6.分布式同步控制

1.物理时钟同步

分布式协同处理：基于真实时间的同步

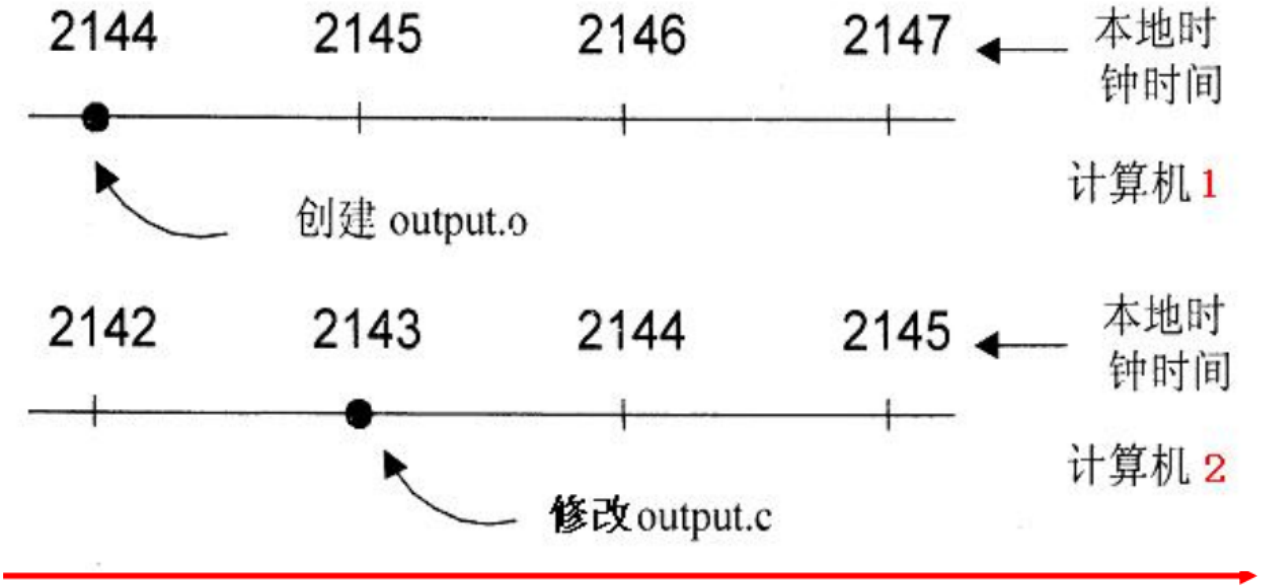
分布式算法的特点：

- 相关信息分布在多个场地上
- 应避免因单点失败造成整个系统的失败
- 不存在公共时钟或精确的全局时间

1.1 时钟同步问题

makefile 误差

两计算机本地时钟不一致导致先后顺序错乱



1.2 时钟同步算法

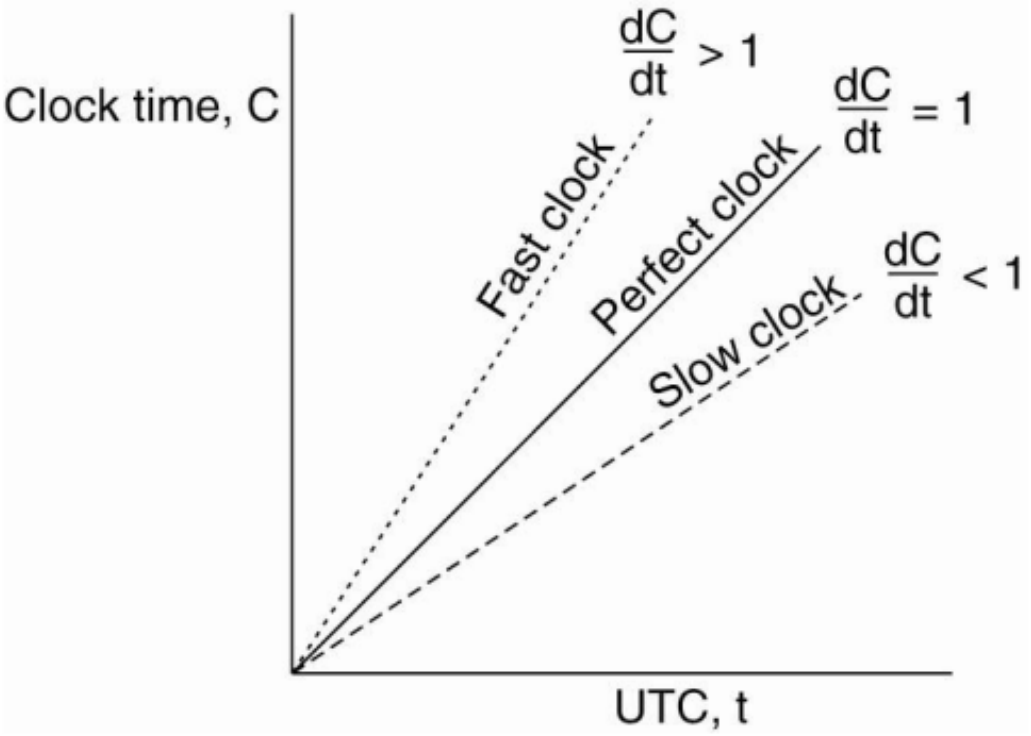
同步问题：

- 如何与现实时钟同步
- 如何使不同机器之间相互同步

设进程 P 的机器时钟值 $C_p(t)$：t 为 UTC 时间

最大偏移率（ρ）

- 精确时钟（理想情况）：$C_p(t) = t$，即 $dC/dt = 1$
- 快时钟：$dC/dt > 1$
- 慢时钟：$dC/dt < 1$



时钟校正：

- 设时钟偏移率为 ρ，两个时钟之间的允许误差为 δ
- 则 Δt 后，最大可能误差为 $2\rho\Delta t$
- 为了保证 $2\rho\Delta t \leq \delta$，则 $\Delta t \leq \delta/2\rho$；即每隔 $\delta/2\rho$ 应该校准时间

校准原则：单调递增

- 假设：每秒产生 100 次中断，每次中断将时间加 10 毫秒
- 若调慢时钟，中断服务程序每次只加 9 毫秒

- 若加快时钟，每次加 11 毫秒

1.3 网络时间协议

（1）Christian 算法

- 时间服务器， 可接受 WWV 的 UTC 时间
- 每隔 $\delta/(2\rho)$ ， 客户机向服务器询问时间
- 服务器返回 CUTC
- 客户机校正自己时间

