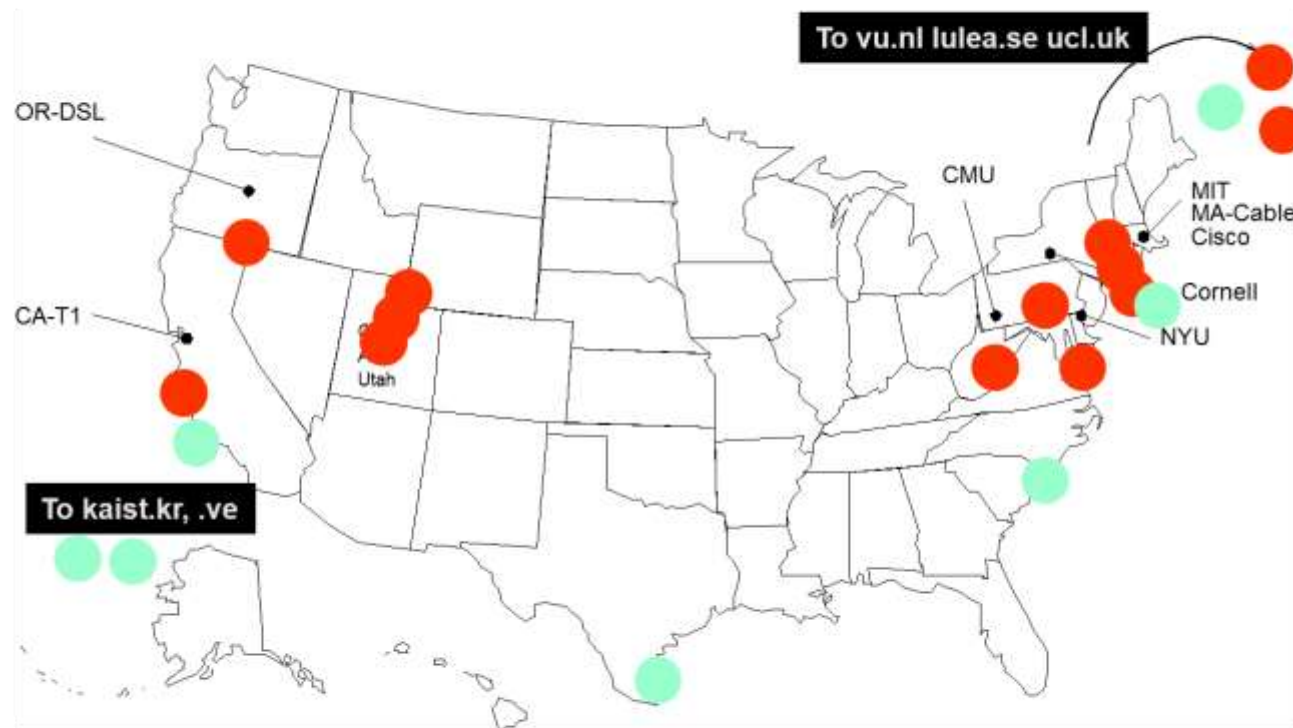




部署与测试

- 作者对有12个节点和132条路径的RON进行了64个小时的实际运行测试
- 测试过程中经历了32次严重的路径失效，平均每次的持续时间超过30分钟
- RON用平均不到20秒的时间就可以检测出这些失效情况并从中恢复
- RON还改善了应用的丢失率，延迟和吞吐率等性能指标

RON deployment (19 sites)

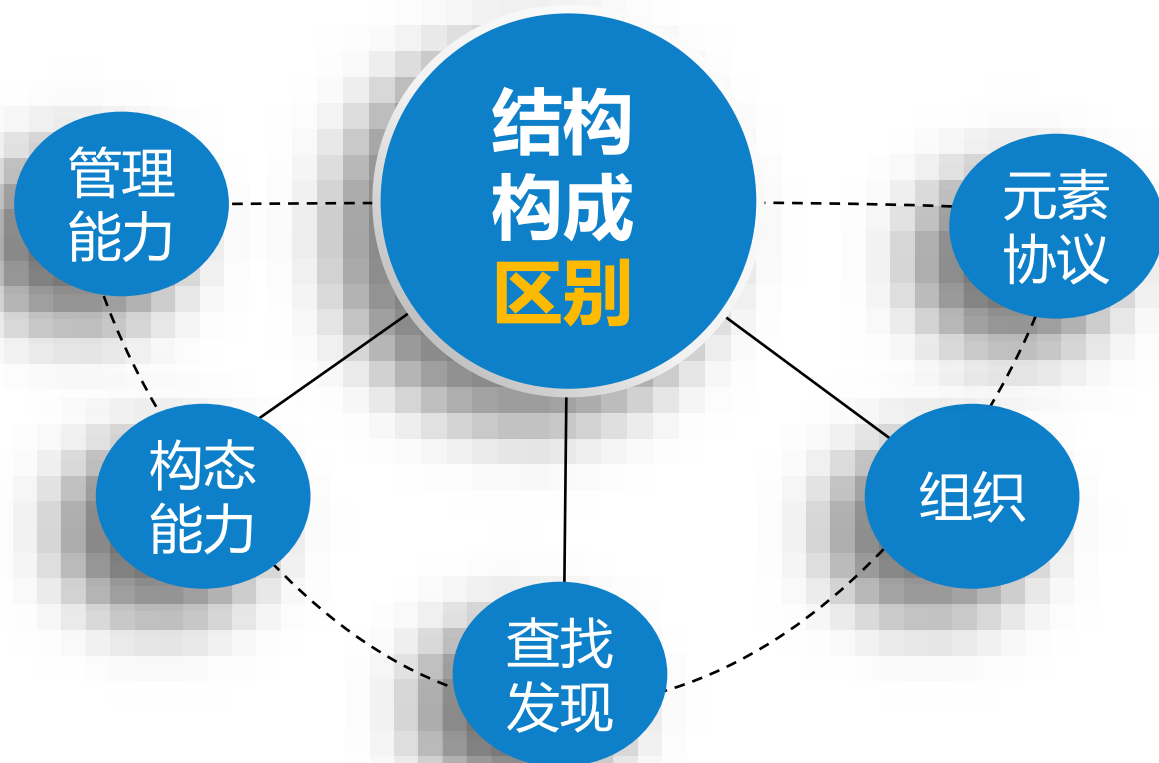


.com (ca), .com (ca), dsl (or), cci (ut), aros (ut), utah.edu, .com (tx)
cmu (pa), dsl (nc), nyu , cornell, cable (ma), cisco (ma), mit,
vu.nl, lulea.se, ucl.uk, kaist.kr, univ-in-venezuela



P2P vs C/S

P2P (Peer-to-Peer) 又称对等网络技术，也是建立在Internet之上的分布式Overlay网络，能够不依赖中心节点而只依靠对等协作实现资源发现与共享



相同



都能运行在不同 (Internet / Intranet) 平台



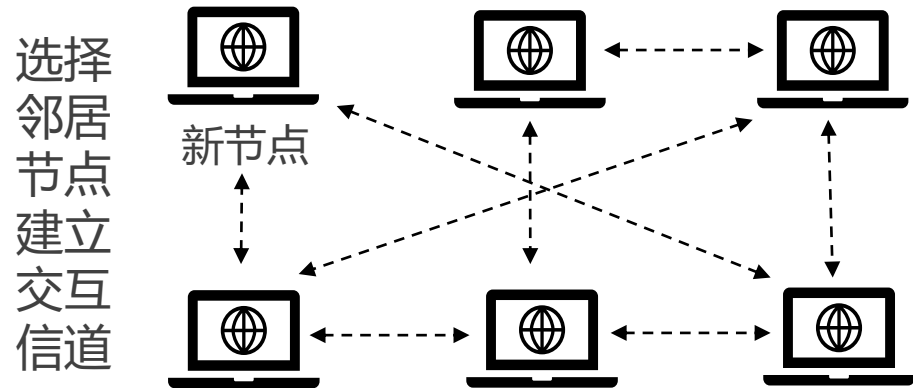
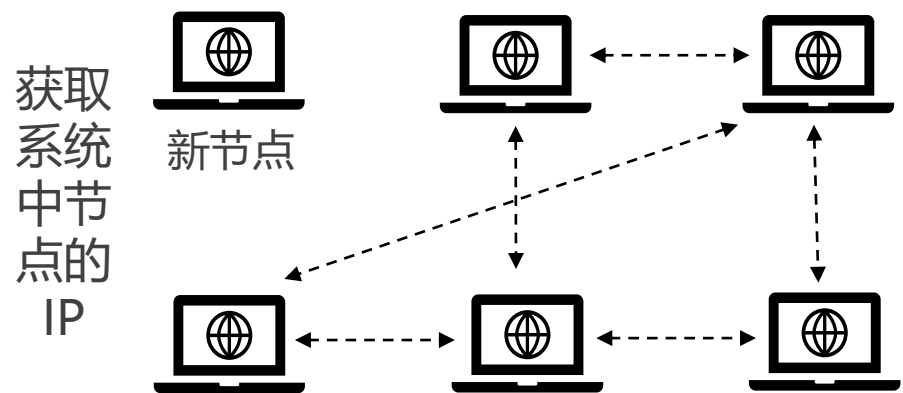
都能服务传统或新的应用:
eBusiness eServices ...



P2P网络的建立流程

P2P网络的建立主要包含网络节点发现和邻居节点维护两部分

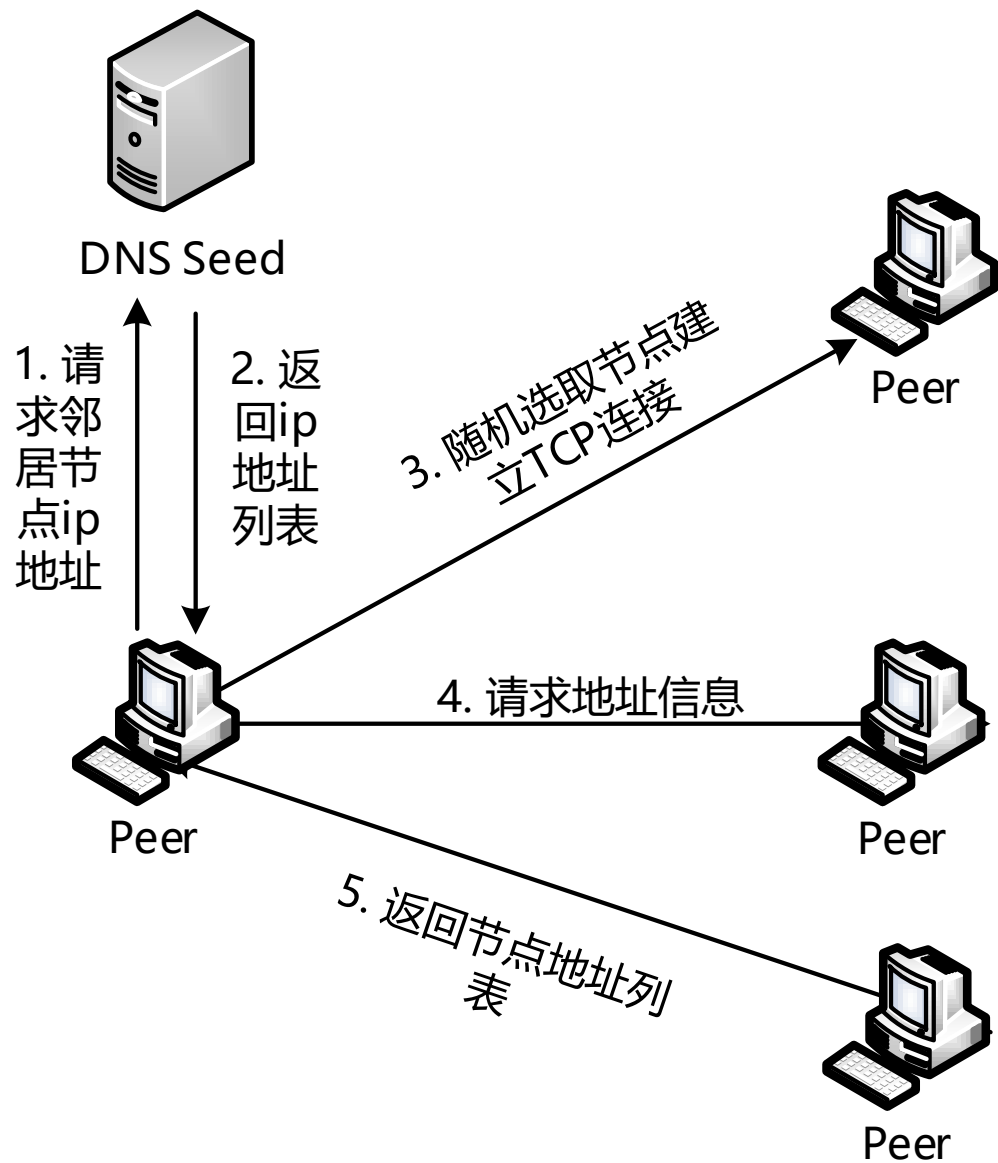
- 网络节点发现
 - 新加入的节点获取系统中其它节点的IP地址，并存储在本地的网络信息表中
- 邻居节点维护
 - 节点从本地的网络信息表中选择IP地址，并请求建立邻居关系





网络节点发现

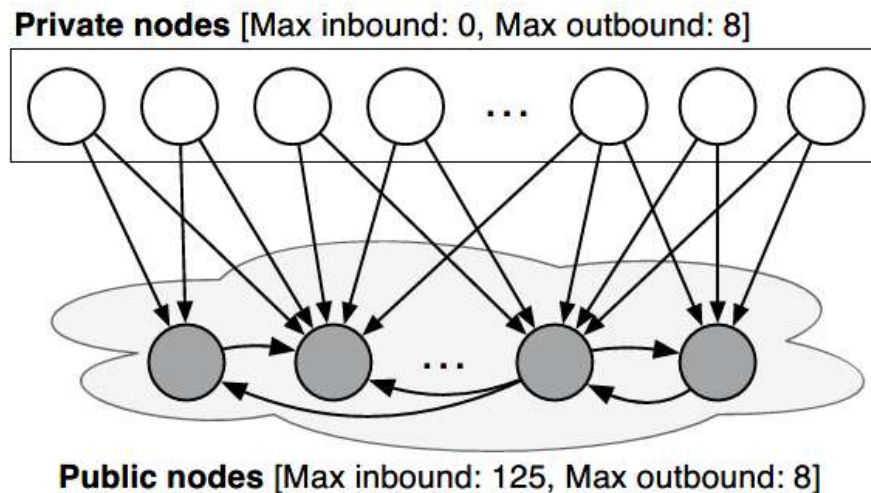
- 程序硬编码
 - P2P客户端程序中写入权威节点IP，用于客户端建立初始连接
- DNS Seed
 - 向Seed服务器请求节点IP地址
- 向邻居请求
 - 已存在邻居后，和邻居节点定期交换所发现的新IP地址
- 节点查询
 - 针对特定的节点id，查询其IP地址，主要在有结构化的P2P网络中存在



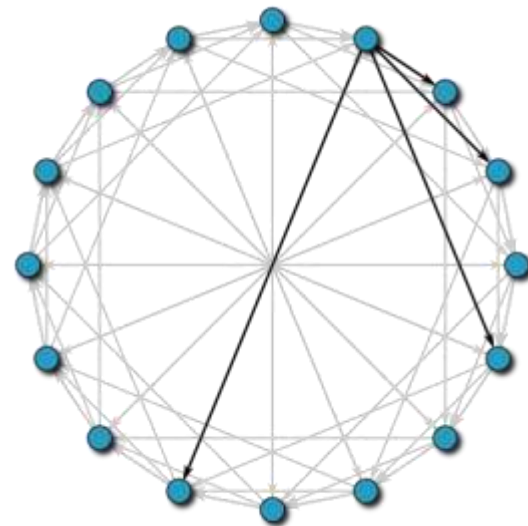


邻居节点维护

- 无结构化
 - 从发现的IP地址中，随机选择邻居
 - 通常限制节点的入向和出向连接数，入向指被请求建立的邻居关系，出向指主动请求建立的邻居关系



- 有结构化
 - 每个节点都有指定的id，并存在相应的公私钥用于id的认证
 - 节点基于id之间的距离按规定确定邻居节点
- 混合模式：有结构化P2P网络中，允许节点随机选择部分邻居节点

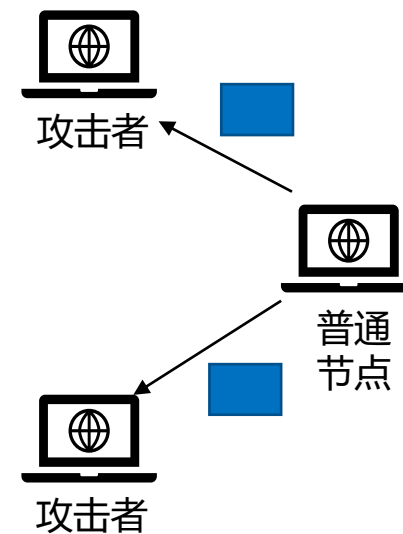
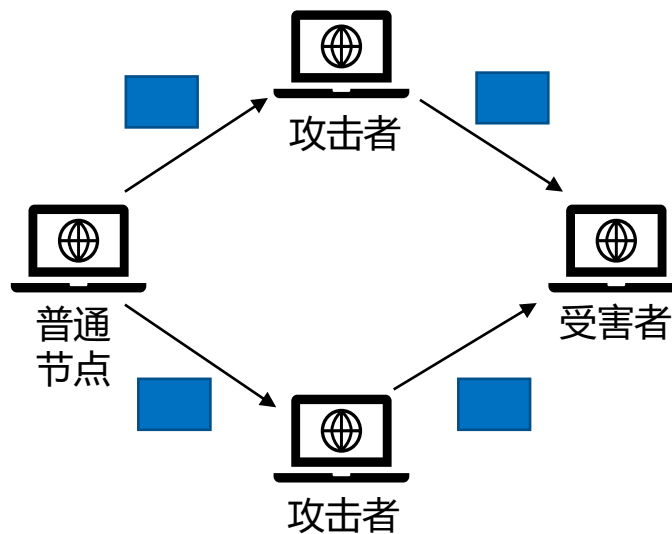
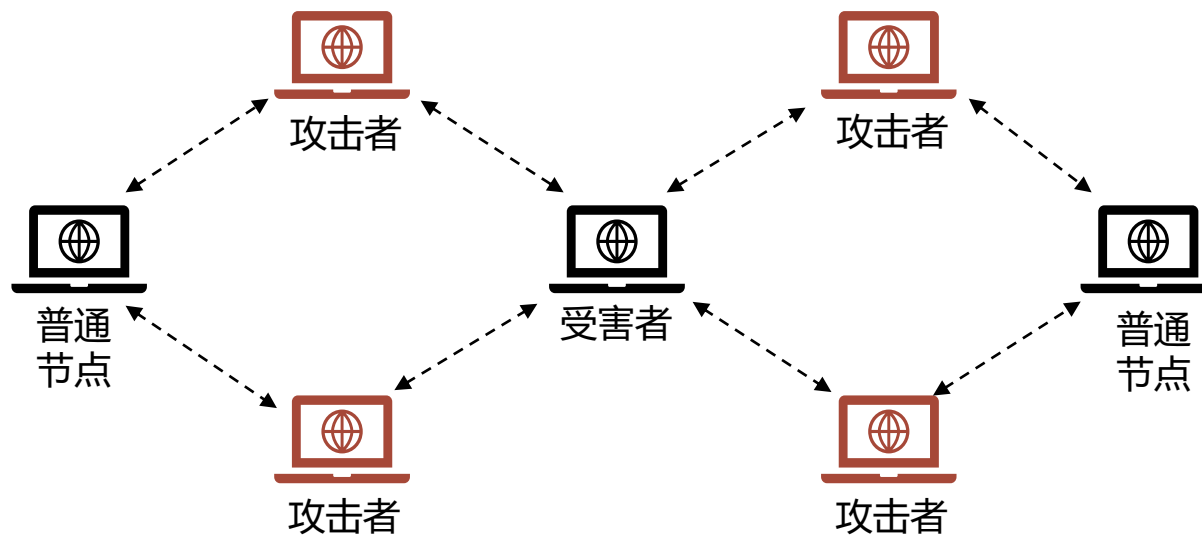




日蚀攻击

攻击目标：针对网络节点发现以及邻居节点维护两个过程进行攻击，使得受害节点的所有邻居节点都是由攻击者所控制的恶意节点

危害：攻击者可以选择性的转发对攻击者有利的消息给受害者，从而配合其它网络攻击；攻击者可以完全隔离受害者与网络中其它节点的信息交互



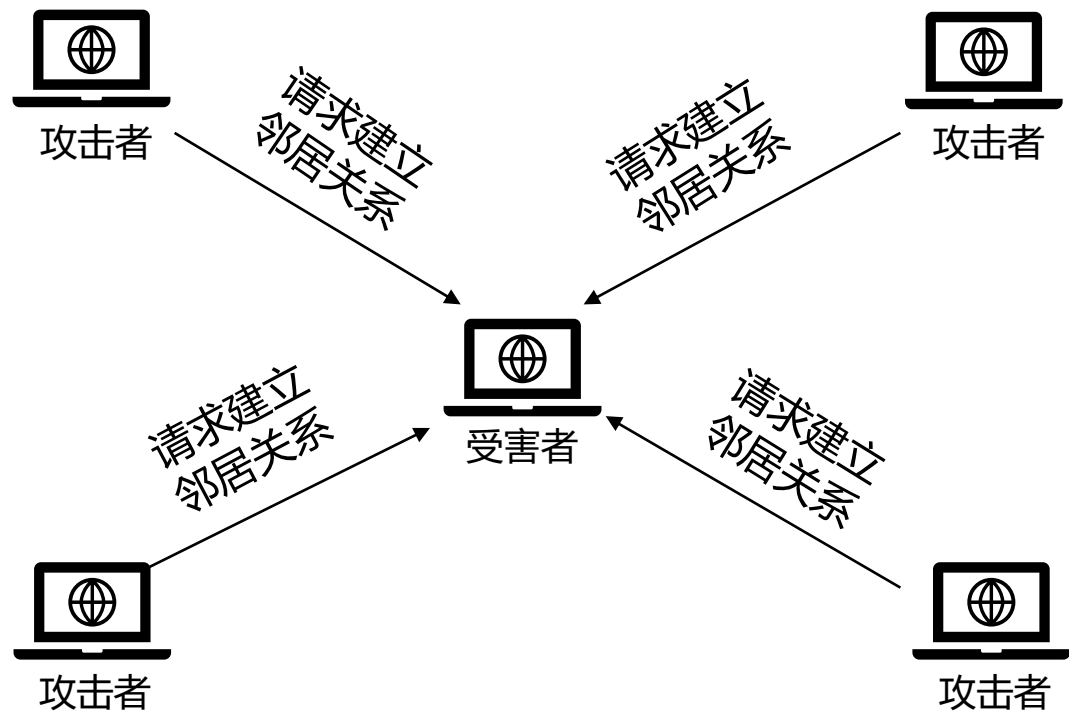


挤占入向连接

主要思路

- 攻击者控制大量节点主动向受害者请求建立邻居关系
- 若受害者已经和其它节点建立邻居关系，则可配合DDoS等其它网络攻击触发受害节点的重启，然后迅速挤占其入向连接

解决思路：
周期性更换邻居节点





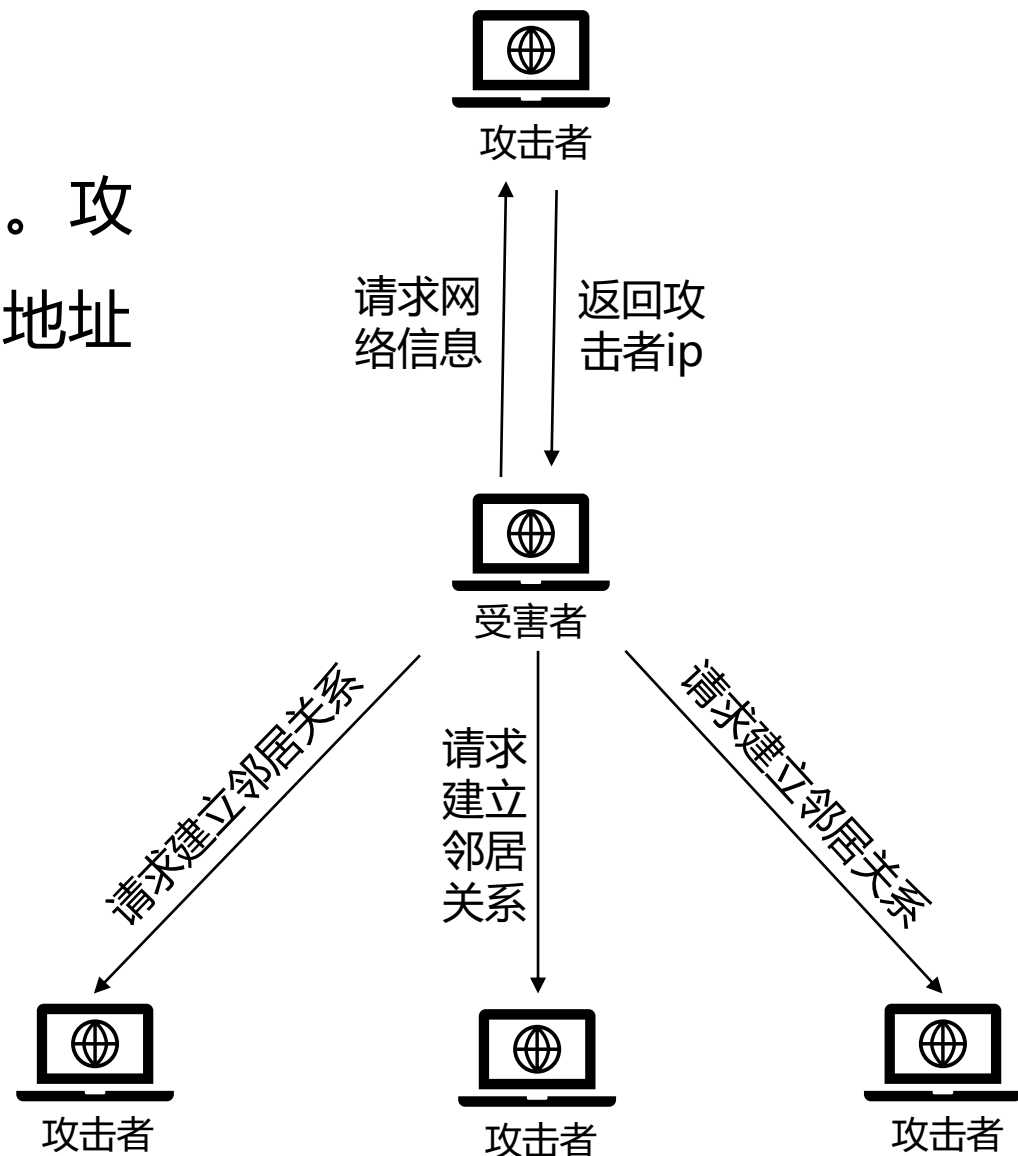
挤占出向连接

主要思路:

针对受害节点发现网络节点的过程进行攻击。攻击者向受害者发送大量由自己所控制的节点地址信息，等待受害者主动请求建立连接

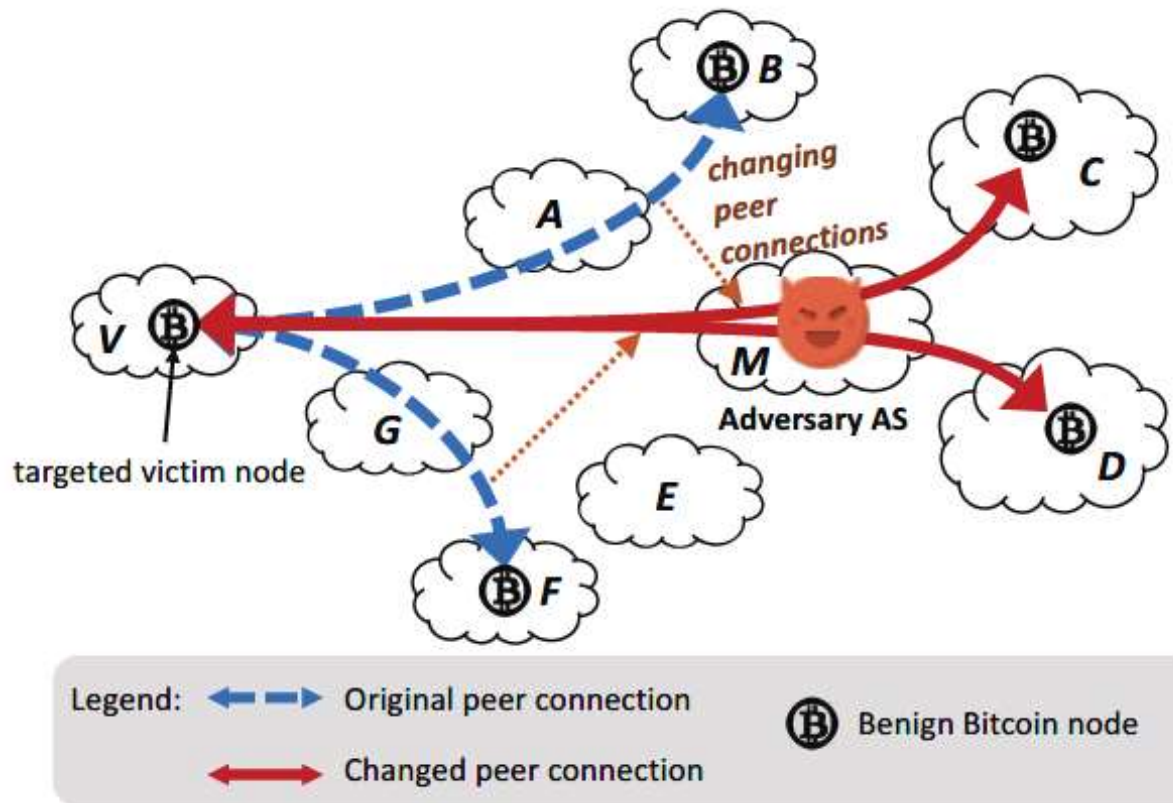
解决思路:

- 增加网络信息表项的大小
- 增加出向连接个数
- 选择少量不易受攻击的可信权威节点作为邻居节点





配合路由劫持攻击



主要思路:

- 向受害节点本地存储的节点地址列表注入大量影子IP（与受害节点间的通讯链路经过攻击者所控制的AS）
- 使受害节点与影子IP建立邻居关系，从而劫持受害者的所有邻居链路

解决思路:

邻居选择时将AS拓扑考虑进去

注. 图片来自来源:

Tran M, Choi I, Moon G J, et al. A stealthier partitioning attack against bitcoin peer-to-peer network[C]//2020 IEEE Symposium on Security and Privacy (S&P). IEEE, 2020: 894-909.



