软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程基础知识

- 编程基础
- 计算机网络（网络编程）
- 密码学
- 网络与信息安全

参考书

- 《软件安全实现 — 安全编程技术》
  - 郭克华　清华大学出版社
- 《安全协议》
  - 卿斯汉 清华大学出版社
- 《编写安全的代码》(第二版)，Writing Secure Code(Second Edition)
  - 作者: Michael Howard, David LeBlanc著
  - 微软公司核心技术书库 机械工业出版社
- 《漏洞发掘的艺术》

课程设置目的

- 背景：
  - 网络安全问题是当前热点问题；
  - 可信软件、可信计算、软件确保领域研究；
- 本课程培养学生采用系统的设计方法，将安全性设计思想贯穿于系统设计、开发、测试过程。
- 使学生掌握最常见的、最新的软件安全技术，能够掌握恶意软件防范技术，能够掌握安全的信息系统设计与实现技术。
- 提高学生的专业素质，拓展知识面，强化信息安全意识。

网络安全离不开 最终危害的是主机，源头也是主机系统，离不开应用软件，利用就是软件开发的缺陷，存在的漏洞实施攻击。

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

课程说明与考核

- 课程说明
  - 总学时：42
  - 讲课学时：30　（1-9周）
- 考核安排
  - Exam：60%-70％
  - 平时：30%-40%
    - 课堂
    - 上机实验课
- 要求

Proverbs

- Nothing great was ever achieved without enthusiasm
- Self-trust is the first secret of success
- Practice makes perfect (Practice doesn't make perfect, only perfect practice makes perfect.)

第一章

软件安全需求

对安全系统的需求

- 1.1 Internet是一个充满敌意的环境
- 1.2 安全性是可信计算的需要
- 1.3 安全误区
- 1.4 攻击者的优势和防御者的劣势
- 1.5 安全事件分类

Internet是一个充满敌意的环境

- 当前互联网络的特点
  - 开放性（协议的开放性、资源共享需要）
  - 应用程序的复杂度提高
    - Windows 3.1 ——300万行代码
    - Windows 2000 ——5000万行代码
  - 漏洞发布时间与攻击时间发生时间缩短

- 需要保证互联网络环境下安全
    - 自主计算机的安全
    - 互联的安全
    - 各种网络应用和服务的安全

Internet是一个充满敌意的环境

- 维护信息载体的安全
  - 网络和系统的安全威胁包括
    - 物理侵犯(如机房侵入、设备偷窃、废物搜寻、电子干扰等)、系统漏洞(如旁路控制、程序缺陷等)、网络入侵(如窃听、截获、堵塞等)、恶意软件(如病毒、蠕虫、特洛伊木马、僵尸网络、信息炸弹等)、存储损坏(如老化、破损等)等。
- 维护信息自身的安全
  - 抵抗对信息的安全威胁
    - 这些安全威胁包括身份假冒、非法访问、信息泄露、数据受损、事后否认等。

Internet是一个充满敌意的环境

- 网络安全当前形势

- 了解最新事件报告、恶意代码、木马、僵尸网络、网页篡改情况
- http://www.cert.org.cn/ **国家计算机网络应急技术处理协调中心（简称CNCERT/CC）**
- http://www.ccert.edu.cn **中国教育和科研计算机网紧急响应组（CCERT）**

- 数据来源：国家互联网应急处理中心《2022年上半年中国互联网网络安全报告年年报》

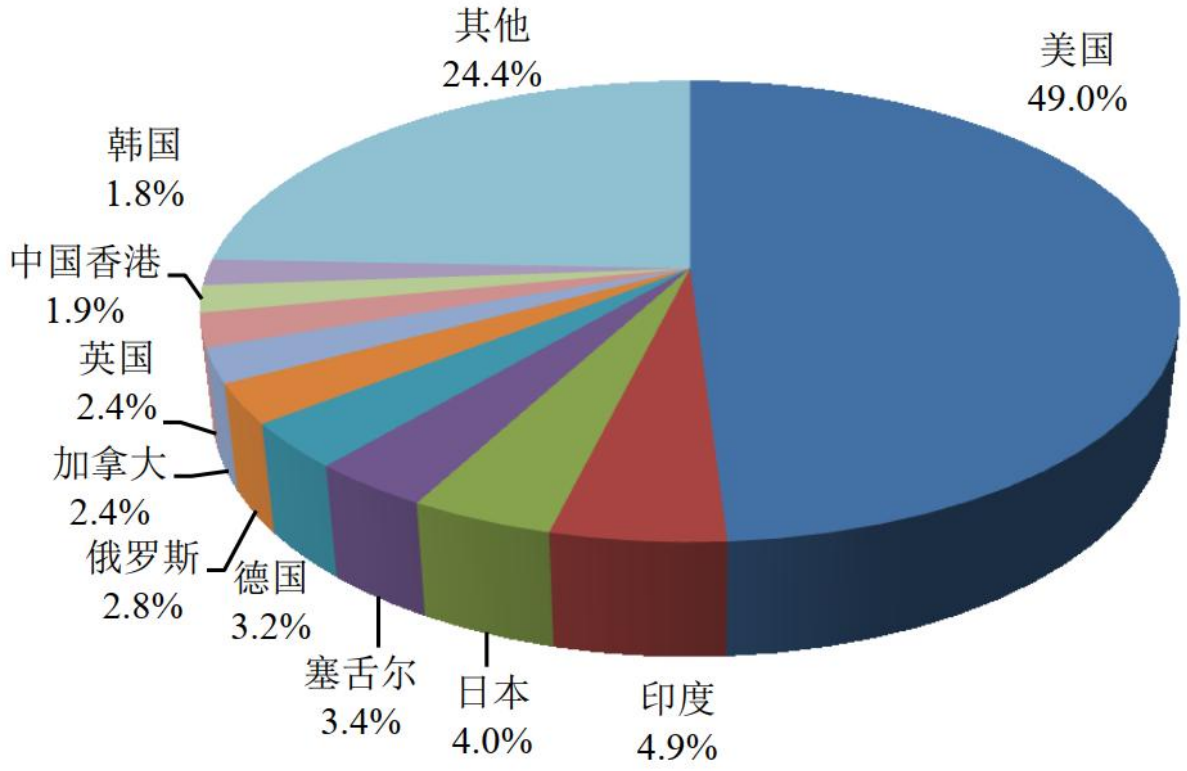


图 1 恶意程序传播源位于境外分布情况

- 我国境内感染计算机恶意程序的主机数量约 446 万台，同比增长 46.8%。位于境外的约 4.9 万个计算机恶意程序控制服务器控制我国境内约 410 万台主机。

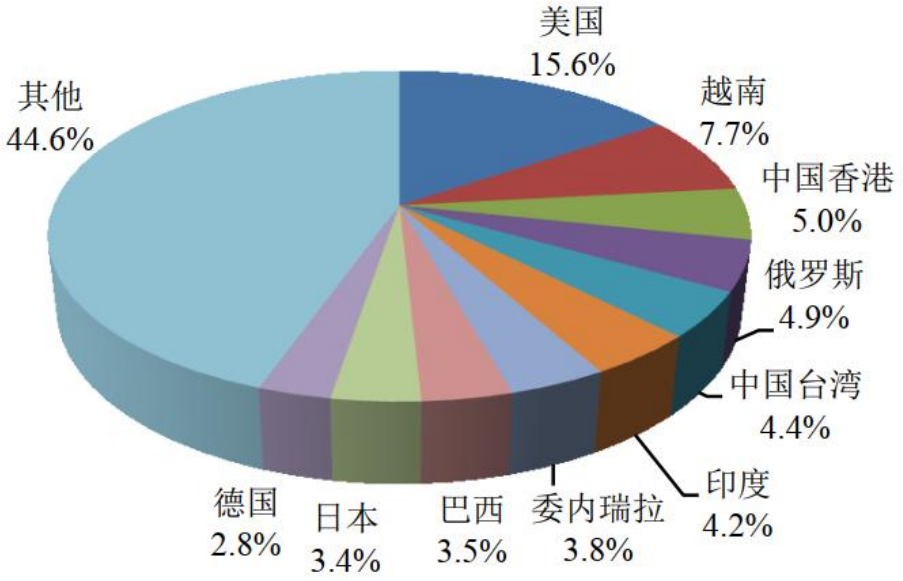


图 3 控制我国境内主机的境外计算机恶意程序控制服务器数量分布

- 我国境内感染计算机恶意程序的主机数量约 446 万台，同比增长 46.8%。位于境外的约 4.9 万个计算机恶意程序控制服务器控制我国境内约 410 万台主机。

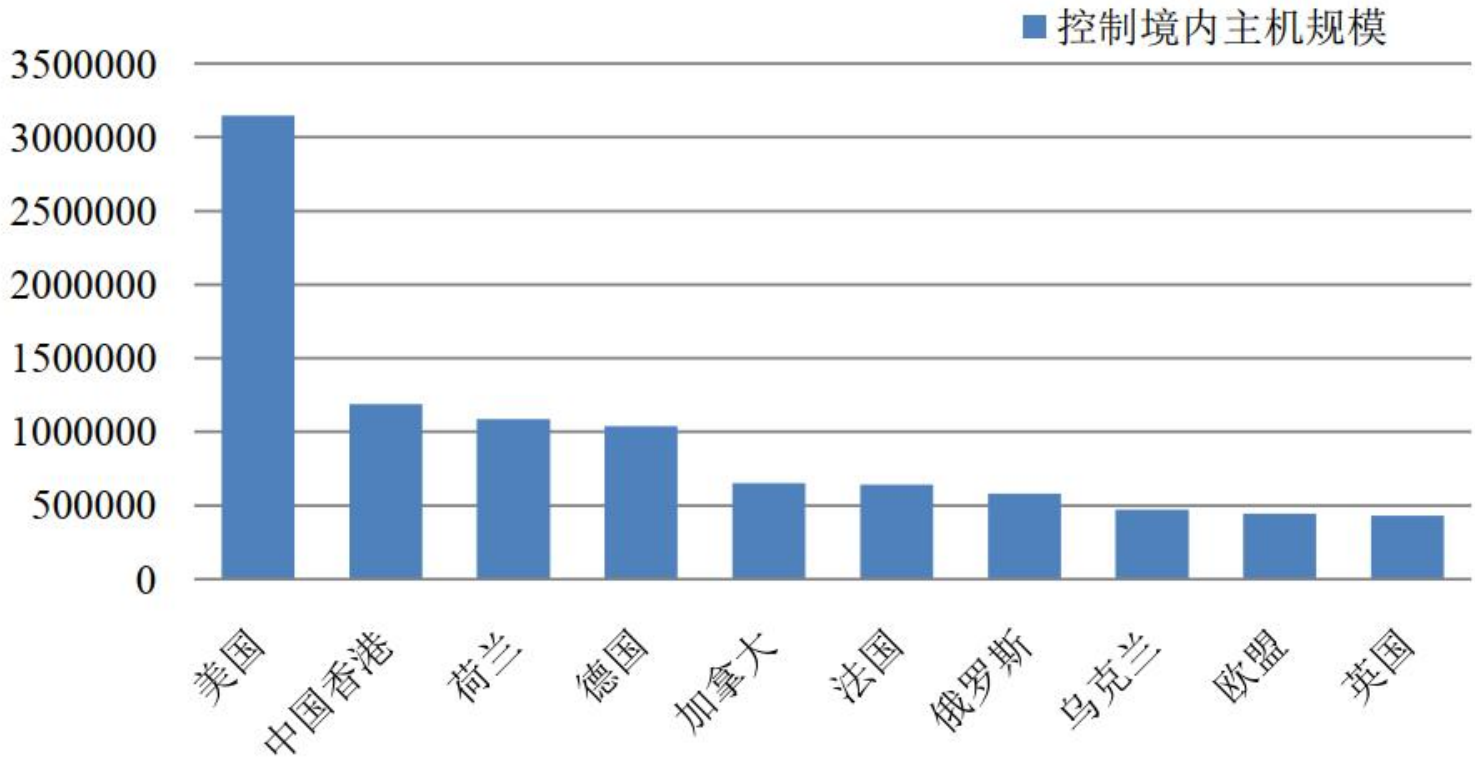


图 4 控制我国境内主机数量 TOP10 的国家或地区

- 境内外 8,289 个 IP 地址对我国境内约 1.4 万个网站植入后门。其中，有 7,867 个境外 IP 地址（占全部 IP 地址总数的 94.9%）对境内约 1.3 万个网站植入后门，位于美国的 IP 地址最多。

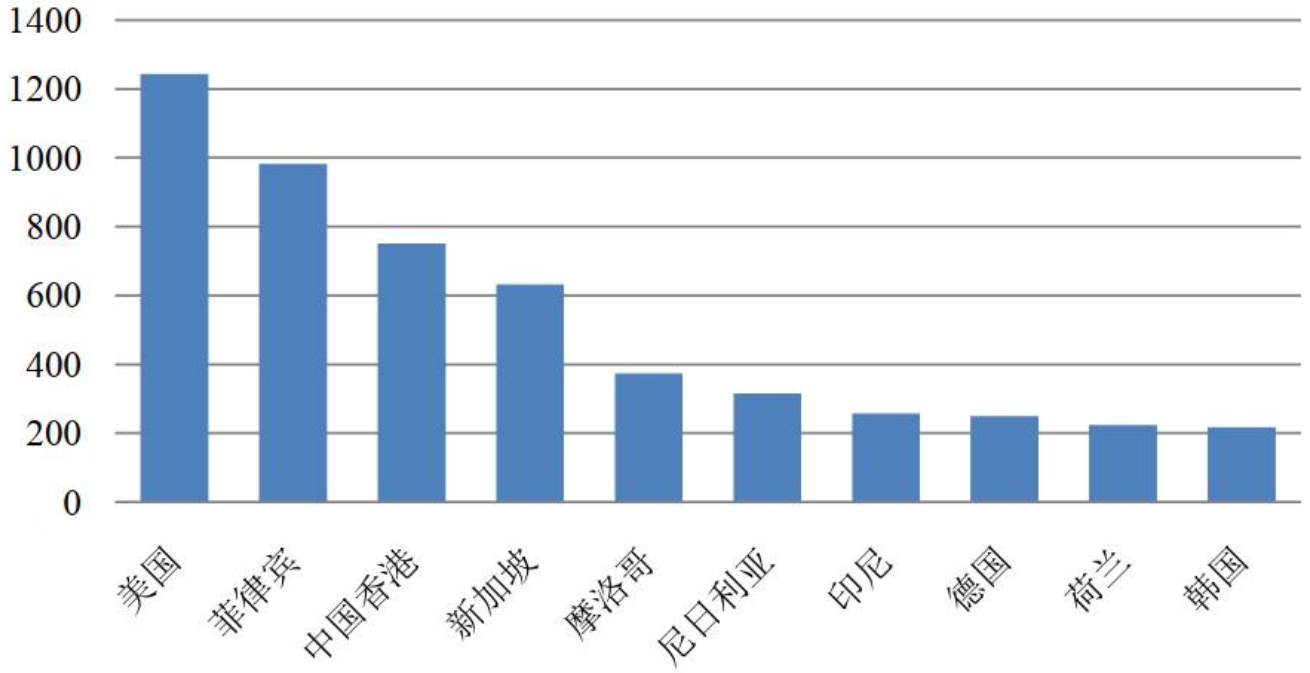


图 9 境外向我国境内网站植入后门 IP 地址所属国家或地区 TOP10

- 2022全年捕获恶意程序样本数量超过4,200万个，日均传播次数为482万余次，涉及恶意程序家族近34.8万个。

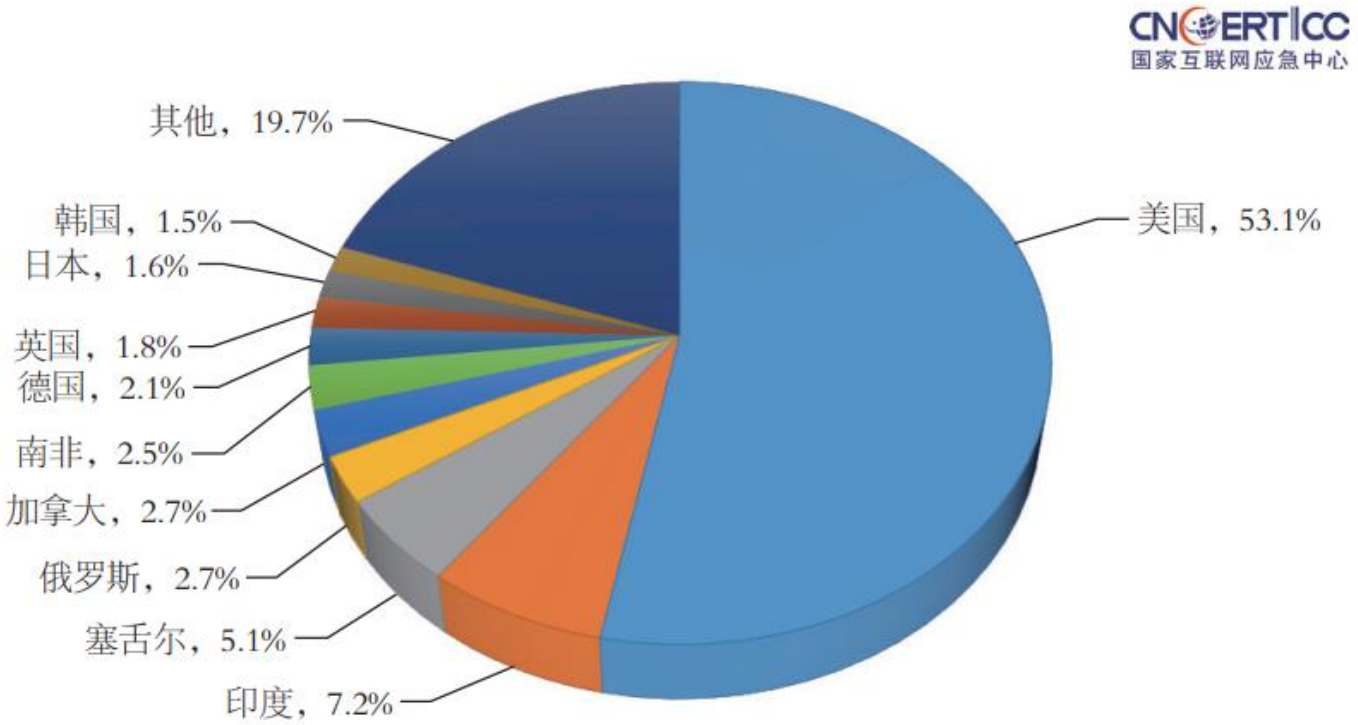


图 1-1　2020 年恶意程序境外传播源占比分布情况（来源：CNCERT/CC）

- 数据来源：国家互联网应急处理中心《2020年中国互联网网络安全报告年年报》，2020年，共发现国内基因数据通过网络出境717万余次，涉及我国境内近2.4万个IP地址，覆盖境内31个省（直辖市、自治区）。

图 2-24　2020 年我国生物基因数据出境次数按月度统计情况（来源：CNCERT/CC）

- 我国基因数据流向境外170个国家和地区，涉及境外IP地址近4.7万个，其中美国IP地址1.3万余个，约占27.7%。

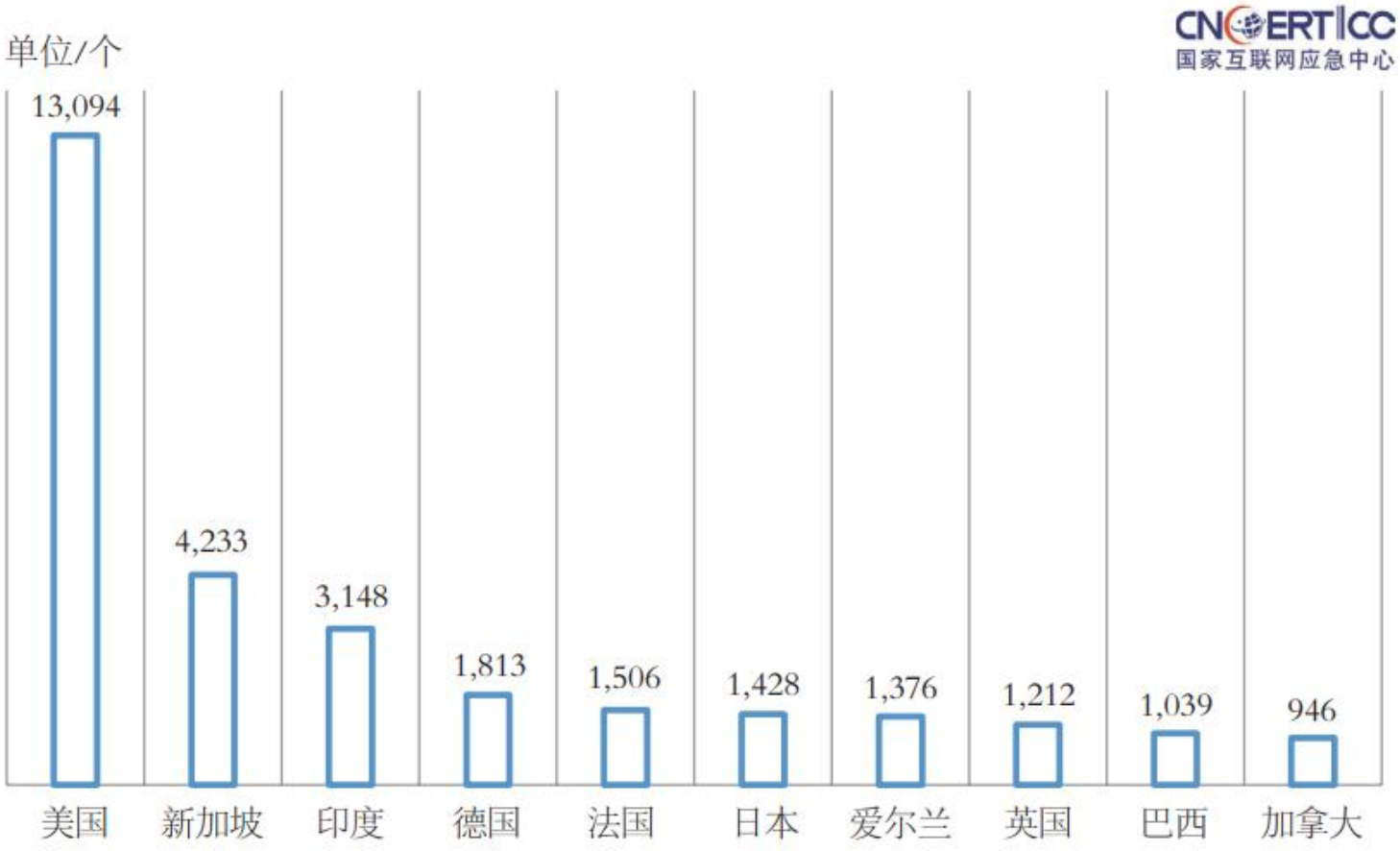


图 2-25　2020 年接收我国生物基因数据的 IP 地址数量排名 TOP10 的国家（来源：CNCERT/CC）

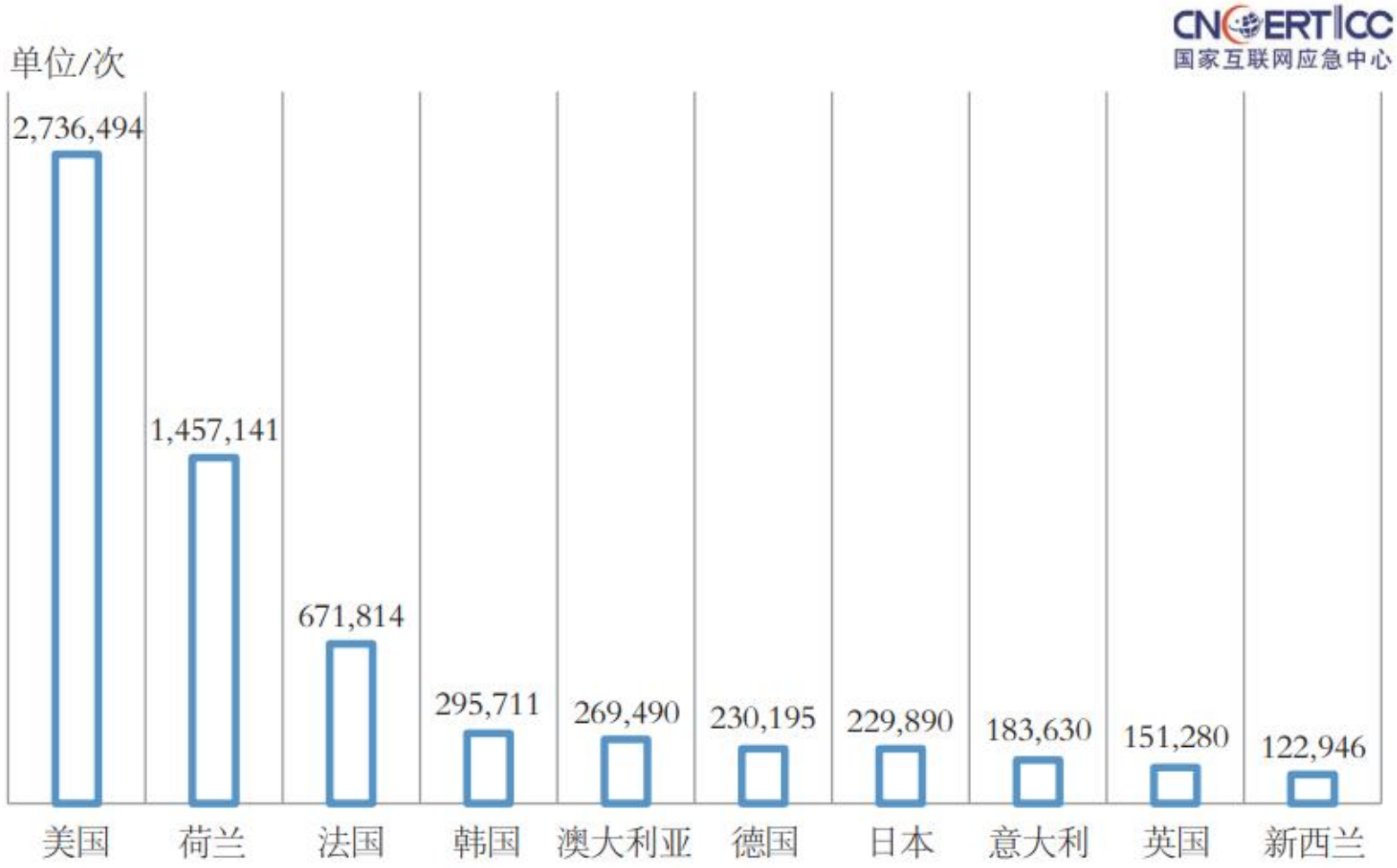


图 2-26　2020 年接收我国生物基因数据次数排名 TOP10 的国家（来源：CNCERT/CC）

- 2020年新冠肺炎病毒数据出境次数达99万余次，占生物数据出境总次数的13.8%

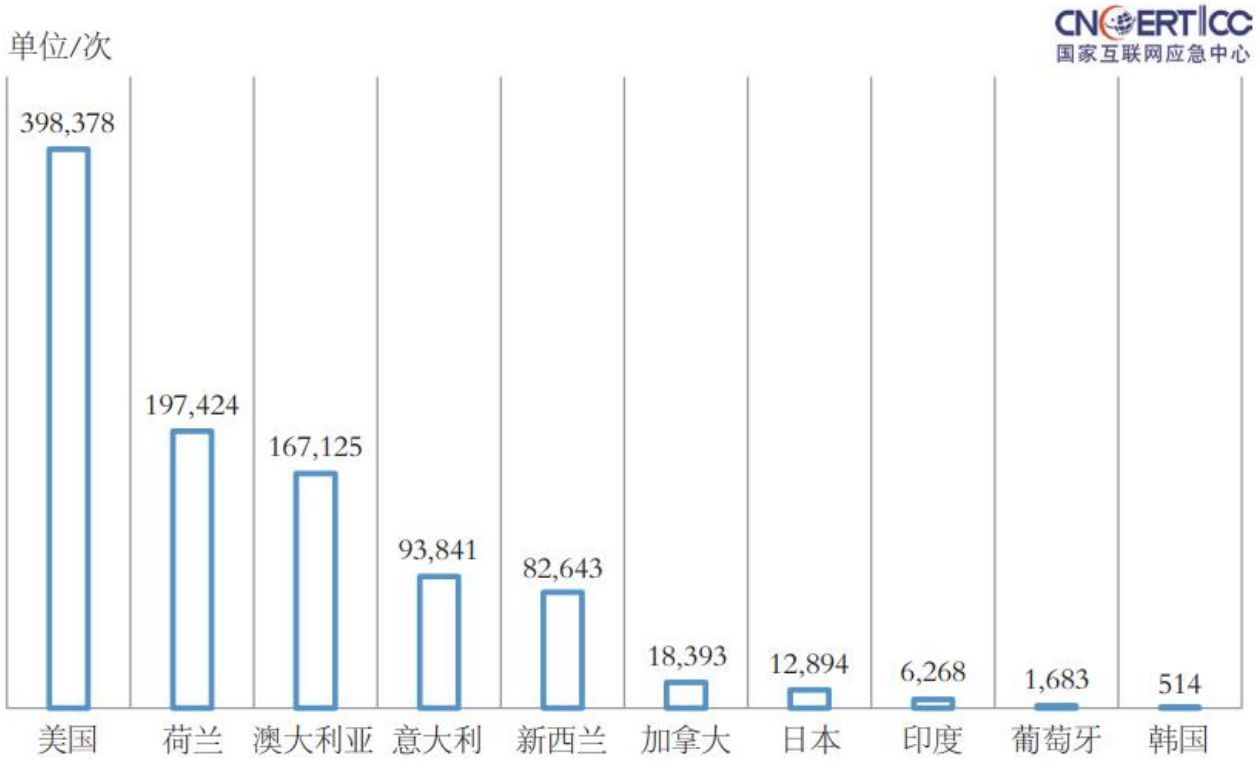


图 2-28　2020 年接收我国新冠肺炎病毒数据次数排名 TOP10 的国家（来源：CNCERT/CC）

- 2020年共发现境内医学影像数据通过网络出境497万余次，涉及境外128个国家和地区，IP地址近4.7万个，其中美国IP地址数量1.3万余个

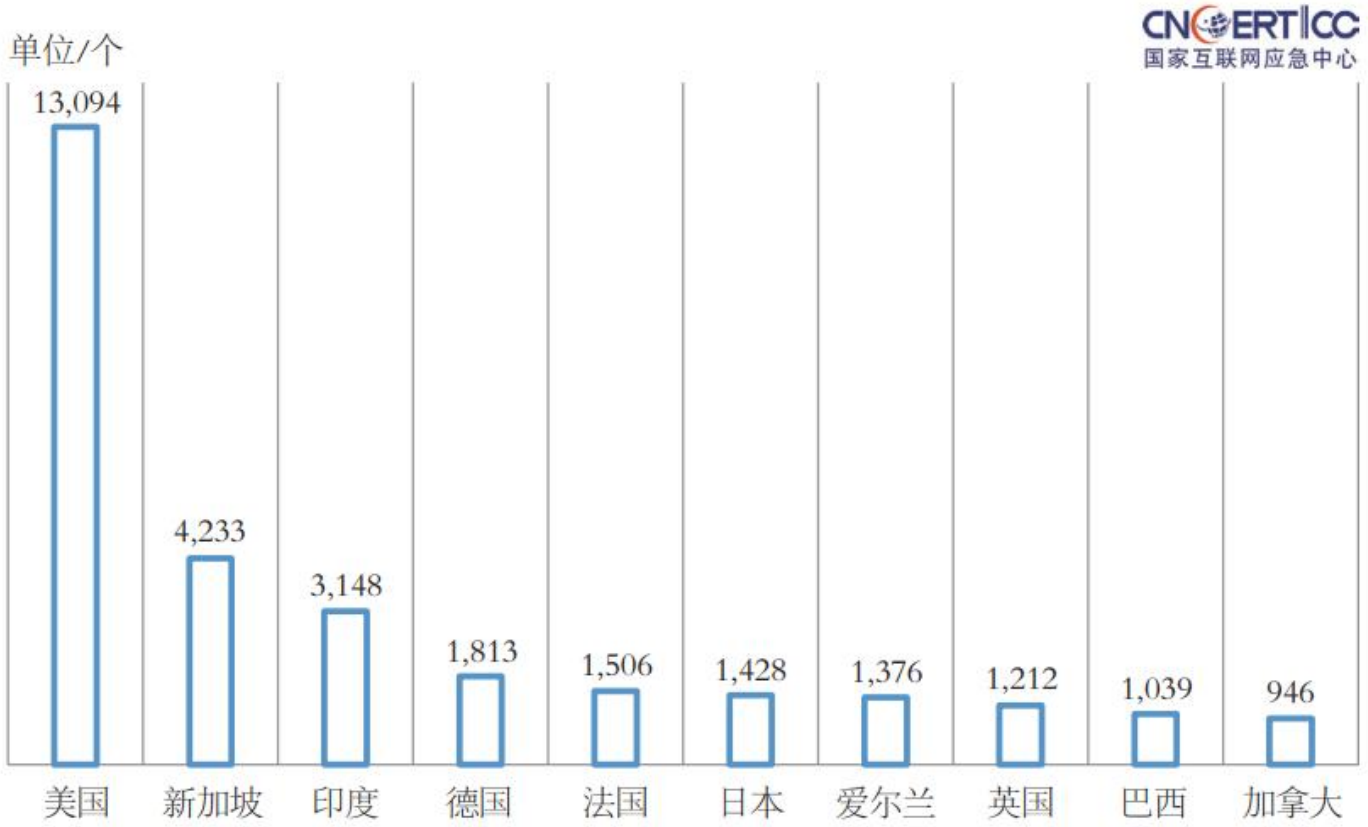


图 2-31　2020 年接收我国医学影像数据的 IP 地址数量排名 TOP10 的国家

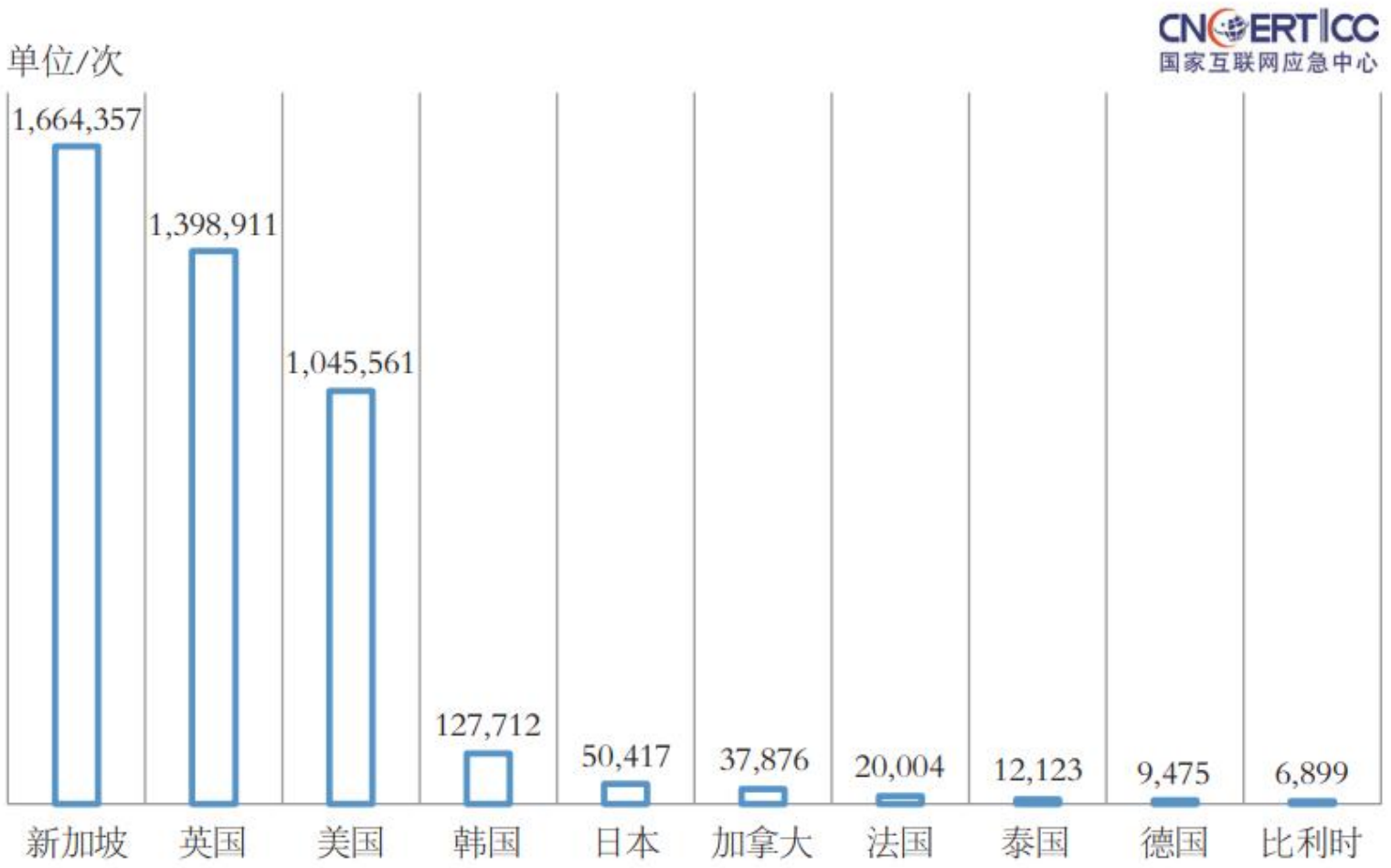


图 2-32　2020 年接收我国医学影像数据次数排名 TOP10 的国家（来源：CNCERT/CC）

- 2020年，英美两国生产的3款进口基因测序仪外联情况，共发现其通过76,237个境内IP地址与境外42个国家和地区的3,156个IP地址进行了跨境数据传输。

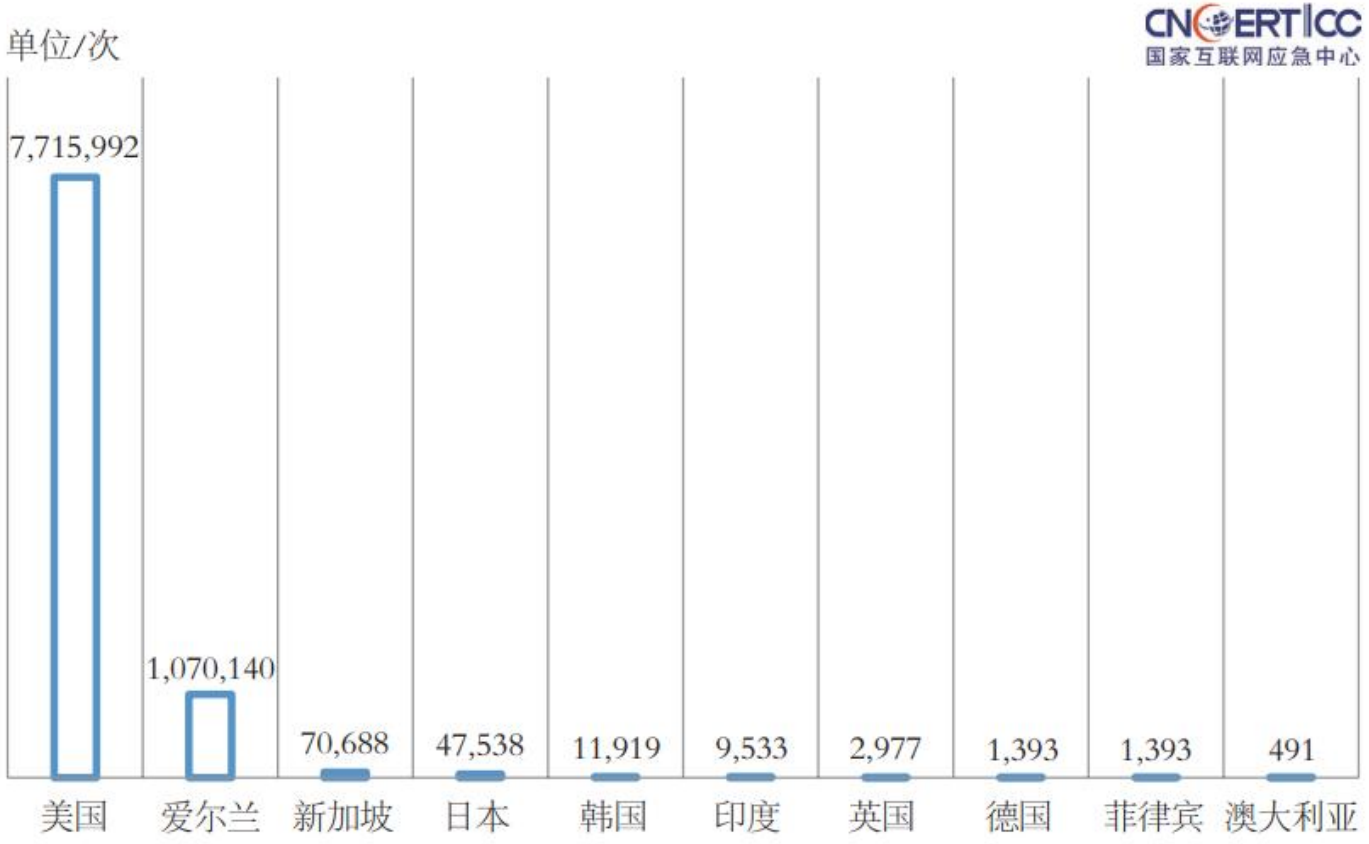


图 2-34　2020 年我国进口基因测序仪外联次数排名 TOP10 的国家（来源：CNCERT/CC）

Internet是一个充满敌意的环境

- 应用程序高度互联，安全问题越来越突出
- 应用程序在设计时并非为了在互联环境中运行，也会受到攻击
- 移动业务越来越普及，已成为恶意攻击的重点目标
- 网络主权问题的争议

- 要点：
  - 永远不要假设你的应用程序只在很少的一些指定环境下运行
  - 应假设你编写的程序将运行在最具敌意的环境中，然后依据这个条件进行设计、编写和测试

安全需要

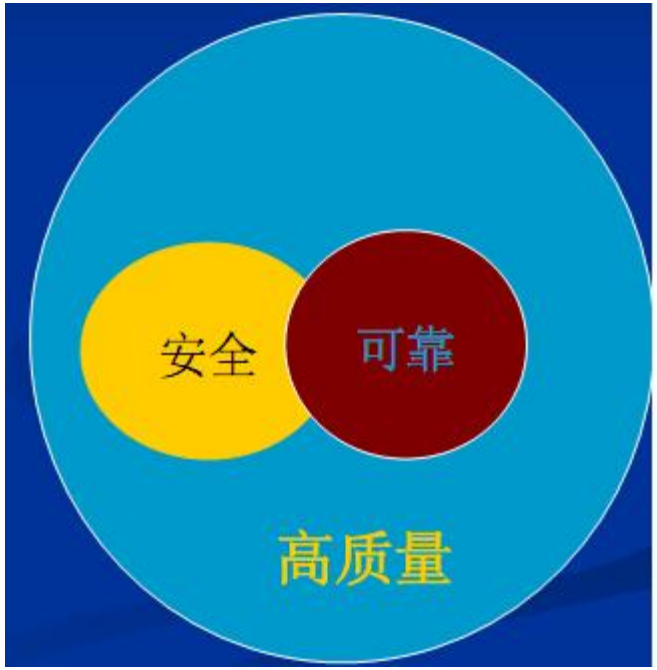
- 安全的系统是高质量的系统
- 设计系统时应将安全特性考虑进来
- 事先考虑安全特性比事后考虑安全性，软件健壮性好
- 修补安全漏洞的代价
  - 开发人员找出漏洞代码开销、修补代码开销
  - 创建补丁开销、在网站发布补丁的开销
  - 改善公众关系的开销
  - 用户应用补丁的开销
  - 用户取消产品应用将带来潜在商业损失

安全性是可信计算的需要

- 计算安全是要保证信息传递、描述和变换过程安全。
- 可信计算并不是保护用户的机器，做用户不想做的事情，而是保护用户机器内的数据拷贝。使它们不被你按照他人不希望的方式进行访问
- 如何利用信任度较低的系统构建可信的网络计算环境
- 如何在不可信的计算环境中，保证业务应用系统的可信运行和维护。
- 在可信计算基础研究的基础上，研究可信计算平台、可信接入、和可信应用。

安全性是可信计算的需要

- 安全的产品是高质量的产品
- 因为我们的计算机需一直运行，从而完成其要求的工作，不能因为接收一封恶意邮件而崩溃，也不能允许未经授权用户使用自己的系统



- 如何使我们的计算机可信？
- 如何使我们的系统能自我防护？

- 需要做大量工作
- 对大型网络、软件的防护是件很吸引人的事，也是个难题

- 本课程的目的是采用软件安全的专业知识，构建真正的能让我们信任的系统

- 安全需求和安全的程度是由环境驱动的
- 不同情况有不同解决方案
- 例：用来保护涉密数据的解决方案，不必是可靠的解决方案。如果系统崩溃了，不泄露秘密数据，仍然是安全的。
- 目标：防御和隔离，提高系统的生存性，生存能力

安全误区

- 有些开发人员、测试人员、设计人员逃避安全设计和安全测试

- 理由 1：没有人会做那样的事
  - 网络上有各种各样的各种系统下的攻击脚本可供下载，不要存侥幸心理

- 理由 2：为什么有人做那样的事
  - 为了破坏而破坏。网络中有无数的潜在攻击者

- 理由 3：我们还没有受到过攻击
  - 有人说：“ 没人关心你的产品”
  - 但过去不表示未来没人关心
  - 攻击者的目的是发现隐患，并通报这个隐患
  - 如：在去往学校的途中有危险的路，学校建议修一条人行横道，以便学生安全的过马路。建议被拒绝，因为没有人受伤。最后学生受伤了，才修路。
- 在安全设计过程中应事先预防，而不是事后修补，否则破坏已经造成

- 理由4：我们是安全的，因为使用了密码
  - 人们在密码方面常犯的错误：
    设计自己的密码算法
    不安全的密码储存

- 理由5：我们是安全的，因为使用ACL
  - 一个好的 ACL 能够保护资源不受攻击
  - 一个坏的ACL能够导致错误的安全保护，并受到攻击
    - 例如，一个人使用了ACL，但ACL 是Everyone 完全控制
  - ACL就足够了吗？

- 理由 6：我们是安全的，我们使用了防火墙
  - 防火墙能够控制端口，但不能控制允许端口传送的内容
  - 如：许多攻击是通过 http80端口进行

防火墙是好的工具，但仅靠防火墙是不能解决所有问题

- 理由 7：我们检查了代码，没有安全bug
  - 代码没有问题，就没有bug了？
  - 如：检查飞机的轮子、引擎、尾翼、油箱能保证飞机起飞，但不能验证飞机的安全性

检查代码的人必须了解攻击者是如何攻击软件的，安全软件如何构成？

- 理由8：默认情况下功能是启动状态，但管理员能关闭
- 管理员不会关闭
  - 他们不知道关闭什么
  - 他们不知道如何关闭
  - 他们不知道关闭后会有何影响
  - 他们已经有了非常坚固的系统，为什么要改动

- **Windows 2000 Server** 默认情况下服务处于打开状态
- **IIS 6**在默认情况下，将大多数特性关闭。如果你需要，你必须启动

- 如果我们不以管理员的身份运行，有些东西不能运行
  - 有些程序写的很糟糕，必须用管理员身份才能完成
  - 软件开发人员，应注意权限问题，应支持以最小特权运行， 即使必须用高级权限，也应该设置合适的权限，而不是更多的权限
- 我们需要更好的开发工具
  - 工具有助于开发，但糟糕的开发人员即使用最好的工具也会写出糟糕的代码
  - 好的开发工具只是一个辅助
  - 好的编码能力是没有替代物的(养成良好的习惯习惯)

攻击者的优势和开发者的劣势

- 软件安装后，就处于防御状态中，就面临潜在的全天候的攻击
- 代码必须经受住攻击，确保本系统保护的资源不会泄露、损害、被删除或被恶意察看
- 即使开发人员采用安全设计方法，也总是处于落后地位，总跟在后面处理已发现的安全问题
- 因为攻击者总处于有利条件，防御者必须构建更高质量的系统

- 被动防御与主动防御

- 因素1：防御者必须对所有环节都进行防御，而攻击者可以选择最薄弱的环节

- 如，城堡防御：城墙、护城河、弓箭手、吊桥、弓箭手装备充足，防止火灾，保证储备充足对付长时间围攻等而攻击者只要找到一个防御不完备点就可以

- 软件攻击，只要找到一个薄弱环节就可以
- 软件防御者，必须确保每一个进入代码的入口点都得到保护

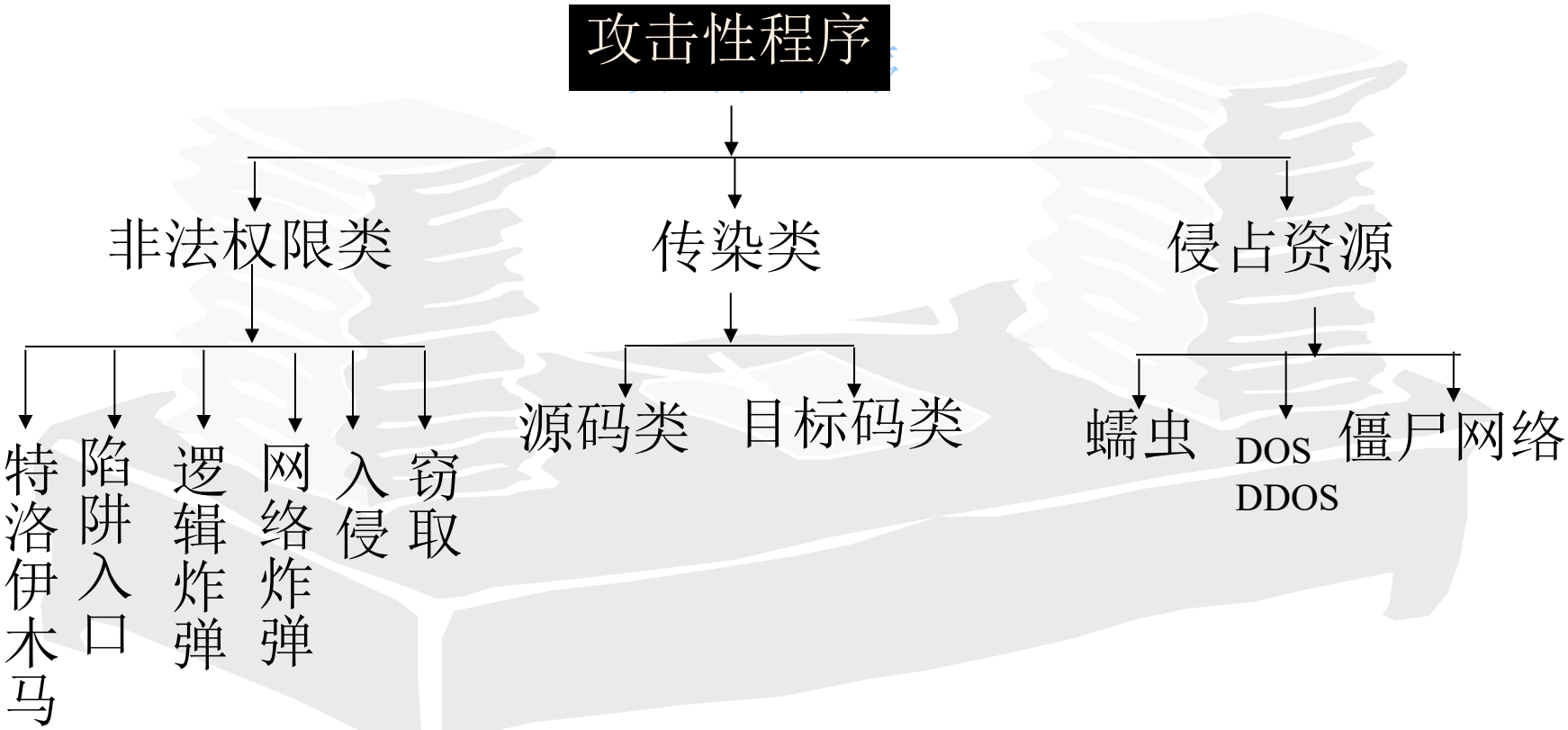
- 因素2：防御者只能针对已知的攻击进行防御，而攻击者则可以探测未知漏洞
  - 如：特洛伊木马，特洛伊人没意识到希腊人的礼物是攻击点
  - 如：IIS 5正确处理了URL中包含转义字符的攻击，但没有准备防御手段来处理利用畸形UTF-8的攻击（参考 http://www.wiretrip.net/rfp/p/doc.asp/i2/d57.htm）

- 对未知攻击进行防御的唯一途径是，如果用户不明确用哪些功能，就禁用这些功能

- 因素3：防御者必须永远保持警惕，攻击者随时可以罢工
  - 管理员必须始终监视系统，查看安全日志，并查找和抵御攻击
  - 软件开发人员必须提供能持续抵御攻击的软件，以及监视工具进行判断

- 因素4：防御者活动必须遵循相应的规则，攻击者可以采用卑鄙的手段
  - 防御者可以使用白帽子工具，如防火墙、入侵检测系统，审计日志和蜜罐

攻击性计算机程序的分类



小结

- 作为防御者，开发人员必须构建始终保持警惕的应用程序和解决方案
- 应充分提高安全门槛，让攻击者发现攻破软件非常困难
- Internet是非常复杂充满敌意的环境，应用程序必须能在此环境下经受住考验
- 任何软件都是不安全的

我们的目标

- 如何设计开发安全的软件
- 如何检测识别恶意软件

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

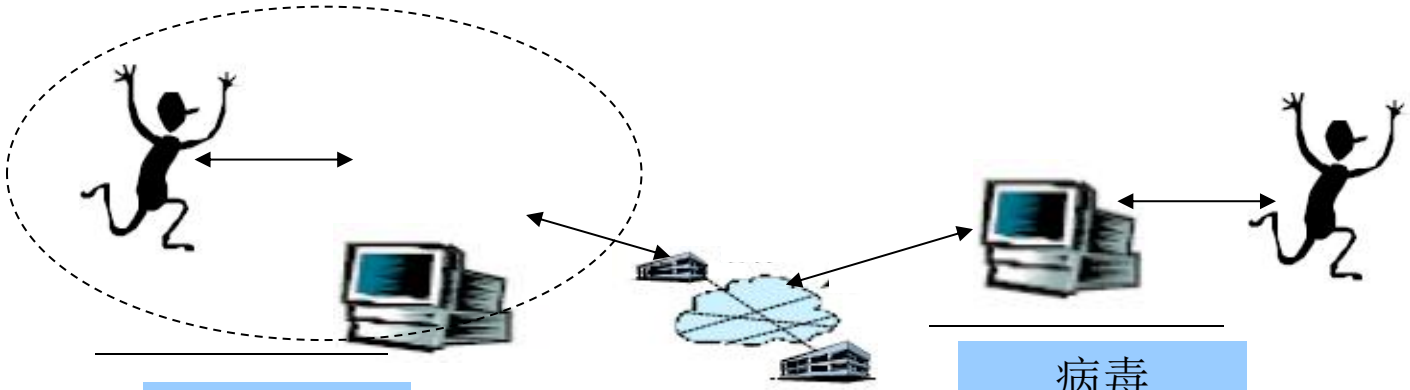
1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件系统面临的威胁

主要内容

- 概述
- 计算机系统缺陷
  - 漏洞利用
  - 恶意代码攻击
- 网络与协议缺陷
  - TCP/IP协议缺陷
  - 网络攻击
- 攻击的分类

软件系统面临的安全威胁



身份冒充
抵赖
权限提升

搭线窃听
DOS攻击
伪造
重放
中断

病毒
后门
木马
缓冲区溢出
信息窃取
篡改
中断

威胁分类	举例
1.人为的错误或失效	事故，员工过失
2.知识产权侵害	盗版，侵犯版权
3.故意的探测或入侵	未经许可的进入和（或）数据收集
4.故意的信息敲诈	对信息泄漏的敲诈
5.故意的攻击或破坏	破坏信息或系统
6.故意偷窃	非法占用设备或信息
7.故意的软件攻击	病毒，蠕虫，宏病毒，拒绝服务攻击
8.自然力	火灾，洪水，地震，闪电
9.背离ISP的服务质量	WAN服务问题
10.技术性硬件失效或错误	设备失效
11.技术性软件失效或错误	程序Bug，编码问题，未知漏洞
12.技术的退化	废弃或过时的技术

图 1-8　2016-2020 年 CNVD 收录的安全漏洞数量对比

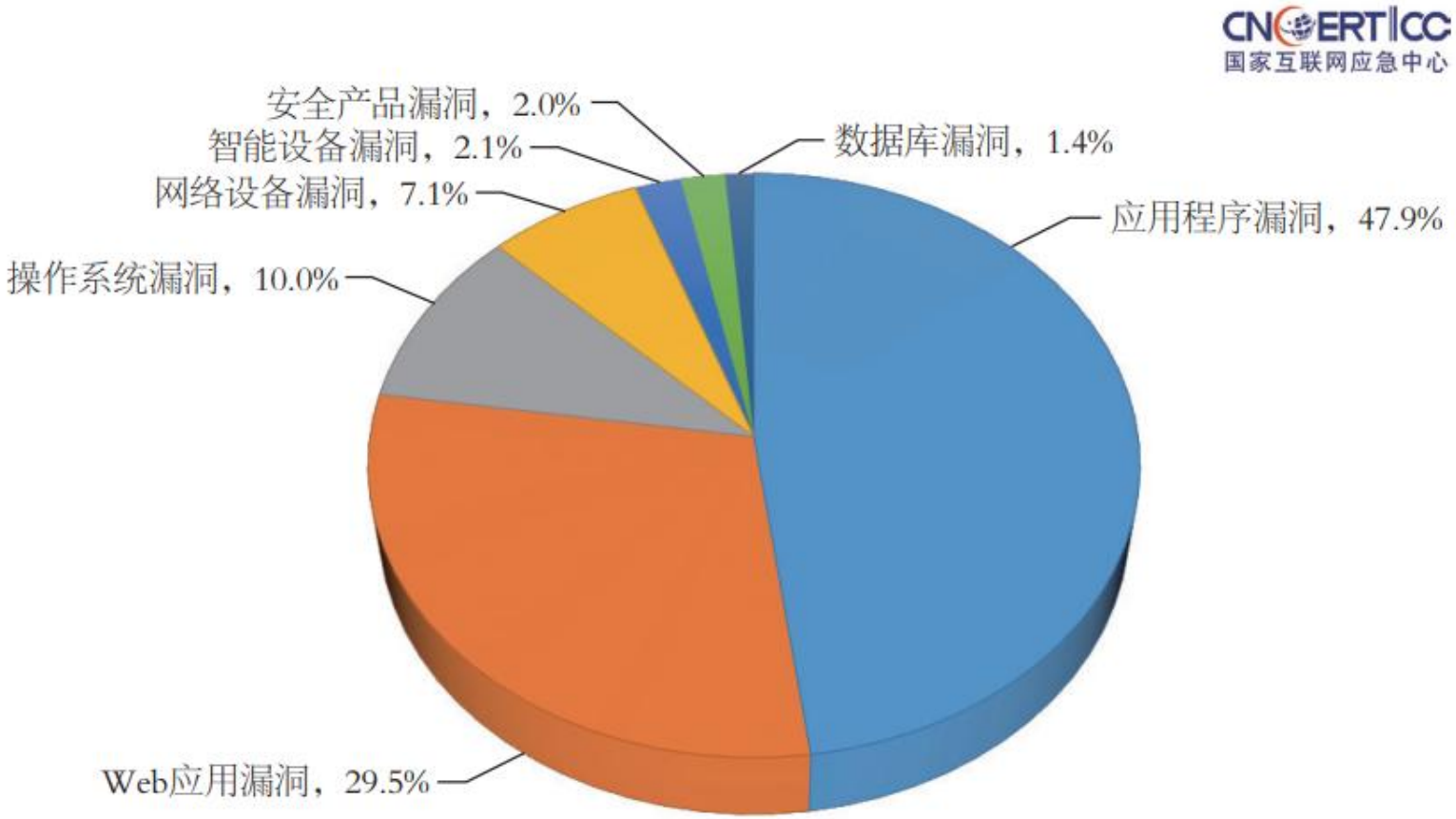


图 1-9　2020 年 CNVD 收录的安全漏洞数量占比按影响对象分类统计（来源：CNCERT/CC）

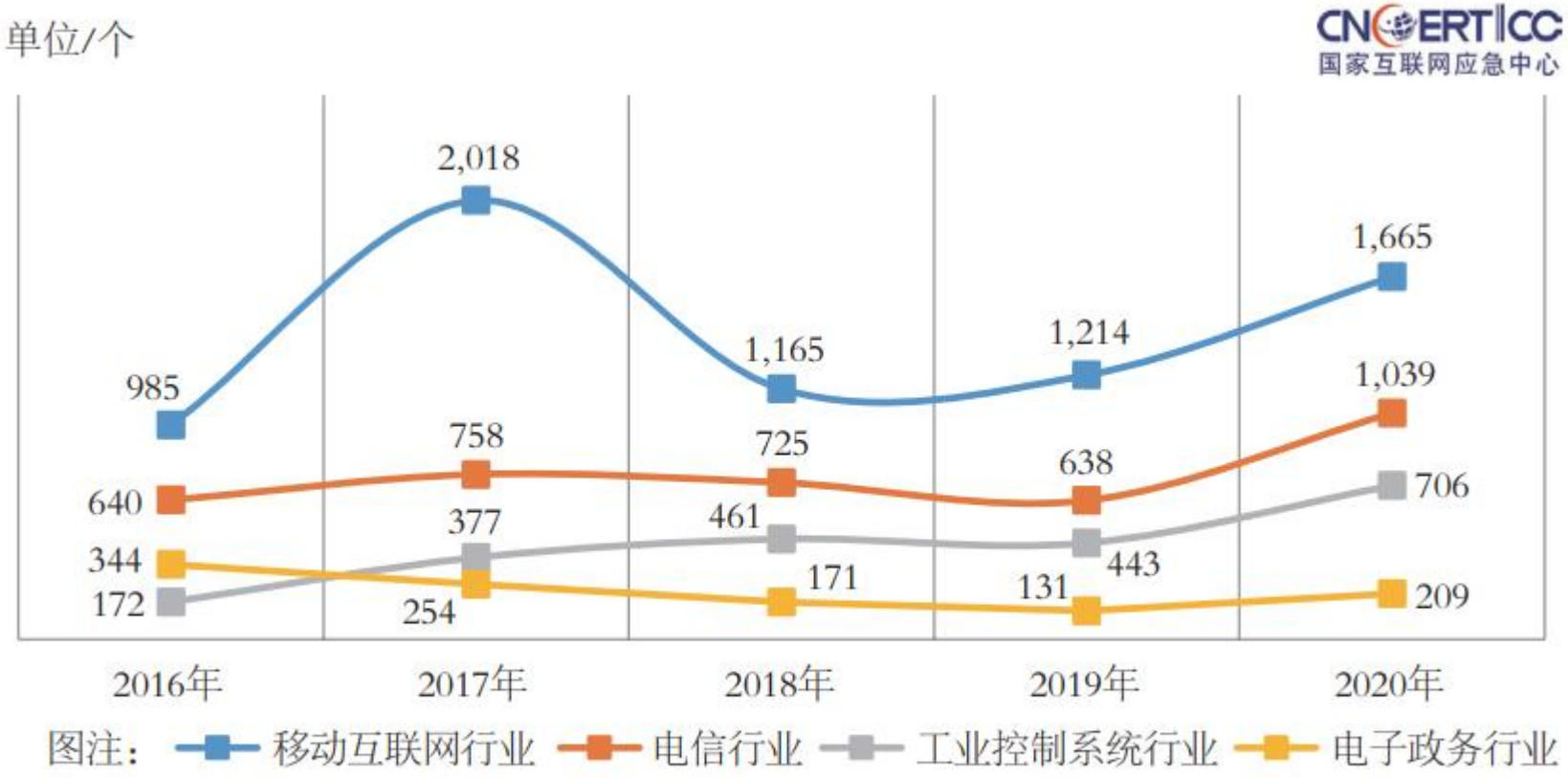


图 1-10　2016-2020 年 CNVD 子漏洞库收录情况对比（来源：CNCERT/CC）

软件系统面临的安全威胁

攻击来源可分为三大类：合法用户、外部攻击者、物理环境

--内部人员、厂商、顾问

--自然灾害、断电、割断线缆

--外部攻击者，国外间谍、黑客

为什么会有攻击？

主要内容

- 概述
- 计算机系统缺陷
  - 漏洞利用
  - 恶意代码攻击
- 网络与协议缺陷
  - TCP/IP协议缺陷
  - 网络攻击
- 攻击的分类

安全漏洞分析

- 系统安全漏洞，是计算机系统在硬件、软件、协议的设计与实现过程中或系统安全策略上存在的缺陷和不足。非法用户可利用漏洞获得计算机系统的额外权限，在未经授权的情况下访问或提高其访问权限，从而破坏系统的安全性。

- 当然漏洞的存在本身并不能对系统安全造成什么损害，关键的问题在于攻击者可以利用这些漏洞引发安全事件。

漏洞的危害

- 不用说了！

漏洞的产生原因

(1)输入验证错误

漏洞的产生是由于未对用户提供的输入数据的合法性做适当的检查。这种错误导致的安全问题最多。

(2)访问验证错误

漏洞的产生是由于程序的访问验证部分存在某些可利用的逻辑错误或用于验证的条件不足以确定用户的身份而造成的。这类缺陷使得非法用户绕过访问控制成为可能，从而导致未经授权的访问。

(3)竞争条件错误

漏洞的产生是由于程序在处理文件等实体时在时序和同步方面存在问题，在处理的过程中可能存在一个机会窗口使攻击者能够施以外来的影响。

漏洞的产生原因

(4)意外情况处置错误

漏洞的产生是由于程序在它的实现逻辑中没有考虑到一些本应该考虑的意外情况。如没有检查文件是否存在就直接打开文件导致拒绝服务、上下文攻击导致执行任意代码等。

(5)配置错误

漏洞的产生是由于系统和应用的配置有误，或是软件安装在错误的地方，或是参数配置错误，或是访问权限配置错误等。

(6)环境错误

由于一些环境变量的错误或恶意设置造成的漏洞，导致有问题的特权程序可能去执行攻击代码。

缓冲区溢出(Buffer Overflow)是目前最普遍的攻击手段, 最普遍的3种系统——Red Hat Linux、Solaris、Windows Server，在默认安装的情况下都有远程溢出漏洞
缓冲区溢出的攻击之所以危险是因为它可以在没有任何系统账号的情况下获得系统的最高控制权，此外它还很难被检测出来