（2）考虑的问题

时间服务请求过程参数

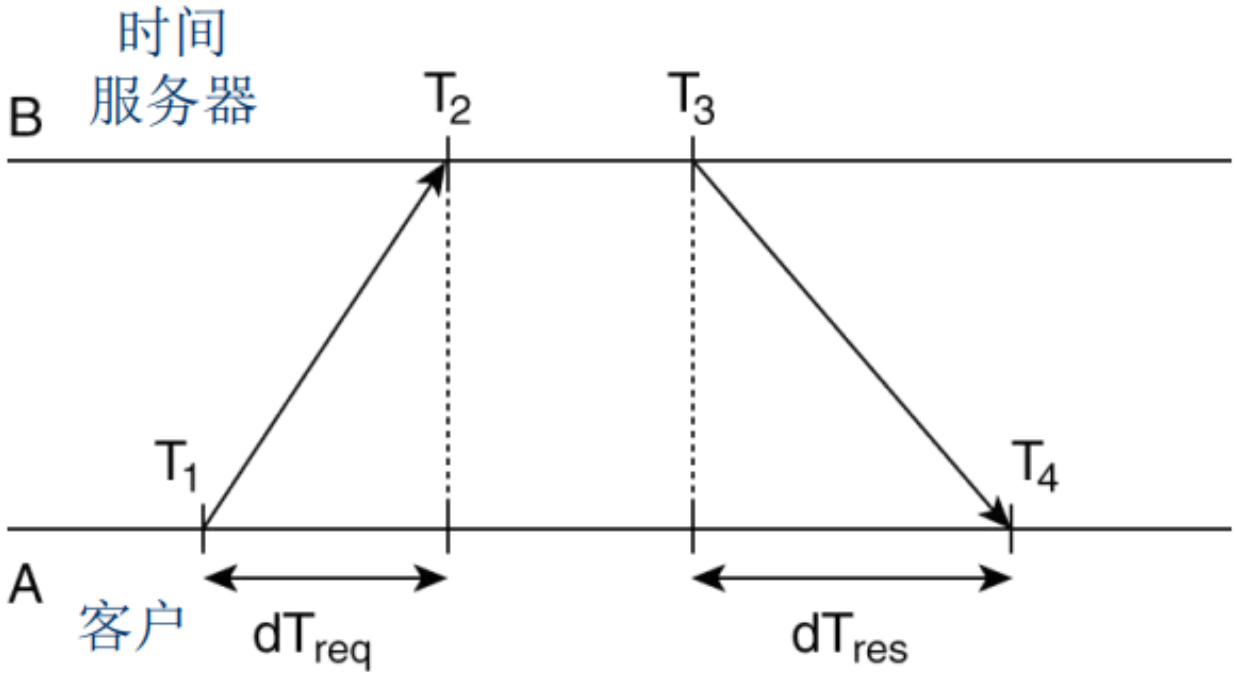
- T_1 ： A 请求时间
- T_2 ： B 接收时间
- T_3 ： B 发送时间
- T_4：A 接收时间

传输延时

- 假定双向路径相同
- $dT_{req} \approx dT_{res}$
- 平均传输时延 $\delta = (dT_{req} + dT_{res})/2 \approx dT_{req} \approx dT_{res}$

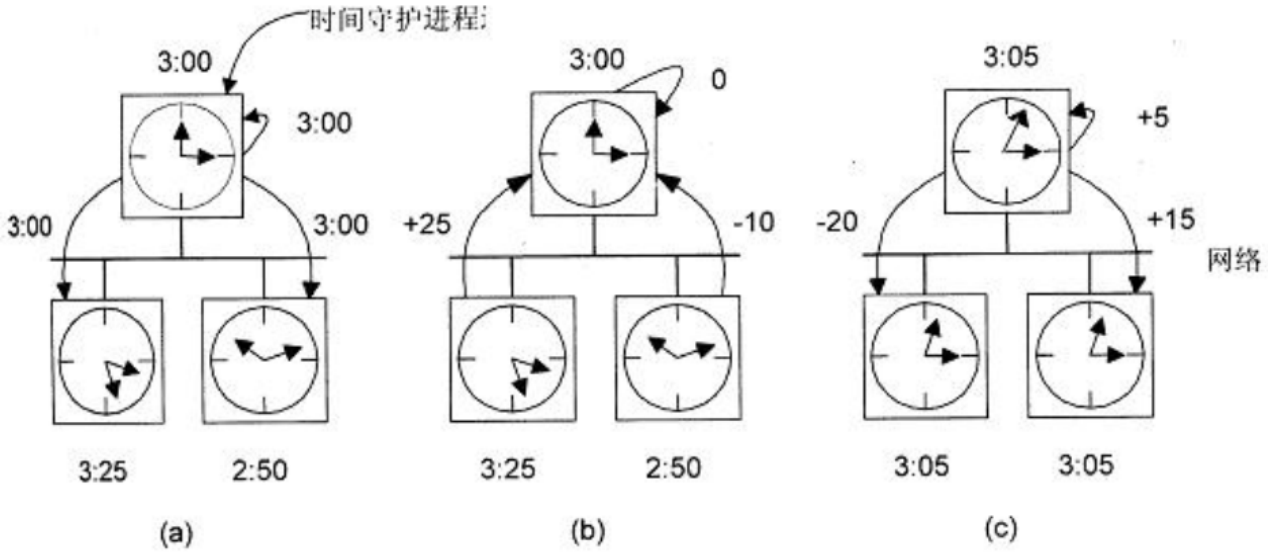
时间偏差 θ

- $T_1 = T_2 - dT_{req} + \theta$
- $T_4 = T_3 + dT_{res} + \theta$
- $\theta = T_4 - T_3 - dT_{res} = T_4 - T_3 - \delta = T_4 - T_3 - \frac{(T_2 - T_1 + \theta) + (T_4 - T_3 - \theta)}{2} = \frac{(T_1 - T_2) + (T_4 - T_3)}{2}$



（3）Berkeley 算法–集中式方法

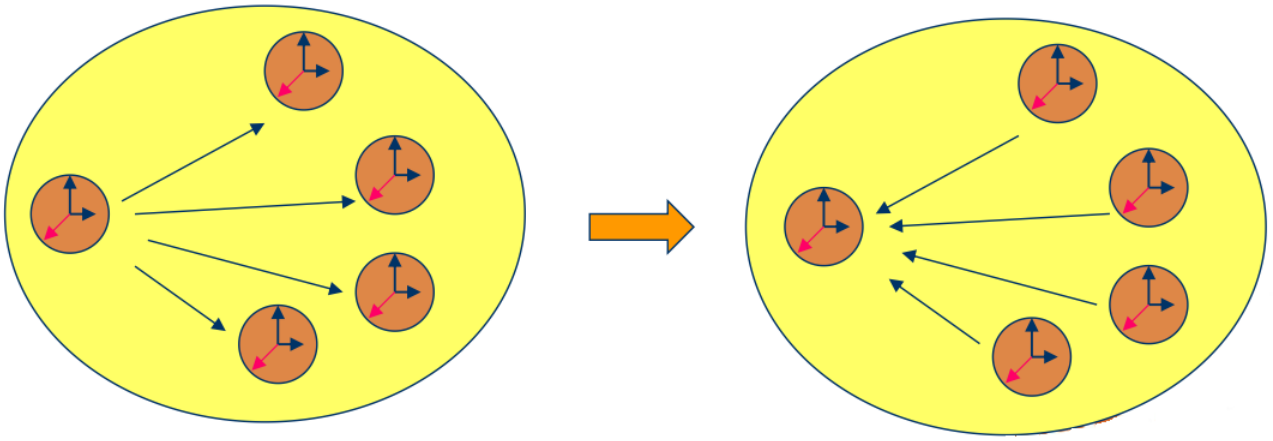
1. 时间监控器定期查询其他机器时间
2. 计算出平均值
3. 通知其他机器调整时间