第2节 分布式算法

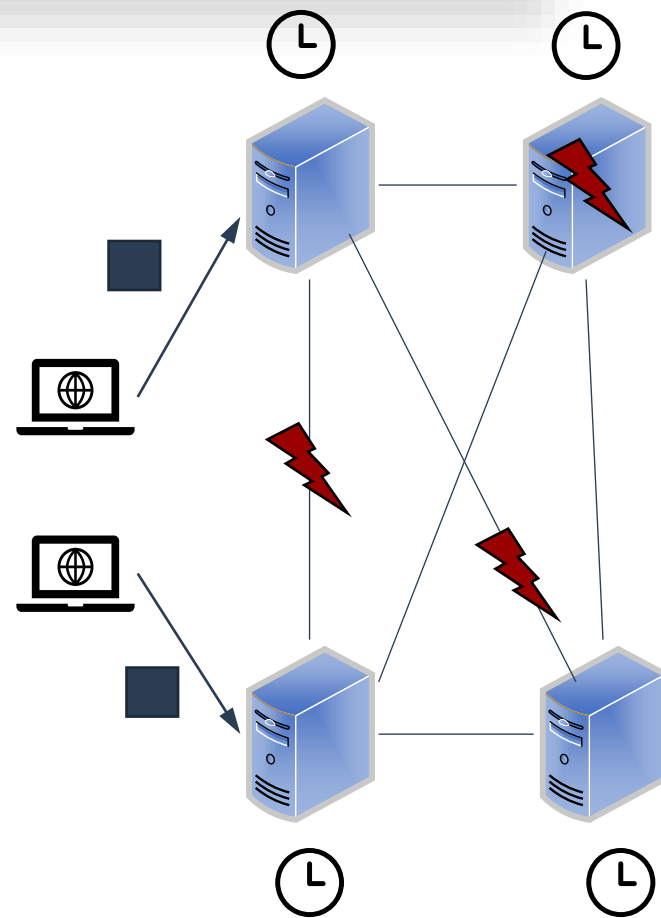
- ✓ 协作过程中的安全隐患
- ✓ 时钟同步算法
- ✓ 并发问题
- ✓ 故障容错的一致性算法



协作过程中的安全问题

分布式算法的目的是协调分布式组件之间的交互和动作，从而确保它们能够稳定协作实现一个共同的任务

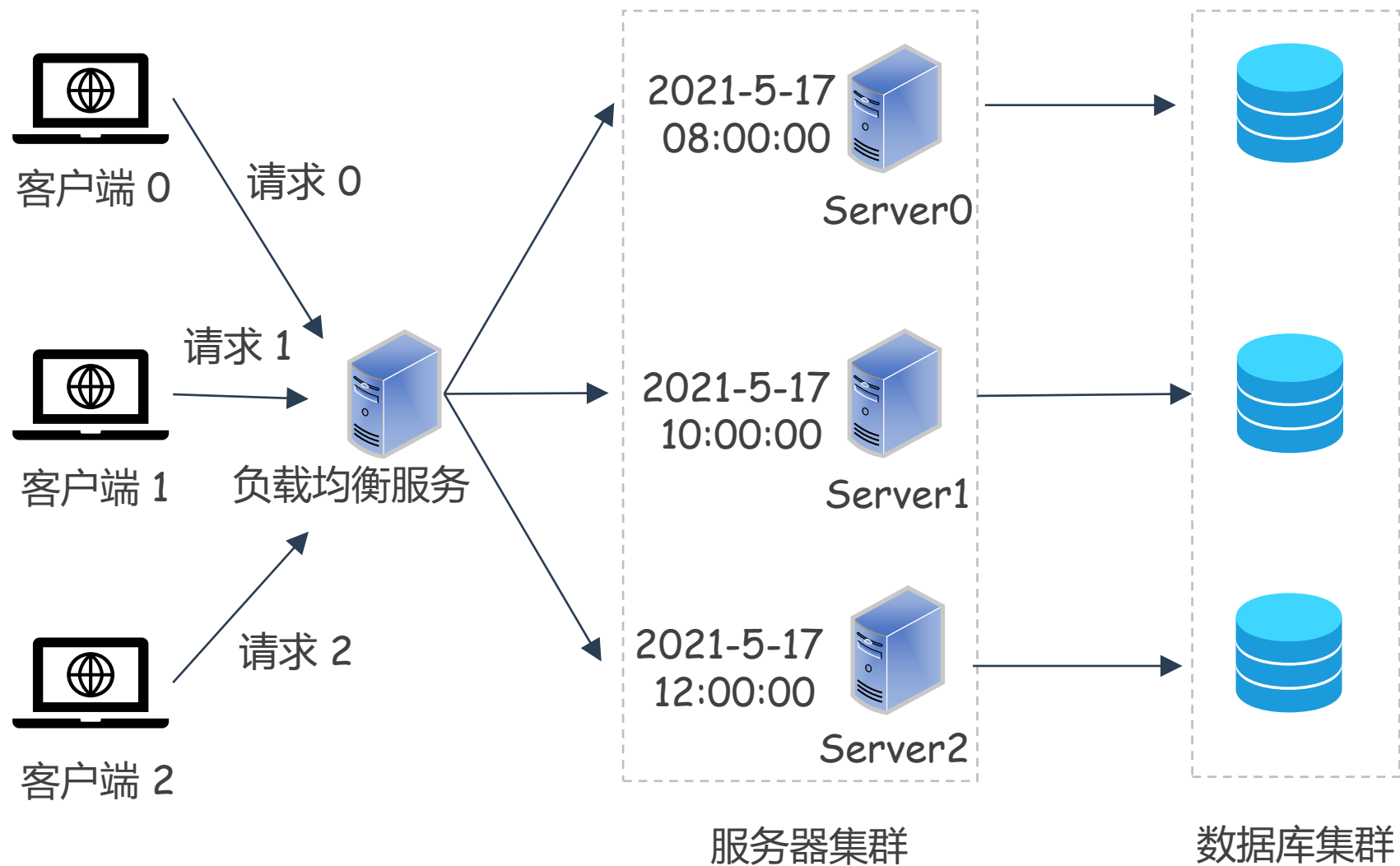
- 缺乏全局时钟：各节点本地时钟可能不同步
 - 物理时钟：基于时钟同步机制完成物理时钟同步
 - 逻辑时钟：基于共识对事件发生的顺序达成一致
- 并发：同一时刻，系统中的不同组件可能面临来自多个客户端的大量并发请求
- 故障：系统组件在协作过程中可能出现故障，从而导致信息的短暂丢失





时钟不同步可能引发的问题

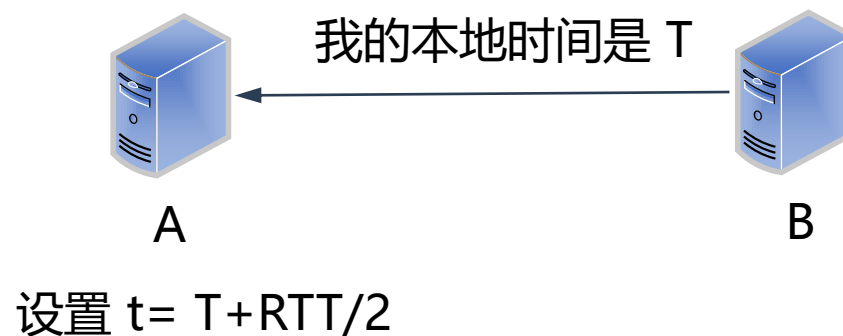
- 三个客户端几乎同时发送请求
- 请求由时钟不同步的服务器处理
- 服务器生成日志, 时钟不同步导致三个请求所标记的时间相差大





时钟同步算法

- **主要思想：**若节点A想与节点B完成时钟同步，则请求节点B发送本地时间T；然后预测A与B的网络时延RTT；之后将时间设置为 $T + RTT/2$ ；RTT预测的越精确，时钟同步的误差越小



- **主要方式**
 - Cristian算法：使用一个时间服务器，其它节点基于该服务器完成时钟同步，主要用于企业网内部
 - Berkeley算法：主要用于分布式集群系统中各节点之间时钟同步，主要用于企业网内部
 - NTP协议：主要运用于互联网，确保网络节点之间能够时钟同步



Cristian算法

基于一个接收UTC（通用协调时间）信号的外部时间服务器完成时钟同步

- 当服务器 P 收到数据包时，本地时钟为 t'' ，可得消息来回延迟为：

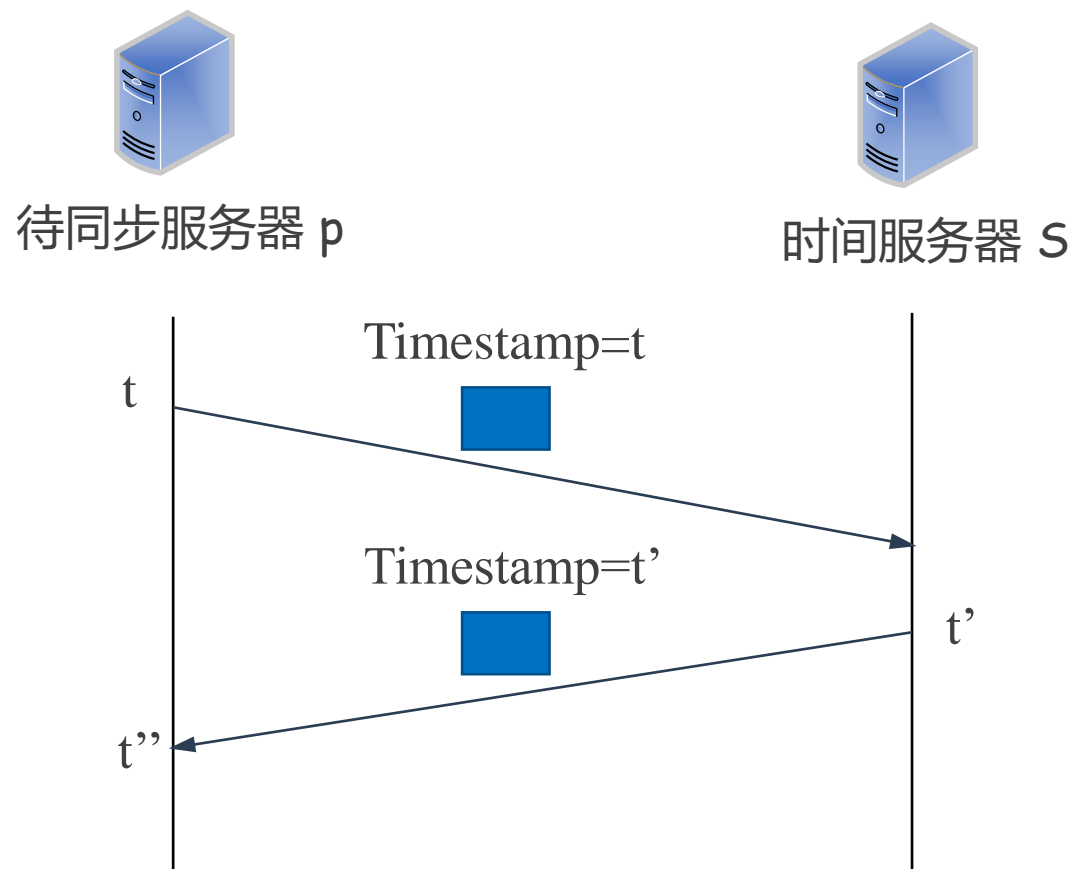
$$T_{round} = t'' - t$$

- 在服务器 P 看来，当 S 收到消息时，P 本地预计时间为 $t + T_{round}/2$ ，所以 P 和 S 的时间差为：

$$t_s - t_p = t' - (t + T_{round}/2)$$

- 所以 P 在收到消息后将时钟设置为：

$$t' + (t'' - t)/2$$





Berkeley算法

主要用于分布式集群服务器内部完成时钟同步

以服务器 k 为例说明Berkeley算法的具体流程：

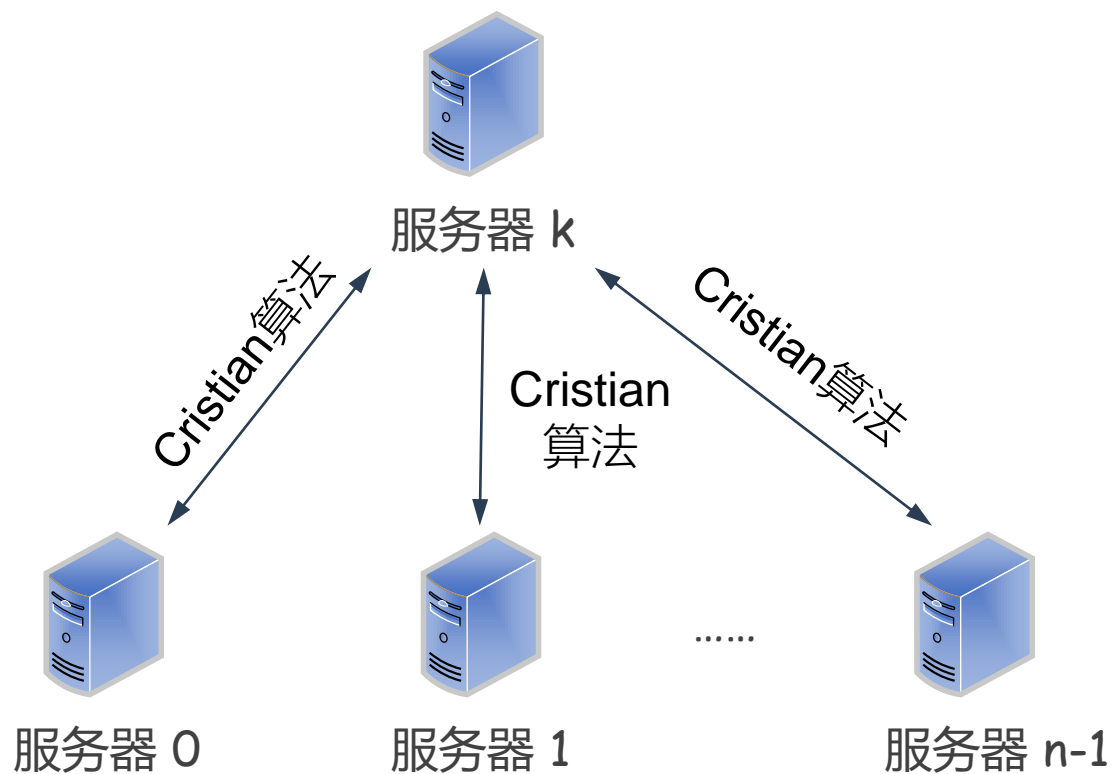
- 服务器 k 与所有其它服务器间分别运行一次Cristian算法，得到：

$$t_0, t_1, \dots, t_{n-1}$$

- 服务器 k 与所有其它服务器间分别运行一次Cristian算法，得到：

$$t = \sum_i t_i / n$$

- 服务器k根据得到的平均值给各个服务器发送时钟偏移值



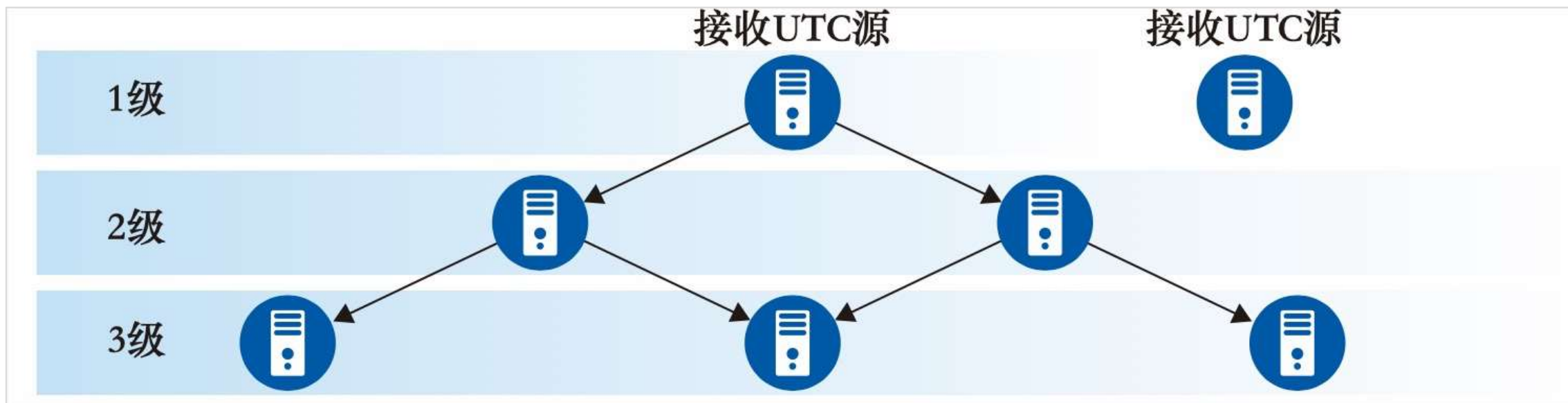


网络时间协议 (NTP)

NTP协议定义了时间服务的体系结构，以及如何在互联网上发布时间信息

- 主要目标：
 - 互联网的用户能够精确地与UTC同步
 - 部分服务器故障，用户依然能同步时钟
 - 避免恶意攻击者干扰时钟同步过程

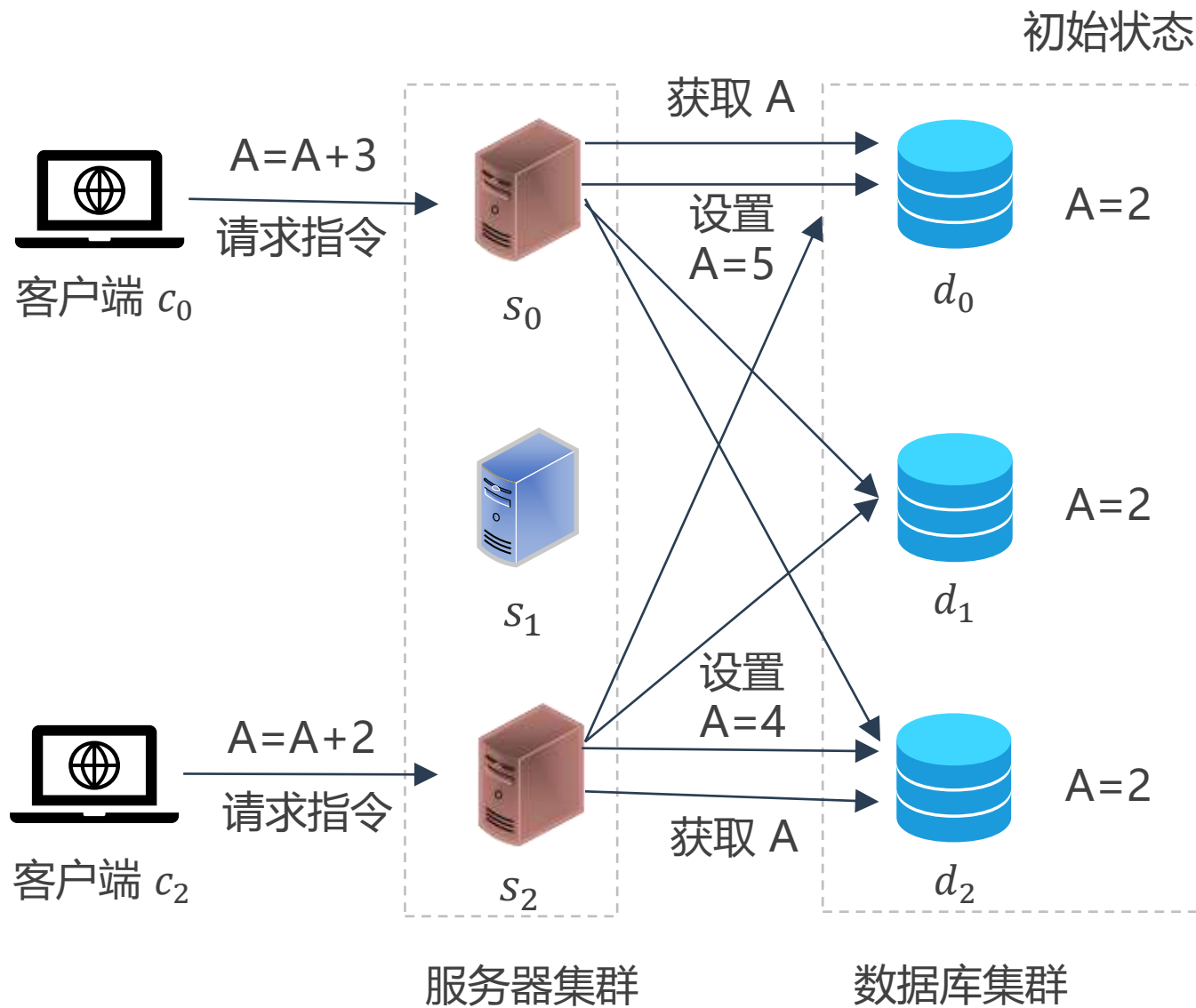
- 整体架构
 - 采用分层架构，1级服务器接收UTC源
 - 基于认证确保时间信息来自权威服务器
 - 同步源故障后，选择其它源作为替代





并发问题：破坏一致性

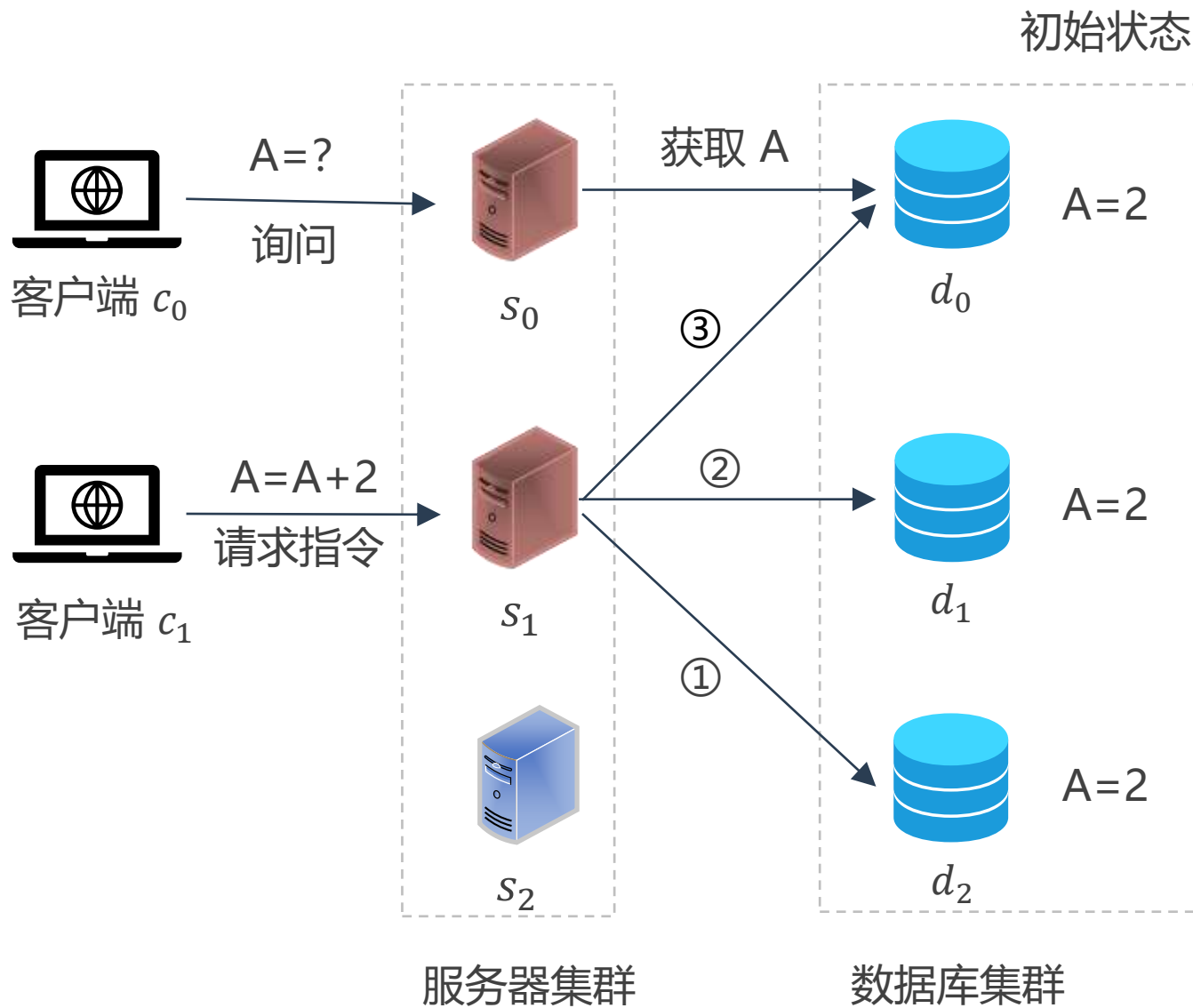
- 假设：状态 A 在三个数据库冗余存储，不存在并发控制措施
- 执行步骤
 - c_0 向 s_0 请求将 A 的值加3
 - c_2 向 s_2 请求将 A 的值加2
 - s_0 和 s_2 分别向 d_0 和 d_2 读取 A 的值并完成计算
 - s_0 和 s_2 依次向三个数据库更新 A 的值
- 由于指令的并发执行，可能导致三个数据库 A 的值不一致





并发问题：指令之间相互影响

- 假设：状态 A 在三个数据库冗余存储，不存在并发控制措施
- 执行步骤
 - c_0 向 s_0 请求 A 的值
 - c_1 向 s_1 请求将 A 的值加2
 - s_1 依次将三个数据库中 A 的值修改为4
 - s_0 向数据库 d_0 读取 A 的值
- 由于指令的并发执行， s_0 读取的值可能为2，也可能为4





事务与分布式事务

事务是原子的一组请求，这组请求不受其它客户端干扰，并且要么全部完成，要么不对服务器造成任何影响

- 事务需要满足ACID特性
 - 原子性 (Atomicity)：操作必须全部完成，或者不留下任何结果
 - 一致性 (Consistency)：将系统从一个一致状态转换到另一个一致状态
 - 隔离型 (Isolation)：每个事务的执行不受其它事务影响
 - 持久性 (Durability)：一旦事务完成，其效果将被保存到持久存储中