安全漏洞

缓冲区溢出攻击

缓冲区溢出

- 1998年Lincoln实验室用来评估入侵检测的5种远程攻击手段中，有2种是缓冲区溢出。
- 在调查中，有2/3的被调查者认为缓冲区溢出漏洞是一个很严重的安全问题。
- 有人曾经对国内大约100台安装了三种系统的主机进行检测，发现其中70％以上的主机可以利用远程溢出直接获得root权限，这其中不乏某些著名的电子商务网站。
- 通过缓冲区溢出进行的攻击占所有系统攻击总数的80％以上。
- 第一个利用缓冲区溢出攻击的病毒是十多年前的Morris蠕虫，它曾造成了全世界6000多台网络服务器瘫痪。

缓冲区溢出攻击

几十年来，缓冲区溢出一直引起许多严重的安全问题：

1. 1988 年， Morris事件。
2. 2001年夏，红色代码蠕虫通过微软的IIS（Internet Information Server）的索引服务动态库的缓冲区溢出漏洞在Internet上广为传播
3. 2001年7月19日的午夜，尼姆达病毒开始发作，一日之后，被感染的主机数达341015台
4. 2001年12月，在微软windowsXP系统中发现通用的即插即用服务中有几个安全BUG,其中一个是栈缓冲区溢出漏洞，它能够使远程攻击者对于任何缺省安装的windowsXP取得管理员权限

缓冲区溢出攻击

- C语言假定程序员负责数据的完整性。如果将这种责任移交给编译器，由于对每个变量都要检查其完整性，最后所得到的二进制速度将会异常慢。并且，这会使程序员失去一个重要的控制层，并且使语言复杂化。
- 但是，如果程序员不小心，这种简单性会导致程序缓冲区溢出和存储器泄漏这样的漏洞。
- 当为某个变量分配了存储空间，但没有内置的安全机制来确保这个变量的容量能适应已分配的存储空间，这种操作很可能导致程序崩溃。这称为缓冲区超限(buffer overrun)或缓冲区溢出
- 例如：程序员把10个字节的数据存入只分配了8个字节空间的缓冲区

缓冲区溢出

- 缓冲区溢出原理：
  - 简单地说，缓冲区溢出就是向一个有限空间的缓冲区拷贝了过长的字符串，覆盖相邻的存储单元，这将会引起程序运行失败。
  - 因为变量保存在堆栈当中，当发生缓冲区溢出的时候，存储在堆栈中的函数返回地址也会被覆盖，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其它指令，以达到攻击的目的。

缓冲区溢出

- 造成缓冲区溢出的原因
  - 程序中没有仔细检查用户输入的参数。所以说缓冲区溢出的缺陷属于输入确认错误。

缓冲区溢出

- 为了理解缓冲区溢出的机制，我们先看一个例子：

```
#include <string.h>
void function(char *str)
{
   char buffer[16];

   strcpy(buffer,str);
}
```

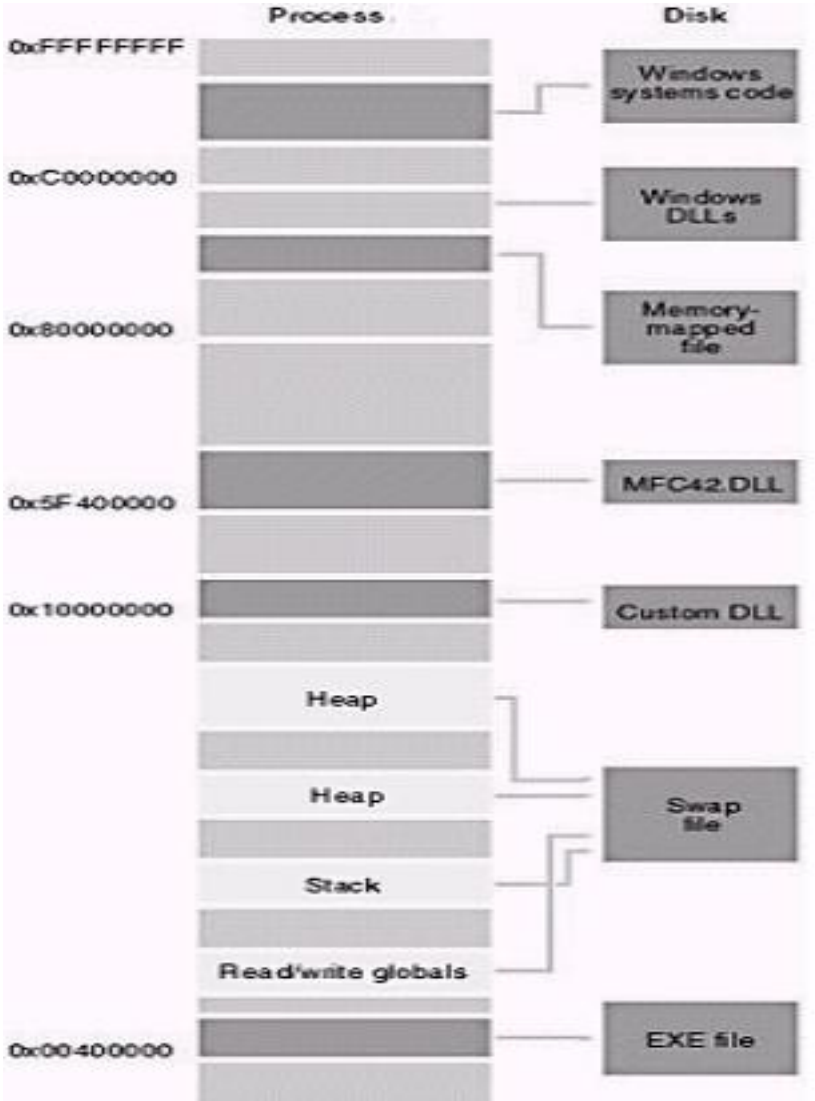
```
void main()
{
   char large_string[256];
   int i;

   for(i=1;i<255;i++)
         large_string[i]='A';
   function(large_string);
}
```

缓冲区溢出

- 编译并运行程序的结果是：
- “0x41414141”指令引用的“0x41414141”内存。该内存不能为“read”。
- 为什么呢？这跟内存存储数据的原理有关。

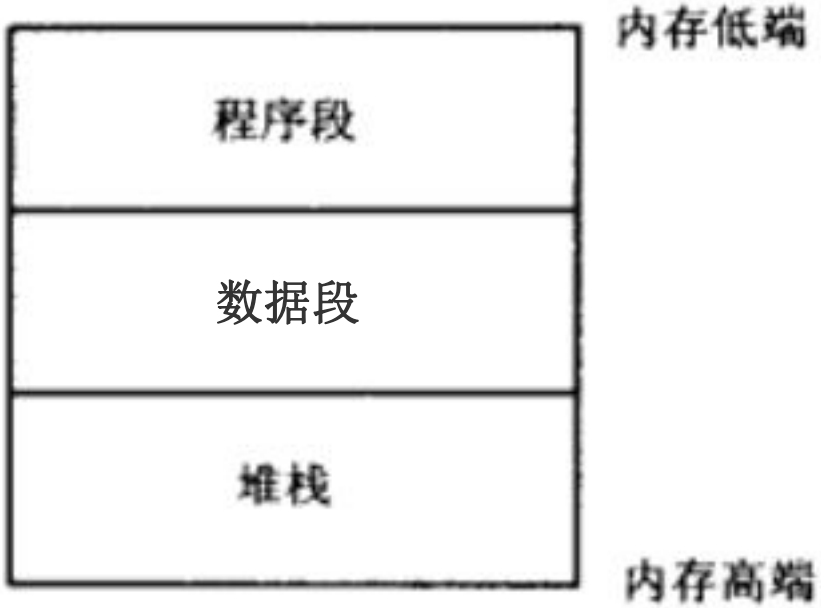
Win32进程内存空间

- 系统核心内存区间
  - 0xFFFFFFFF~0x80000000 (4G~2G)
  - 为Win32操作系统保留
- 用户内存区间
  - 0x00000000~0x80000000 (2G~0G)
  - 堆: 动态分配变量(malloc), 向高地址增长
  - 静态内存区间: 全局变量、静态变量
  - 代码区间: 从0x00400000开始
  - 栈: 向低地址增长
    - 单线程进程： (栈底地址: 0x0012FFXXXX)
  - 多线程进程拥有多个堆/栈
  - Example: ./win32/background/memory.c



缓冲区溢出

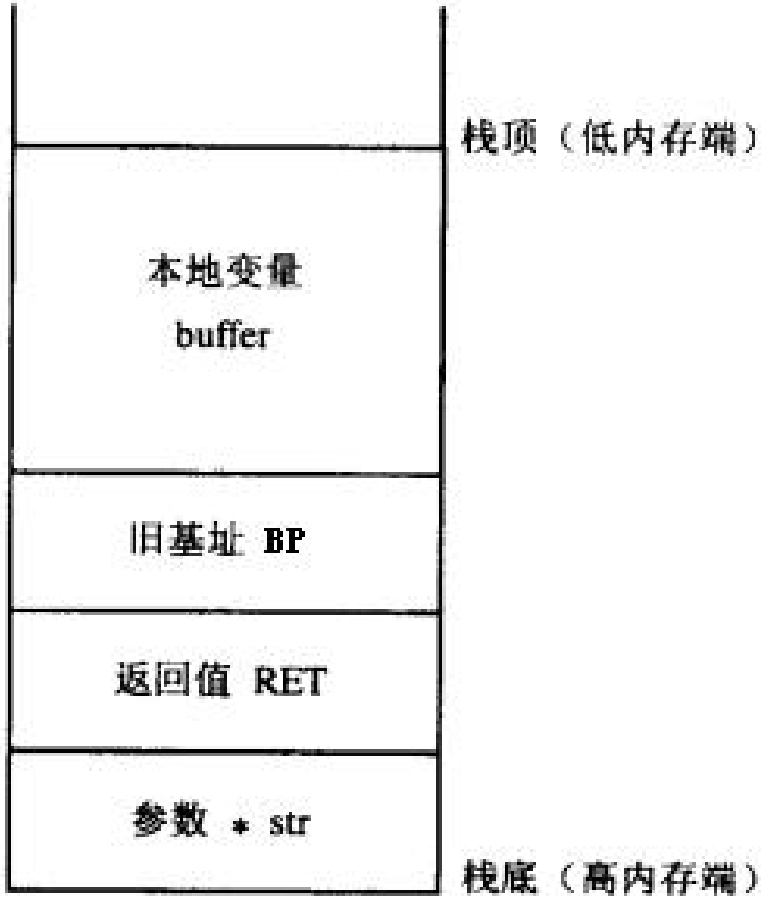
- 一个程序在内存中通常分为程序段，数据段和堆栈段3部分。
  - 程序段里放着程序的机器码和只读数据。
  - 数据段放的是程序中的静态数据。
  - 动态数据则通过堆栈来存放。



一个程序在内存中的存放

缓冲区溢出

- 当程序中发生函数调用时，计算机做如下操作：
  - 首先把参数压入堆栈；
  - 然后保存指令寄存器(IP)中的内容作为返回地址(RET)；
  - 第三个放入堆栈的是基址寄存器(BP)：然后把当前的栈指针(SP)拷贝到BP，作为新的基地址；
  - 最后为本地变量留出一定空间，把SP减去适当的数值。



调用一个函数后的堆栈

缓冲区溢出

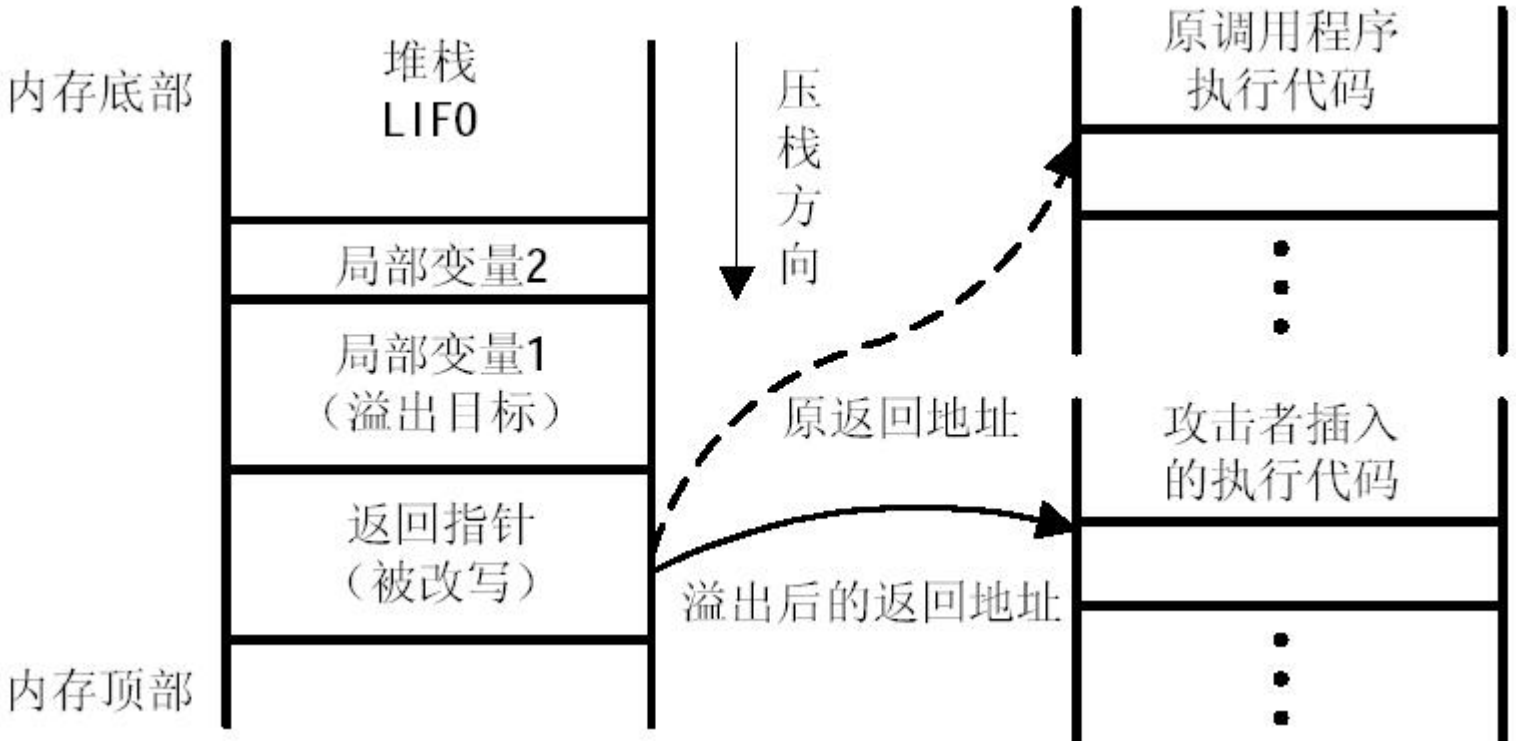
- 很显然，程序执行的结果是“Segmentation fault(core dumped)”或类似的出错信息。
  - 因为从buffer开始的256个字节都将被*str的内容‘A’覆盖，包括BP，RET，甚至*str。‘A’的十六进值为0x41，所以函数的返回地址变成了0x41414141，这超出了程序的地址空间，所以出现上面的错误。

缓冲区溢出

- 是否能够通过控制返回地址，转到想要执行的程序入口，进而可以控制整个系统呢?
  - 这正是缓冲区溢出方法的精华所在。

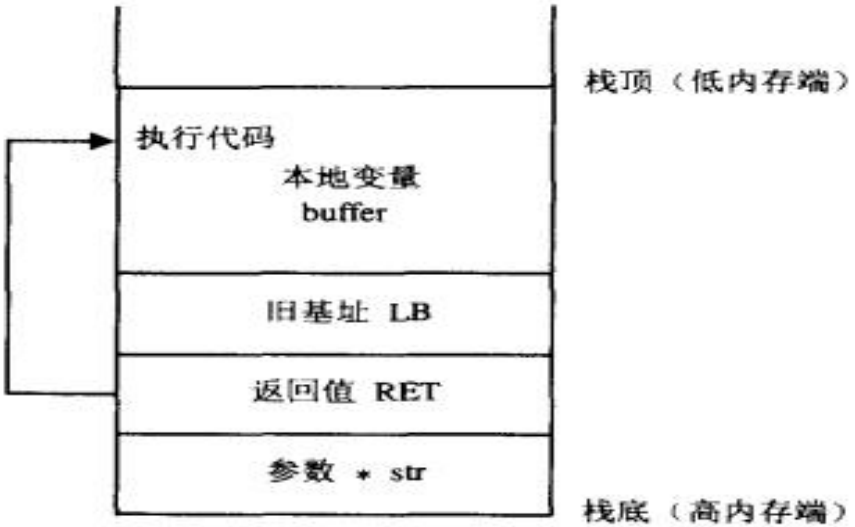
控制程序转移到攻击代码的方法

- 利用活动记录
  - 溢出栈中的局部变量，使返回地址指向攻击代码



控制程序转移到攻击代码的方法

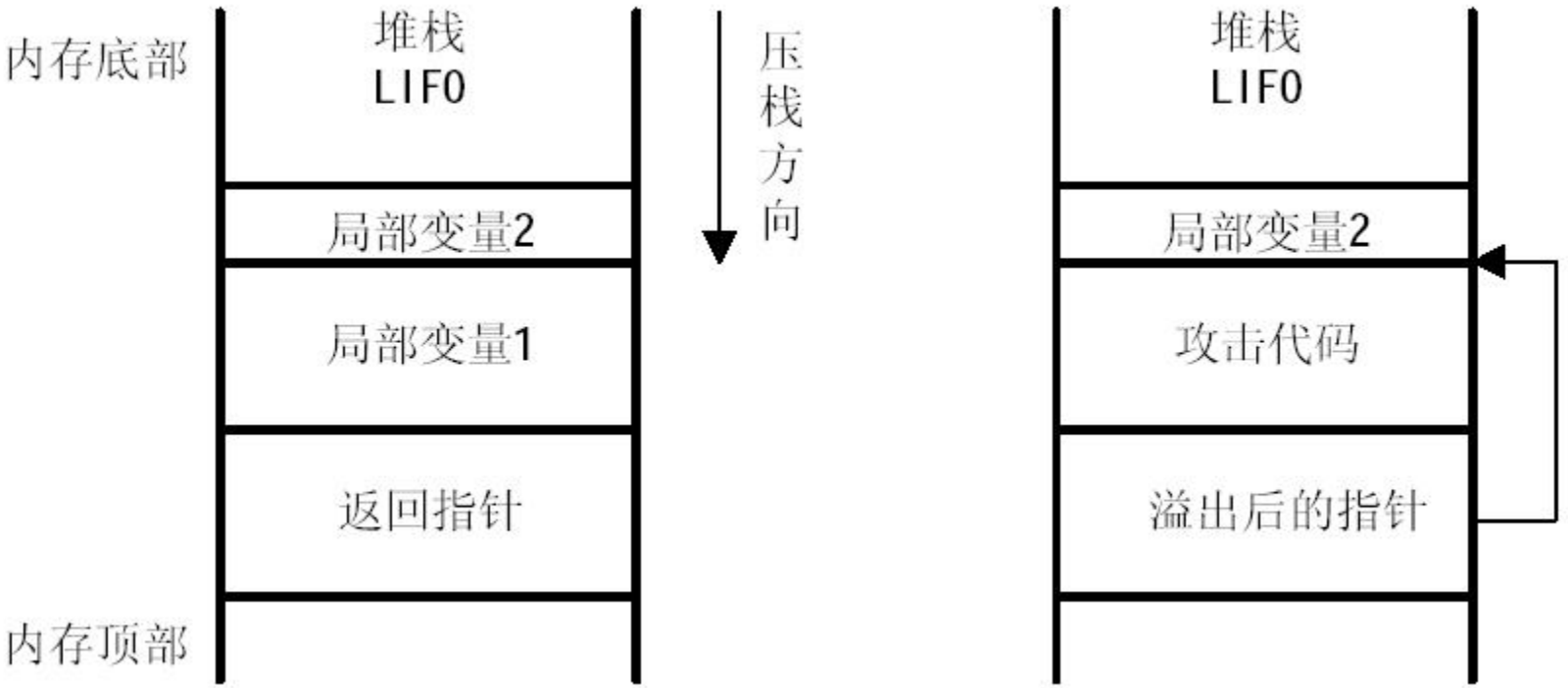
- 如果在溢出的缓冲区中写入我们想执行的代码，再覆盖返回地址(ret)的内容，使它指向缓冲区中某可执行代码的开头，就可以达到运行其指令的目的。



通过堆栈返回指令执行代码

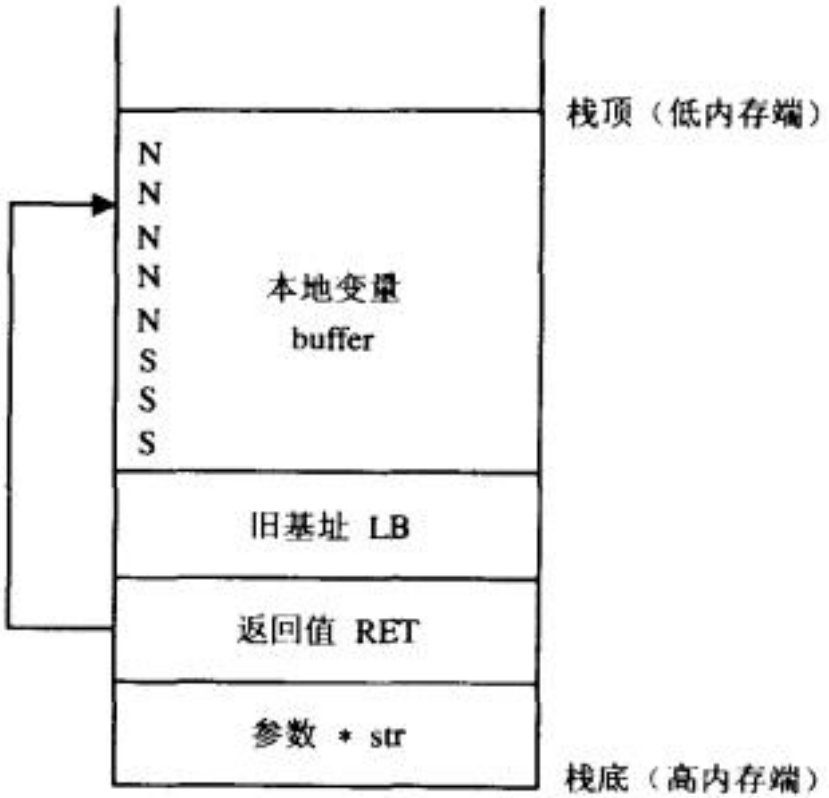
控制程序转移到攻击代码的方法

- 一个字符串中完成代码植入和跳转



缓冲区溢出

- 利用缓冲区溢出进行的系统攻击
  - 如果已知某个程序有缓冲区溢出的缺陷，如何知道缓冲区的地址，在哪儿放入攻击代码呢?
    - 由于每个程序的堆栈起始地址是固定的，所以理论上可以通过反复重试缓冲区相对于堆栈起始位置的距离来得到。但这样的盲目猜测可能要进行数百上千次，实际上是不现实的。
  - 解决的办法是利用空指令NOP。在攻击代码前面放一长串的NOP，返回地址可以指向这一串NOP中任一位置，执行完NOP指令后程序将激活攻击进程。这样就大大增加了猜中的可能性。



通过堆栈返回指令执行代码

NOP	Shellcode	填充字节	Shellcode地址

缓冲区溢出攻击字符串的设计

缓冲区溢出

- 如果large_string是由用户输入的？

```
#include <string.h>
void function(char *str)
{
   char buffer[16];

   strcpy(buffer,str);
}
```

```
void main()
{
   char large_string[256];
   int i;

   for(i=1;i<255;i++)
         large_string[i]='A';
   function(large_string);
}
```

基于堆的溢出

```
#include <stdio.h>
#include <stdlib.h>
Int main(int argc, *argv[])
{file *fd;
Char *userinput=malloc[20];
Char *outputfile=malloc[20];
If (argc<2)
{
Printf(“error”);
Exit(0);}
}
Strcpy(outputfile,”/tmp/notes”);
Strcpy(userinput,argv[1]);
Printf(“userinput @%p:%s\n”,userinput,userinput);
Printf(“outputfile @%p:%s\n”,outputfile,outputfile);
printf(“distance between:%d\n”,outputfile-userinput);
Printf(“----------\n\n”);
```

```
Printf("writing to \"%s\"to the end of
    %s …\n",userinput,outputfile);
Fd=fopen(outputfile,"a");
If (fd==null)
{fprint(stederr,"error opening %s\n",outputfile);
Exit(1);}
Fprint(fd,"%s\n",userinput);
Fclose(fd);
Return(0);
}
```

```
$gcc –o heap heap.c
$sudo chown root.root heap
$sudo chmod u+s heap
```

第一次运行

```
$./heap testing

Userinput @0x80498d0:testing
Outputfile @0x80498e8:/tmp/notes
Distance between:24
-----------------------
Writing to “testing”  to the end of  /tmp/notes…
$cat /tmp/notes
testing
```

第二次运行

$./heap 12345678901234567890123

Userinput @0x80498d0: 12345678901234567890123
Outputfile @0x80498e8:/tmp/notes
Distance between:24

Writing to “12345678901234567890123” to the end of /tmp/notes…
$cat /tmp/notes
Testing
12345678901234567890123

第三次运行

$./heap 123456789012345678901234
Userinput @0x80498d0:123456789012345678901234

Outputfile @0x80498e8:
Distance between:24

Writing to “123456789012345678901234” to the end of …
Error opening
$cat /tmp/notes
Testing
123456789012345678901234

第四次运行

$./heap 1234567890123456789012345testfile
Userinput @0x80498d0:1234567890123456789012345testfile

Outputfile @0x80498e8:testfile
Distance between:24

Writing to “1234567890123456789012345testfile” to the end of testfile;
$cat testfile
1234567890123456789012345testfile

- 由于字符串testfile溢出到outputfile缓冲区。所以程序写的是testfile而不是预期的/tmp/notes。
- 这个程序是suid程序，所以数据可以加到任何文件上
- 在/etc/passwd上加一个有用的串似乎是个不错的想法

$cat /etc/passwd

Root:x:0:0:root:/root:/bin/bash

登陆名 密码 用户 id 组 id 用户名 根目录 登陆shell

如果想加入一个系统管理员的帐户需要下面的字符串

Myroot::0:0:me:/root:/bin/bash

但是我们如果要写入/etc/passwd 则字符串的末尾必须是/etc/passwd

安全漏洞

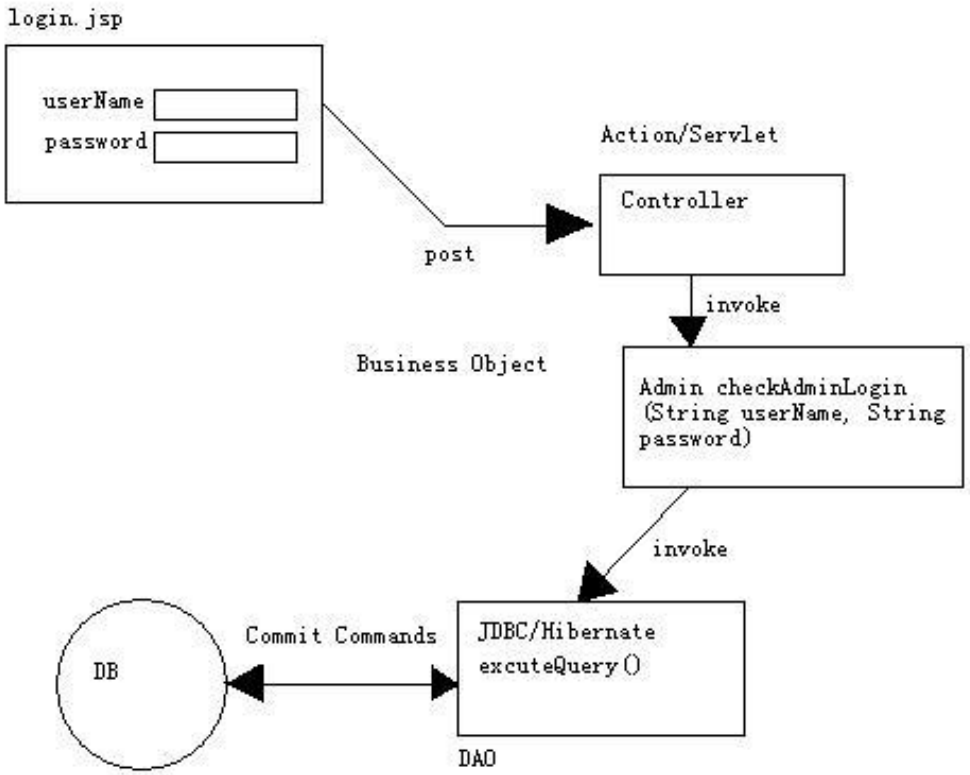
SQL注入

SQL注入

- SQL注入，用户可以提交一段数据库查询代码，根据程序返回的结果，获得某些他想得知的数据或者权限，这就是所谓的SQL Injection，即SQL注入
- SQL 注入的成因主要是因为向数据库提供的SQL查询语句是用字符串拼装的方式生成的
- 最经常遭受SQL注入的页面通常是管理员/用户登陆点。不论是asp 或是jsp，如果不正确的编码，都会出现这样的漏洞

- 举例：
  - 攻击者通过在应用程序中预先定义好的查询语句结尾加上额外的SQL语句元素，欺骗数据库服务器执行非授权的任意查询。

假设我们有一个JSP 页面login.jsp，它会把用户名与密码提交到指定的模块Controller调用checkAdminLogin(String userName, String passWord) 进行登陆验证如果从表中找到匹配的记录，则表示验证成功。否则返回Null 表示登陆验证失败。



checkAdminLogin 将收集到的用户名和密码信息拼装出SQL 字符串，供下层使用，以从数据库中的管理员记录表中读取数据

```
public Admin checkAdminLogin(String userName, String password){
String strSQL ="SELECT * FROM TD_ADMIN AS A WHERE A.USERNAME='" +
userName + "' AND A.PASSWORD='" + password + "'";
```

正常情况下：
在登陆名输入框中输入 yxz
而在密码输入框中输入 123456
那么提交给数据库的将是:
SELECT * FROM TD_ADMIN AS A WHERE A.USERNAME='yxz' AND A.PASSWORD='123456'

如果有人试图在这里进行恶意攻击：
在登陆名输入框中输入 123 (任意值)
而在密码输入框中输入 ' OR '1'='1
由于SQL是靠拼出来的，所以最终提交给数据库的将是:

SELECT * FROM TD_ADMIN AS A WHERE A.USERNAME='123' AND A.PASSWORD='' OR '1'='1'

很显然，这句SQL 由于后面被加上了永真条件，登陆是一定成功的。

防止SQL注入

- 四种方法：
- (1) 在服务端正式处理之前对提交数据的合法性进行检查；
- (2) 封装客户端提交信息；
- (3) 替换或删除敏感字符/字符串；
- (4) 屏蔽出错信息。

Microsoft VBScript 运行时错误 错误 '800a000d'

类型不匹配: 'Clng'

/Inc/syscode.asp, 行 77

htu.cn/Inc/syscode.asp

Microsoft JET Database Engine 错误 '80004005'

'D:\xscxsc\xsc\Inc\database\yiuwekdsodksldfslwifds.mdb'不是一个有效的路径。 确定路径名称拼写是否正确，以及是否连接到文件存放的服务器。

/Inc/conn.asp，行 8

网站使用的是**Access**数据库，通过**JET**引擎连接数据库，而不是通过**ODBC**。

2.程序没有判断客户端提交的数据是否符合程序要求。

3.该**SQL**语句所查询的表中有一名为**ID**的字段。

安全漏洞

脚本注入

- 这里的脚本通常指的是JavaScript脚本，虽然JavaScript运行于客户端，并且有安全沙箱的保护，但是放任恶意JavaScript脚本是十分危险的
- 一个网站，如果对输入未做合理的验证或过滤，在显示输出的时候又未做合适的格式化，那么便存在这种漏洞
- 与SQL注入不同，脚本注入不直接攻击服务端，而是攻击客户端
  - 破坏、窃取信息、植入恶意代码

假设有一个新闻站点，每个新闻允许浏览者进行评论
浏览者提交的评论将被存储在数据库专门的表中，并显示在新闻的下边

如果开发者除了字符长度外没有做任何输入合法验证，那么这个站点的评论输入，就存在安全漏洞。

假设我们在评论中输入如下内容:

<script language="javascript">alert("这里存在脚本注入漏洞.");</script>

那么，这句话**将被存储在数据库评论表中**。
以后，每一个浏览者打开这个新闻页面是，都将会弹出这样一个消息框。

上面的攻击者很仁慈，没有做过多的破坏。但是如果输入:

<script language="javascript">window.location.href="www.baidu.com";</script>

那么打开这个新闻页面，该页面将被从定向到baidu的页面上。
如果目标页面一个有恶意代码的页面，那么每个浏览者的机器都可能中毒。

如果输入:

好文! 顶 <iframe src=”带病毒的页面” width=”0” height=”0”></iframe>

那么在新闻页面上看不到任何异状
但点击该信息的浏览器会悄悄下载病毒

WEB2.0的流行使页面效果更加绚丽，同时也使脚本注入的攻击力提高不少

对策：
提供合理的过滤或者转换程序，在输入存放于数据库前，或者是显示在页面前对数据进行处理。
尽一切可能，避免用户的输入有执行的可能。

安全漏洞

缺省输入

1．缺省SNMP字串

– 简单网络管理协议(SNMP，Simple Network Management Protocol) 是管理员广泛使用的协议，用来管理和监视各种各样与网络连接的设备，如路由器、打印机和计算机。

操作系统级的缺陷

1. Sendmail
   - Sendmail是在UNIX和Linux上用得最多的发送、接收和转发电子邮件的程序。
   - Sendmail在Internet上的广泛应用使它成为攻击者的主要目标。
   - 计算机网络开始的几年里就发现了很多缺陷。

著名的莫利斯事件

著名的莫利斯事件

让Sendmail接收并执行

查通信簿
向外发送

从通信簿中找地址向外发送

从通信簿中找地址向外发送

查通信簿
向外发送

查通信簿
向外发送

操作系统级的缺陷

2. Unicode

- Unicode是一种编码。
- 不论何种平台，何种程序，何种语言，Unicode为每一个字符提供了一个独一无二的序号。
- Unicode标准被包括Microsoft在内的很多软件开发商所采用。

操作系统级最危险的缺陷

2. Unicode

– 通过向IIS服务器发出一个包括非法Unicode UTF-8序列的URL、攻击者可以迫使服务器逐字“进入或退出”目录并执行任意脚本，这种攻击被称为目录转换(Directory Traversal)攻击。

操作系统级最危险的缺陷

2. Unicode

- Unicode用“%2f”和“%5c”分别代表“/”和“\”。但也可以用所谓的“超长”序列来代表这些字符。
- “超长”序列是非法的Unicode表示符，它们比实际代表这些字符的序列要长。
  - “/”和“\”均可以用一个字节来表示。
  - 超长的表示法，例如用“%c0%af”代表“/”用了两个字节。

操作系统级最危险的缺陷

2. Unicode

- IIS不对超长序列进行检查。
- 这样在URL中加入一个超长的Unicode序列，就可以绕过IIS的检查。
  - 如果发出的请求来自一个可执行的目录，攻击者可以在服务器上运行可执行文件。
  - 安装IIS 4.0的Microsoft windows NT 4.0和安装了IIS 5.0，而没有安装service Pack 2的Windows 2000 server都存在着这个漏洞。

操作系统级最危险的缺陷

2. Unicode

- 这样的一个URL示例：
  http://192.168.0.27/Scripts/..%c0%af../winnt/system32/cmd.exe?/c+dir+d:\
- 如果使用的是三字节UTF-8编码，也可以进行相同的攻击。下面的URL等价于上面的URL:

  http://192.168.0.27/Scripts/..%e0%c0%af../winnt/system32/cmd.exe?/c+dir+d:\

操作系统级最危险的缺陷

2. Unicode

- 这样的攻击是如何完成的呢？
  - 这里的“%c0%af”是“/”的一个非法的Unicode表示。
  - URL使得Web服务器的操作系统把这个Unicode字符理解为反斜杠，有效地退回了/Scripts/所在文件夹之上的两个文件夹层，并锁定了/wint/system32/cmd.exe。
    - /Scripts/文件夹通常位于c:\inetpub\Scripts目录这个位置。

操作系统级最危险的缺陷

2. Unicode

- 这样的攻击是如何完成的呢？
  - 在正常情况下，Web服务器绝不会允许URL访问Web文档目录（在这里是C:\inetpub）之外的任何位置。怎样绕过检测的呢？

- Web服务器软件（IIS）在执行目录位置检验时并没有识别出“/”的Unicode表示。而在服务器内部，..%c0%af../被解释为../../并且Web服务器访问的资源现在变成了C:\inetpub\scripts\..\..\winnt\system32\cmd.exe，而这个地址的最后又指向了c:\winnt\system32\cmd.exe，并执行了这个命令。

漏洞库

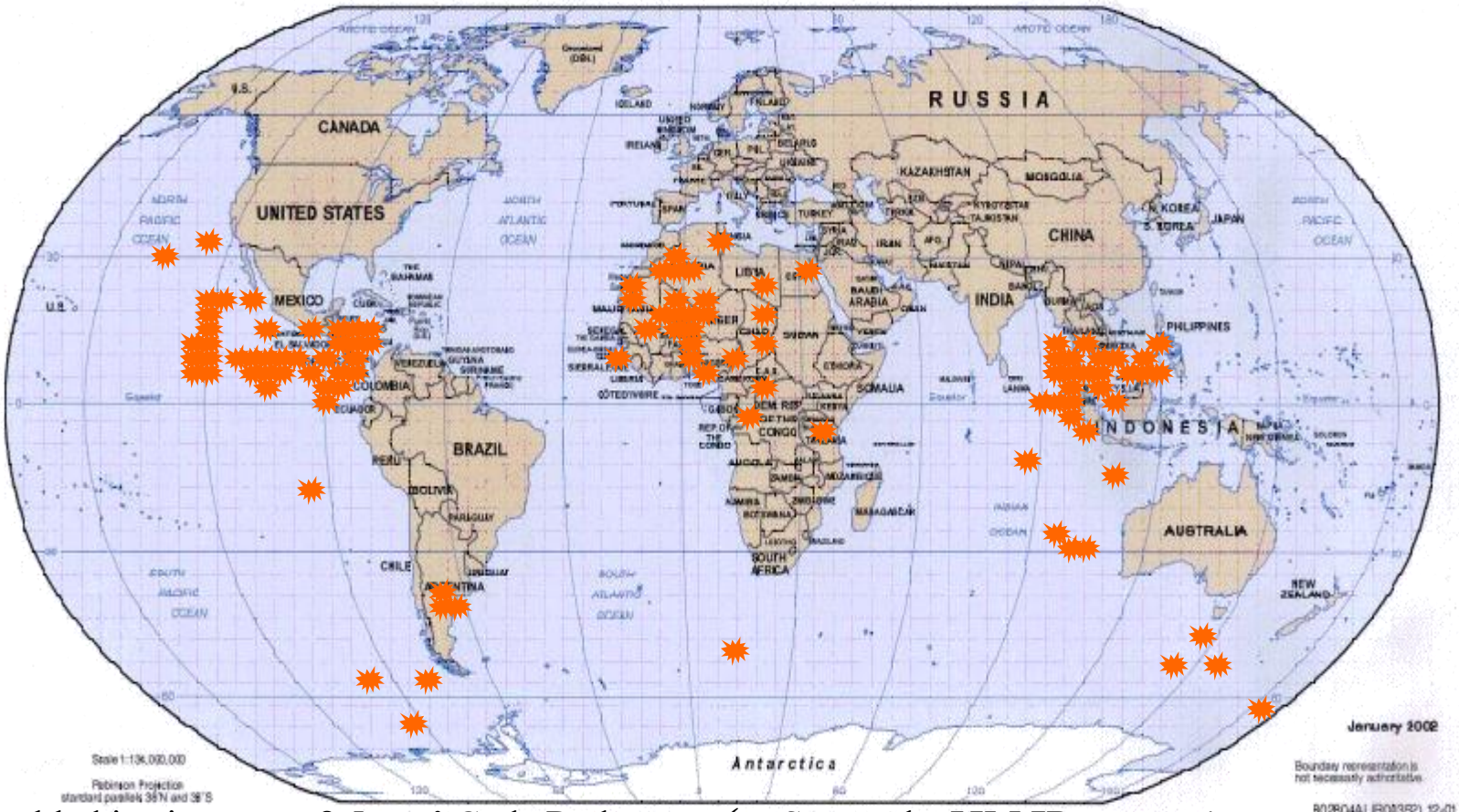
一些权威的漏洞库站点

http: / / WWW. securityfocus. corn　国外著名的漏洞发布站点，即Bugtraq
http: / / WWW. eve. mitre. org　国际标准CVE
http: / / ntsecurity. net　关于Windows系统安全的综合性网站
http: / / WWW. cert. org　美国计算机应急响应小组
http: / / xforce. iss. net　由ISS公司发布的漏洞库
http: / / WWW. bugnet. com　漏洞修补网站
http: / / icat. nist. gov / icat. cfm　ICAT漏洞发布及漏洞库搜索站点
http: / / WWW. nipc. org. cn　国家计算机网络入侵防范中心
http: / / WWW. nsfocus. com　：绿盟科技
http: / / WWW. chinafirst. org. cn　中国信息安全论坛
http: / / WWW. cert. org. cn　中国计算机网络应急处理中心
http: / / WWW. nipc. org. cn /　国家计算机网络入侵防范中心

恶意代码攻击

- 恶意软件：人为编写、违背计算机信息系统用户的意愿、执行时以破坏、窃取、恶意利用等为目的软件
  - 恶意的目的、攻击性
  - 本身是程序或代码
  - 通过执行发生作用

感染速度



- ✓ Doubled in size every 8.5**secs!** Code Red
- ✓ Peaked about **8 mins** after released!
- ✓ Scanned >**55M IPs** per sec!
- ✓ Reached **90%** of vulnerable machines world-wide (75,000) in 10mins !

计算机病毒

- 计算机病毒，是一段附着在其他程序上的可以实现自我繁殖的程序代码。复制生成后的新病毒同样具有感染其他程序的功能。

恶意网页攻击

- 利用一些script语言编写的一些恶意代码，当用户登录某些含有网页病毒的网站时，下载有关的恶意代码到本机后，网页病毒会被解释执行。

- 利用系统资源进行破坏：轻则修改用户的注册表，使用户的首页、浏览器标题改变，重则可以关闭系统的很多功能，使用户无法正常使用计算机系统，严重者则可以将用户的系统进行格式化。

红色代码 II

- “红色代码II”是一种专门攻击Windows NT 4.0以及Windows 2000系统的恶性网络蠕虫VirtualRoot(虚拟目录)病毒。
- 该病毒利用微软Index Server(ida/idq)ISAPI扩展远程溢出漏洞，通过80端口发送一个构造后的HTTP GET请求到服务器，导致处理函数的堆栈溢出(Overflow)。
- 当函数返回时，原返回地址已被病毒数据包覆盖，系统强迫运行病毒代码，此时病毒被激活，并运行在IIS服务程序的堆栈中。这种蠕虫病毒修改Windows注册表并放置特洛伊木马程序(boot.exe并存在scripts/目录下)。
- 通过HTTP GET的请求运行scripts/root.exe来获得对受感染机器的完全的控制权。

冲击波

- 冲击波病毒(Wrom. MSBlast. 6176)(原名“新流言”)长度6176B，病毒类型属蠕虫型，该病毒利用RPC漏洞进行快速传播。病毒程序采用UPX压缩技术，使得病毒体积较小，便于在网络上快速传播。
- 该病毒代码由下面几个模块组成：
- （1）修改注册表。在注册表项HKEY—LOCAI。一MACHINE\SOFTWARE\Microso~t\Windows\Current Version\Run下添加键值：“windows auto update”-“msblast. exe’’，以使蠕虫可以开机自动运行。
- (2)攻击模块。攻击病毒自我生成的IP地址和RPC服务默认端口，为传播自己做准备。
- (3)监听UDP 69端口，当有服务请求，就发送Msblast.exe文件。
- (4)发送命令到远端计算机(被攻击计算机)，以使其连接被感染计算机(本地计算机)下载并运行该病毒；
- (5)设置触发条件：如果当前月份大于8月，或当前日期大于15号，对“Windows update.COIIl”实施DOS攻击。

后门程序

后门程序

- 由于程序员在设计一些功能复杂的程序时，一般采用模块化的程序设计思想，将整个项目分割为多个功能模块，分别进行设计、调试，这时的后门就是一个模块的秘密入口。
- 在程序开发阶段，后门便于测试、更改和增强模块功能。正常情况下，完成设计之后需要去掉各个模块的后门，不过有时由于疏忽或者其他原因(如将其留在程序中，便于日后访问、测试或维护)后门没有去掉，一些别有用心的人会利用穷举搜索法发现并利用这些后门，然后进入系统并发动攻击。

木 马

设置木马

该恶意软件利用系统漏洞进入用户的计算机系统，通过修改注册表等方式自启动，运行时有意不让用户察觉，将用户计算机中的所有信息都暴露在网络中。大多数黑客程序的服务器端都是木马病毒。

僵尸网络（Botnet）

- 僵尸网络
  - 僵尸网络是可被攻击者远程控制的被攻陷主机所组成的网络。
  - 攻击者利用互联网秘密建立的可以集中控制的计算机群。其组成通常包括被植入“僵尸”程序的计算机群，一个或多个控制服务器，控制者的控制终端等。
  - 僵尸网络是攻击者出于恶意目标，传播**僵尸程序**将大量主机感染成僵尸主机，并由**僵尸控制程序**通过一对多的命令和控制信道所组成的网络。

熊猫烧香病毒

- 该病毒发作时

熊猫烧香（变种 spoclsv.exe）

- 感染机制：IE的漏洞、木马、局域网、移动存储设备。
- 感染过程：
  - 运行后复制自身到系统目录:%System%\drivers\spoclsv.exe。
  - 创建启动项 [HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run]"svcshare"="%System%\drivers\spoclsv.exe“
  - 修改注册表信息干扰“显示所有文件和文件夹”设置：HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Advanced\Folder\Hidden\SHOWALL]"CheckedValue"=dword:00000000
  - 在各分区根目录生成副本：X:\setup.exe X:\autorun.inf
  - 关闭对头窗口、进程，禁用对头服务，禁用系统安全项
  - 遍历目录，感染exe、com、scr、pif等文件；将自身捆绑在被感染文件前端，并在尾部添加标记信息：QUOTE:.WhBoy{原文件名}.exe.{原文件大小}.
  - 在本局域网内尝试弱口令

熊猫烧香（变种 SVCH0ST.exe ）

- 每隔一段时间（18秒）访问固定链接，查询新的变种
- 刷新bbs.qq.com，某QQ秀链接。
- 连接*****.3322.org下载某文件，并根据该文件记录的地址，去www.****.com下载某ddos程序，下载成功后执行该程序。
- 连接ddos2.****.com，获取攻击地址列表和攻击配置，并根据配置文件，进行相应的攻击。
  配置文件如下：
  www.victim.net:3389
  www.victim.net:80
  www.victim.com:80
  www.victim.net:80

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件系统面临的威胁

主要内容

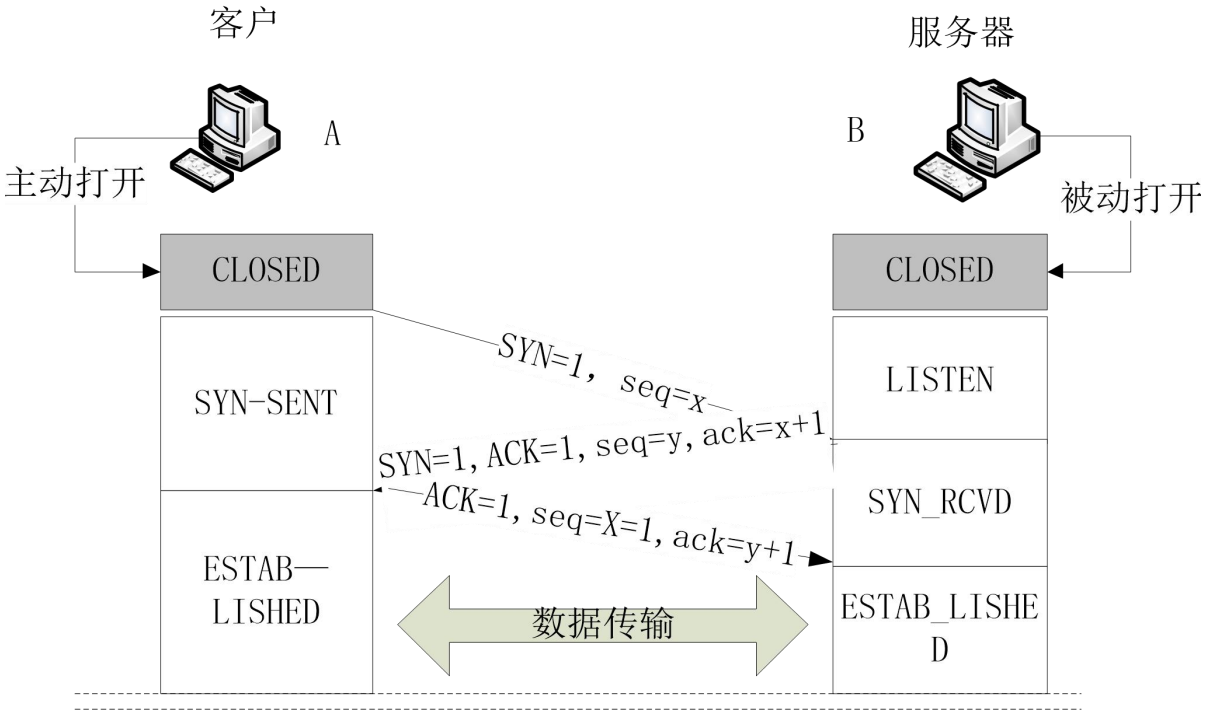
- 概述
- 计算机系统缺陷
  - 漏洞利用
  - 恶意代码攻击
- 网络与协议缺陷
  - TCP/IP协议缺陷
  - 网络攻击
- 攻击的分类

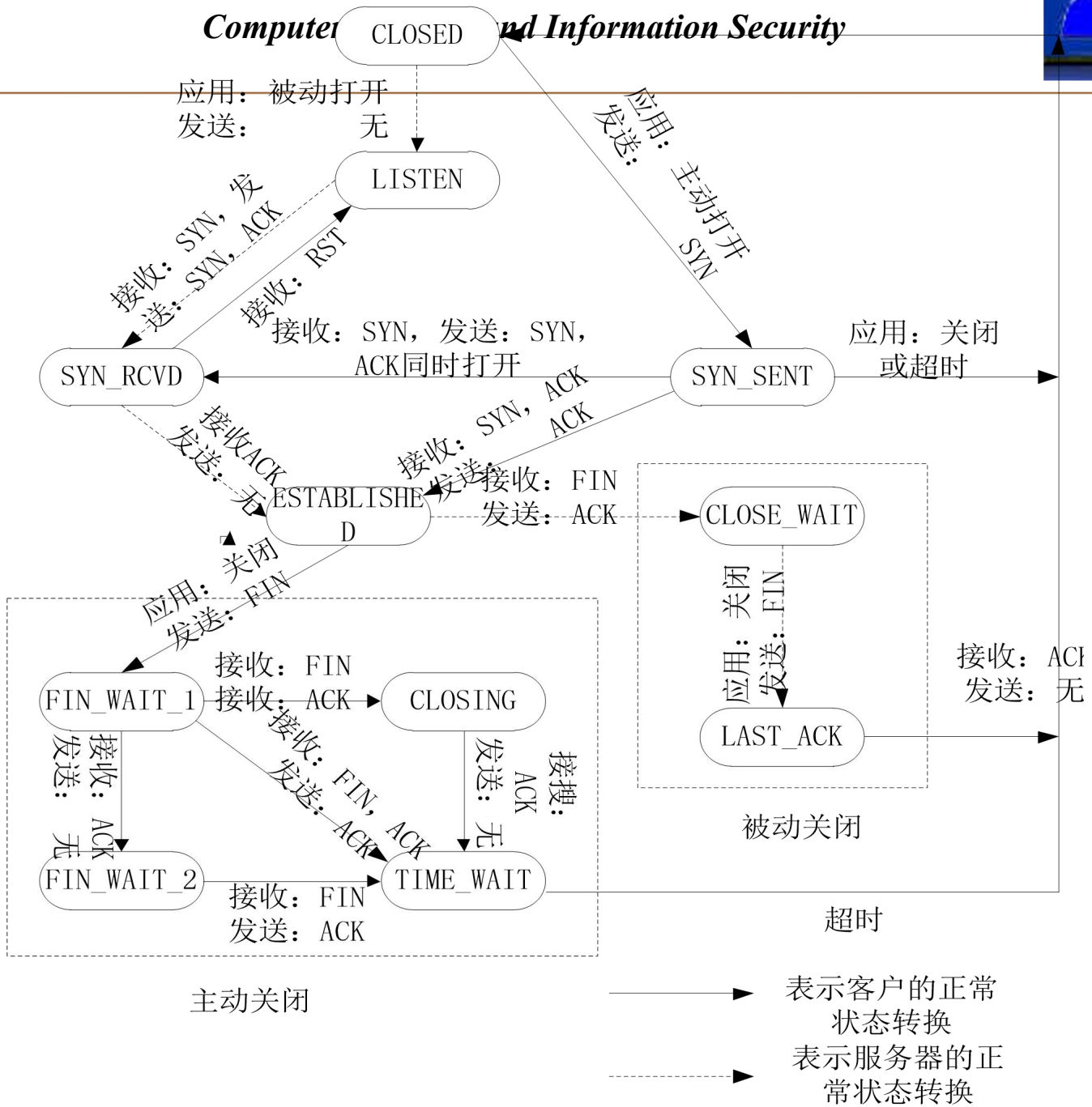
TCP/IP协议缺陷

- TCP/IP的优点
  - 简单性、可扩展性强、尽力而为等原则。
- TCP/IP协议也存在着一系列的安全缺陷。
  - 协议的公开性
  - 有的缺陷是由于源地址的认证问题造成的
  - 有的缺陷则来自网络控制机制、路由协议等等。
  - 这些缺陷，是所有使用TCP/IP协议的系统所共有的。

TCP协议过程

- 握手


客户
服务器
A
B
主动打开
被动打开
CLOSED
CLOSED
SYN=1, seq=x
SYN-SENT
LISTEN
SYN=1, ACK=1, seq=y, ack=x+1
SYN_RCVD
ACK=1, seq=X=1, ack=y+1
ESTAB—LISHED
数据传输
ESTAB_LISHED


CLOSED
应用：被动打开
发送：无
LISTEN
接收：SYN，发送：SYN，ACK
接收：RST
应用：主动打开
发送：SYN
SYN_RCVD
接收：SYN，发送：SYN，ACK同时打开
SYN_SENT
应用：关闭或超时
接收ACK
发送：无
接收：SYN，ACK
发送：ACK
ESTABLISHED
接收：FIN
发送：ACK
CLOSE_WAIT
应用：关闭
发送：FIN
LAST_ACK
接收：ACK
发送：无
被动关闭
应用：关闭
发送：FIN
FIN_WAIT_1
接收：FIN
接收：ACK
CLOSING
接收：ACK
发送：无
接收：FIN，ACK
发送：ACK
接收：ACK
发送：无
FIN_WAIT_2
接收：FIN
发送：ACK
TIME_WAIT
超时
主动关闭
表示客户的正常状态转换
表示服务器的正常状态转换

- UDP协议
  - 面向无连接的网络协议

TCP/IP协议缺陷导致的威胁

- 地址伪造
- 网络窃听
- 劫持篡改
- 中间人攻击
- 重放式攻击
- DDOS攻击

地址伪造

- A->B: SRC=A, DES=B,SYN序列号=M
- B->A: SRC=B, DES=A,SYN序列号=N,ACK应答序列号=M+1
- A->B: SRC=A, DES=B,ACK应答序列号=N+1

- 伪造：X伪装成A
- X->B: SRC=A, DES=B,SYN序列号=M
- B->A: SRC=B, DES=A,SYN序列号=N,ACK应答序列号=M+1
- X->B: SRC=A, DES=B,ACK应答序列号=N+1

网络窃听

- 网络监听
  - 网络监听(嗅探)是一种监视网络状态、数据流以及网络上传输信息的技术
  - 以旁路监听模式复制网上传输的信息。也就是说，使用网络监听可以有效地截获网上的数据，这是黑客使用最多的方法。

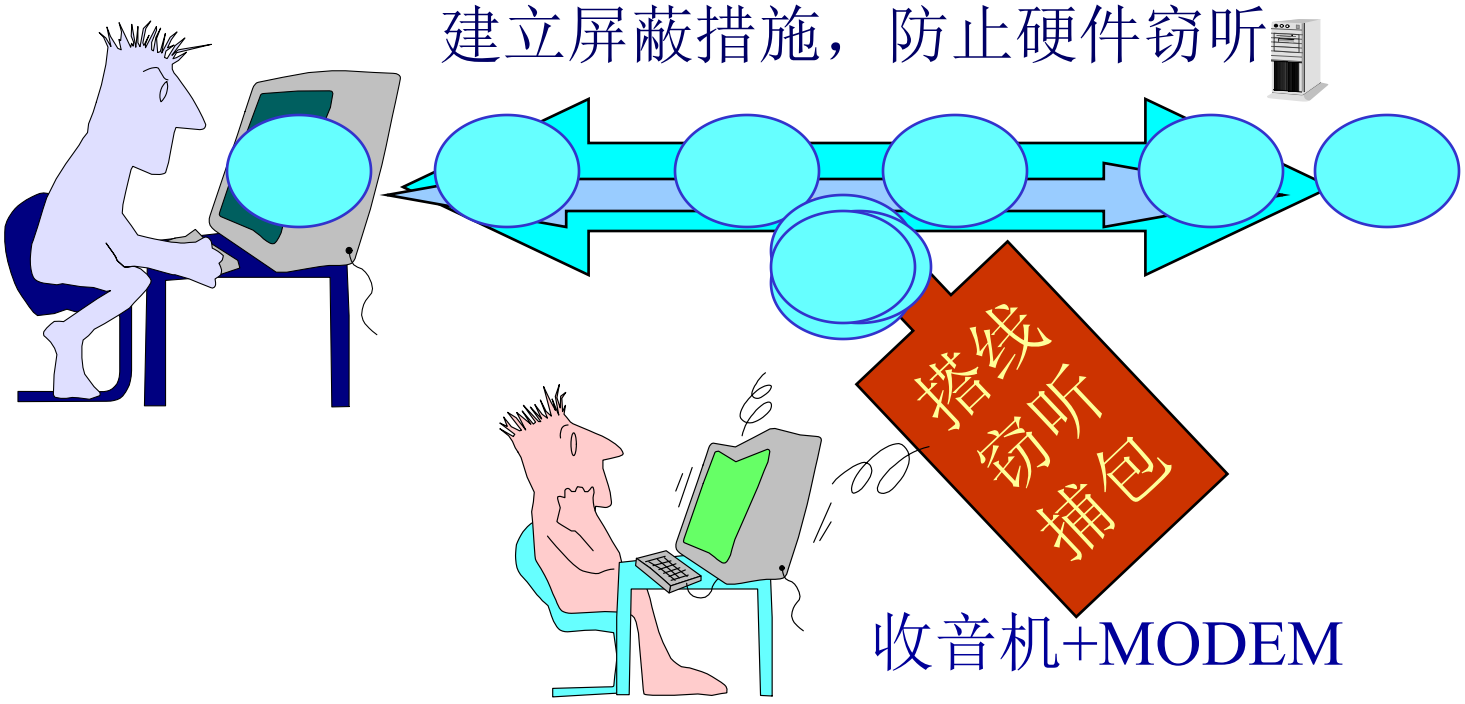
Tcpdump—原理：通过调用libpcap中的例程来捕获包。但它既可以对IP包，也可以对IPX及ether、FDDI等包作出分析。可给出数据链路层、网络层、传输层的包头格式及包数据。可指定主机(host)、网络(net)端口(port)、协议（proto)等来过滤捕获。

1、消除last命令中的痕迹：执行wedit程序可删除此用户的在last命令中的登录痕迹；执行wnedit程序可删除此用户在last命令中最近n次登录痕迹

2.消除/var/adm/messages中的痕迹:用vi来修改messages

3.消除home目录下的.bash_history 文件中的痕迹

DPDK,linsniff,sniffit,tcpdump



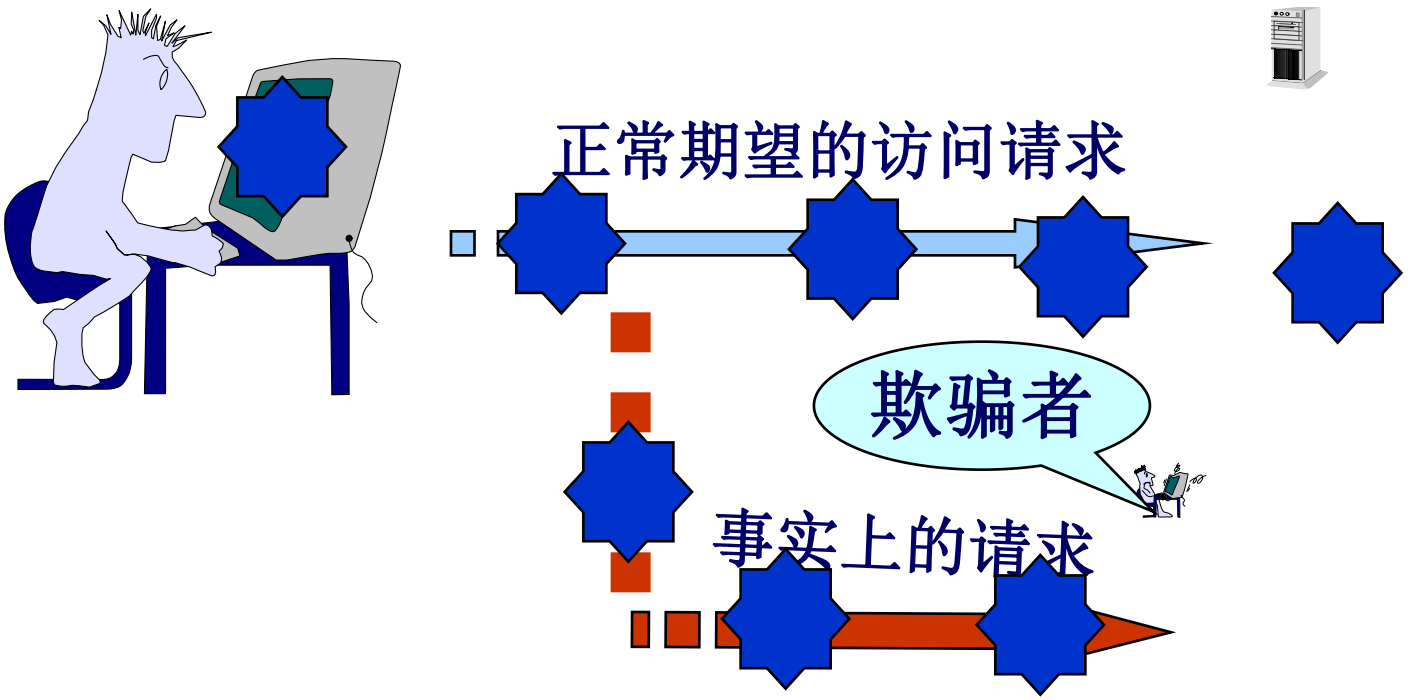
会话劫持

- 会话劫持（Session Hijack）
  - 结合了嗅探以及欺骗技术在内的攻击手段。
  - 在一次正常的会话过程当中，攻击者作为第三方参与到其中，可以篡改正常数据包，插入恶意数据；也可以在双方的会话当中进行监听，并代替某一方主机接管会话。

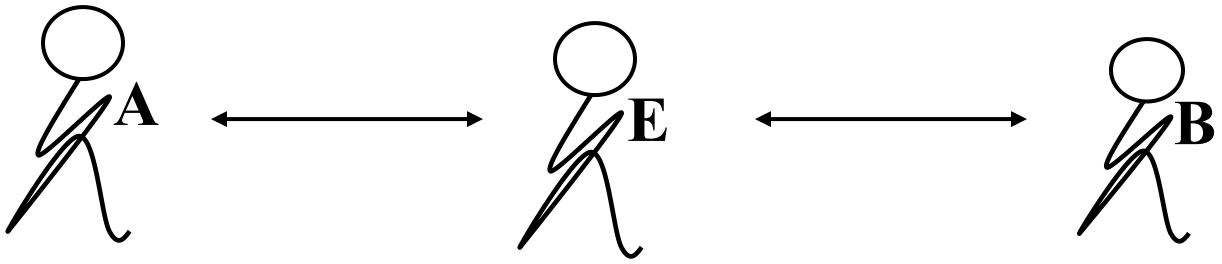
TCP会话劫持

- TCP会话劫持
  - 攻击者嗅探到正在进行TCP通信的两台主机之间传送的报文，得知该报文的源IP、源TCP端口号、目的IP、目的TCP端号，从而可以计算得知其中一台主机对将要收到的下一个TCP报文段中seq和ackseq值的要求。
  - 在该合法主机收到另一台合法主机发送的TCP报文前，攻击者根据所获得信息向该主机发出一个带有Payload的TCP报文，如果该主机先收到攻击报文，就可以把合法的TCP会话建立在攻击主机与被攻击主机之间。
  - 带有Payload的攻击报文能够使被攻击主机对下一个要收到的TCP报文中的确认序号（ackseq）的值的要求发生变化，从而使另一台合法的主机向被攻击主机发出的报文 被 被攻击主机拒绝。
  - TCP会话劫持使攻击者避开了身份验证和安全认证，从而使攻击者直接进入对被攻击主机的的访问状态。

TCP欺骗与会话劫持



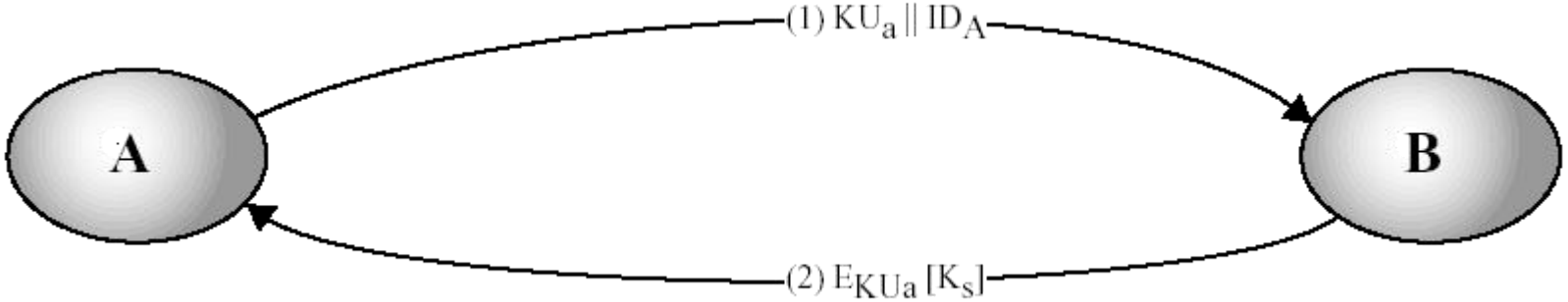
中间人攻击
(MITM, man in the middle)