（4）平均值算法–非集中式方法

1. 划分固定时间间隔 R
2. 在每个间隔，所有机器广播自己的时钟时间
3. 启动本地计时器手机在 S 时间间隔中到达的其他机器广播的时间

4. 执行平均时间计算算法，得到新的时间值

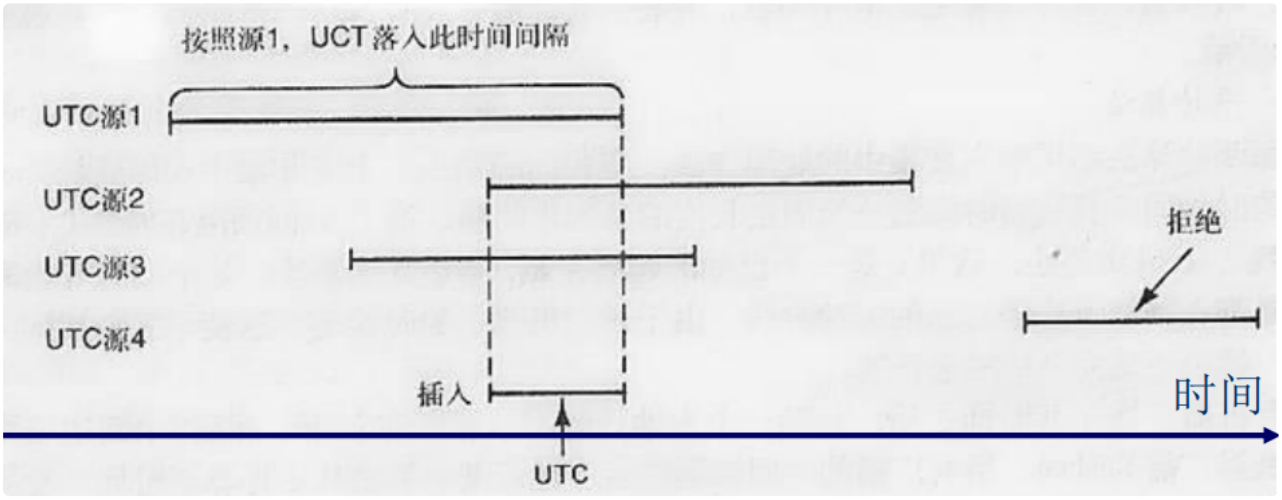


（5）多重外部时间源法

消除传播延迟造成的误差

例：OSF DCE 方法

1. 接收所有时间源的当前 UTC 区间
2. 去掉与其他区间不相交的区间
3. 将相交部分的中间作为校准时间



（6）无线网络中的时间同步

传统分布式系统特点：

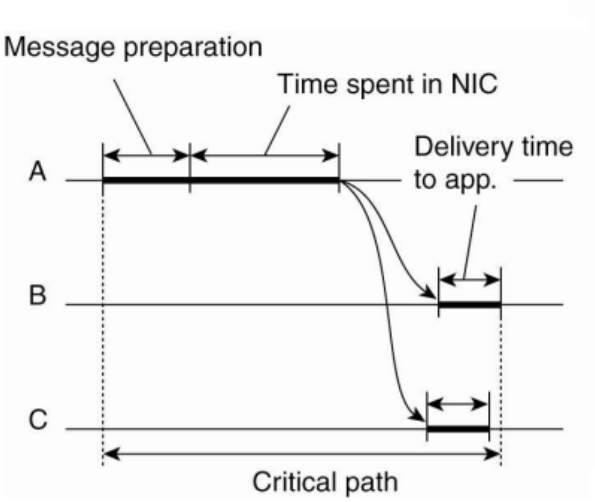
- 时间服务器容易部署
- 机器相互联系
- 双向协议

无线网络系统特点

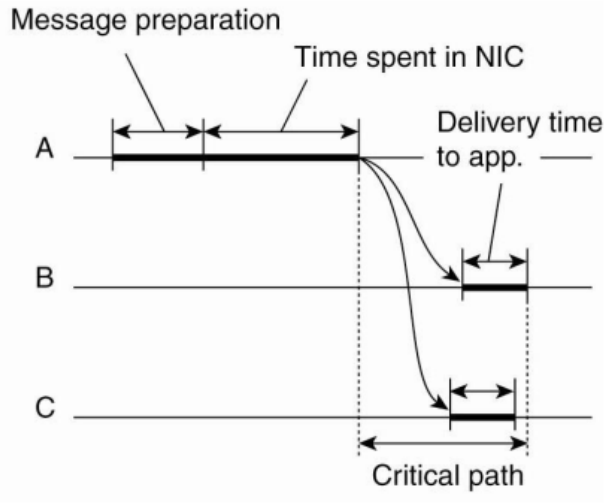
- 结点资源受限
- 多跳路由器代价高

参考广播同步协议（RBS）

- 没有具体精确时间结点
- 目标：接收器之间相对同步



普通的网络延时关键路径



RBS的网络延时关键路径

（7）参考广播同步协议（RBS）

一个节点广播一个消息 m 后，其他节点记录本地接收时间 $T_{p,m}$ 。P 和 Q 交换各自的接收时间，计算相互偏差

$$偏差[p,q]=\frac{\sum_{k=1}^{M}(T_{p.k}-T_{q,k})}{M}\quad M=\{m_1,m_2,...,m_k\}$$

偏移量随时间增大，采用线性回归方法估计

$$偏差[p,q](t)=\alpha t+\beta$$

其中，系数 α 、β 由 $(T_{p.k},T_{q,k})$ 对计算确定

2.逻辑时钟同步

2.1 基本概念

物理时钟：真实事件

逻辑时钟：相对时间

确定事件的先后顺序，而不精确到事件。例子： 记录 input.c 的版本号， 而不是物理时间， 和 input.o 进行版本比对。

“之前”关系（happens-before）： $\rightarrow$

- 同一进程：事件 a 在 b 之前出现，则： $a \rightarrow b$
- 不同进程：a 为发送消息 m，b 为接收 m，则： $a \rightarrow b$
- 具有传递性： $a \rightarrow b$, $b \rightarrow c$ ，则 $a \rightarrow c$