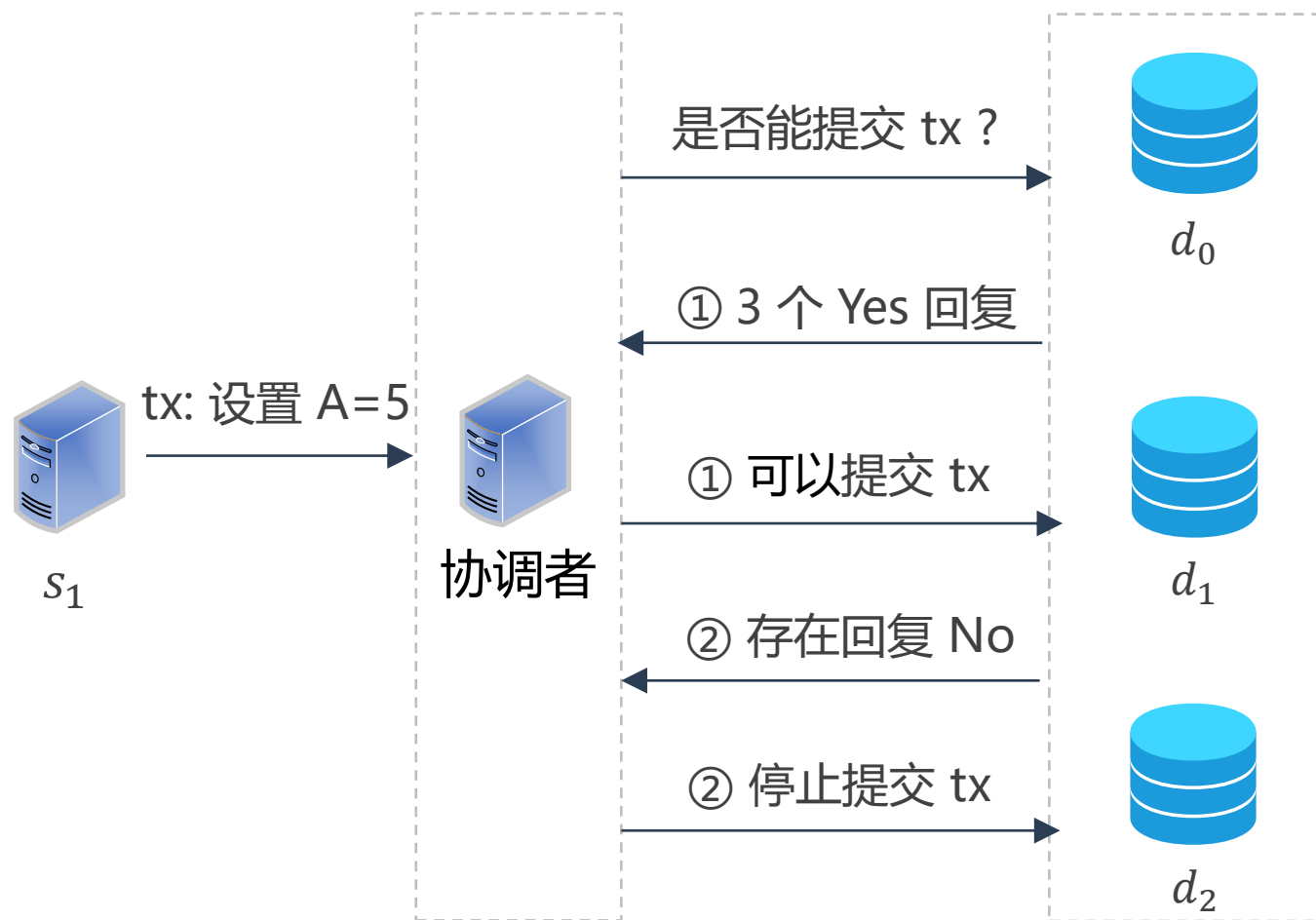
分布式事务是访问由多个服务器管理的对象的事务



两阶段提交确保一致性

假设：状态A在三个数据库冗余存储，存在一个协调者引导二阶段提交的执行

- 执行步骤
 - s_1 向协调者请求将A设为5
 - 协调者向所有存储状态A的数据库询问是否可提交事务
 - 数据库若判断不冲突，则返回Yes；否则返回No
 - 当协调者收到所有数据库的Yes，通知它们提交该事务；如果收到No，则通知停止



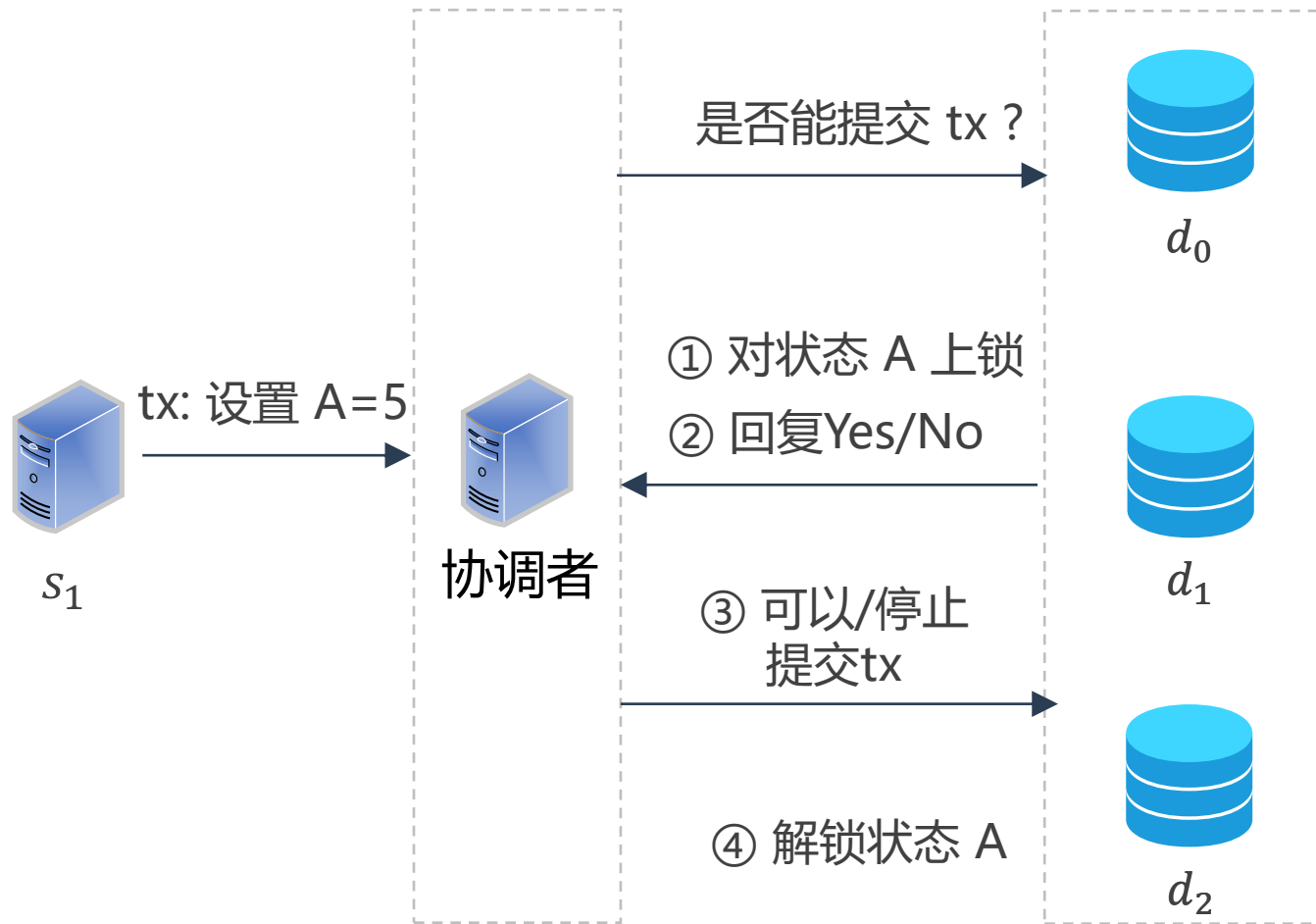


基于锁机制确保隔离性

假设：状态A在三个数据库冗余存储，存在一个协调者引导二阶段提交的执行

- 执行步骤

- s_1 向协调者请求将A设为5
- 协调者向存储状态A的数据库询问是否可提交事务
- 数据库若判断不冲突，则返回Yes，并对状态A上锁
- 数据库等待协调者的通知，若收到Do commit，完成对状态A的设置并解锁状态A；若收到Abort，则直接解锁状态A





讨论

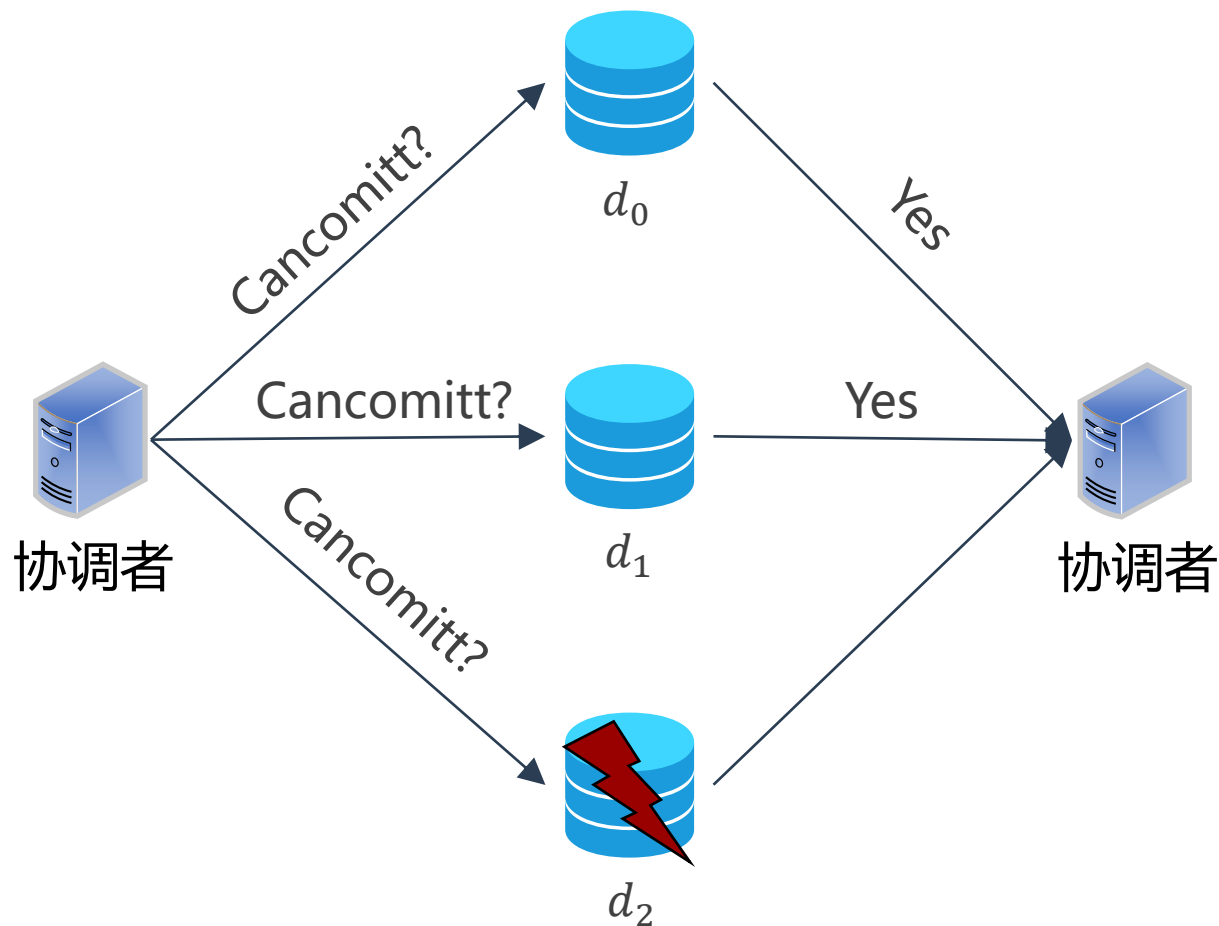
在两阶段提交过程中出现故障怎么办？



数据库节点故障

存在一个协调者和三个存储相同状态变量A的数据库，其中一个数据库出现故障

- 执行步骤
 - 协调者向三个数据库询问是否可以提交一个更改A值的事务
 - 数据库 d_0 和 d_1 向协调者返回Yes
 - 数据库 d_2 出现故障，没有反应
 - 协调者无法收到足够的Yes，陷入无限等待，或者采用计时机制，超时后通知所有节点停止提交

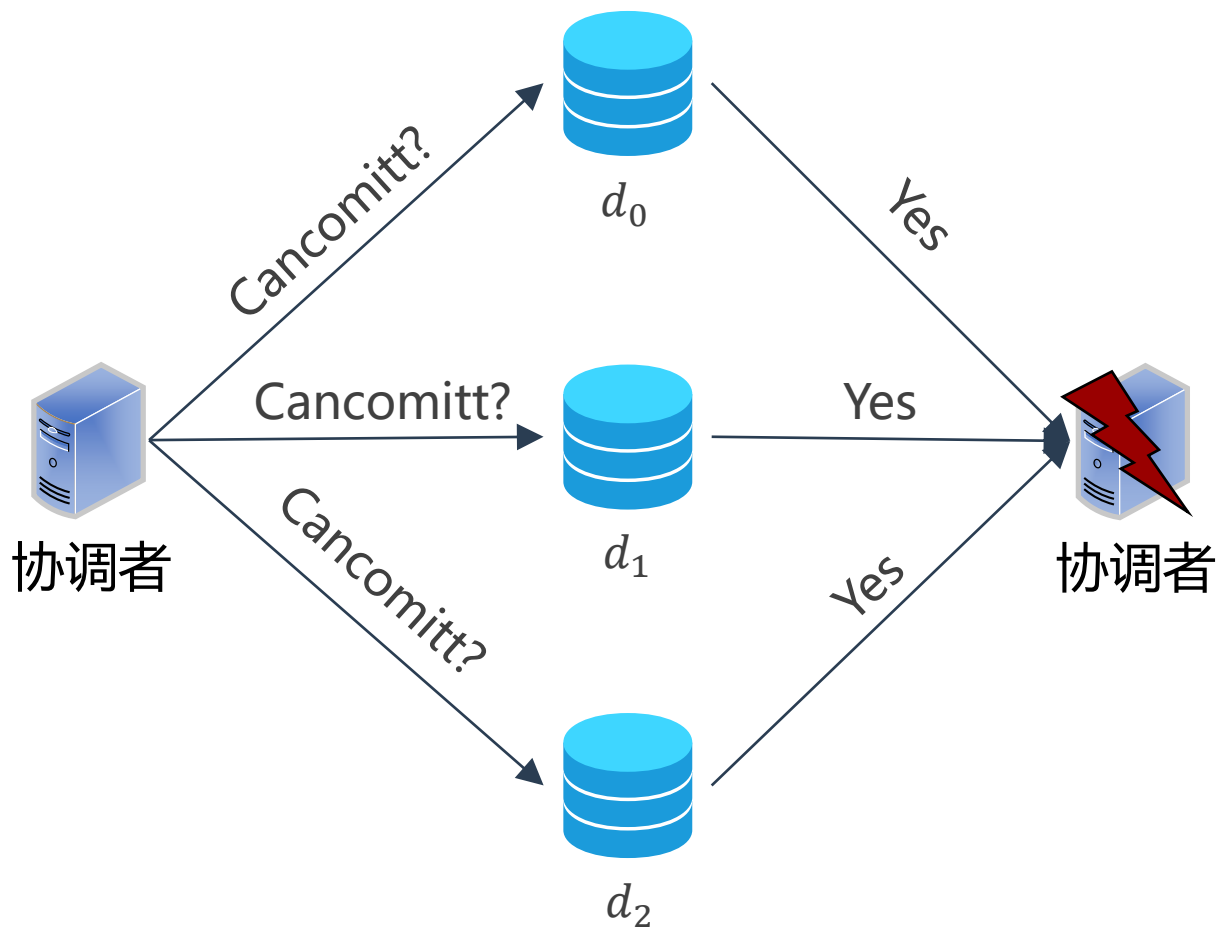




协调者故障

存在一个协调者和三个存储相同状态变量A的数据库

- 执行步骤
 - 协调者向三个数据库询问是否可以提交一个更改A值的事务
 - 三个数据库都向协调者返回Yes
 - 此时协调者出现故障
 - 数据库一直无法等到协调者的确认提交或停止提交消息，因此一直处于等待状态，且相关资源一直处于上锁状态

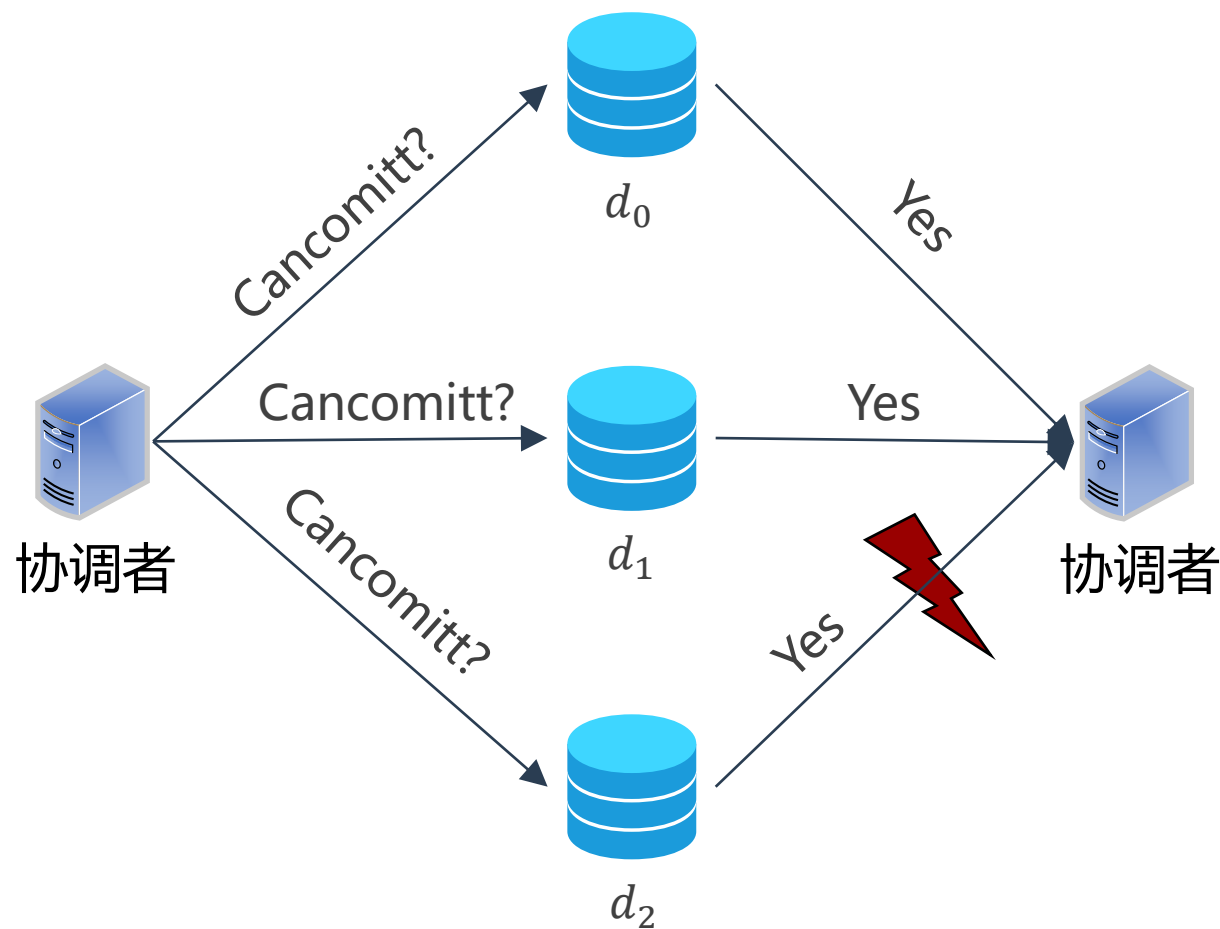




网络拥塞/链路故障

存在一个协调者和三个存储相同状态变量A的数据库

- 执行步骤
 - 协调者向三个数据库询问是否可以提交一个更改A值的事务
 - 三个数据库都向协调者返回Yes
 - 数据库 d_2 与协调者之间的通信链路故障，导致Yes消息没被收到
 - 协调者无法收到足够的Yes，陷入无限等待，或者采用计时机制，超时后通知所有节点停止提交





无法容错的根源

在简单的二阶段提交协议中，必须所有相关节点都进行回应后才成功提交，因此协议无法容忍任何故障



印度

我要当常任！

提案

(同意，同意，同意，
同意，反对)

必须五常国全部同意才能通过，只要保持每次投票的同意票数为4，提案永远无法通过

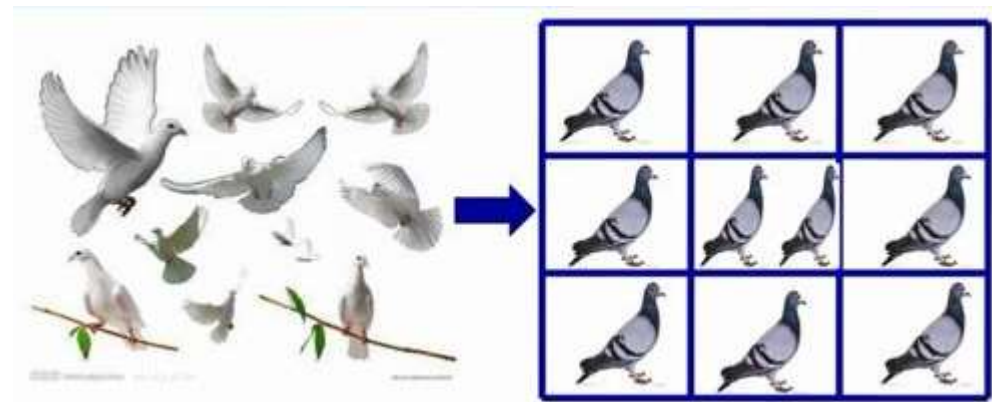


五常国



Quorum设计准则

- **鸽巢原理**：如果 $k+1$ 个鸽子要放在 k 个鸽巢中，那么至少有一个鸽巢存在两个鸽子



- 假设 N 个副本，更新操作在 W 个副本中更新成功之后，才认为此次更新操作成功
- 对于读操作而言，至少需要读 R 个副本才能读到此次更新的数据。其中， $W+R>N$ ，即 W 和 R 有重叠，一般， **$W+R=N+1$**
- 假设系统中有5个副本， $W=3$ ， $R=3$ 。初始时数据为 $(V1, V1, V1, V1, V1)$ 当某次更新操作在3个副本上成功后，认为此次更新操作成功，数据变成： $(V2, V2, V2, V1, V1)$ 最多只需要读3个副本，一定能够读到 $V2$ (此次更新成功的数据)



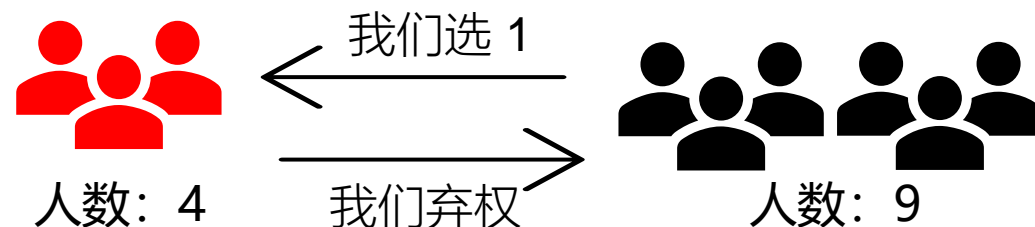
Quorum设计准则

基本规则：分布式系统每阶段提案中，诚实节点最多只对一个提案投一次票



$q=8$ ❌

$q=9$ ✅



$q=10$ ❌

$q=9$ ✅

- 目的1-安全性：Quorum值 q 应该足够大，避免通过两个**矛盾提案**

解决方案： 针对两个提案可能产生的最高总投票数 $\leq 2*q - 1$

- 目的2-活性：Quorum值 q 应该足够小，避免错误节点不投票使提案**无法通过**

解决方案： $q \leq$ 最低可能产生的投票数； $q >$ 拜占庭节点个数



故障容错算法-Paxos

134 • L. Lamport

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxos civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

The author does give a brief discussion of the Paxos Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lamport [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

Keith Marzullo
University of California, San Diego



Leslie B. Lamport

美国著名的计算机科学家，微软研究院首席研究员，著名的排版系统LaTeX的作者，拜占庭将军问题和Paxos算法的提出者，2000年获得了Dijkstra奖，2012年获得冯·诺依曼奖，**2013年获得图灵奖**



故障容错算法-Paxos

这个世界上只有一种一致性算法，那就是Paxos，其它的算法都是残次品

——Google Chubby的作者Mike Burrows

- 问题场景：
 - 分布式数据库系统中，各节点初始状态一致
 - 部分节点可能出现故障
- 主要目的：
 - 所有正确节点按相同顺序执行相同的操作指令
 - 所有正确节点得到一致的本地状态
- 应用实例：
 - Cassandra, Google Spanner, Google Chubby



Paxos的主要思想

存在有Leader和无Leader两种模式，基于Quorum机制确保一致性和容错性

Paxos算法中的3种角色，一个节点可同时担任多个角色，通常情况下由客户端担任提议者，服务端担任接受者和学习者

- Proposer(提议者), Acceptor(接受者), Learner(学习者)

Paxos的基本流程

- 提议者为交易分配一个序号，从而形成提案，然后将其发送给所有的接受者
- 接受者选择需要接受的提案，然后对提议者进行回应
- 当提议者收到超过半数接受者的回应后，则通知接受者最终接受该提案
- 接受者接受提案后告知学习者，学习者收到多数接受者信息后学习该提案

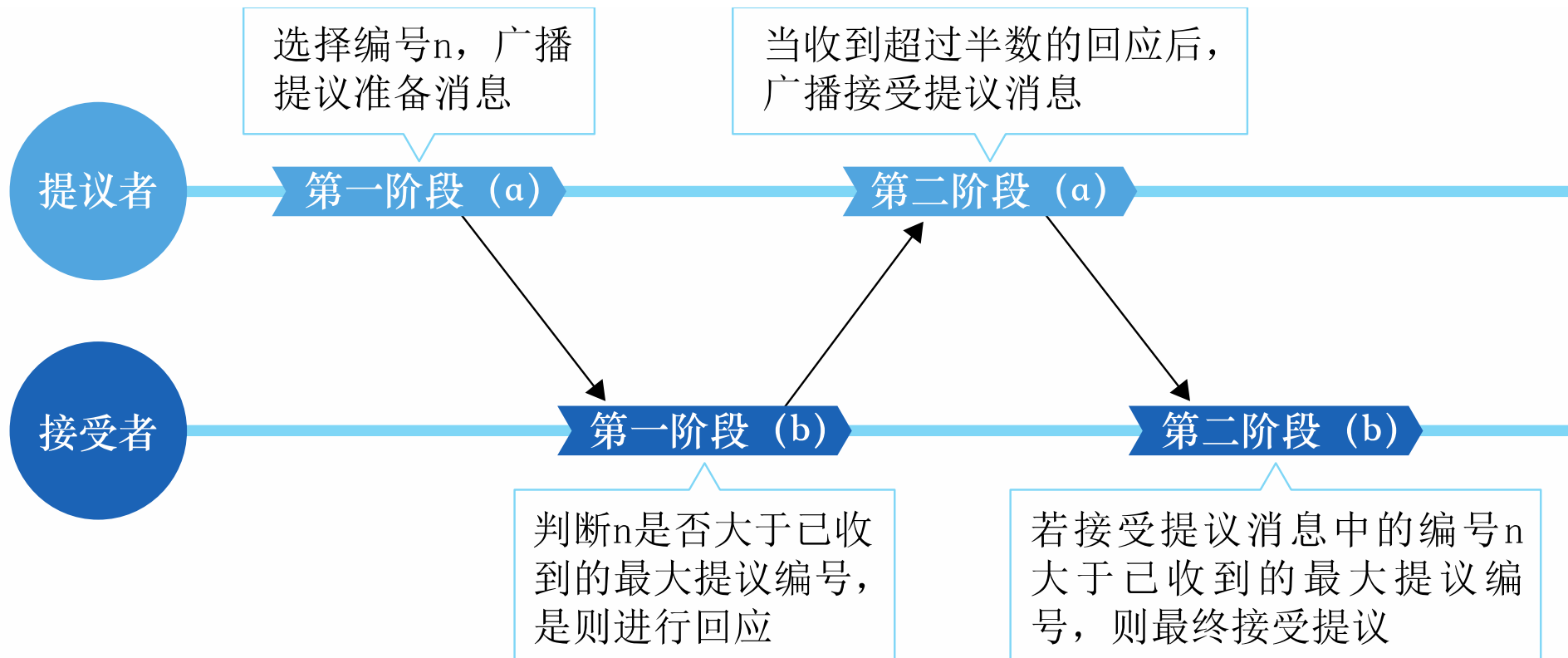
注. 详情可参考:

Lamport L. Generalized consensus and Paxos[J]. 2005.