- 如果通讯双方没有任何先决条件，那么这种攻击总是存在的
- A和B的协商过程很容易受到这一类攻击

例子

- Merkle协议



A生成{KUa,KRa}, A→B: (ID_A,KUa)
B生成随机密钥Ks, B→A: E_{KUa}(Ks)
A解密E_{KUa}(Ks)得到Ks: $D_{KRa}(E_{KUa}(Ks))$
A丢弃{KUa,KRa},B丢弃KUa

Merkle协议的中间人攻击

① A生成**{KUa,KRa}**,**A→B:(ID_A,KUa)**

② E截获,生成**{KUe,KRe}**冒充**A→B:(ID_A,KUe)**

③ B生成随机密钥**Ks**,**B→A:E_{KUe}(Ks)**

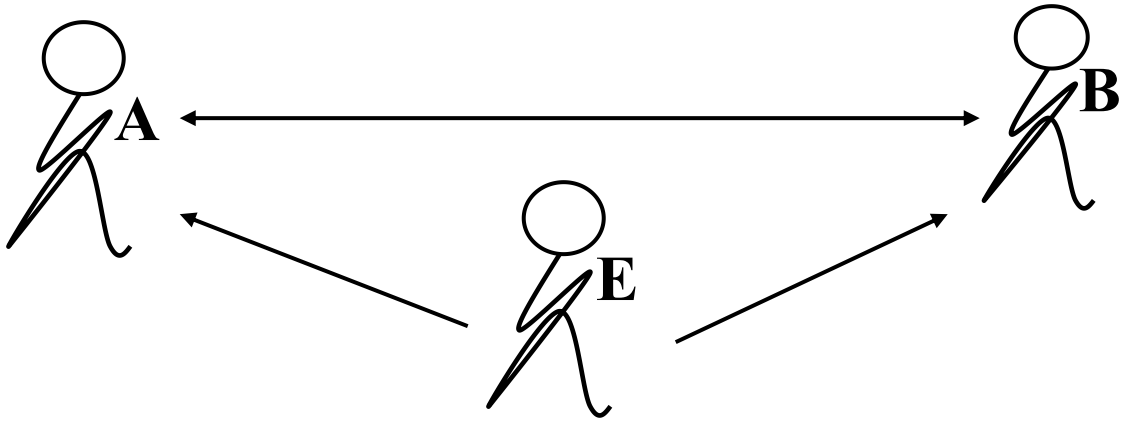
④ E截获,解密后再用**E_{KUa}**加密**Ks→A:E_{KUa}(Ks)**

⑤ A丢弃**{KUa,KRa}**,B丢弃**KUa**

- E获得了**Ks**,故以后只需进行窃听.
- A,B并不知晓它们被攻击了

重放式攻击

- 重放
  - 涉及一个数据单元的被动获取以及后继的重传，以产生一个未授权的效果。



- **偷听者可以记录下当前的通讯流量，以后在适当的时候重发给通讯的某一方，达到欺骗的目的**

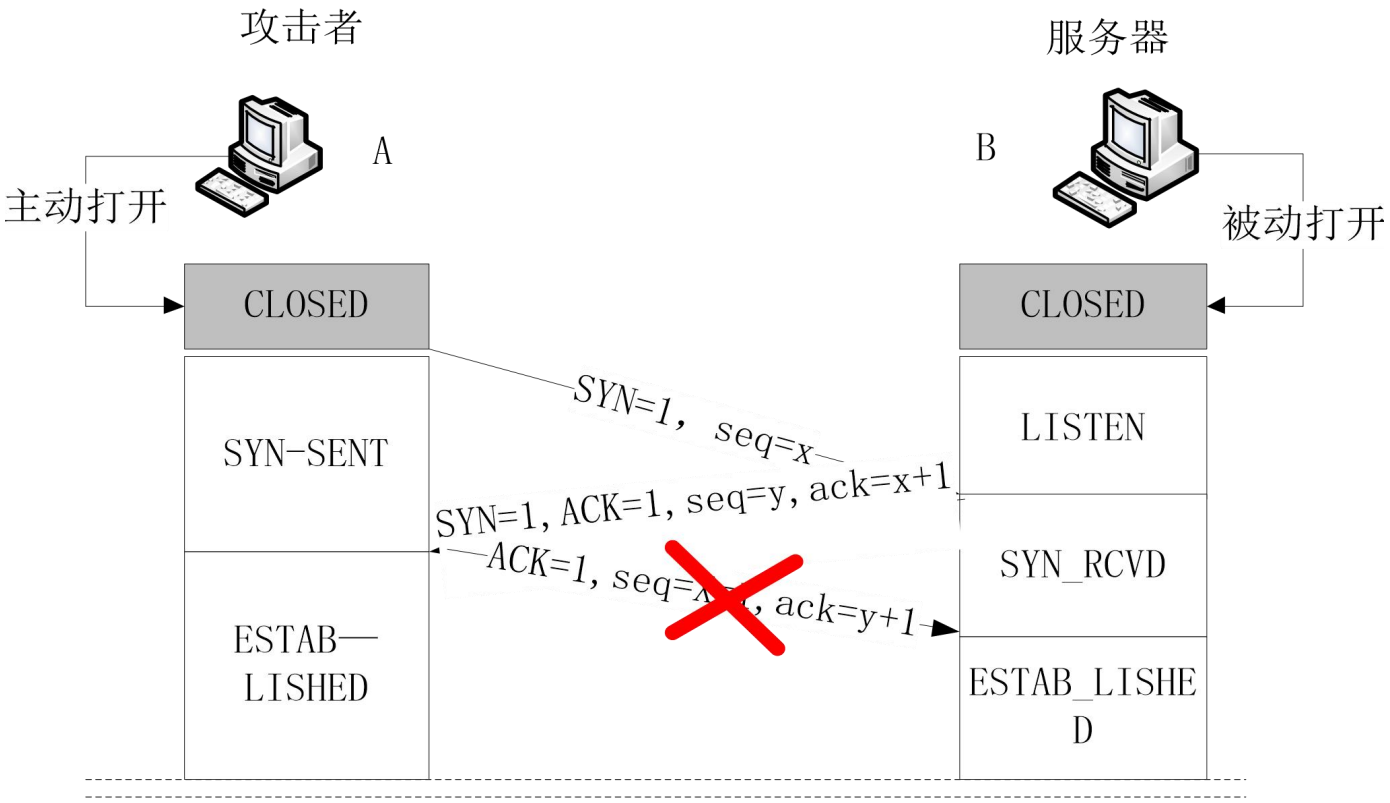
重放

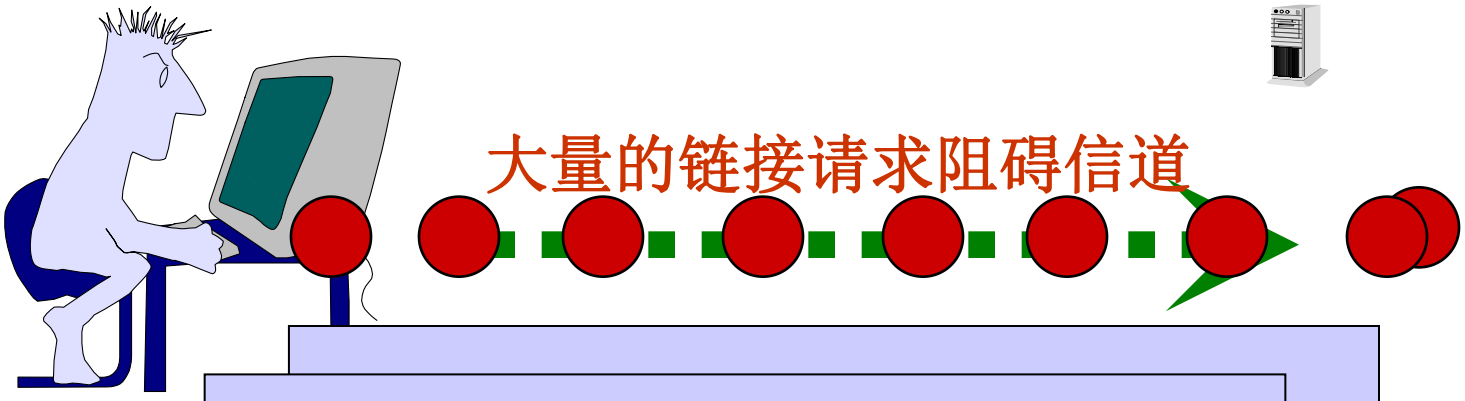
- 常见的消息重放攻击形式有：
  - 简单重放：攻击者简单复制一条消息，以后再重新发送它；
  - 可被日志记录的重复：攻击者可以在一个合法有效的时间窗内重放一个带时间戳的消息；
  - 不能被检测到的重复：原始信息已经被拦截，无法到达目的地，而只有重放的信息到达目的地。
  - 反向重放，不做修改。向消息发送者重放。
    - 当采用传统对称加密方式时，这种攻击是可能的。
    - 因为消息发送者不能简单地识别发送的消息和收到的消息在内容上的区别。

拒绝服务攻击

- 1、带宽耗用
  - 带宽耗用(bandwidth-consumption)攻击，其本质是攻击者消耗掉通达某个网络的所有可用带宽。这可以发生在局域网上，不过更常见的是攻击者远程消耗资源。
- 2 、服务资源衰竭
  - 资源衰竭(resource-starvation)攻击与带宽耗用攻击的差异在于前者集中于系统资源而不是网络资源的消耗。一般地说，它涉及诸如CPU利用率、内存、文件系统限额、系统进程总数、系统连接总数之类系统资源的消耗。攻击者往往拥有一定数量系统资源的合法访问权。资源衰竭DoS攻击通常会因为系统崩溃、文件系统变满、进程被挂起或服务连接资源占满等原因而导致服务资源不可用，如正常用户与Web Server不能建立连接。

Syn-flood


攻击者
服务器
A
B
主动打开
被动打开
CLOSED
CLOSED
SYN-SENT
LISTEN
SYN=1, seq=x
SYN=1, ACK=1, seq=y, ack=x+1
SYN_RCVD
ACK=1, seq=x+1, ack=y+1
ESTAB—
LISHED
ESTAB_LISHE
D



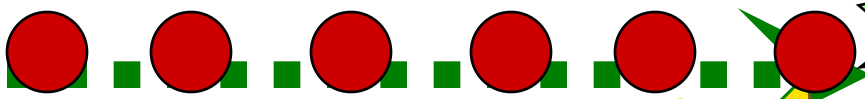
在TCP包中通过将同步位设为1的方式来请求连接，如果得到连接确认后不予理会，而是继续发出申请，将有可能耗尽服务器端的有限的响应连接的资源

恶意数据流

- 网络用户流量行为模型
  - 合作数据流：相同网络用户之间行为遵从拥塞控制机制；不同随机网络用户之间行为相互独立；
  - 恶意数据流：不遵从以数据包丢弃为触发的网络拥塞控制，即数据包丢弃概率与数据包发送速率不相关。相同网络用户之间行为不遵从拥塞控制机制；不同随机网络用户之间行为可能不相互独立

- 恶意流性质
  - 非合作性：恶意数据流的一个本质属性就是非合作性。具体来说，当网络发生拥塞时，TCP类合作数据流至少会降低拥塞窗口的一半作为拥塞响应；拥塞停止后，则以某一恒定速率增加拥塞窗口，但增加的速率最多是每个RTT一个数据包。而恶意数据流在网络发生拥塞的时候，它不会根据拥塞得严重程度而调节其数据发送速率。
  - 高带宽性： 一方面，由于恶意数据流既没有慢启动机制，也不遵从拥塞控制机制，其到达路由器的速率远远大于TCP友好流的到达速率；另一方面，即使发生了数据包丢弃，也不会降低其发送数
- 带来的问题
  - 带宽分配不公平，行为正常流的“带宽饥饿”

大量的链接请求阻碍信道



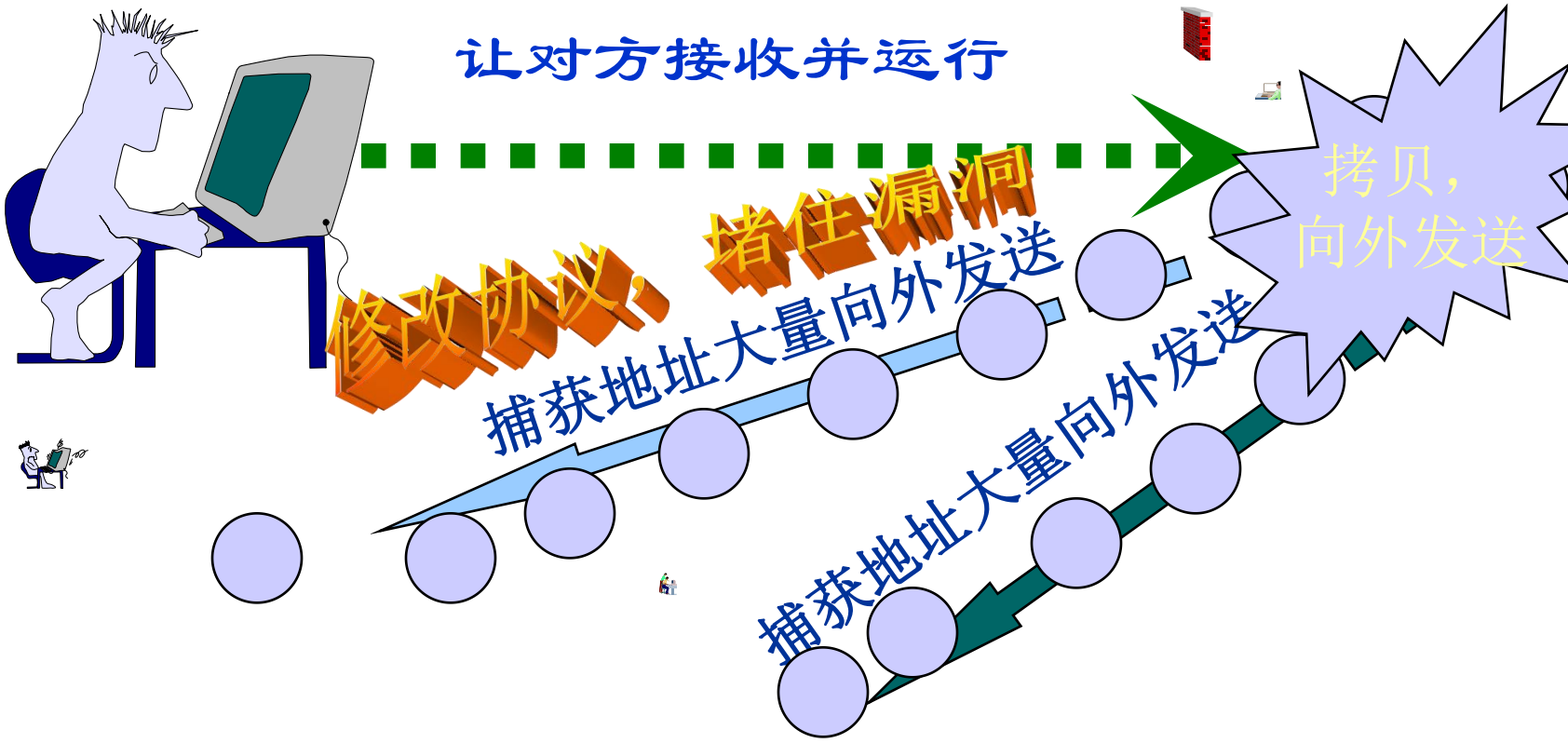
阻塞

pong, synk4

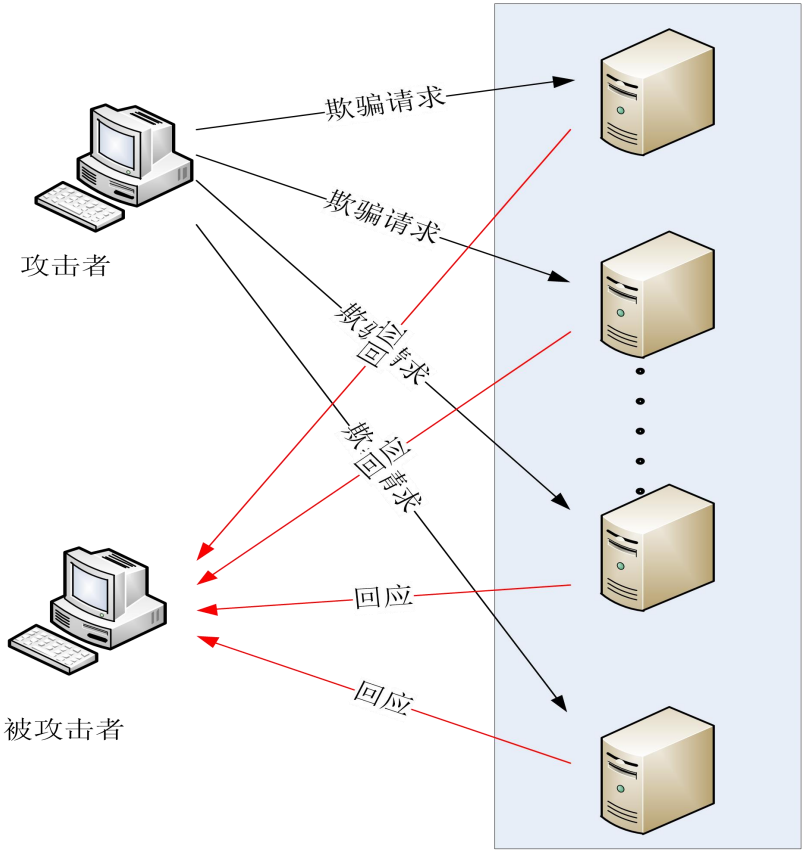
合理的访问请求得不到响应

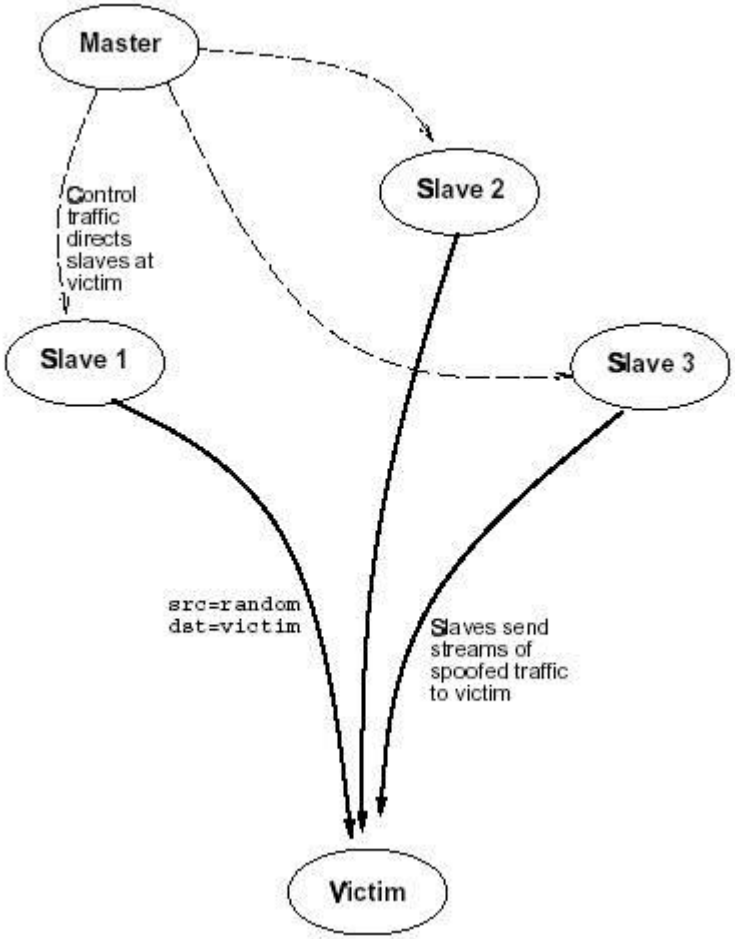
这类攻击通常都设置假IP地址

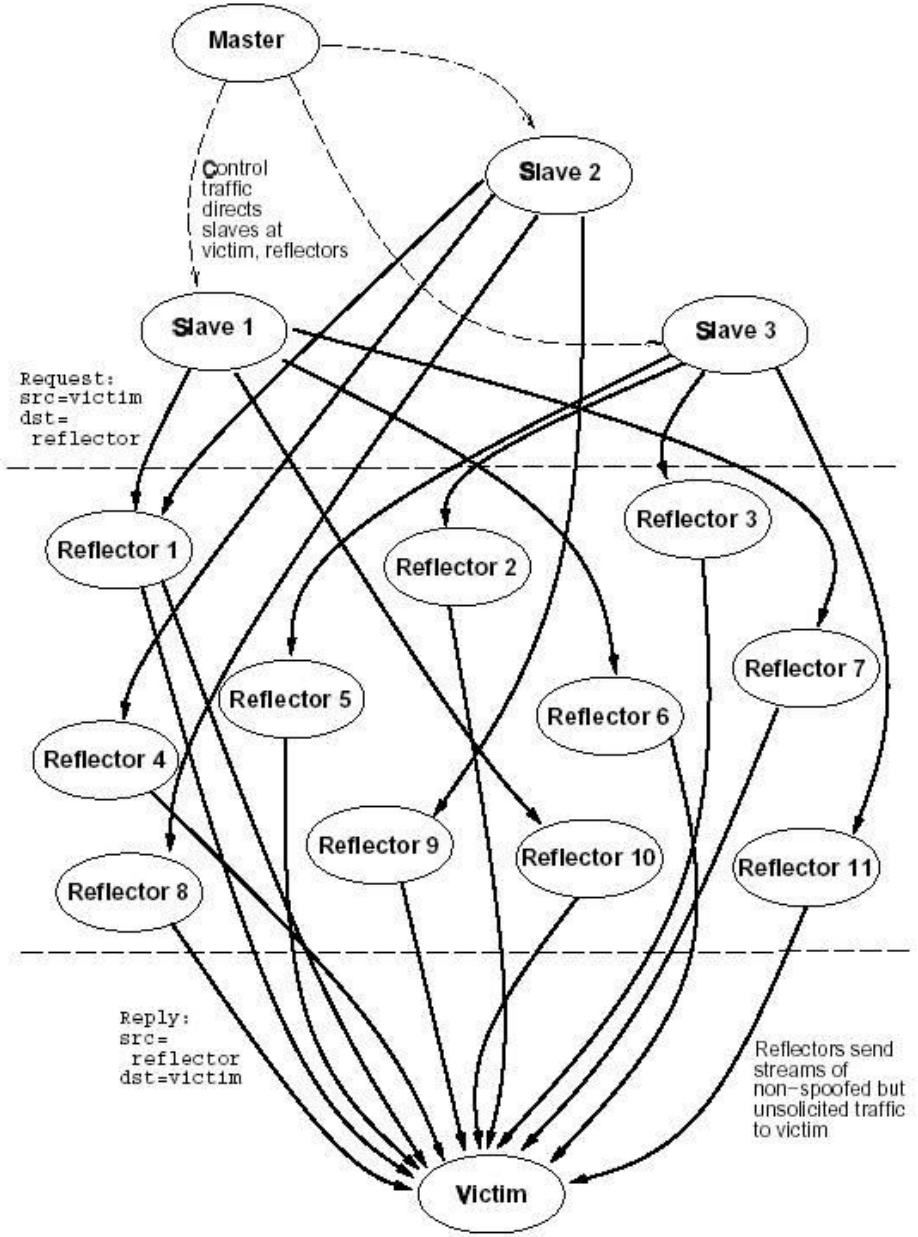
分布式拒绝服务攻击



Smurf攻击



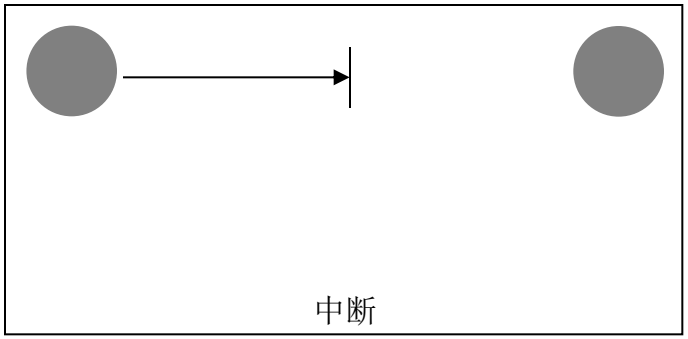

Master
Control traffic directs slaves at victim
Slave 1
Slave 2
Slave 3
src=random
dst=victim
Slaves send streams of spoofed traffic to victim
Victim

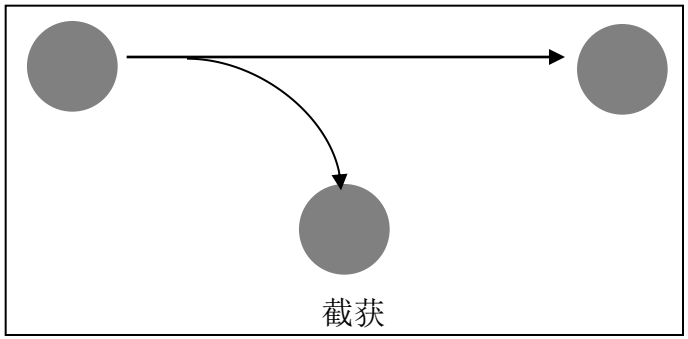

Master
Control traffic directs slaves at victim, reflectors
Slave 1
Slave 2
Slave 3
Request:
src=victim
dst=
reflector
Reflector 1
Reflector 2
Reflector 3
Reflector 4
Reflector 5
Reflector 6
Reflector 7
Reflector 8
Reflector 9
Reflector 10
Reflector 11
Reply:
src=
reflector
dst=victim
Reflectors send streams of non-spoofed but unsolicited traffic to victim
Victim

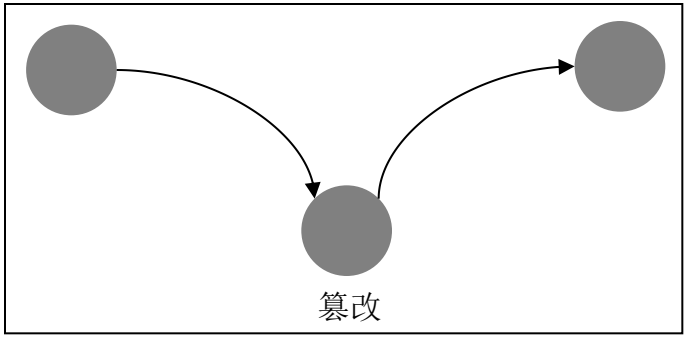
主要内容

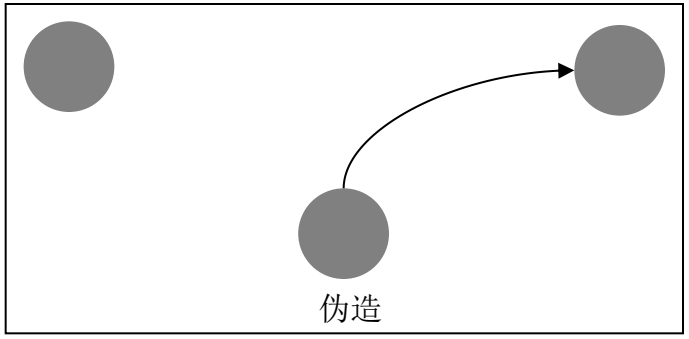
- 概述
- 计算机系统缺陷
  - 漏洞利用
  - 恶意代码攻击
- 网络与协议缺陷
  - TCP/IP协议缺陷
  - 网络攻击
- 攻击的分类

对安全的攻击





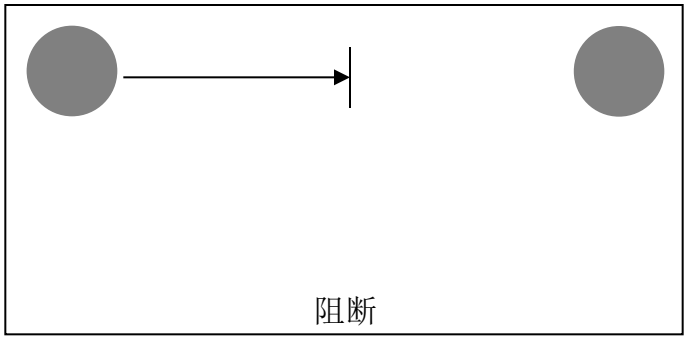




对安全的威胁

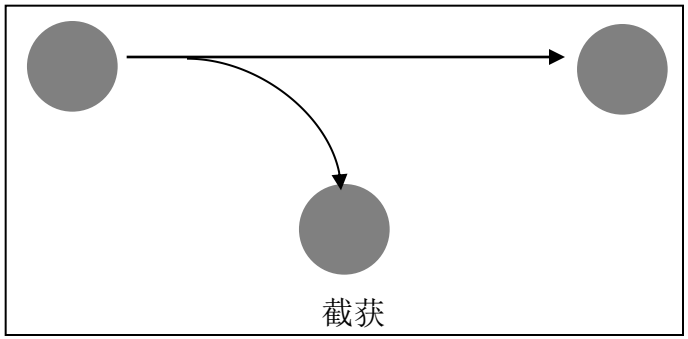
对安全的攻击

- 中断（阻断）：
  - 该系统的资产被破坏或变得不可利用或不能使用，这是对可用性的攻击。
    - 例子-包括部分硬件(如一个硬盘)的毁坏、通信连接的切断、某文件管理系统的失效、病毒的破坏。



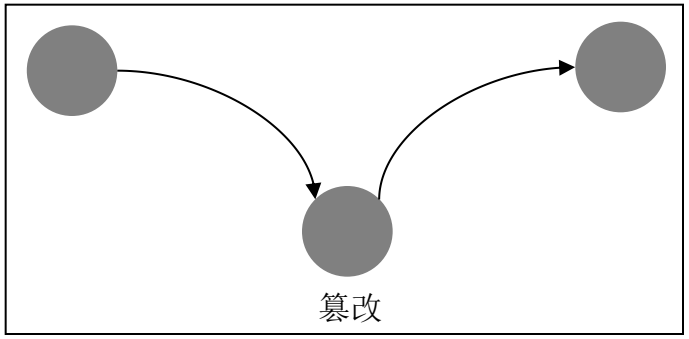
对安全的攻击

- 截获：
  - 一个未授权方获取了对某个资产的访问，这是对机密性攻击。该未授权方可以是一个人、一个程序或一台计算机。
    - 例如：在网络上搭线窃听以获取数据，违法复制文件或程序，等等。



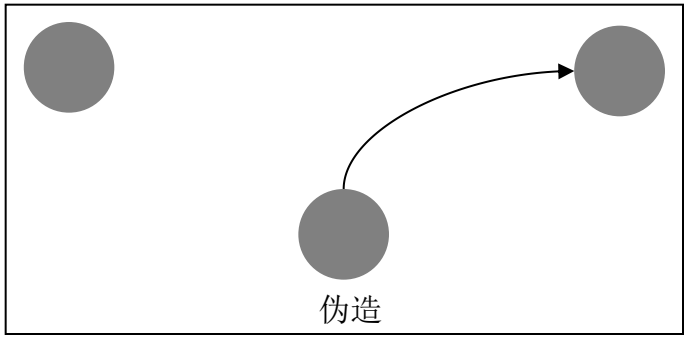
对安全的攻击

- 篡改：
  - 未授权方不仅获得了访问而且篡改了某些资产，这是对完整性的攻击。
    - 例如：改变数据文件的值，改变程序软件使得它的执行结果不同，篡改在网络中传输的消息的内容，等等。



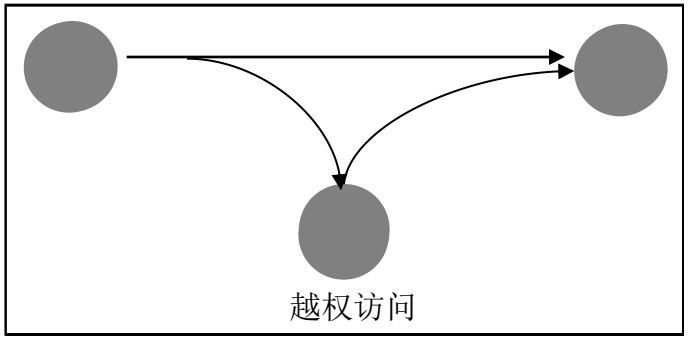
对安全的攻击

- 伪造：
  - 未授权方将伪造的对象插入系统，这是对不可抵赖性的攻击。
    - 例子包括在网络中插入伪造的消息或为文件增加记录。



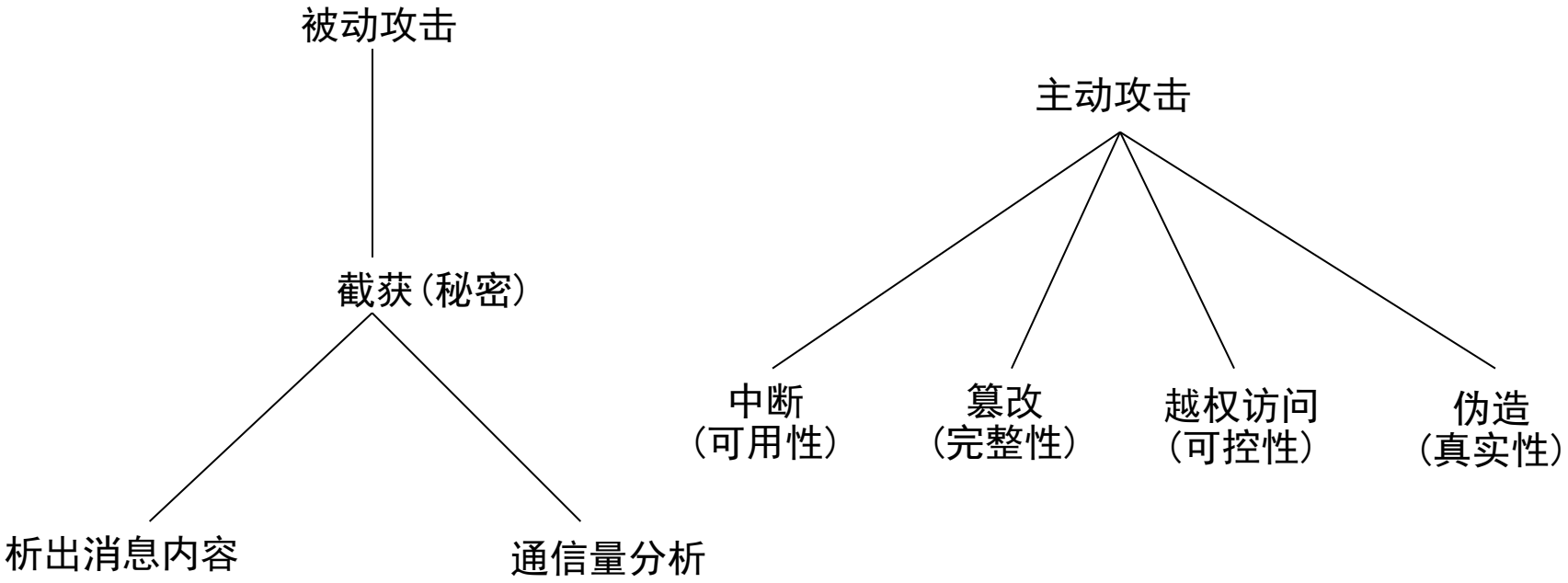
对安全的攻击

- 越权访问等：
  - 未授权方对系统、软件、信息的访问，这是对可控性的攻击。
    - 例子对网络连接的重放式攻击、非授权的操作等。
    - 缓冲区溢出攻击
    - 病毒等



安全攻击的分类

- 主动攻击与被动攻击



主动和被动网络安全威胁

被动攻击

- 本质是在存储、处理或传输中的偷听或监视、其目的是从中获得信息。
- 两类被动攻击
  - 析出消息内容
    - 电话交谈、电子邮件信息、存储或传送的文件
  - 通信量分析

被动攻击

- 通信量分析
  - 更为微妙
  - 假定用某种方法屏蔽了消息内容，即使对于获取了该信息也无法从消息中提取信息。
    - 屏蔽内容的常用技术是加密。
  - 对手也许还能观察这些消息的模式或信息通信量等。这些信息对猜测正在发生的通信的性质是有用的。
    - 测定通信主机的位置和标识
    - 观察被交换消息的频率和长度

被动攻击

- 难以检测
  - 因为它们并不会导致数据有任何改变。
- 防止这些攻击是可能的
  - 对付被动攻击的重点是防止而不是检测与消除。

主动攻击

- 涉及恶意软件的攻击、某些数据流的篡改或一个虚假流的产生。
  - 中断、伪造、伪装、篡改、欺骗劫持、缓冲区溢出、木马、入侵、资源占用
- 主动攻击中攻击者会直接或间接的与被攻击者接触

主动攻击

- 与被动攻击相反。
  - 被动攻击难以检测(发现)，但可采用措施防止此类攻击。
  - 完全防止主动攻击是相当困难的
    - 因为这需要在所有时间都能对所有主机、通信设施和路径进行物理保护。
  - 相反，防止主动攻击的办法是检测(发现)主动攻击、消除攻击的影响、并从主动攻击引起的任何破坏或时延中予以恢复。
    - 因为检测具有某种威慑效应，因此它也许能起到防止攻击的作用。

PDRR

- 一个最常用的安全模型就是PDRR模型。
  - Protection(防护)、
  - Detection(检测)、
  - Response(响应)、
  - Recovery(恢复)。
- 这四个部分组成了一个动态的信息安全周期。

PPDRR

- PDRR是在经典的P2DR模型基础上建立起来的。

PDRR模型

- 防御
  - 根据系统已知的所有的安全问题做出防御的措施。
  - 如访问控制、数据加密、打补丁等等。
- 检测
  - 攻击者如果穿过了防御系统，检测系统就会检测出来。
  - 检测的功能就是检测出入侵者的身份，包括攻击源、系统损失等。
- 响应
  - 一旦检测出入侵，响应系统开始响应包括事件处理和其他业务。
- 恢复
  - 在入侵事件发生后，把系统恢复到原来的状态。

防护

- 网络安全政策PDRR模型的最重要的部分。
  - 防护是预先阻止攻击可以发生的条件产生，让攻击者无法顺利地入侵。
  - 防护可以减少大多数的入侵事件。

防护

- 保证物理安全的防护：指的是物理设备、介质等的安全防护；数据与程序运行的安全防护
- 保证运行安全的防护：指的是网络管理的安全、网络传输的安全、操作系统的安全等的防护。
- 保证数据安全的防护：指的是数据本身的保密性、完整性和可用性的防护
  - 数据加密就是信息安全防护的重要技术。

防护

- 数据存储与恢复
- 数据加密
- 鉴别技术
- 安全通信技术

- 风险评估
- 访问控制
- 防火墙

防护

数据存储与恢复

- 防护数据在存储中遭到完整性、可用性的安全威胁。
- 通过增加数据本身的冗余度而达到对数据完整性进行保护的目的。

防护

- 数据加密
  - 数据加密技术是信息全防护的重要技术。
  - 加密技术防护数据在存储和传输过程中的保密性安全威胁。

防护

- 鉴别技术
  - 信息安全防护的重要技术，它和数据加密技术有很紧密的关系。
  - 鉴别技术用在安全通信中，对通信双方互相鉴别对方的身份以及传输的数据。
  - 鉴别技术防护数据通信的两个方面：通信双方的身份认证和传输数据的完整性。
  - 主要使用公开密钥加密算法的鉴别过程。

防护

- 鉴别技术
  - 数字签名是在电子文件上签名的技术，确保电子文件的完整性。
    - 数字签名首先使用消息摘要函数计算文件内容的摘要，再用签名者的私有密钥对摘要加密。
    - 在鉴别这个签名的时候，先对加密的摘要用签名者的公开密钥解密，然后跟原始摘要相比较。
    - 如果比较结果一致则数字签名是有效的，也就是说数据的完整性没有被破坏。

防护

- 鉴别技术
  - 身份认证需要每个实体(用户)登记一个数字证书。
    - 这个数字证书包含该实体的信息(如用户名、公开密钥)。
  - 这个证书应该有一个有权威的第三方签名，保证该证书的内容是有效的。
    - 数字证书类似于生活中的身份证。
  - 数字证书确保证书上的公开密钥是属于证书上的用户ID的，为了鉴别一个人的身份，只要用他的数字证书中的公开密钥去鉴别就可以了。

防护

- 鉴别技术
  - 公钥基础设施PKI就是一个管理数字证书的机构。
    - 其中包括发行、管理、回收数字证书。
  - PKI的核心是认证中心CA。
    - 它是证书认证链中有权威的机构，对发行的数字证书签名，并对数字证书上的信息的正确性负责。

防护

- 使用安全通信
  - 属于网络安全防护类型。使用安全通信可以防止数据在传输过程中的泄漏。
  - 安全通信的基本技术就是上述的数据加密和鉴别技术。

防护

- 使用安全通信
  - 点对点的安全通信
    - 可以应用在互联网模型的不同层次，得到不同的安全保护功能。

防护

- 使用安全通信
  - 网络层，即IP层
    - 在网络层实现点对点的安全通信的好处在于它与应用层的用户程序完全隔离
      - IPSec就是在网络层实现点对点的安全通信协议。
    - 限制就是不能做到终端用户到终端用户的安全通信，只能做到主机到主机的安全通信，甚至有的系统只能做到网络到网络的安全通信。
      - 虚拟专用网VPN就是使用IPsec技术建立的一个安全专用网。

防护

- 使用安全通信
  - 传输层，即TCP层。
    - 安全插座层SSL
    - SSL给应用层程序提供数据安全通信的服务。应用层程序只要支持SSL，就可以享受SSL的服务。
      - HTTPS就是通过SSL实现安全的HTTP协议。

防护

- 使用安全通信
  - 应用层
    - 即特殊应用程序之间建立点对点的安全通信。
    - 这种方式可以达到真正的终端用户到终端用户的数据安全通信。
      - PGP电子邮件加密和Secure Shell SSH就是典型的例子。

防护

- 风险评估
  - 风险评估属于网络运行安全防护类型。
  - 风险评估就是发现并修补系统和网络存在的安全漏洞。
    - 风险评估减少黑客攻击系统的条件，从而达到防护系统的目的。
    - 绝大多数入侵事件都是利用系统具有的安全漏洞进行攻击。

系统漏洞一般来自用操作系统本身以及各种应用软件。发现并修补系统的所有漏洞是防御系统的重要工作。

防护

风险评估——漏洞扫描

- 对于允许远程攻击的安全漏洞，可以使用网络漏洞扫描工具去发现。
  - 网络漏洞扫描工具一般从系统的外边去观察系统。
    - 其实，它扮演一个黑客的角色，只不过它不会破坏系统。

防护

风险评估——漏洞扫描

- 漏洞扫描工具首先扫描系统所开放的网络服务端口。
- 然后通过该端口进行连接，试探提供服务的软件类型和版本号。
  - 在这个时候，漏洞扫描工具有两种方法去判断该端口是否有漏洞：
  - 第一，根据版本号，在漏洞库中查处是否存在漏洞。
  - 第二，根据已知的漏洞特征，模拟一次攻击，如果攻击表示可能会成功就停止并认为是漏洞存在(要停止攻击模拟避免对系统的损害)。
- 漏洞挖掘：尝试、构造一些输入试探该端口的反应

防护

访问控制

- 访问控制限制某些用户对某些资源的操作。访问控制通过减少用户对资源的访问，从而减少资源被攻击的概率，达到防护系统的目的。
  - 例如只让可信的用户访问资源而不让其他用户访问资源，这样资源受到攻击的概率几乎很小。

防护

防火墙

- 在Internet /Intranet中的广泛使用防火墙已经成了不争的事实。
- 防火墙技术可以工作在网络层、传输层和应用层，完成不同粒度的网络层面的访问控制。
- 防火墙可以阻止大多数的攻击但是不是全部，有很多入侵事件通过防火墙所允许的规则进行攻击，例如通过80端口进行的攻击。

防护

防病毒软件与个人防火墙

- 防病毒软件和个人防火墙都是系统安全工具，属于系统安全防护类型。
  - 病毒就是计算机的一段可执行代码，一旦计算机被感染上病毒，这些可执行代码可以自动执行，破坏计算机系统。安装并经常更新防病毒软件对系统安全起防御作用。
- 防病毒软件根据病毒的特征，检查用户系统上是否有病毒。这个检查过程可以是定期检查，也可以是实时检查。
- 个人防火墙除了具有访问控制功能以外，还有病毒检测，甚至有入侵检测的功能，是网络安全防护中的一个重要发展方向。

检测(D)

- 防护系统的局限
  - 防护系统除掉入侵事件发生的条件，可以阻止大多数入侵事件的发生，但是它不能阻止所有的入侵。
    - 特别是那些利用新的系统缺陷、新的攻击手段的入侵。
  - 防护作用的层面受限
- 检测是信息安全的第二个安全屏障

检测(D)

- 检测和防护有根本性的区别。
  - 防护主要在设计层面增加系统的安全性能，或者通过外部安全系统弥补系统和网络的缺陷，从而消除攻击和入侵的条件。
  - 检测并不是从网络和系统的缺陷出发，而是面向攻击事件，分析攻击事件的特征并根据特征去检测发现攻击行为。

检测(D)

- 根据检测的对象，检测可分为两种：
  - 面向主机的检测
    - 基于主机上的系统日志，审计数据等信息。
  - 面向网络的检测
    - 一般侧重于网络流量分析。
- 根据检测所使用方法，检测可分为两种：
  - 误用检测(Misuse Detection)
  - 异常检测(Anomaly Detection)

- 基于主机的IDS的优点在于它不但能够检测本地入侵(Local Intrusion)，还可以检测远程入侵(Remote Intrusion)。而缺点则是它对操作系统依赖较大，检测的范围较小。
- 基于网络的入侵检测系统的优点则在于检测范围是整个网段，独立于主机的操作系统。

检测(D)

- 误用检测技术
  - 需要建立一个入侵规则库
    - 它对每一种入侵都形成一个规则描述，只要发生的事件符合于某个规则就被认为是入侵。
  - 好处在于它的误警率(False Alarm Rate)比较低
  - 缺点是查全率(Probability of Detection)完全依赖于入侵规则库的覆盖范围
  - 另外由于入侵规则库的建立和更新完全靠手工，且需要很深的网络安全知识和经验，所以维持一个准确完整的入侵规则库是一件十分困难的事情。

检测(D)

- 异常检测技术
  - 对正常事件的样本建立一个正常事件模型
  - 如果发生的事件偏离这个模型的程度超过一定的范围，就被认为是入侵。
  - 优点：具有一定的通有性
    - 由于事件模型是通过计算机对大量的样本进行分析统计而建立的，因此异常检测克服了一部分误用检测技术的缺点。
  - 缺点：相对误用检测来说误警率较高。

响应(R)

- 响应即已知一个攻击(入侵)事件发生之后，进行处理。
- 计算机紧急响应小组
  - 在一个大规模的网络中，响应这个工作都是有这个特殊部门负责。
  - 世界上第一个计算机紧急响应小组CERT，位于关围CMU大学的软件研究所(SEI)，于1989建立，是世界上最著名的计算机响应小组。
  - 从CERT建立之后，世界各国以及各机构的网络也纷纷建立自己的计算机紧急响应小组。
  - 我国第一个计算机紧急响应小组CCERT，于1999年建立，主要服务于中国教育和科研网

响应(R)

- 响应的主要工作分为两部分。
  - 事发前，主要包括咨询、培训和技术支持。
  - 事发时(应急响应)，当安全事件发生时采取应对措施

恢复(R)

- 即事件发生之后，把系统恢复到原来的状态，或者比原来更安全的状念。
- 分为两个方面
  - 系统恢复
  - 信息恢复

恢复(R)

- 系统恢复指的是修补该事件所利用的系统缺陷，不让黑客再次利用这样的缺陷入侵。
- 一般系统恢复包括系统升级、软件升级和补丁等。系统恢复的另一个重要工作是除去后门。
- 系统恢复是根据检测和响应环节提供有关事件的资料进行的。
- 一般来说，黑客在第一次入侵的时候都是利用系统的缺陷。在第一次成功入侵之后，黑客就在系统打开一些后门，如安装一个特洛伊木马。所以，尽管系统缺陷已经被打补丁，黑客下一次还可以通过后门进入系统。

恢复(R)

- 信息恢复指的是恢复丢失的数据。
  - 数据丢失的原因可能是由于黑容入侵造成，也可以是由于系统故障、自然灾难等原因造成。
  - 信息恢复就是从备份和归档的数据恢复原来数据。
    - 信息恢复过程跟数据备份过程有很大的关系。
    - 数据备份做得是否充分对信息恢复有很大的影响。
  - 信息恢复过程的一个特点是有优先级别。
    - 直接影响日常生活和工作的信息必须先恢复，这样可以提高信息恢复的效率。

相关的技术

- 风险分析、安全评估
  - 如何评估？如何验证？规模？
- 漏洞扫描技术
  - 漏洞挖掘技术
  - 基于关联的弱点分析技术
  - 基于用户权限提升的风险等级量化技术
- 访问控制技术
  - 多级、多态访问控制技术
- 病毒防护，侧重于网络制导、主动遏制。
  - 侧重网络病毒，强调大规模、主动、遏制

相关的技术

- 隔离技术
  - 网闸
  - 单向路径
- 拒绝服务攻击的防护
  - DDoS是个致命的问题，需要有解决办法
- 预警技术
  - 大规模网络异常检测

相关的技术

- 分布式入侵检测
  - 入侵检测信息交换协议
  - IDS的自适应信息交换与防攻击技术
- 木马检测技术
  - 守护进程存在状态的审计
  - 守护进程激活条件的审计
  - 控制端发现技术

相关的技术

- 响应技术
  - 快速判定、事件隔离、证据保全
    - 紧急传感器的布放，传感器高存活，网络定位
  - 蜜罐技术
    - 漏洞再现及状态模拟应答技术
    - 沙盒技术，诱捕攻击行为
    - 动态身份替换
    - 异常负载的转配

相关的技术

- 基于Key-Value的数据存储技术
  - 构建综合备份中心IBC（Internet Backup Center）
  - 基于内存存储的远程存储技术
  - 数据消冗技术
- 容侵（intrusion-tolerant）技术
  - 受到入侵时甩掉被攻击部分。
  - 防故障污染。
- 生存（容忍）技术
  - 可降级运行，可维持最小运行体系

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件安全开发

- 3.1 软件安全的指导原则
- 3.2 软件安全需求
- 3.3 软件安全编码基本方法与技术
- 3.4 软件安全设计

软件安全设计原则

- 原则1：确保最薄弱环节安全
  - 木桶原理
  - 发现与检测薄弱环节
  - 互不重叠的安全功能部件
- 原则2：深度防御
  - 多层防御机制
  - 冗余安全措施
- 原则3：失效安全
  - 故障和失效不可避免
  - 检测与恢复

软件安全的指导原则

- 原则4：遵循最小特权的原则
  - 仅仅授予为了执行某种操作而必须的最小访问权限
  - 只分配访问所必须的最小时间
  - 权限控制
- 原则5：分割
  - 隔离有安全特权的代码同时，把系统分割成尽可能小的单元以将损害降到最低
  - 防止全有或全无
- 原则6：简化
  - 尽可能的简单易行
  - 尽可能考虑组件复用

软件安全的指导原则

- 原则7：提升隐私权
  - 个人信息管理
  - 信息加密
- 原则8：隐藏秘密很困难
  - 保守秘密很困难，它几乎总是安全风险的来源
  - 不要轻信

软件安全编码原则

- （1）规范编码
  - 统一、标准的编码规范
  - 可读性、易维护性、运行效率
- （2）代码简洁
  - 短小精悍
  - 控制函数内代码量
- （3）处理警告
  - 启用编译器的警告和错误
- （4）验证输入
  - 验证不可信数据源的输入
  - 命令行参数、网络接口、环境变量、用户控制的文件

- （5）净化数据
  - 检查程序组件中要传递的数据
  - 恶意数据、不必要数据处理干净
- （6）最少反馈
  - 尽量将最少的数据反馈到用户
  - 运行日志另当别论
- （7）检查返回
  - 函数返回值进行检查
  - 正确返回、错误返回的相关信息要多
- （8）加强检查
  - 代码的人工检查和工具检查
- （9）安全编译

软件安全需求

- 保密性需求（机密性需求）
- 完整性需求
- 可用性需求
- 认证与鉴别需求
- 可审性需求

软件安全需求

- 保密性需求（机密性需求）
  - 保密性需求分析
    - 传输过程
    - 处理过程
    - 存储过程
  - 保密机制
    - 公开加密机制
    - 隐蔽加密机制
    - 屏蔽机制

软件安全需求

- 完整性需求
  - 完整性需求分析
    - 可靠性保证，防止未经授权的修改
    - 系统或数据的完备性和一致性
  - 完整性机制
    - 输入验证机制
    - 校验检查
    - 散列函数

软件安全需求

- 可用性需求
  - 可用性需求分析
    - 恢复
    - RTO（Recovery Time Object）与RPO （ Recovery Point Object ）
    - 负载均衡
  - 可用性机制
    - 冗余机制
    - 恢复机制
    - 监测与迁移

软件安全需求

- 认证与鉴别需求
  - 认证需求分析
    - 用户身份鉴别
    - 信息鉴别
  - 认证与鉴别机制
    - 身份认证
    - 抗抵赖机制
    - 授权机制

软件安全需求

- 可审性（可控性）需求
  - 可审性需求分析
    - 主体是谁
    - 动作是什么
    - 动作执行对象是那一个
    - 动作执行时间是什么
  - 可审性机制
    - 先验知识
    - 时间戳
    - 临时值

软件安全编码基本方法与技术

- 安全编码基础方法
- Web应用安全编程
- 数据安全编程

安全编码基础方法

- 防范缓冲区溢出攻击
- 防范整数溢出
- 处理竞争条件
- 正确处理异常
- 防范参数安全问题

防范缓冲区溢出攻击

- 防范缓冲区溢出攻击原则
  - 避免使用危险函数
  - 检查数据长度
  - 使用安全函数或函数库
  - 其他技术（外部使用的技术）

1、不带长度参数的串操作函数，strcpy字符串拷贝、strncpy(指定长度拷贝、strcpy_s

2、输入数据的最大长度比较

3、有一些库比较安全 libmib，libsafe。由于Libsafe的实现依赖于Linux系统为动态链接库所提供的预载机制，因此对于使用静态链接库的具有缓冲区溢出漏洞的程序Libsafe也就无能为力了。

建议禁用的API	替代的StrSafe函数	替代的Safe CRT函数
有关字符串拷贝的API		
strcpy, wcscpy, _tcscpy, _mbscpy, StrCpy, StrCpyA, StrCpyW, lstrcpy, lstrcpyA, lstrcpyW, strcpyA, strcpyW, _tccpy, _mbccpy	StringCchCopy, StringCbCopy,StringCchCopyEx, StringCbCopyEx	strcpy_s
有关字符串合并的API		
strcat, wcscat, _tcscat, _mbscat, StrCat, StrCatA, StrCatW, lstrcat, lstrcatA, lstrcatW, StrCatBuffW, StrCatBuff, StrCatBuffA, StrCatChainW, strcatA, strcatW, _tccat, _mbccat	StringCchCat, StringCbCat,StringCchCatEx, StringCbCatEx	strcat_s
有关sprintf的API		
wnsprintf, wnsprintfA, wnsprintfW, sprintfW, sprintfA, wsprintf, wsprintfW, wsprintfA, sprintf, swprintf, _stprintf	StringCchPrintf, StringCbPrintf,StringCchPrintfEx, StringCbPrintfEx	_snprintf_s

防范缓冲区溢出攻击的技术

- 栈保护技术
  - 栈溢出检测
    - 函数开始执行的时候会先往栈里插入cookie信息，该cookie往往放置在ebp/rbp的正上方，当函数真正返回的时候会验证cookie信息是否合法，如果不合法就停止程序运行，cookie值的随机性
  - 返回地址保护
    - 在函数调用后插入获取返回地址的语句，在函数结束返回前检查返回地址并覆写

调用一个函数后的堆栈

防范缓冲区溢出攻击的技术

- 数据执行保护机制（DEP）
  - 溢出攻击的根源在于计算机对数据和代码没有明确的区分这一缺陷
  - DEP将数据所在的内存页标识为不可执行，当程序尝试在数据页执行指令时，就会抛出异常。主要是为了阻止数据页（如默认的堆页、各种堆栈页以及内存池页）执行代码。

- DEP - 数据执行保护的缩写，Data Execution Prevention。
- 数据执行保护 (DEP) 是一套软硬件技术，能够在内存上执行额外检查以帮助防止在系统上运行恶意代码。在 Microsoft Windows XP Service Pack 2、Microsoft Windows Server 2003 Service Pack 1 、Microsoft Windows XP Tablet PC Edition 2005 、 Microsoft Windows Vista 和 Microsoft windows7 中，由硬件和 软件一起强制实施 DEP。
- 是可以帮助防止数据页执行代码。通常情况下，不从默认堆和 堆栈执行代码。硬件实施 DEP 检测从这些位置运行的代码，并在发现执行情况时引发异常。软件实施 DEP 可帮助阻止恶意代码利用 Windows 中的 异常处理机制进行破坏。
- 硬件实施 DEP 是某些 DEP 兼容处理器的功能，可以防止在已标记为数据存储区的内存区域中执行代码。 此功能也称为非执行和执行保护。 Windows XP SP2 还包括 软件实施 DEP，其目的在于减少利用 Windows 中的例外处理机制的情况。
- 与 防病毒程序不同，硬件和 软件实施 DEP 技术的目的并不是防止在计算机上安装有害程序。 而是监视已安装程序，帮助确定它们是否正在安全地使用系统内存。 为监视程序，硬件实施 DEP 将跟踪已指定为“不可执行”的内存区域。 如果已将内存指定为“不可执行”，但是某个程序试图通过内存执行代码，Windows 将关闭该程序以防止恶意代码。 无论代码是不是恶意，都会执行此操作。

- 在探讨DEP的原理前，我们先区分两个容易引起混淆的概念：软件DEP（Software DEP）和硬件DEP（Hardware-enforced DEP）。

- 软件DEP，并不是真正意义上的DEP。简单的说，它的原理是检查异常处理是否安全（SEH-Safe Exception Handling）。它是完全通过软件支持的一种安全特性。在以后的安全编码实践中我们会专门讨论SEH。

- 硬件DEP，则是需要CPU提供支持的，同软件DEP相比，硬件DEP提供的保护更为全面。以后我们提到的DEP，都是指硬件DEP。在AMD64位CPU上，在页面表（page table）中的页面信息加了一个特殊的位，NX位（No eXecute）。如果NX位为0，这个页面上可以执行指令。如果NX位为1，这个页面上不允许执行指令。如果试图执行指令的话，就会产生异常。Intel在它的CPU上也提供了类似的支持，称为XD 位（ eXecute Disable），其原理和AMD的NX是完全一样的。

- 为了应对DEP技术，一种攻击方法称之为ret2libc，即return-to-libc，返回到系统库函数执行 的攻击方法。构建栈结构首先在执行栈结构中，将EIP填充为system函数的地址，然后函数返回时，跳到system函数中执行。在执行刚进入system函数时，此时esp指向的地址为前EIP高4字节的地址，然后在system函数，从它的视角来看，esp指向的是它的返回地址（EIP)，而esp + 8就是它的函数，整个结构如下图所示：
- Ret2libc 攻击除了使用 system() 函数以外，还会使用 exec(), mprotect() 函数。mprotect() 函数可以改变 页表的属性，从而使 DEP 攻击失效。
- 在上面对Ret2libc攻击方式的介绍中，我们看到最为关键的一点是攻击者事先预知了特定函数，如system或VirtualProtect的入口地址。在Windows XP或Windows 2000上，这些函数的入口地址是固定的，即攻击者事先可以确定的。在Windows Vista中引入了ASLR安全特性。它的原理就是在当一个应用程序或动态链接库，如kernel32.dll，被加载时，如果其选择了被ASLR保护，那么系统就会将其加载的基址随机设定。这样，攻击者就无法事先预知动态库，如kernel32.dll的基址，也就无法事先确定特定函数，如VirtualProtect，的入口地址了。

ASLR是需要和DEP配合使用的。如果CPU不提供对于DEP的硬件支持，或者应用程序没有选择被DEP保护的话，恶意代码一旦可以执行，就可以通过程序进程表结构来获得特定DLL的加载基址
ASLR需要操作系统和程序自身的双重支持。

防范缓冲区溢出攻击的技术

- 地址空间布局随机化（ASLR）
  - ASLR通过加载程序的时候不再使用固定的加载基址，从而干扰shellcode定位
  - 映像随机化（dynamicbase）
    - PE文件映射到内存时，对其加载的虚拟地址进行随机化处理，这个地址是在系统启动时确定的，系统重启后这个地址会有变化
    - 动态链接库加载地址的随机化
  - 堆栈随机化
    - 程序运行时随机选择堆栈的基址，与映像随机化不同的是堆栈的基址不是在系统启动时确定的，而是在程序打开时确定的，也就是说同一个程序任意两次运行时的堆栈基址都不相同

防范缓冲区溢出攻击的技术

- 安全结构化异常处理（SafeSEH）
  - 思想：在程序调用异常处理函数前，对要调用的异常处理函数进行一系列有效性检验，当发现处理函数不可靠时将终止异常处理函数的调用
  - 编译器启用SafeSEH保护选项后，编译器在编译程序时将程序所有的异常处理函数地址提取出来，编入一张安全SEH表，并将这张表放入程序的映像里，当程序调用异常处理函数时，会将函数地址与安全SEH表进行匹配，检查调用的异常处理函数是否位于安全SEH表中。

防范缓冲区溢出攻击的技术

- 堆保护机制
  - PEB random：PEB的随机化给堆攻击增加了难度。
  - Safe Unlink：拆卸双向链表时，对指针指向的内容是否是相应的堆块做安全检查。
  - heap cookie：与栈中的Security Cookie类似，在堆中也引入cookie，用于检测堆溢出的发生，cookie被布置在堆首部，占一个字节大小。
  - 元数据加密：块首中的一些重要数据在保存时会与一个4字节的随机数进行异或运算，在使用这些数据时，需要在进行一次异或运算来还原。

安全编码基础方法

- 防范缓冲区溢出攻击
- 防范整数溢出
- 处理竞争条件
- 正确处理异常
- 防范参数安全问题

防范整数溢出

- 整数溢出
  - 整数大于小于其范围时，会“回绕”而产生整数溢出，例如超出带符号32位整数表示的上限值，即会变为负数。
  - 检验整数输入是否位于上下限范围内。

处理竞争条件

- 竞争
  - 线程间以及进程间共享缓冲区时，如多个线程和进程同时读写数据时，如代码编写不当，可能会造成每次程序运行结果不同。
  - 共享资源加锁：同步与互斥

正确处理异常

- 软件异常
  - 软件因为某个条件不满足而中断正常处理流程，即软件异常。软件程序当提供异常处理代码，使其能检测出各种异常，并处理可能的运行路径。不安全的异常处理或错误则可能造成信息泄露，甚至系统宕机、数据丢失。
  - 建议：
    - 不要在错误响应中泄露敏感信息：如系统详细信息、会话标识符、账号信息等
    - 避免显示调试或者堆栈跟踪信息
    - 使用通用的错误消息、定制的错误页面
    - 错误发生时，适当清空分配的内存

防范参数安全问题

- 交互数据软件异常
  - 软件和用户之间有输入操作，程序和环境变量、用户之间可能需要经常交互数据，对交互数据的检查是必要的。
  - 环境变量
    - 环境变量不一定是可靠的，也可能有长度过长、内容被篡改、格式不符、植入恶意字符等。加强变量值的检查、限制使用权限
  - 文件：软件常常对文件操作，对文件名和内容检查。
    - 不要信任可能被其他人设置的文件名，Linux允许使用任意字符序列成为文件名，对目录和文件名仔细检查，例如&-等特殊字符
    - 不可信环境下文件内容也不可信任
    - 小心从当前目录获得配置信息
    - 临时文件的安全性
  - 命令行：确定命令行数据的可信，检查参数的格式、长度、内容

软件安全编码基本方法与技术

- 安全编码基础方法
- Web应用安全编程
- 数据安全编程

Web应用安全编程

- 防范SQL注入攻击
- 防范跨站脚本攻击
- 其他Web攻击防范

防范SQL注入攻击

- SQL注入攻击的两个关键条件
  - 用户能够控制输入
  - 程序执行的代码拼接了用户输入数据
- 防范
  - 输入验证
    - 使用结构化的机制分离数据和代码，输入数据参数化
    - 对输入进行校验，输入变量尽可能滤掉可疑字符
    - 输入数据转义，如将输入中的单引号转换成双引号
  - 实施并生成命令清单（指令白名单）
  - 使用安全存储过程
  - 错误信息处理，最小的出错信息，或所有出错信息指向同一个出错页面

- 在web、数据库服务器上删除所有不必要的函数和程序，以及能允许用户运行的扩展程序；
- 最小授权，基于视图的最小授权，对表的查询和程序的访问控制映射到具体的用户集合，数据库用户需被授予最小权限，具体到数据库读、数据库写，而不是全部权限
- 对数据库中敏感重要的数据不以明文显示，加密
- 查询审计

防范跨站脚本攻击

- 跨站脚本攻击
  - 网站的漏洞
  - 网站中被放入一段可执行的代码
  - 用户访问网站后，代码被执行
- 防范
  - 网站开发者角度
    - 任意输入数据的验证，黑名单字符集过滤等
    - 任意输出数据的验证：对输出的数据进行编码，以代替用户自己的数据格式。数据转义、字符替换

- 用户角度
  - 不访问不受信任的网站
  - 设置关闭JavaScript

其他Web攻击防范

- 避免URL操作攻击
  - URL：协议、主机名、资源名称（路径表示）、其他信息（如查询字符串等）
  - URL操作攻击：猜测资源的存放地址、资源名称等
  - 防范：
    - 对每一个只有登录成功才能访问的页面进行session检查
    - 限制用户访问未授权资源，可考虑在每个资源访问时携带用户信息