并发事件（concurrent）：

- 两个事件相互对立。既不 $a \rightarrow b$ ，不 $b \rightarrow a$

2.2 Lamport 算法：校正算法

（1）Lamport 算法

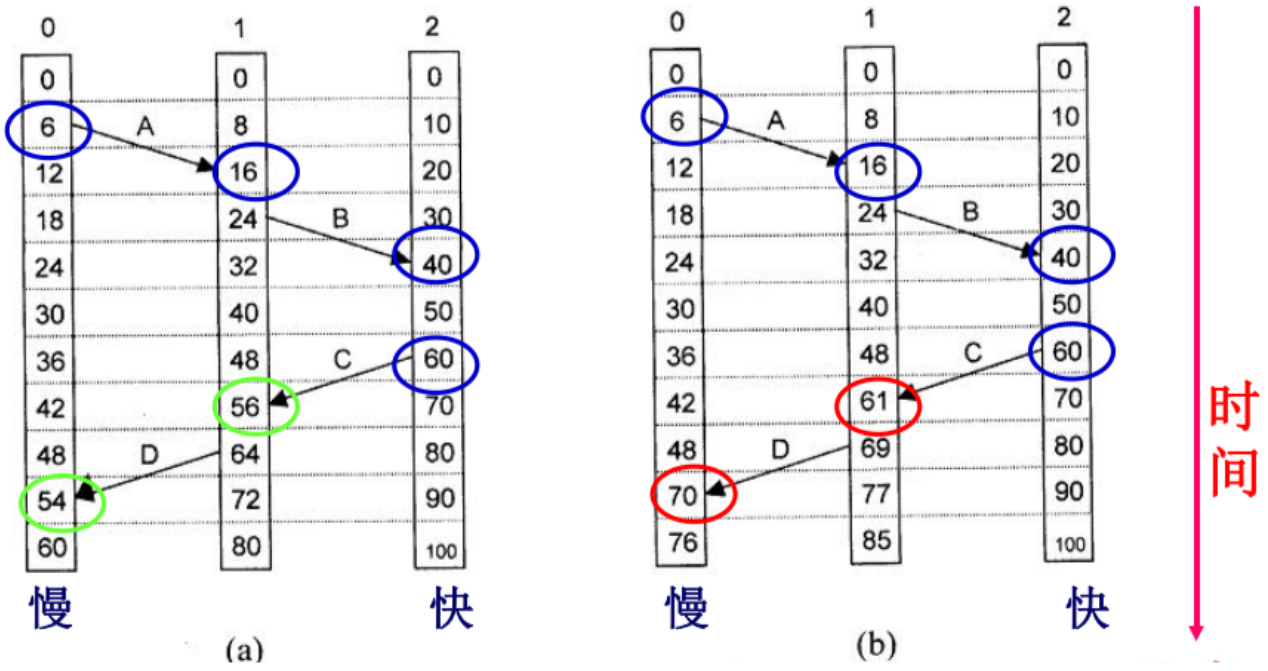
C(a)表示事件 a 的时钟值。性质：

- if $a \rightarrow b$, then $C(a) < C(b)$
- $\forall a, b \quad C(a) \neq C(b)$
- C 是递增的

校正算法

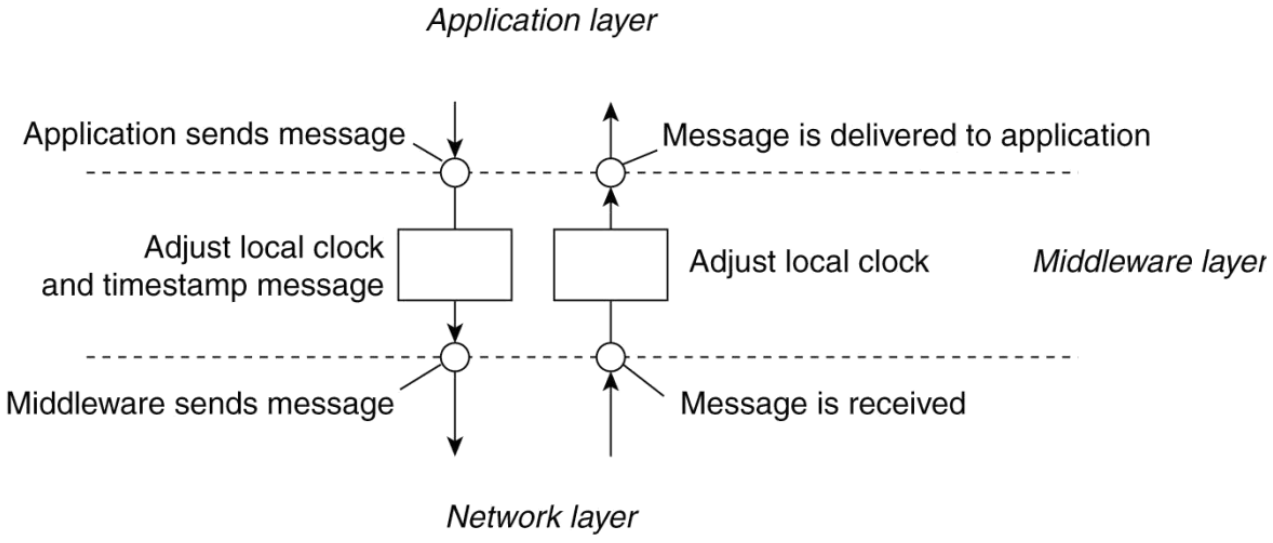
- $a \rightarrow b$,
- if $C(b) < C(a)$, then $C(b) = C(a) + 1$

三个进程，各有自己的局部时钟，他们速率不同；通过 Lamport 算法，校正时钟



校正算法：

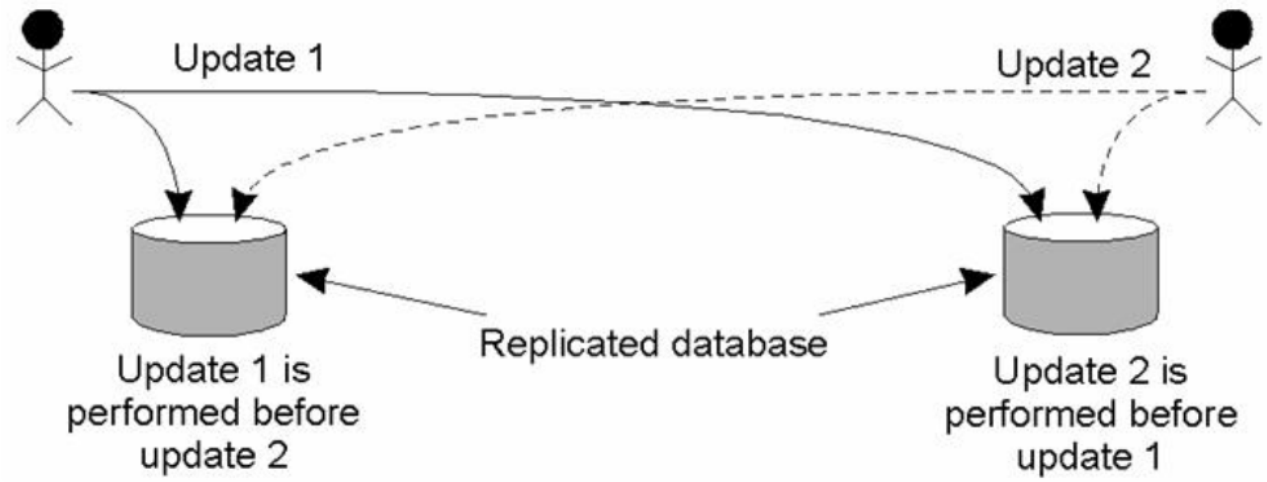
1. P_i 在执行一个事件之前， P_i 执行 $C_i \leftarrow C_i + 1$
2. P_i 在发送消息 m 给 P_j 时，时间戳 $ts(m) \leftarrow C_i$
3. P_j 接收到消息 m 后， $C_j \leftarrow max\{C_j, ts(m)\}$



(2) 全序多播：Lamport 应用示例

问题：两个进程分别对同一个复制数据库进行更新时，造成不一致状态

原因：全局次序不一致。 $u1 \rightarrow u2; u2 \rightarrow u1$



解决方案：全序多播

- 发送进程多播发送消息 m 时，给 m 带上当前时间戳 ts
- 当接收进程收到消息 m 后，存放其局部队列 q，并按时间戳排序
- 接收进程向所有进程多播发送应答
- 当消息 m 被所有进程应答，且排在队列 q 队首后， 方可处理（递交给接收进程，从队列中删除）