无Leader模式下的运行流程

一个理想的 共识流程



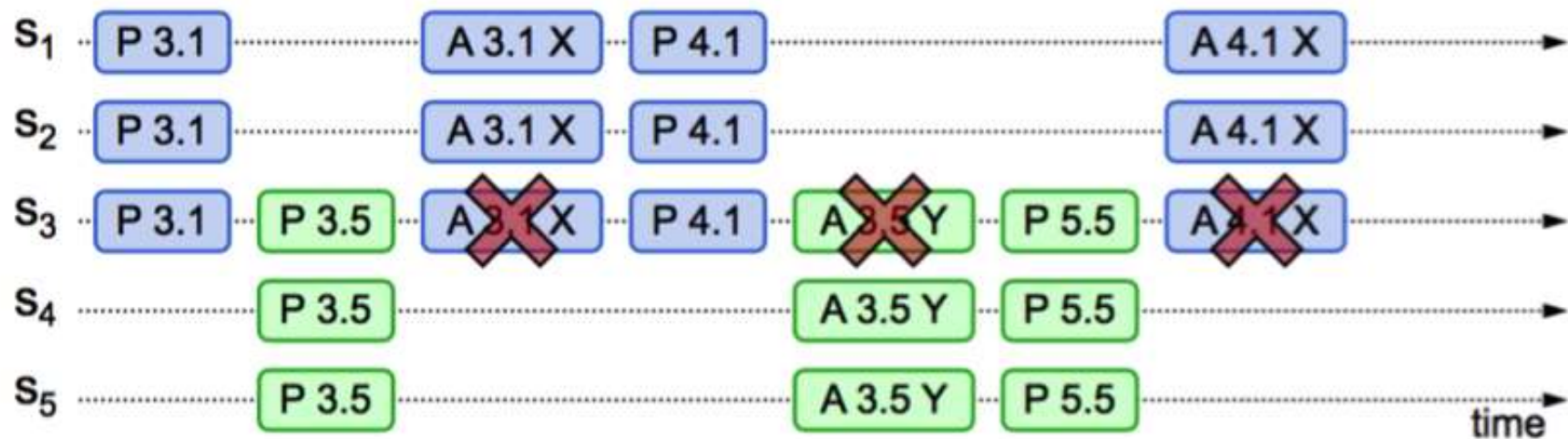
- 实际情况：多个提议者存在，每个提议者单独提议各自的提案；接受者面临不同提议者的提案，可能需要不断接受提议编号更大的提案
- 共识达成条件：提案完成前三次交互之前无更大提议号提案出现



一种永远无法达成共识的极端场景

每个提议者独立提议时，不仅效率低下，还可能产生活锁，即不断产生冲突

- Competing proposers can livelock:



解决思路：在提议者中选取一个领导者进行提议，其它提议者不参与提议

注. Paxos及更多改进的详细介绍可参考：

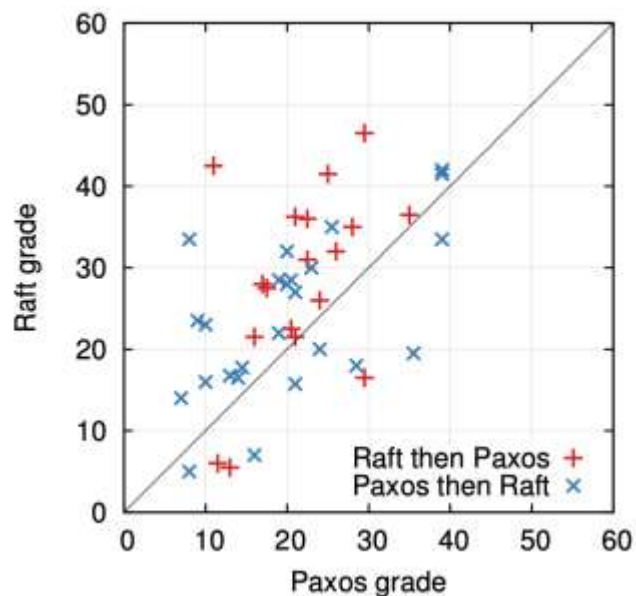
Lamport L. Fast paxos[J]. Distributed Computing, 2006, 19(2): 79-103.



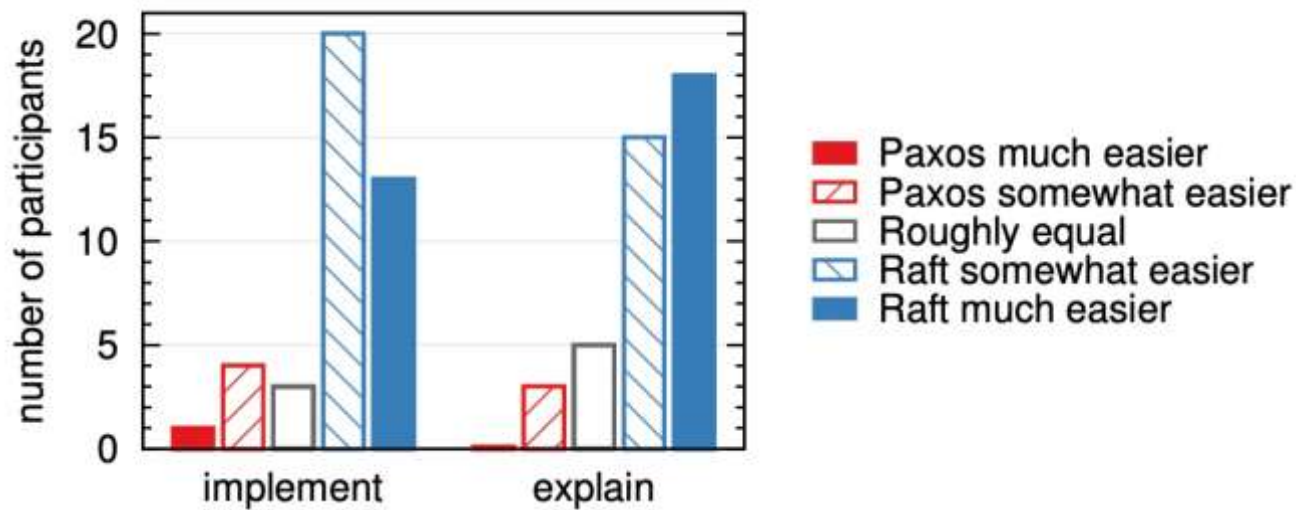
故障容错算法-Raft

目标：在保证容错的基础上，设计一个更容易理解的故障容错算法。

——Diego Ongaro and John Ousterhout



对参与学生的测试成绩



对参与学生的调查问卷

结论：根据测试结果和调查问卷，说明Paxos的学习难度要远高于Raft



故障容错算法-Raft

Paxos Made Simple

Leslie Lamport

01 Nov 2001

Abstract

The Paxos algorithm, when presented in plain English, is very simple.

Contents

1	Introduction	1
2	The Consensus Algorithm	1
2.1	The Problem	1
2.2	Choosing a Value	2
2.3	Learning a Chosen Value	6
2.4	Progress	7
2.5	The Implementation	7
3	Implementing a State Machine	8
	References	11



Raft的主要思想

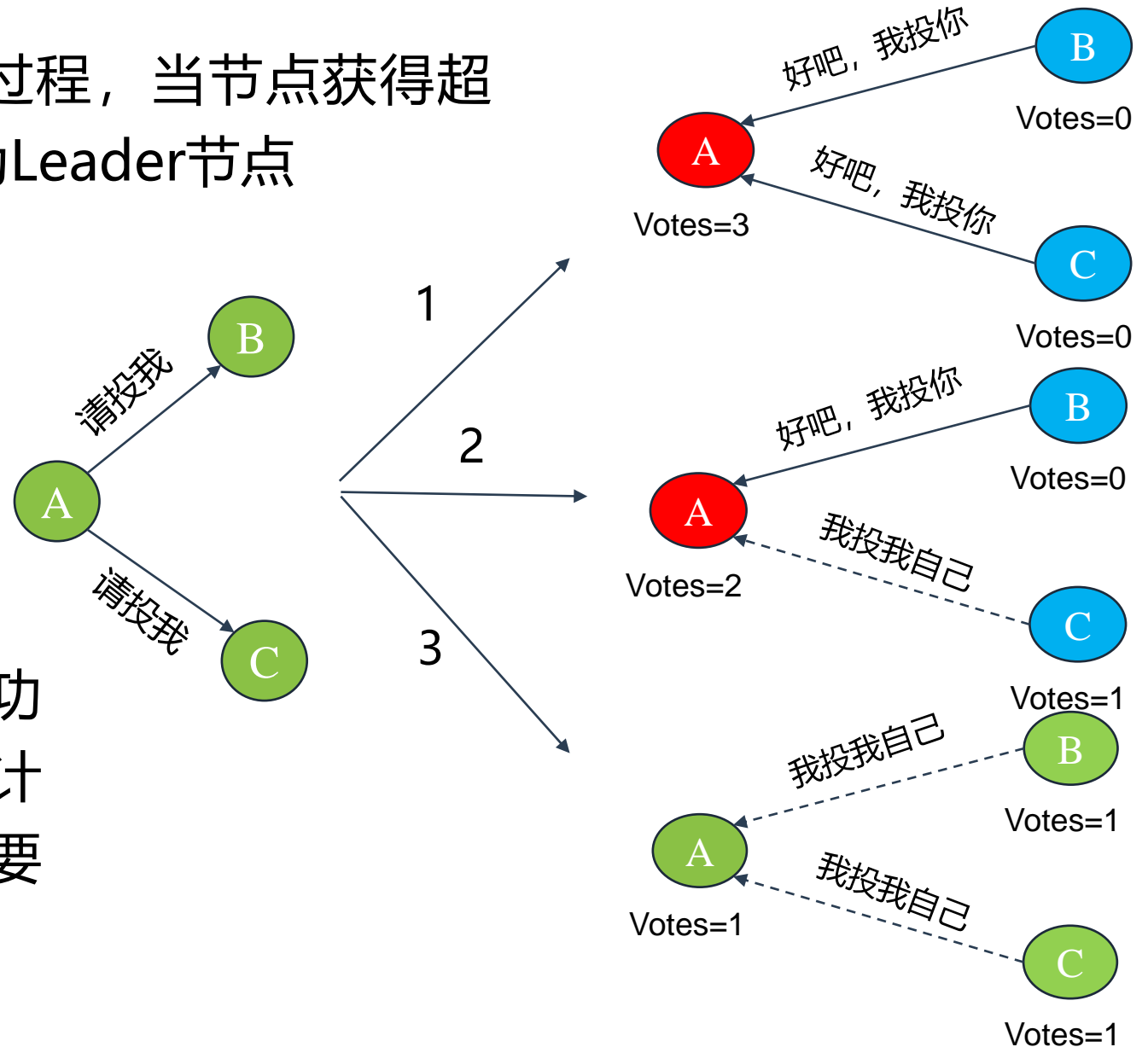
Raft基于Leader完成共识过程，并基于Quorum机制确保一致性和容错性

- **在Raft中，服务器可以扮演下面角色之一：**
 - Leader: 处理所有客户端交互，日志复制等，一般一次只有一个Leader
 - Follower: 类似选民，完全被动
 - Candidate: 候选人，可能被选为新的Leader
- **一般状态下Raft运行的基本流程：**
 - 所有交易被发往Leader，由Leader为其分配序号发给所有Follower
 - 当收到超过半数Follower的回应时，提交并执行交易，然后回复客户端
 - Follower提交交易后，在后续的交互中通知Follower提交相同的交易



Raft运行流程

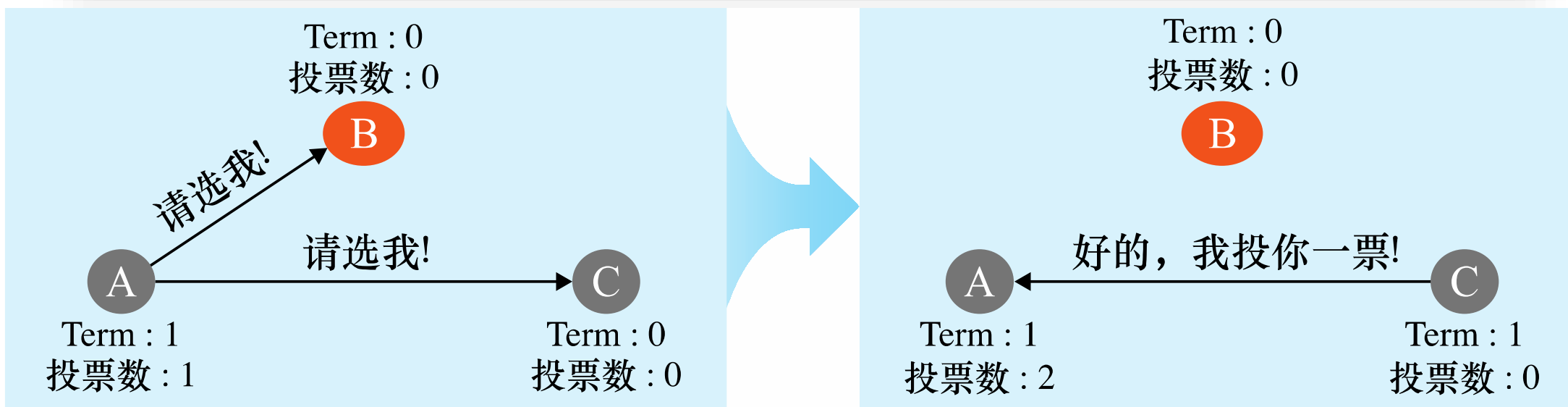
- Raft首先基于投票完成选举过程，当节点获得超过半数投票时，则成功竞选为Leader节点
- 当竞选过程中出现严重分票导致无节点成为Leader时，节点触发超时机制，并进入下一轮Leader选举
- 为了确保必然有节点最终成功当选Leader，节点采用随机计时机制，每个节点超时所需要等待的时间不一样





Raft运行流程

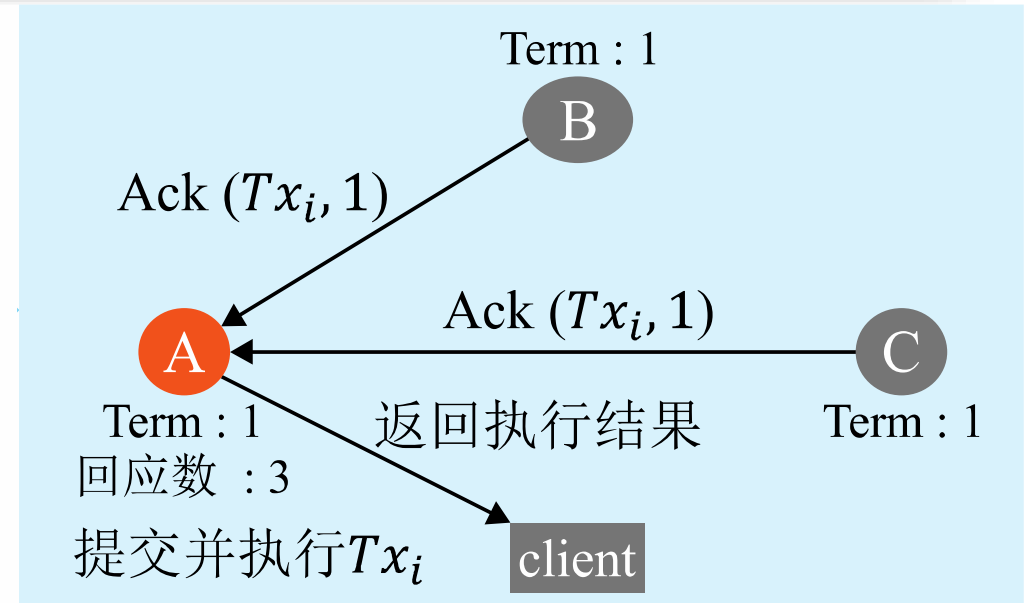
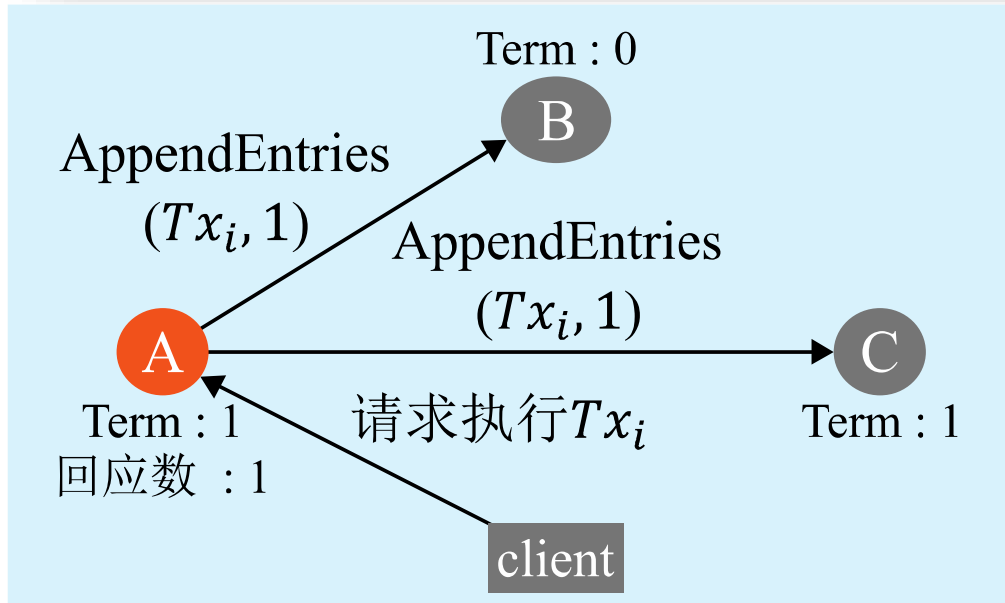
- 第一阶段：节点选举
 - 每个节点包含一个150ms~300ms的计时器
 - 若计时结束后还未从Leader节点收到任何回应，则触发超时机制
 - 节点将所在的term加1，并成为candidate，向其它节点发送“请选我”的信息，当收到超过半数投票后，则成为leader角色





Raft运行流程

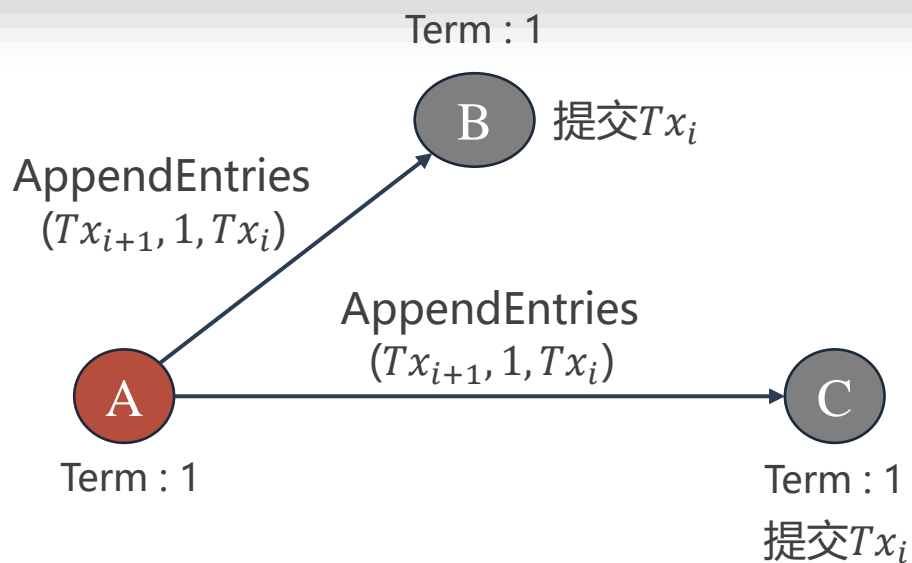
- 第二阶段：基于leader的日志复制
 - 客户端向leader发送交易，等待执行后的结果
 - Leader为交易分配序号并加入本地日志，然后通知其它节点将该交易加入本地日志，当收到大多数节点的Ack后，提交并执行该指令，然后回复客户端
 - 任意节点，若收到包含更大term的信息，则变为follower，并修改本地term





Raft运行流程

- 第二阶段：基于leader的日志复制
 - Leader节点在发送AppendEntries时，会包含最近被提交的交易
 - 其它节点收到AppendEntries后，会跟随leader提交之前的交易



AppendEntries RPC

Invoked by leader to replicate log entries (§5.3); also used as heartbeat (§5.2).

Arguments:

term	leader's term
leaderId	so follower can redirect clients
prevLogIndex	index of log entry immediately preceding new ones
prevLogTerm	term of prevLogIndex entry
entries[]	log entries to store (empty for heartbeat; may send more than one for efficiency)
leaderCommit	leader's commitIndex

Results:

term	currentTerm, for leader to update itself
success	true if follower contained entry matching prevLogIndex and prevLogTerm



第3节 信任问题

- ✓ 访问控制
- ✓ 信任评价模型
- ✓ 拜占庭容错



零信任网络

“从来不相信，始终在校验” (Never Trust, Always Verify)

- 零信任模型不依靠建立隔离墙来保护可信的资源，而是接受“不可信”或“坏人”无处不在的现实，试图让全体资源都拥有自保的能力
- 零信任默认不应该信任企业网络内部和外部的任何人/设备/应用，需要基于认证和授权重构访问控制的信任基础

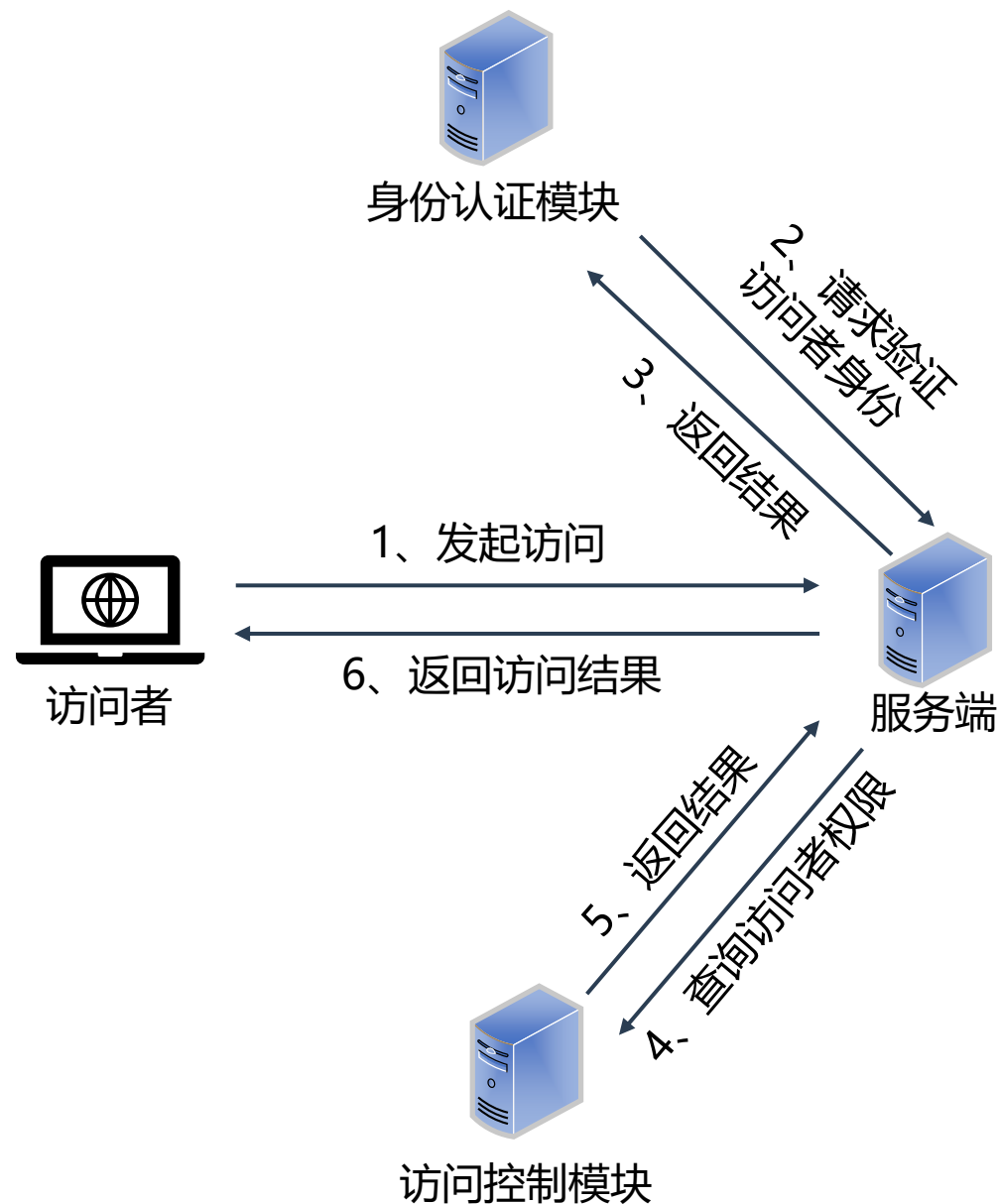


零信任对传统访问控制机制进行了范式上的颠覆
其本质是以身份为基石的动态可信访问控制



零信任下访问的基本流程

- 访问者发起对服务端的访问
- 服务端收到访问指令后对其进行验证，主要包含以下两个步骤
 - 请求身份认证模块验证访问者身份
 - 根据身份以及被访问资源请求访问控制模块判断访问者是否存在权限
- 若验证都通过后，服务端允许此次访问，并返回访问结果

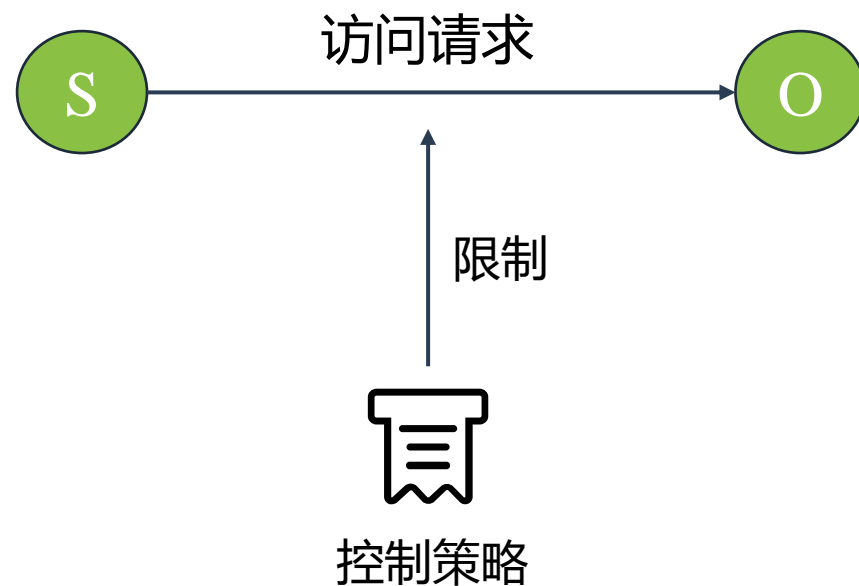




访问控制

访问控制（Access Control）指系统按用户身份，以及该身份所在的策略组来限制用户对某些信息项的访问，或限制其对某些控制功能的使用

- 主体S (Subject)
 - 提出对资源访问的具体请求
- 客体O (Object)
 - 提出对资源访问的具体请求
- 控制策略A (Attribution)
 - 是主体对客体的相关访问规则集合，即属性集合
- 访问控制规定了主体（用户或者进程）对客体（数据资源）的访问权限





基于角色的访问控制

RBAC设置不同的角色，并将各种身份信息与角色进行绑定，然后只对角色的访问权限进行控制，可极大地简化权限管理

包含用户、角色和权限三个部分

- 角色可以看作是一组操作的集合，不同的角色具有不同的操作集，这些操作集由系统管理员分配给角色
- 用户与角色是多对一或者多对多关系。一个用户对应一个角色、一个用户对应多个角色
- 用户最终权限取多个角色权限的并集