软件安全编码基本方法与技术

- 安全编码基础方法
- Web应用安全编程
- 数据安全编程

数据安全编程

- 常用密码算法和接口库
  - 密码算法选择
  - 常用的密码算法实现函数库
  - 我国商用密码算法
- 随机数生成
- 密钥的生成和使用
- 哈希计算

面向绕过DPI检测的攻击

背景

- **在计算机顶级会议上，出现很多针对各国国家防火墙的测量和网络渗透的研究，其中一部分是专门针对我国的国家防火墙进行网络渗透。**

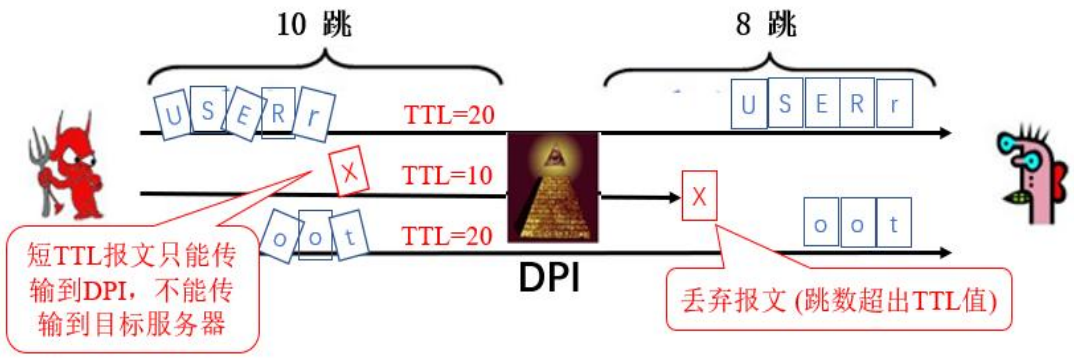
网络渗透攻击

- **TCP状态机攻击：攻击者通过伪造报文干扰深度报文检测系统的TCP状态机，误导其丢弃攻击流量。**
- **多路径传输攻击：攻击者通过将攻击数据分片，利用MPTCP协议将每个分片经过单独网络向目标传输，由于深度报文检测缺少分布式检测MPTCP协议，导致对攻击数据分片出现误判。**

TCP状态机攻击模型

伪造攻击数据

- 测量客户端与服务端的网络距离，设置TTL字段小于测量距离
- 数据内容填充垃圾数据，修改TCP的发送序号或者发包顺序。



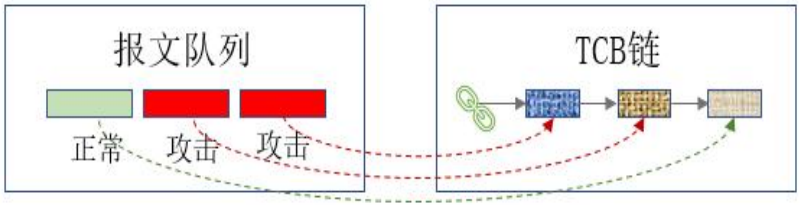
TCP状态机攻击检测

TCP状态机攻击检测

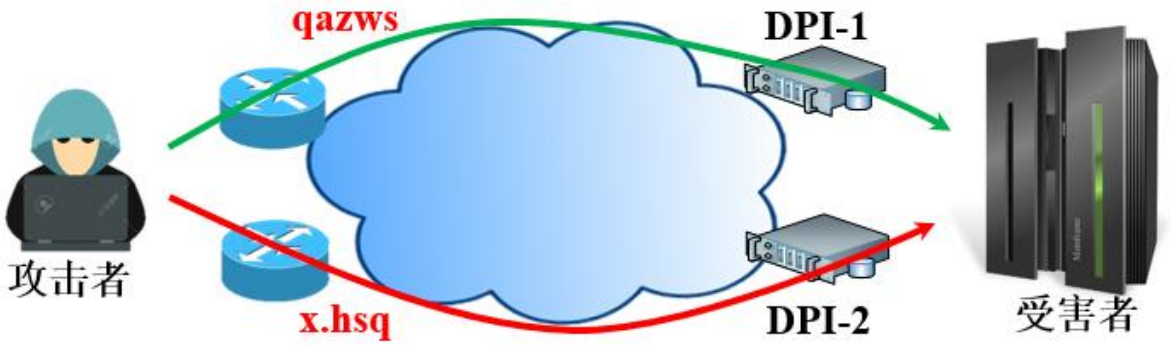
- **伪造TTL字段的检测**

服务识别 (ip 和 port 组成的哈希值)	TTL 阈值

- **伪造TCP发送序号和数据的检测**



多路径传输攻击



作为一种现成的工具，多路径TCP（MPTCP）为攻击者发起多路径传输攻击提供了支持。在多网卡设备中，攻击者可以将敏感内容切分成多个子串，然后从不同的网卡发送出去。以图5-2为例，假设攻击者的计算机（客户端）有两个网卡连接到公共网络，而受害者的（服务器）企业有两个网关，每个网关都部署了一个DPI服务器来监视通过网关的所有流量。

攻击者首先通过两个网卡与目标服务器的两个子连接建立MPTCP连接，使得两个子连接产生的流量通过不同的DPI。此时，攻击者通过发送内容来执行QAZ Worm客户端登录“qazwsx.hsq”. 为了逃避检测，内容被随机分成两个子串（例如，“qazws”，“x.hsq”），这两个子串由不同的子连接进行发送。由于每个DPI独立运行，只接收一段内容，因此无法命中规则库中的特征，检测失败。然而，目标服务器可以接收所有的片段，并通过MPTCP协议成功地恢复内容。

多路径传输攻击检测

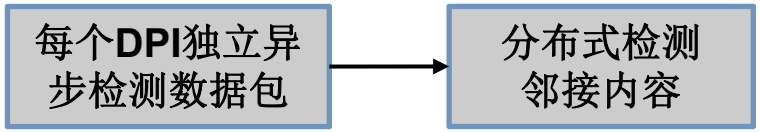
哈爾濱工業大學 HARBIN INSTITUTE OF TECHNOLOGY

问题定义：多路径传输攻击检测问题抽象为文本切割并行匹配

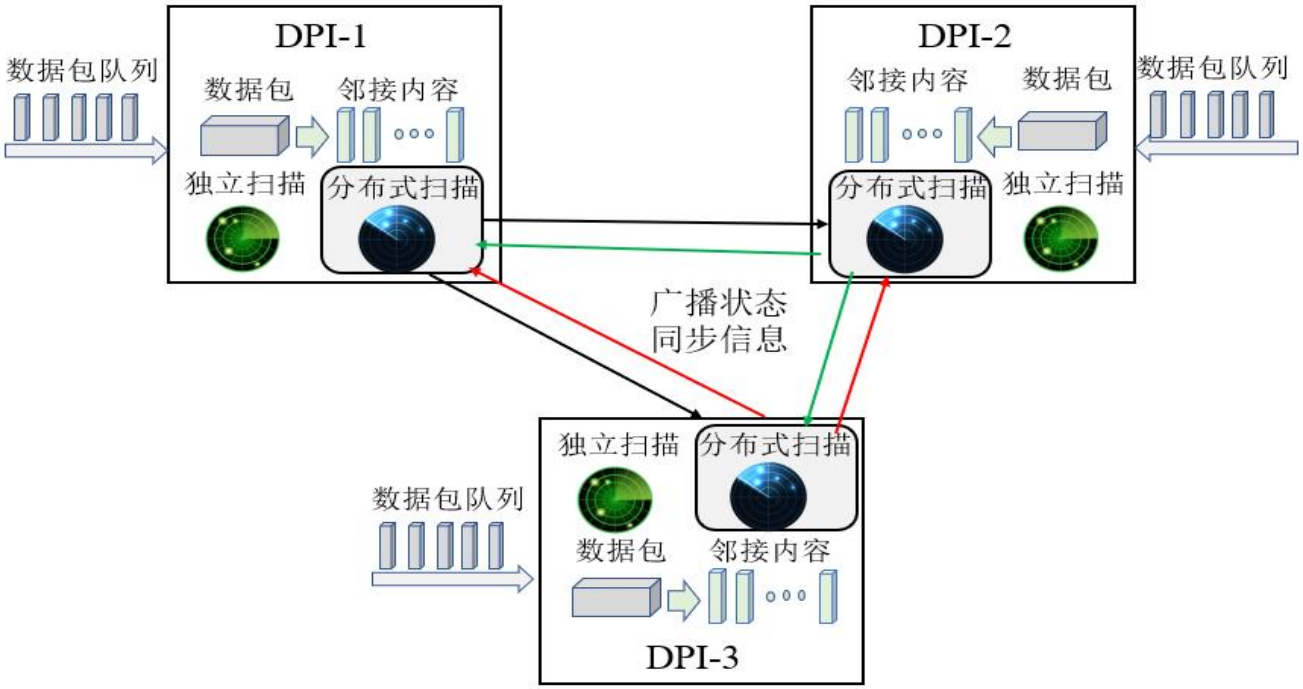
- **给定文本字符串*P*, 随机划分为n个子串，得到** $P=\{p_1,p_2,...,p_n\}$ **。假设存在q个DPI，找到所有算法中扫描时间最小的。**

$$min\ max_{i=1}^{+\infty}\{T_1^i,T_2^i,...,T_q^i\}$$

解决方案：分布式异步并行检测算法(DADDA)



多路径传输攻击检测


DPI-1
数据包队列
数据包
邻接内容
独立扫描
分布式扫描
DPI-2
邻接内容
数据包
数据包队列
分布式扫描
独立扫描
广播状态
同步信息
独立扫描
分布式扫描
数据包队列
数据包
邻接内容
DPI-3

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件安全开发

- 3.1 软件安全的指导原则
- 3.2 软件安全需求
- 3.3 软件安全编码基本方法与技术
- 3.4 软件安全设计

面向网络攻击的软件安全设计

- 可用性安全
- 机密性安全
- 完整性安全
- 可控性安全
- 可审性安全

一定的概率并不是指全部，所以，可以允许有错误，因此，入侵容忍还有对纠错理论的联想：即利用纠错码可以在一个错误百出、但有信道容量的信道中准确无误地传输数据，网络系统就这样在错误中“生存”下来的，这就是我们说的入侵容忍体系，

软件安全设计

- 入侵容忍体系
  - 攻击行为在一定的概率下是可预知的，系统在一定的概率下能够正确完成基本的功能。
- 入侵容忍实现方式
  - 攻击响应：不需要重新设计系统，可通过高效的检测系统发现异常，利用资源配置系统调整系统资源，并对对错误进行修补（修补系统）
  - 攻击遮蔽：需要重新设计整个系统，并通过冗余、容错技术、门槛密码学等技术，完成允许存在少数节点失效\作恶（消息可能被伪造）场景下的一致性达成问题

- 在线技术
  - 用于系统运行阶段，即通过对系统的某种改造，使系统具有以下能力：当系统受到恶意攻击或发生局部故障影响系统可用性时，系统能够自动检测这种状态，并在一定程度上具有自动恢复、维持、延续关键性服务的能力
- 离线技术
  - 用于系统设计阶段，即通过优化系统体系结构、控制参数等，使形成的系统内在的生存能力得到提升

失效的概念

- 失效模型
  - 失效是由攻击、入侵、故障等事件引发
  - fail-stop失效：系统失效后，即停止工作。
  - Byzantine(拜占庭)失效：系统失效后，处于一种不确定的工作状态。
- 与错误的区别
  - 错误：逻辑上把硬件(软件)的实际输出与理论输出不一致称为错误，把导致错误的原因称为故障。
  - 容错追求的是细微力度的故障替换，不允许有任何事实上的损失。（部件级）
  - 入侵容忍针对入侵及较大规模不可抗拒的故障，要求任务的持续性，允许有一定程度的损失。（节点级）

软件安全设计

- 两层含义
  - 数据可用，确保业务不因为失效而带来重大损失 。（减少损失）
  - 软件系统可用，确保业务不因为失效事件的发生而中断，但允许有一定程度的间断及损失。（但要保证业务的可持续性 ）
- 实现中可分为两个层次：
  - 数据可用：建立可靠的数据存储与恢复机制和系统
  - 应用可用：在数据可用的基础上,保持业务的持续运行。
- 应用可用是高层次的，但数据可用是基础

- 设计上包含两个方面：
  - 数据存储系统提供应用系统的数据后援，确保在任意情况下数据具有完整的恢复能力。
  - 高可用系统确保应用系统在多机环境下具有抗御单点或多点失效的能力，一旦系统发生影响可用性的事件(如软件系统故障、攻击、入侵、网络故障等)，高可用系统可以在最短的时间迅速确保系统的应用继续运行。

相关指标

- Unplanned Downtime：
  - 系统失效发生后，不可预知的停机时间，一般靠经验来估计。
- DOI(Degraded operation interval)：
  - 系统失效发生后，系统失去部分功能部分而降级使用的时间
    - 功能降级
    - 抗风险降级
- Planned Downtime ：
  - 系统失效发生后，系统因恢复数据或系统切换而导致的计划中系统停止服务的时间。

相关指标

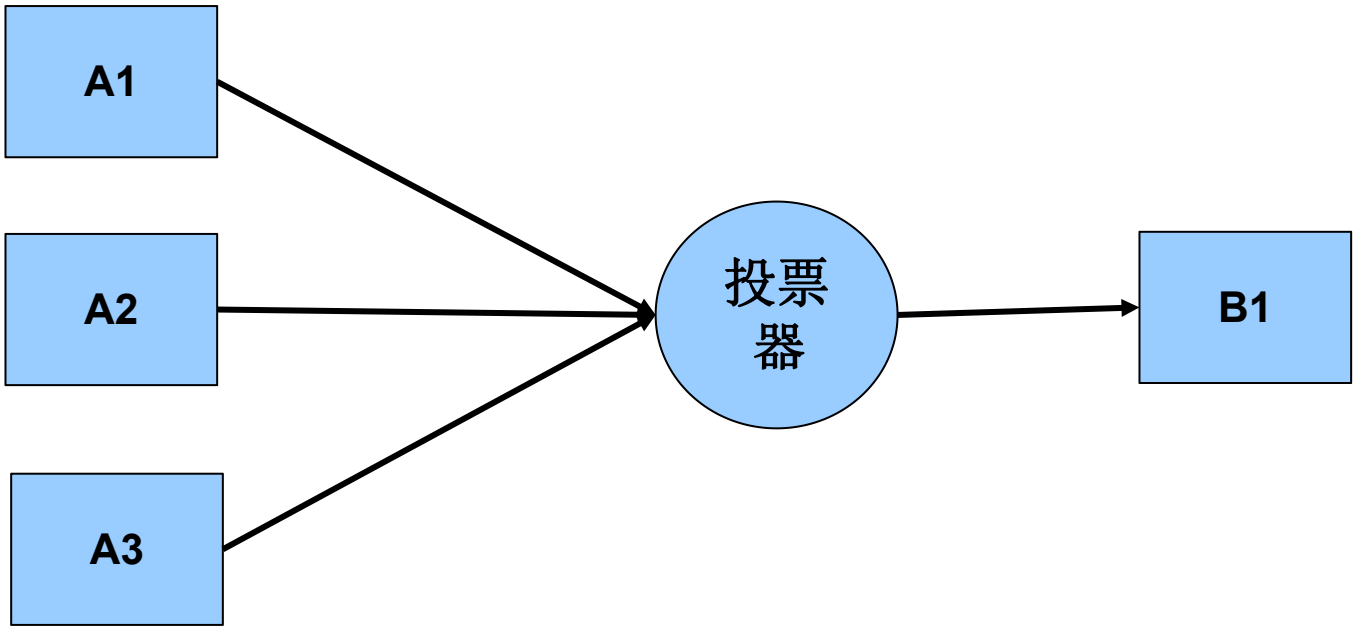
- *RTO(Recovery time objective)：*
  - 指系统失效发生后到系统恢复正常状态的时间目标
  - 包括对Unplanned Downtime、Planned downtime和DOI的分别度量
- *RPO(Recovery point objective)：*
  - 指系统失效发生后，系统恢复到某个具有一致性的点的目标
  - 包括对数据损失程度的度量，即发生失效造成的数据丢失量

系统可用设计思路

- Fail-stop失效
  - 冗余资源：f+1，f为可能失效的最大值
- Byzantine失效
  - 冗余资源：2f+1，f为可能失效的最大值
  - 投票机制
- 如果需要考虑网络也可能同时失效的情况呢？
  - Fail-stop失效：2f+1
  - Byzantine失效：3f+1，拜占庭容错(BFT)

Byzantine(拜占庭)失效处理机制

- 投票机制

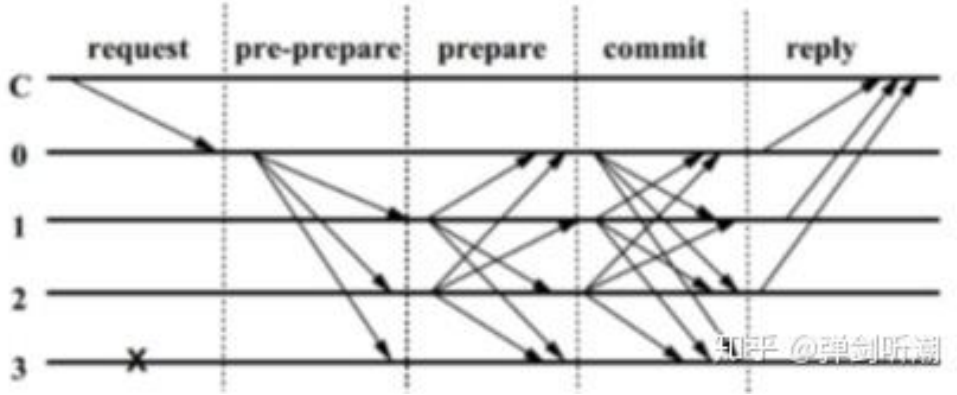


Byzantine(拜占庭)失效处理机制

- 拜占庭容错共识算法
  - 分布式系统中能够处理拜占庭错误的共识算法
  - 拜占庭错误恶意节点：它为阻挠真实信息的传递以及有效一致的达成，会向各个节点发送前后不一致的信息
- 拜占庭容错算法
  - 实用拜占庭容错算法(PBFT)

实用拜占庭容错算法(PBFT)

- PBFT
  - 主节点和从节点：主节点仅有1个，主节点负责将客户端的请求排序；从节点按照主节点提供的顺序执行请求。
  - request、pre-prepare、prepare、commit、reply



实用拜占庭容错算法(PBFT)

- PBFT
  - PBFT是一种状态机副本复制算法。在某个视图中，一个副本作为主节点，其它的副本节点作为从节点。主节点通过随机算法选出，用来负责与提案的客户端通信。
  - 主节点选出后，客户端发送提案给主节点，主节点将客户端请求进行编号，然后发送预准备消息给所有的从节点。
  - 每一个从节点在收到来自主节点的预准备消息之后，都要检查消息的正确性，然后发送准备消息给除了自己以外的其他所有节点。同时它也会收到其他各节点发来的准备消息。在收到消息后，各节点对其他节点的准备消息进行验证，如果正确就将准备消息写入消息日志，集齐规定数量的准备消息之后，它就进入准备状态。
  - 节点进入准备状态后，在全网范围内广播commit消息，当各节点集齐规定数量个验证过的commit消息后，就表示请求处理完毕，当前网络中的大部分节点已经达成共识，于是发送处理结果给客户端，应答客户端的请求。

实用拜占庭容错算法(PBFT)

- PBFT
- 主节点作恶
  - 如果主节点作恶，它可能会给不同的请求编上相同的序号，或者不去分配序号，或者让相邻的序号不连续。从节点应主动检查这些序号的合法性。如果主节点掉线或者作恶不广播客户端的请求，客户端设置超时机制，超时的话，向所有副本节点广播请求消息。副本节点检测出主节点作恶或者下线，发起View Change协议。
- 从节点作恶
  - 少量从节点作恶，最终客户端仍能收到f+1个正确的应答。

Fail-stop失效可用性实现技术

- 多样性冗余
- 自适应方式
- 失效检测
- 隔离和修复
- 迁移

多样性冗余

- 冗余是硬件容错、软件容错等领域广泛运用的设计方法，是提高系统可用的核心思想
  - 资源冗余
  - 方法冗余
  - 时间冗余
  - 信息冗余

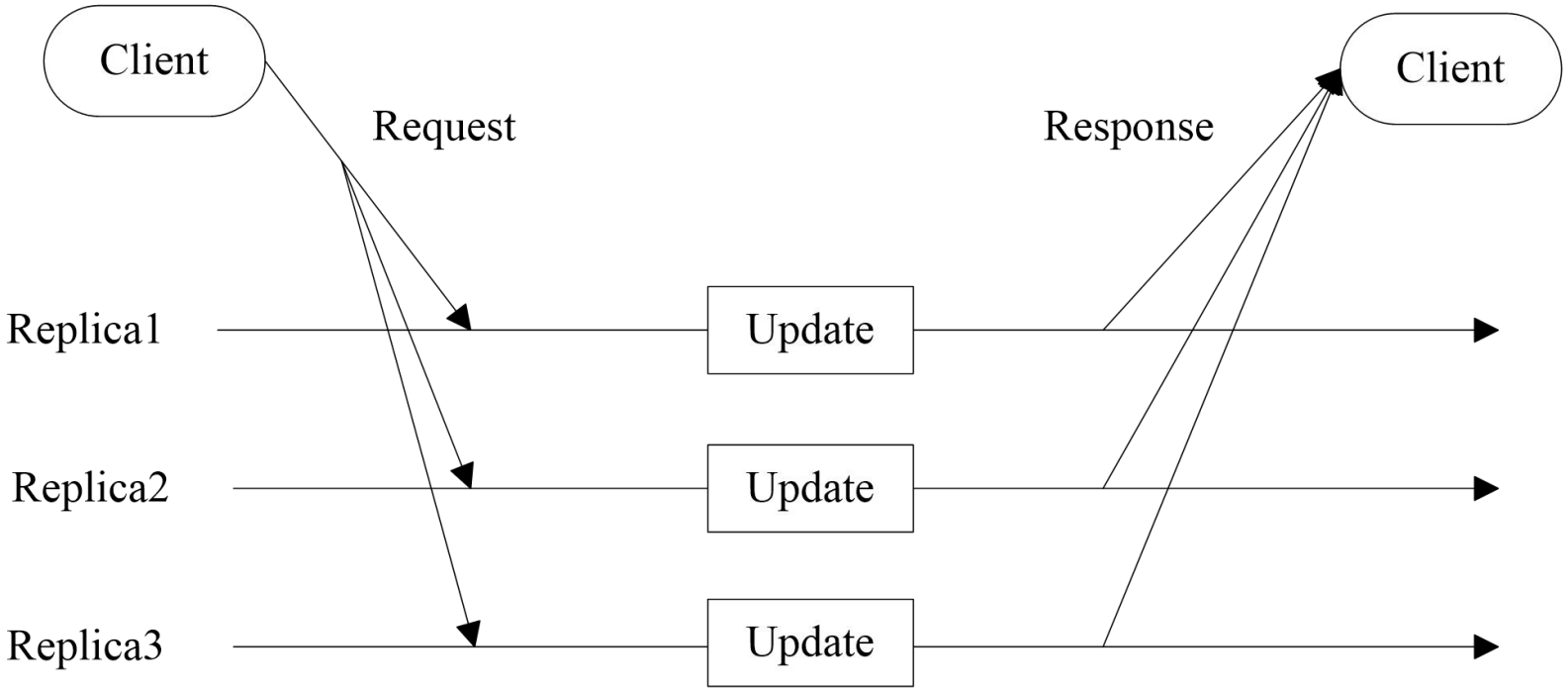
资源冗余

- 资源冗余： 系统各部件的冗余，资源复制是关键
- 复制方法
  - 按方式分类
    - 主动复制
    - 被动复制
  - 按实时性分类
    - 同步复制
    - 异步复制
- 复制的关键问题
    - 一致性问题

资源复制的一致性
用户决策的一致性

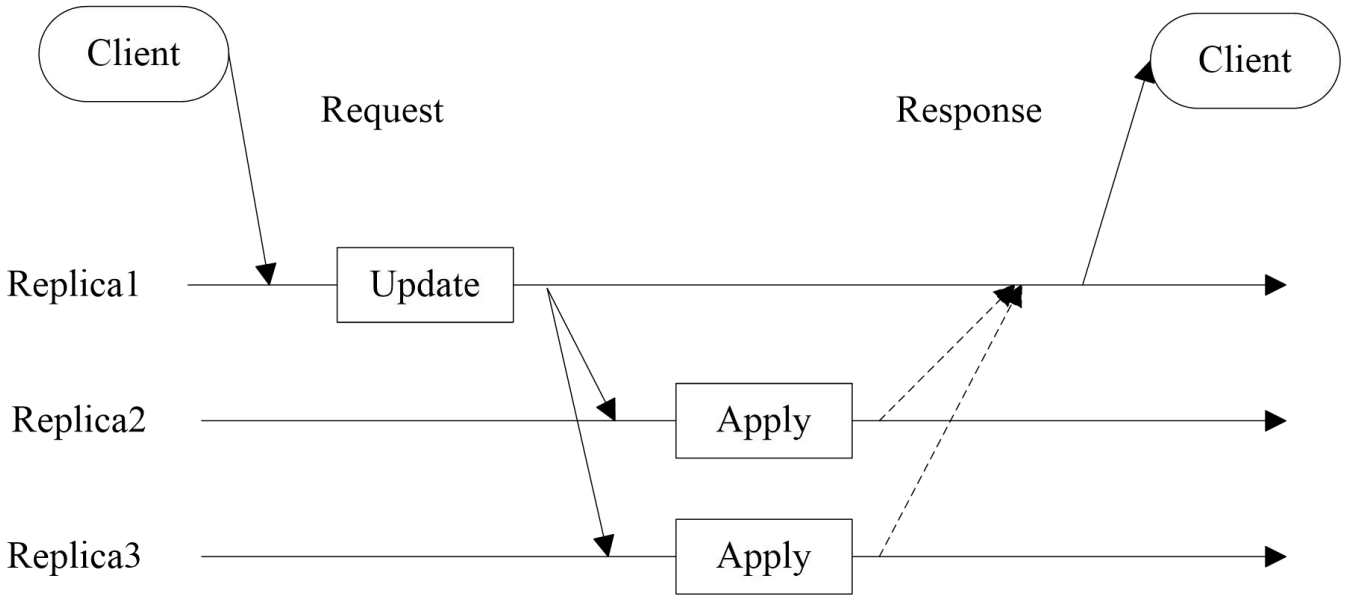
- 主动复制（Active Replication）：又叫做状态机方法（State Machine Approach）。
  - 它有如下确定性假定：对于每一个复制服务器，相同的请求输入按照相同的顺序执行，会产生相同的输出结果，即服务器是一个确定性的自动机。在确定性前提下，客户将请求发往所有服务器，服务器执行请求就使得服务器的状态保持一致。因此，当某个服务器失效后，其他的服务器仍然可以处理客户请求，达到不间断服务的目的。
    - 时间戳的应用
  - 应对Byzantine失效的投票机制

- 优点：失效对客户透明，响应迅速，缺点是比较耗费资源。

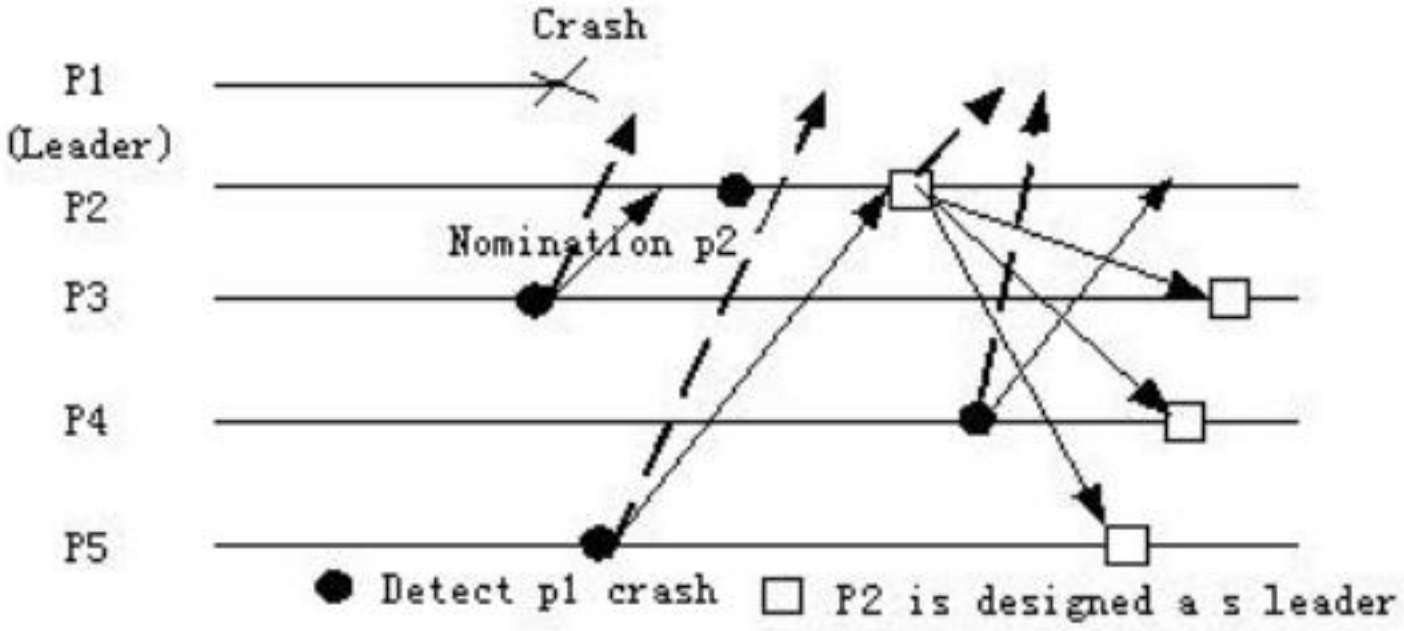


- 被动复制（Passive Replication）：
  - 在主动复制中各复制服务器的地位相等，都处理来自客户的请求；而在被动复制中只有一个服务器处理客户请求，此服务器称为主服务器（Primary Server），其它服务器处于准备状态，称为备服务器（Backup Server）。只有主服务器处理客户请求，只有它的状态在更新。要使各服务器之间状态同步，还要对备服务器状态进行更新。

- 与主动复制相比，被动复制的优点是节省计算资源，可采用较低层次的硬件级复制来降低复制的复杂性。
- 被动复制的缺点是主服务器的失效对客户是不透明的，且无法应对Byzantine失效，另外从备服务器中选出一个作为接替者的选举过程也较慢。



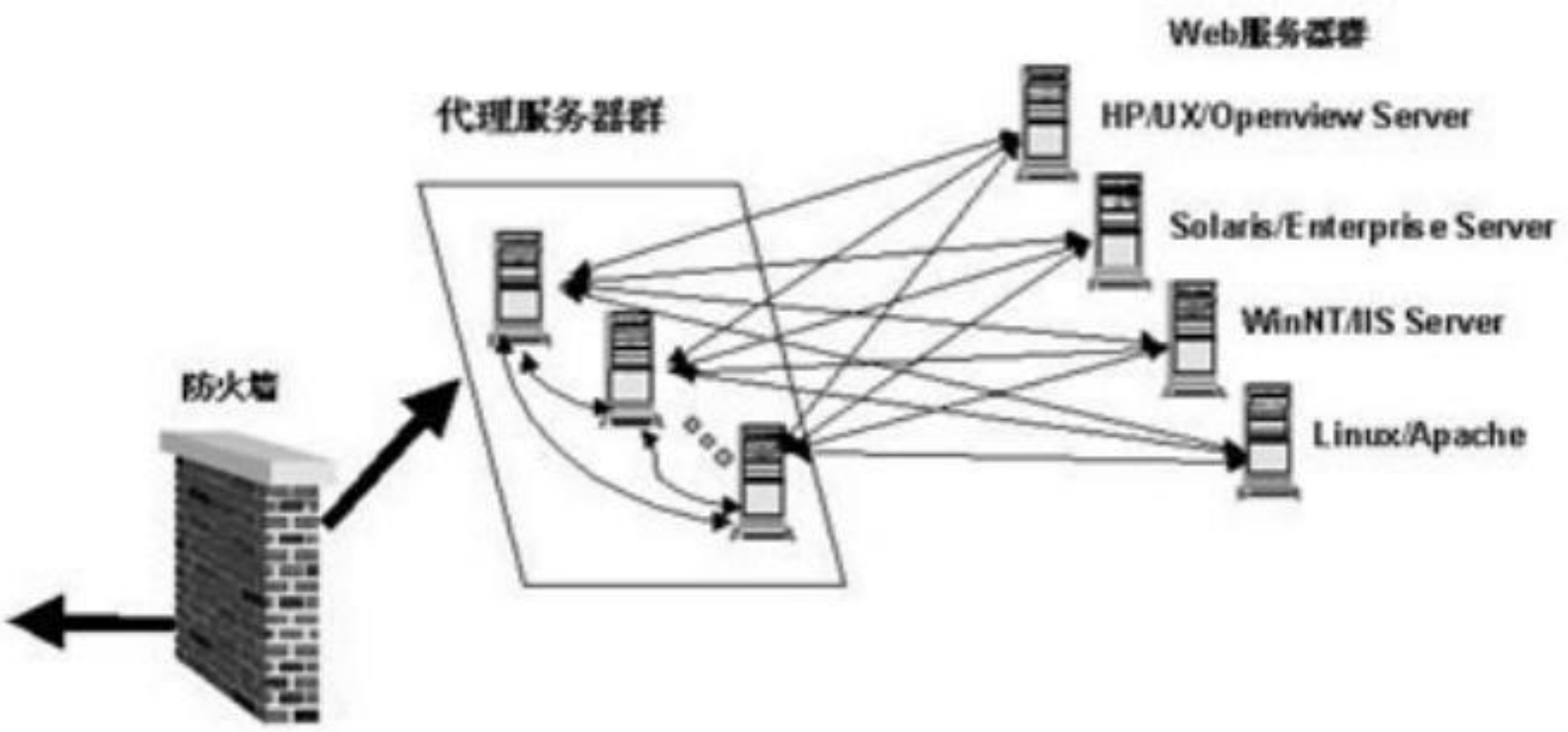
- 主服务器的选举


Crash
P1
(Leader)
P2
Nomination p2
P3
P4
P5
Detect p1 crash
P2 is designed a s leader

- 方法冗余：多种方法来实现系统的特性和功能，通过这种方法可以增强系统保持某些服务和特性的能力。
- 系统具有异构特点
  - 平台异构
    - 设备异构
    - 链路异构
    - 操作系统异构
    - 软件平台异构

- 数据异构
  - 数据存储方式、组织结构等方面的异构设计
- 方法异构
  - 实现方法上的异构
  - 相同的输入能得到相同的输出，但是实现过程是完全不同的。
- 关键问题
  - 如何降低实现方法的相关性，如何组合冗余的实现方法
  - 如何动态地平衡代价和收益
  - 如何设计能够灵活有效的支持方法冗余的软件系统

Web服务的入侵容忍


Web服务器群
代理服务器群
HP/UX/Openview Server
Solaris/Enterprise Server
WinNT/IIS Server
Linux/Apache
防火墙

- 时间冗余：时间冗余就是同一个操作多次重复执行。通信协议中的超时重发机制。
- 信息冗余：提高系统保护数据完整性的能力，提高系统的数据可用性。最简单的一种信息冗余方式是数据的完全复制，将数据完全复制到冗余的存储设备上，这种信息冗余方式能够在发生意外事故的情况下保证数据的完整性

自适应方式

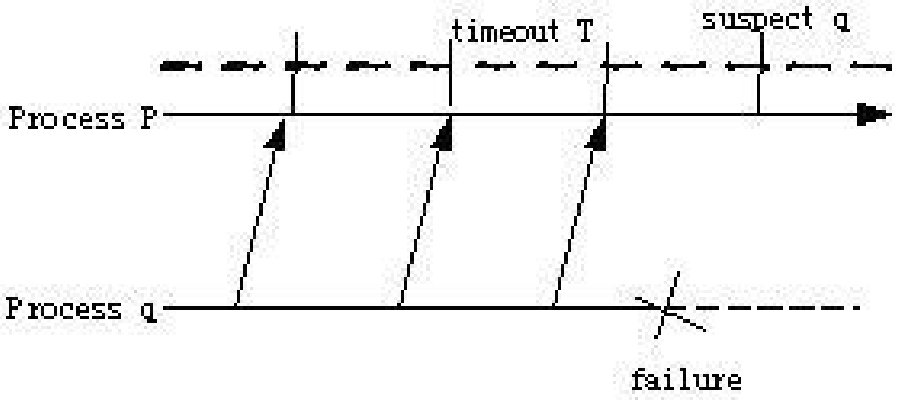
- 自适应：自适应性指的是系统在运行过程中通过更新自身的配置、调节自身的行为来适应环境、满足用户需要的过程。例如TCP协议的拥塞控制、流控
  - 当系统某部分因恶意攻击或者意外事故而被破坏时，通过自适应技术排除被破坏的部件，进入降级的工作状态，保证为用户提供必要的服务；
  - 系统通过自适应可以不断的改变自身的状态、行为，使攻击者很难获取自身确切的信息，从而增加了攻击者破坏系统的难度；
  - 通过自适应来调整自身的生存性机制，当处于危险状态时，使用高安全机制，用足够的系统资源来保证系统的安全；而处于正常状态时，使用一般安全机制，使更多的系统资源用于为用户提供服务。

失效检测

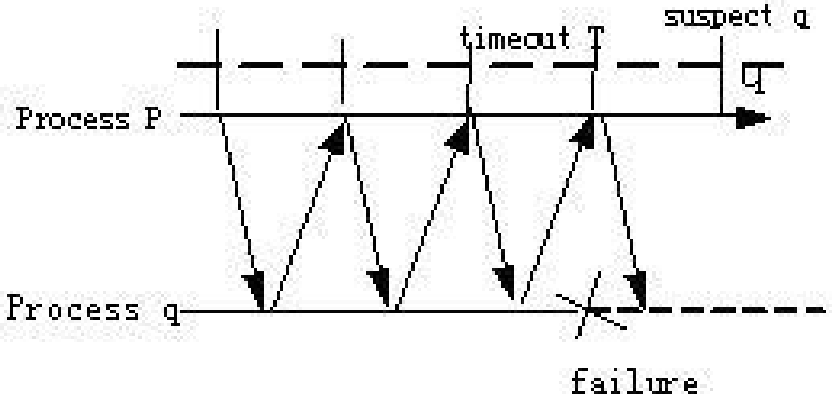
- 是分布式系统的基本问题之一
- 超时检测机制
  - 心跳消息
  - 夹带
- 面向Failstop故障
  - 崩溃
  - 有时候将网络断开也归入Failstop故障

实现检测方法

- 消息的发送方式
  - Push 模式
  - Pull 模式
  - 夹带模式



Push 模式



Pull 模式

隔离和修复

- 隔离和修复
  - 检测：从服务的角度对系统进行检测，主要是对系统的业务交互过程和系统状态进行检测，及时发现恶意的业务或者被破坏的系统部件
  - 隔离管理：用来隔离恶意的业务交互、系统中被破坏的数据或者部件，防止失效在系统中扩散
  - 修复：按照一定的策略修复系统中被破坏的数据或者部件，对于数据的修复一般采用根据执行日志回朔的方法，对于部件的修复一般采用重配置或者重启的方法。

迁移(failover)

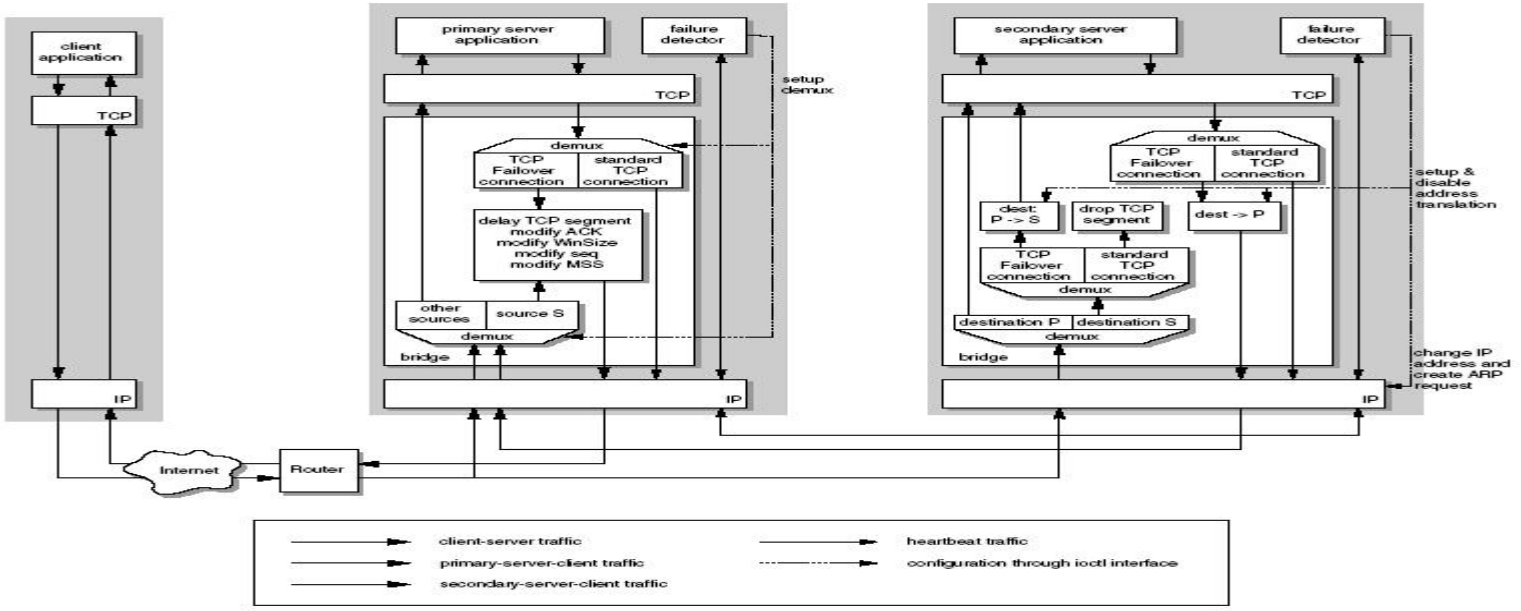
- 迁移(failover)
  - 业务由系统中的一个节点向另一个节点移交控制的过程 。
- 迁移的基础
  - 存储数据迁移
  - 系统状态 (粗粒度允许当前进行的业务丢失,细粒度保证当前进行的业务不能丢失)
- 要求
  - 要透明于应用。
  - 速度要快。

迁移(failover)

- 传统的方法
  - IP takeover(改变IP): 也叫“漂移”IP地址
    - 典型应用：HA
  - IP aliasing
- 上述方法可以完成切换，失效时正在进行的业务无法保持

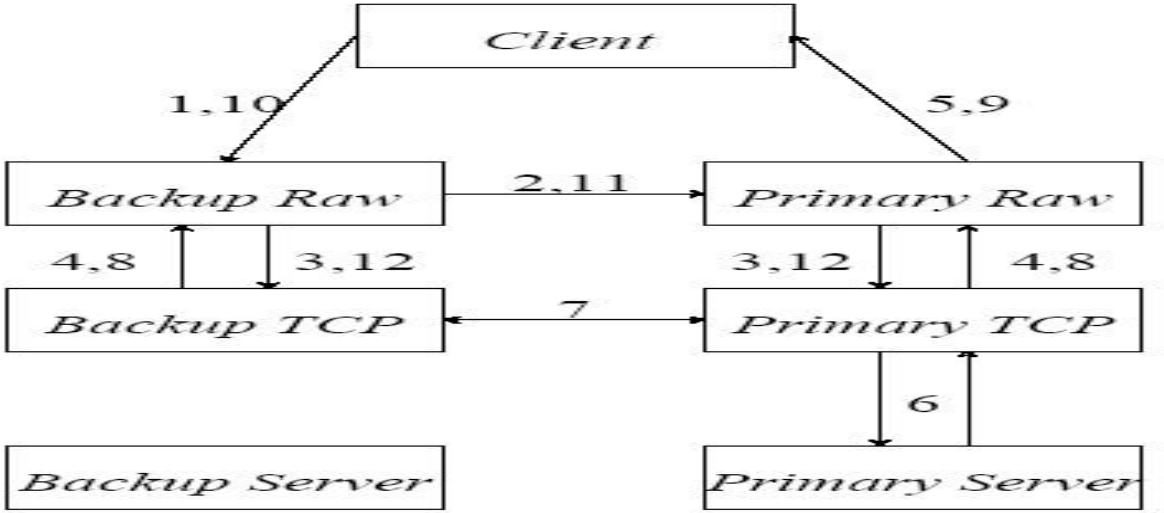
迁移(failover)

- TCP failover
  - 通过对服务端的协议栈修改，实现TCP连接的迁移
  - University of California, Santa Barbara进行了这方面的研究，实现了FT-TCP。



迁移(failover)

- 针对服务的failover
  - UCLA(加利福尼亚大学洛杉矶分校)的Navid Aghdaie针对Web服务系统的透明容灾问题进行了研究 。

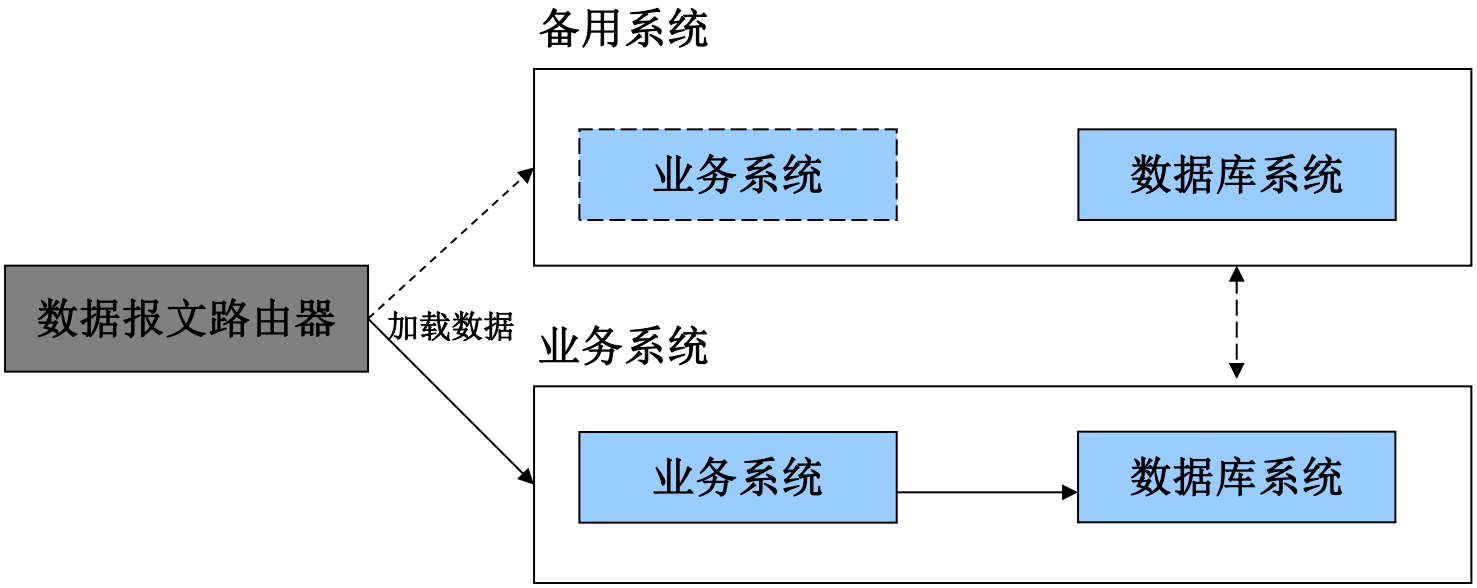


一些应用领域

- 通讯网络
  - 其核心问题是如何设计网络拓扑结构，使得当一部分线路或节点因攻击或故障失效时，存在备份传输路由，同时最大可能降低网络的成本。
- 信息存储系统
  - 可生存信息存储系统（Survivable Information Storage System）的目标是：在假定攻击和故障条件下系统仍可以提供可信的存储服务。

- 数据库
  - 数据库安全问题的一个方面是如何针对恶意事务（Malicious Transactions）的容侵，针对恶意事务引起的损伤，如何最大可能减小损伤的范围。
- 中间件
  - 提高中间件系统的自适应能力为中心，综合应用多种技术提升分布式系统的可用性能力
- 代理系统
  - 代理系统在提供高效率、基于组件的开发的同时，由于网络之间存在复杂的并行交互，以及不可预测的攻击行为，常常以牺牲代理系统为代价提高整个系统的可用性。

- 一个例子



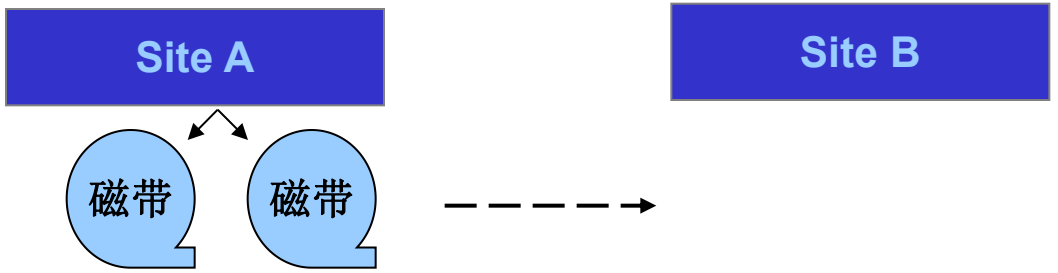
平台异构

- 操作系统不同，数据库相同
  - 较容易在数据库层进行数据备份和复制
- 操作系统相同，数据库不同
  - 可以在备份系统上同时装有两个数据库
  - 异构的数据库用于备用系统的临时接管，预防暂时性失效。
  - 同构的数据库用于在生产系统发生永久灾难时，备用系统接替生产系统
- 两者都不同
  - 只能在应用层的进行数据复制，复制的数据量不能很大

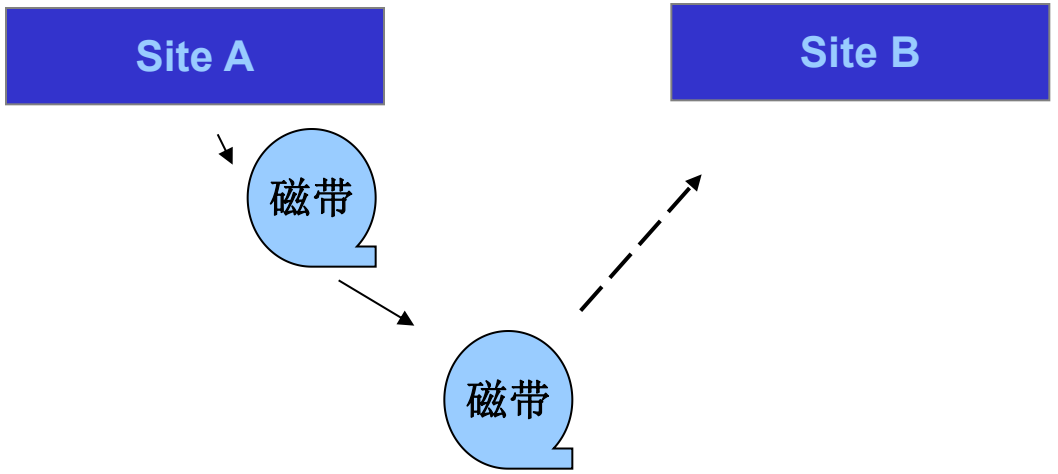
平台同构

- 数据复制技术
  - 可以灵活采用的操作系统级或数据库级的数据复制技术
  - 可以获得商业软件的支持
- 可管理性
  - 管理较容易
  - 文件、数据都可以为两个系统所用

备份方案1



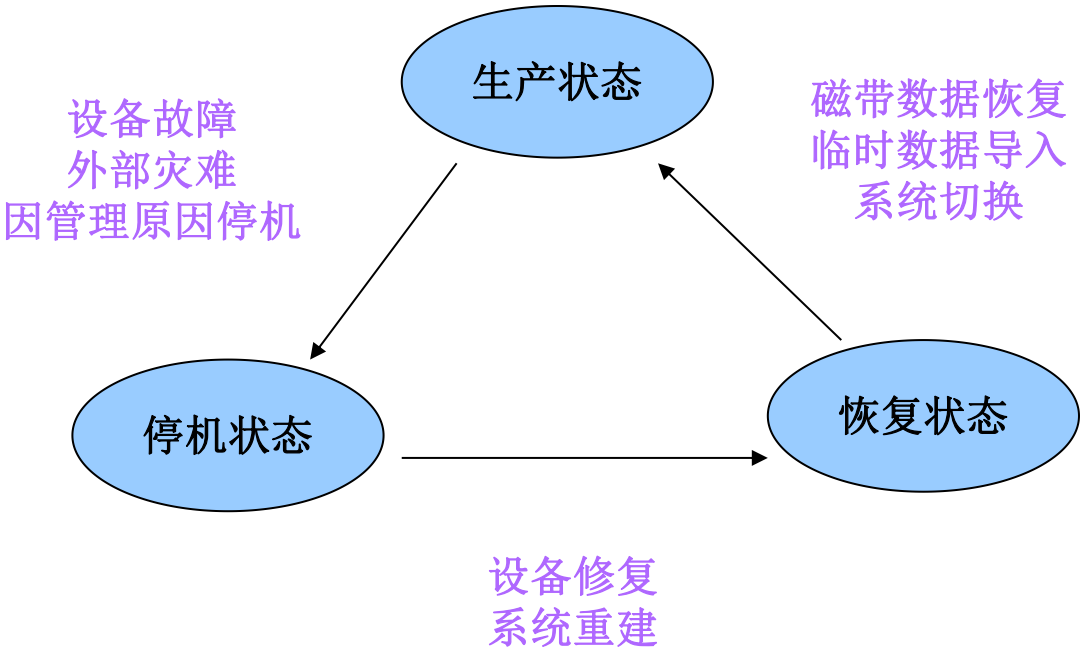
备份方案2



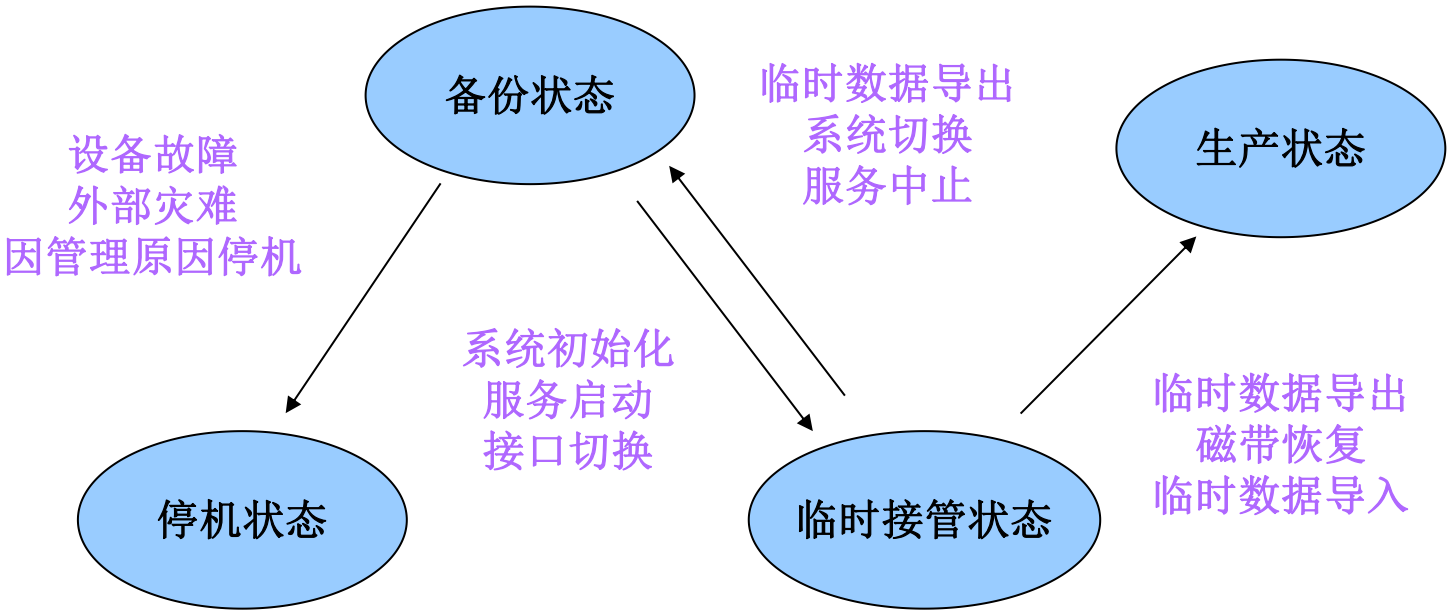
管理的内容

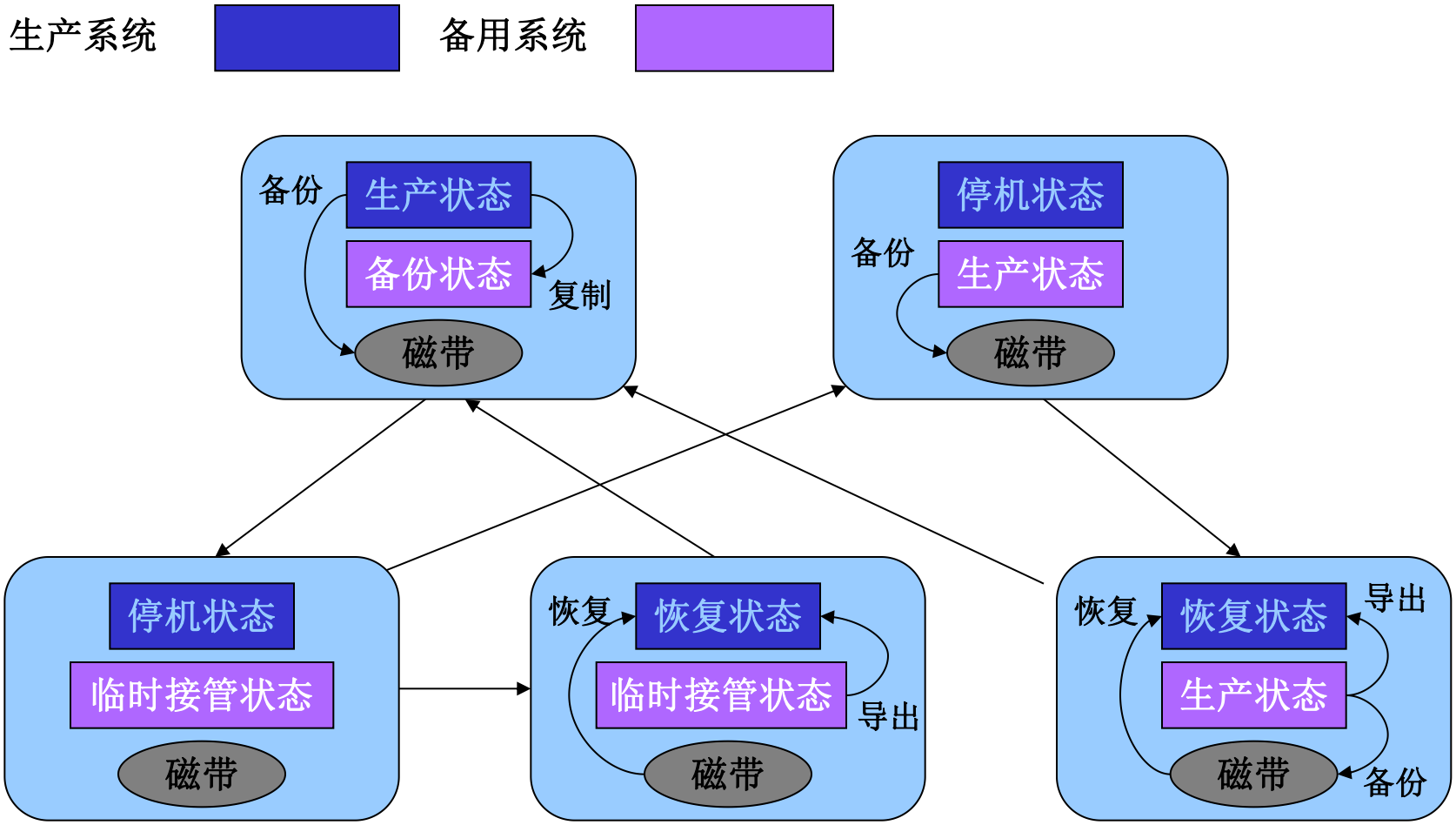
- 实体冗余管理
  - 系统状态控制
- 数据一致性管理
  - 备份管理
  - 复制管理：事务双写，同步复制，异步复制
  - 数据转移：把备用系统中的临时数据导到业务系统中
- 失效或灾难事故报告处理及记录

生产系统

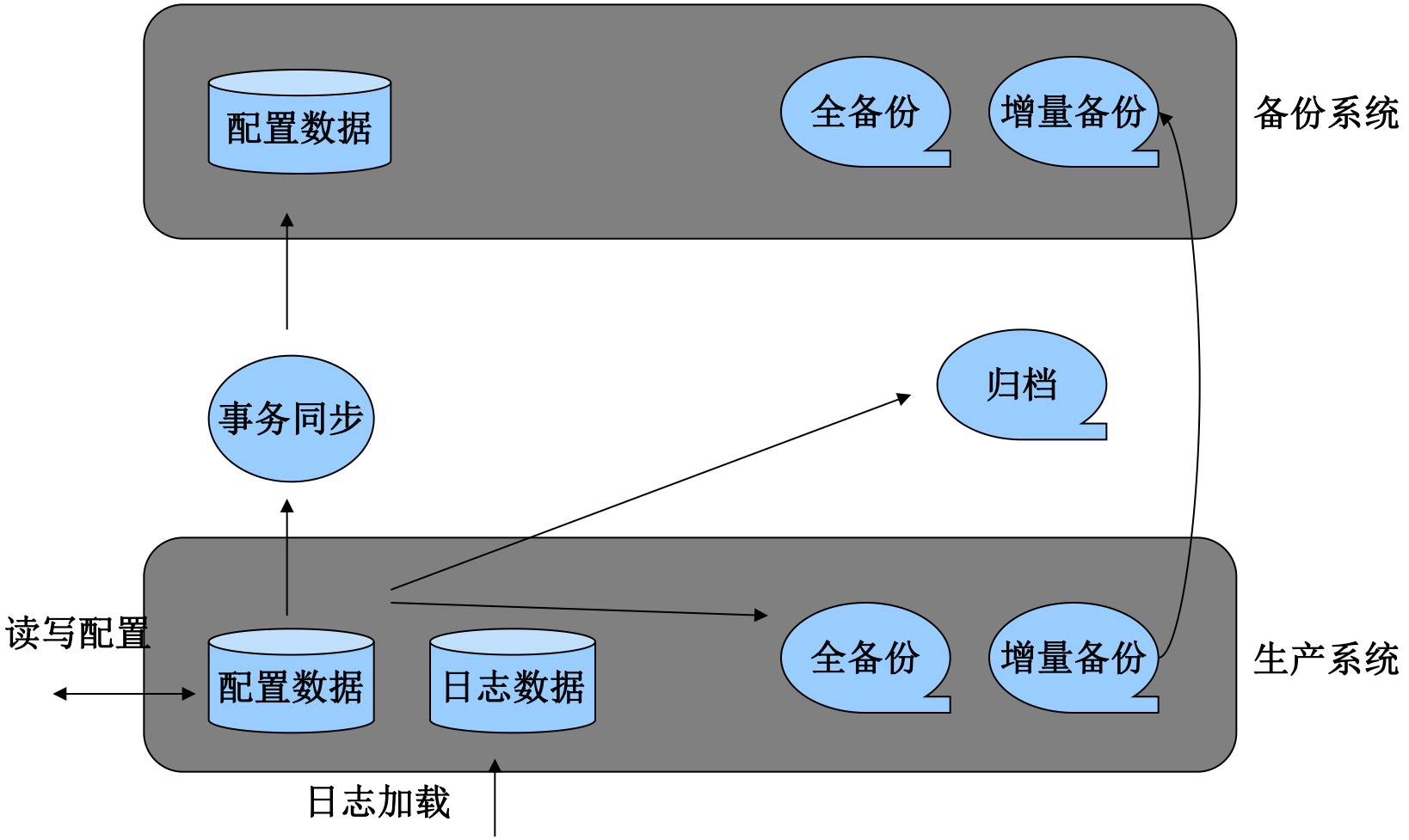

生产状态
设备故障
外部灾难
因管理原因停机
磁带数据恢复
临时数据导入
系统切换
停机状态
恢复状态
设备修复
系统重建

备用系统

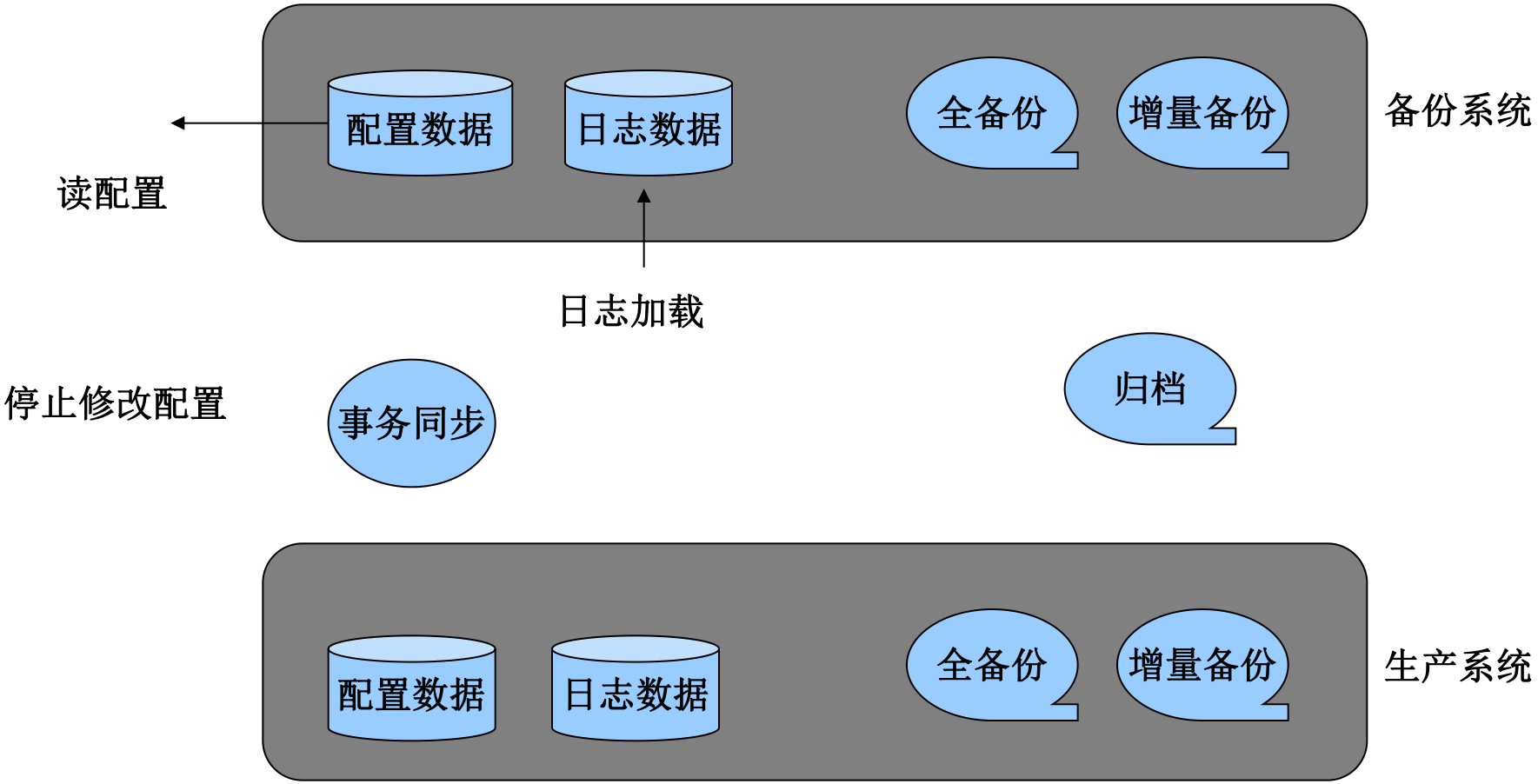

备份状态
生产状态
停机状态
临时接管状态
设备故障
外部灾难
因管理原因停机
临时数据导出
系统切换
服务中止
系统初始化
服务启动
接口切换
临时数据导出
磁带恢复
临时数据导入