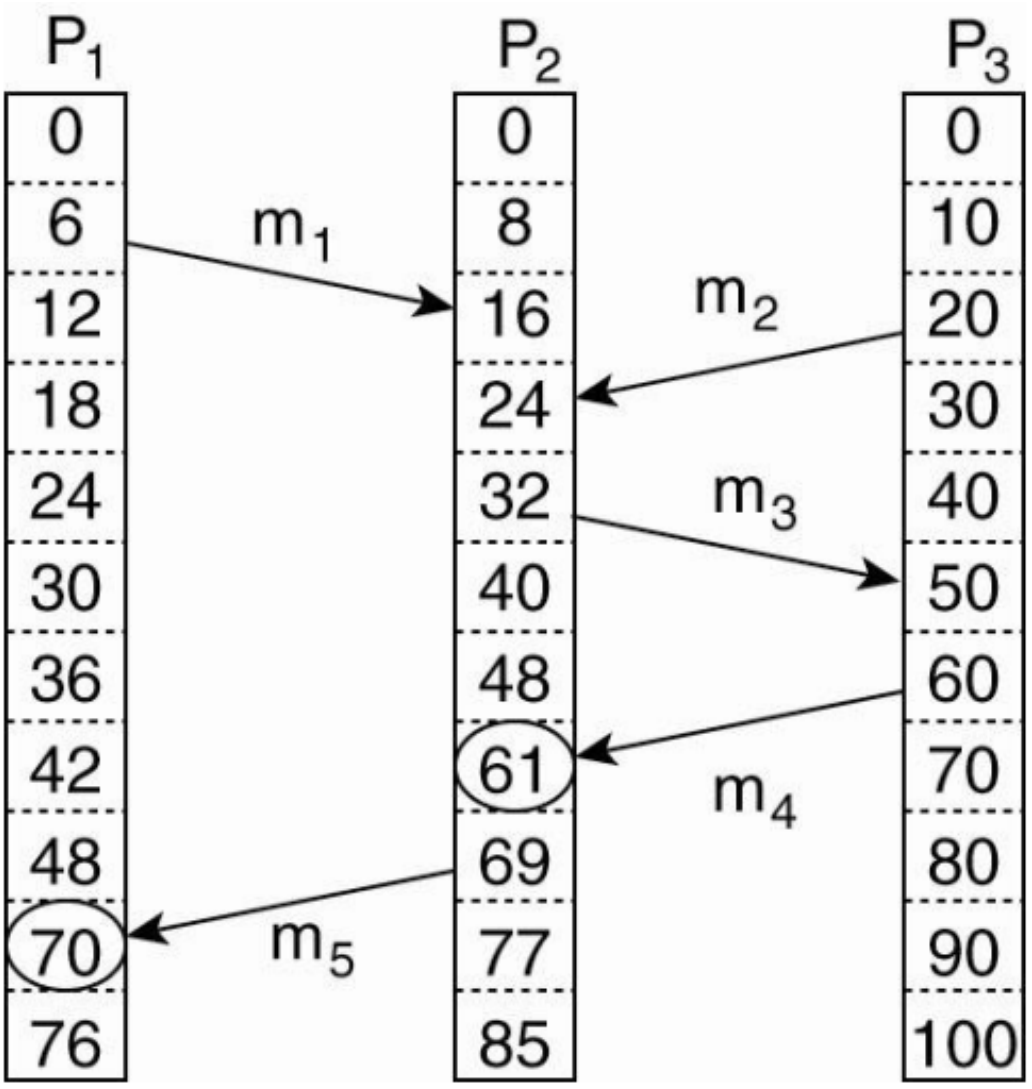
定理：各个进程的局部队列的值最终都相同

2.3 向量时钟

（1）因果性

如果事件 a， b 存在因果关系， a 为因， b 为果， 则 $C(a) < C(b)$； 但反之不一定成立。

通过向量时钟捕获因果关系



（2）Vector Clock

如果 $VC(a) < VC(b)$， 则 a 与 b 为因果关系

（3）进程 Pi 上的向量时钟 VC_i 的基本性质

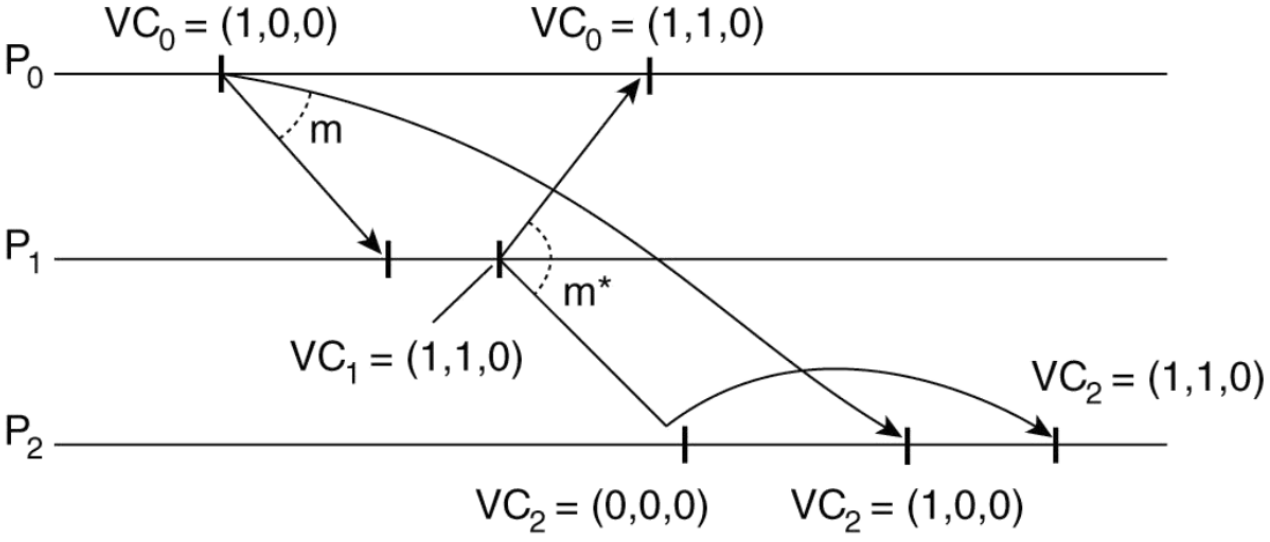
1. $VC_i[i] = n$, 在 Pi 中发生了 n 个事件
2. $VC_i[j] = k$， Pi 已知在 Pj 中发生了 k 个事件

（4）向量修改规则

1. Pi 在执行一个事件之前， Pi 执行 $VC_i[i] \leftarrow VC_i[i] + 1$
2. 当进程 Pi 发送消息 m 时, $ts(m) = VC_i$
3. 当进程 Pj 收到 m 后,置 $VC_j[k] = maxVC_j[k], ts(m)[k]$

（4）Pi 的消息 m 在进程 Pk 正确递交的条件:

- $ts(m)[i] = VC_k[i] + 1$
- $ts[m][j] \leq VC_k[j] \ \ for \ \ all \ \ i \neq j$ （符合因果关系）