RBAC模型授权管理步骤

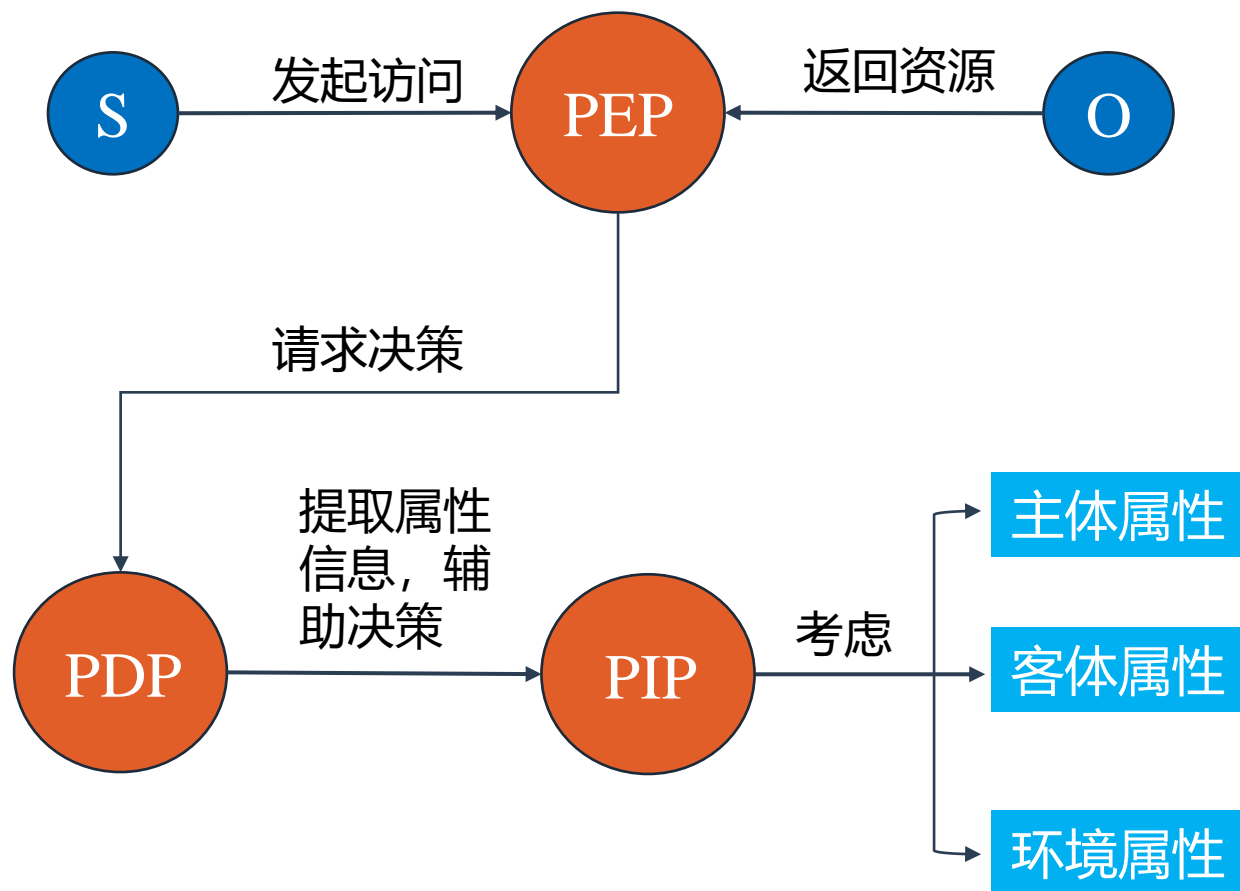
- 根据任务需要定义具体不同的角色
- 为不同角色分配资源和操作权限
- 为一个用户组指定一个角色



基于属性的访问控制

用户属性指的是可以对用户进行区分的特性。用户是访问控制的主体，以其属性作为访问网络数据的依据，可以实现更细粒度的访问控制

- 策略实施点 PEP：拦截访问用户的请求，并请求PDP对该请求进行评估，判断是否允许此次访问
- 策略决策点 PDP：根据访问属性及策略判断访问请求是否被允许
- 策略信息点PIP：提供属性信息，辅助访问控制

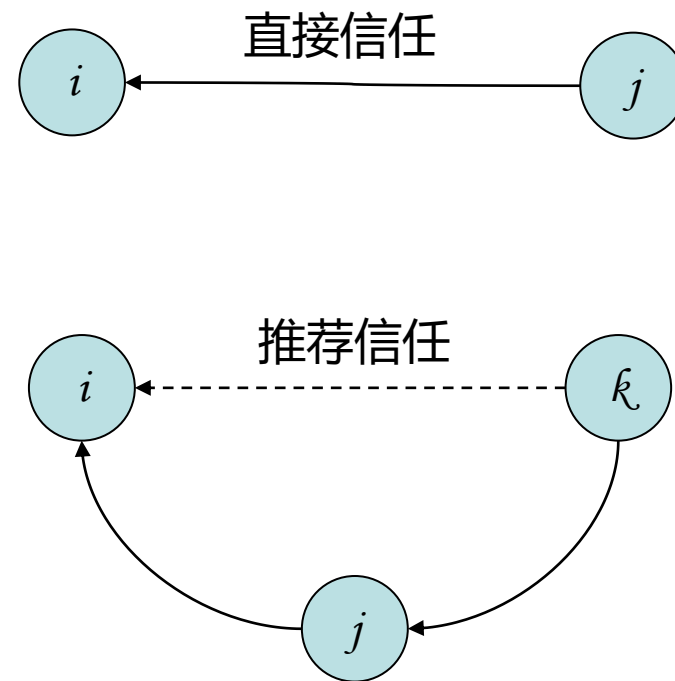




基于信任管理的访问控制

基于用户之间的初始信任和历史交互记录得到信任值，
然后根据信任值等级的不同给予相应的访问权限

- 基本信任：代表着节点由于自身的属性所具有的默认信任值，依据可以是节点的角色、节点所使用的操作系统、登录地点、地址等
- 直接信任：由主体和客体之间的交互历史产生，直接信任随着两者的持续交互而逐渐积累
- 推荐信任：依据信任值的传递性产生，主体可以参考其它节点与客体的交互历史对客体进行信任评价，也称作“间接信任”





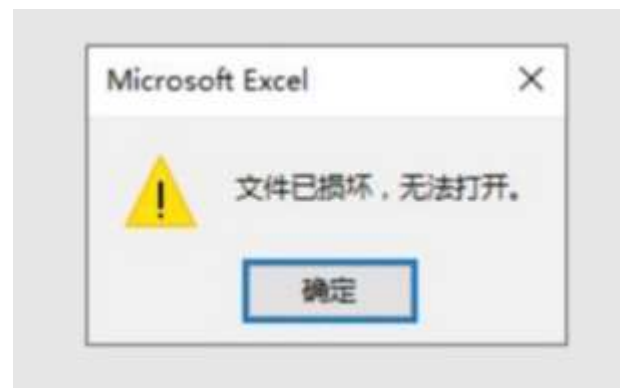
访问控制的局限性

**访问控制只能解决被访问者对访问者的信任问题，
但无法解决访问者对被访问者的信任问题，访问者可能访问到恶意、无效资源**

- 访问者对被访问者的信任问题主要表现在：
 - 文件下载
 - 资源共享
 - 访问网站链接
- 典型场景：P2P文件共享解决方法：信任评价模型



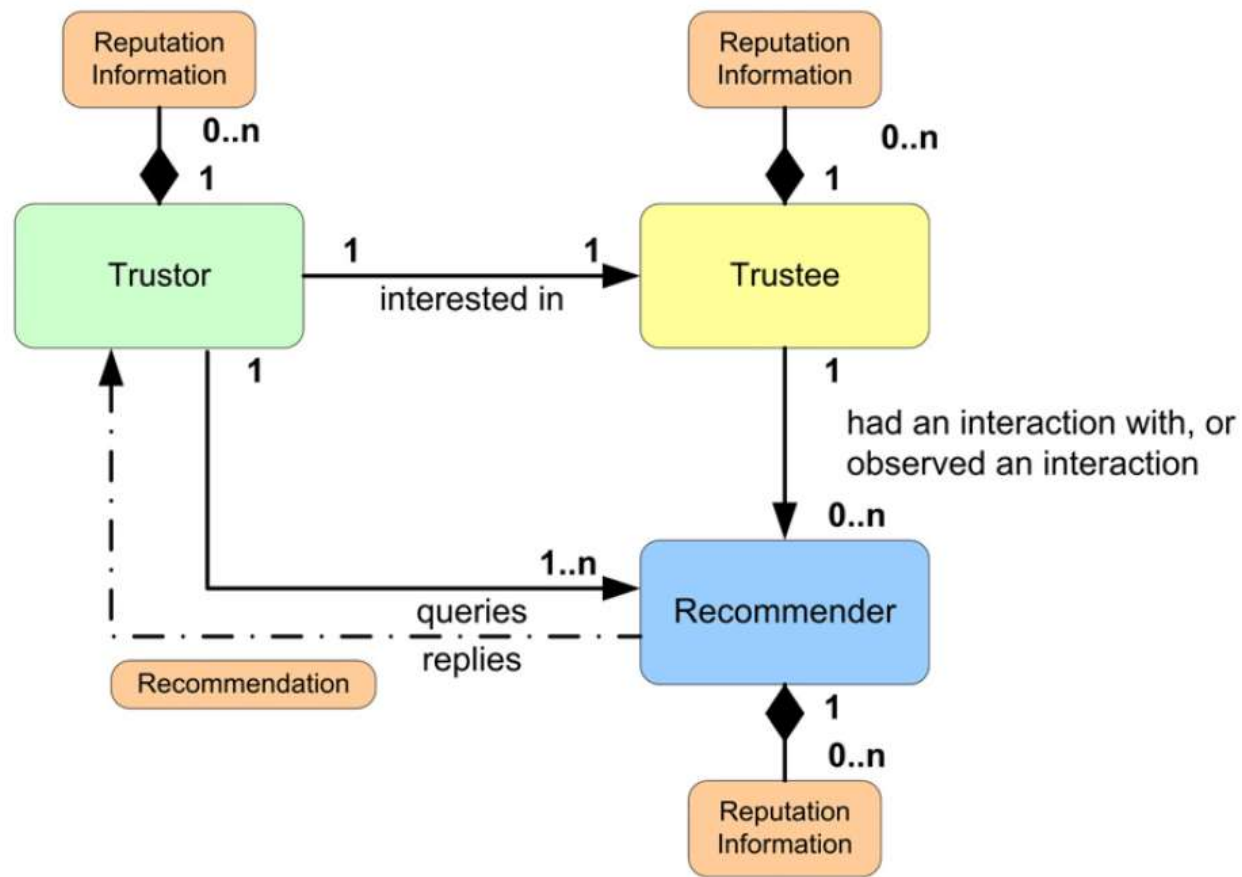
gnutella





信任评价模型

- 问题：如何描述？如何计算？如何传递与更新
- 信任模型细化：行为模型；计算模型；传播模型
- 包含角色
 - Trustee：待评价节点
 - Trustor：期待与Trustee交互节点
 - Recommender：与Trustee交互过

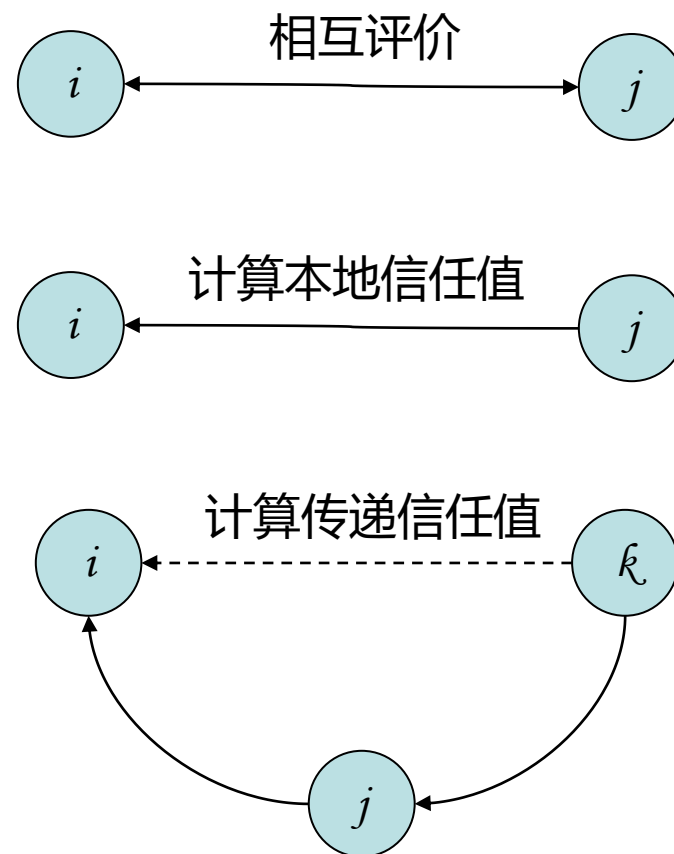




信任评价系统举例—Eigentrust

主要思想：节点对每次交互过程中的交互节点进行评价，并根据历史的交互行为评价计算节点信誉值

- 基本流程
 - 节点根据本地的历史交互评价为每个节点计算本地信誉值
 - 节点根据其它节点的本地信誉值计算出传递信誉值，并通过多次迭代确保收敛
 - （可选）所有节点根据计算的传递信任值计算出全局信任值





信誉系统的五个要求

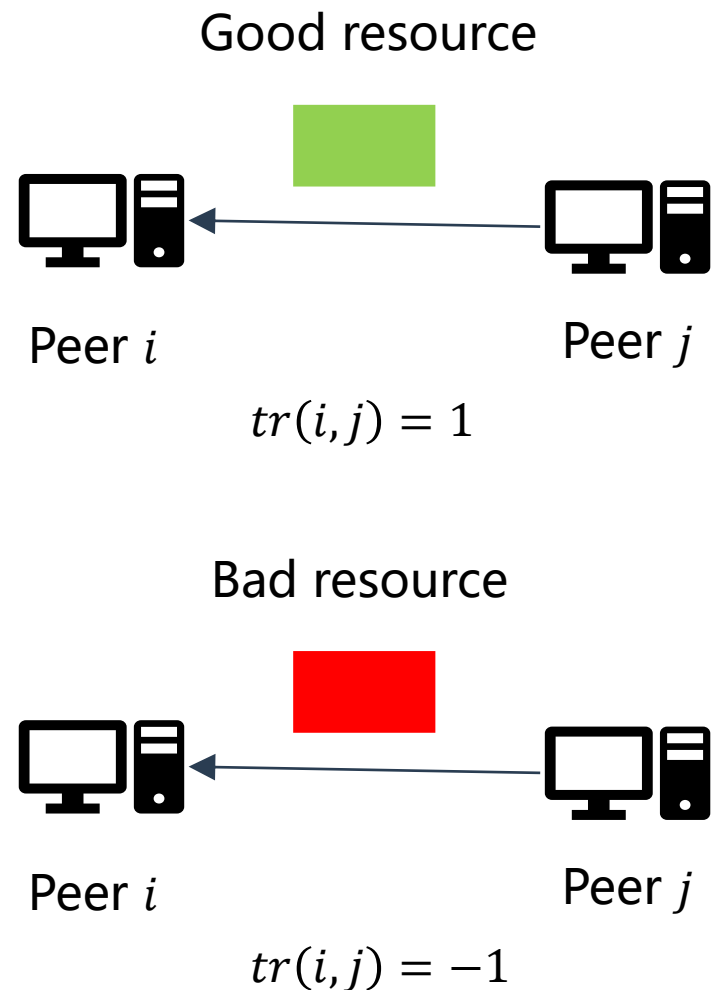
- 信用评分不依赖中央机构
- 信用评分应该保证匿名化，不应考虑IP地址等外部因素
- 信用评分应该靠良好行为累积提升，不应该特别照顾新用户
- 系统应该确保低开销，保证实用性
- 系统要能抵抗少部分节点联合发起的恶意评价行为



基本概念

对节点的本地信誉值为与其交互的历史好评数减去历史差评数

- 对于节点 i 和节点 j 之间的每次交互，两者之间存在相互评价
 - $tr(i, j) = 1$ 表示节点 i 认为节点 j 表现正常
 - $tr(i, j) = -1$ 表示节点 i 认为节点 j 表现不正常
- 节点 i 对节点 j 评价正常的总次数为 $sat(i, j)$
- 节点 i 对节点 j 评价不正常的总次数为 $unsat(i, j)$
- 节点 i 对节点 j 本地信任值为 s_{ij}
 - $s_{ij} = sat(i, j) - unsat(i, j) = \sum tr(i, j)$





传递信誉值的计算

节点交换各自的本地信誉值得到本地信誉值矩阵 C ，再乘以本地信誉值向量后可得到一阶传递信誉值向量 t_i^1 ，迭代该过程可得 k 阶传递信誉值

- 将本地信誉值 s_{ij} 的归一化，得到 c_{ij}

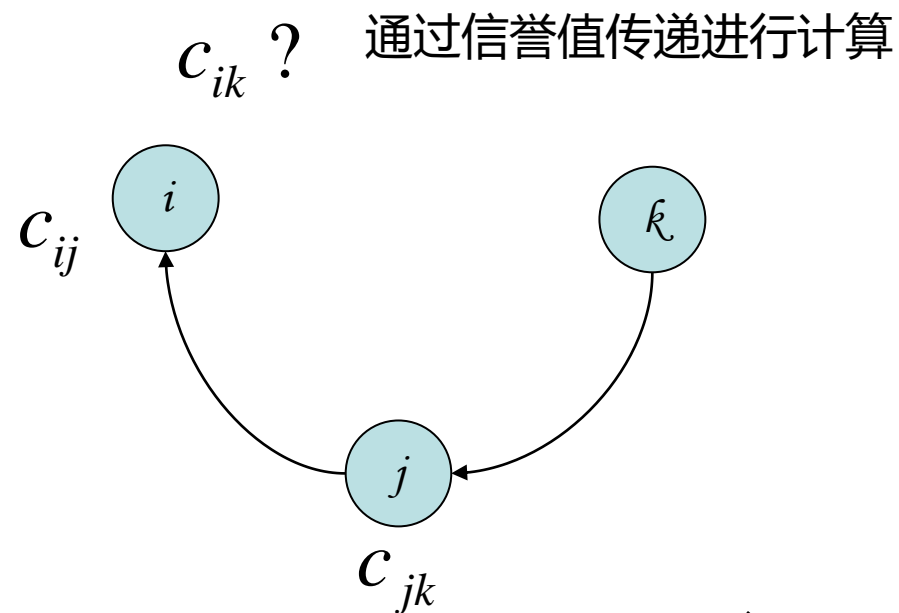
- $$c_{ij} = \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)}$$

- 基于信任传递计算本地一阶传递信誉值 t_{ik}^1

- $$t_{ik}^1 = \sum_j c_{ij} c_{jk}$$

- 令归一化本地信誉值矩阵为 $C = [c_{ij}]$ ，则节点 i 的本地传递信誉值向量为 $\overrightarrow{t_i^1}$

- $$\overrightarrow{t_i^1} = C^T \overrightarrow{c_i} \longrightarrow \overrightarrow{t_i^{k+1}} = C^T \overrightarrow{t_i^k} \longrightarrow \overrightarrow{t_i^n} = (C^T)^n \overrightarrow{t_i^k}$$





实现时需要考虑的问题

- 初始信任值的设置，可以自行设置，或者采用默认设置

- $p_i = 1/n \longrightarrow \overrightarrow{t_i^1} = C^T \vec{p} \longrightarrow \overrightarrow{t_i^n} = (C^T)^n \vec{p}$

- 因此，在对本地信誉值 s_{ij} 归一化时，计算改为如下模式：

- $$c_{ij} = \begin{cases} \frac{\max(s_{ij}, 0)}{\sum_j \max(s_{ij}, 0)} & \text{if } \sum_j \max(s_{ij}, 0) = 0 \\ p_j & \text{otherwise} \end{cases}$$

- 为了避免恶意节点合作影响其它节点的信誉值，可以在每次迭代计算时保留自己的初始信任值：

- $\overrightarrow{t_i^{k+1}} = (1 - a)C^T \overrightarrow{t_i^k} + a\vec{p}$



协作计算全局信誉值

- 节点之间交换本地计算的传递信誉值，然后根据所有节点的传递信誉值计算全局信誉值

- $$t_i^{(k+1)} = (1-a)(c_{1i}t_1^{(k)} + \dots + c_{ni}t_n^{(k)}) + ap_i$$

- 存在的问题：

- 恶意节点可能上报错误的本地传递信誉值

解决方案：每个节点的传递信誉值都由多个节点进行计算，再根据多个节点的计算结果选取最终的值

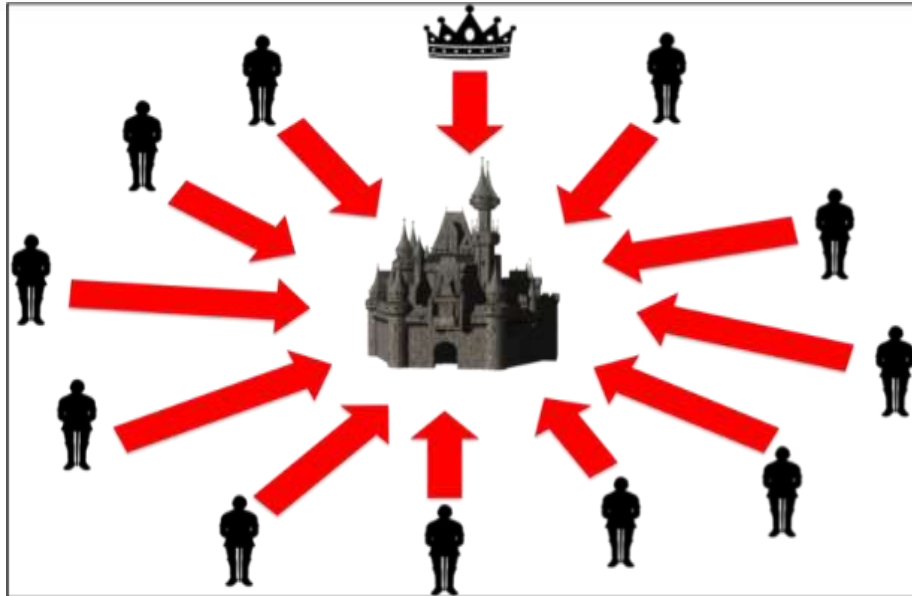
- 恶意节点可能对不同节点上报不一样的传递信誉值（协作过程中的信任问题）

解决方案：基于**拜占庭容错算法**实现信誉值的广播

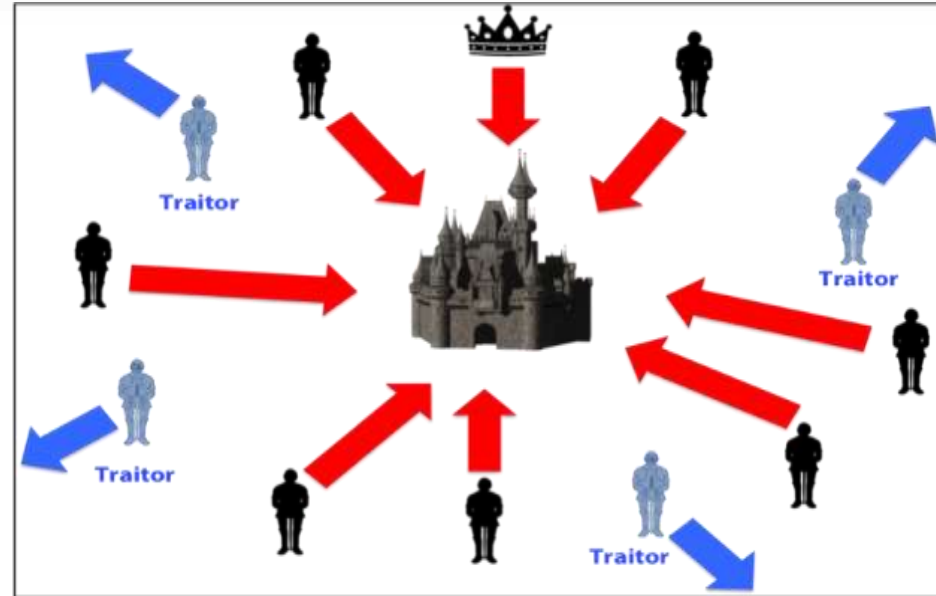


拜占庭将军问题

- **背景：**拜占庭罗马帝国疆域辽阔，许多将军守卫疆土；战时，部队相隔远，只能靠信差传递消息；至少超过半数的将军对是否共打敌人达成一致，否则失败；内部可能存在叛徒
- **问题：**存在叛徒时，忠诚的将军们如何达成共识



Coordinated Attack Leading to Victory

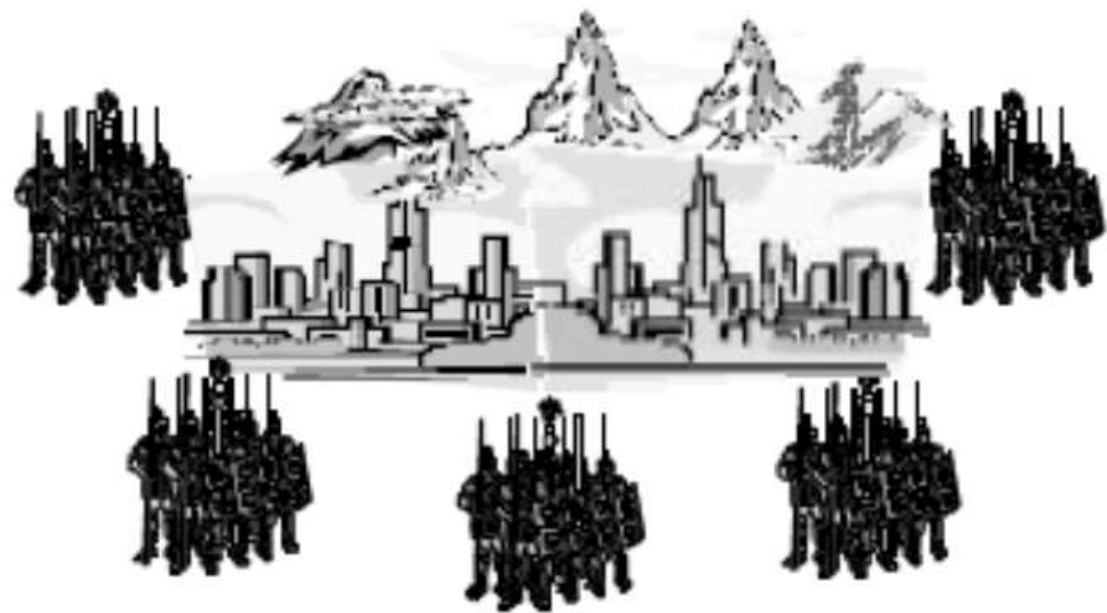


Uncoordinated Attack Leading to Defeat



拜占庭将军问题

- **概念：**将军 等价于 计算机系统的部件
- **问题抽象：**
 - 每个部队由一个将军领导
 - 一共有 n 个将军，其中一些是叛徒
 - 所有军队将敌方包围，观察敌军
 - 将军间靠通信兵交流



- **目标：**存在所有忠诚的将军采取**同样**的行动；**少量**叛徒无法使诚实的将军们接受一个**坏**的计划
- **注意：**不必识别出叛徒



拜占庭将军问题

The Byzantine Generals Problem

Leslie Lamport (SRI International),

Robert Shostak (SRI International),

Marshall Pease (SRI International)

Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement. It is shown that, using only oral messages, this problem is solvable if and only if more than two-thirds of the generals are loyal; so a single traitor can confound two loyal generals. With unforgeable written messages, the problem is solvable for any number of

This research was supported in part by the National Aeronautics and Space Administration under contract NAS1-15428 Mod. 3, the Ballistic Missile Defense Systems Command under contract DASG60-78-C-0046, and the Army Research Office under contract DAAG29-79-C-0102.

Authors' address: Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1982 ACM 0164-0925/82/0700-0382 \$00.75

Paper originally published in *Transactions on Programming Languages and Systems*, 4(3), July 1982, pp. 382-401.



Leslie B. Lamport

[PDF] [The Byzantine generals problem](#)

L Lamport, R Shostak, M Pease - *Concurrency: the Works of Leslie ...*, 2019 - [dl.acm.org](#)

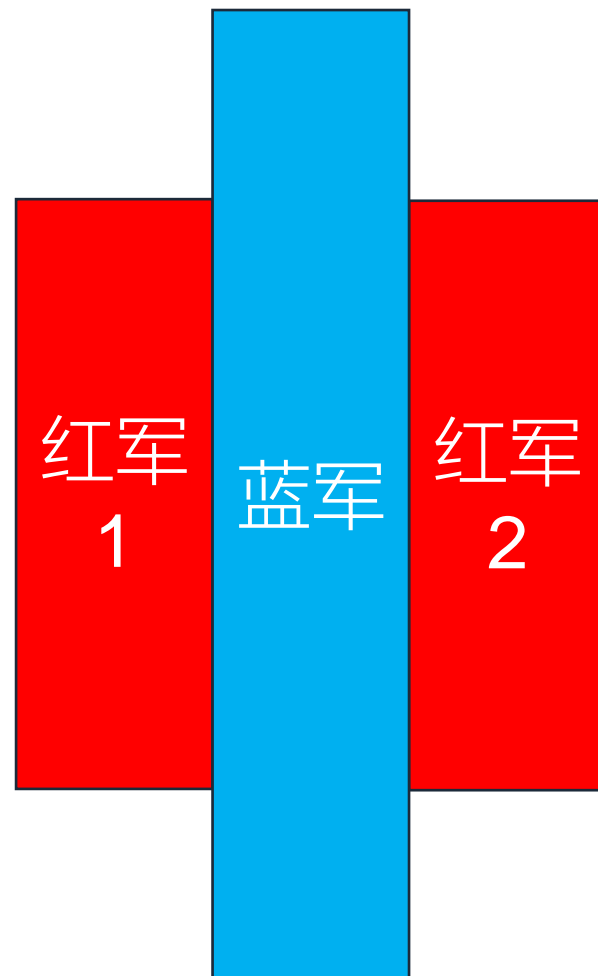
Page 1. The **Byzantine Generals Problem** Leslie Lamport (SRI International), Robert Shostak (SRI International), Marshall Pease (SRI International) ... 382-401. Page 2. 204 The **Byzantine Generals Problem** **generals** and possible traitors ...