生产系统
备用系统
备份
生产状态
备份状态
复制
磁带
停机状态
备份
生产状态
磁带
停机状态
临时接管状态
磁带
恢复
恢复状态
临时接管状态
导出
磁带
恢复
恢复状态
导出
生产状态
磁带
备份

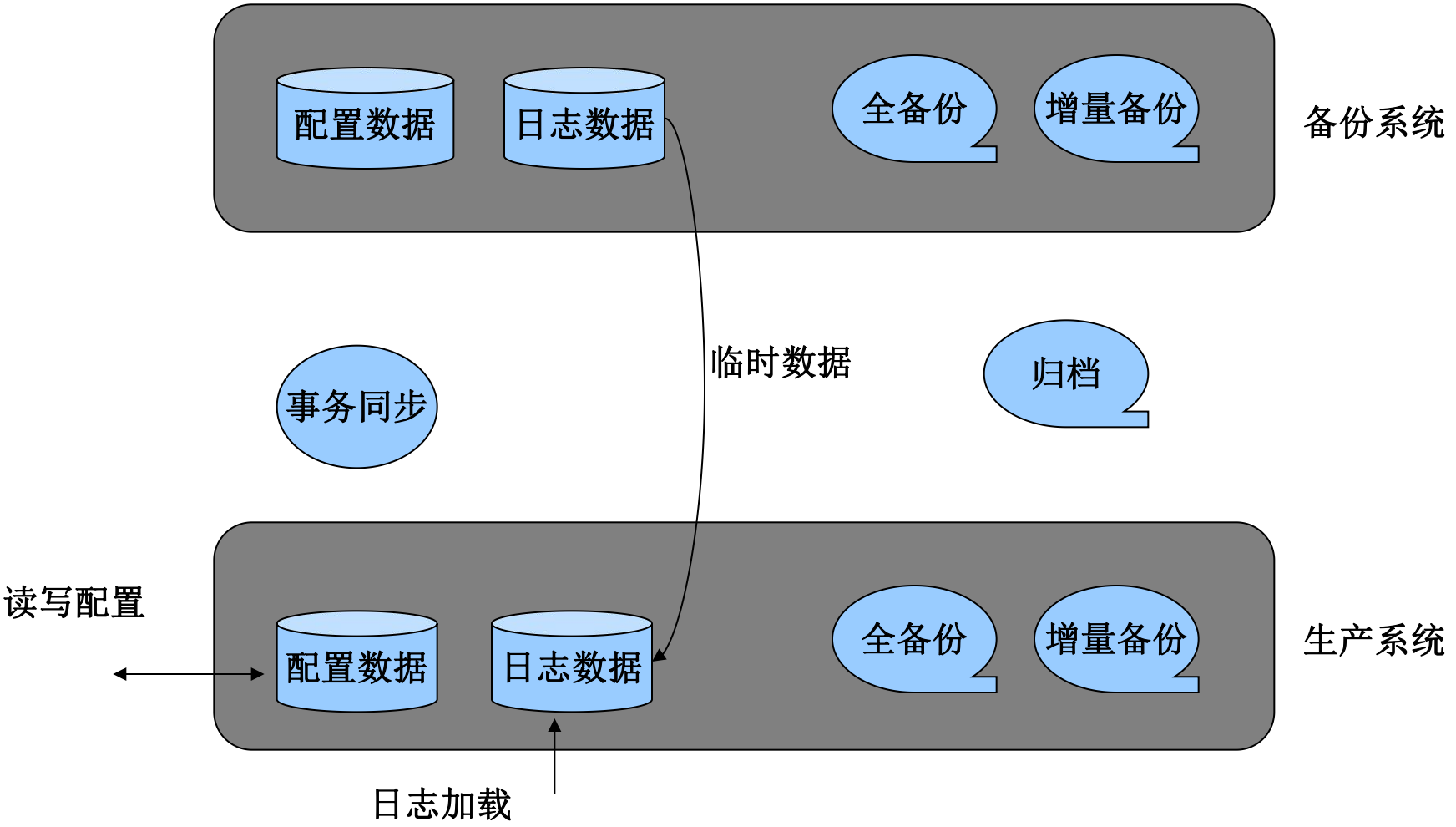
正常时工作状态：



生产系统停机时工作状态：



生产系统恢复时工作状态：



软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件安全开发

- 3.1 软件安全的指导原则
- 3.2 软件安全需求
- 3.3 软件安全编码基本方法与技术
- 3.4 软件安全设计

面向网络攻击的软件安全设计

- 可用性安全
- 机密性安全
- 完整性安全
- 可控性安全
- 可审性安全

数据可用设计

- 高频数据与快照

快照有三种基本形式：基于文件系统式的、基于子系统式的和基于卷管理器/虚拟化式的，而且这三种形式差别很大。

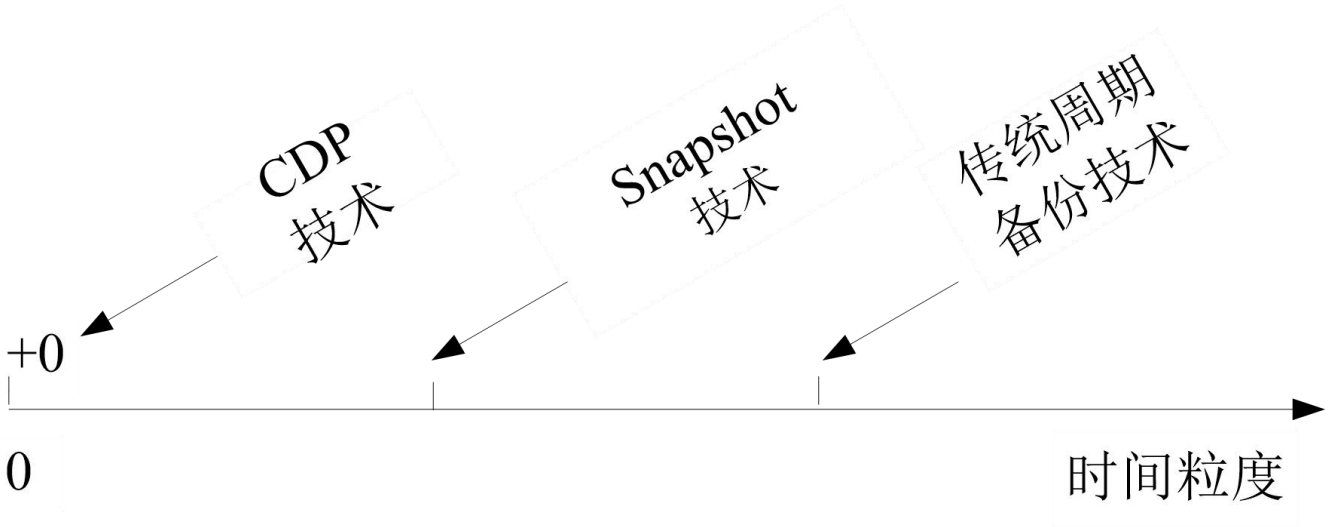
- 分布式存储

高频数据存储与恢复

- 需求
  - 对象需求：重要数据、变化较为频繁
  - 可能引起数据丢失的原因
    - 逻辑错误(由误操作、软件错误、恶意攻击等引起的数据污染)
    - 物理错误(由介质损毁引起的数据丢失)
  - 数据恢复的需求：
    - 数据存储与恢复的粒度较细
    - 需要恢复到具体某个时间点的状态
    - Redo，Undo恢复
  - 难题
    - 数据量大，难以在有限的时间内完成全备份
    - 数据备份不可用失败，例如在备份时，有数据从一个未备份的目录移动到已经备份过的目录
    - 会不会影响应用系统的性能
  - 数据的存储技术能否是与应用无关

CDP（Continuous Data Protection）数据复制为背景，根据多版本间的时序性分析，建立同时满足前滚和后滚的双向检索索引结构。与当前索引技术相比，使用同一种索引结构支持双向检索操作。双向索引结构适于应用在密集写操作的应用场景，结合检查点技术可以提供面向应用一致性的数据恢复方法。

- 高频数据保护技术分类



快照存储技术

- 什么是快照？
  - 是特定数据集的一个完整可用拷贝，该数据集包含源数据在拷贝点的静态映象；快照可以是数据再现的一个副本或者复制。
  - “快照”通常被定义为一组文件、目录或卷在某个特定时间点的副本。“快照”这个名字的含义与“照片”相似，它所捕获的是一组特定数据在某个时间点的映像。
- 快照有三种基本形式：基于文件系统式的、基于卷管理器/虚拟化式的和基于系统式的。对应不同类型的实现主体
  - 主机文件系统(包括服务器、台式机、笔记本电脑)
  - 逻辑卷管理器(LVM)
  - 存储虚拟化装置
  - 网络附加存储系统(NAS)
  - 磁盘阵列
  - 主机虚拟化管理程序
  - 数据库

快照的价值

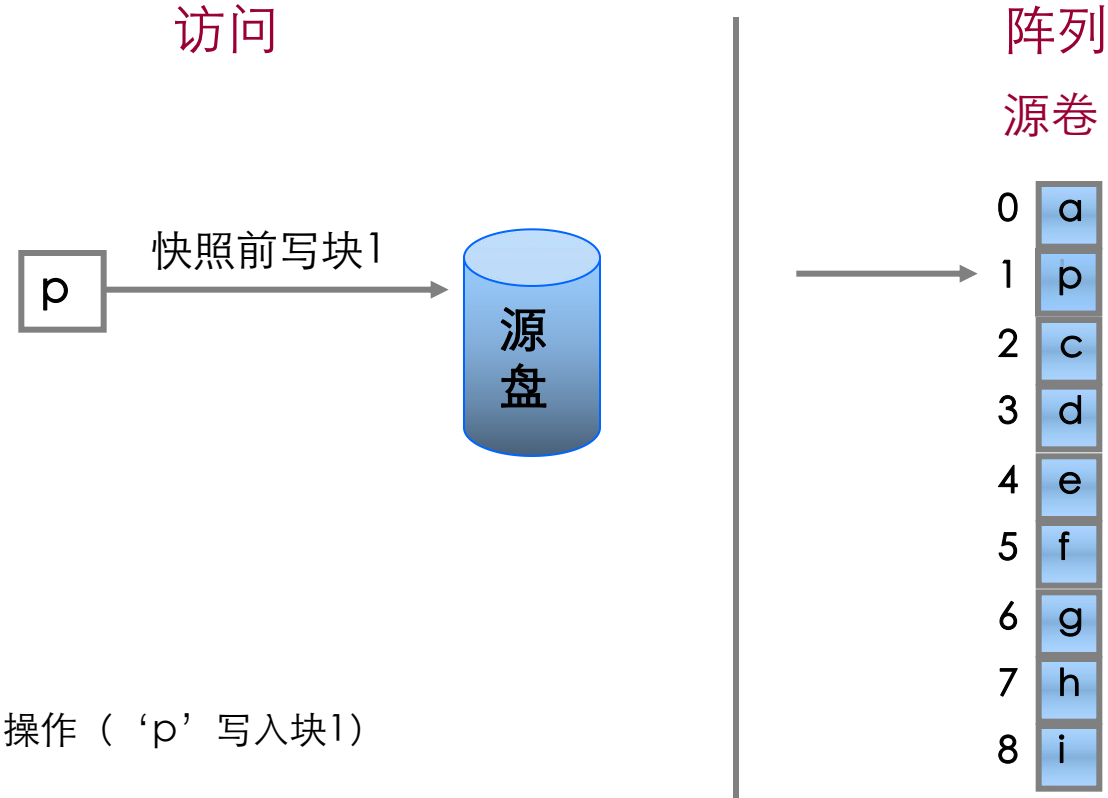
- 快速备份/恢复
  - 快照可以迅速建立，并可用作传统备份和归档的数据源，所以快照可以缩小或消除备份窗口；
  - 快照存储在磁盘上，可以快速直接存取。
- 多个恢复点
  - 基于磁盘的快照使存储设备有灵活和频繁的恢复点（或称恢复点目标：RPO），可以快速通过不同时间点的快照尽快恢复数据。
- 快照增多的代价是提高成本。

快照工作原理分类

- 快照
  - 全拷贝快照
    - 分离镜像（“Splitting”a mirror）
  - 差分快照
    - 写即拷贝（CoW:Copy On Write)
    - 写即重定向(RoW:Redirect On Write)

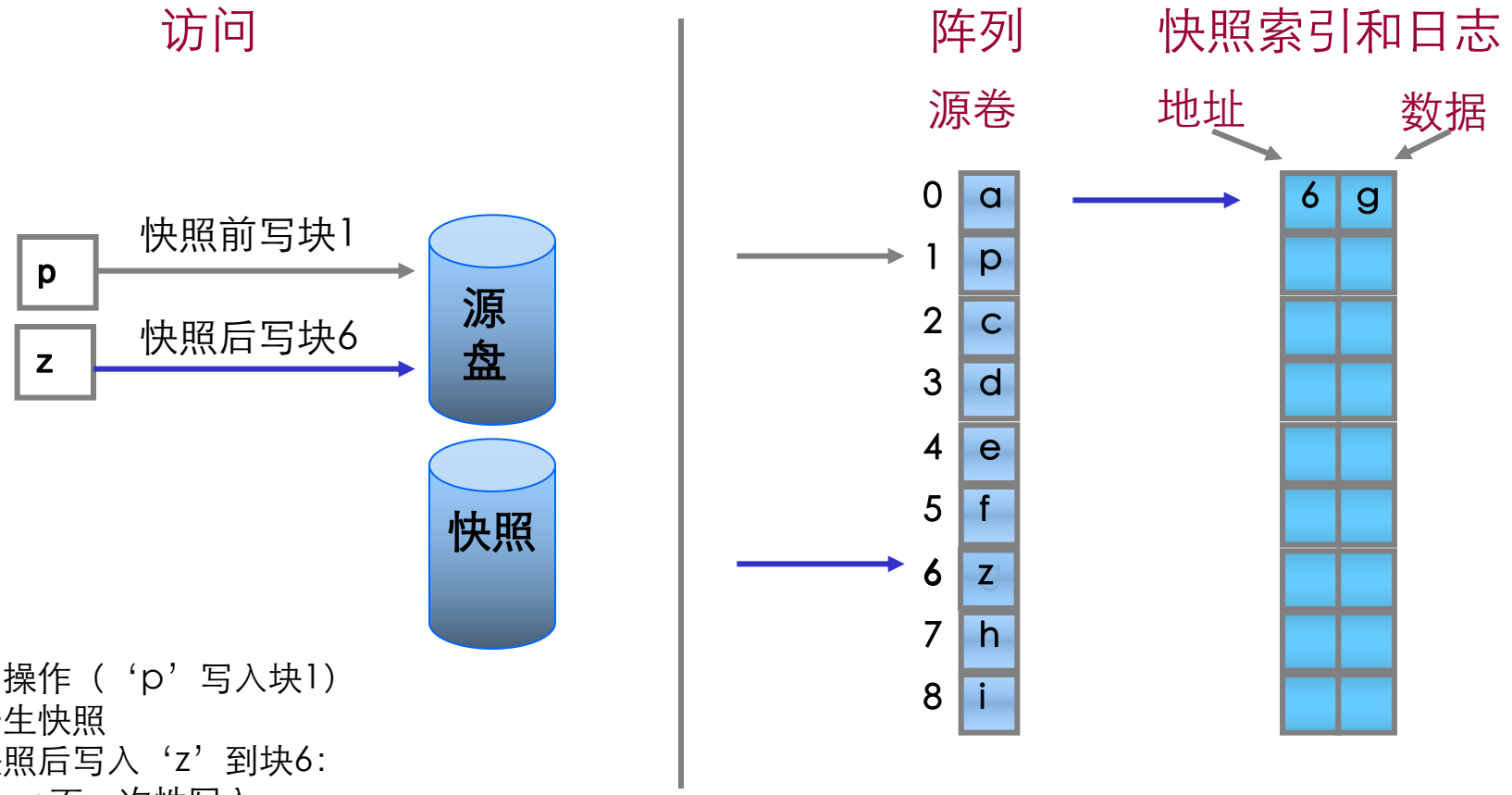
- 快照复制技术
  - 分离镜像（“Splitting”a mirror），克隆
    - 快照所创建的是数据的完整副本
  - 差异复制
    - 保存的是数据变化的内容
      - Copy-on-write (COW)写即拷贝,复制写快照
        - » COW跟踪数据卷的写操作和数据块变化。当某个数据块发生改变时，在将旧的数据覆盖之前，首先将该块的旧数据复制到预留的快照卷，然后再更新数据卷。
      - Redirect On Write（RoW）写即重定向
        - » RoW跟踪数据卷的写操作和数据块变化。当某个数据块发生改变时，在数据卷上的旧数据不覆盖，首先将该块的新数据复制到预留的快照卷，然后再将该新数据的索引重定向到数据卷。

差分快照实现：CoW



1.写操作（‘p’写入块1）

差分快照实现：CoW



1.写操作（‘p’写入块1）
2.产生快照
3.快照后写入‘z’到块6:
- 不一次性写入
- 先将块6内容移入日志
- ‘z’写入源卷

CoW特点

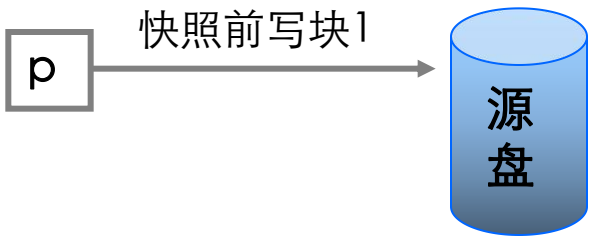
- 源盘保持最新状态
- 两次写操作
- 适合什么恢复？
- Undo操作

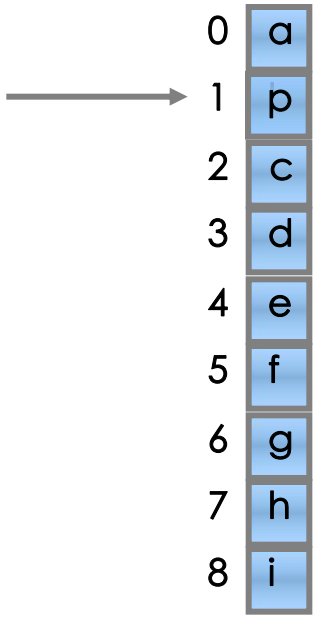
差分快照实现：RoW

访问

阵列

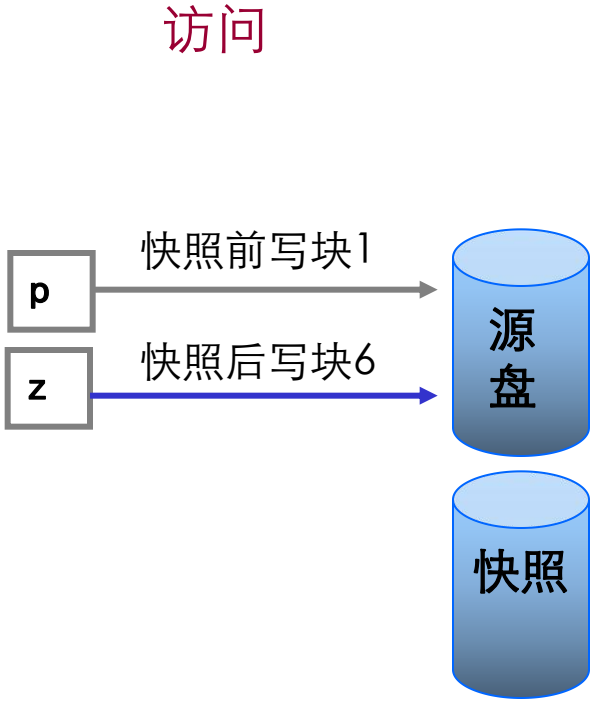
源卷

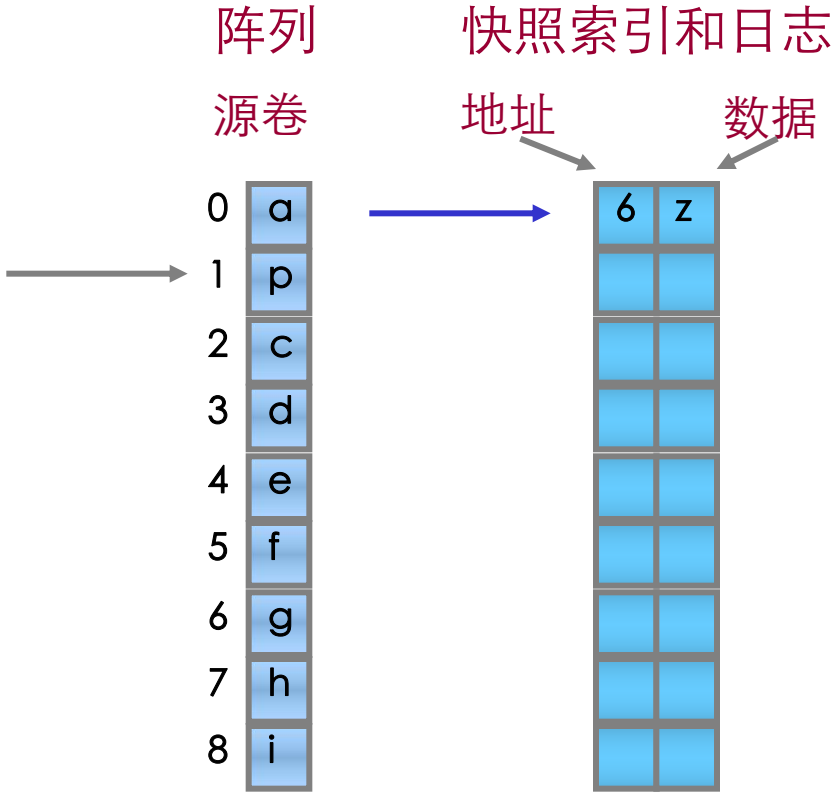




1.写操作（‘p’写入块1）

差分快照实现：RoW



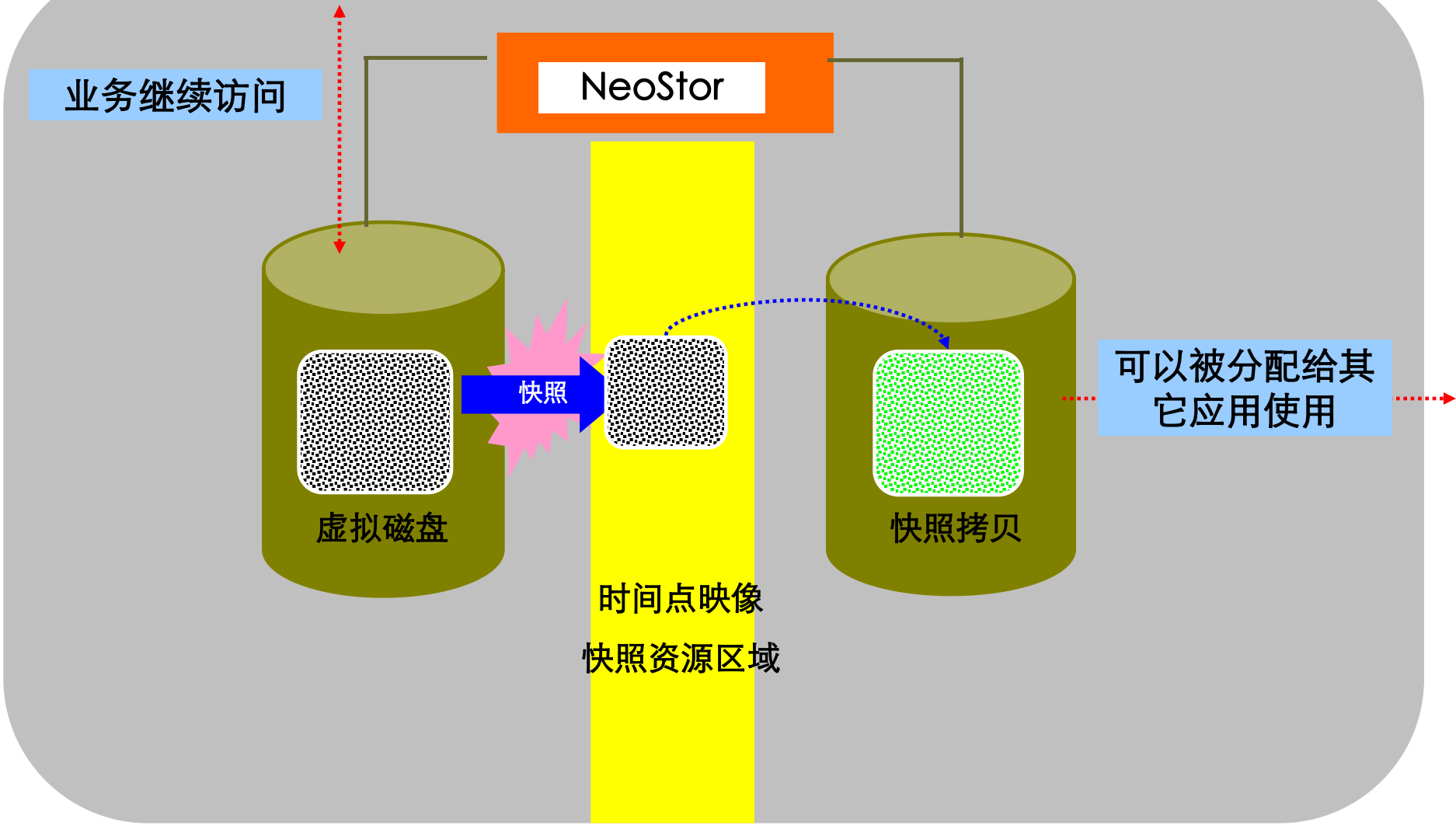


1.写操作（‘p’写入块1）
2.产生快照
3.快照后写入‘z’到块6:
- 源卷块6内容不变
- ‘z’写入日志

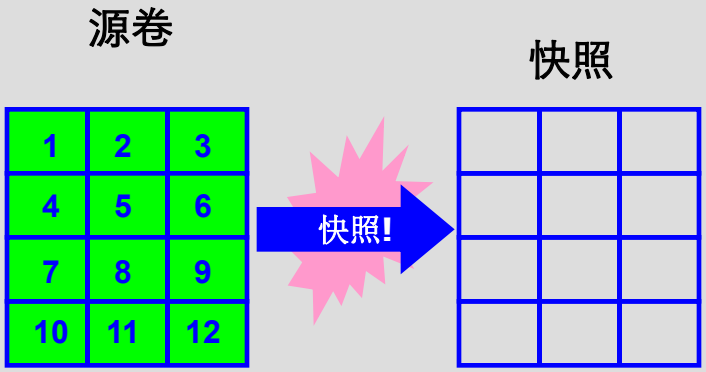
RoW特点

- 源盘保持初始状态
- 一次写操作
- 适合什么恢复？
- redo操作

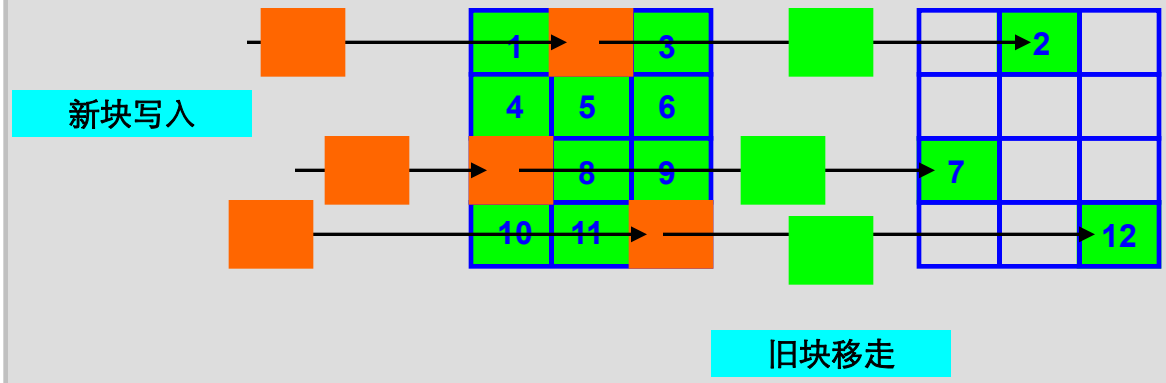
快照拷贝



快照拷贝实现 – Copy-on-Write

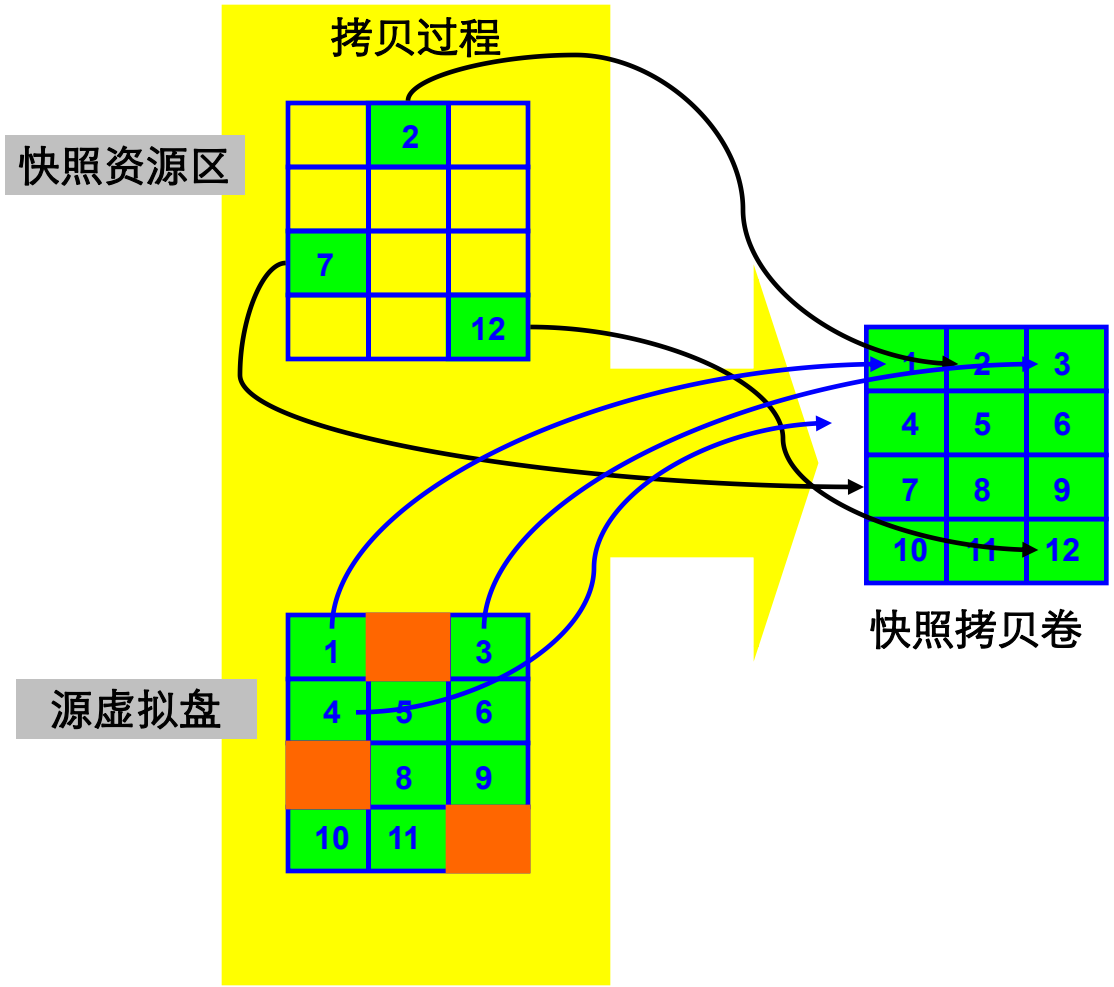


第一次做快照时，快照资源区无数据. 内存中仅维护源卷的一个地址映射.



源卷有新块写入时，旧块数据首先被写到快照资源区。

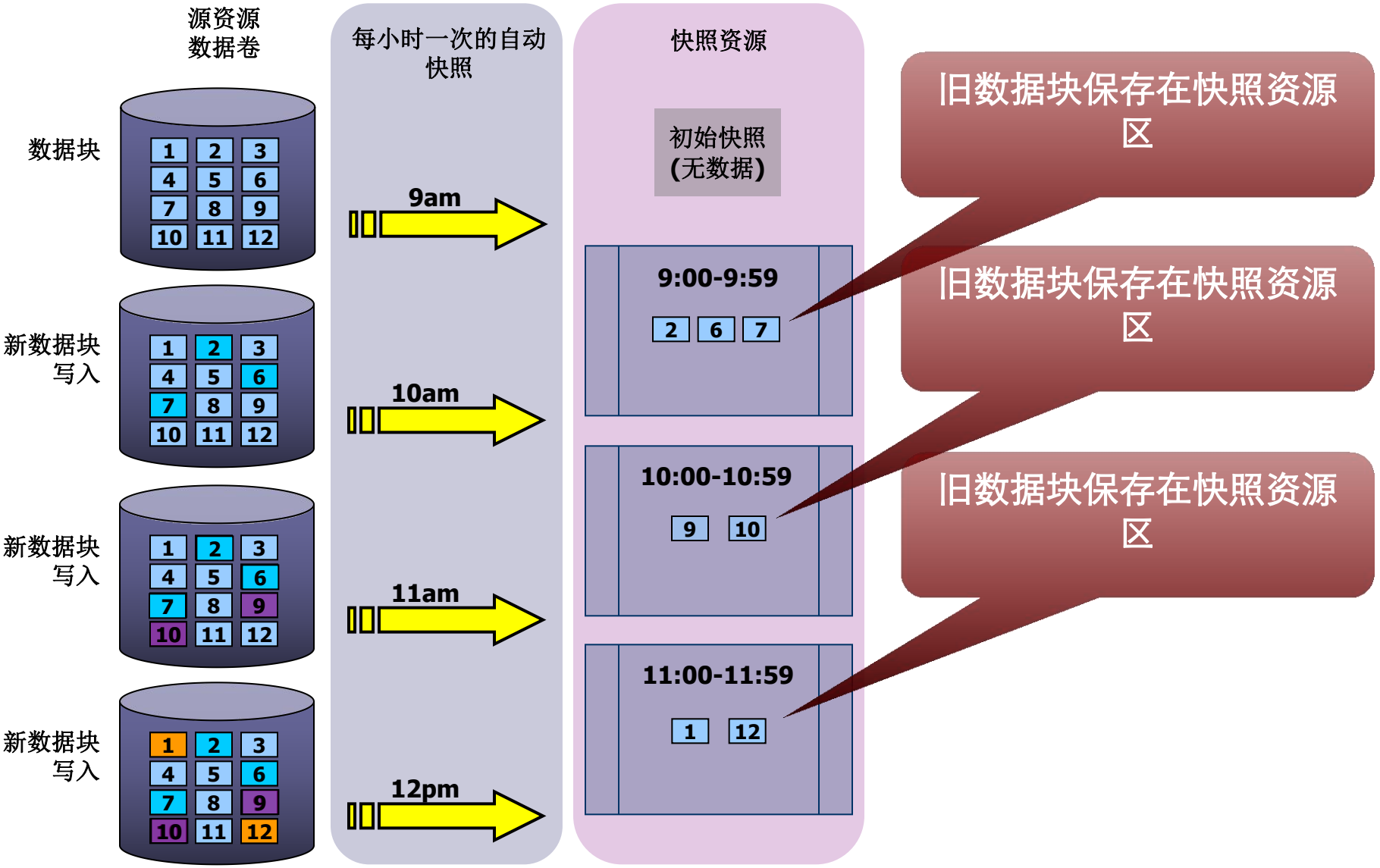
快照拷贝实现 – Copy-on-Write



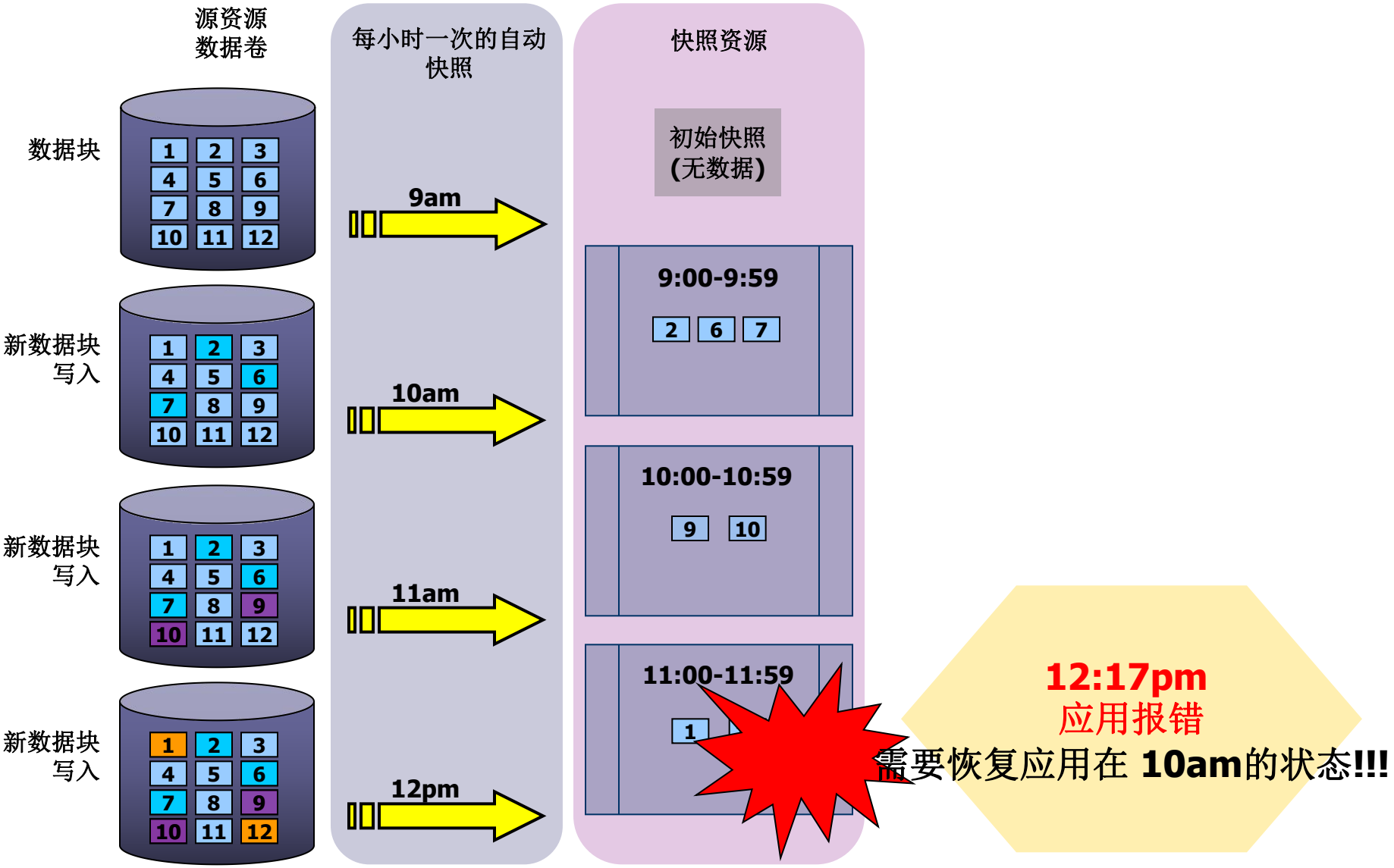
数据拷贝时，检查快照资源区的块是否包含数据，如果有数据就复制到拷贝卷；如果没有数据，则直接从源卷上复制未改变过的数据；

最终结果是产生一个源卷在快照点的数据副本。

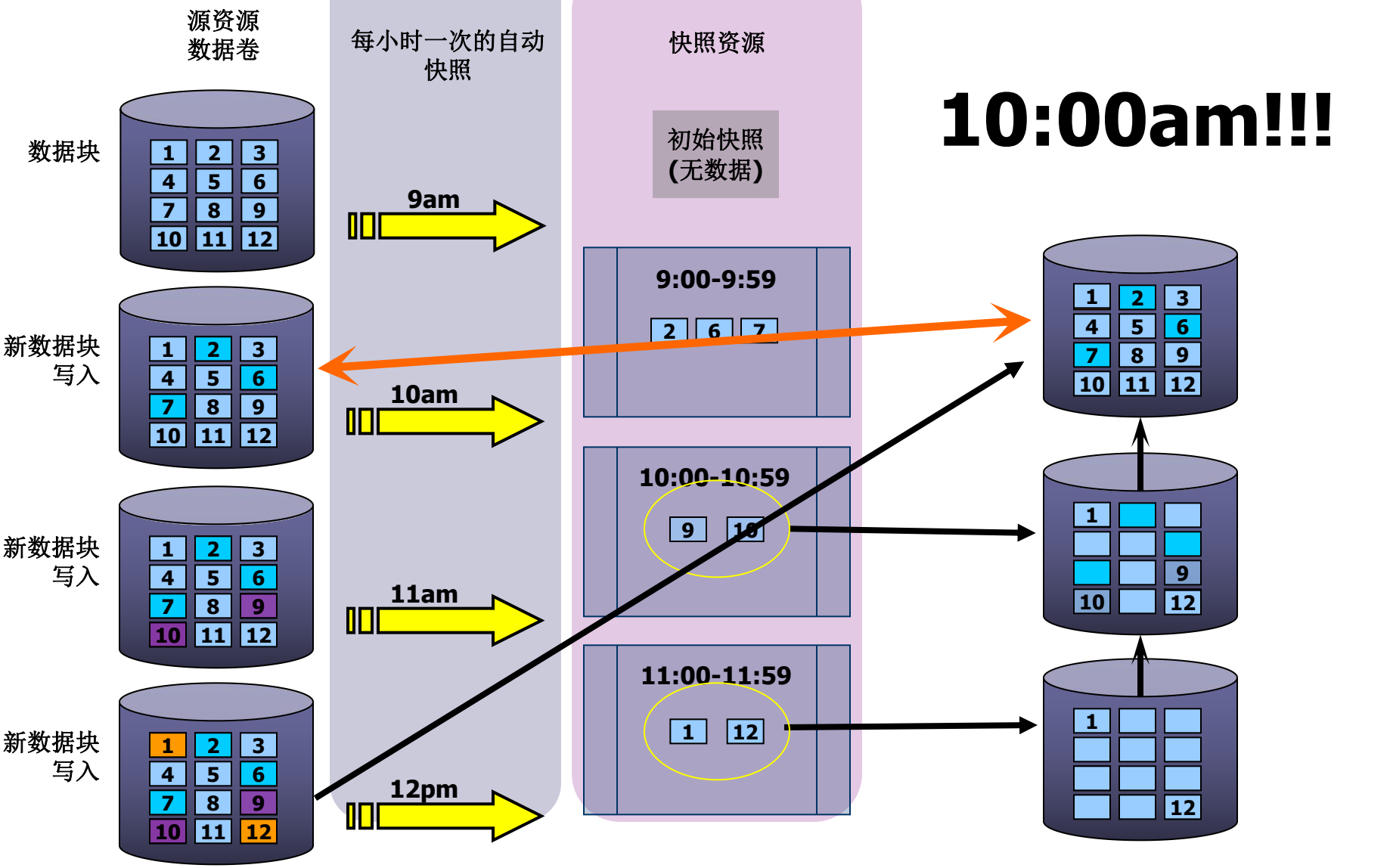
TimeMark


源资源
数据卷
每小时一次的自动
快照
快照资源
数据块
初始快照
(无数据)
9am
10am
11am
12pm
新数据块
写入
新数据块
写入
新数据块
写入
9:00-9:59
2 6 7
10:00-10:59
9 10
11:00-11:59
1 12
旧数据块保存在快照资源区
旧数据块保存在快照资源区
旧数据块保存在快照资源区

TimeMark


源资源
数据卷
每小时一次的自动
快照
快照资源
数据块
初始快照
(无数据)
9am
9:00-9:59
2 6 7
新数据块
写入
10am
10:00-10:59
9 10
新数据块
写入
11am
11:00-11:59
1
新数据块
写入
12pm
12:17pm
应用报错
需要恢复应用在 10am的状态!!!

TimeMark

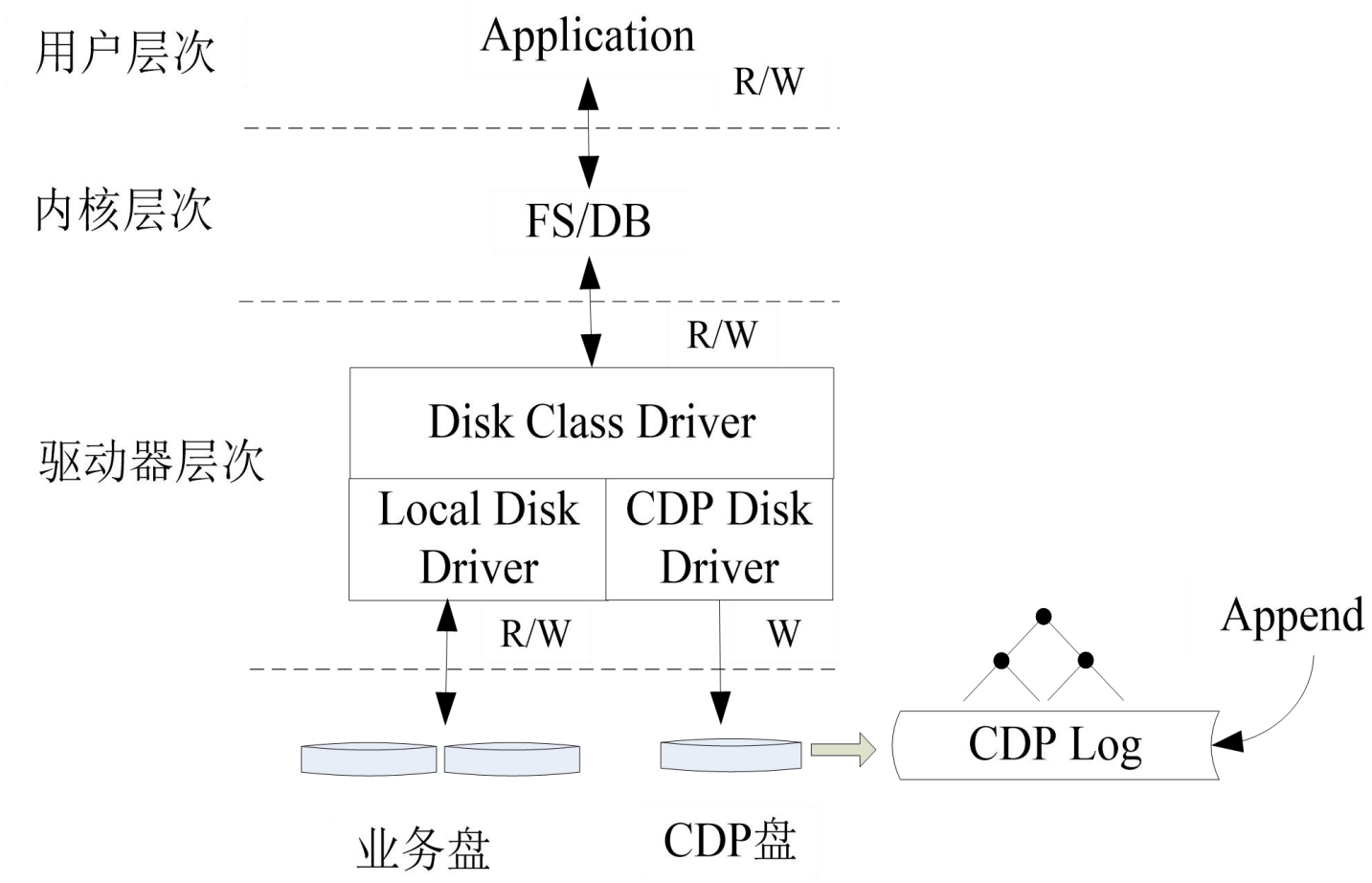

源资源
数据卷
每小时一次的自动
快照
快照资源
数据块
初始快照
(无数据)
10:00am!!!
9am
10am
11am
12pm
新数据块
写入
新数据块
写入
新数据块
写入
9:00-9:59
10:00-10:59
11:00-11:59

什么是CDP？

- CDP:Continuous Data Protection
  - CDP数据保护技术需要实时复制所有版本的更新数据，是数据保护技术的最高级别

CDP技术

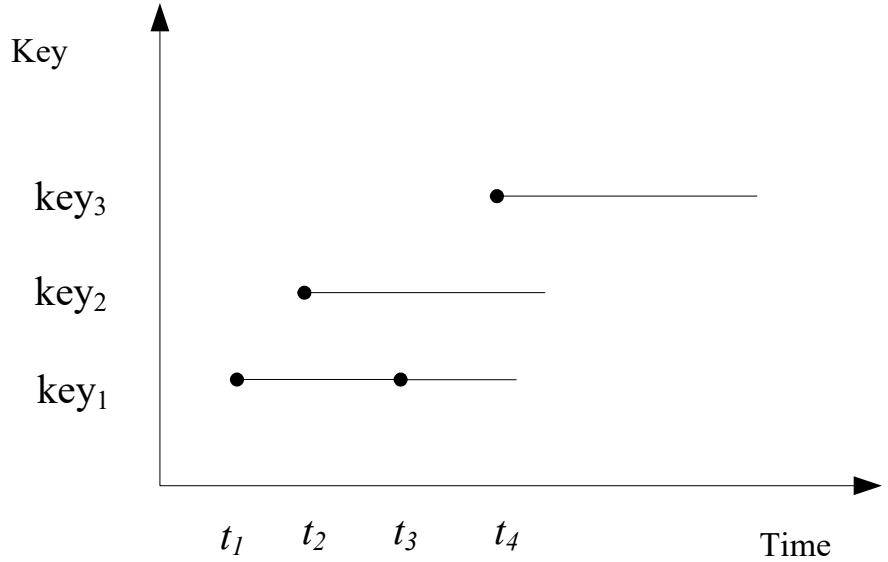
- 驱动器层次产生的CDP备份数据不会带来明显的写操作延迟，适用于极高频背景下的数据复制
- CDP备份系统常用的部署方式是为业务盘配置CDP数据盘。
- 在driver层次为业务盘驱动器配置相匹配的CDP盘驱动器，当业务盘发生写操作（W）时，在driver层次复制写操作，并保存到CDP数据盘中。
- 备份数据的写入操作与业务盘中的数据更新操作几乎是同步进行的
- CDP盘中的备份数据需要建立专用的数据组织结构，以加快数据的写入过程，并提供与之对应的数据检索方法。
- CDP盘中保存了所有版本的写操作数据，随着时间的延长，数据存储量会急剧的增长。


用户层次
Application
R/W
内核层次
FS/DB
R/W
驱动器层次
Disk Class Driver
Local Disk Driver
CDP Disk Driver
R/W
W
Append
CDP Log
业务盘
CDP盘

高频数据的检索与恢复

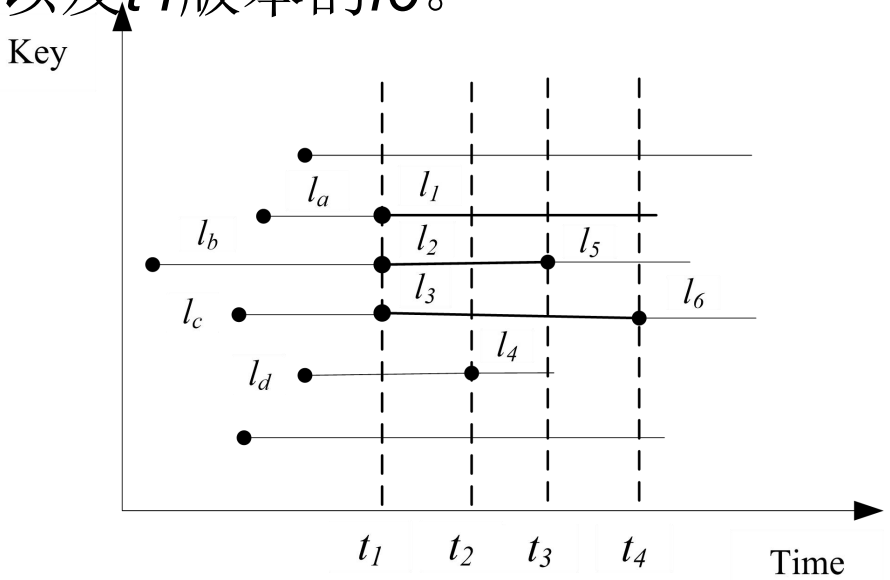
- 在快照、CDP等高频备份数据管理中面临两个基本问题：
  - 数据检索效率问题
  - 数据一致性问题
    - 引入检查点技术
    - 在结合检查点的数据恢复过程中，通常采用前滚或后滚的检索方式，逐渐接近目标点，实现面向应用一致性的数据恢复。

- 二维坐标表示高频数据的时空二维属性分布，纵坐标表示数据的空间分布特性，以Key为标示；横坐标表示时间属性，以Time为标示。



- 基本的索引结构为：<*key*, *starttime*, *endtime*, *info*>。其中*key*表示数据项与时间无关的检索标识符；数据的有效时间是[*starttime*, *endtime*)的闭开区间内；*info*表示索引项记录具体内容。
- *key1*在*t1*时刻开始，产生索引项为：<*key1*, *t1*, *, *info1*>。其中“*”表示数据的生命周期尚未结束。在*t2*时刻有Key=*key2*的索引项产生，表示为：<*key2*, *t2*, *, *info1*>。*key1*在*t3*时刻有新版本数据产生，旧版本的结束时间设为*t3*，索引项变为：<*key1*, *t1*, *t3*, *info1*>，表示旧版本数据生命周期结束，新版本数据生命周期开始，新版本索引项为：<*key1*, *t3*, *, *info2*>。

- 前向检索：从一个可恢复状态点开始依次根据数据的产生时间（starttime）进行检索。假设*t1*是前向检索的起点，则前向检索过程为：*t1*版本的*l1*，*l2*，*l3*；*t2*版本的*l4*；*t3*版本的*l5*以及*t4*版本的*l6*。



前向检索很简单

后向检索就是从最后一个到他前面一个

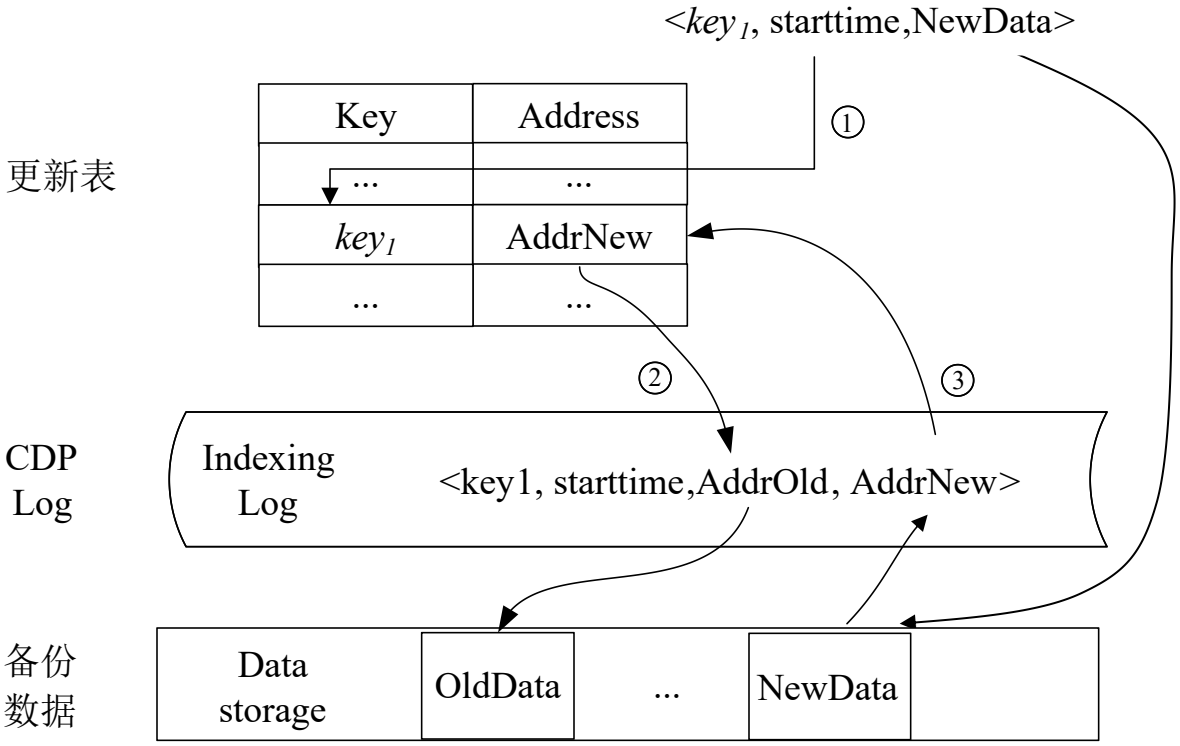
- 后向检索：从当前状态根据索引项的结束时间（endtime）依次检索的过程。假设从当前时刻*t4*开始执行状态回滚，检索顺序为：*t4*版本下的*l3*；*t3*版本下的*l2*，*t2*版本下的*ld*，以及*t1*版本下的*la*，*lb*，*lc*。

- 问题：
  - 比如在ROW等数据复制技术中，通常根据数据的产生时间依次存储所有版本的备份数据，满足starttime有序。可以有效支持RF检索，但是索引项的endtime之间是乱序的，在执行RB检索时可能需要遍历所有版本的备份数据才能获得一份endtime有序的索引项排列。在COW等数据复制技术中，复制被修改前的旧版本数据并按序保存。备份数据版本之间endtime有序，可以直接支持RB检索。但是索引项之间的starttime是乱序的。
  - 因此在高频备份数据管理中，目前的索引结构无法同时支持两种检索方式。

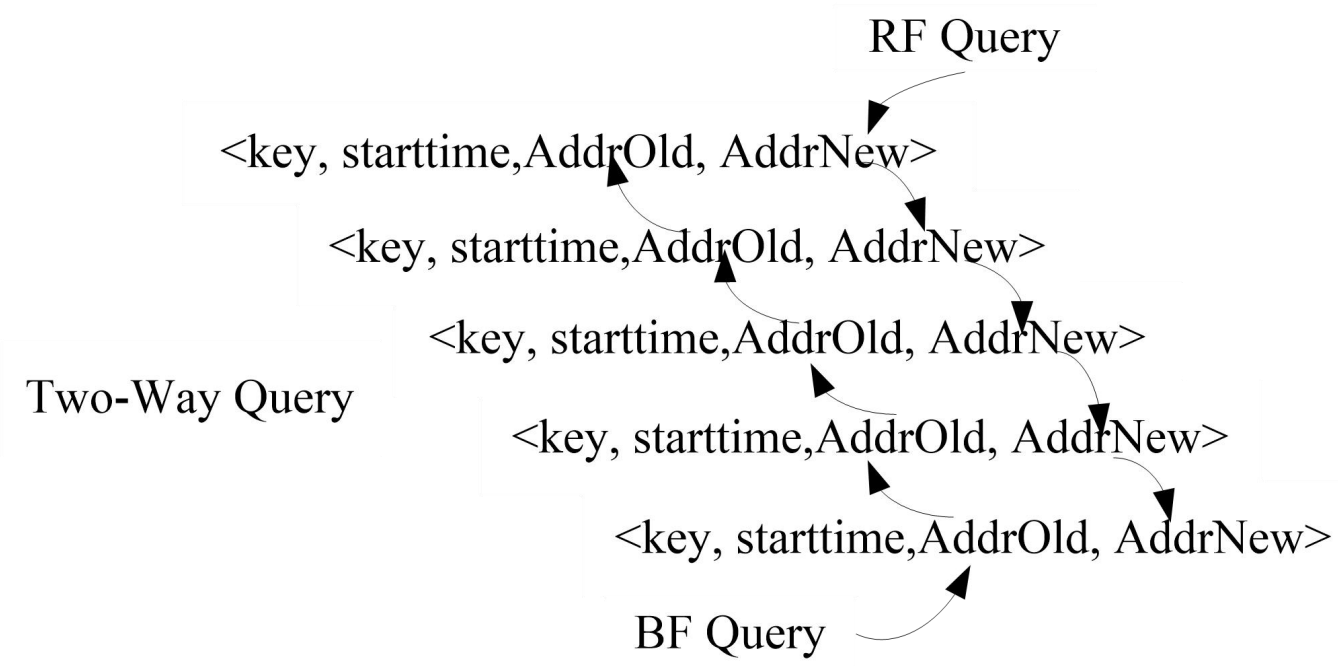
双向检索索引结构设计

- 引入双向索引（Indexing Log）和更新表（Updating Table）两种基本结构
- Indexing Log索引项格式：
  - *<Key，Time，AddrOld，AddrNew>*
  - *key*表示与时间独立的检索关键字
  - *Time*是新版本数据的开始时间，同时也是对应旧版本数据的结束时间
  - *AddrOld*：旧版本数据的存储位置指针
  - *AddrNew*：新版本数据的存储位置指针
- 更新表中：记录到当前为止所有更新数据最新版本的索引结构，基本索引项为*<Key, Address>*。
  - 更新表主要作用是支持旧版本数据存储位置的快速查找

- 当有新数据写入时，根据key值首先在更新表中查找与key对应的旧版本数据的存储位置，设为AddrNew1。在Data Storage中为新数据分配存储空间，新开辟地址为AddrNew2。产生一条双向索引记录：其*AddrOld*值为AddrNew1；*AddrNew*值为AddrNew2。索引项以按序追加方式保存在Indexing Log中。最后把AddrNew2更新到更新表中对应的记录。



- Indexing Log中的前向(redo)和后向(undo)检索



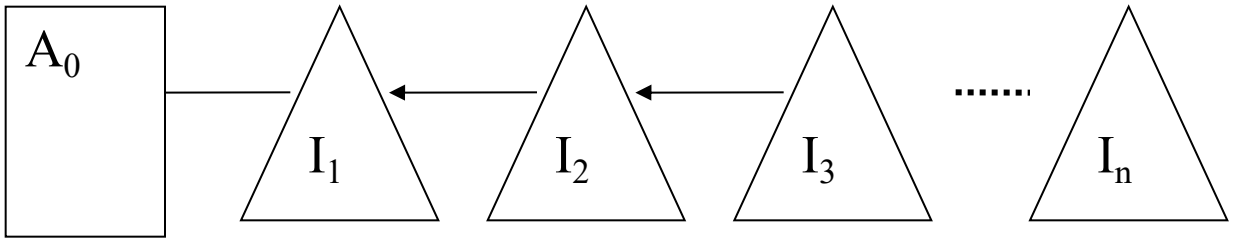
索引融合技术

- 需求
  - Log项太多
  - 恢复时的操作太多，效率低
- 能否去除冗余项？
  - 索引融合技术

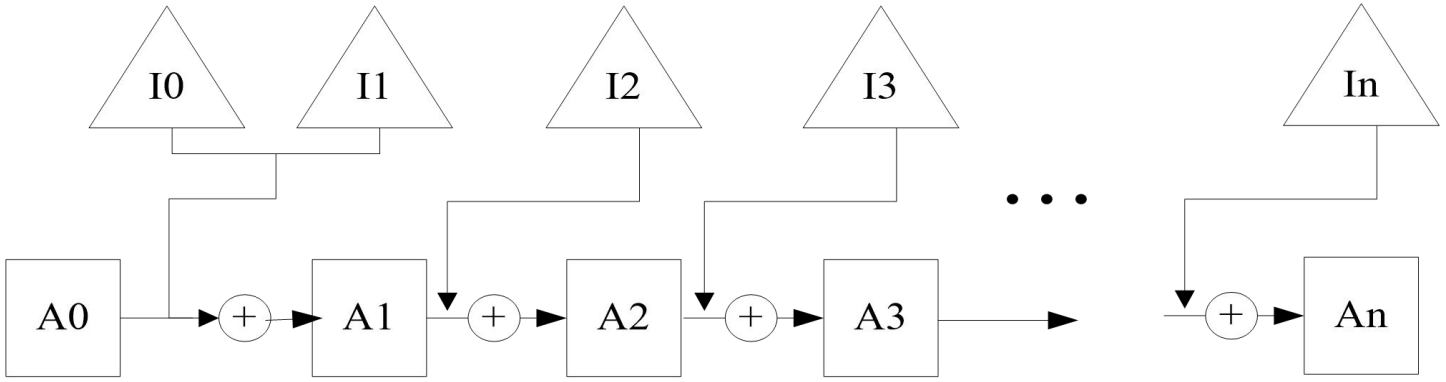
直接保留了在备份过程中产生的Mapping Log之间的依赖关系，具有最好的存储效率，但是随着备份点的增加，Log Chain的长度呈线性的增长，具有最低的检索效率。

- **Log Chain**
  - 无环、无分支有向图描述不同备份点索引之间的依赖关系。
  - 每次差量备份，差量备份产生的Log索引项作为图中的**节点**，索引之间的依赖关系构成图的**边**，能够反映备份数据完整镜像的索引文件称为图的**源**，把这种由索引之间依赖关系构成的图称为**Log Chain**