3.互斥控制

3.1 基本概念

互斥访问：当一个进程使用某个共享资源，其他进程不允许对这个资源操作

临界区：对共享资源进行操作的程序段

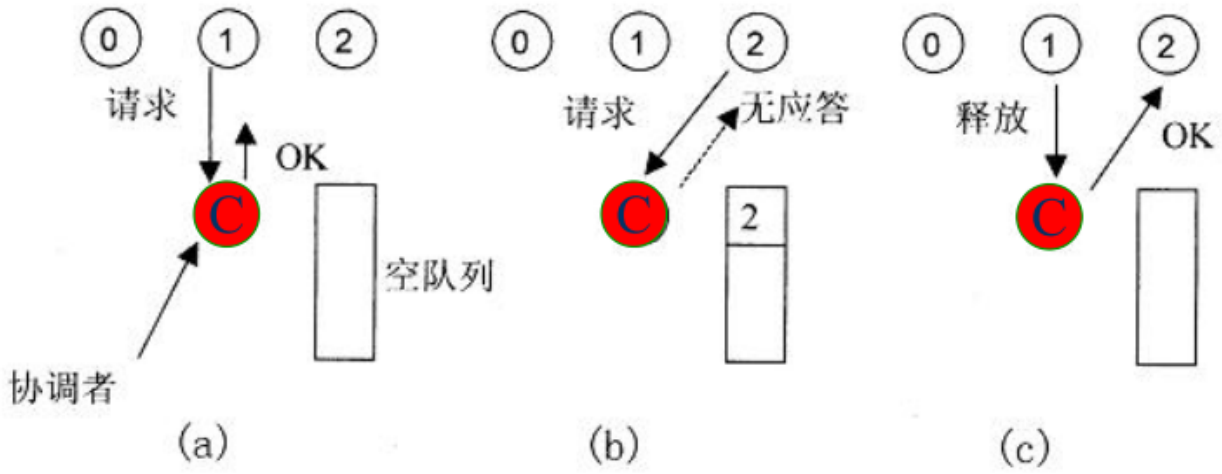
基本方法：信号量、管程

问题：死锁、饥饿

3.2 集中式算法

协调者：确定那个进程可进入临界区

通信量：3 个消息：请求–许可–释放



优点：通信量少，实现简单，不会死锁、饿死

缺点：单点失败；单点瓶颈（大规模系统中）

3.3 分布式算法（Ricart–Agrawala 算法）

（1）算法

在一个进程 P 打算进入临界区 R 之前，向所有其他进 程广播消息 ＜临界区R名、进程号、时间戳＞

当一个进程 P’收到消息后，做如下决定：

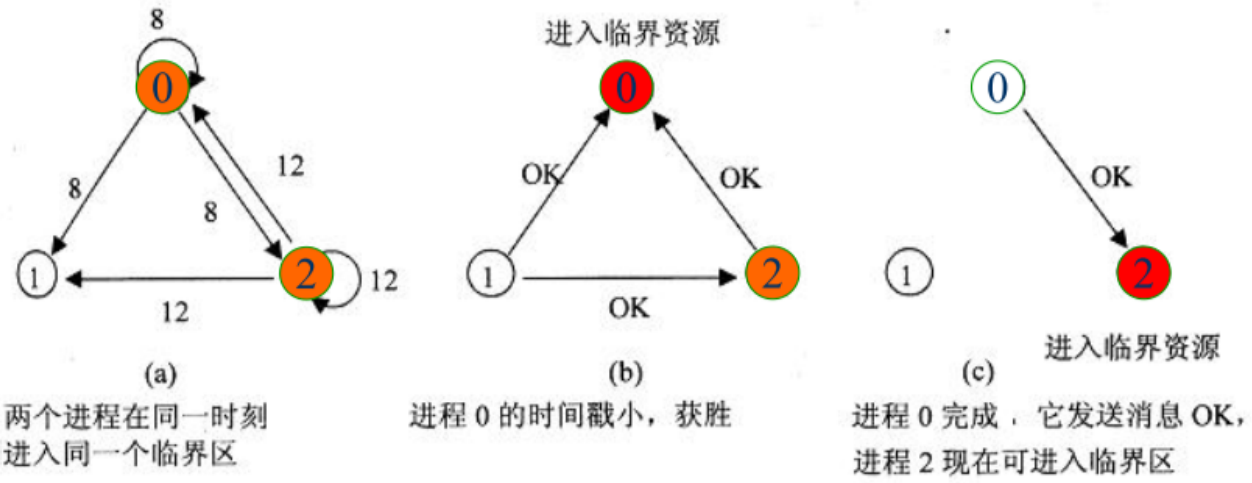
- 若 P’不在临界区 R 中，也不想进入 R，它就向 P 发送 OK；
- 若 P’已经在临界区 R 中，则不回答，并将 P 放入请求队列；
- 若 P’也同时要进入临界区 R，但是还没有进入时，则将发来的消息和它发送给其余进程的时间戳对比。如果 P 时间戳小， 则向 P 发送 OK；否则，不回答，并将 P 放入请求队列；

当 P 收到所有的 OK 消息后，进入 R。否则，等待。

当 P 退出 R 时，如果存在等待队列，则取出全部请求者，向其发送 OK 消息

（2）举例

共有 0，1，2 三个进程，进程 0，2 申请进入临界区



（3）算法评价

优点：不会死锁和饿死

缺点：

- n 点失败
- n 点瓶颈
- $2(n-1)$ 个消息

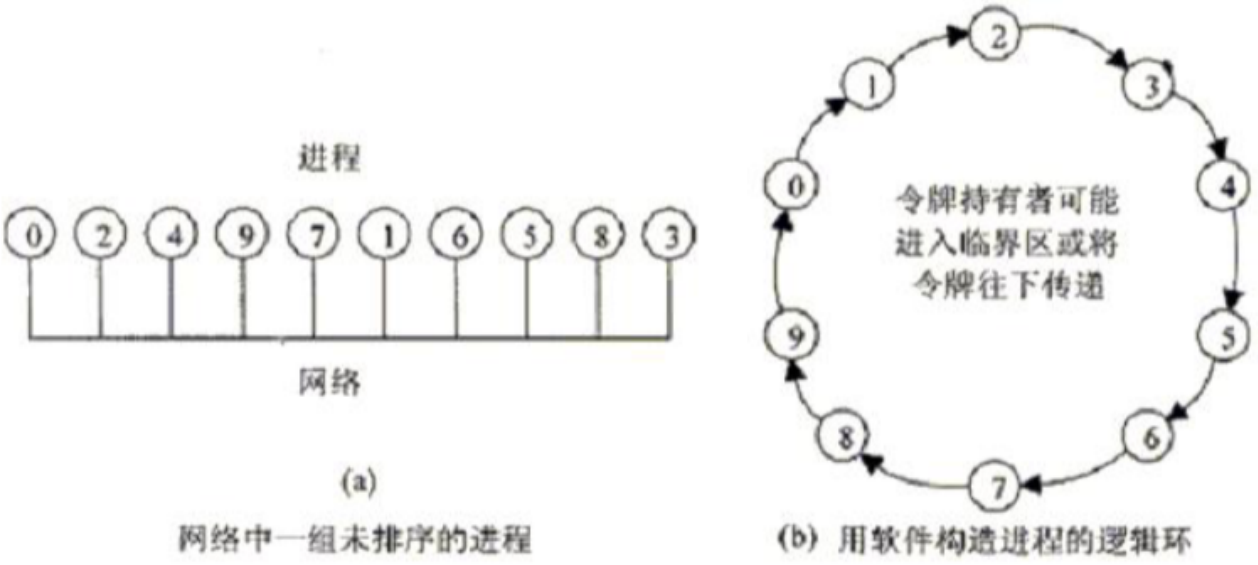
改进方案：

- 总是发送应答
- 超时重发请求
- 组通信（进程少且不改变组成员时）
- 简单多数同意（>1/2）

3.4 令牌环算法

构造一个逻辑环，得到令牌才可进入临界区

问题：令牌丢失检测



3.5 三种互斥算法的比较

算法	每次进出需要的消息	进入前的延迟（按消息次数）	存在问题
集中式	3	2	协调者崩溃
分布式	2(n–1)	2(n–1)	n 点崩溃
令牌环	1 到 ∞	0 到 n–1	丢失令牌，进程崩溃