☆ 59 被引用次数: 7814 相关文章 所有 199 个版本



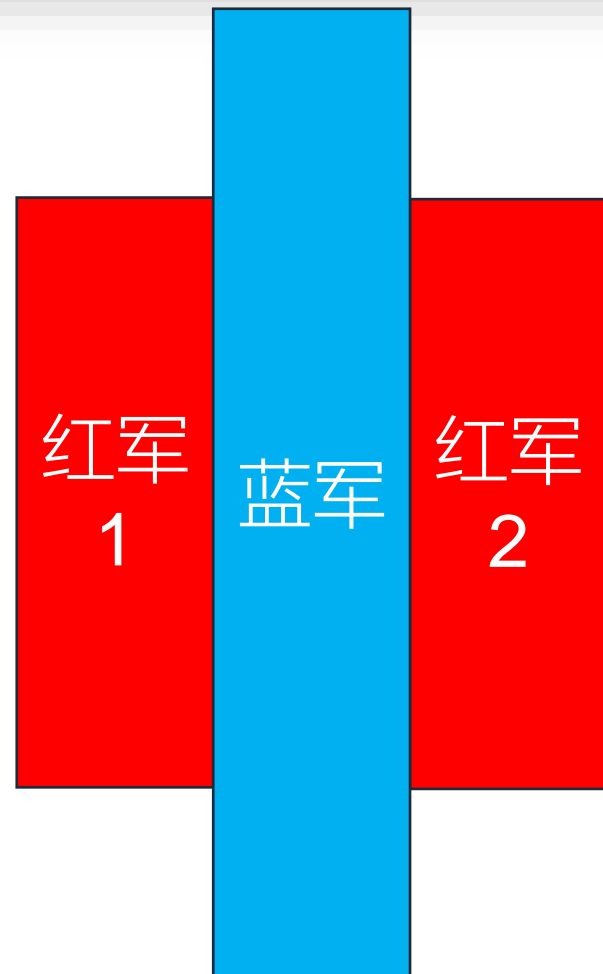
问题定义

- 拜占庭将军问题的前提
 - 可靠信道：信息可以准确无误的传达给接收方
- 不可靠信道：两军问题
 - 红军必须在同一时间攻打蓝军才能胜利
 - 红军1、2通信必须经过蓝军区域，消息可能被截获、篡改
 - 如何确定攻打时间？





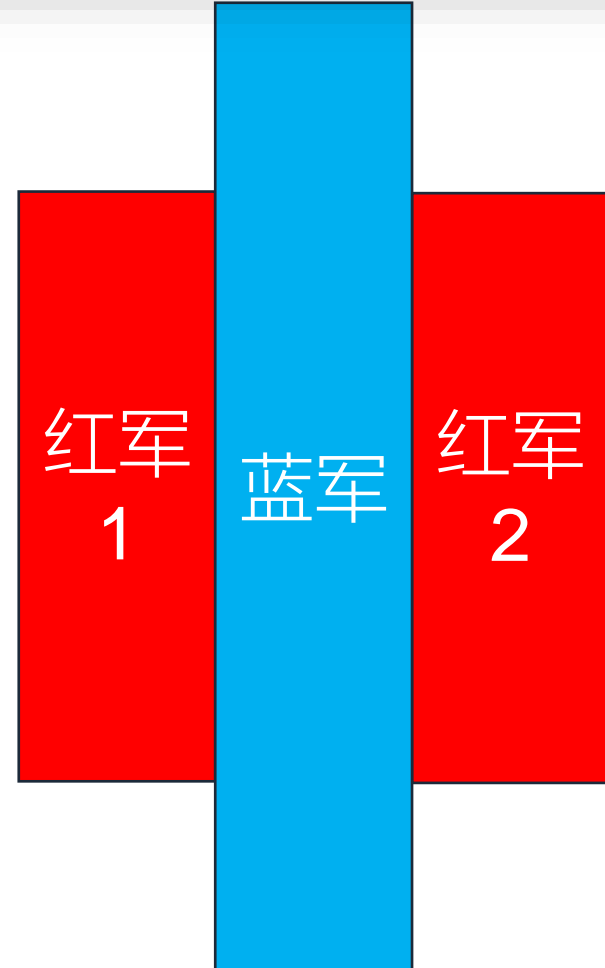
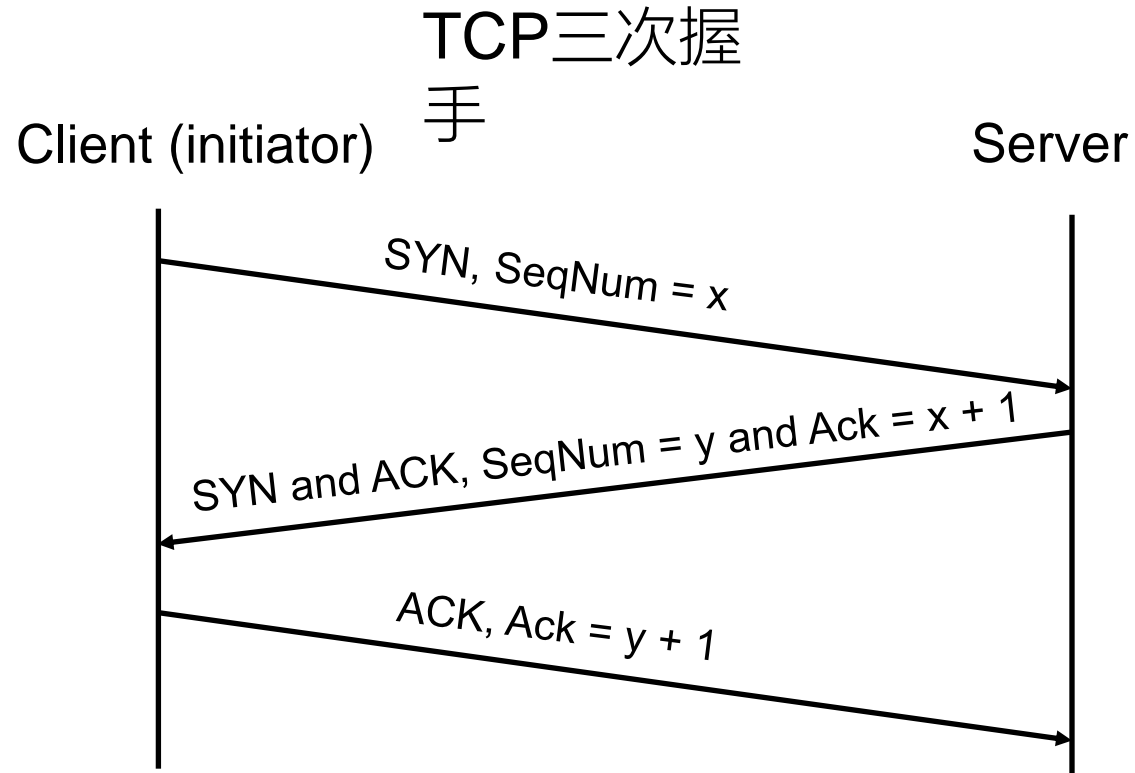
前提条件：不可靠信道：两军问题； 结论：两军问题不可解





问题定义

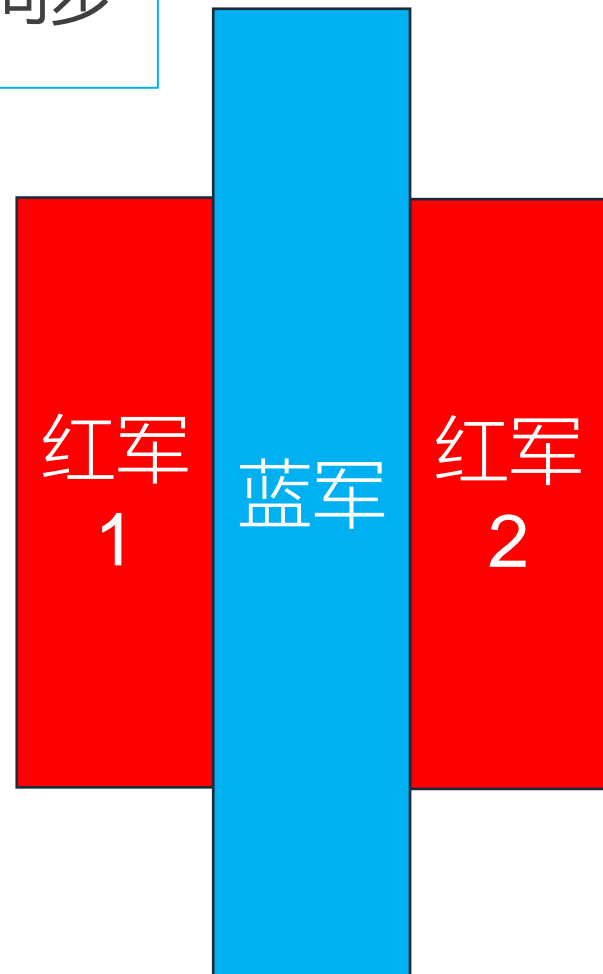
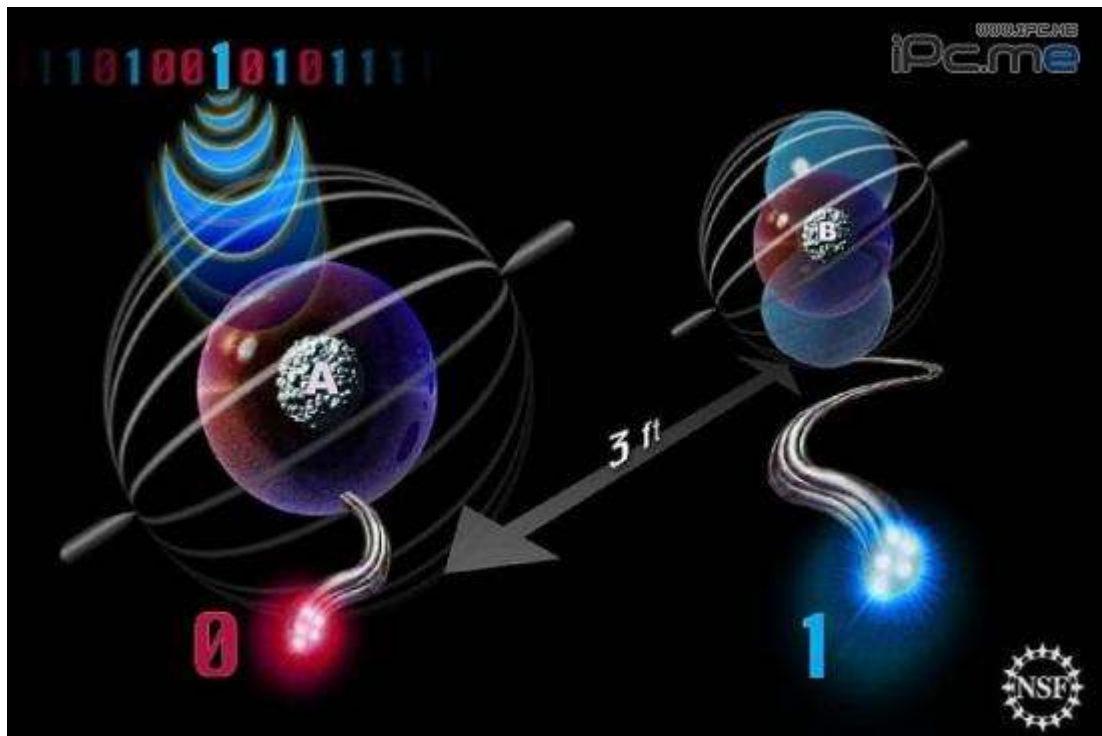
两军问题实例:TCP通信





问题定义

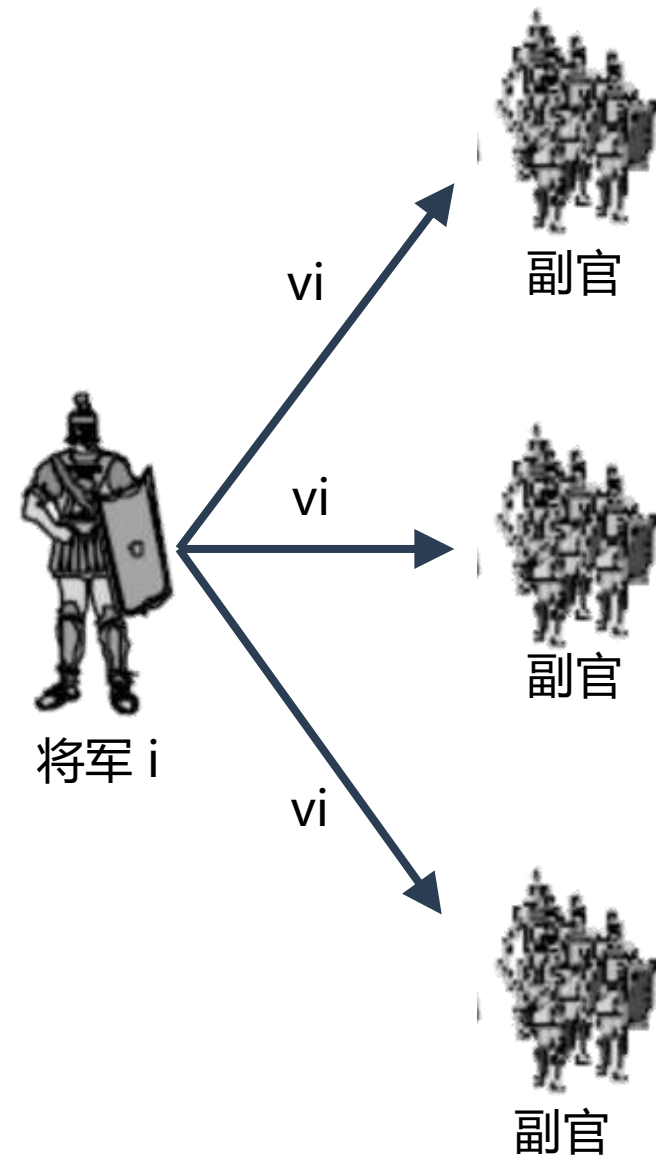
- 两军问题终极解决方案：量子通讯
- 处于量子纠缠的两个粒子，无论相隔多远都可以同步





问题简化

- 将问题简化为： **一个将军**将消息发送给**其他的副官**
- 拜占庭将军问题
 - 将军（发布命令者）必须将他的命令发送给他的 $n-1$ 个副官
- 交互一致性（Interactive Consistency, IC)
 - IC1: 所有忠诚的副官都服从同样的命令
 - IC2: 如果发布命令的将军是忠诚的，那么每个忠诚的副官都服从他的命令
- 注意：如果将军是忠诚的，那么 IC2 等价于 IC1





一致性和正确性条件

- 一致性 (等价IC1)
 - 每个将军必须得到相同的(v_1, v_2, \dots, v_n)指令向量或指令集合
 - 忠诚的将军不一定使用*i*将军发来的信息作为 v_i , *i* 将军有可能是叛徒

**一致性条件不够充分，忠诚将军的信息有可能被修改，
所以必须引入另一个正确性的条件**

- 正确性 (等价IC2)
 - 若*i*将军是忠诚的，其他忠诚的将军必须使用他发出的值作为 v_i



解决方法

主要问题：由于将军可能是叛徒，会对不同的副官发布不同的指令，因此副官不能完全信任将军的指令

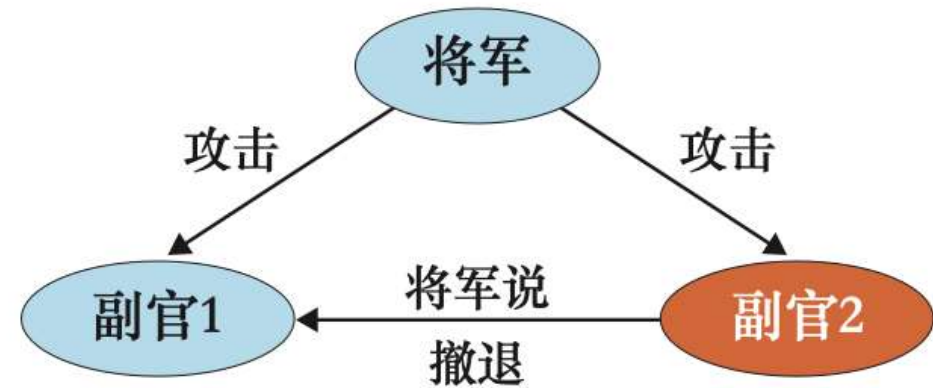
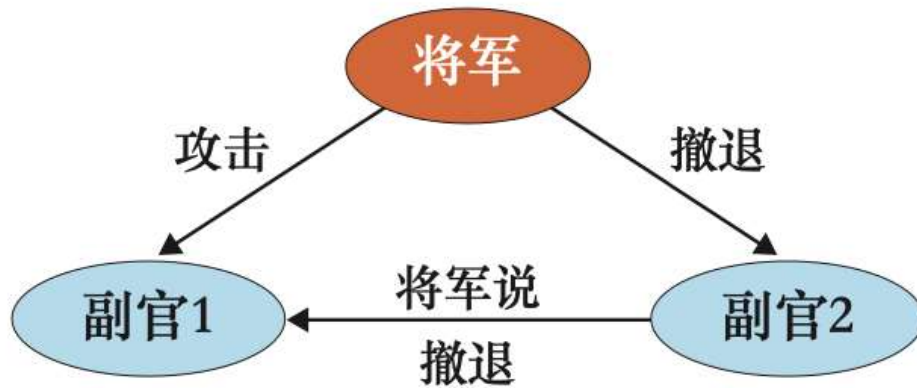
解决思路：副官在收到将军的指令后，互相交换从将军处收到的指令值，再基于交互结果选择最终执行指令

- 两种方法
 - 口头消息协议：所有副官在交换从将军处收到的指令值时基于口头传播，因此可以说谎（对应于系统中**无**数字签名的场景）
 - 签名消息协议：副官从将军处收到的指令附带签名，因此无法伪造将军的指令，将军也无法对已发布的指令抵赖（对应于系统中**有**数字签名的场景）



口头消息下三将军问题无解

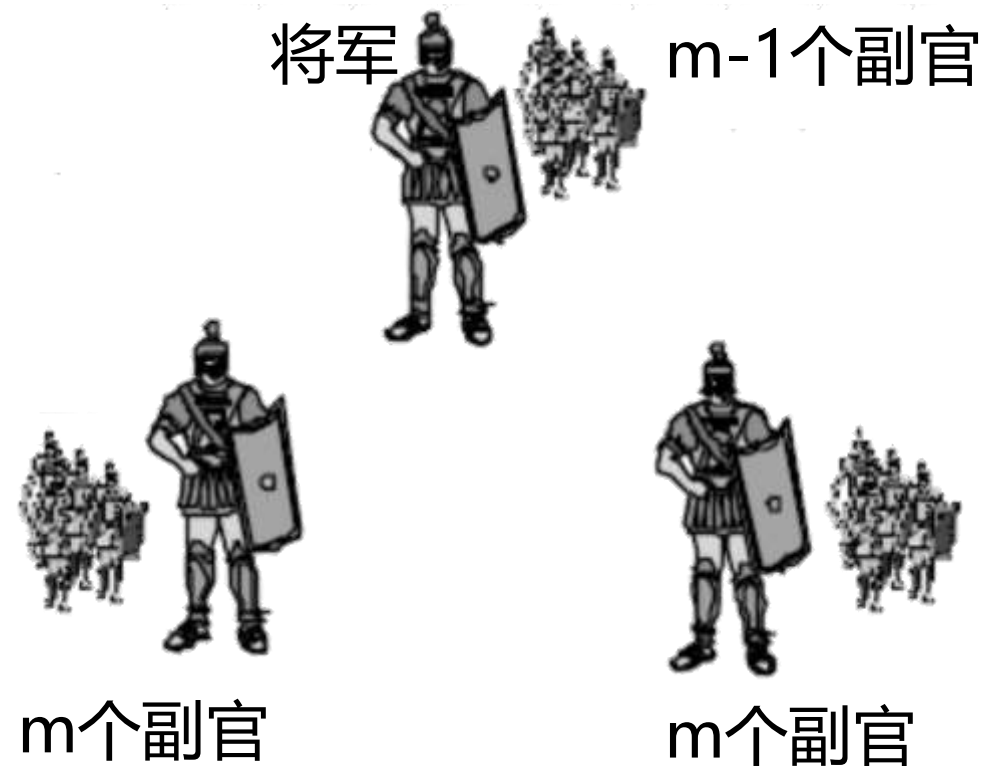
- 现有三个将军，其中一个叛徒
 - 两种消息：攻击、撤退
 - 深色的是叛徒
- 副官1在两种情形下收到的消息是一样的，无法判断谁是叛徒
 - 如果确保左图副官1做出正确选择撤退，那么在右图中会作出错误选择，即和忠诚将军的指令不一致
 - 如果确保右图副官1作出正确选择，在左图则会和忠诚的副官2做出不一致的选择





口头消息下的不可能性

- 一般情况下，将军与将军之间基于口头消息传播信息时
 - 在有 m 个叛徒时，少于 $3m+1$ 个参与者（包含叛徒），问题无解
- 通过反证法证明：
 - 假设共有 $3m$ 个将军，其中 m 个是叛徒
 - 将问题简化为3将军问题
- $3m$ 个将军有解 等价于 3将军问题有解





口头消息(OM)协议

结论：假设将军个数为 N ，且最多存在 m 个叛徒时，当 $N \geq 3m + 1$ 时， $OM(m)$ 算法可以确保所有忠诚的将军达成一致

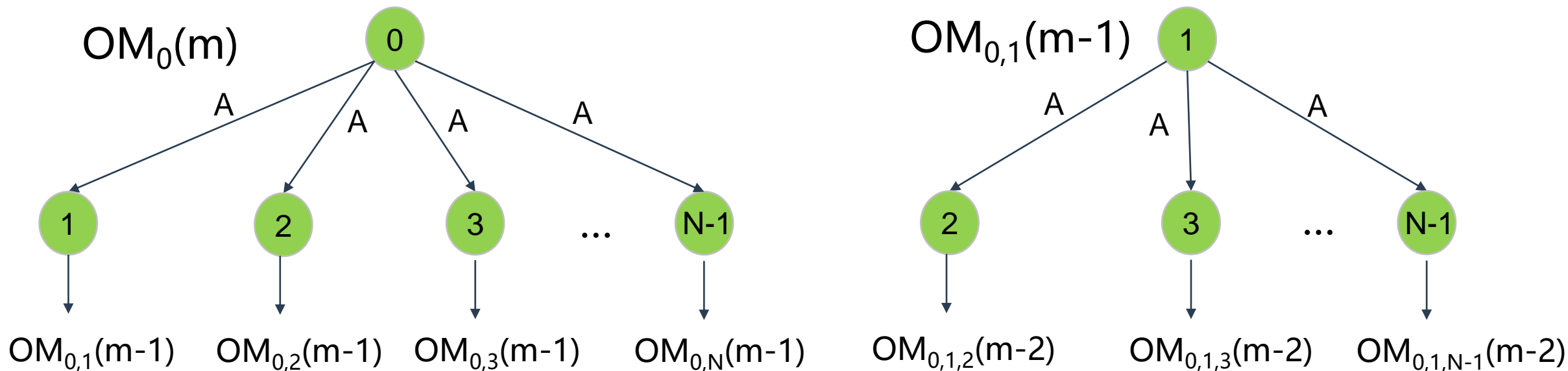
- 假设
 - 每个消息都被正确发送
 - 被发送的消息在回合结束前都能被正确接收（同步假设）
 - 消息的接收者知道谁是发送者
- 确保
 - 叛徒不能通过说谎来影响共识
 - 叛徒不能通过保持沉默来影响共识

证明思想：根据数学归纳法完成证明，即 $m=1$ 时成立，并且当 $m=k$ 时成立，能推导出 $m=k+1$ 时也成立



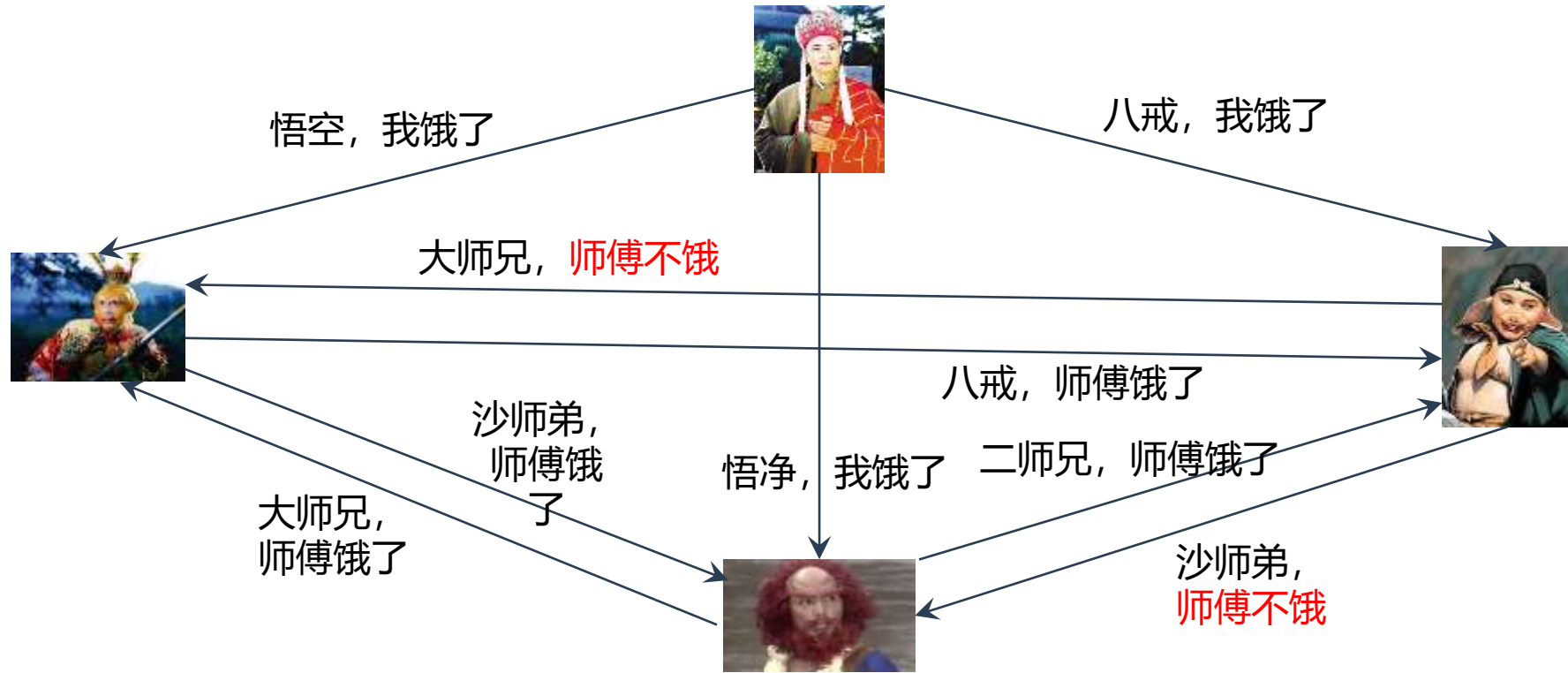
口头消息协议

- 假设 N 个副官编号分别为 $0, 1, \dots, N-1$ ，初始状态下 0 为将军，并执行 $OM_0(m)$
- $OM_0(m)$ 的运行流程为：将军 0 给其它副官发送指令，然后其它副官分别执行 $OM_{0,i}(m-1)$ ，将从将军处收到的指令发给剩余副官
- $OM_{k_1, k_2, \dots, k_v, i}(m-v)$ 表示的 OM 算法是在除去副官 k_1, k_2, \dots, k_v 后剩余的副官中执行，且副官 i 将从副官 k_v 执行 $OM_{k_1, k_2, \dots, k_v}(m-v+1)$ 时发送的指令发给剩余副官