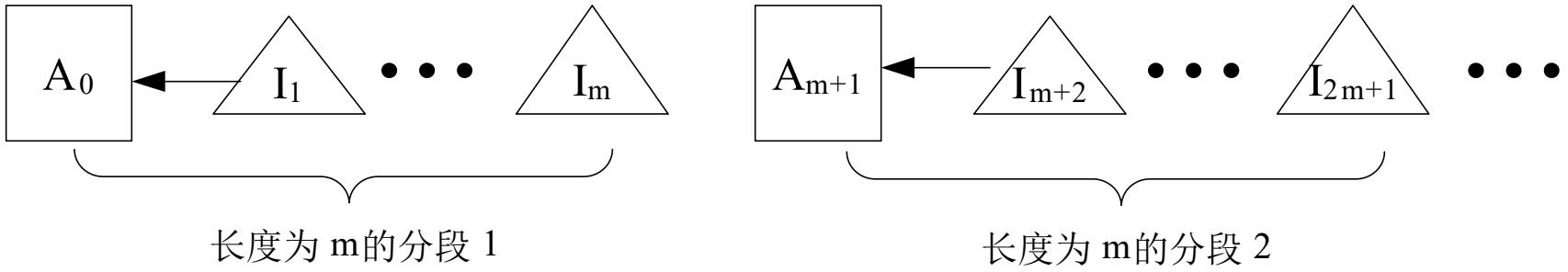
- 间接依赖关系管理



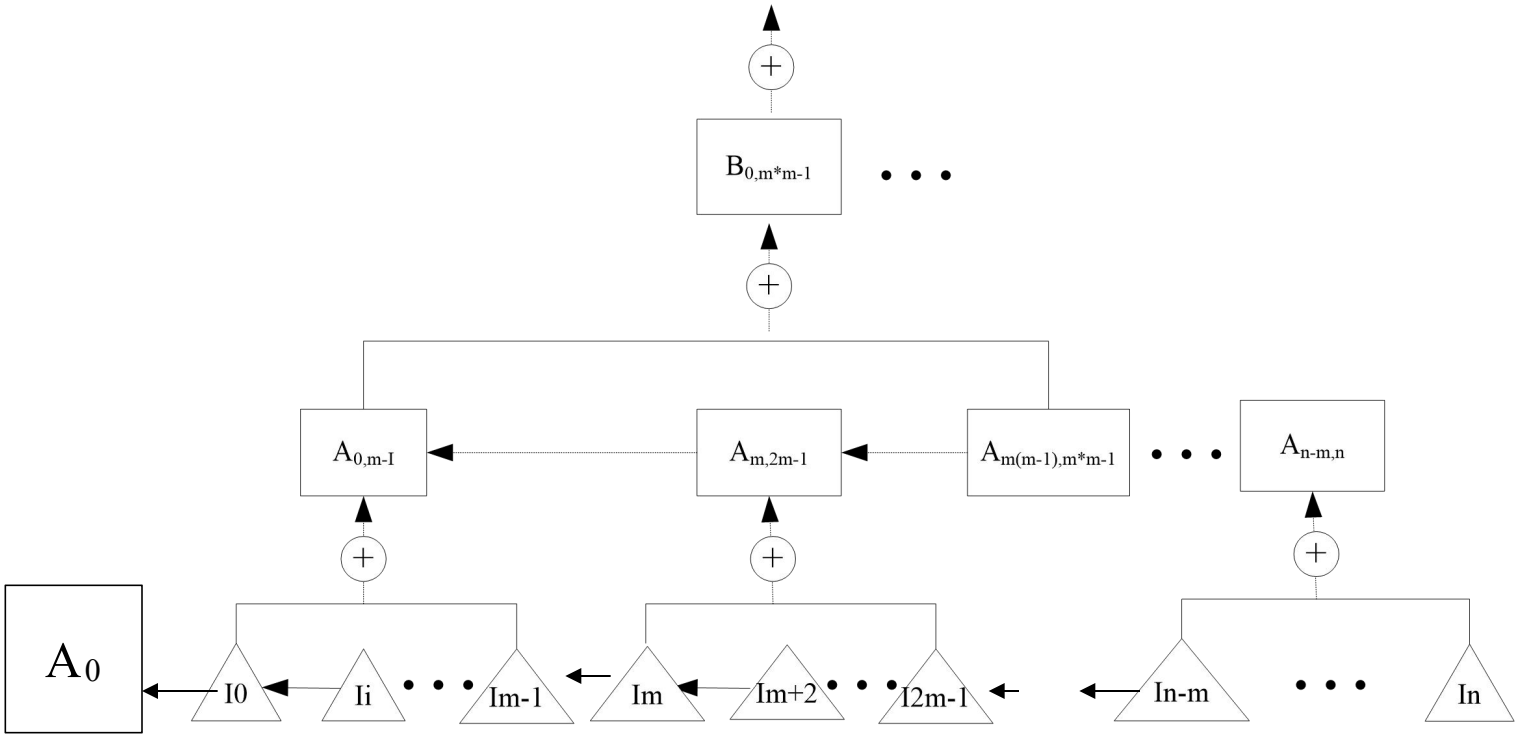
- 直接依赖关系管理

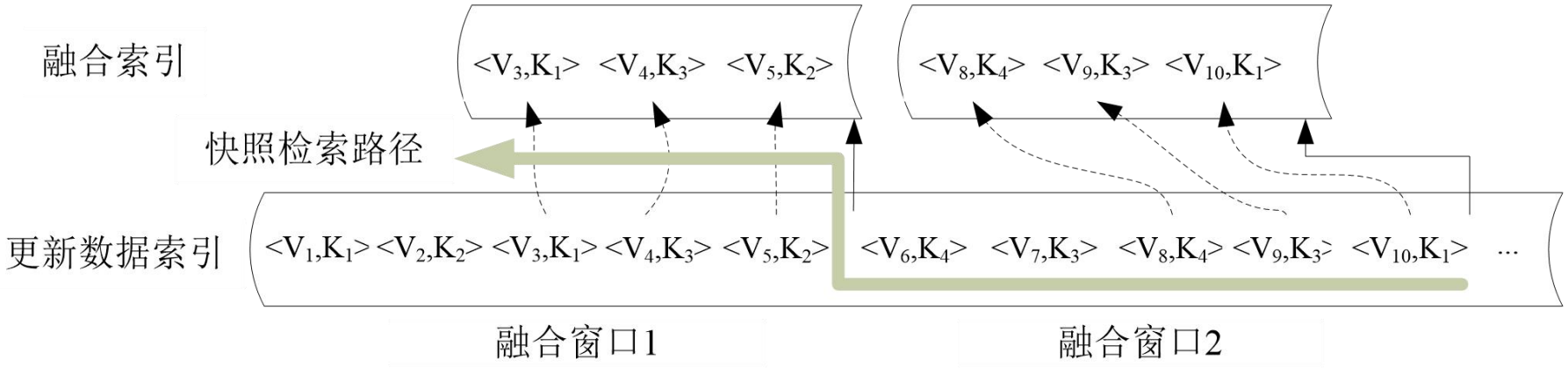


- 分段依赖关系管理

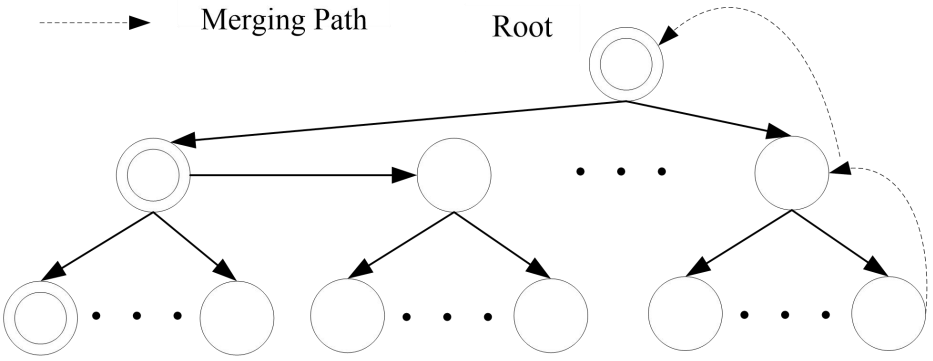


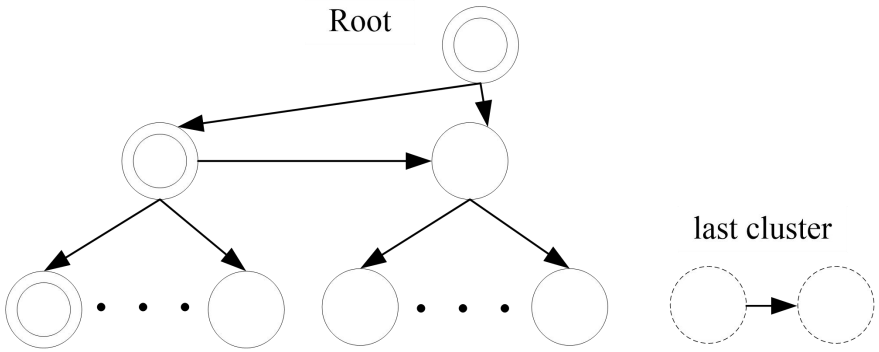
- 分段、分层叠加索引结构：关键是叠加算法
  - 索引融合：累积相关索引数据的变化过程



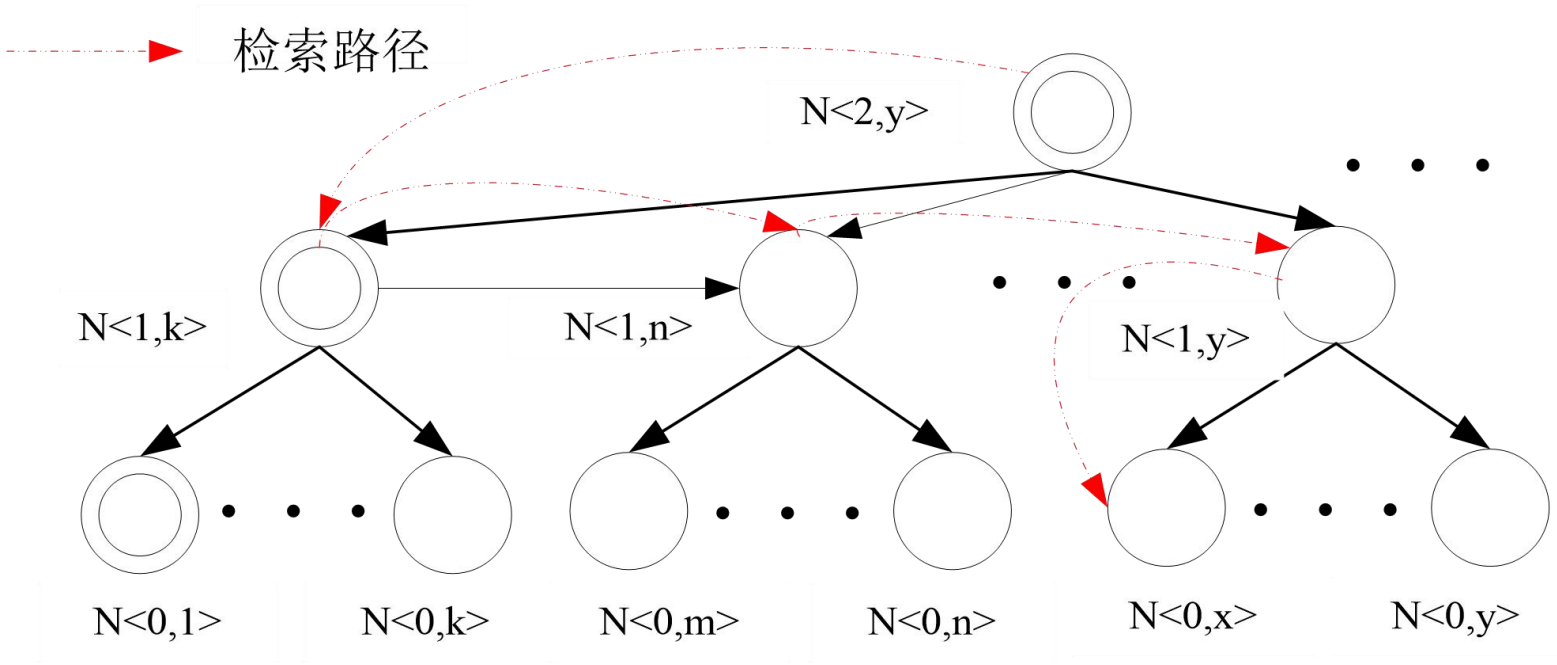

融合索引
<V3,K1> <V4,K3> <V5,K2>
<V8,K4> <V9,K3> <V10,K1>
快照检索路径
更新数据索引
<V1,K1> <V2,K2> <V3,K1> <V4,K3> <V5,K2>
<V6,K4> <V7,K3> <V8,K4> <V9,K3> <V10,K1> …
融合窗口1
融合窗口2

- 最后一个分段融合操作
  - 每当有新版本到达时直接向上层进行索引融合，一直到达根节点为止
  - 只有当融合窗口达到预定的窗口值时，才向上层融合索引





- 叠加索引检索
  - 分段、分层叠加索引上层索引结构反映了下层的索引的数据累计变化，在计算某个时间点的叠加索引时，首先从上层开始，逐层向下，最后检索一个分段内部的Log Chain

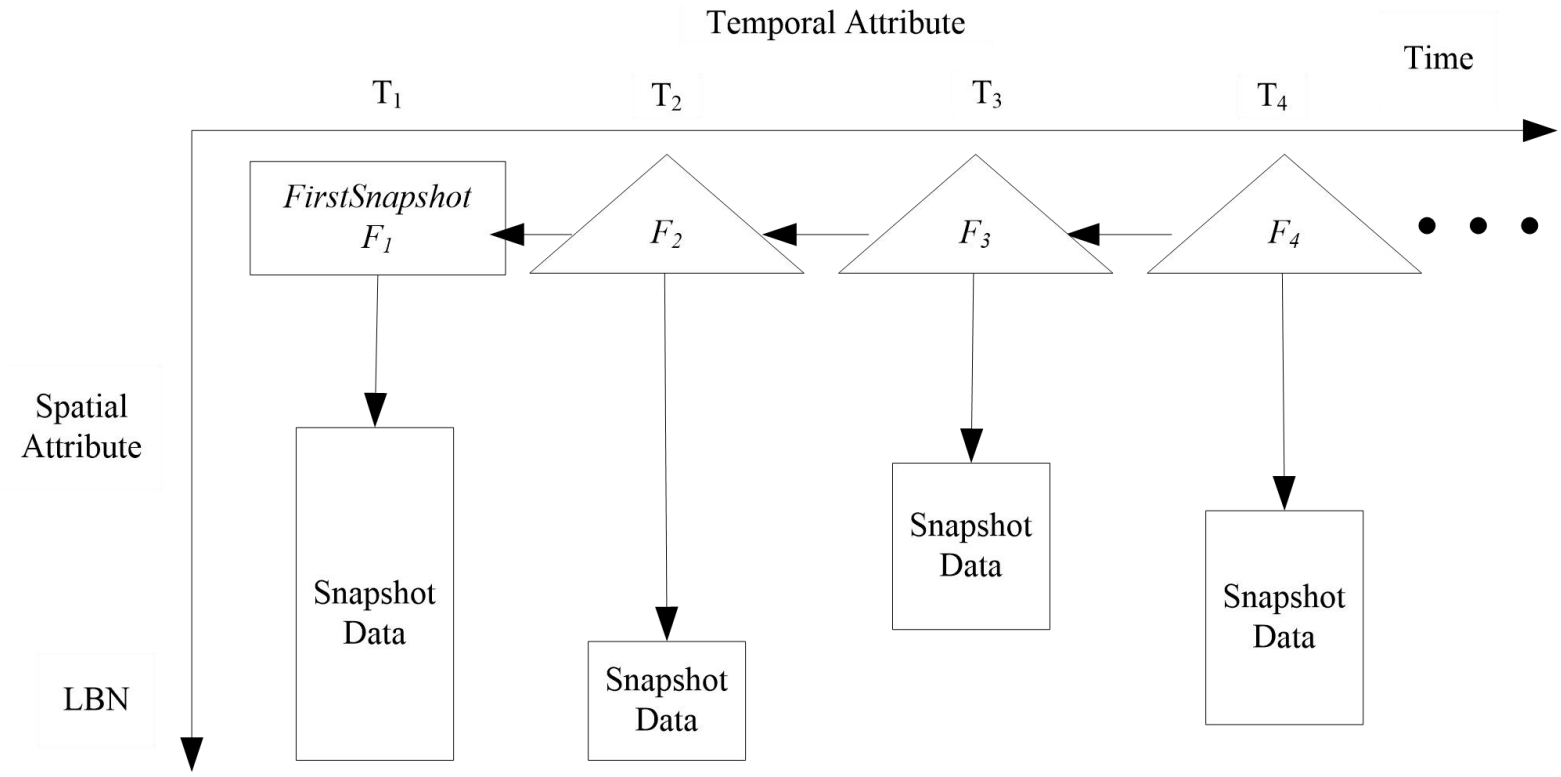

检索路径
N<2,y>
N<1,k>
N<1,n>
N<1,y>
N<0,1>
N<0,k>
N<0,m>
N<0,n>
N<0,x>
N<0,y>

管理模式	空间复杂度	Log Chain 的长度
间接依赖关系管理	$S = O(\sum_{i=0}^{n} l_i)$	$Length_{chain} = n$
直接依赖关系管理	$S = O(\sum_{i=0}^{n} li + \sum_{i=0}^{n}\sum_{j=0}^{i} l_j)$	$Length_{chain} = 1$
分段依赖关系管理（分段长度为 m）	$S = O(\frac{n}{m}\sum_{i=0}^{n} l_i)$	$Length_{chain} \le m$
分段，分层依赖关系管理（分段长度为 m）	$S = O(\log_m n \sum_{i=0}^{n} l_i)$	$Length_{chain} \le \log_m n + m$

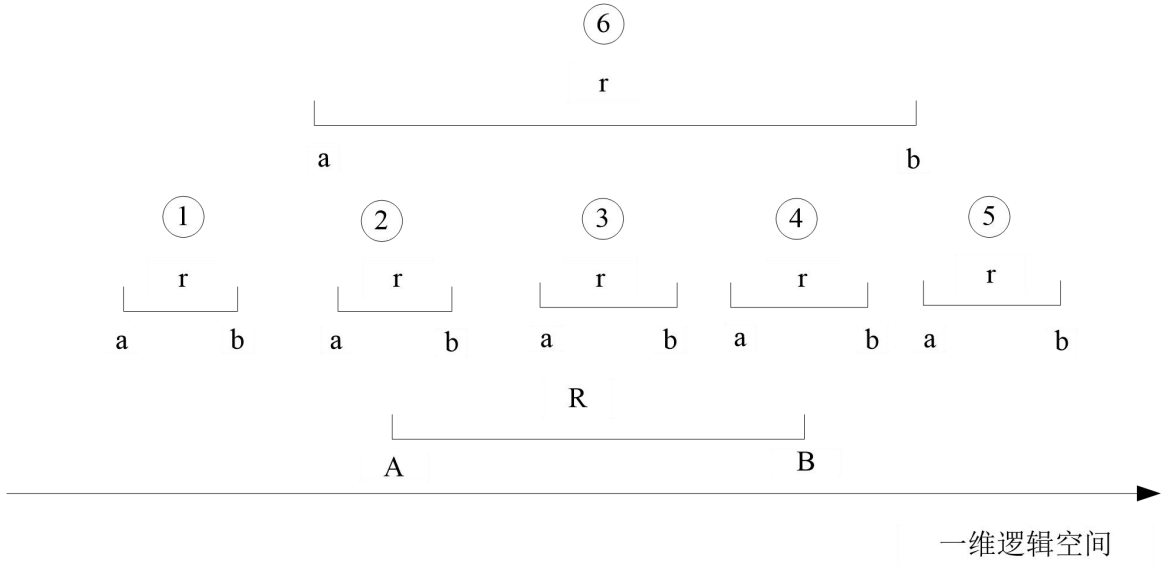
- 分层、分段依赖关系管理算法的特点
  - 收敛的
    - 在m值不断增大时，算法收敛于间接依赖关系管理模式；
    - 算法收敛于直接依赖关系管理模式；
  - 此外，在分段、分层叠加索引结构，还可以进一步引入启发式的索引保存策略，如只对当前的一个分段内部保存底层的Mapping Log，支持Undo操作，而对于早期的备份数据，有选择的保存上层叠加索引，支持Redo操作，这样不仅符合备份数据的恢复特征，同时可以进一步提高元数据的存储效率。

变长数据管理技术

- block-level快照备份
  - 基于Block的增量快照，其数据可能是不规则长度的数据块
    - 增量备份的数据块之间存在着更复杂的关系
    - 需要新的索引和融合方法


Temporal Attribute
Time
T1
T2
T3
T4
FirstSnapshot
F1
F2
F3
F4
Spatial
Attribute
Snapshot
Data
Snapshot
Data
Snapshot
Data
Snapshot
Data
LBN

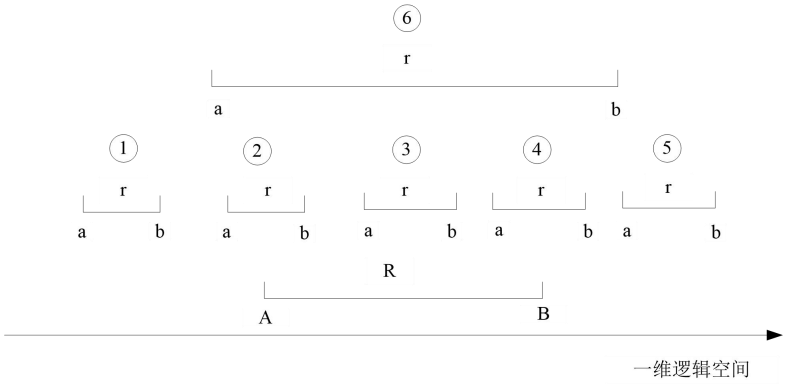
- 数据块之间的关系
  - 其中R为旧的数据、r为新的数据



变长数据块索引

- 基本索引结构为：<*Interval*, *Value*>
  - Log Chain
  - 其中*Interval*为一个连续的数据块区间，如区间为 [*a*, *b*]，表示区间范围为地址*a* — *b*
  - 两个区间的相对关系需要在一维空间上进行比较，设源区间为*r* = [*a*, *b*], *a*≤*b*；目标区间*R* = [*A*, *B*]，*A*≤*B*；源区间*r*相对于目标区间*R*的相对关系可以描述。

– 区间关系描述



– 设源区间为$r = [a, b]$, $a \le b$；目标区间$R = [A, B]$, $A \le B$；r与R的关系可概括为6种关系，两种运算

» 左独立①, 左重叠②，包含③，右重叠④，右独立⑤，覆盖⑥；

- 区间索引关系运算逻辑（索引融合）
  - **1**、在满足*Overlapping*关系条件下,即*r*覆盖*R*时,*R*中的点在*r*中都有与之对应的部分,使用*r*替代*R*的过程称为区间叠加,记为:***r+R***;
  - **2**、在区间关系*LeftOverlapping*, *Included*, *RightOverlapping*条件下,把*R*进行分割,产生的区间子项*Ra*,*Rb*,*Rc*其中*Ra*特指*r*与*Ra*具有*RightIndependent*的区间关系;*r*与*Rb*具有*Overlapping*关系,*r*与*Rc*具有*LeftIndependent*的区间关系,把这一过程称为区间分割,记为:***R/r***;
  - **3**、在区间分割运算中,如果分割*R*产生的子区间*Rb*不为空,即在*R*中获得与*r*相匹配的子区间的过程称为区间相减,记为:***R-r***.

- 区间运算的物理意义：在*r*与*R*具有区间关系左独立和右独立的情况下,直接写入新版本数据块索引;在具有覆盖关系情况下,使用新版本数据索引完全覆盖掉旧版本数据索引;而重叠或包含关系时有部分旧版本的数据索引被覆盖掉.

多版本数据备份管理技术

- 背景
  - 多版本技术决定着数据的存储和恢复效率
    - 备份数据长期存储过程中需要结合备份数据的多版本管理技术检索数据
    - 决定备份数据检索效率的因素包括版本内索引数据和遍历版本数目，如何减少遍历过程中的索引数据量是提高多版本管理效率的主要途径
  - 问题描述
    - 传输效率效率
    - 版本管理效率
    - 版本的删除

多版本管理方法

- 多版本管理方法
  - 假设多次备份后产生的版本序列记为：$S=\{F_1, F_2, \ldots, F_n\}$
  - 版本融合算法
    - 根据索引融合或区间索引运算逻辑.
    - 把版本融合过程使用算符“⊕ ”表示, 版本融合过程可以表示为: $F_t=F_r \oplus F_R$. 也把版本融合称作版本叠加
      - » $F_{FullSnapshot}=F_i \oplus F_{i-1} \oplus \ldots \oplus F_2 \oplus F_1$
    - 版本融合是版本删除的基础运算
      - » 版本删除基本思想是把待删除版本与下一版本融合，融合结果保留了可能被将来版本共享的数据

它首先计算一个版本的全快照索引，然后在端侧生成校验文件，并在主端侧进行校验以发现错误。一旦检测到不一致，它将生成一个包含错误的数据块地址的文件，并将其发送回端侧进行恢复。

– 多版本差异恢复方法：diffdo

- Step1. $F_{FullSnapshot}(T)=F_T \oplus F_{T-1} \oplus \ldots \oplus F_1$;
  计算版本T的快照索引: $F_{FullSnapshot}(T)$;
- Step2. Check($F_{FullSnapshot}(T)$)→CheckFile(T);
  从端根据 $F_{FullSnapshot}(T)$计算快照数据的校验文件: CheckFile(T)，并发送到主端;
- Step3. CheckFile(T)→CheckErrorFile(T);
  主端根据CheckFile(T), 记录校验不一致的数据块对应的逻辑地址, 生成CheckErrorFile(T), 并发送到从端;
- Step4. CheckErrorFile(T)+$F_{FullSnapshot}(T)$→DiffdoLog(T);
  存储端根据CheckErrorFile(T)和$F_{FullSnapshot}(T)$检索备份数据, 计算差异恢复索引文件DiffdoLog(T), 并根据DiffdoLog(T)进行差异数据恢复。

- 小结
  - 数据保护技术可以解决两类错误：逻辑错误、物理错误
  - 是密集的写应用环境，对索引结构的更新效率、存储效率、检索效率都有很高要求
  - 随着数据复制频率提高，产生数据量急剧增长，版本之间依赖关系复杂，影响数据可恢复性和恢复效率

分布式存储技术

- 分布式存储
  - 对象：大量数据，变化频率较低，对恢复粒度要求较低
- 数据的冗余存储
  - 数据完全复制
    - 采用了完全复制(Complete replication)的方式实现数据容灾机制。如分布式存储系统PAST 、Freenet等
  - 编码技术：如磁盘阵列技术(RAID Redundant Array of Inexpensive Disks) 。
    - 将多个独立的磁盘组织成一个逻辑盘，通过数据分割、多通道并行来提高数据的I/O速率，并且通过保存冗余的数据、校验信息来提高存储的可靠性。
- 关键问题
  - 数据格式，目的：通过合理的额外存储来提供高可靠性和可用性
  - 数据的存储结构，以及如何索引

数据副本存储

- 完全复制
  - 就是通过将文件的多个副本分布到系统中不同节点，实现冗余容错，文件副本越多，数据的可用性越好，可靠性越高。完全复制不涉及编码运算，文件创建和读取不需要编解码操作，读写效率高，容错性能较好。占用空间较大
    - 完整的副本
    - 分割的副本

数据（文件）分割

- 在全备份方式的基础上引入分割的理念
- 定义
  - 一个四元组(m, n, b, r)，设其编码函数为E，解码函数为D，对于消息M = (M_1, M_2, …, M_m)，其中M_i(1≤i≤m)为大小为b bits 的消息包，编码后的消息E(M) = (M'_1,M'_2, …, M'_n)，其中M'_i(1≤i≤n)大小仍为b bits。设E(M)’为E(M)中任意r(r≥m)个包组成的子消息，则D(E(M)’) = M，即对E(M)中任意r个消息包进行解码就可以得到原始消息。
  - (m = 1, n = 2)的分割编码相当于完全备份（这里其实r必然等于1，否则不是全备份），RAID 5 则可以描述成(m = 4, n = 5)的分割编码系统。
- 目的：通过合理的额外存储来提供高可靠性和可用性

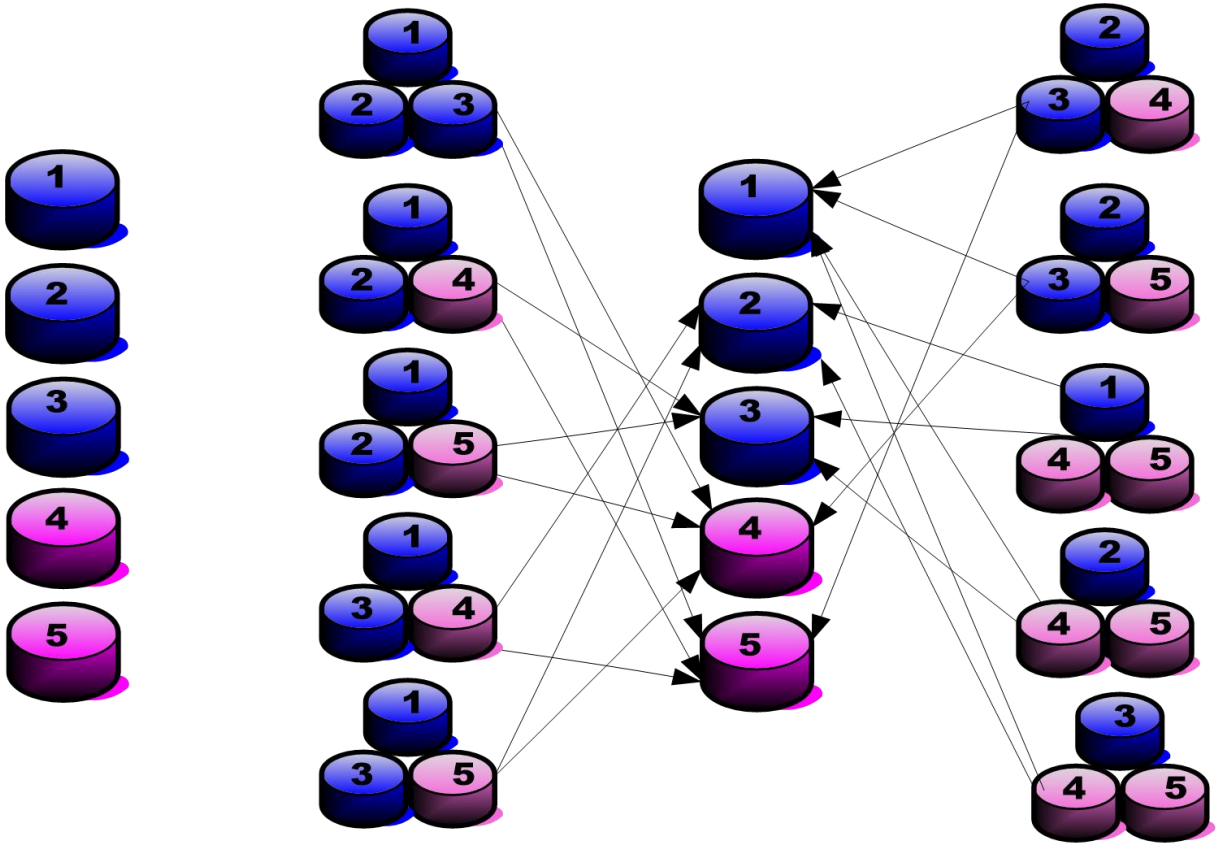
数据分割方法

- 文件碎块分割方法
  - 思想：将文件分割成碎块，然后对碎块进行组合、冗余，生成多个文件碎片。
    - 一个文件分割成m个碎块，将m个碎块组合、冗余生成n个碎片文件，每个碎片文件包含k个碎块。达到即使丢失r个碎片文件，剩下的n-r个碎片文件也可恢复一个完整的文件
    - 原始具有m个碎块的文件，分割成n个具有k个碎块的文件。占据n*k的存储空间或者m*(r+1)
    - （每个原始碎块都会在r+1 个不同的碎片文件中出现以便恢复）

数据（文件）编码分割方法

- Erasure code编码方法：基于Vandermonde矩阵的Reed-Solomon算法
  - Erasure code 作为一种FEC (Forward Error Correction)技术主要应用在网络传输中避免包的丢失，利用它来提高存储可靠性
  - 将要存储在系统中的文件分割成m 块，然后对其编码得到n个纠错块，得到m+n文件块，只要存在r个可用的文件碎片，满足：r≥m就可以重构得到原始文件。因此即使有m + n - r个文件碎片丢失，也仍然可以恢复出原来的整个文件。其中，r越大冗余度越小，r越小冗余度越大。

- 将一个文件分割为3块（标号为1、2、3的圆柱体），生成2个校验块（标号为4、5的圆柱体），总共得到5个文件碎片，则只需得到其中任意3个文件碎片即可重构得到原始数据。



SQL or NoSQL

- SQL库表时代
  - 随着用户增长，将会出现的问题
    - 查询压力过大
  - 通常的解决方案
    - MySQL replication及主从分离
  - 用户数会继续增大，超出单表写的负载
    - 单表数据库出现瓶颈，读写效率过低
    - 分库分表
  - 性能低

NoSQL的优势

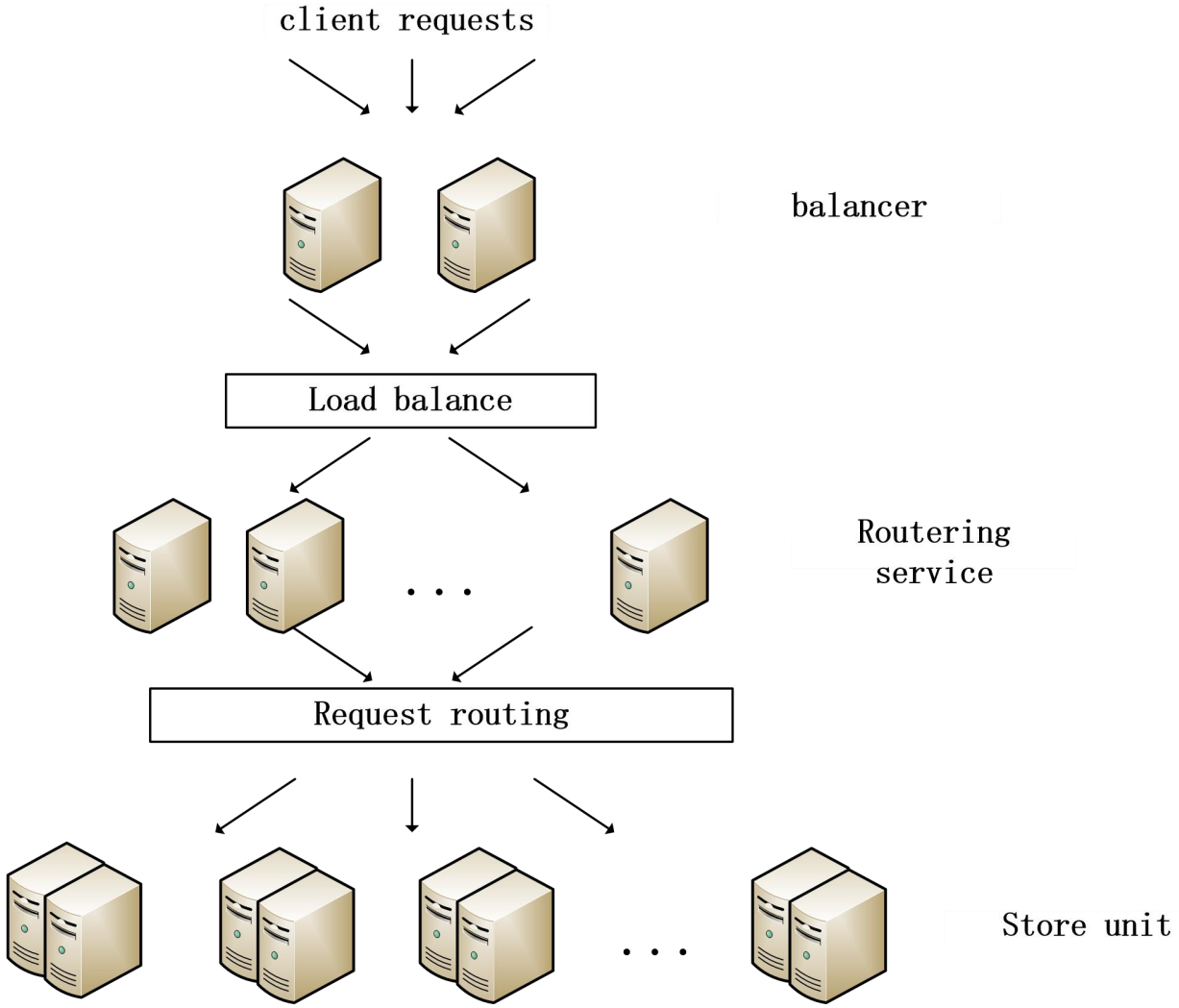
- 高性能
  - 比MySQL快1个数量级以上（oracle比mysql更慢）
- 可扩展性好
- 简洁性
- 便宜
  - 相比于oracle等商用软件
- 可维护性好

NoSQL现状

- 各大互联网公司都有自己的nosql产品，大多数为私有服务
  - Google，bigtable
  - Amazon，dynamo
  - Yahoo，PNUTS
  - Taobao，tfs、oceanbase
  - Baidu，bailing
- 开源产品众多，良莠不齐

- Key-value的分布式存储
  - Simple storage
  - 满足一类业务的kv存储需求
    - 用户管理
      - 腾讯游戏用户管理
    - 长期数据存储
      - Amazon S3 (Simple Storage Service)，在线存储服务
      - 百度：图片、音乐的存储，在线存储
      - 阿里：在线存储

- 架构图


client requests
balancer
Load balance
Routering service
Request routing
Store unit

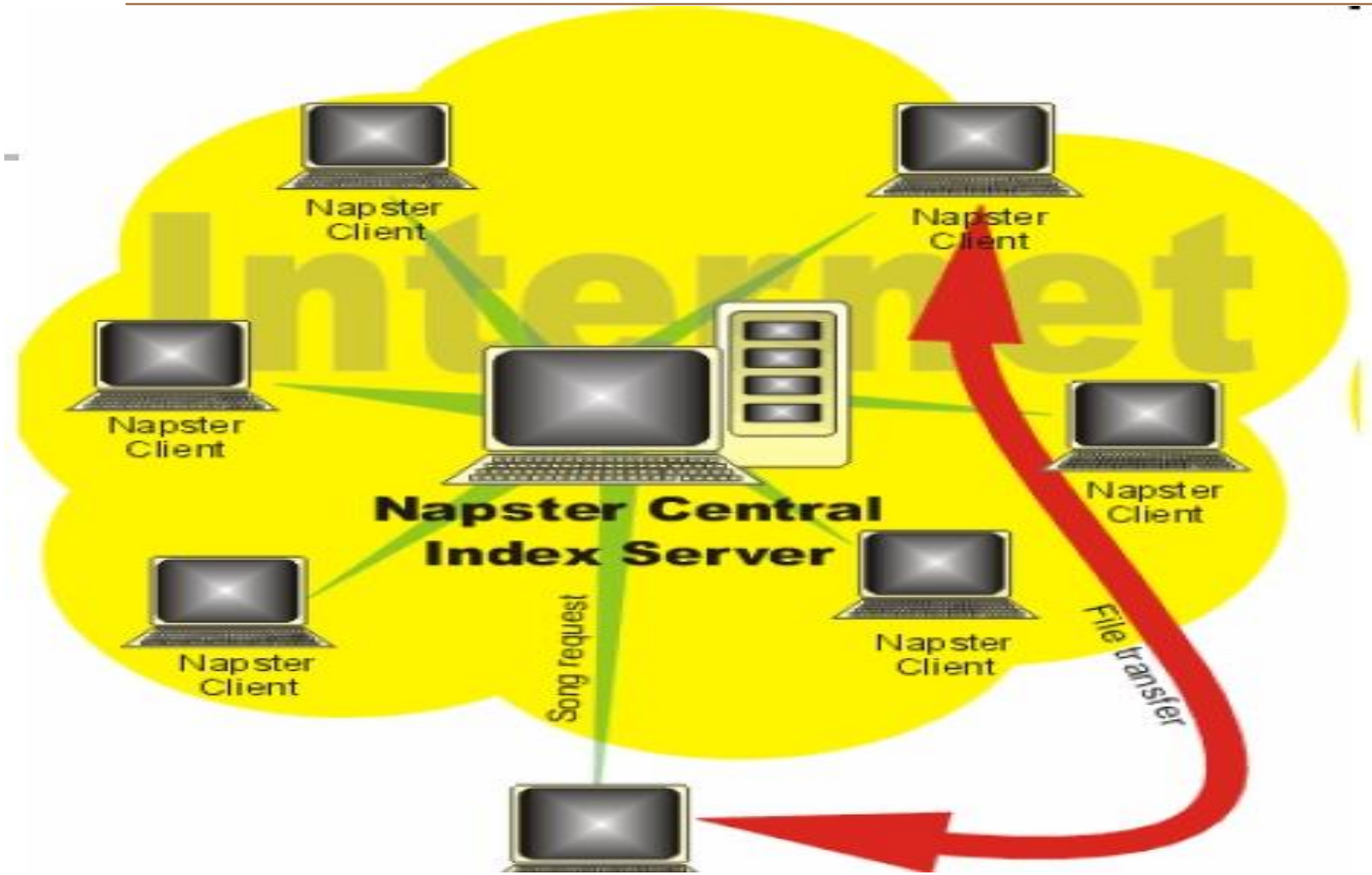
- 架构说明
  - 负载均衡模块
    - 万兆光纤接入
    - 健康检查及负载均衡
    - Round robin策略将请求分发到任一台request router
  - 请求路由模块
    - 一致性哈希
    - 将请求精准分发到所属store unit
  - 存储单元模块
    - 数据存储实体
    - 单元内的容错及数据备份
- 对外提供HTTP接口
- 弱一致性

- 负载均衡模块
  - 商业HA软件、开源LVS方案
  - 小集群可省略此模块，与routing模块合并
- 请求路由模块
  - 每个routing server配置与功能都一致
  - 可根据集群规模routing server可随意增减
  - 对store unit实行一致性哈希策略

- 存储单元模块
  - 各存储单元完全独立
  - 2副本使用dual-master模式
    - master-slave模式出现问题时需要人工干预
  - 增减存储单元（扩/削容）时如何操作？
    - 增加unit时需要将老unit上的数据均分一部分给新unit
    - 减少unit时需要将它的数据均分给其他unit
    - 增减unit时，短时间的服务质量下降

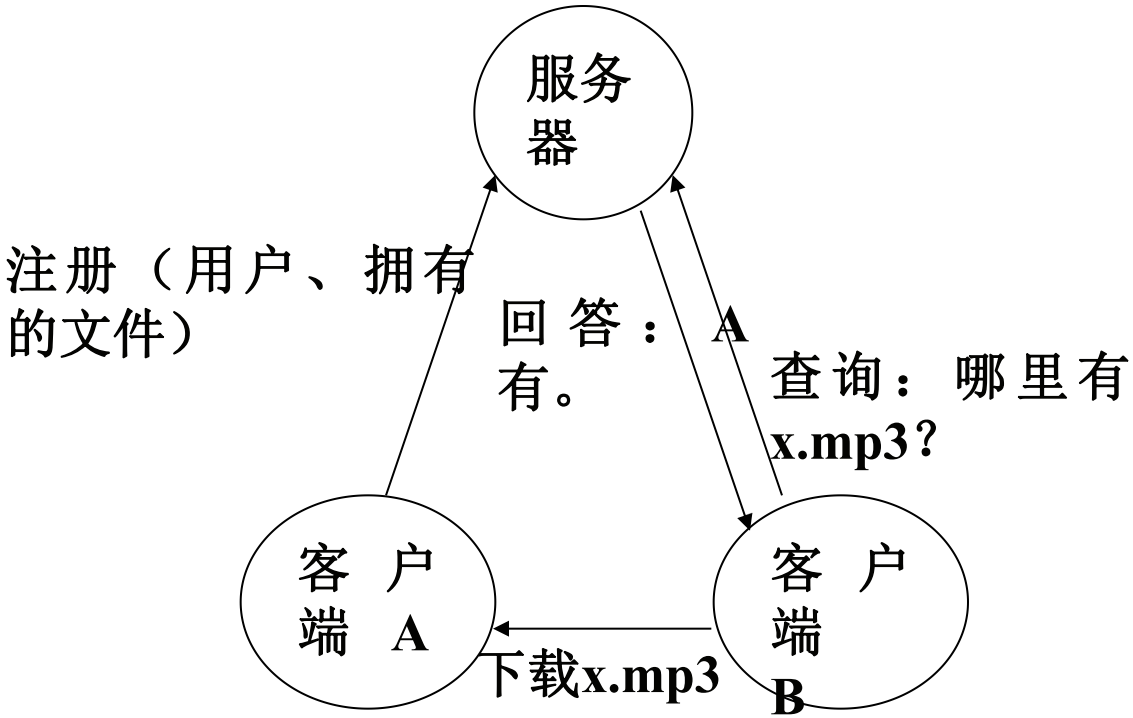
- 关键问题
  - 分布式存储的前提是找到并定位资源，这是关键，即资源的索引问题。索引问题就是对于给出的一个关键字，找到相应的文件，并且给出这些文件的位置。基于(Key,value)的索引。解决这个问题的常用的策略是建立一个Overlay Network，它是位于应用层的，在这个Overlay Network上，通过具体的分配和路由机制实现索引。
  - 在索引算法上，经历了大致三个发展阶段。
    - 集中式索引算法
    - 基于Flooding的分布式索引算法
    - 基于分布式散列表的索引算法（DHT）

- 中心化拓扑（Centralized Topology）
  - 在中心化拓扑模型中，一群高性能的中央服务器保存着网络中所有活动对等计算机共享资源的目录信息。当需要查询某个文件(key)时，会向一台中央服务器发出文件查询请求。中央服务器进行相应的检索和查询后，会返回符合查询要求的对等机地址信息列表。查询发起的机器接收到应答后，会根据网络流量和延迟等信息进行选择，和合适的对等机建立连接，并开始文件传输


Napster Client
Napster Client
Napster Client
Napster Client
Napster Client
Napster Client
Napster Central Index Server
Song request
File transfer

Napster结构

Napster不提供任何形式的加密，文件的共享和传输都是明文的。除了**Napster**本身以外，还有很多软件是遵从**Napster**的协议来实现的，如**Opennap**，**Rapster**等。

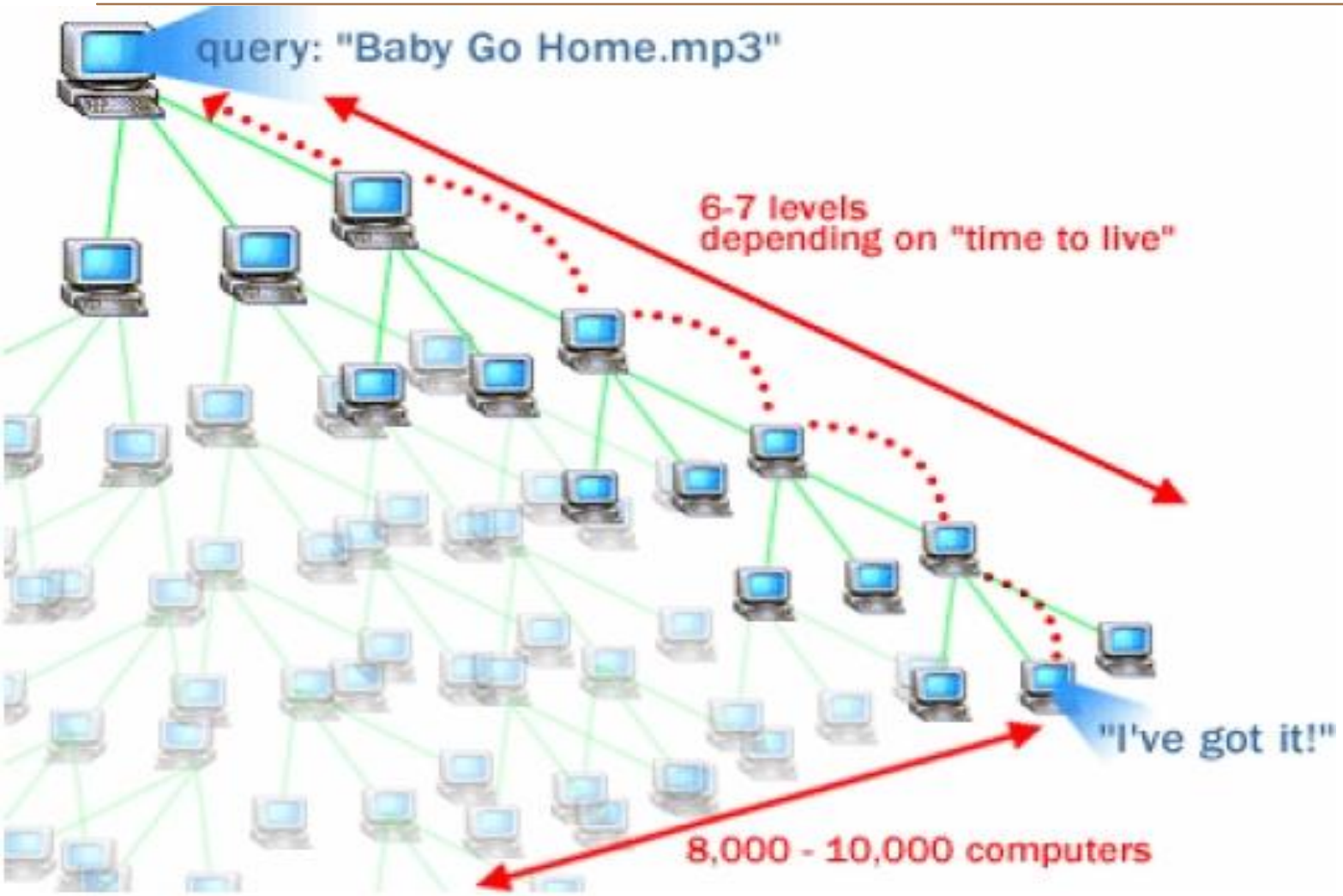


- 中心化拓扑优点
  - 由于资源的发现依赖中心化的目录系统，中心化拓扑最大的优点是维护简单发现效率高，算法灵活高效并能够实现复杂查询。
- 中心化拓扑存在的问题，主要表现为：
  - 中央服务器的瘫痪容易导致整个网络的崩馈，可靠性和安全性较低。
  - 随着网络规模的扩大，对中央索引服务器进行维护和更新的费用将急剧增加，所需成本过高。
  - 中央服务器的存在引起共享资源在版权问题上的纠纷，并因此被攻击为非纯粹意义上的P2P网络模型。
- 总结来看对小型网络而言，其集中目录式模型在管理和控制方面占一定优势。但对于大型网络应用，该模型并不适合。

最大的问题与传统客户机/服务器结构类似，容易造成单点故障，访问的“热点”现象和法律等相关问题

- 全分布非结构化网络
  - 在重叠网络（overlay）采用了随机图的组织方式，从而能够较快发现目的结点，面对网络的动态变化体现了较好的容错能力，因此具有较好的可用性。同时可以支持复杂查询，最典型的案例是Gnutella
  - Gnutella没有索引服务器，它采用了基于完全随机图的洪泛（Flooding）发现和随机转发（Random Walker）机制。为了控制搜索消息的传输，通过TTL (Time To Live)的减值来实现。

- Gnutella机制
  - 在Gnutella分布式对等网络模型中，每一个联网计算机在功能上都是对等的，既是客户机同时又是服务器。
  - 系统中有这样三类消息，成员关系消息(Ping, Pong)，查询消息(Query, Response)，文件下载消息(Get, Push)。Gnutella Network的建立依靠成员关系消息。
    - 当一个结点加入到Gnutella中时，它广播一个Ping消息，告诉其它结点它的存在。当某个结点接受到Ping消息时，它一边返回Pong消息给新节点，一边forward这个Ping消息给其它结点，同时在本地节点列表中记录新节点的信息。新节点根据收到的Pong消息获得一张结点列表。
    - 当一个结点发送查询时，它将Query请求Flood到本地节点列表中的相邻结点，相邻结点收到Query后，在本地查看是否有符合该请求的查询，有的话将文件信息Response给请求结点。查看同时也同样地将Query请求 Flood出去，直到TTL（time-to-live）值减少到0。从而Query就这样在Gnutella网络中散播开来。



最初的Gnutella采用的Flooding搜索算法示意图

Gnutella机制分析

- 优点
  - 没有中央服务器，容错性好，支持复杂的查询。能较好的处理动态结点加入和退出，以及节点故障问题。
- 缺点
  - 这种Flood的方法产生了大量的消息，占用网络带宽，影响了系统的性能。从而导致网络中部分低带宽节点因网络资源过载而失效。
  - 由于非结构化网络将重叠网络认为是一个完全随机图，结点之间的链路没有遵循物理的拓扑来构建。没有确定拓扑结构的支持，非结构化网络无法保证资源发现的效率。即使需要查找的目的结点存在发现也有可能失败。查询的结果可能不完全，查询速度较慢。
  - 由于采用TTL（Time-to-Live）、洪泛（Flooding）、随机漫步或有选择转发算法，因此索引的直径不可控，可扩展性较差。

- 完全分布式结构化拓扑网络
  - 最新的研究成果体现在采用分布式散列表(DHT) 。分布式散列表(DHT)实际上是一个由广域范围大量结点共同维护的巨大散列表。散列表被分割成不连续的块，每个结点被分配给一个属于自己的散列块，并成为这个散列块的管理者。
    - 采用新的拓扑图构建重叠路由网络，以减少路由表容量和路由延时。
  - DHT类结构能够自适应结点的动态加入/退出，有着良好的可扩展性、鲁棒性、结点ID分配的均匀性和自组织能力。最经典的案例是Tapestry，Chord，CAN，和Pastry。

- DHT的基本原理
  - DHT的核心思想是通过将存储对象的特征(关键字)经过哈希运算，得到键值(Key)，并将key的拥有权进行划分，分布到系统中的各个节点上，各个节点负责的一定量的数据key。每个节点需维护少数几个其它节点的信息，并且根据设定的路由原则，能够有效地将消息路由到任何给定key的拥有者。
    - 关键问题：由于对象的分布式存储是依据键值来进行的，因此关键在于开发相应的分布式查找协议，该协议可将指定的关键字(Key) 映射到对应的结点，并且利于索引查询
  - 每个节点只需要在网络中维护一部分其它节点的信息，通常是$O(\log N)$个节点的信息，从而当网络发生变化时，每个节点不需要执行很多的工作。

- DHT的结构
  - key空间划分机制
    - key空间划分机制将对key的所有权进行划分并且分布到所有节点上
  - 覆盖网络
    - 覆盖网络连接所有的节点并且允许节点查询给定key的拥有者。

- Key的空间划分机制
  - 引入散列函数，将key与节点进行匹配。对数据资源进行散列计算，每个数据资源被指派一个唯一的Key，每个节点有一个唯一的标识ID，ID为i的节点拥有与i最接近的所有的key，即这些Key的数据资源存储在ID为i的节点上。

- 覆盖网络
  - 每个节点维护一组到其它节点的连接，包括它的邻居或者路由表等，从而，对于任何key，节点或者拥有该key，或者拥有一个连接（路由），该连接指向根据在前面定义的key空间距离来计算离该key最近的节点。
  - 将消息转发给ID离key最近的邻居。这种形式的路由有时被称为基于key的路由。邻居是通过结构化的方式选择的，称为网络的拓扑，从而在任何路由线路中的跳数（网络的维数）和每个节点的邻居数（度数）都会比较少。

Chord协议

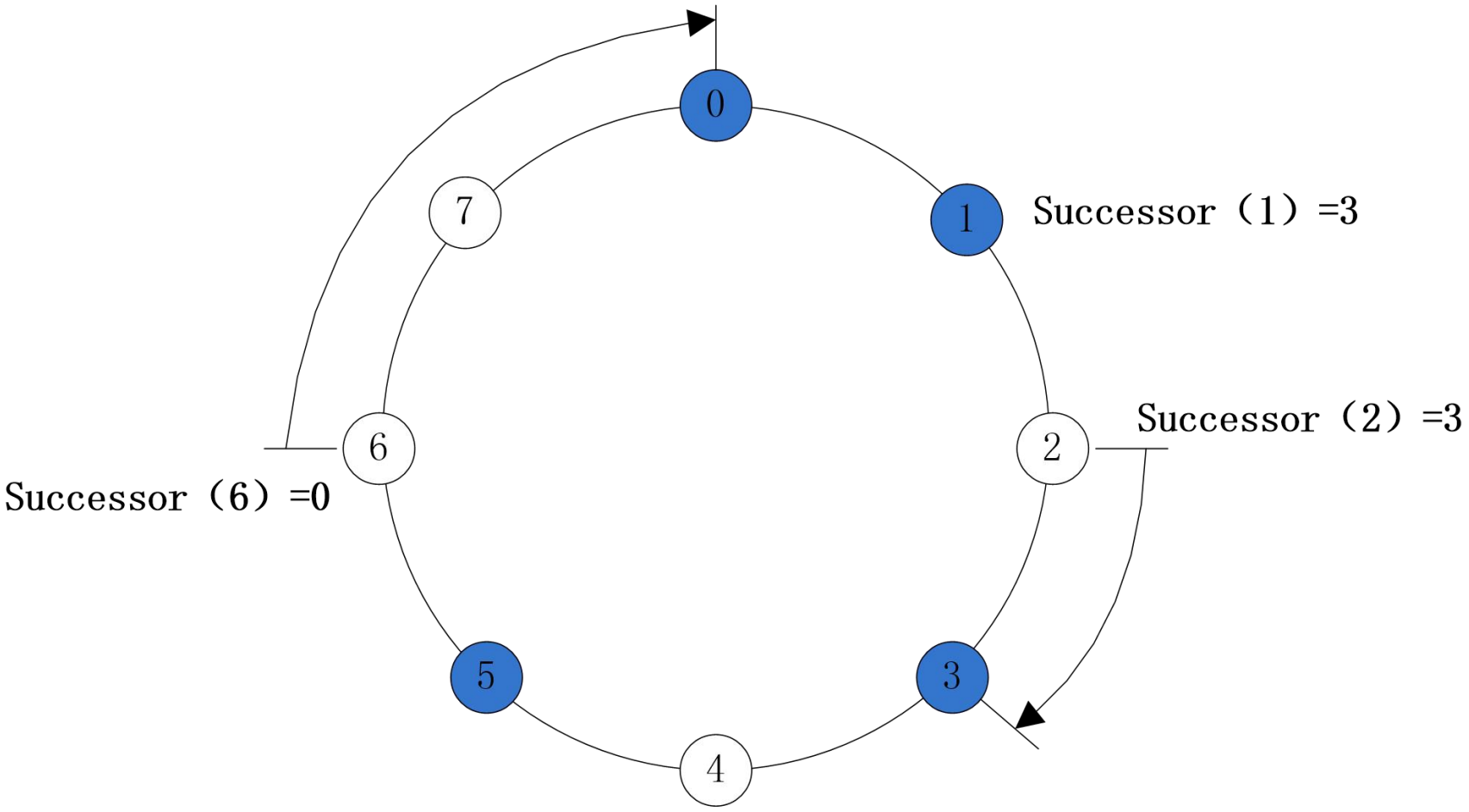
- Chord的主要贡献是提出了一个分布式查找协议，该协议可将指定的关键字(Key)映射到对应的节点(Node)。
  - Key的空间划分：一致性哈希算法
  - 覆盖网络：环状网络

- 一致性哈希
  - 平衡性(Balance)
    - 平衡性是指哈希的结果能够尽可能分布到所有的缓冲中去，这样可以使得所有的缓冲空间都得到利用。
  - 单调性(Monotonicity)
    - 单调性是指如果已经有一些内容通过哈希分派到了相应的缓冲中，又有新的缓冲加入到系统中。哈希的结果应能够保证原有已分配的内容可以被映射到新的缓冲中去，而不会被映射到旧的缓冲集合中的其他缓冲区。
  - 分散性(Spread)
    - 防止哈希的结果不一致，相同的内容被不同的终端映射到不同的缓冲区 中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。
  - 负载(Load)
    - 负载问题实际上是从另一个角度看待分散性问题。既然不同的终端可能将相同的内容映射到不同的缓冲区中，那么对于一个特定的缓冲区而言，也可能被不同的用户映射为不同的内容。与分散性一样，这种情况也是应当避免的，因此好的哈希算法应能够尽量降低缓冲的负荷。

- 简单的哈希算法往往不能满足单调性的要求，如最简单的线性哈希：
- $x \rightarrow ax + b \bmod (P)$
- 在上式中，P表示全部缓冲的大小。不难看出，当缓冲大小发生变化时(从P1到P2)，原来所有的哈希结果均会发生变化，从而不满足单调性的要求。
- 哈希结果的变化意味着当缓冲空间发生变化时，所有的映射关系需要在系统内全部更新。而在P2P系统内，缓冲的变化等价于Peer加入或退出系统，这一情况在P2P系统中会频繁发生，因此会带来极大计算和传输负荷。单调性就是要求哈希算法能够避免这一情况的发生。

  - 在分布式环境中，终端有可能看不到所有的缓冲，而是只能看到其中的一部分。当终端希望通过哈希过程将内容映射到缓冲上时，由于不同终端所见的缓冲范围有可能不同，从而导致哈希的结果不一致，最终的结果是相同的内容被不同的终端映射到不同的缓冲区中。这种情况显然是应该避免的，因为它导致相同内容被存储到不同缓冲中去，降低了系统存储的效率。

- 覆盖网络
  - 在一个含有N个节点的网络中，将整个网络看做一个圆环，节点按标识符从小到大顺时针组成一个环。对象分配在节点n上，n是从节点标识符大于等于对象标识的节点开始顺时针方向遇到的第一个活着的节点。


0
1
2
3
4
5
6
7
Successor（1）=3
Successor（2）=3
Successor（6）=0

- **Chord协议的数据定位**
  - 网络中共可能存在**N**个节点，**ID**值**0-N**。组成一个环，**Chord**协议规定每个节点一定知道其后继节点，查找只能是顺时针进行。
  - **Chord**为每个节点维护一个路由表，路由表的大小为**m**项（$2^m>=N$），称其为 **finger table**。**finger table**保存的节点不是直接相邻的节点，而是按照**ID**间隔2^i的关系排列(**i**表示表中的数组下标)，表中包括**m**个后继节点和一个前驱节点。前驱节点即**NodeID**比本节点小的最近节点
  - 假设本节点 **NodeID**为**X**，则**m**个后继节点分别为**NodeID**大于等于$X+2^0$，$X+2^1$，…，$X+2^m$的第一个节点。其中第**i**项指向节点**X**的后继集中第一个活动的并满足条件大于等于$n+2^i-1$的最小节点**s**

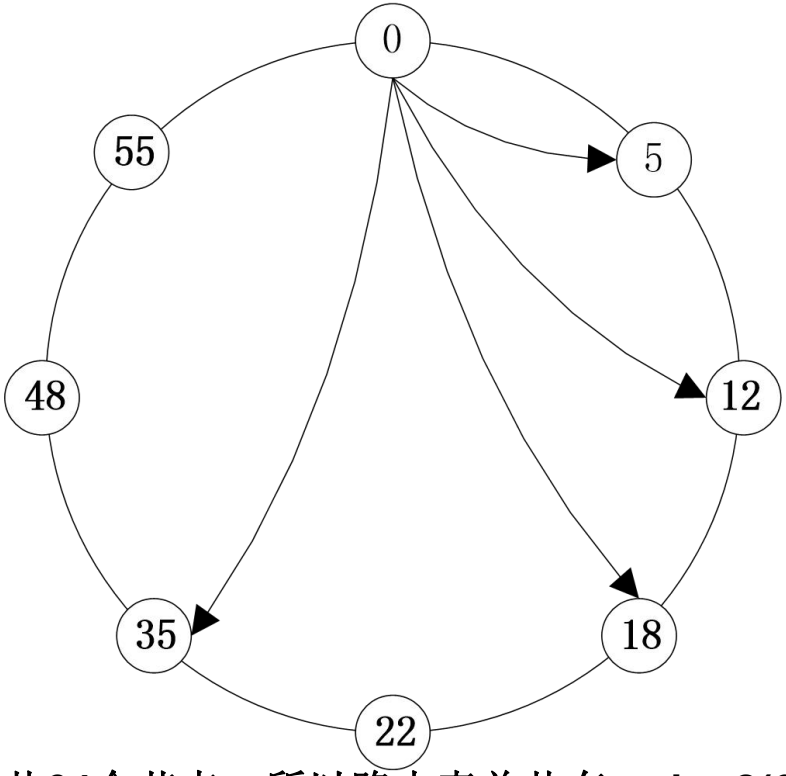
- Chord协议的数据查找
  - Chord协议在查询的过程中，查询节点将请求发送到附近(键值最接近)的节点上。收到查询请求的节点如果发现自身存储了被查询的信息，可以直接回应查询节点(这与一致性哈希完全相同)；如果被查询的信息不在本地，就根据查询表将请求转发到与键值最接近的节点上。这样的过程一直持续到找到相应的节点为止。
  - 查询过程实际上就是折半查找的过程。虽然Chord中的每个节点维护了O(log(N))个信息，但是Chord协议提高了数据路由和定位的效率，从O(N)提高到了O(log(N))，查询时信息的转发也减少到了O(log(N))。

1、查看Key的哈希是否落在节点n和其直接successor之间，若是结束查找，n的successor即为所找

2、在n的Finger表中，找出与hash(Key)距离最近且<hash(Key)的n的successor，该节点也是Finger表中最接近Key的predecessor，把查找请求转发到该节点