4.选举算法

4.1 基本概念

作用：

- 在分布式进程之间做出统一的的决定
- 例如：确定协调者

前提：

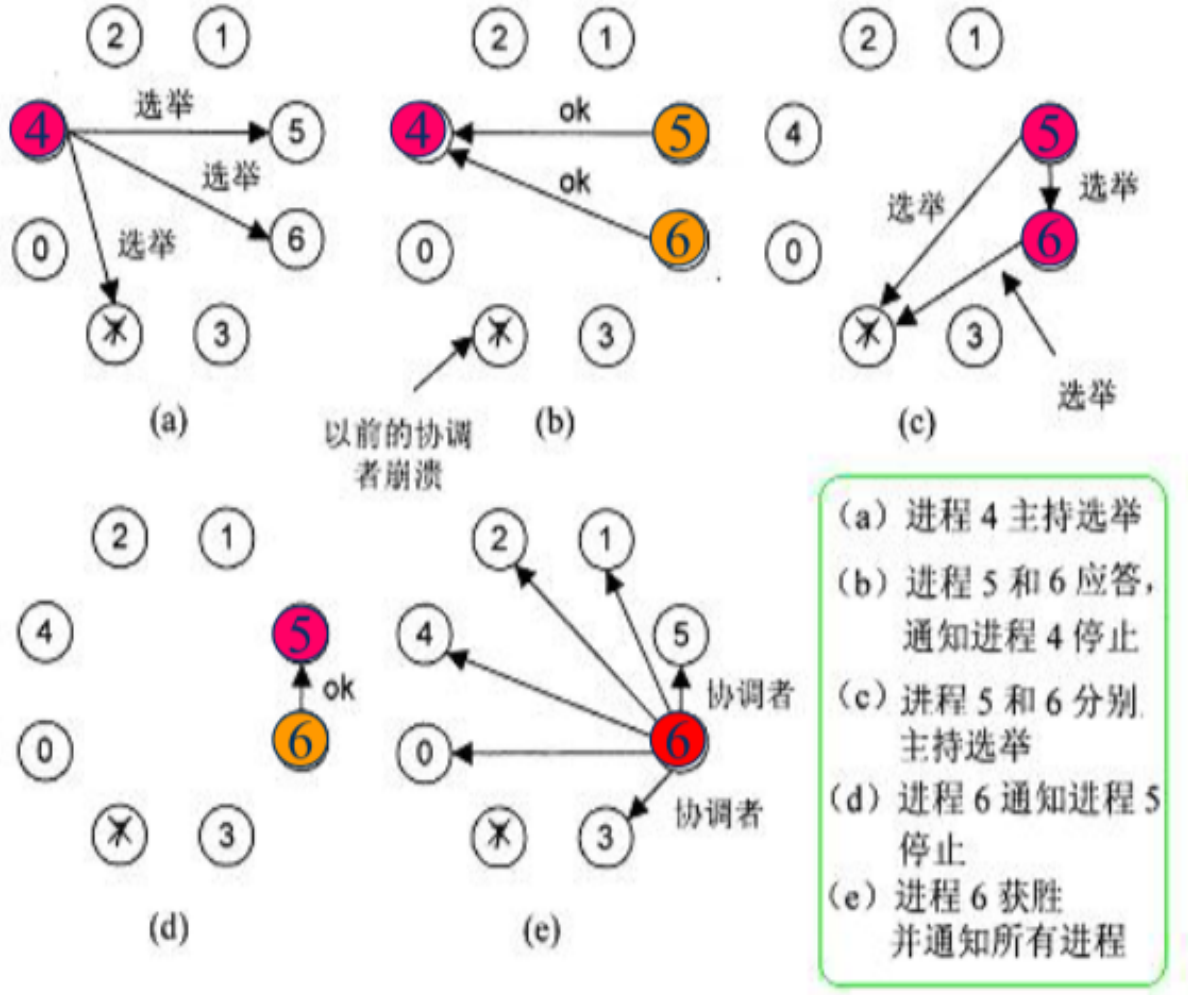
- 每个进程具有唯一的号码，如 IP 地址
- 每个进程知道其它进程的号码

选举标准：确定具有最大号码的进程

4.2 霸道（Bully）算法

将进程进行排序

1. P 向号码高的进程发 E 消息
2. 如果没有响应，P 选举获胜
3. 如果有进程 Q 响应，则 P 结束，Q 接管选举并继续下去。

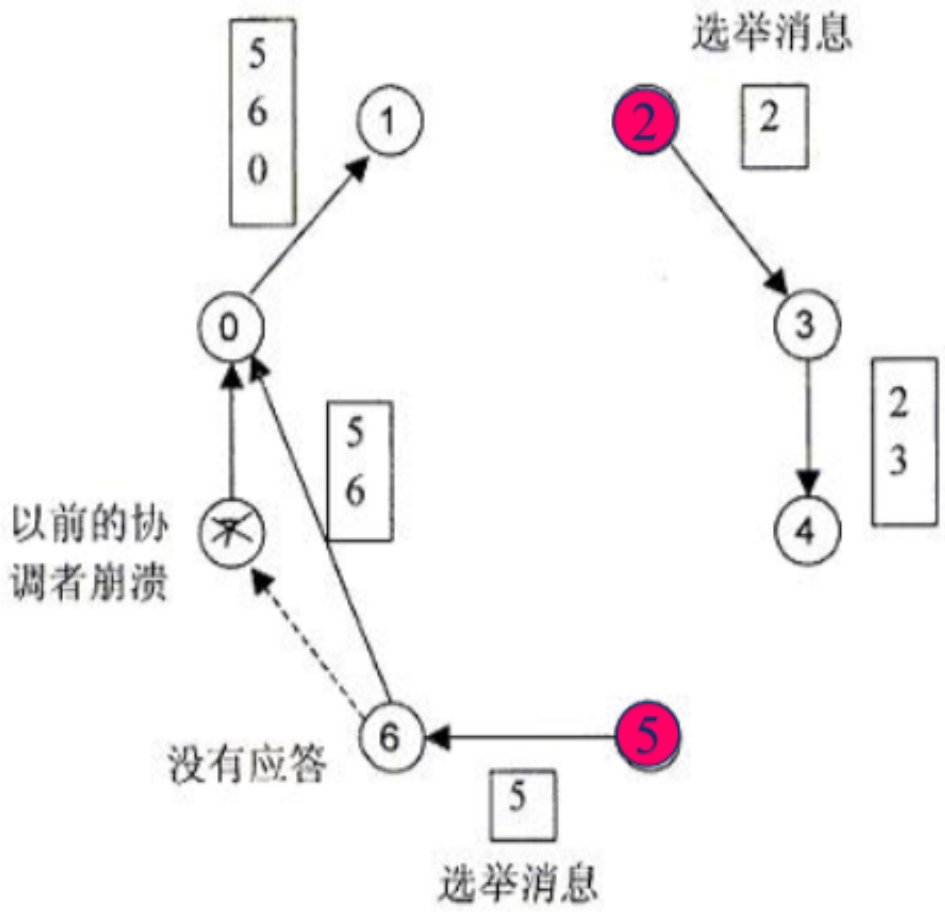


4.3 环算法

所有进程按逻辑或物理次序排序，形成一个环

1. 当一个进程 P 发现协调者 C 失效后，向后续进程发送 E 消息
2. 每个进程继续向后传递 E 消息，直到返回 P

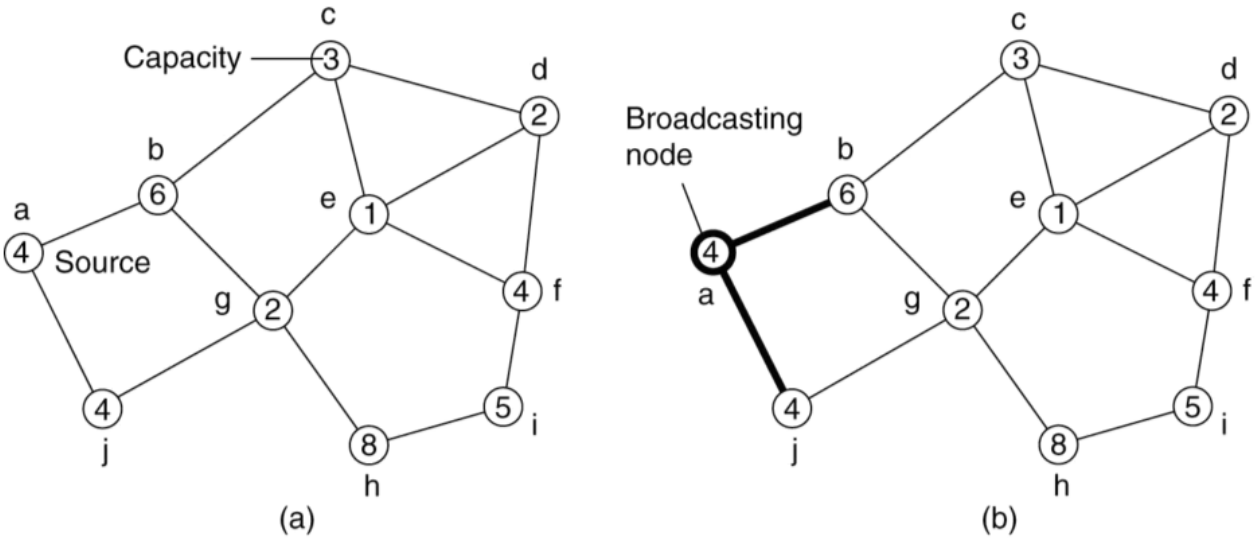
3. P 再将新确定的协调者 C' 传给所有进程



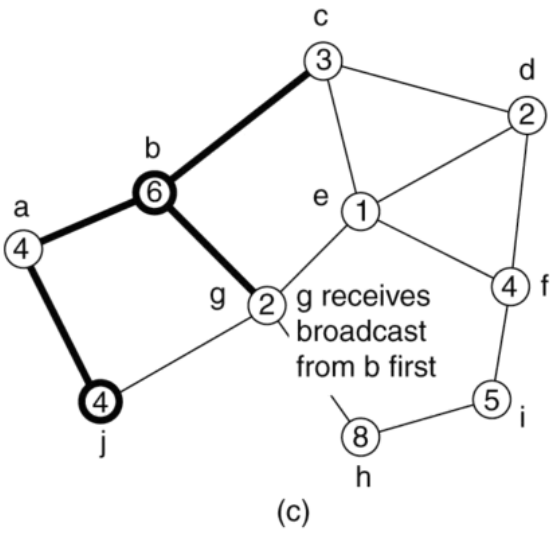
4.4 无线网络系统的选举算法

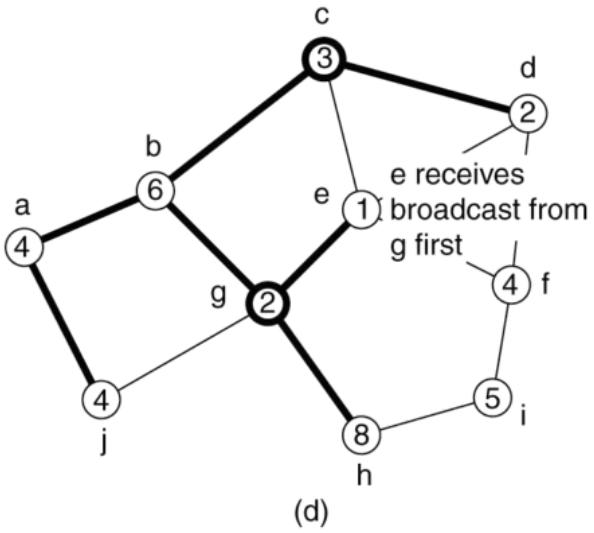
选举一个协调者，它具有最大的能力