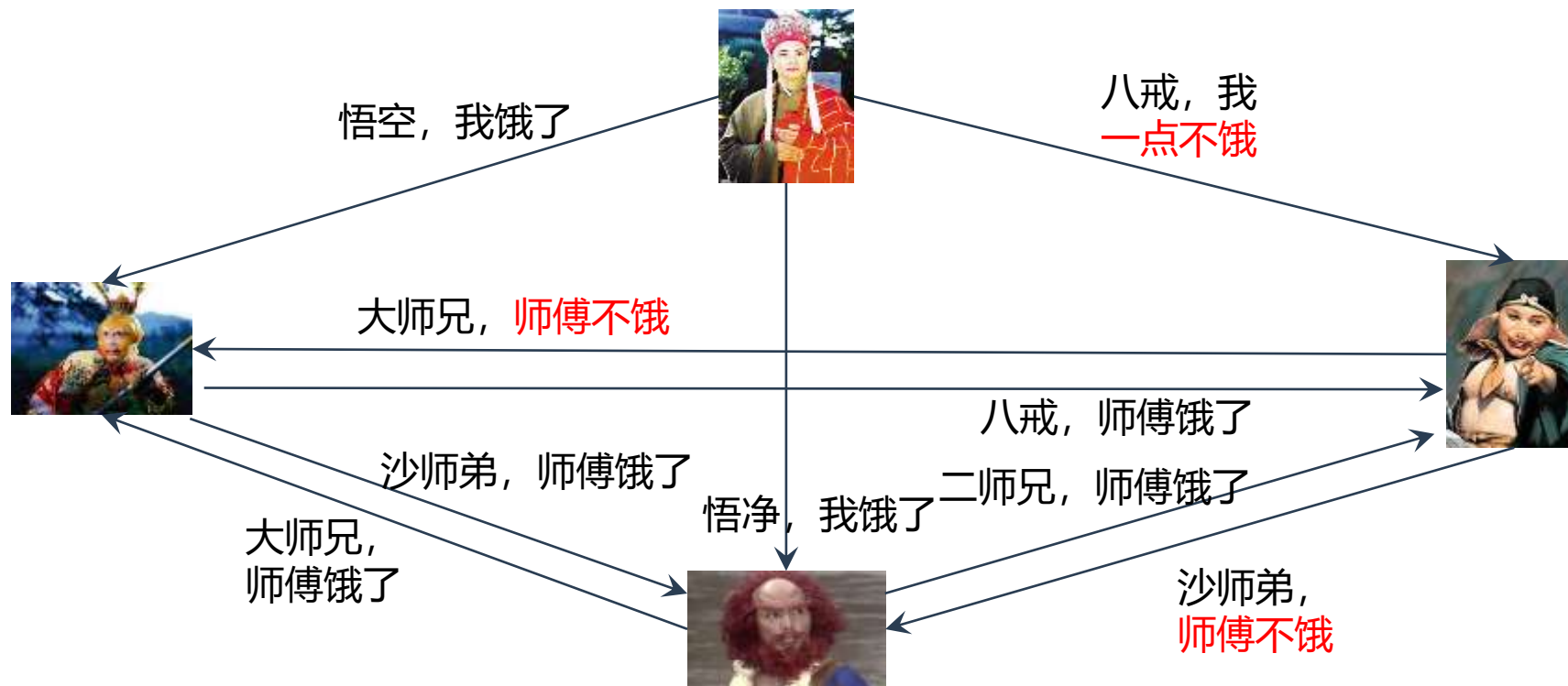
四将军问题实例：懒惰的八戒



- 悟空和沙僧都收到了（饿了，饿了，不饿），最后都认定师傅饿了
- 满足了IC1，师傅、悟空、沙僧的意见是一致的
- 满足了IC2，悟空和沙僧都听师傅的



四将军问题实例：机智的师傅

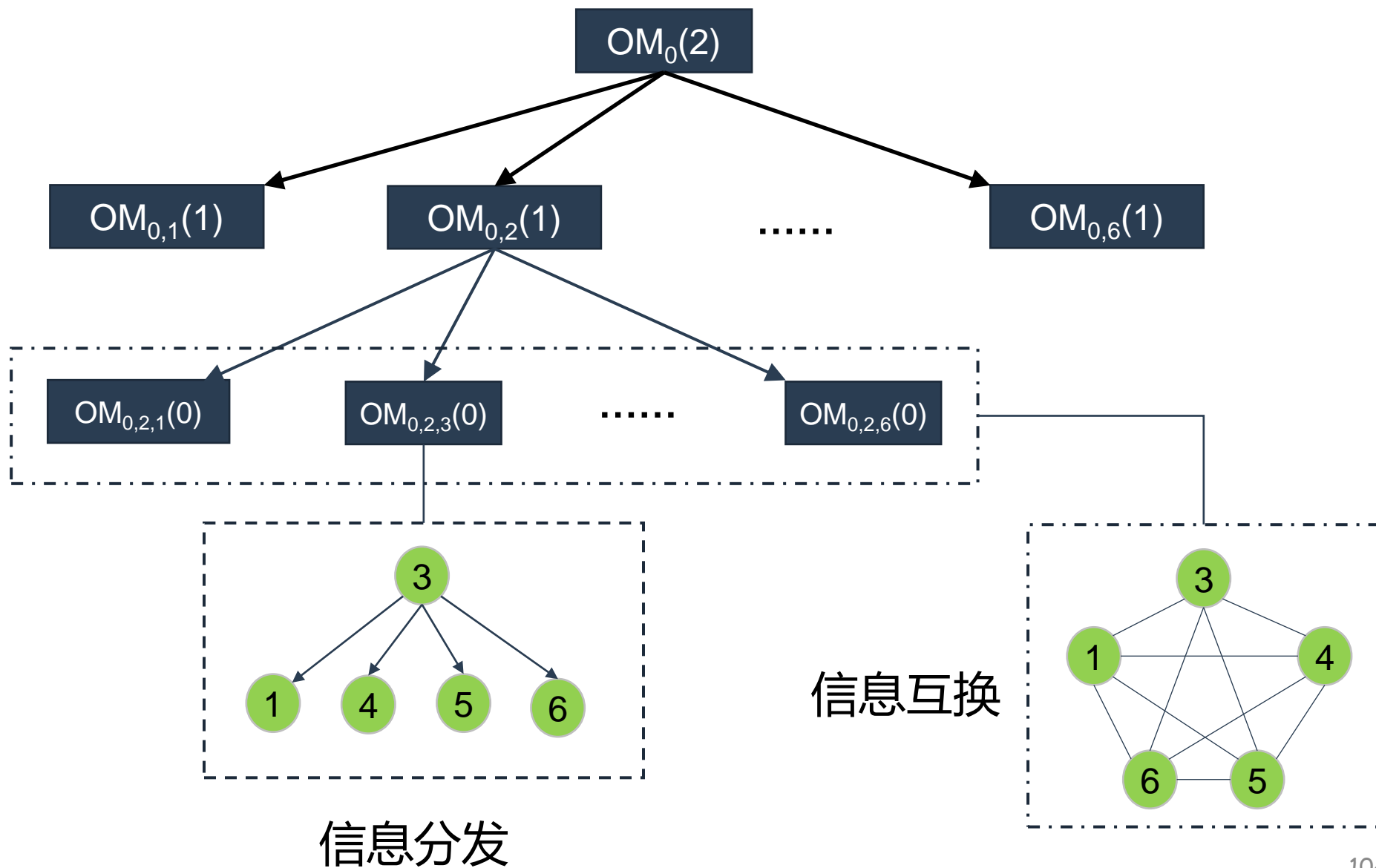


- 悟空、沙僧、八戒都收到了（饿了，饿了，不饿），都认定师傅饿了
- 满足了IC1，悟空、沙僧、八戒的意见是一致的
- 无需满足了IC2，师傅撒谎了



7将军2叛徒算法分析

- OM(2): 将军0将消息发送给副官1~6
- OM(2) 算法是为了0~6之间的一致
- OM(1) 本质上是某节点完成一次信息的广播后, 剩余节点对收到的消息进行互换
- OM(0) 本质上是完成一次信息的广播

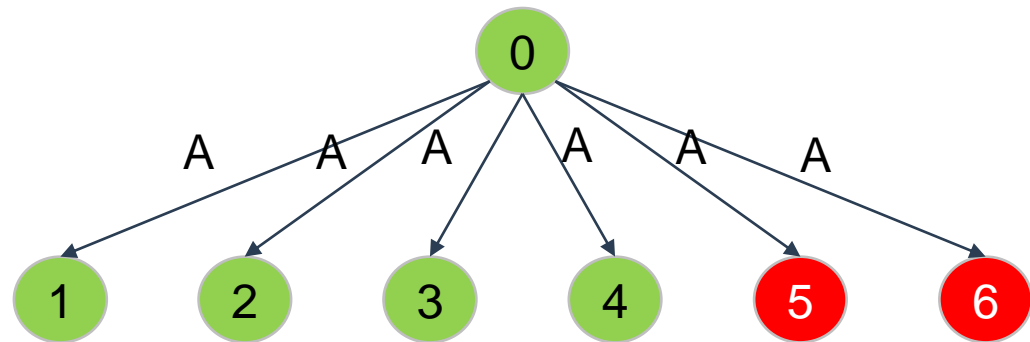




场景一：将军是忠诚的

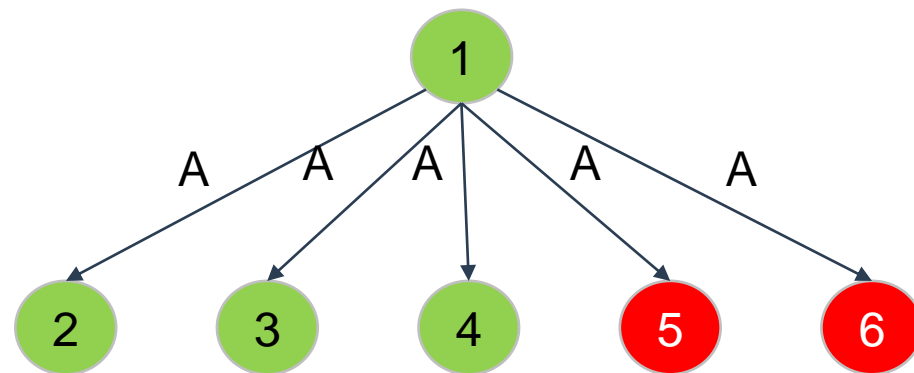
- $OM_0(2)$ 分发消息(A:attack, R:retreat)

接收 发送	1	2	3	4	5	6
0	A	A	A	A	A	A



- 每个节点都执行一次 $OM_{0,i}(1)$ ，以1执行 $OM_{0,1}(1)$ 为例：为了达成对1的判断，2~6间交换信息

接收 发送	2	3	4	5	6
1	A	A	A	A	A

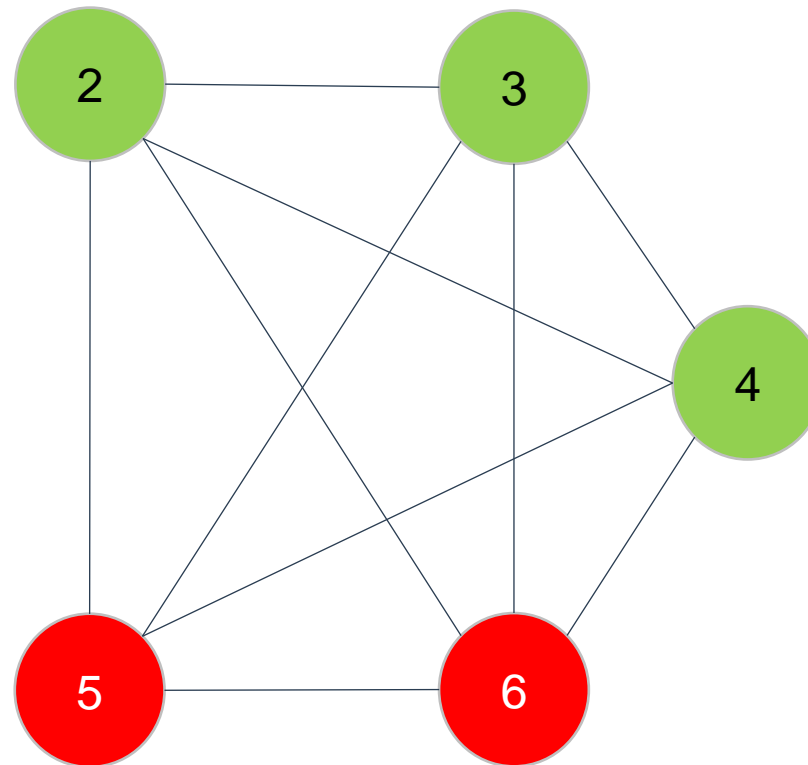




场景一：将军是忠诚的

- $OM_{0,1,i}(0)$: 为了达成对1的判断, 2~6间交换信息

接收 发送	2(A)	3(A)	4(A)	5	6
2		A	A	A	A
3	A		A	A	A
4	A	A		A	A
5	x2	x3	x4		x6
6	y2	y3	y4	y5	
决策	A	A	A	a	b

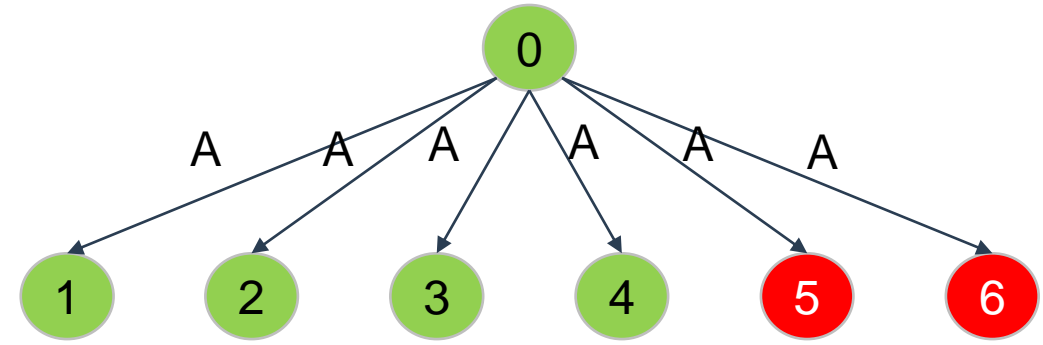
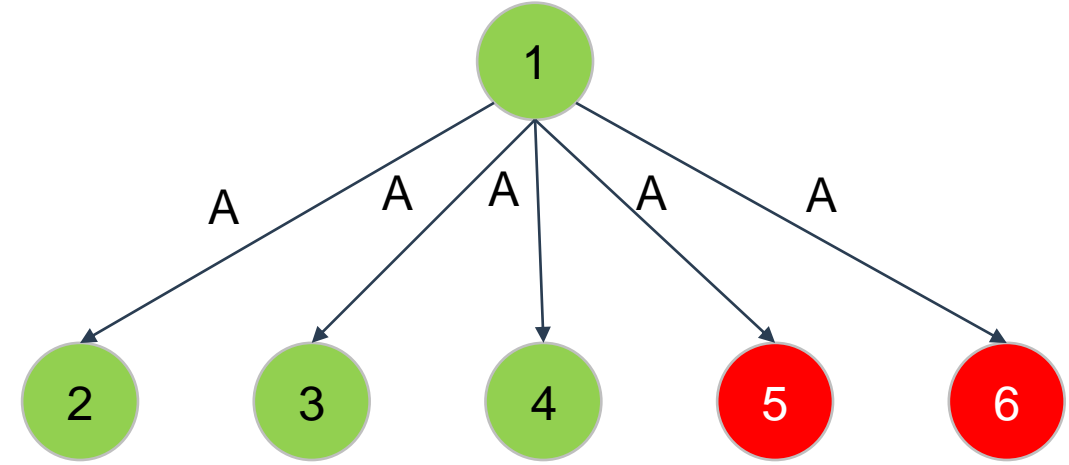


- 无论5和6发送什么消息, 2/3/4得到的信息序列都为(A, A, A, x, y)形式的



场景一：将军是忠诚的

- 回到 $OM_{0,1}(1)$ 中，由于副官1发送的指令为A，所以2、3、4的最终序列为(A, A, A, A, x, y)，故决策为A，与忠诚的副官1一样
- 回到 $OM_0(2)$ ，可得到副官1~4在分别执行 $OM_{0,i}(1)$ 时，每个副官都能获得决策A
- 因此，6个 $OM_{0,i}(1)$ 执行完后，副官1~4得到的决策序列都为 (A, A, A, A, A, x, y)；故最终决策为A，与将军0一样

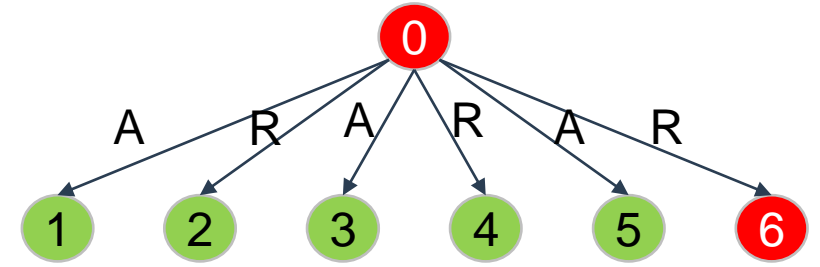




场景二：将军是叛徒

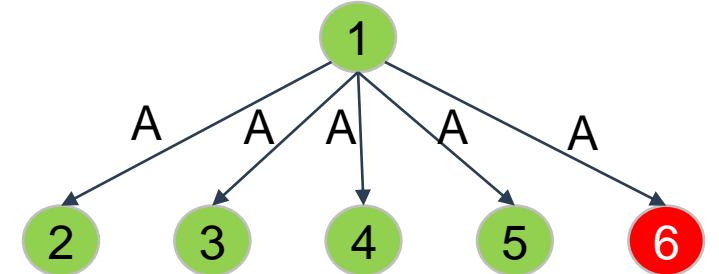
- $OM_0(2)$ 分发消息(A:attack, R:retreat)

接收 发送	1	2	3	4	5	6
0	A	R	A	R	A	R



- $OM_{0,1}(1)$ 以副官1作为将军发送消息为例

接收 发送	2	3	4	5	6
1	A	A	A	A	A



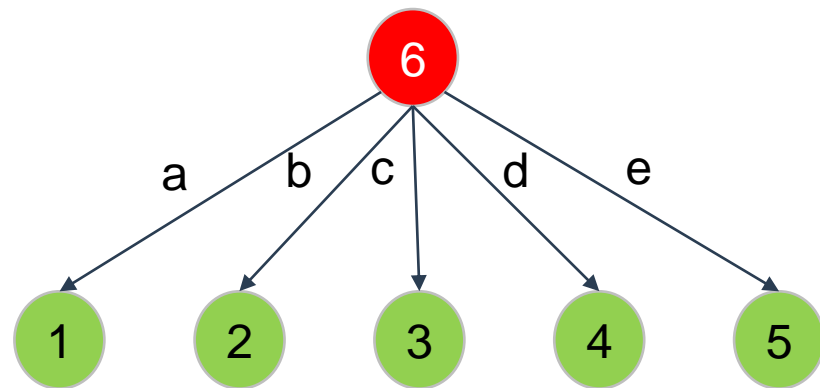
- $OM_{0,1,i}(0)$ 中2~6交换信息，显然最终2~5会认定1的决策为A
- 故副官1的 $OM_{0,1}(1)$ 过程1~5对于副官1的决策为A达成一致
- 同理，可得2,3,4,5在 $OM_{0,i}(1)$ 中的决策结果分别为R, A, R, A（由于他们是诚实的，所以同步的结果与将军发送的一致）



场景二：将军是叛徒

- 在调用完 $OM_{0,i}(1)$ 之后，1~5可以得到如下决策， x_i 是副官 i 在 $OM_{0,6}(1)$ 中从副官6处收到的值

1	2	3	4	5	6
A	R	A	R	A	x_i



- 此时只需说明 $x_1 = x_2 = x_3 = x_4 = x_5 = C$ 即可得到忠诚副官所得到的决策向量都为 (A, R, A, R, A, C) ，由此可得忠诚副官的行为一致
- 此时无论 $a \sim e$ 为何值，由于1~5为忠诚的，经过 $OM_{0,6,i}(0)$ 交换信息之后，他们得到的向量均为 (a, b, c, d, e) ，故他们对于副官6的决策是一致的



口头消息的缺点

- 开销巨大
 - $OM(m)$ 调用 $n - 1$ 次 $OM(m-1)$
 - $OM(m-1)$ 调用 $n - 2$ 次 $OM(m-2)$
 - $OM(m-2)$ 调用 $n - 3$ 次 $OM(m-3)$
 -
 - $OM(m-k)$ 被调用 $n-1, n-2, \dots, n-k$ 次
 - $O(n^m)$
- 口头消息不能溯源（无法确定消息是否被篡改），导致需要传输大量的信息



方案二：签名消息(SM)协议

改进：在实际系统中，可以引入签名机制确保节点发送的消息无法被篡改，从而得到签名消息算法SM(Signed Messages)

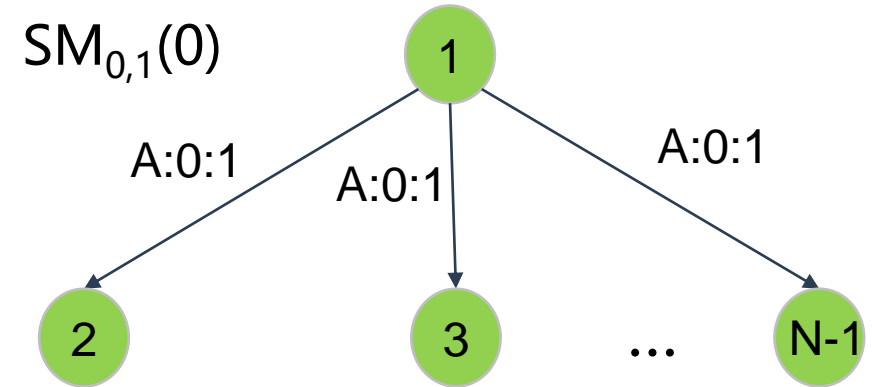
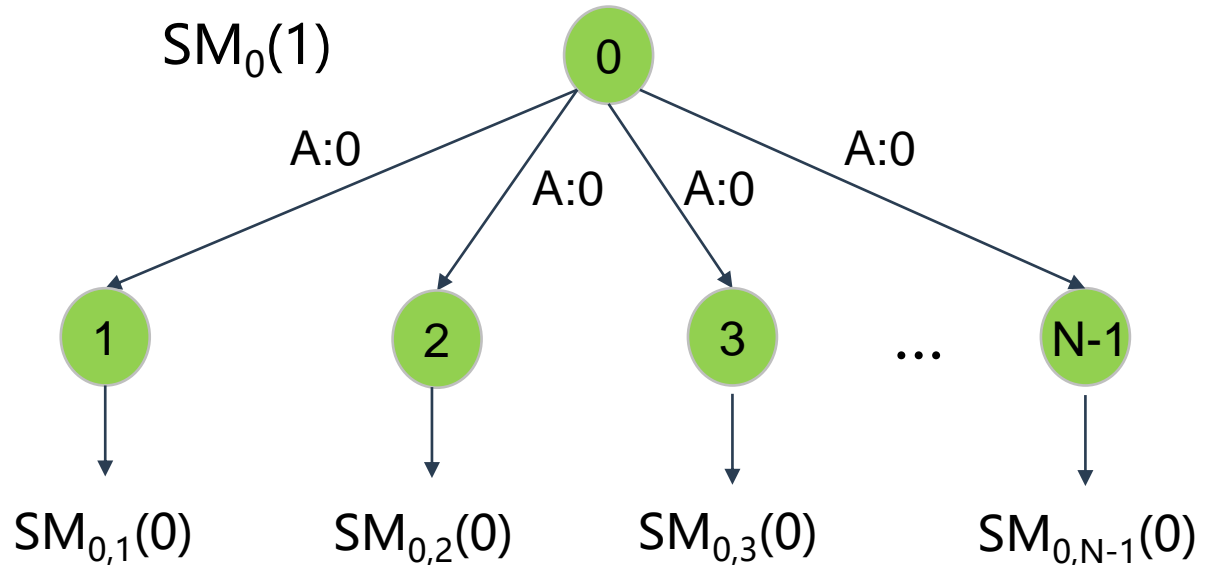
- 在口头消息协议的基础上添加假设
 - 忠诚将军的签名不可伪造
 - 任何人都可以验证将军的签名
- 确保
 - 叛徒不能通过说谎来影响共识
 - 叛徒不能通过保持沉默来影响共识

结论：假设将军个数为 N ，且最多存在 m 个叛徒时，当 $N \geq 2m + 1$ 时，SM(1) 算法可以确保所有忠诚的将军达成一致



签名消息协议

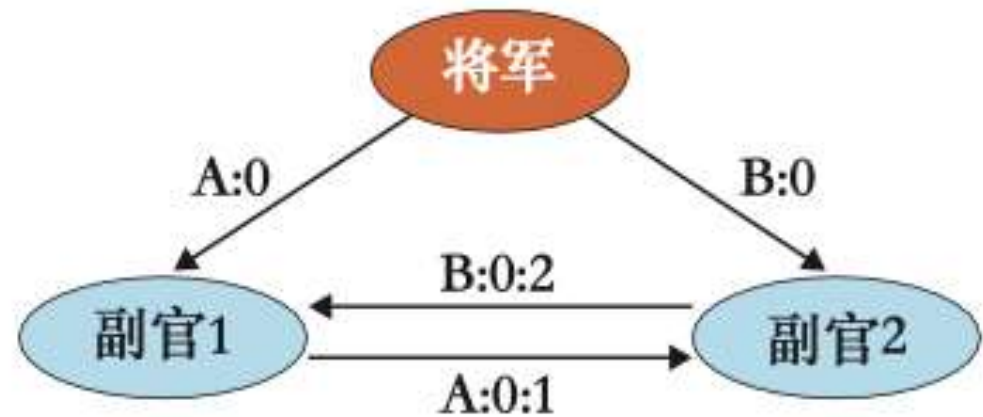
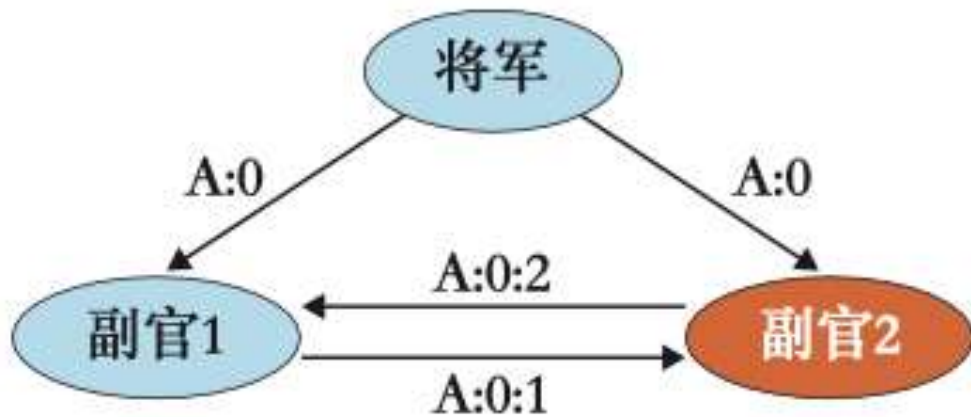
- 假设 N 个副官编号分别为 $0, 1, \dots, N-1$ ，初始状态下 0 为将军，并执行 $SM_0(1)$ ，从而给其它副官发送附带签名的指令 $x:0$
- 收到 $x:0$ 后，副官 i 执行 $SM_{0,i}(0)$ ，即在签名指令 $x:0$ 上附带自己签名得到 $x:0:i$ ，然后将其发送给其它副官
- 每个副官收集来自所有副官的指令（包括自己和将军在内），当出现不同指令时则撤退（说明将军是叛徒）；否则当超过半数为相同指令时选择该指令





实例分析: 3将军1叛徒

- 如果将军是忠诚的，副官无法伪造信息，因此副官2只能发送将军的原指令；且无论副官2是否发送指令，副官1都将收到两个指令A（自己和将军的），然后执行A，和将军一样；
- 如果将军是叛徒，若将军发送不同指令，副官通信之后，收到的消息都为{A:0, B:0}，验证签名后则发现将军叛变了，副官最终选择撤退