3、继续上述过程，直至找到Key对应的节点

- 例如64个节点的Chord环



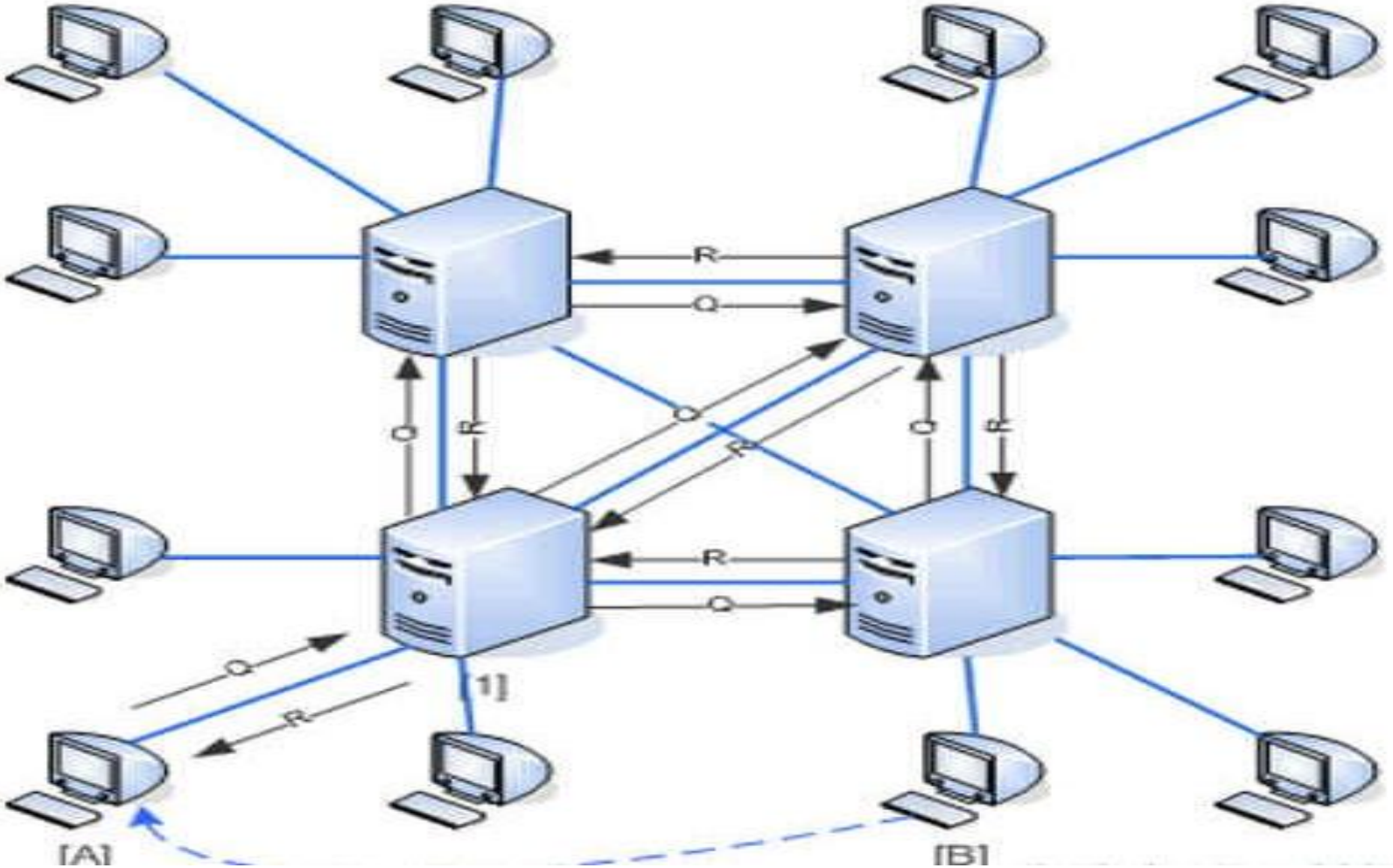
节点0的finger table

i	start	successor
0	1	5
1	2	5
2	4	5
3	8	12
4	16	18
5	32	35

总共64个节点，所以路由表总共有m=log2(64)=6项
i=0时，start=2^0
...
i=i时，start=x+2^i
这个successor是从图上看出来的，顺时针第一个就是。

- Chord协议其他的一些要素
  - 节点的加入与退出
    - 节点的加入必须有一个节点推荐
  - 节点内保存的数据资料的迁移
  - 实际中多采用 (key, value) 对来实现，key决定存储的目标节点，value则是存储在目标节点的信息，可以是内容的索引，也可能是内容本身(如文件中的一个片段)
  - 超级节点

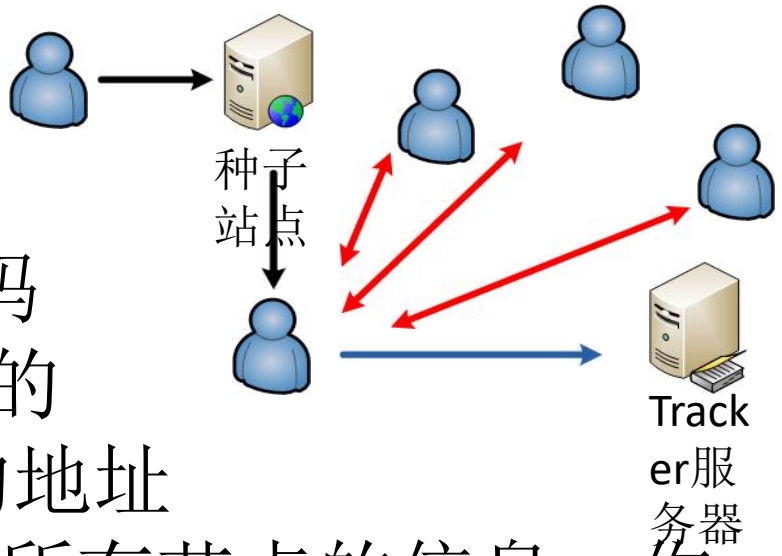
- 半分布式拓扑结构网络（Partially Decentralized Topology）
  - 半分布式结构吸取了中心化结构和全分布式结构化拓扑的优点，选择性能较高（处理、存储、带宽等方面性能）的结点作为超级点。
  - 半分布式结构也是一个层次式结构，超级点之间构成一个高速转发层，超级点和所负责的普通结点构成若干层次。
  - 结合第一、第三种索引算法
    - 在各个超级点上存储了系统中其他部分结点的信息，发现算法仅在超级点之间转发，超级点再将查询请求转发给适当的叶子结点。
    - 也可采用DHT协议进行索引查询
  - 半分布式结构的优点是性能、可扩展性较好，较容易管理，但对超级点（中心节点）依赖性大，易于受到攻击，容错性也受到影响。



半分布式结构（含有**SuperNode**）

半分布式拓扑结构网络代表 — BT系统

- 种子站点是一个普通的Web站点，通常不作为BT的组成
- 种子：一个Bencode编码的元信息文件，最重要的包含了Tracker服务器的地址



- Tracker服务器：储存了所有节点的信息，作为节点彼此了解的“约会地点”
- 节点：文件共享的实际参与者，其中种子节点拥有了全部文件内容
- 种子的INFO_HASH(数据文件的唯一标识)是作为key在DHT网络中储存

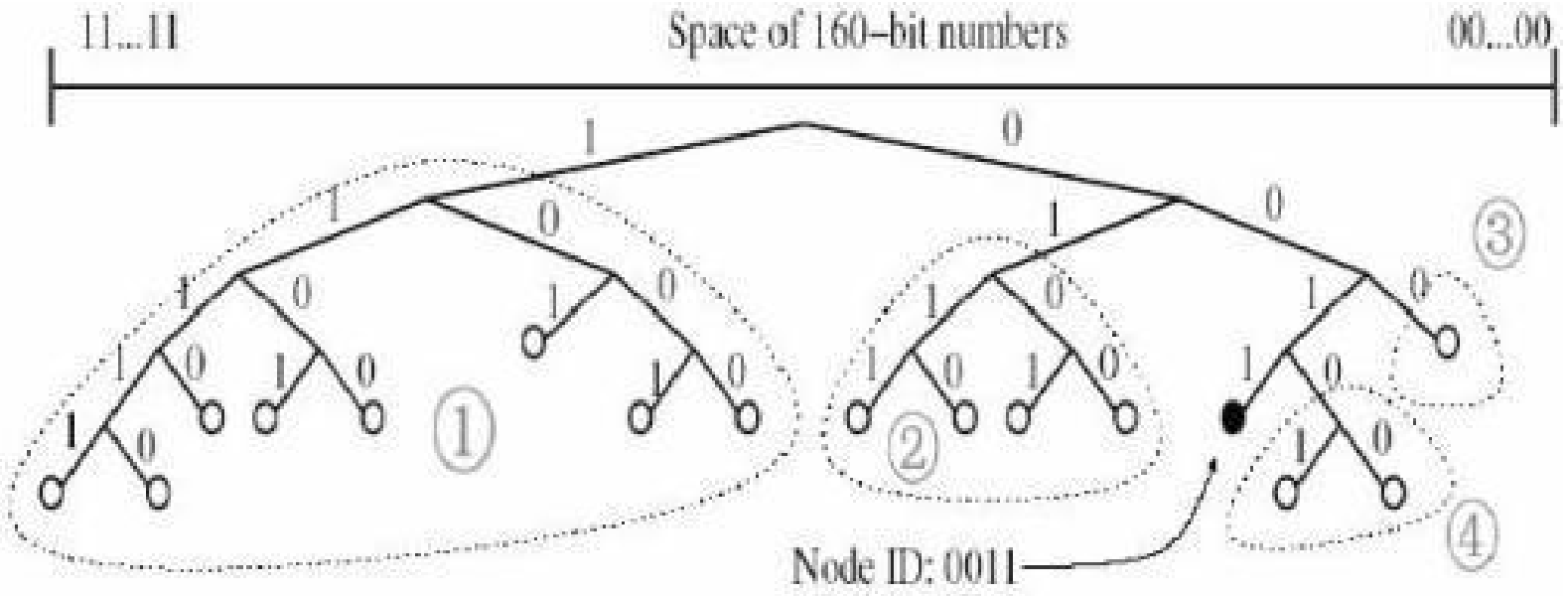
- 每个节点根文件计算INFO_HASH，生成种子文件的时候，种子文件里包含INFO_HASH和tracker的信息。然后会在tracker上登记INFO_HASH以及共享该文件的节点IP、端口。
- 每个节点下载数据的时候，同时也会在Tracker登记自己的IP、端口
- Tracker包含了共享该文件的所有节点的IP、端口
- 在Tracker查询该种子相关信息的时候，是通过INFO_HASH查找

- 在Tracker查找的同时，通过DHT也开始查找。
  - 种子生成之后，会通过INFO_HASH与ID的对应关系，利用DHT查找相应的ID的节点，这些节点会保存INFO_HASH以及共享该文件的节点的IP、端口
  - 其他节点下载该文件时，会通过INFO_HASH与ID的对应关系，通过DHT查找相应的ID的节点，然后也登记自己的IP、端口

BT采用的DHT协议

- KAD协议
  - Key的划分
    - Kad网络中每个节点都有一个160bit的ID值作为标志符，数据文件的INFO_HASH也是一个160bit的标志符，记为Key。每一个加入Kad网络的计算机都会在160bit的key空间被分配一个节点ID（node ID）值，<key,value>对的数据(拥有该文件的节点的IP、端口)就存放在ID值“最”接近key值的节点上。
  - 覆盖网络
    - 所有节点都被当作一颗二叉树的叶子
    - 每一个节点的位置都由其ID值的最短前缀唯一的确定。
      对于任意一个节点，都可以把这颗二叉树分解为一系列连续的，不包含自己的子树。

Kad协议确保每个节点知道其各子树的至少一个节点



根据**NodeID**的前缀可以将其他部分分为**4**组

- KAD节点的定位与查询
  - 节点间距离
    - 采用异或运算判断两个节点x,y的距离远近
    - 对于任意给定的节点x和距离⊿≧0，总会存在一个精确的节点y，使得d(x,y)= ⊿
  - 定位与查询
    - 查找本地节点列表中离目的节点最近的节点
    - 将查询请求发给该节点

- KAD节点的距离

$$Dis(M,N) = XOR(M,N)$$

  - 节点之间的距离越近，意味着节点ID的公共前缀越长
  - 公共前缀长度将这颗二叉树分解为一系列不包含自己的子树。顶层的子树，由整棵不包含自己的树的另一半组成，即与节点之间的公共前缀长度为0；下一层子树由剩下部分不包含自己的一半组成，即公共前缀长度为1；依此类推，直到分割完整的树。

- 节点距离与分组
  - 节点A, B, C, D与节点M的公共前缀长度为0，将其归为一个单元0，节点F, G与M的公共前缀长度为1，将其归为单元1，节点E与M的公共前缀长度为2，将其归为单元2，因此，通过与M的距离，可以将网络中的这7个节点分为3组

$$Dis = 0 : \{A, B, C, D\}$$
$$Dis = 1 : \{F, G\}$$
$$Dis = 2 : \{E\}$$

- 构建路由表
  - 构建路由表的本质是建立到网络全局的地图，目标是：对于节点M，给定任意节点X，可以根据节点很容易计算出距离X更近的节点列表。
  - 虽然目标是本地一步到位的查找，但这是不现实的，这需要维护数量巨大的全局节点信息。
  - 采用迭代查找的思路：每次查找要距离目标更近一点点。

- KAD的路由表
  - K桶结构
  - 每个K桶覆盖距离的范围呈指数关系增长，形成离自己近的节点的信息多，离自己远的节点的信息少，从而可以保证路由查询过程是收敛。
    - K桶内的节点数目超过K值时，分裂，将与本地节点更近(即更大)节点迁移至新桶
  - 经过证明，对于一个有N个节点的Kad网络，最多只需要经过logN步查询，就可以准确定位到目标节点。

- KAD其他要素
  - 节点加入
    - 通过向已知KAD节点发送含有本节点信息的消息，该KAD节点返回一定数量的KAD节点列表。
  - 获取路由列表信息
    - 与其他KAD节点交互路由表，完善本节点的K桶

- 数据存储小结
  - 高频数据存储
    - 快照、增量、索引结构
  - 非高频数据存储
    - 分布式存储(key-value存储)
      - 数据分割
      - 数据组织结构、索引

高可用系统

- Mapreduce
- Hadoop
- Spark

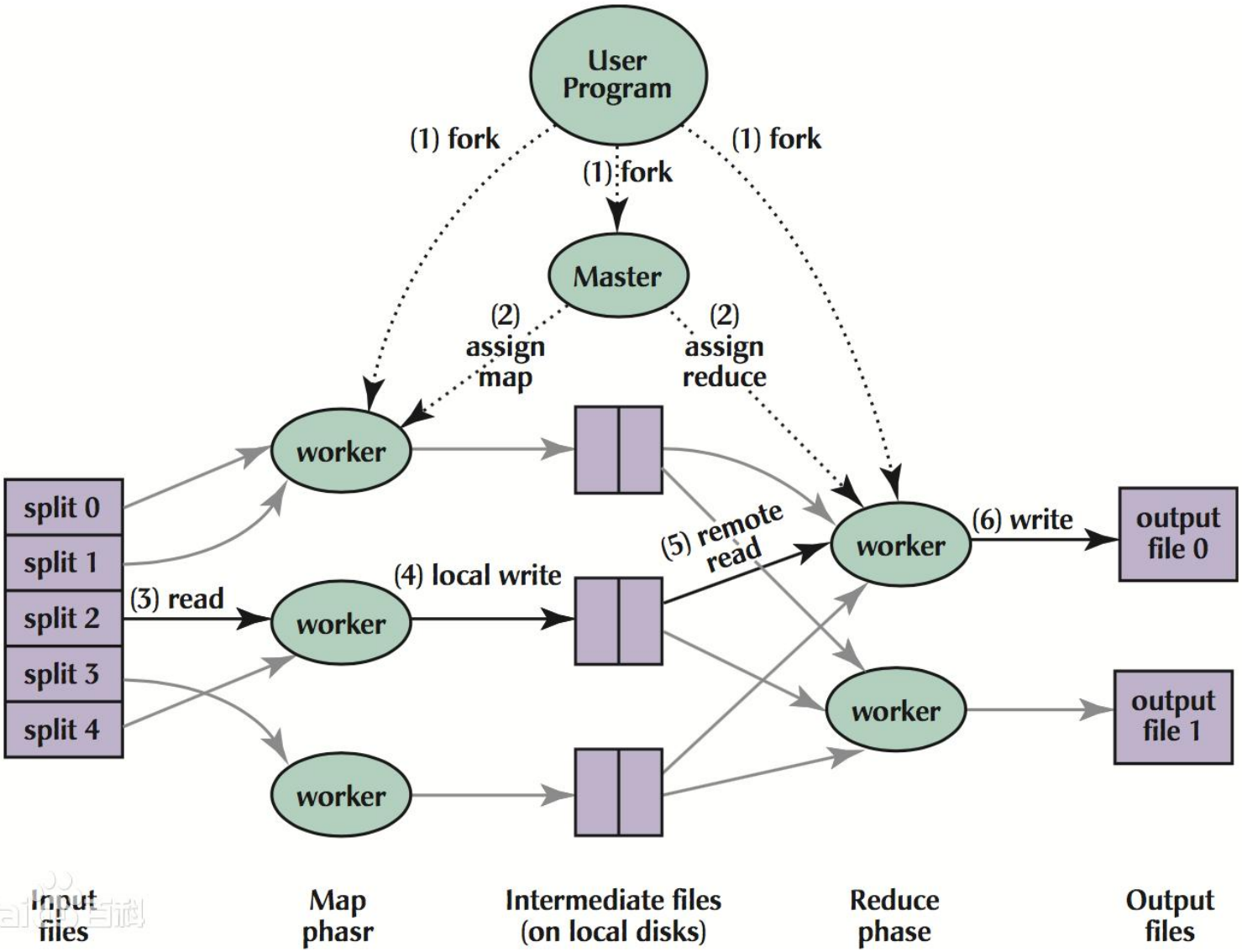
Mapreduce

- MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。
- "Map（映射）"和"Reduce（归约）“
  - 指定一个Map（映射）函数，用来把一组键值对映射成一组新的键值对，指定并发的Reduce（归约）函数，用来保证所有映射的键值对中的每一个共享相同的键组。

- 1）MapReduce是一个基于集群的高性能并行计算平台（Cluster Infrastructure）。它允许用市场上普通的商用服务器构成一个包含数十、数百至数千个节点的分布和并行计算集群。
- 2）MapReduce是一个并行计算与运行软件框架（Software Framework）。能自动完成计算任务的并行化处理，自动划分计算数据和计算任务，在集群节点上自动分配和执行任务以及收集计算结果，将数据分布存储、数据通信、容错处理等并行计算涉及到的很多系统底层的复杂细节交由系统负责处理，减少软件开发人员的负担。
- 3）MapReduce是一个并行程序设计模型与方法（Programming Model & Methodology）。用Map和Reduce两个函数编程实现基本的并行计算任务，提供了抽象的操作和并行编程接口，以简单方便地完成大规模数据的编程和计算处理

Mapreduce主要功能

- **数据划分和计算任务调度：**系统自动将一个作业（Job）待处理的大数据划分为很多个数据块，每个数据块对应于一个计算任务（Task），并自动调度计算节点来处理相应的数据块。
- **本地化数据处理：**本地化数据处理，即一个计算节点尽可能处理其本地磁盘上所分布存储的数据，这实现了代码向数据的迁移；当无法进行这种本地化数据处理时，再寻找其他可用节点并将数据从网络上传送给该节点（数据向代码迁移）。
- **出错检测和恢复：**检测并隔离出错节点，并调度分配新的节点接管出错节点的计算任务。同时，系统还将维护数据存储的可靠性，用多备份冗余存储机制提高数据存储的可靠性，并能及时检测和恢复出错的数据。
- **其他优化措施：**中间结果数据进入Reduce节点前会进行一定的合并处理；一个Reduce节点所处理的数据可能会来自多个 Map节点，为了避免Reduce计算阶段发生数据相关性，Map节点输出的中间结果需使用一定的策略进行适当的划分处理，保证相关性数据发送到同一个Reduce节点；此外，系统还进行一些计算性能优化处理，如对最慢的计算任务采用多备份执行、选最快完成者作为结果。


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phasr
Intermediate files (on local disks)
Reduce phase
Output files

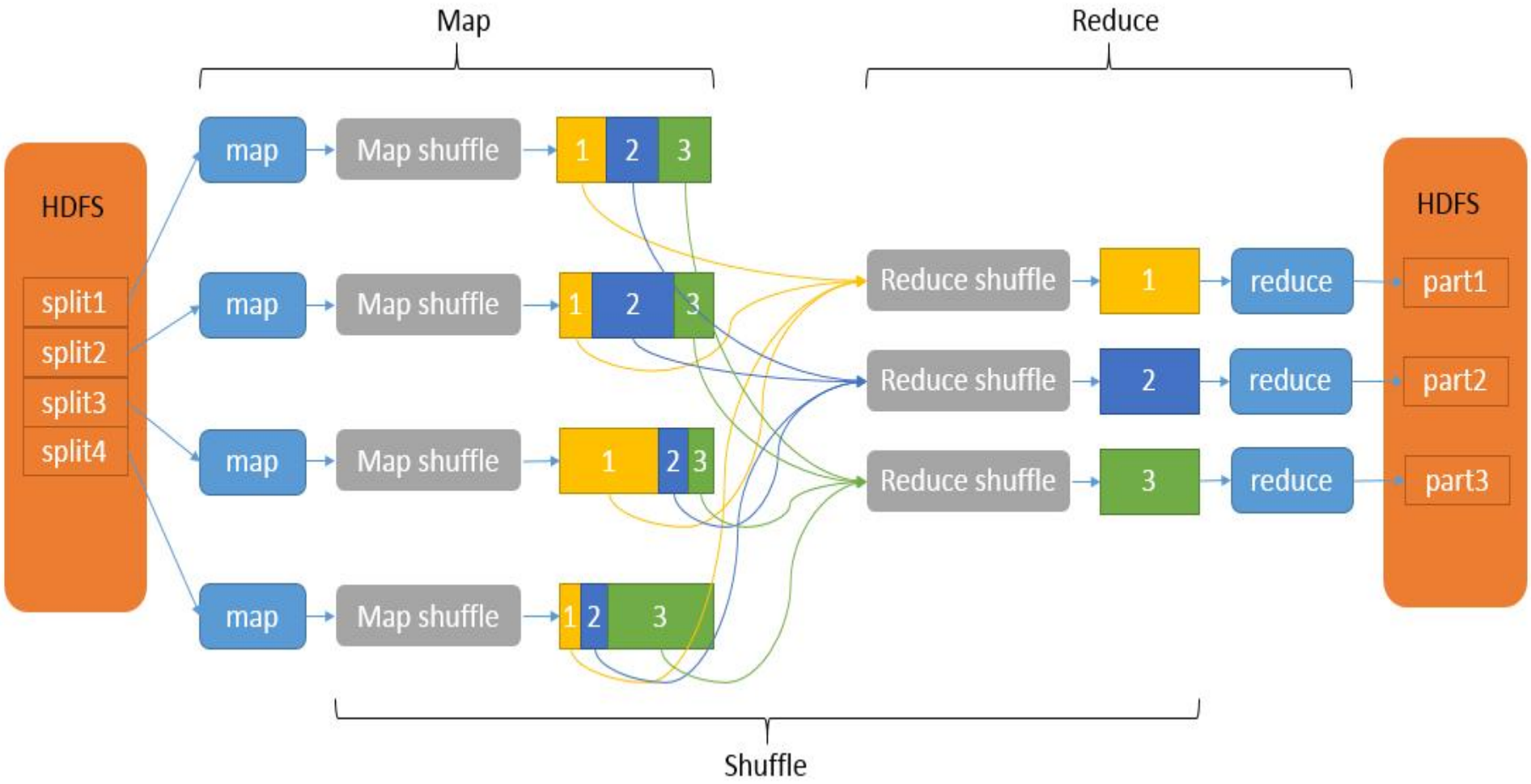
Mapreduce工作原理

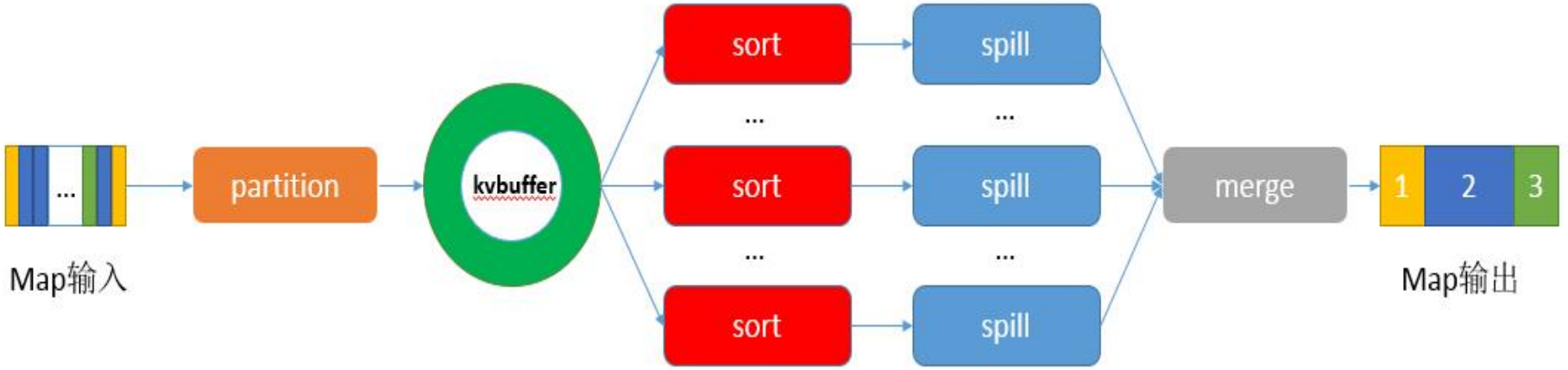
- 从user program开始，user program链接了MapReduce库，实现最基本的Map函数和Reduce函数。
- 1.MapReduce库先把user program的输入文件划分为M份（M为用户定义），然后使用fork将用户进程拷贝到集群内其它机器上。
- 2.user program的副本中有一个称为master，其余称为worker，master是负责调度的，为空闲worker分配作业（Map作业或者Reduce作业），worker的数量也是可以由用户指定的。
- 3.被分配了Map作业的worker，开始读取对应分片的输入数据，Map作业数量是由Master决定；Map作业从输入数据中抽取出键值对，每一个键值对都作为参数传递给map函数，map函数产生的中间键值对被缓存在内存中。
- 4.缓存的中间键值对会被定期写入本地磁盘，而且被分为R个区，R的大小是由用户定义的，将来每个区会对应一个Reduce作业；这些中间键值对的位置会被通报给master，master负责将信息转发给Reduce worker。

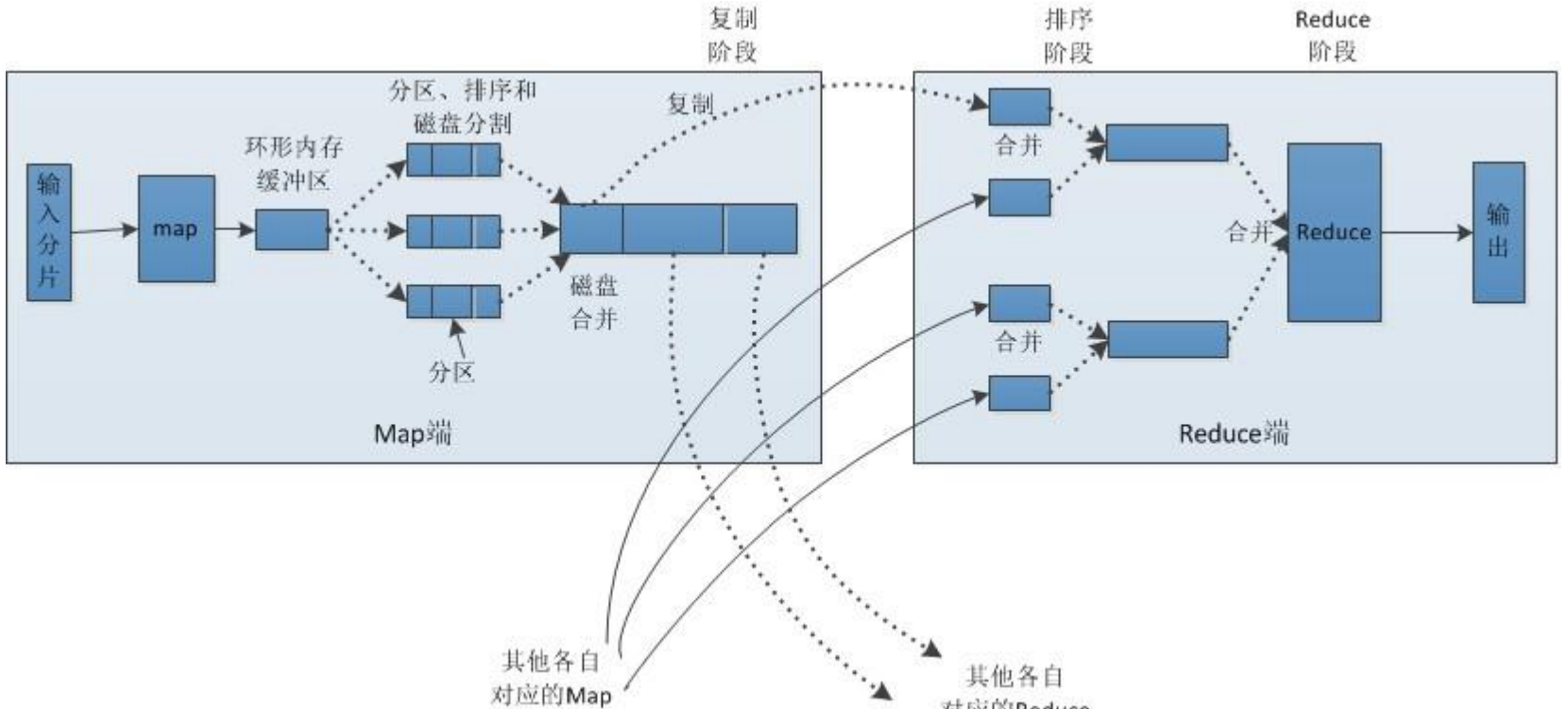
Mapreduce工作原理

- 5.master通知分配了Reduce作业的worker它负责的分区在什么位置（每个Map作业产生的中间键值对都可能映射到所有R个不同分区），当Reduce worker把所有它负责的中间键值对都读过来后，先对它们进行排序，使得相同键的键值对聚集在一起。因为不同的键可能会映射到同一个分区也就是同一个Reduce作业（分区少）。
- 6.reduce worker遍历排序后的中间键值对，对于每个唯一的键，都将键与关联的值传递给reduce函数，reduce函数产生的输出会添加到这个分区的输出文件中。
- 7.当所有的Map和Reduce作业都完成了，master唤醒user program，MapReduce函数调用返回user program的代码。
- 所有执行完毕后，MapReduce输出放在了R个分区的输出文件中。整个过程中，输入数据是来自底层分布式文件系统（GFS）的，中间数据是放在本地文件系统的，最终输出数据是写入底层分布式文件系统（GFS）的。

Mapreduce处理流程

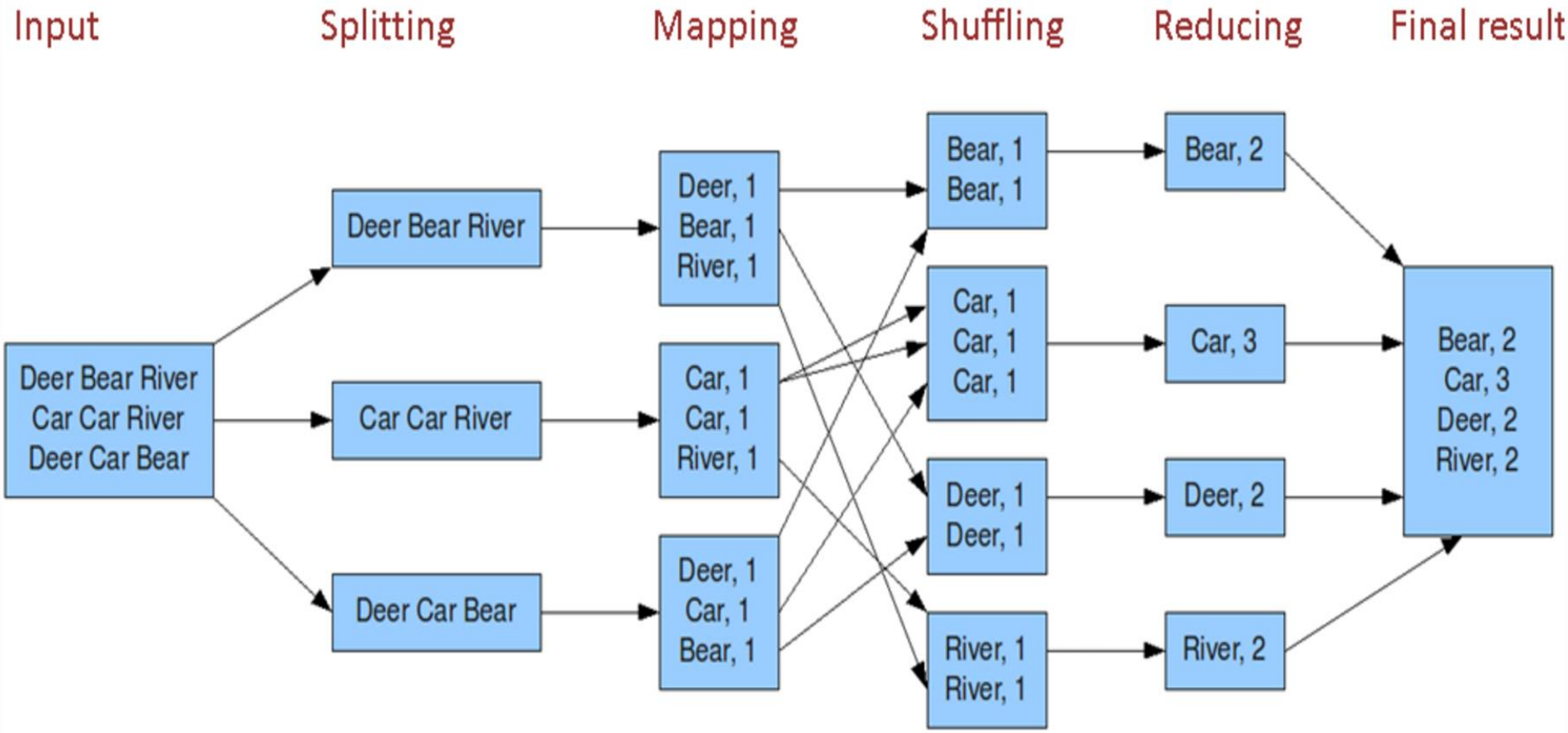


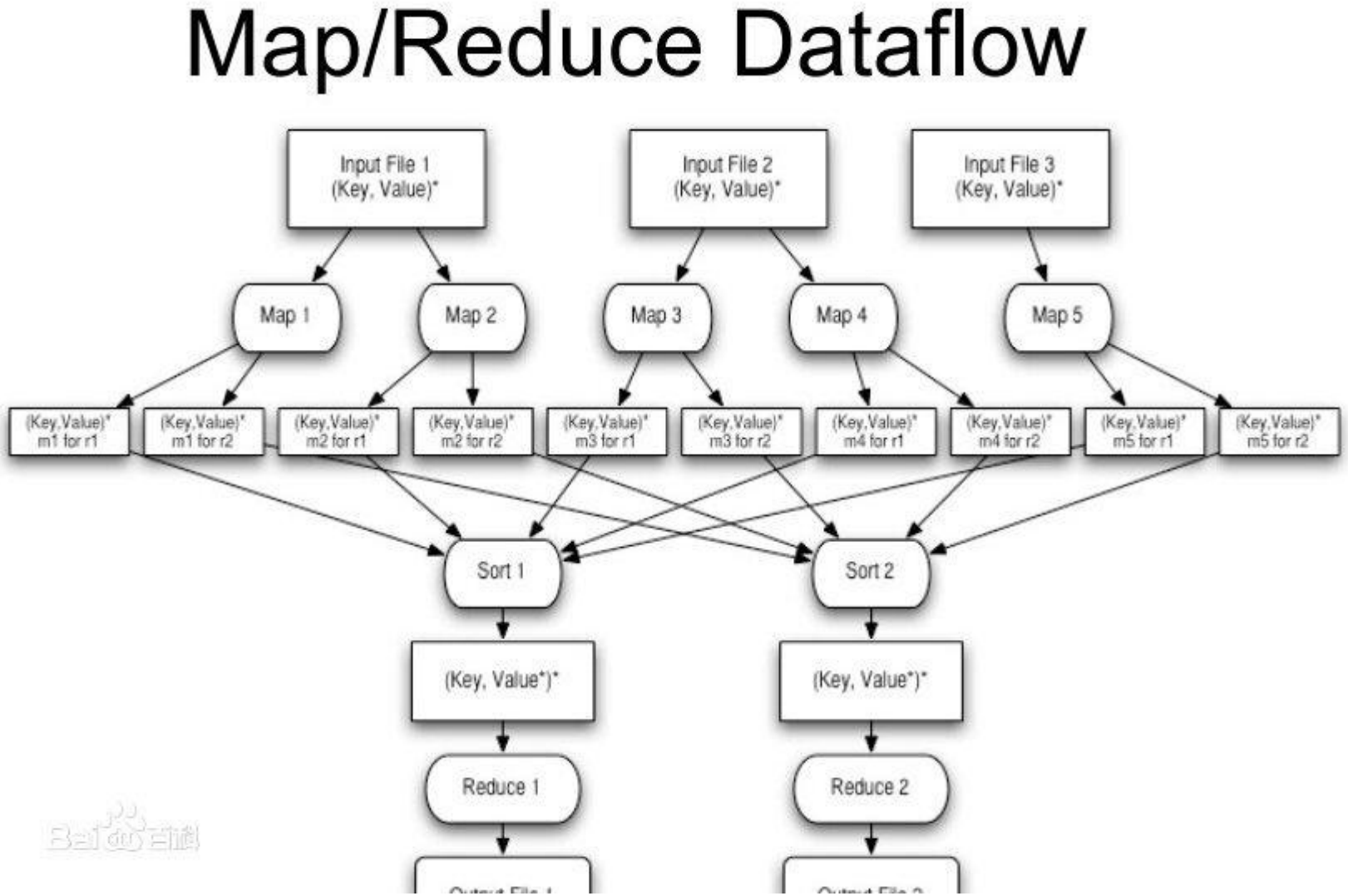

...
partition
kvbuffer
sort
sort
sort
spill
spill
spill
merge
1
2
3
Map输入
Map输出


复制
阶段
排序
阶段
Reduce
阶段
输入分片
map
环形内存
缓冲区
分区、排序和
磁盘分割
复制
磁盘
合并
分区
Map端
合并
合并
合并
Reduce
输出
Reduce端
其他各自
对应的Map
其他各自

shuffle的本意是洗牌、混洗的意思，把一组有规则的数据尽量打乱成无规则的数据。而在MapReduce中，shuffle更像是洗牌的逆过程，指的是将map端的无规则输出按指定的规则“打乱”成具有一定规则的数据，以便reduce端接收处理。

一个例子




Map/Reduce Dataflow
Input File 1 (Key, Value)*
Input File 2 (Key, Value)*
Input File 3 (Key, Value)*
Map 1
Map 2
Map 3
Map 4
Map 5
(Key,Value)* m1 for r1
(Key,Value)* m1 for r2
(Key,Value)* m2 for r1
(Key,Value)* m2 for r2
(Key,Value)* m3 for r1
(Key,Value)* m3 for r2
(Key,Value)* m4 for r1
(Key,Value)* m4 for r2
(Key,Value)* m5 for r1
(Key,Value)* m5 for r2
Sort 1
Sort 2
(Key, Value*)*
(Key, Value*)*
Reduce 1
Reduce 2

Mapreduce的容错机制

- Task失效
  - Task 失败是在map或reduce task中的用户代码抛出一个运行时异常。如果发生这种情况，worker将向master报告一个错误，错误最终会被写进用户的日志中。application master将task标记为失败，并且释放worker。然后重新调度执行task，对于一些应用，如果仅有少数的task失败，并不希望终止job，因为尽管有些失败，但是结果也许还是可以用的。
- Worker失效
  - 当Worker （也可称作TaskTracker）进程崩溃或者所在节点故障时， master 将接收不到其发来的心跳，那么将会认为该节点失效并且在该节点运行过的任务都会被认为失败，这些将会被重新调度到别的节点执行。
- Master失效
  - Master （也可称作JobTracker ）出错是非常严重的额情况，一旦出错，那么正在运行的所有作业的内部状态信息将会丢失，即使JobTracker马上恢复了，作业的所有任务都会被认为是失败的，即所有作业都需要重新执行。

Hadoop

- Hadoop是一个由Apache基金会所开发的分布式系统基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群进行高速运算和存储。
  - 受google的MapReduce启发，结合自身的分布式存储技术，2006年2月发布的一套完整而独立的软件，并被命名为Hadoop
- Hadoop是在分布式服务器集群上存储海量数据并运行分布式分析应用的开源框架，其核心部件是HDFS与MapReduce。

Hadoop

- Hadoop实现了一个分布式文件系统（Hadoop Distributed File System），HDFS有高容错性的特点，适合那些有着超大数据集（large data set）的应用程序。HDFS为海量的数据提供了存储，而MapReduce则为海量的数据提供了计算 。
  - HDFS可理解为一个分布式的，有冗余备份的，可以动态扩展的用来存储大规模数据的大硬盘。
  - 把MapReduce理解成为一个计算引擎，按照MapReduce的规则编写Map计算/Reduce计算的程序，完成计算任务。

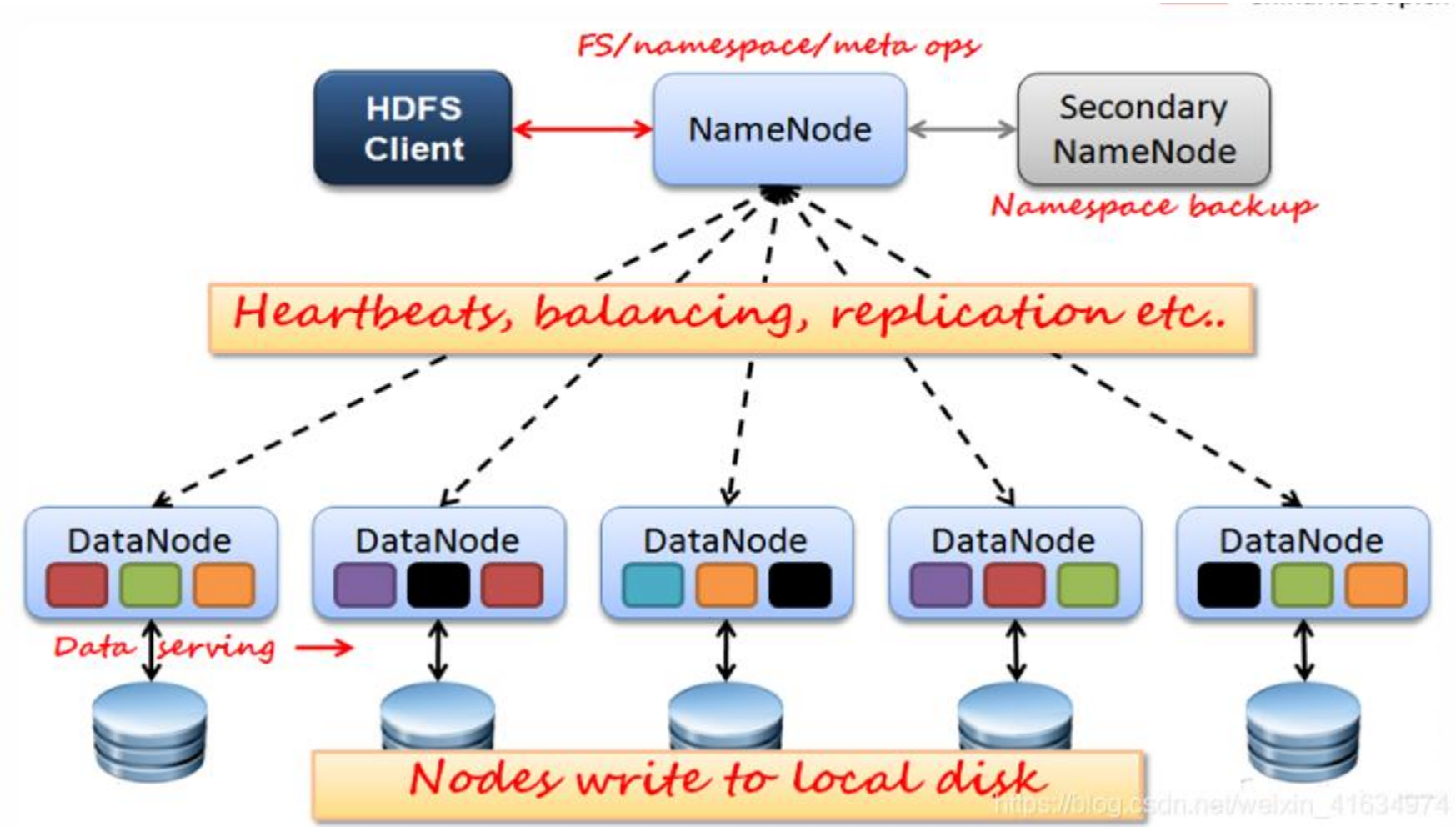
Hadoop优点

- **1.高可靠性**。Hadoop假设计算元素和存储会失败，因此它维护多个工作数据副本，确保能够针对失败的节点重新分布处理 。
- **2.高扩展性**。Hadoop是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中[4] 。
- **3.高效性**。Hadoop能够在节点之间动态地移动数据，并保证各个节点的动态平衡，处理速度快 。
- **4.高容错性**。Hadoop能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。
- **5.低成本**。hadoop是开源的，低价格服务上大规模部署

HDFS

- Hadoop 由许多元素构成。其最底部是 Hadoop Distributed File System（HDFS），它存储 Hadoop 集群中所有存储节点上的文件。HDFS就像一个传统的分级文件系统。可以创建、删除、移动或重命名文件等。
  - 这些节点包括 NameNode（1个或2个），它在 HDFS 内部提供元数据服务
  - DataNode，它为 HDFS 提供存储块。
  - 存储在 HDFS 中的文件被分成块，这些块复制到多个计算机中（DataNode）。NameNode 可以控制所有文件操作。

HDFS


FS/namespace/meta ops
HDFS Client
NameNode
Secondary NameNode
Namespace backup
Heartbeats, balancing, replication etc..
DataNode
DataNode
DataNode
DataNode
DataNode
Data serving
Nodes write to local disk

HDFS

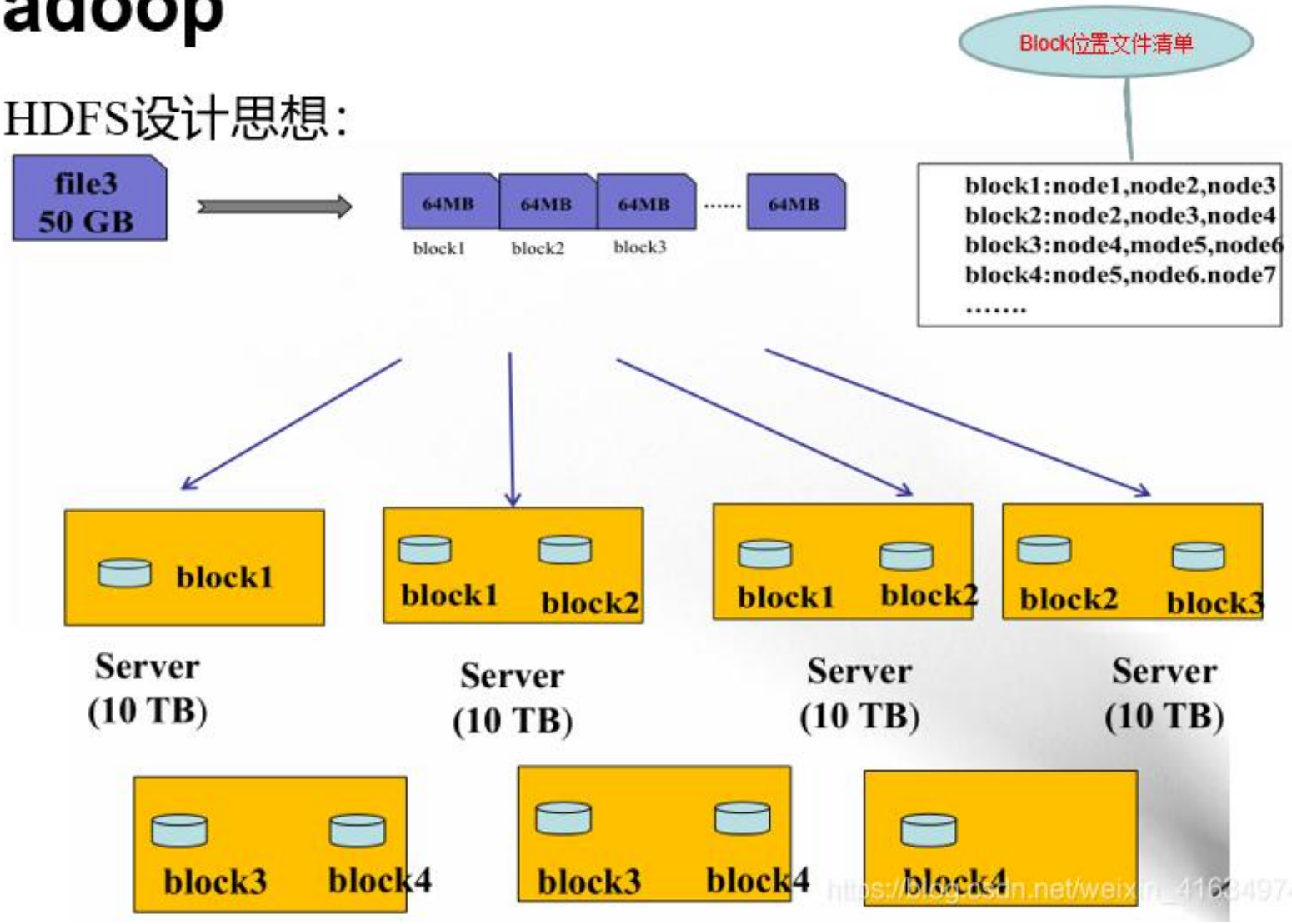
- 文件线性切割成块（Block）block分散存储存储在集群节点中，单一文件Block大小一致，文件与文件可以不一致。Block可以设置副本数，副本无序分散在不同节点（默认是3个）副本数不能超过节点数量
- NameNode 负责管理文件系统名称空间和控制外部客户机的访问。NameNode 决定文件映射到 DataNode 上的复制块上。
- 实际的 I/O事务并没有经过 NameNode，只有表示 DataNode 和块的文件映射的元数据经过 NameNode。
- 当外部客户机发送请求要求创建文件时，NameNode 会以块标识和该块的第一个副本的 DataNode IP 地址作为响应。这个 NameNode 还会通知其他将要接收该块的副本的 DataNode 。

HDFS

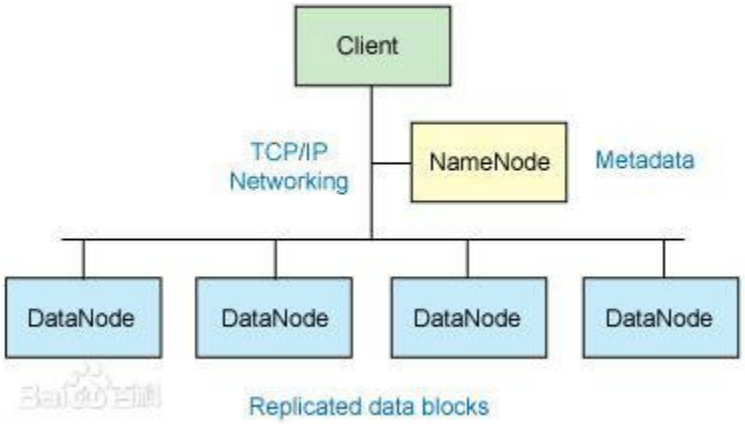
- DataNode 。Hadoop 集群包含一个 NameNode 和大量 DataNode。DataNode 通常以机架的形式组织，机架通过一个交换机将所有系统连接起来。
- DataNode 响应来自 HDFS 客户机的读写请求。还响应来自 NameNode 的创建、删除和复制块的命令。
- NameNode 依赖来自每个 DataNode 的定期心跳（heartbeat）消息。每条消息都包含一个块报告，NameNode 可以根据这个报告验证块映射和其他文件系统元数据。如果 DataNode 不能发送心跳消息，NameNode 将采取修复措施，重新复制在该节点上丢失的块。

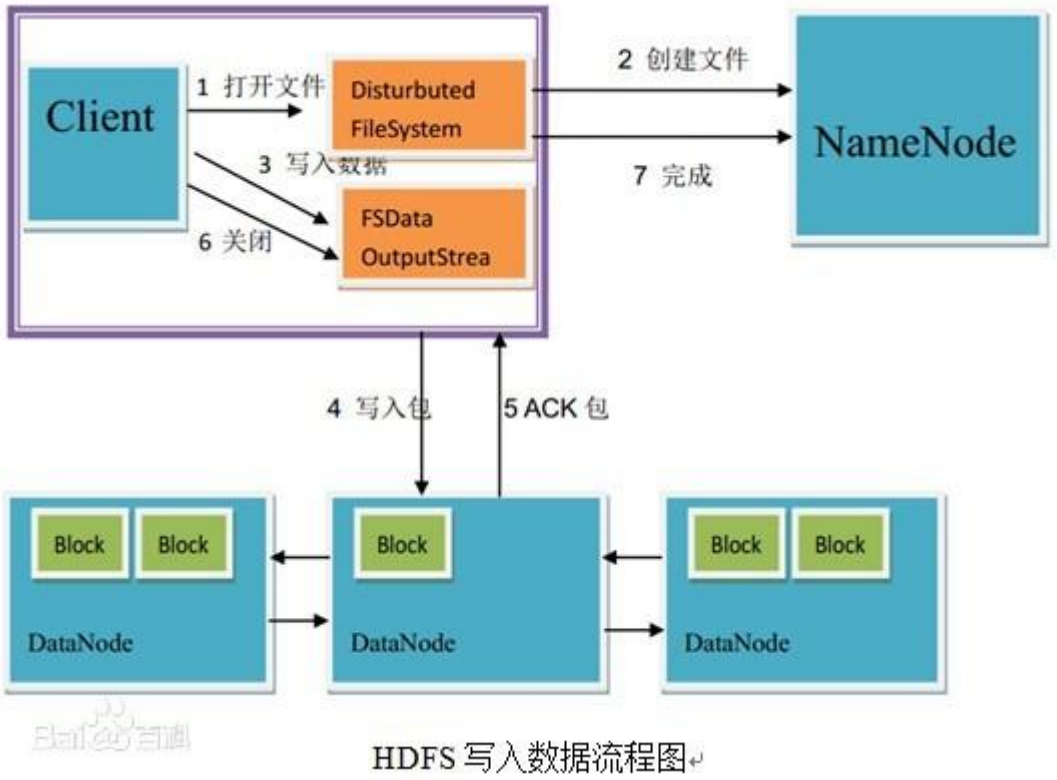
Hadoop

- HDFS设计思想：



HDFS

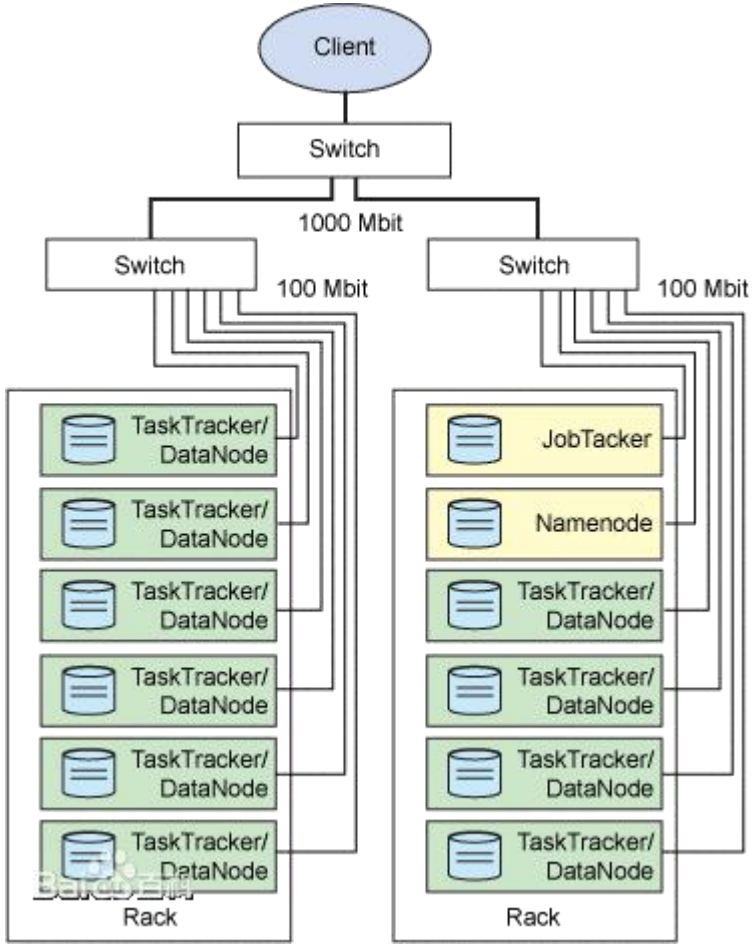




HDFS 写入数据流程图

Hadoop：HDFS+Mapreduce

- 将处理移动到存储
- JobTracker。应用程序提交之后，将提供包含在 HDFS 中的输入和输出目录。
- JobTracker 使用文件块信息（物理量和位置）确定如何创建 TaskTracker 从属任务。
- MapReduce应用程序被复制到每个出现输入文件块的节点。将为特定节点上的每个文件块创建一个惟一的从属任务。
- 每个 TaskTracker 将状态和完成信息报告给 JobTracker。



Hadoop适用场景

- 大数据存储：分布式存储
- 日志处理：擅长日志分析
- ETL：数据抽取到oracle、mysql、DB2、mongdb及主流数据库
- 搜索引擎：海量网页的处理
- 数据挖掘：机器学习
  - Yahoo 使用4000个节点的Hadoop集群来支持广告系统和Web 搜索
  - Facebook 使用1000个节点的集群运行Hadoop，存储日志数据，支持其上的数据分析和机器学习；
  - 百度用Hadoop处理每周200TB 的数据，从而进行搜索日志分析和网页数据挖掘工作；
  - 淘宝的Hadoop 系统用于存储并处理电子商务交易的相关数据。
  - 中国移动研究院基于Hadoop 开发了“大云”（Big Cloud）系统，用于相关数据分析，并对外提供服务；

- Hadoop是专为离线和大规模数据分析而设计的，并不适合那种对几个记录随机读写的在线事务处理模式

Spark

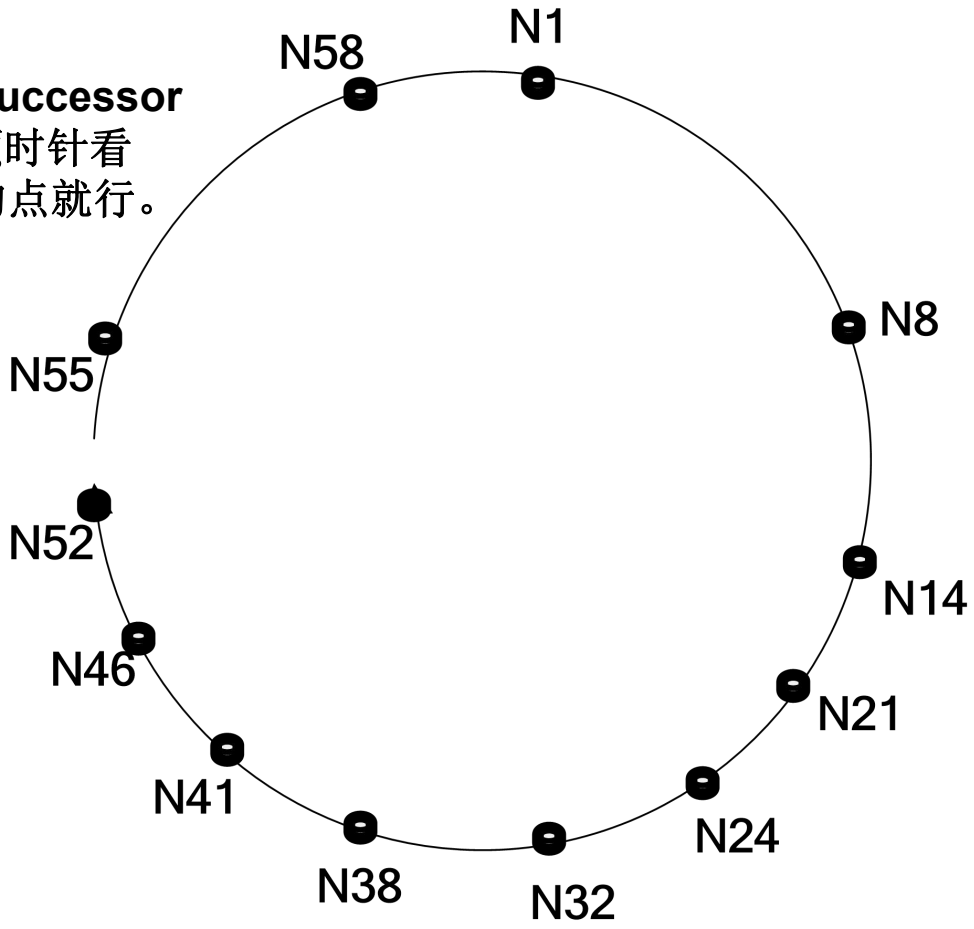
- Spark 是专为大规模数据处理而设计的快速通用的计算引擎。Spark是UC Berkeley AMP lab 所开源的类Hadoop MapReduce的通用并行框架
  - Job中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark能更好地适用于数据挖掘与机器学习等需要迭代的MapReduce的算法。
  - 能进行批任务处理和流任务处理

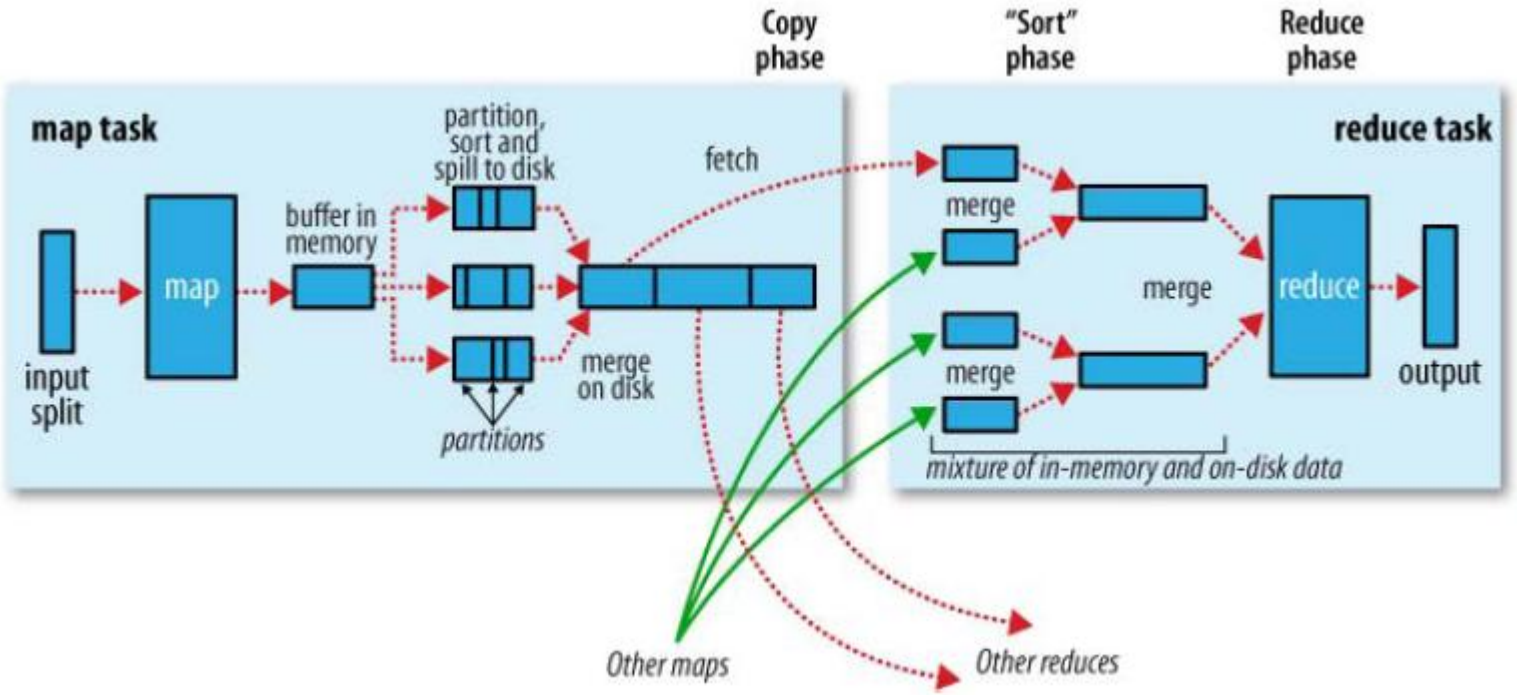
Spark优点

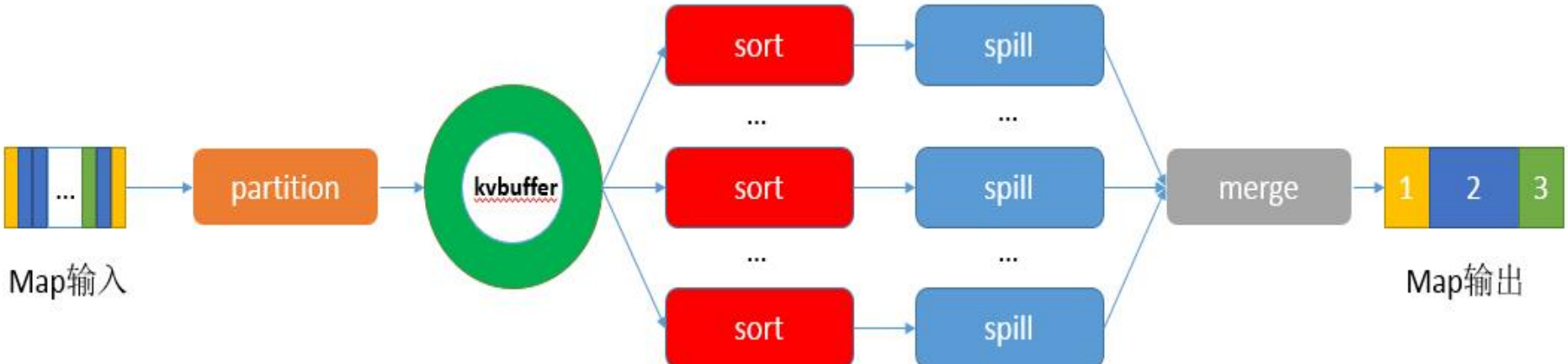
- 更快的速度：内存计算下，Spark 比 Hadoop 快100倍。
- 易用性：Spark 提供了80多个高级运算符。高级API 剥离了对集群本身的关注，Spark 应用开发者可以专注于应用所要做的计算本身
- 通用性：Spark 提供了大量的库，包括Spark Core、Spark SQL、Spark Streaming、MLlib、GraphX。 开发者可以在同一个应用程序中无缝组合使用这些库。