1、发起者，提出选举



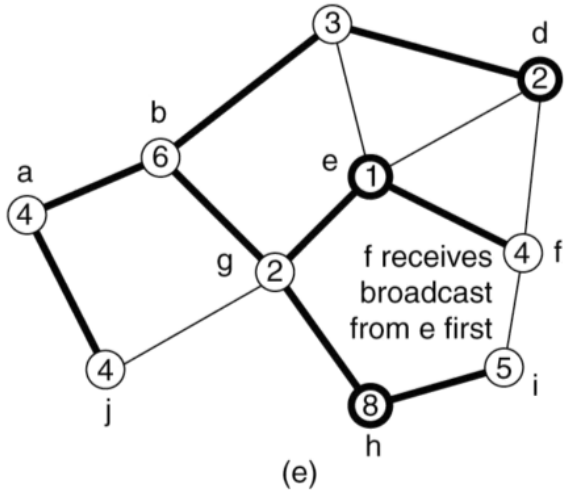
2、向邻居结点扩展，形成一个生成树（spanning tree）

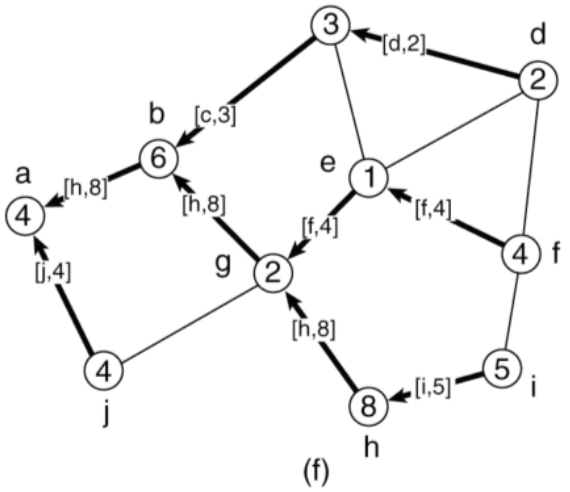




3、沿生成树向父节点返回 $[i, c_{max}]$， c_{max} 为最大值

4、发起者，向其余节点发布协调者





4.5 大型系统的选举

大型系统中需要选举多个节点

- 如 p2p 系统中的超级节点

对如何选择超级节点（superpeer）的要求:

- 普通节点对超级节点的访问延迟要小
- 超级节点应均匀地分布在覆盖网络中
- 相对于覆盖网络中的节点数量，应有一定数量的预先定义好的超级节点

- 每个超级节点服务的普通节点个数不能超过规定的数量

例：一个小型 chord 系统 m=8(长度), k=3(预留)

- P AND 11100000 作为超级节点的键值
- N 个节点中平均有 2^k 个超级节点

M 维空间中的超级节点选举

- 首先，在 N 个随机选择的节点中，放置 N 个令牌
- 每个节点不允许拥有一个以上的令牌
- 每个令牌具有排斥力，推动另一个令牌移动
- 通过互相排斥，最终达到在空间中的均匀分布