$2m+1$ 将军m叛徒分析

- 与3将军类似，若将军不是叛徒，则所有忠诚的副官都能收到超过半数与将军相同的指令
- 若将军为叛徒，其发送不同指令时会被发现，因此只能发送相同指令

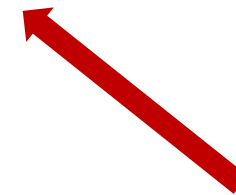
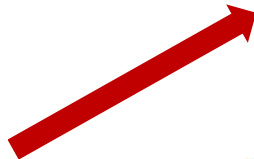
问题：若将军是叛徒且不发送指令怎么办？

解决思路：副官们可以约定一个时间，若超过该时间还未收到将军的指令，则选择废除将军，并选择新的将军。该过程在实际系统中也称为Leader更换过程，在签名消息协议的基础上进行改进，并结合Leader更换协议，就能获得现阶段实际使用的**拜占庭容错共识**



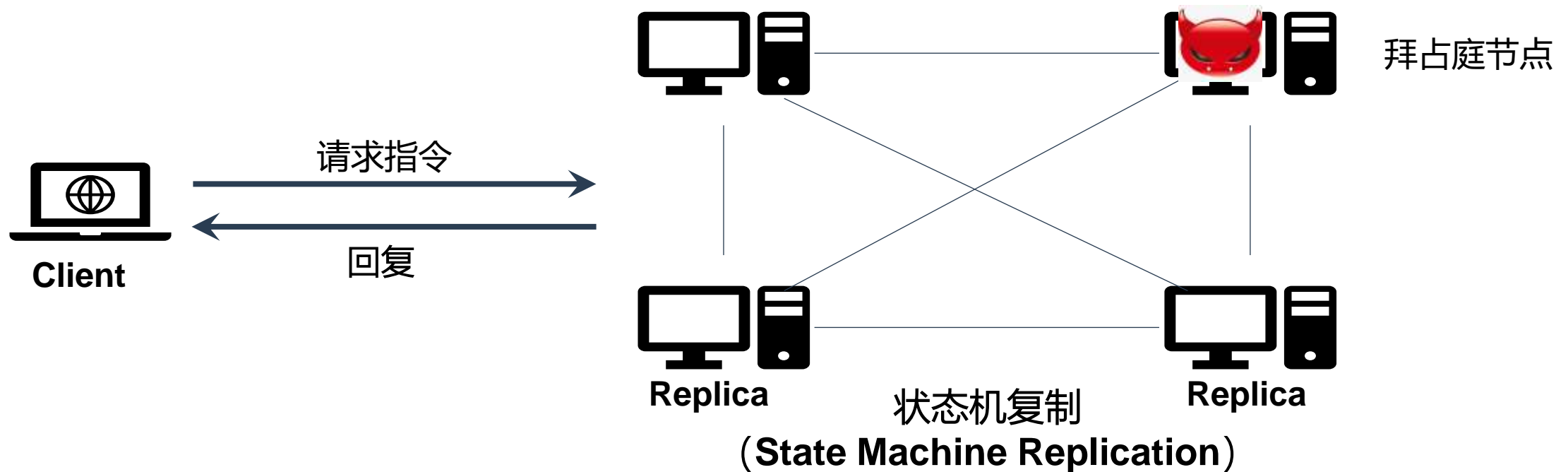
拜占庭容错共识的应用场景: 区块链

区块链：构造一个对客户端来说可信的分布式账本；容忍少数拜占庭节点作恶
解决分布式系统不同节点间协作时的信任问题





实际模型—状态机复制



- **安全性 (Safety)** : 对于客户端的所有请求, 服务端任意两个正确节点都能获得一致的排序结果并执行 (something “bad” will never happen)
- **活性 (liveness)** : 服务端最终会就结果达成一致, 因此对于任意客户端的请求, 服务端最终总能产生正确的答复 (something “good” will must happen (but we don’ t know when))



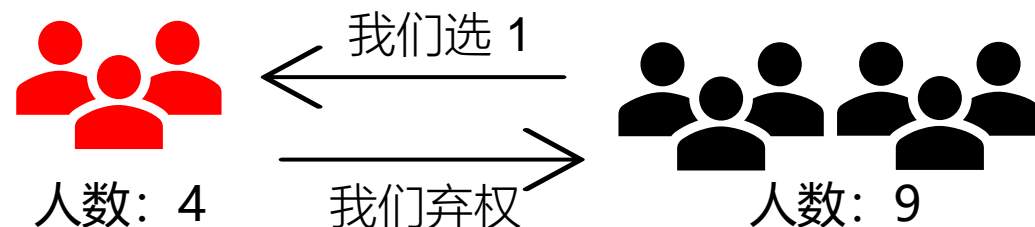
拜占庭共识：Quorum设计准则

基本规则：分布式系统每阶段提案中，诚实节点最多只对一个提案投一次票



$q=8$ ❌

$q=9$ ✅



$q=10$ ❌

$q=9$ ✅

- 目的1-安全性：Quorum值 q 应该足够大，避免通过两个**矛盾提案**

解决方案： 针对两个提案可能产生的最高总投票数 $\leq 2*q - 1$

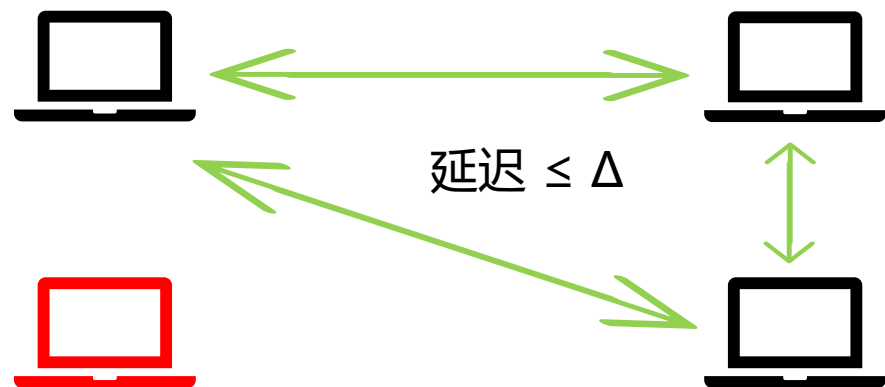
- 目的2-活性：Quorum值 q 应该足够小，避免错误节点不投票使提案**无法通过**

解决方案： $q \leq$ 最低可能产生的投票数； $q >$ 拜占庭节点个数



同步BFT共识中的Quorum设计

同步网络模型：实际网络延迟不超过**已知最大值** Δ ，即网络一直都是保持稳定状态，通过计时机制可以确定所有其他节点都正确收到了所有消息。



为了容忍 f 个拜占庭节点，Quorum的值 q 以及节点总数 N 的设计：

具体设计：系统中出现多个矛盾提案时能被检测到，只存在一个提案；

最低票数为 $N-f$ ；

边界条件： $f+1 \leq q \leq N-f \Rightarrow N \geq 2f+1, f+1 \leq q \leq N-f$

最优解： $q = f+1, N = 2f + 1$ 。



异步BFT共识中的Quorum设计

异步网络模型： 实际网络延迟不存在任何延迟上限，但节点间经过无限重传能确保消息最终被正确接收



为了容忍 f 个拜占庭节点，Quorum的值 q 以及节点总数 N 的设计：

具体设计： 针对两个提案可能产生的最高票数为 $N+f$;

最低票数为 $N-f$;

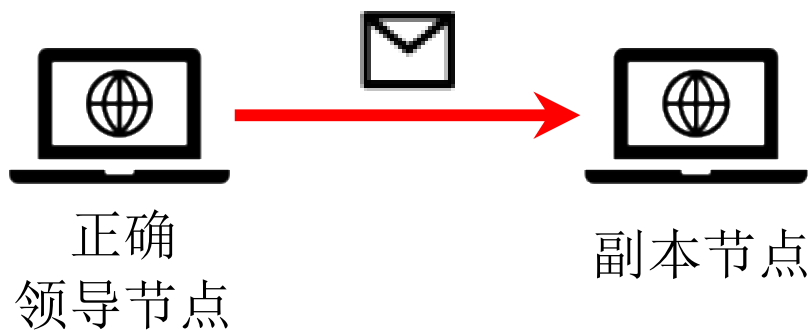
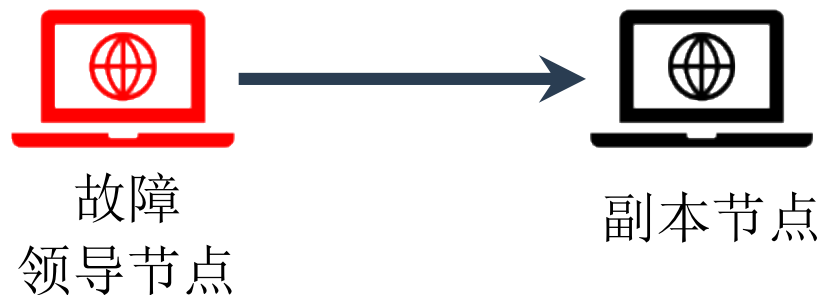
边界条件： $f+1 \leq q \leq N-f$, $N+f+1 \leq 2*q \Rightarrow N \geq 3f+1$, $2f+1 \leq q \leq N-f$

最优解： $q = 2f+1$, $N = 3f + 1$ 。



同步和异步BFT共识的缺陷

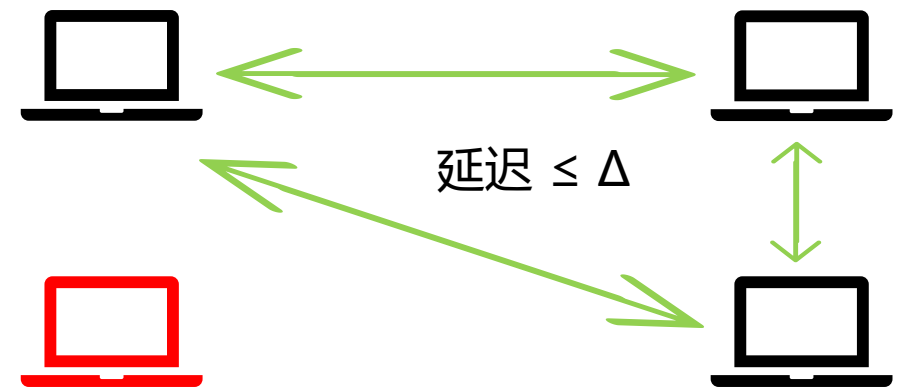
- 同步BFT共识假设网络一直保持稳定状态，若网络出现不稳定现象，即延迟超过预设最大值，则会出现安全问题
- 异步BFT共识只能采用**无Leader的共识模式**，因为异步网络中无法分辨Leader错误和网络延迟过高两种情况；因此，异步BFT共识不仅交互轮数过多，还可能导致理论上的活锁（类似于前面分析的Paxos共识）





引入部分同步网络模型

部分同步网络模型：网络状态不可预测，但总存在一个网络满足同步假设的时间窗口，此时共识能稳定运行；网络异步时，则专注于确保共识的安全性



- 部分同步BFT共识的典型示例
 - PBFT：第一个实用化的BFT共识机制；基于三阶段共识实现节点间状态一致；采用数字签名机制实现投票
 - Hotstuff：2019年被提出，是对BFT共识的一次重大改进；基于四阶段共识实现节点间状态一致，降低leader更换开销；采用门限签名机制实现投票，并采用流水线机制减少交互信息量



实用拜占庭容错-PBFT



Barbara Liskov

美国杰出计算机科学家，麻省理工学院电子电气与计算机科学系教授

2004年约翰·冯诺依曼奖得主，2008年图灵奖得主

美国第一位计算机科学女博士，历史上第二位获得图灵奖的女性

- Practical Byzantine Fault Tolerance, OSDI 1999
- 状态机复制系统
- 目标：即使在少部分服务器为拜占庭节点的情况下，使来自客户端的请求在每个服务器上都按照一个确定的顺序执行
- 适用于有强一致性要求的私有链和联盟链场景，IBM超级账本可选共识协议



PBFT的主要思想

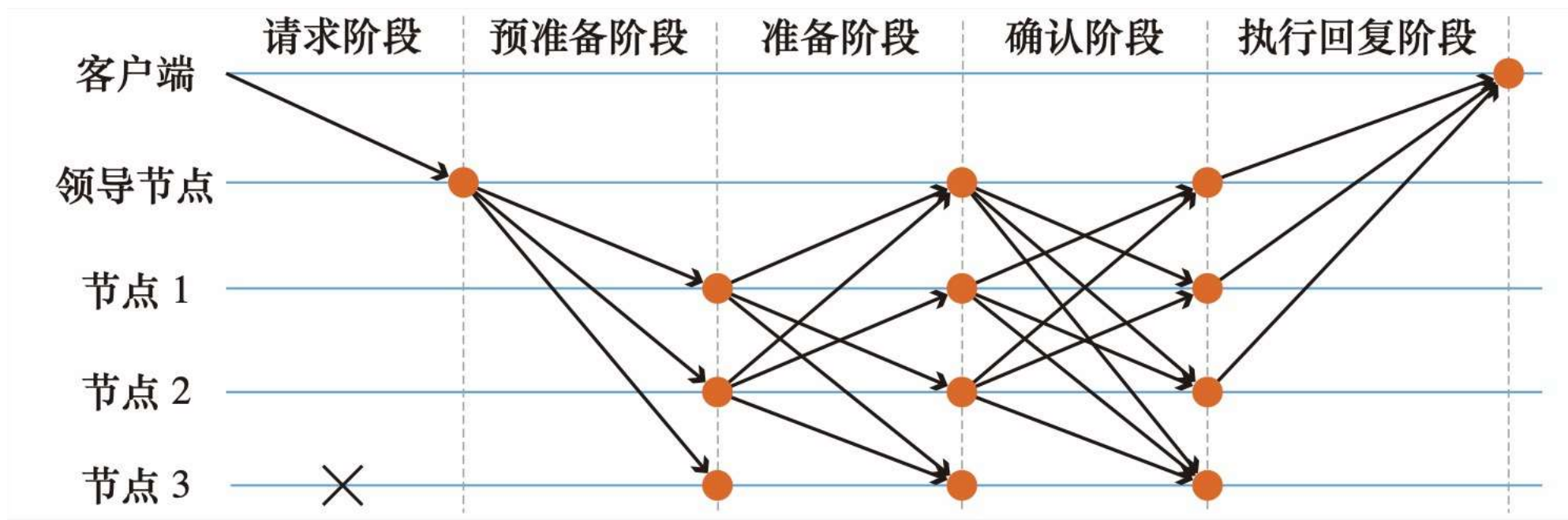
**PBFT基于Leader完成共识过程，并基于Quorum机制确保安全性和活性；
PBFT采用部分同步假设，假设故障节点为 m 个，总节点 $3m+1$ 个，
Quorum设计中， $q=2m+1$**

- PBFT在Paxos、Raft等故障容错算法的二阶段共识的基础上增加了一个阶段
- 二阶段共识后的交易处于准备完成状态，此时，节点广播信息告诉其它所有节点自己已完成准备
- 当节点收到 q 个准备完成的投票时提交并执行该交易
- 所有节点都对Leader节点进行监测，若长时间未收到Leader发送的消息，则触发视图变更协议，更换Leader节点



4节点实例分析

- Leader节点收到客户端请求后，为其分配序号，然后发送给其它节点
- 其它节点收到预准备信息后，广播一个准备信息
- 当节点收到3个准备信息，则完成准备，然后广播提交信息
- 当收到3个提交信息后，则执行相应的请求指令，并回复客户端





HotStuff: BFT Consensus with Linearity and Responsiveness

——PODC 2019

- **目标：实现具备线性复杂度的拜占庭共识**

HotStuff与libra的关系：

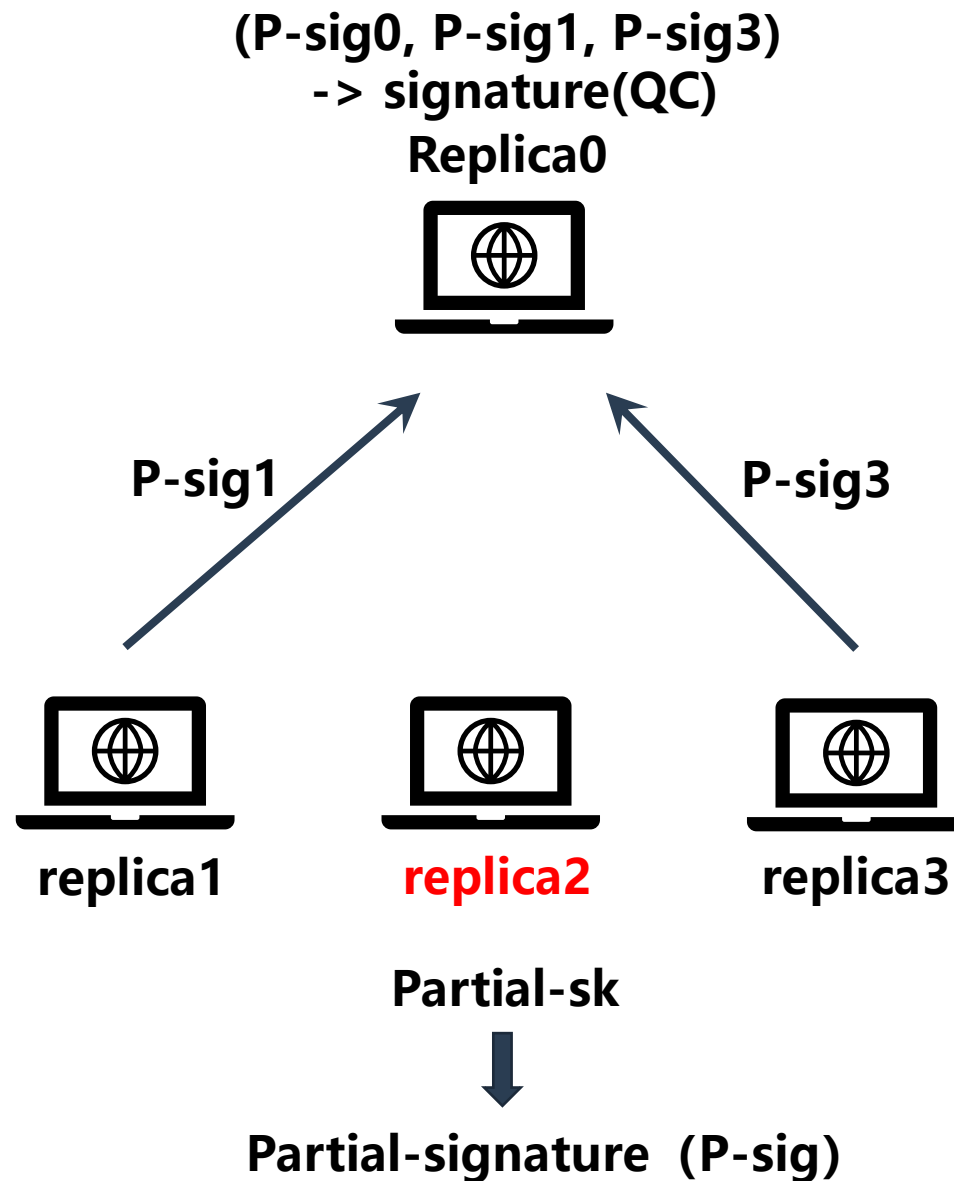
Facebook在2019年发布了libra白皮书，里面提到的LibraBFT共识是基于HotStuff演化而来





协议基础

- **门限签名**: 在 (k, n) 门限签名中, 每个节点拥有部分私钥 Partial-sk, 可进行部分签名, 任意 k 个节点的部分签名可以合成一个完整签名, 并能被全局公钥验证
- **View**: 共识视图, 每个视图由不同 leader 节点引导完成一轮共识
- **投票**: 节点对提案进行部分签名
- **QC(Quorum Certificate)**: 由 $2m+1$ 个不同节点的部分签名合成的一个全局签名, 可通过全局公钥验证并证明 $2m+1$ 个节点进行了投票

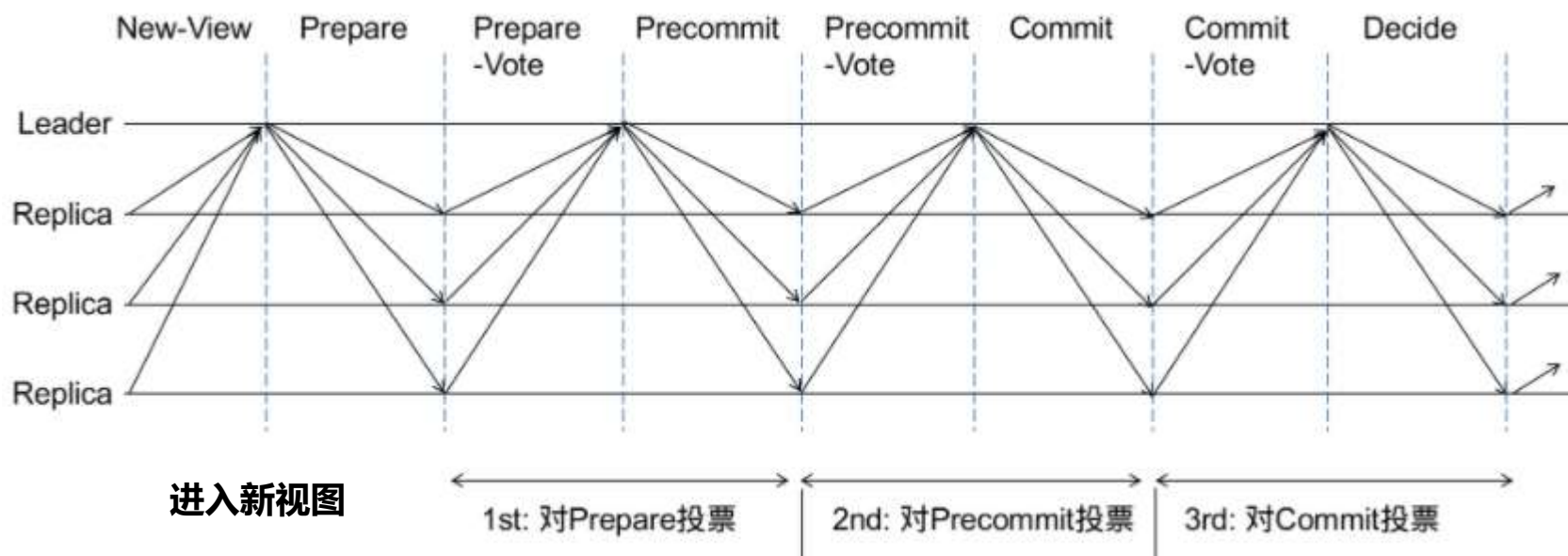




Hotstuff (基本方案)

通过门限签名把PBFT的流程改成了四阶段算法（去除了客户请求和回复客户两个阶段），存在以下特点：

- 每次交互只在Leader和其他节点间进行，无需所有节点都进行广播，减少通信复杂度 ($O(n^2) \rightarrow O(n)$)
- 每个阶段都是**同构**的

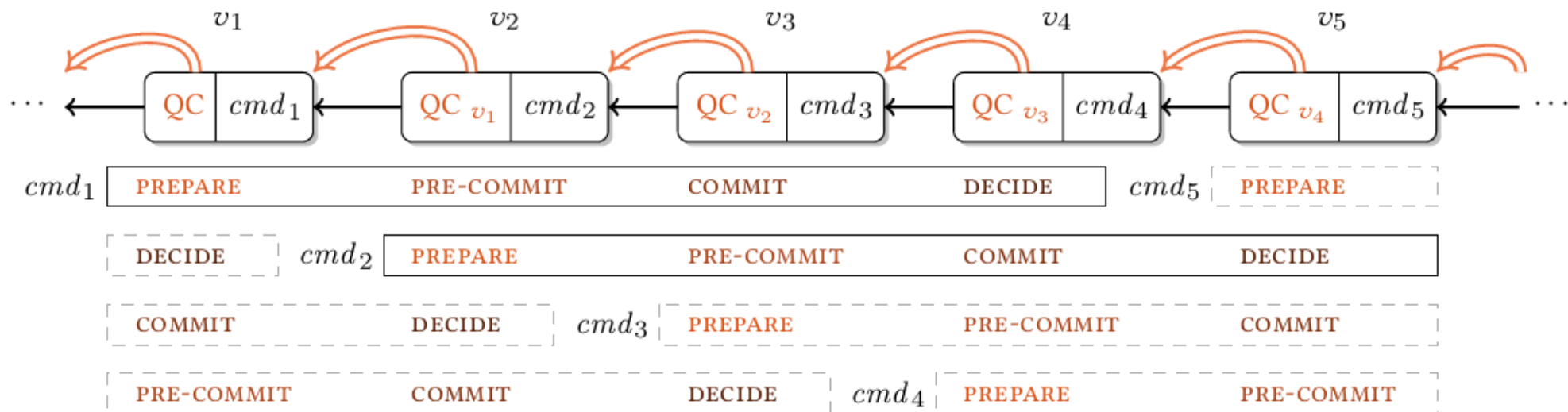




Hotstuff (流水线方案)

基本方案存在的问题：四阶段过程中，每个节点都需要广播准备、预提交、提交三次信息

解决方案：每个区块都引用上一个区块，并只进行一个阶段的投票。当区块后面链接三个区块时，相当于完成了四阶段共识，处于可执行状态

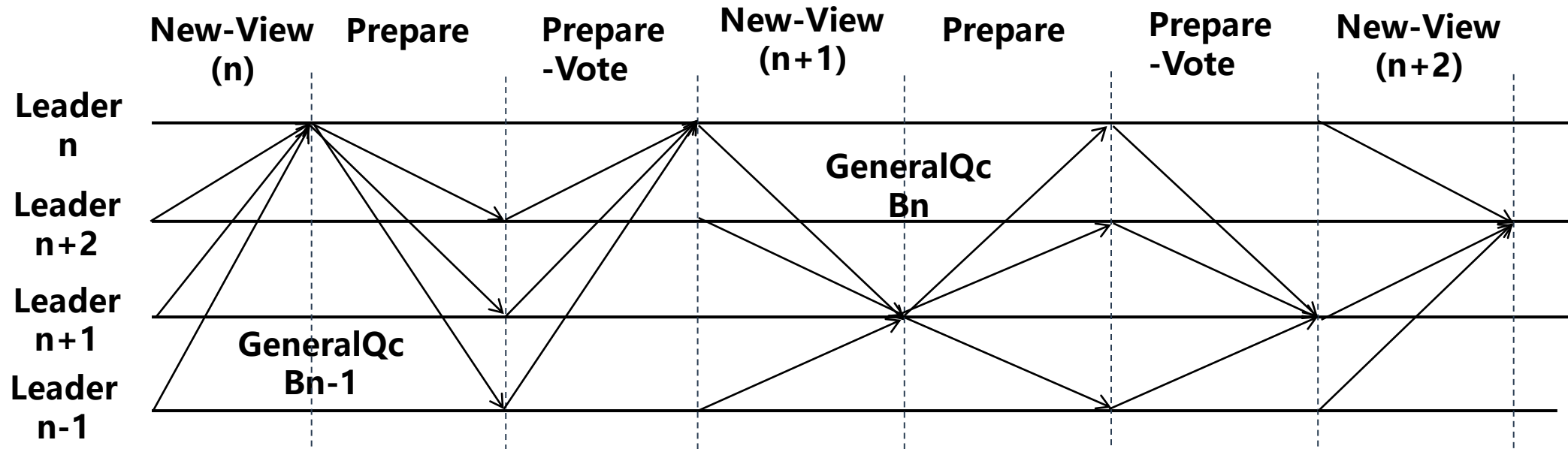




Hotstuff (流水线方案)

一阶段协议，分为三个步骤：

- 所有节点对新的Leader投票，旧Leader把获得的全局签名发给新Leader
- 新Leader将上一视图的全局签名和新区块发给所有有节点
- 节点对新区块投票并发给Leader，Leader合成全局签名





第4节 总结



总结

剖析分布式系统安全问题的根源，探讨如何从交互网络、分布式算法和信任三方面构建安全、稳定、可信的分布式系统



第一节

建立安全、稳定的交互网络

- 交互网络的组成
- 交互信道安全
- 弹性覆盖网络
- 防御日蚀攻击

了解交互网络的组成以及如何构建安全的交互网络



第二节

分布式算法

- 协作过程中的安全隐患
- 时钟同步算法
- 并发问题
- 故障容错的一致性算法

挖掘分布式系统各组件之间如何进行协同



第三节

信任问题

- 访问控制
- 信任评价模型
- 拜占庭容错

了解分布式系统中的信任问题及解决方案



展望

新的应用场景对分布式系统的研究提出了新的挑战



- 随着比特币的出现，**区块链**技术得到飞速发展，但是如何进一步提升区块链性能并拓展其应用范围对分布式系统研究提供了新的方向
- 近些年来，为了打破数据孤岛，**联邦学习**被提出并得到发展，如何协调互不信任的组织之间实现可信数据协同成为新的挑战
- 互联网的发展**暴露了传统设计的弊端**，如何构建可信互联网基础设施至关重要