- 习题：一个采用Chord协议的分布式存储系统中，假设节点ID值范围为0-63。现共有12个节点构成一个Chord环。节点N1 、N14的路由表是：

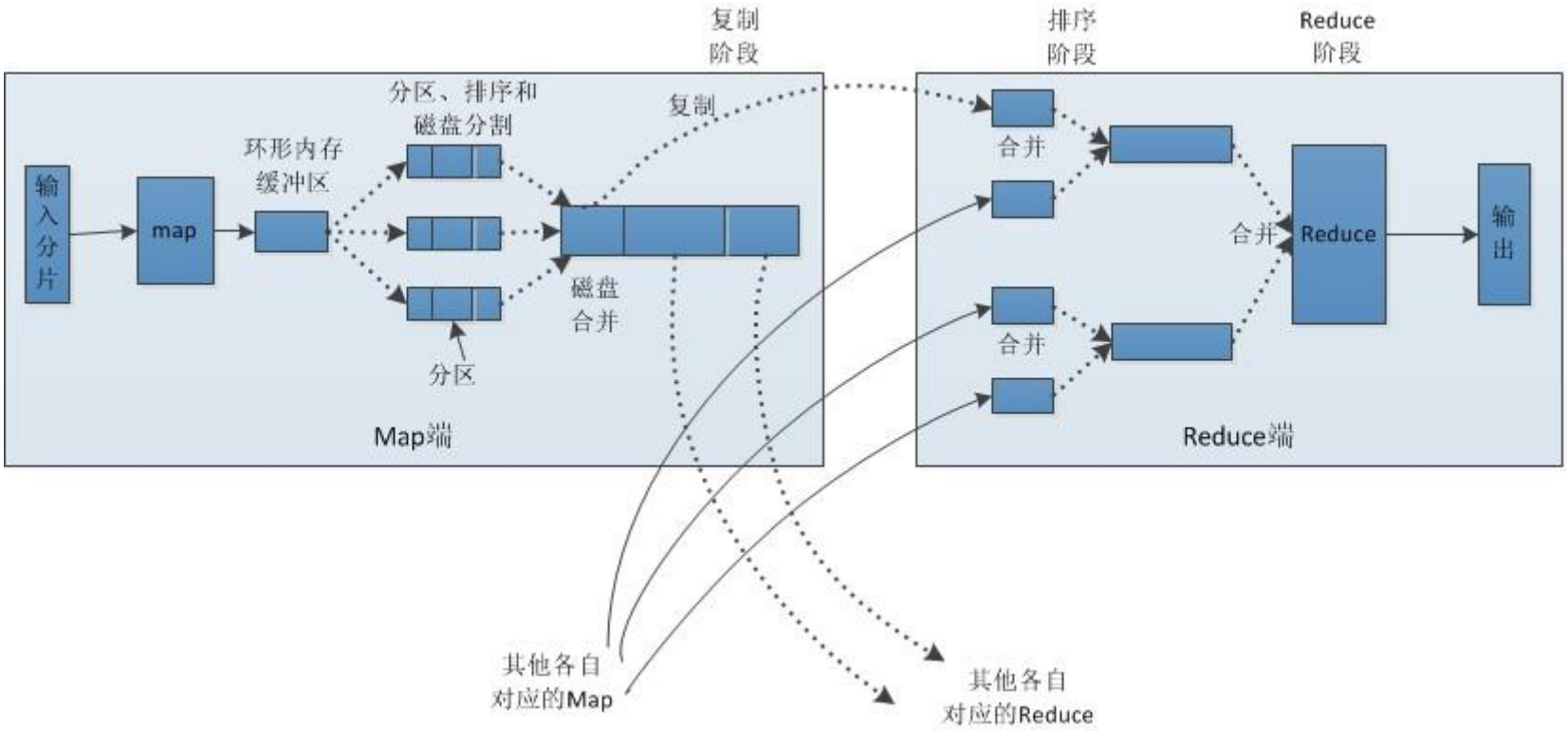
6个行
每个行有start和successor
successor就是顺时针看
第一个图上出现的点就行。




Copy phase
"Sort" phase
Reduce phase
map task
reduce task
partition, sort and spill to disk
fetch
buffer in memory
map
input split
merge on disk
partitions
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces


...
partition
kvbuffer
sort
sort
sort
spill
spill
spill
merge
1
2
3
Map输入
Map输出

Mapreduce


复制
阶段
排序
阶段
Reduce
阶段
分区、排序和
磁盘分割
复制
环形内存
缓冲区
输入分片
map
磁盘
合并
分区
Map端
合并
合并
合并
Reduce
输出
Reduce端
其他各自
对应的Map
其他各自
对应的Reduce

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件安全开发

- 3.1 软件安全的指导原则
- 3.2 软件安全需求
- 3.3 软件安全编码基本方法与技术
- 3.4 软件安全设计

面向网络攻击的软件安全设计

- 可用性安全
- 机密性安全
- 完整性安全
- 可控性安全
- 可审性安全

信息鉴别

- 需求与问题
- 消息的鉴别
- 身份的鉴别

需求与问题

- 网络协议缺陷带来的问题
  - 地址伪造
  - 信息截获
  - 信息重放

  - 信息篡改
  - 资源占用攻击
  - 抵赖

需求与问题

- 保密性
  - 截获(泄露)：消息内容发布给非授权人
  - 流量分析：发现团体之间信息流的结构模式。在一个面向连接的应用中，可以用来确定连接的频率和持续时间长度。
- 完整性
  - 内容修改：消息内容被插入删除变换修改。
  - 顺序修改：插入删除或重组消息序列。
- 可控性
  - 伪造消息：从一个假冒信息源向网络中插入消息。
  - 时间修改：消息延迟或重放
- 不可否认性
  - 接受者否认收到消息发送者否认发送过消息
  - 身份的伪造
- 可用性
  - 资源占用攻击

对攻击者的假设

- 攻击者模型：Dolev-Yao模型
  - 可以窃听所有经过网络的消息
  - 可以阻止和截获所有经过网络的消息
  - 可以存储所获得或自身创造的消息
  - 可以根据存储的消息伪造消息，并发送该消息
  - 可以做为合法的主体参与协议的运行

信息鉴别的目标

- 证实信息来自可信的源点且未被篡改
  - 消息的鉴别(Authentication):
    - 真伪性
    - 验证信息的完整性，在传送或存储过程中未被篡改，重放或延迟等。
    - 信息的加密
  - 消息主体身份的认证(Certification)
    - 资格审查
    - 验证信息的发送者是真正的，而不是冒充的，此为信源识别；
    - 身份认证
- 收到的消息可以做为证据证明是其声称的发出者所发出
  - 信息的可审性

信息的鉴别

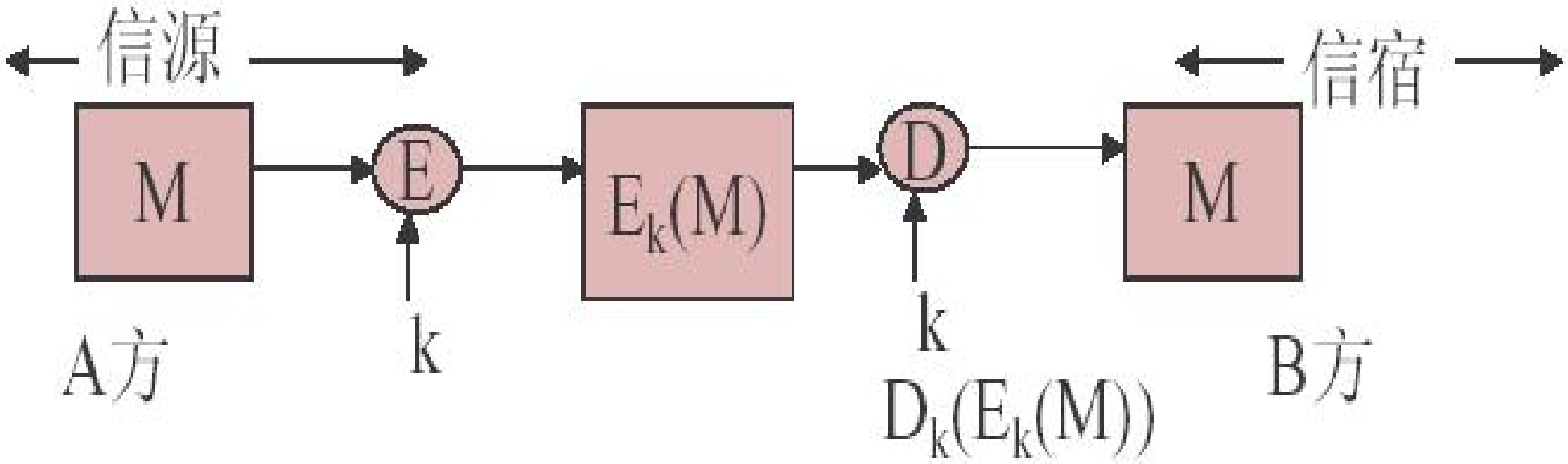
- 鉴别方法：
  - (1)消息加密函数(Message encryption)：用完整信息的密文作为对信息的加密与鉴别。
  - (2)散列函数(Hash Function)：将任意长的信息映射成一个固定长度的信息。
  - (3)消息鉴别码MAC(Message Authentication Code)：散列函数+密钥产生一个固定长度的值作为鉴别标识

消息加密函数

- 加密函数分二种
  - 对称密钥加密技术：一种是常规的对称密钥加密函数，加密密钥和解密密钥相同；
  - 非对称密钥加密技术：一种是公开密钥的双密钥加密函数，加密密钥和解密密钥不相同。
- 通信双方是用户A为发信方，用户B为接收方，用户B接收到信息后，通过解密来判断信息是否来自A、信息是否是完整的、有无窜扰。

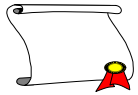
对称密钥（秘密密钥）加密技术

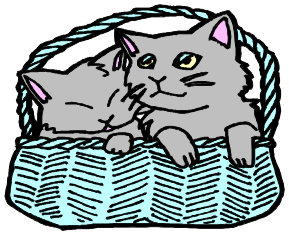
加密算法与解密算法是相同的且是公开的，加密密钥与解密密钥是同一个且不可被推算出来的单向函数算法。密钥的生成与发送需严格管理。



(a) 常规加密：具有机密性，可认证

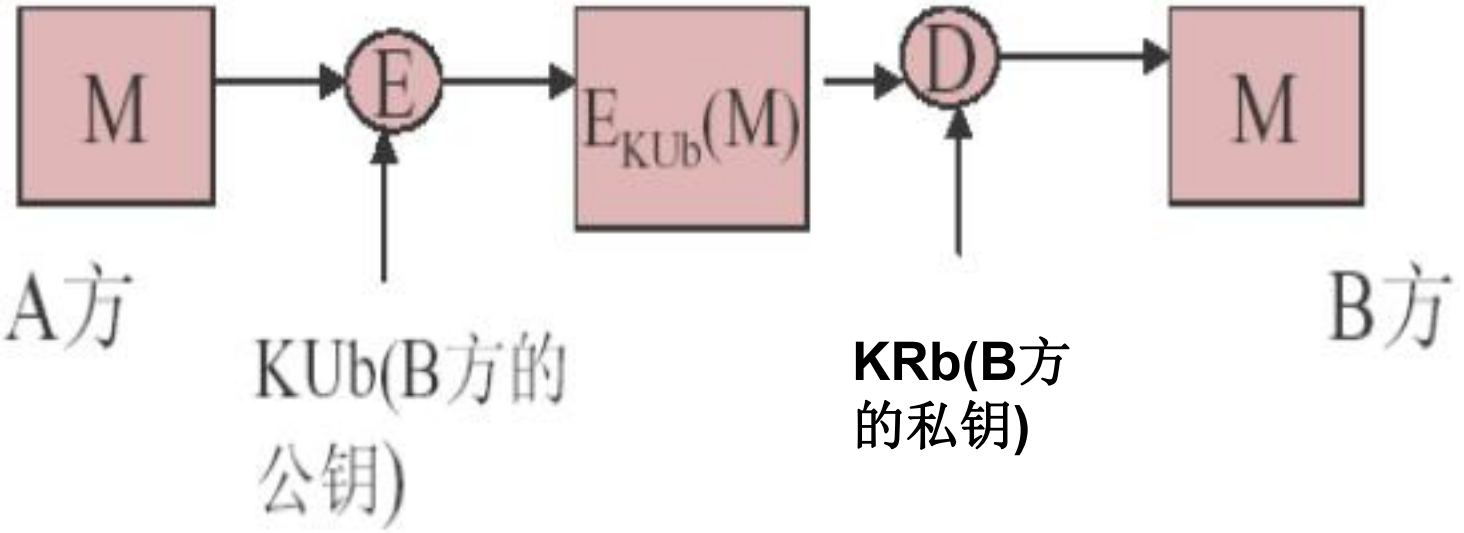
什么是公开密钥加密技术



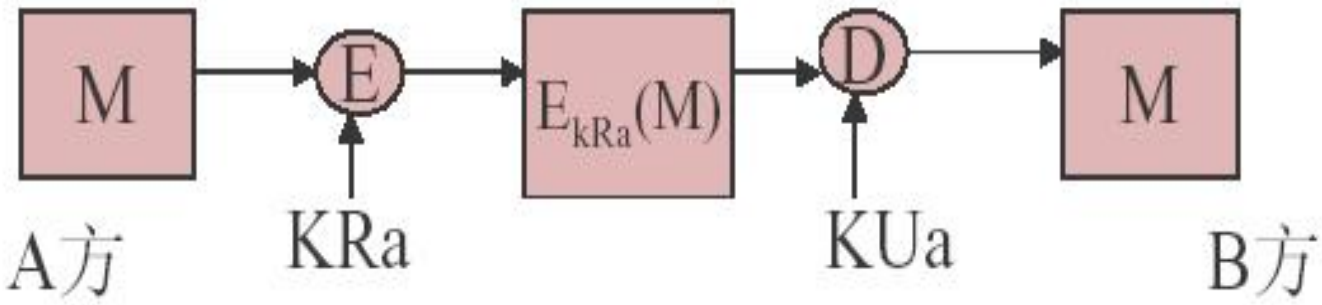


加密密钥是公开的，解密密钥同时是私钥，是保密的. 解密密钥不可被推算出来的单向函数算法。所有的通信仅涉及公钥，而任何私钥不会被传输。

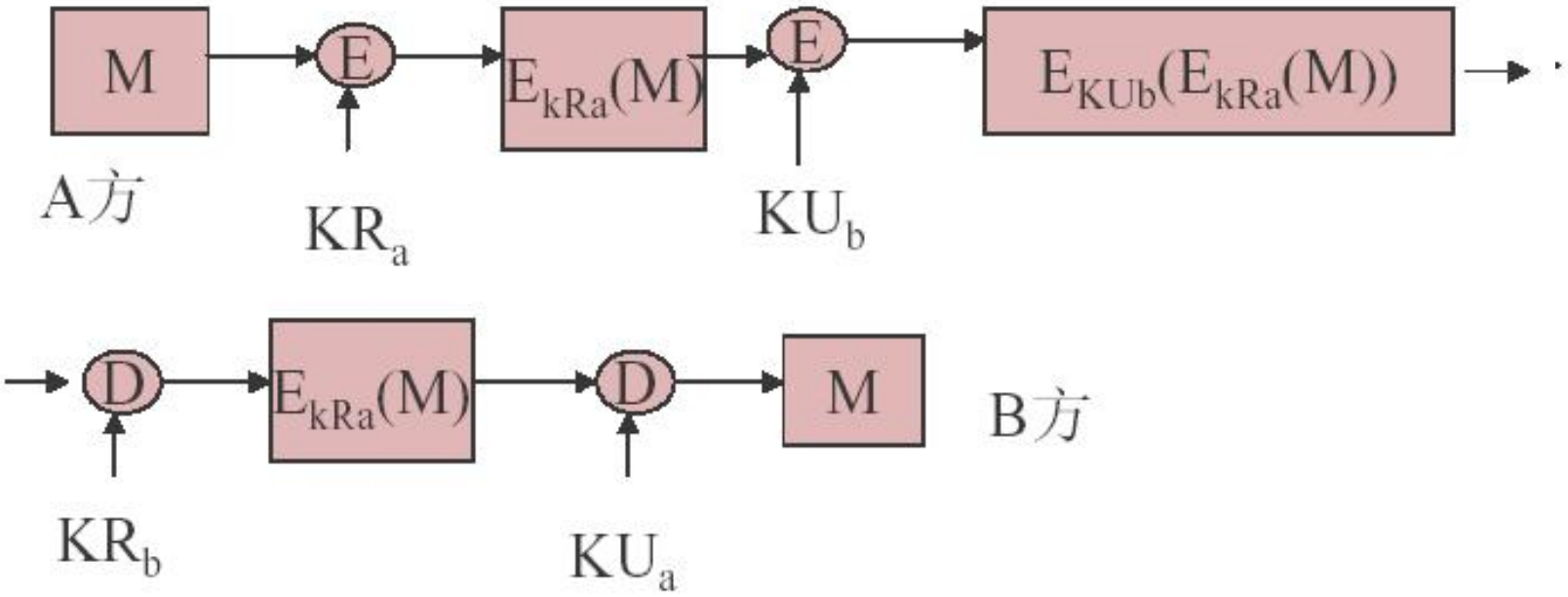
- 公有密钥/私有密钥加密
  - 公有密钥/私有密钥加密是非对称密钥加密的典型例子；
- 使用公有密钥/私有密钥系统的安全主机都有一个公共密钥和一个私有密钥，公钥是公开发布的，私钥是不公开的；
  - 所有和该主机进行安全通信的设备都可以使用该主机的公有密钥对数据进行加密后发送给该主机；该主机用其私有密钥对接收到的数据进行解密；
  - 私有密钥是该主机惟一的，并且是保密的；其他主机没有该主机的私有密钥就无法对数据进行解密，这样数据的机密性就有了保证；
- 公有密钥/私有密钥系统提供了保密性和认证功能；



(b) 公钥加密：具有机密性



(c) **私钥加密：** 认证和签名



(d)公私钥加密:机密性，可认证和签名

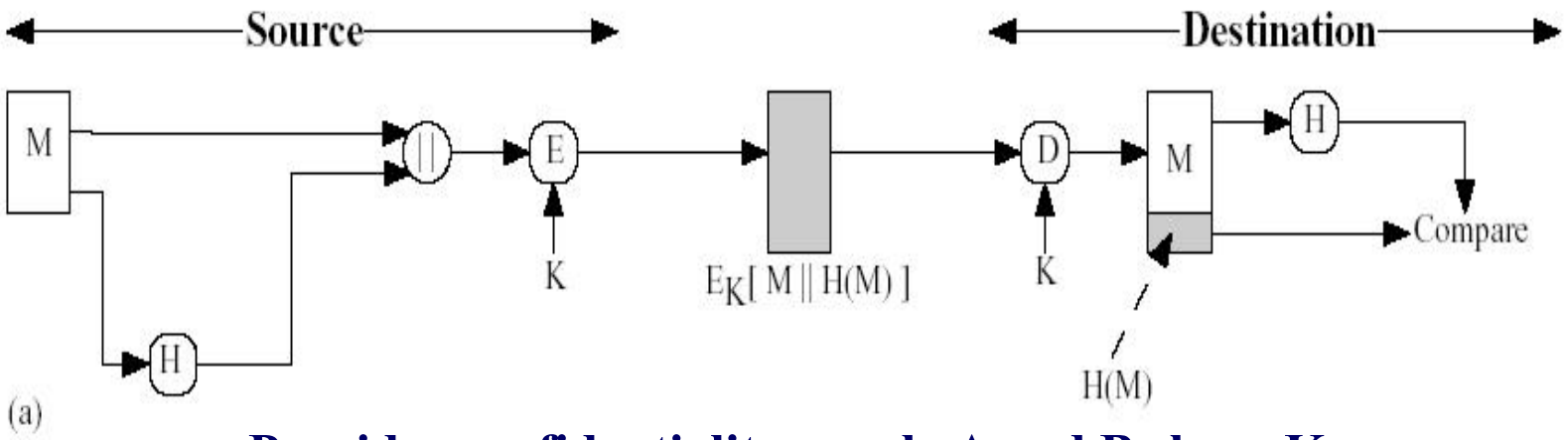
散列函数Hash Function

- H(M): 输入为任意长度的消息M; 输出为一个固定长度的散列值，称为消息摘要Message Digest）。
- 这个散列值是消息M的所有位的函数并提供错误检测能力：消息中的任何一位或多位的变化都将导致该散列值的变化。
- 又称为：哈希函数、数字指纹（Digital finger print)、压缩（Compression)函数、紧缩（Contraction ）函数、数据鉴别码DAC（Data authentication code）、篡改检验码MDC(Manipulation detection code)

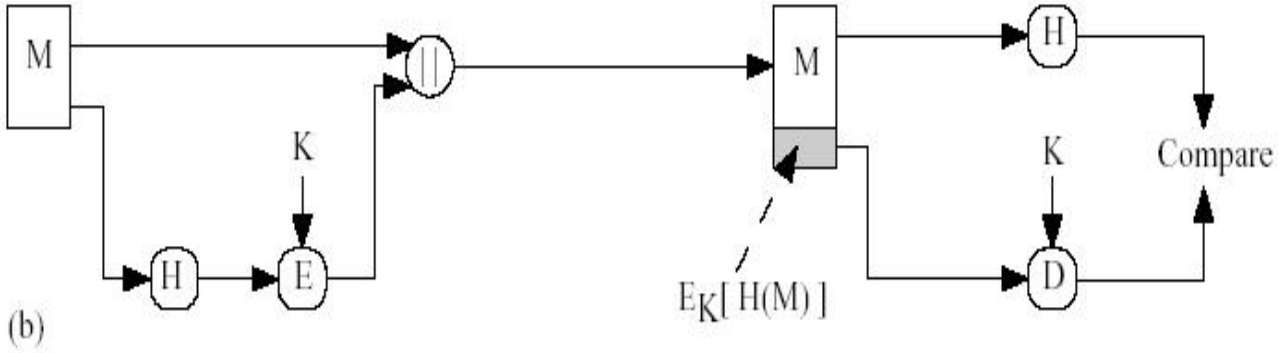
几种常用的HASH算法

- MD5
- SHA-1、SHA-3
- RIPEMD-160
- HMAC

散列函数的基本用法（a、b）

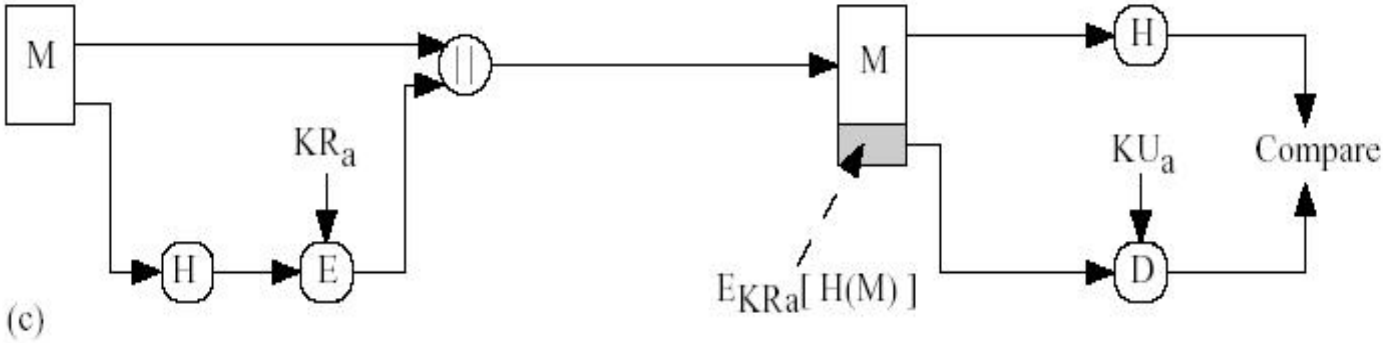


Provides confidentiality -- only A and B share K
Provides authentication -- H(M) is cryptographically protected



Provides authentication -- H(M) is cryptographically protected

散列函数的基本用法（c）

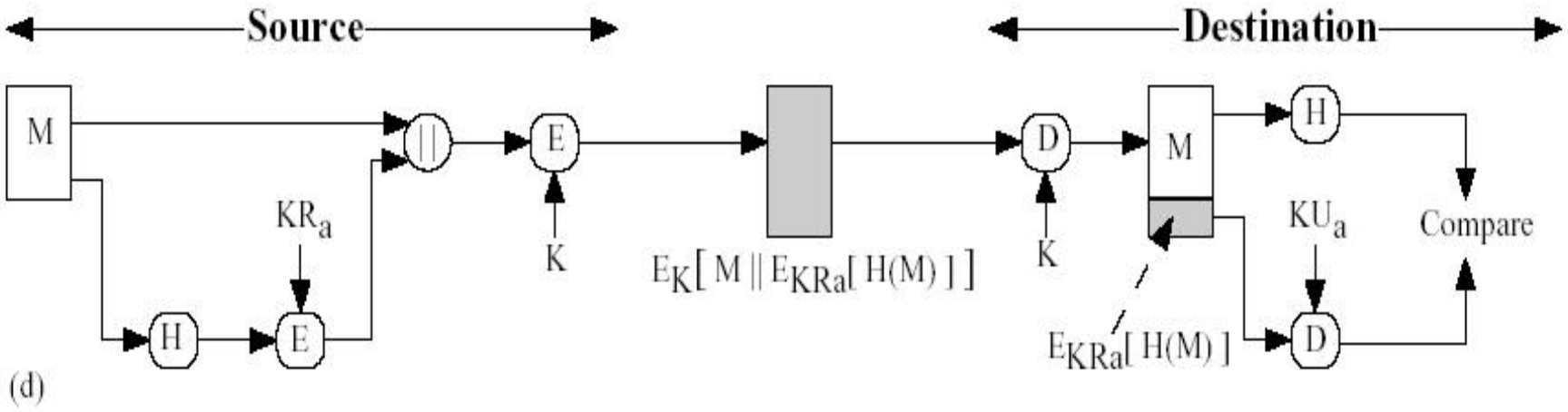


Provides authentication and digital signature

-- H(M) is cryptographically protected

-- only A could create $E_{KRa}[H(M)]$

散列函数的基本用法(d)



(d) A→B: $E_K[M||E_{KRa}[H(M)]]$

Provides authentication and digital signature

Provides confidentiality

hash函数小结

- **hash函数把变长信息映射到定长信息**
- **hash函数不具备可逆性**
- **hash函数速度较快**
- **hash函数与对称密钥加密算法有某种相似性**
- **对hash函数的密码分析比对称密钥密码更困难**
- **hash函数可用于消息摘要，完整性验证**
- **hash函数可用于数字签名**

消息鉴别码MAC

- MAC：
  - 使用一个密钥生成一个固定大小的小数据块，并加入到消息中，称消息鉴别码（MAC）， 或密码校验和（cryptographic checksum）
  - MAC=C(K,M)生成，其中M是变长的报文，K是仅由收发双方共享的密钥，生成的MAC是定长的认证码
  - 接收者可以确信消息M未被改变
  - MAC函数类似于加密函数，主要用于消息的校验和鉴别。

消息鉴别码(Message Authentication Code)也叫 密码校验和（cryptographic checksum）

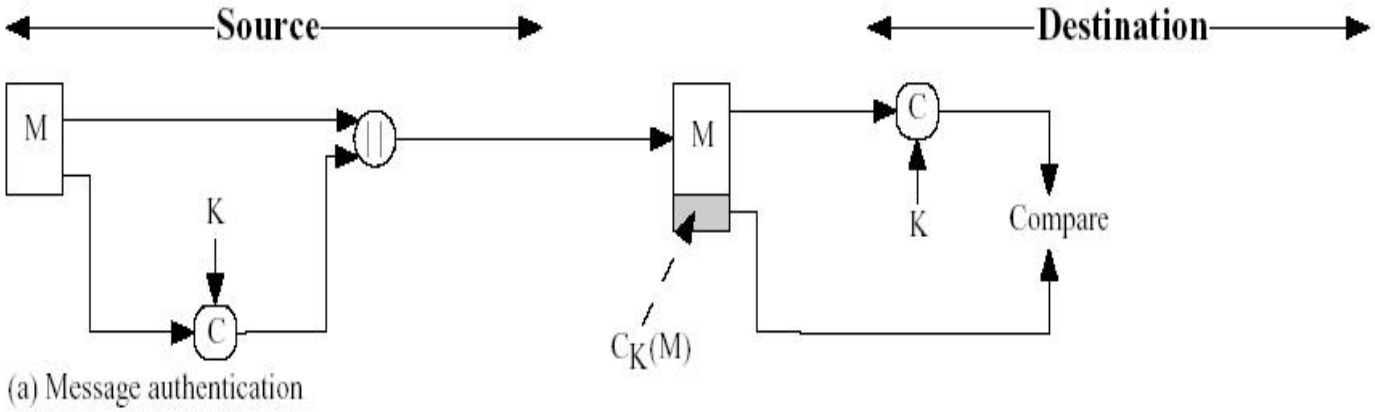
鉴别函数的一种.

消息鉴别码实现鉴别的原理是,用公开函数和密钥产生一个固定长度的值作为认证标识,用这个标识鉴别消息的完整性.使用一个密钥生成一个固定大小的小数据块,即MAC，并将其加入到消息中,然后传输.接收方利用与发送方共享的密钥进行鉴别认证等.

使用消息鉴别码进行认证：消息鉴别码（MAC）也称为密码校验值，是对数据单元进行加密变换所得到的信息。它由MAC=C(K,M)生成，其中M是变长的报文，K是仅由收发双方共享的密钥，生成的MAC是定长的认证码，发送者在发送消息时，将计算好的认证码附加到消息的末尾发送，接收方根据接收到的消息，计算出鉴别码，并与附在消息后面的认证码进行比较。

哈希函数认证：哈希函数是将任意长的数字串M映射成一个较短的定长输出数字串H的函数。以h表示哈希函数，则有H=h(M)。其中H称为M的哈希码，有时也称为杂凑码、数字指纹、消息摘要等。哈希函数与MAC的区别是，哈希函数的输入是消息本身，没有密钥参与。

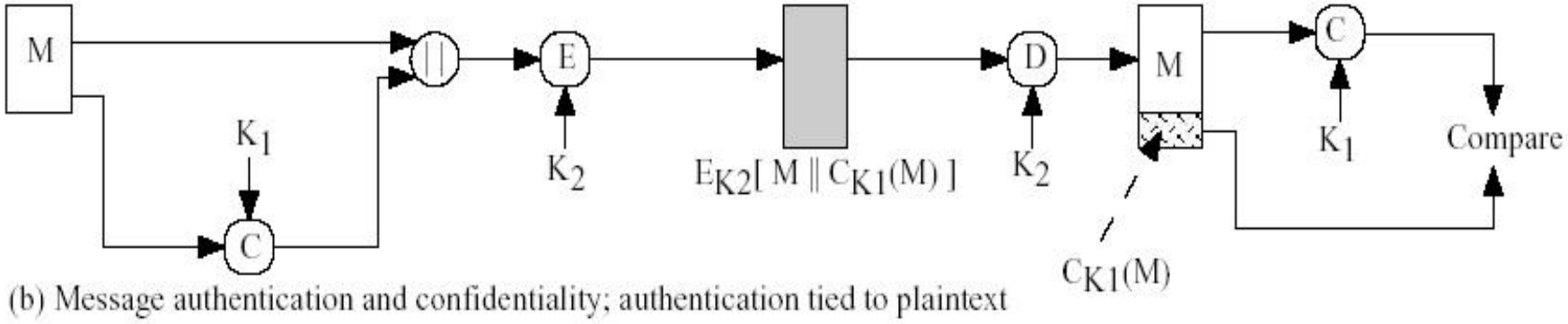
MAC的基本用法(a)



消息鉴别

Provides authentication ---- only A and B share K

MAC的基本用法(b)

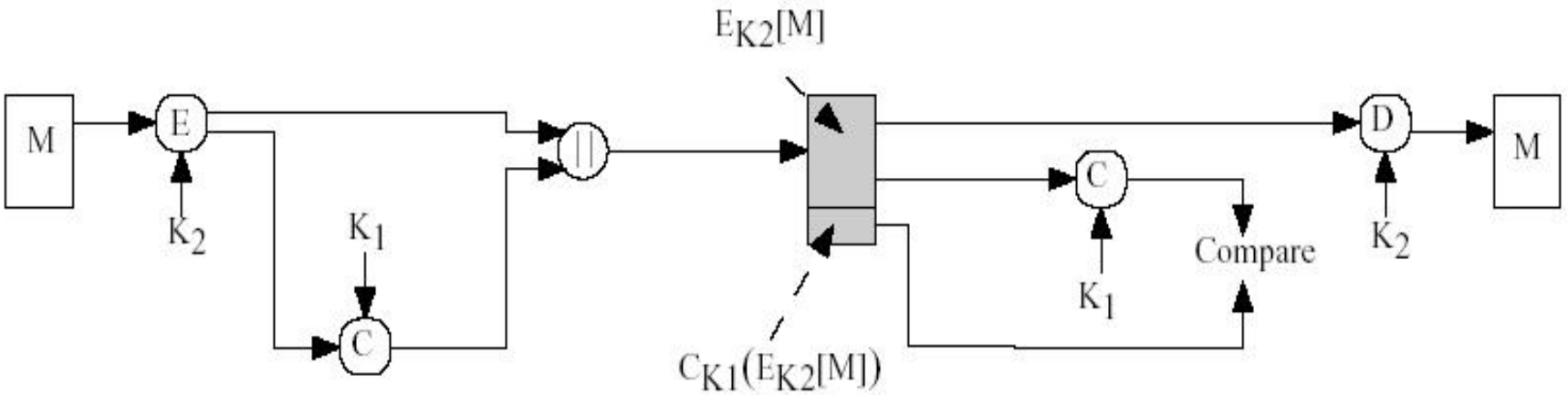


(b) Message authentication and confidentiality; authentication tied to plaintext

消息鉴别与保密，鉴别与明文连接

Provides authentication -- only A and B share K1
Provides confidentiality -- only A and B share K2

MAC的基本用法(c)



(c) Message authentication and confidentiality; authentication tied to ciphertext

消息鉴别与保密，鉴别与密文连接

Provides authentication -- Using K1
Provides confidentiality -- Using K2

数字签名分类

- 以签名方式分
  - ①直接数字签名direct digital signature
  - ②仲裁数字签名arbitrated digital signature

直接数字签名（DDS）

(1) A→B: E_{KRa}[M]

提供了鉴别与签名：

- **只有A具有KRa进行加密；**
- **传输中没有被篡改；**
- **需要某些格式信息/冗余度；**
- **任何第三方可以用KUa 验证签名**

(1') A→B: E_{KUb} [E_{KRa}(M)]

提供了保密(KUb)、鉴别与签名(KRa)：

直接数字签名

(2) A→B: $M||E_{KRa}[H(M)]$

提供鉴别及数字签名

-- H(M) 受到密码算法的保护；

-- 只有 A 能够生成 $E_{KRa}[H(M)]$

(2') A→B: $E_K[M||E_{KRa}[H(M)]]$

提供保密性、鉴别和数字签名。

直接数字签名的缺点

- 验证模式依赖于发送方的密钥；
  - 发送方要抵赖发送某一消息时，可能会声称其私有密钥丢失或被窃，从而他人伪造了他的签名。
    - 通常需要采用与私有密钥安全性相关的行政管理控制手段来制止或至少是削弱这种情况，但威胁在某种程度上依然存在。
  - 改进的方式例如可以要求被签名的信息包含一个时间戳（日期与时间），并要求将已暴露的密钥报告给一个授权中心。
- X的某些私有密钥确实在时间T被窃取，敌方可以伪造X的签名及早于或等于时间T的时间戳。

仲裁数字签名

- 引入仲裁者。
  - 所有从发送方X到接收方Y的签名消息首先送到仲裁者A
  - A将消息及其签名进行一系列测试，以检查其来源和内容
  - 后将消息加上日期并与已被仲裁者验证通过的指示一起发给Y。
- 仲裁者在这一类签名模式中扮演敏感和关键的角色。
  - 所有的参与者必须极大地相信这一仲裁机制工作正常。（trusted system）

仲裁数字签名技术

(a) 单密钥加密方式，仲裁者可以看见消息

(1) X→A：$M||E_{K_{xa}}[ID_x||H(M)]$

(2) A→Y：$E_{K_{ay}}[ID_x||M||E_{K_{xa}}[ID_x||H(M)]||T]$

X与A之间共享密钥K_{xa}，Y与A之间共享密钥K_{ay}；

X：准备消息M，计算其散列码H(M)，用X的标识符ID_x和散列值构成签名，并将消息及签名经K_{xa}加密后发送给A；

A：解密签名，用H(M)验证消息M，然后将ID_x，M，签名，和时间戳一起经K_{ay}加密后发送给Y；

Y：解密A发来的信息，并可将M和签名保存起来。

解决纠纷：

Y：向A发送 $E_{K_{ay}}[ID_x||M||E_{K_{xa}}[ID_x||H(M)]]$

A：用K_{ay}恢复ID_x，M，和签名（$E_{K_{xa}}[ID_x||H(M)]$），然后用K_{xa}解密签名并验证散列码

特点：

在这种模式下Y不能直接验证X的签名，Y认为A的消息正确，只因为它来自A。因此，双方都需要高度相信A:

- X必须信任A没有暴露K_{xa}，并且没有生成错误的签名

$$E_{K_{xa}}[ID_x \| H(M)]$$

- Y必须信任A仅当散列值正确并且签名确实是X产生的情况下才发送的 $E_{K_{ay}}[ID_x \| M \| E_{K_{xa}}[ID_x \| H(M)] \| T]$
- 双方都必须信任A处理争议是公正的。

只要A遵循上述要求，则X相信没有人可以伪造其签名；Y相信X不能否认其签名。

上述情况还隐含着A可以看到X给Y的所有信息，因而窃听者也可能看到。

(b) 单密钥加密方式，仲裁者不可以看见消息

(1) $X \rightarrow A$： $ID_x \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_x \| H(E_{K_{xy}}[M])]$

(2) $A \rightarrow Y$： $E_{K_{ay}}[ID_x \| E_{K_{xy}}[M] \| E_{K_{xa}}[ID_x \| H(E_{K_{xy}}[M])] \| T]$

在这种情况下，X与Y之间共享密钥K_{xy}，

X：将标识符ID_x ,密文 $E_{K_{xy}}[M]$，以及对ID_x和密文消息的散列码用K_{xa}加密后形成签名发送给A。

A：解密签名，用散列码验证消息，这时A只能验证消息的密文而不能读取其内容。然后A将来自X的所有信息加上时间戳并用K_{ay}加密后发送给Y。

(a)和(b)共同存在一个共性问题：

A和发送方联手可以否认签名的信息；

A和接收方联手可以伪造发送方的签名；

(c) 双密钥加密方式，仲裁者不可以看见消息

(1) $X \rightarrow A$： $ID_x \| E_{KR_x}[ID_x \| E_{KU_y}(E_{KR_x}[M])]$

(2) $A \rightarrow Y$： $E_{KR_a}[ID_x \| E_{KU_y}[E_{KR_x}[M]] \| T]$

X：对消息M双重加密：首先用X的私有密钥KRx，然后用Y的公开密钥KUy。形成一个签名的、保密的消息。然后将该信息以及X的标识符一起用KRx签名后与IDx 一起发送给A。这种内部、双重加密的消息对A以及对除Y以外的其它人都是安全的。

A：检查X的公开/私有密钥对是否仍然有效，是，则确认消息。并将包含IDx、双重加密的消息和时间戳构成的 消息用KRa签名后发送给Y。

本模式比上述两个模式具有以下好处：

1、在通信之前各方之间无须共享任何信息，从而避免了联手作弊；

2、即使KRx 暴露，只要KRa 未暴露，不会有错误标定日期的消息被发送；

3、从X发送给Y的消息的内容对A和任何其他人是保密的。

身份鉴别

鉴别Authentication

- **The property that ensures that the identity of a subject or resource is the one claimed. Authenticity applies to entities such as users, processes, systems and information.**
- 鉴别就是确认实体是它所声明的。
- 鉴别是最重要的安全服务之一。鉴别服务提供了关于某个实体身份的保证。（所有其它的安全服务都依赖于该服务）
- 鉴别可以对抗假冒攻击的危险

鉴别的需求和目的

- 问题的提出
  - 身份欺诈
- 鉴别需求：
  某一成员（声称者）提交一个主体的身份并声称它是那个主体。
- 鉴别目的：
  使别的成员（验证者）获得对声称者所声称的事实的信任。

身份鉴别

- 定义：证实客户的真实身份与其所声称的身份是否相符的过程。
- 依据：
  - Something the user know (所知）
    - 密码、口令等
  - Something the user possesses （拥有）
    - 身份证、护照、密钥盘等
  - Something the user is (or How he behaves)
    - 指纹、笔迹、声音、虹膜、DNA等

公开密钥的管理

- 公钥密码体制的密钥分配要求与对称密码体制的密钥分配要求有着本质的差别：
  当分配一个公钥时，不需要机密性。然而，公钥的完整性是必需的。

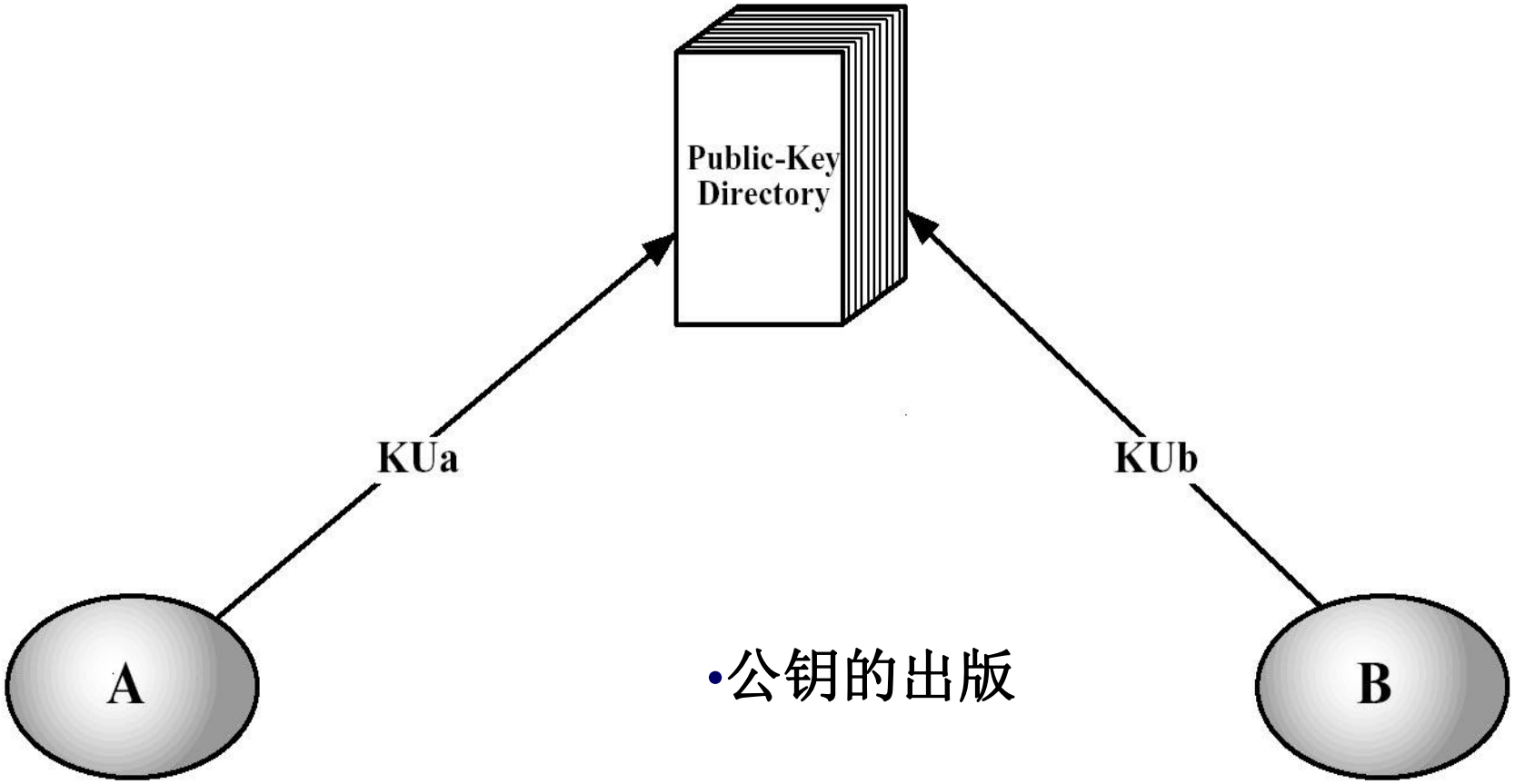
公开密钥的分配

- 公开宣布 **Public announcement**
- 公开可以得到的目录**Publicly available directory**
- 公开密钥管理机构**Public-key authority**
- 公钥证书**Public key certificates**

Public announcement

- 直接把公钥散发出去
  - 如使用PGP并且把公钥附上发送给公开论坛。
  - 一个缺点：任何人都可以伪造一个这样的公开告示。
    - 也就是说，某个用户可能假装是用户A，并发送一个公开密钥给另一个参与者或者广播这样一个公开密钥。
    - 直到用户A发觉了依靠并警告其他参与者

Publicly available directory



- 公钥的出版

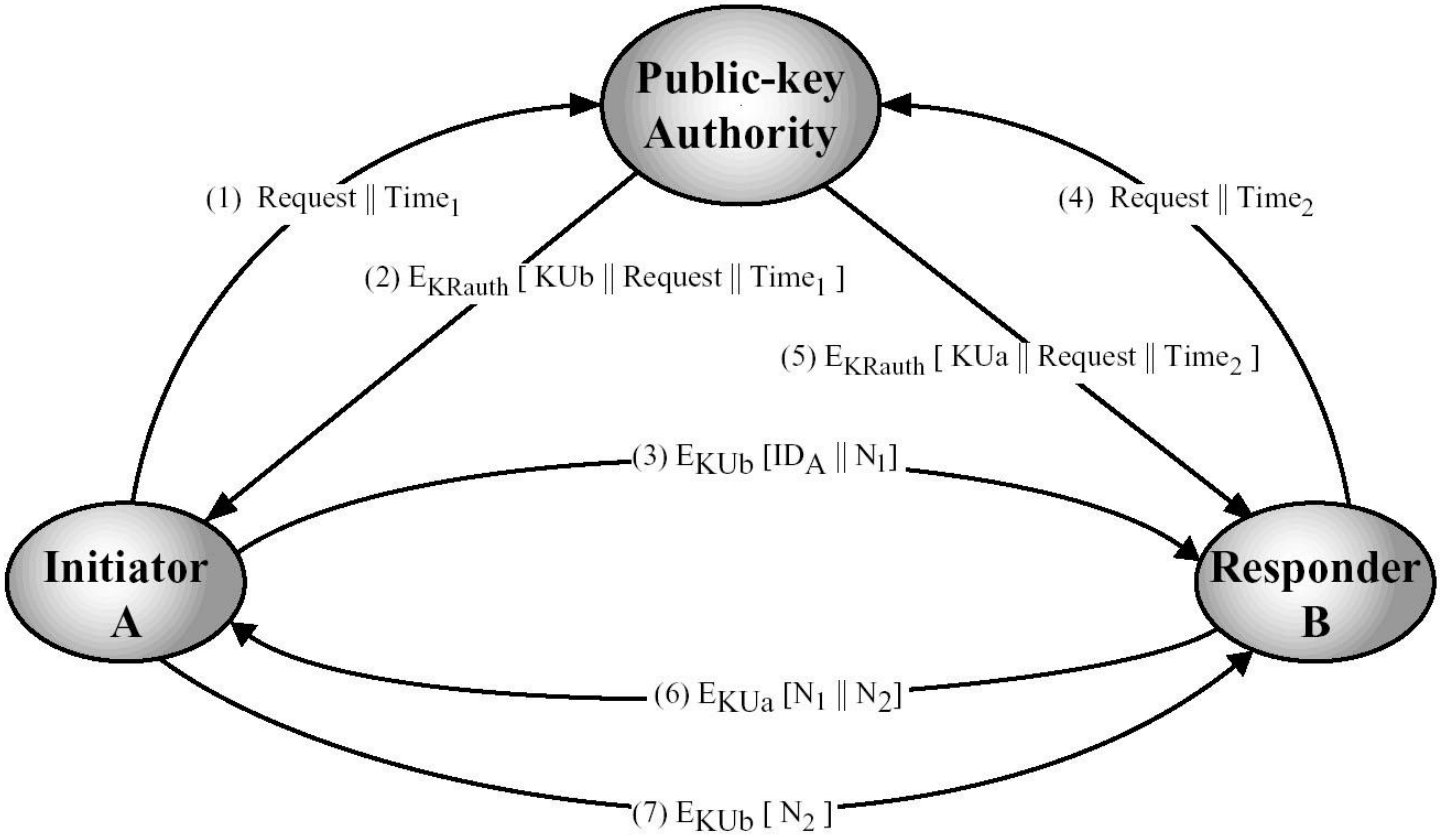
Publicly available directory

- 需要可信任的中央授权机构
  - 授权机构维护着动态{name,public key}列表
  - 用户在授权机构注册其public key(安全通道)
  - 用户可以替换其public key
  - 授权机构定期发布或更新整个目录
  - 用户可在网络上直接访问公共目录(安全通道)

Publicly available directory

- 这个方案明显比各个参与者单独进行公开告示更加安全，但是它仍然有弱点。
  - 如果敌对方成功地得到或者计算出了目录管理机构的私有密钥，敌对方就可以堂皇地散发伪造的公开密钥，并随之假装成任何一个参与者并窃听发送给该参与者的报文。
  - 另一个达到同样目的的方法是敌对方篡改管理机构维护的记录。

Public-key authority



Public-key Distribution Scenario

Public-key authority

- 缺点：
  - 公开密钥管理机构可能是系统中的一个瓶颈，因为一任用户对于他所希望联系的其他用户都必须借助于管理机构得到公开密钥
  - 管理机构所维护的名字和公开密钥目录也可能被篡改。

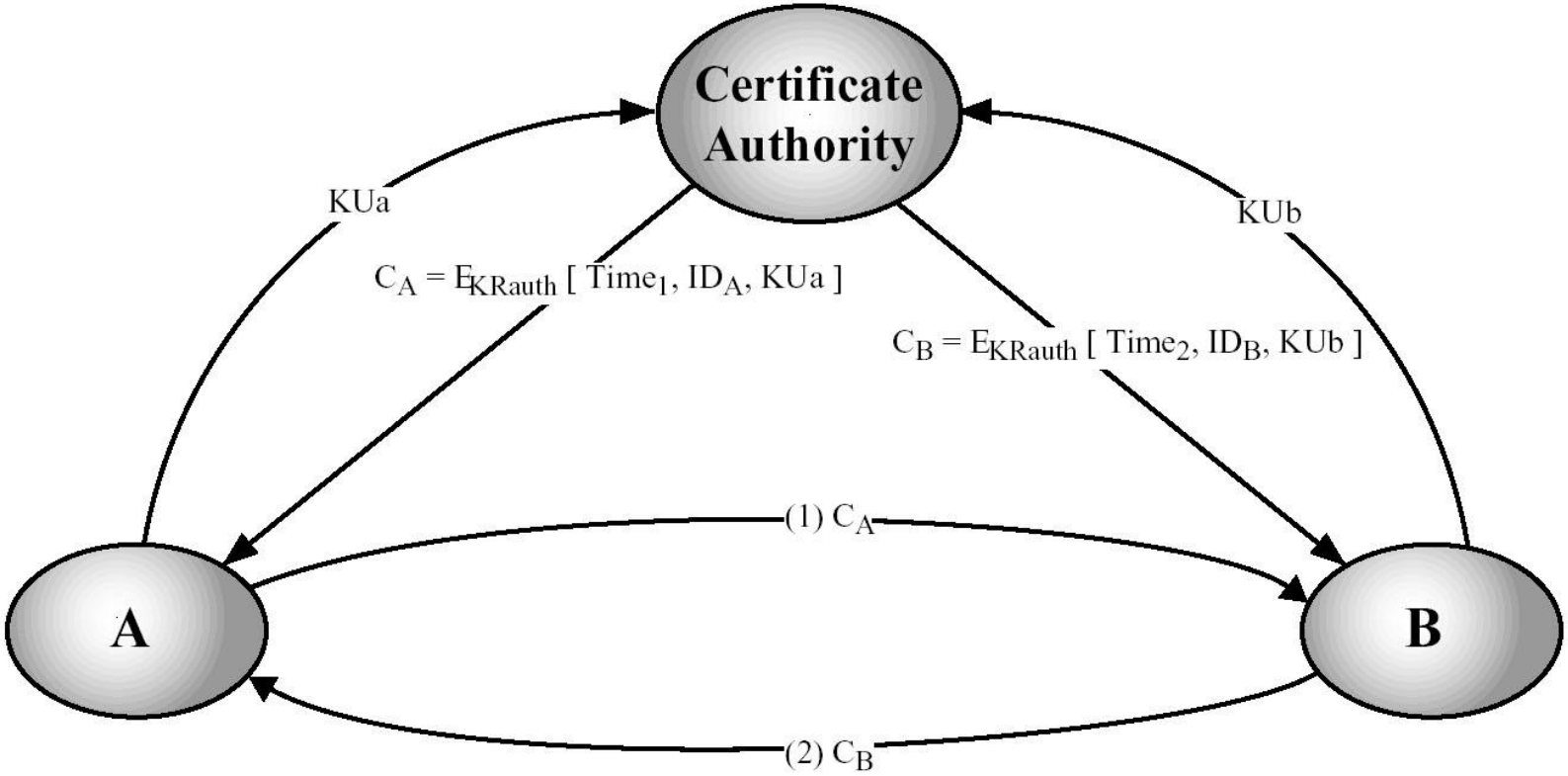
Public-key certificates

- 数字证书：参与者使用这个证书在联系一个公开密钥管理机构之前就可以交换密钥。
  - 每个证书包含一个公开密钥以及其他信息，它由一个证书管理机构制作，并发给具有相匹配的私有密钥的参与者。
  - 一个参与者通过传输它的证书将其密钥信息传送给另一个参与者，其他参与者可以验证证书是否是管理机构制作的。

Public-key certificates

- 对这种方案的要求：
  - 任何人可以阅读证书以确定证书拥有者的姓名和公钥
  - 任何人可以验证证书是由授权机构发出而非伪造的
  - 只有授权机构才可以发行和更新证书
  - 任何人可以验证证书的时效性

Exchange of public-key certificates



一种混合方案

- 使用KDC
  - KDC与每个用户共享主密钥(对称密钥密码)
  - KDC与每个用户都拥有{公钥,私钥}对，其分发由主密钥完成
  - 会话密钥的分发由主密钥或公钥完成
  - 主密钥更新由公钥完成

认证协议 —— 鉴别协议

- 相互鉴别 (**mutual authentication**)
- 单向鉴别 (**one-way authentication**)

相互鉴别协议

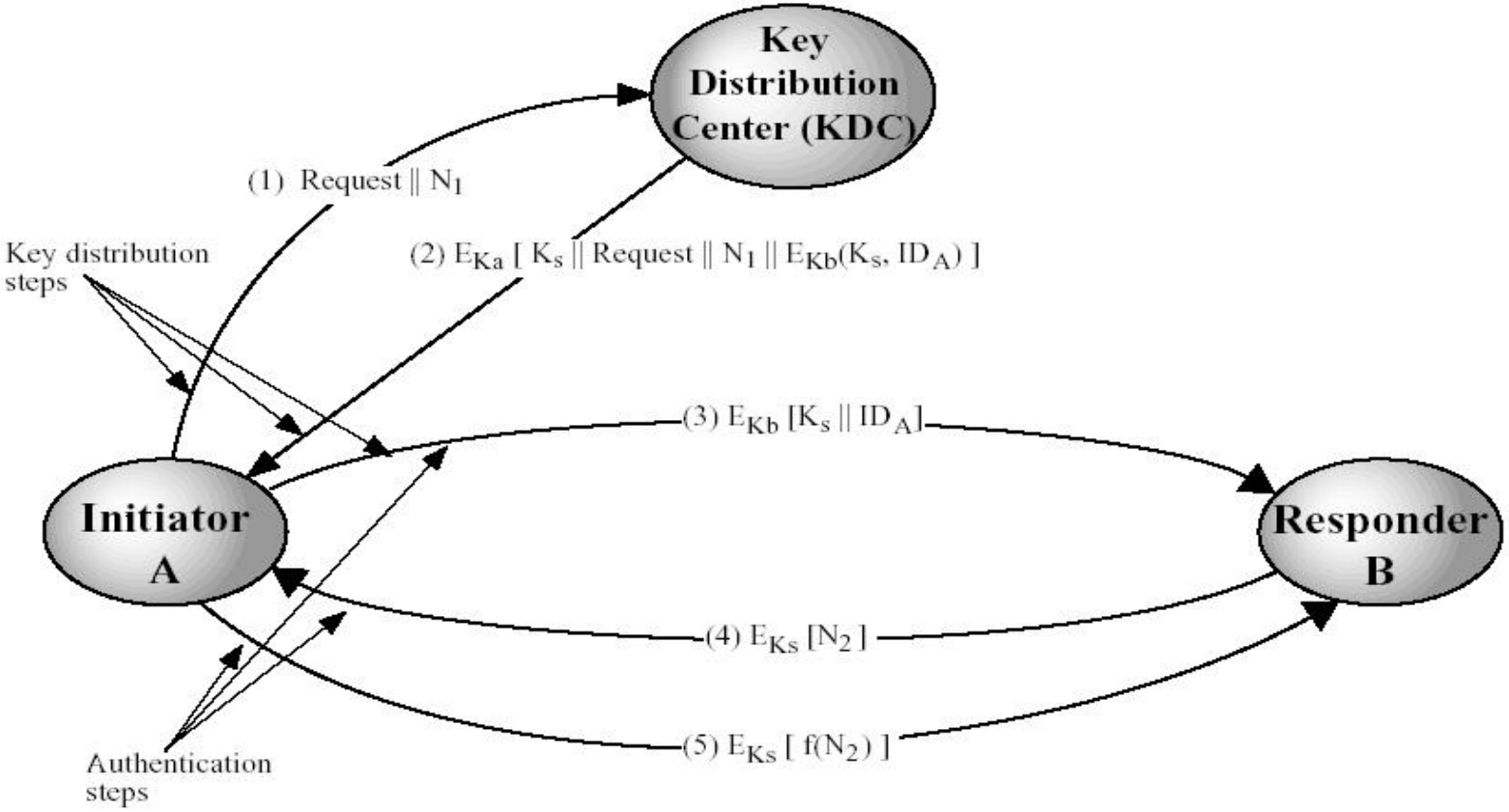
- 最常用的协议。该协议使得通信各方互相认证鉴别各自的身份，然后交换会话密钥。
- 基于鉴别的密钥交换核心问题有两个：
  - 保密性
  - 时效性

为了防止伪装和防止暴露会话密钥，基本鉴别和会话密码信息必须以保密形式通信，这就要求预先存在保密或公开密钥供实现加密使用。第二个问题也很重要，因为涉及防止消息重放攻击。

相互鉴别协议

- 传统加密方法
  - Needham/Schroeder Protocol [1978]
  - Denning Protocol [1982]
  - KEHN92
- 公钥加密方法
  - 一个基于临时值握手协议：WOO92a
  - 一个基于临时值握手协议：WOO92b

Needham/Schroeder Protocol



Needham/Schroeder Protocol [1978]

- 保密密钥Ka和Kb分别是A和KDC、B和KDC之间共享的密钥。本协议的目的就是要安全地分发一个会话密钥Ks给A和B。
  - A在第2步安全地得到了一个新的会话密钥，
  - 第3步只能由B解密、并理解。
  - 第4步表明B已知道Ks了。
  - 第5步表明B相信A知道Ks并且消息不是伪造的。
- 第4，5步目的是为了防止某种类型的重放攻击。特别是，如果敌方能够在第3步捕获该消息，并重放之，这将在某种程度上干扰破坏B方的运行操作。

1、A → KDC：$ID_A||IDB||N1$
2、KDC → A：
$EKa[Ks||ID_B||N1||EKb[Ks||ID_A]]$
3、A → B：　$EKb[Ks||ID_A]$
4、B → A：　EKs[N2]
5、A → B：　EKs[f(N2)]

Denning Protocol [1982] 改进

- 这种修改包括在步骤2和步骤3中增加时间戳。协议假定主密钥Ka和Kb是安全的。

1、A → KDC：$ID_A||ID_B$
2、KDC → A：$E_{Ka}[Ks||ID_B||T||E_{Kb}[Ks||ID_A||T]]$
3、A → B：$E_{Kb}[Ks||ID_A||T]$
4、B → A：$E_{Ks}[N1]$
5、A → B：$E_{Ks}[f(N1)]$

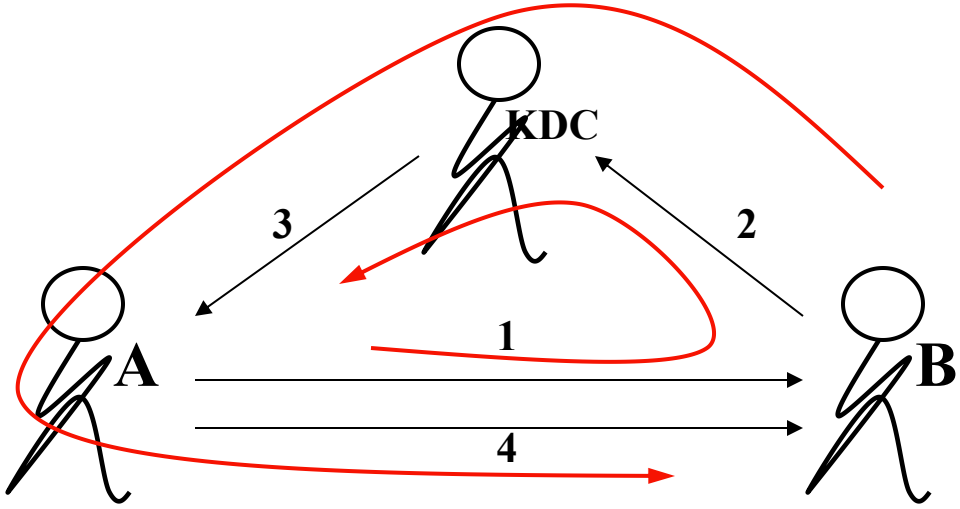
| Clock - T | < $\Delta t1 + \Delta t2$
其中：$\Delta t1$ 是KDC时钟与本地时钟（A或B）之间差异的估计值；
$\Delta t2$ 是预期的网络延迟时间。

Denning Protocol [1982] 改进

- **Denning Protocol**比**Needham/Schroeder Protocol**在安全性方面增强了一步。然而，又提出新的问题：这个新机制需要信任通过网络进行同步的时钟。
- 那么就存在一种危险，这种危险基于这样的事实：分布时钟由于阴谋破坏或同步时钟、同步机制的故障变得不同步。当发方的时钟快于预想的收方时钟时，这个问题就会发生，在这种情况下，对手可能截获发自A的报文，当报文中的时间戳变成收方当前时间时就重放该报文。这种重放可导致不可预料的结果。（称为抑制重放攻击）。
- 一种克服抑制重放攻击的方法是强制各方定期检查自己的时钟是否与KDC的时钟同步。
- 另一种避免同步开销的方法是采用临时数握手协议。

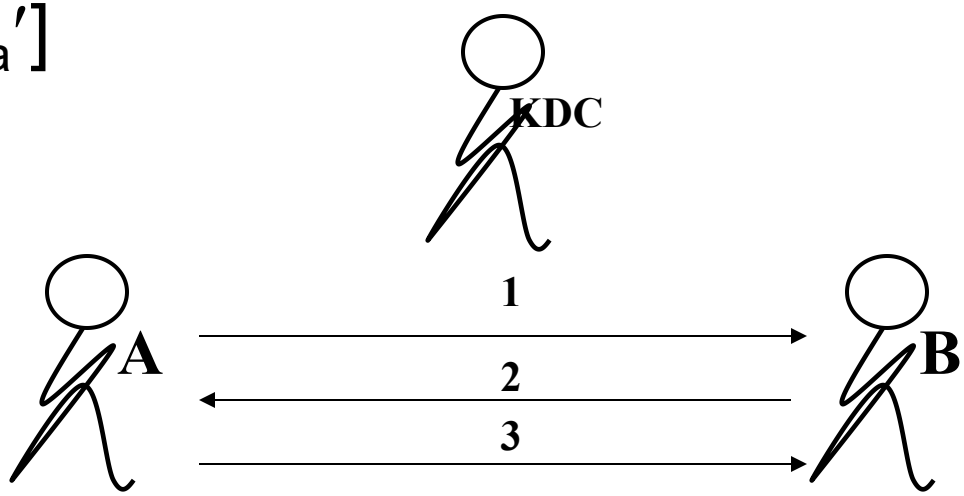
Kehn92

1. A → B： $ID_A||N_a$
2. B → KDC ： $ID_B||N_b$ || $E_{Kb}[ID_A || N_a || T_b]$
3. KDC → A ： $E_{Ka}[ID_B||N_a ||K_s|| T_b]$ || $E_{Kb}[ID_A || K_s || T_b]$ || N_b
4. A → B ： $E_{Kb}[ID_A || K_s || T_b]$ || $E_{Ks}[N_b]$



关于Kehn92协议

- $E_{Kb}[ID_A \| K_s \| T_b]$相当于一个ticket
- 如果A要再次访问B，可以不再通过KDC
  - $A \rightarrow B$：$E_{Kb}[ID_A \| K_s \| T_b] \| N_a'$
  - B检查ticket是否在有效时间，若是，则
    $B \rightarrow A$：$N_b' \| E_{Ks}[N_a']$
  - $A \rightarrow B$：$E_{Ks}[N_b']$



公开密钥加密方法

- 一个使用时间戳的方法

1、$A \rightarrow AS$：$ID_A||ID_B$

2、$AS \rightarrow A$：$E_{KR_{as}}[ID_A||KU_a||T] || E_{KR_{as}}[ID_B||KU_b||T]$

3、$A \rightarrow B$：$E_{KR_{as}}[ID_A||KU_a||T] || E_{KR_{as}}[ID_B||KU_b||T] || E_{KU_b}[E_{KR_a}[K_s||T]]$

在这个协议中，中心系统被称为鉴别服务器(AS)，因为实际上它并不负责密钥的分配。AS实际上提供公开密钥证书。会话密钥的选择和加密由A完成；因此没有AS泄漏密钥的危险。时间戳防止危及密钥安全的重放攻击。

这个协议是紧凑的，但和前面的一样，也需要时钟的同步。

一个基于临时值握手协议：WOO92a

1、$A \rightarrow KDC$：$ID_A||ID_B$

2、$KDC \rightarrow A$：$E_{KR_{auth}}[ID_B||KU_b]$

3、$A \rightarrow B$：$E_{KU_b}[N_a||ID_A]$

4、$B \rightarrow KDC$：$ID_B||ID_A || E_{KU_{auth}}[N_a]$

5、$KDC \rightarrow B$：$E_{KR_{auth}}[ID_A||KU_a]|| E_{KU_b}[E_{KR_{auth}}[N_a||K_s||ID_B]]$

6、$B \rightarrow A$：$E_{KU_a}[E_{KR_{auth}}[N_a||K_s||ID_B]||N_b]$

7、$A \rightarrow B$：$E_{K_s}[N_b]$

步骤1：A通知KDC它打算与B建立一个安全连接。

步骤2： KDC向A返回B的公开密钥证书。

步骤3：使用B的公开密钥，A通知B它期望进行通信并发送临时值Na。

步骤4：B向KDC请求A的公开密钥证书和一个会话密钥；包含A的临时值以便使KDC能使用该临时值对会话密钥进行标记，这个临时值使用KDC的公开密钥进行保护。

步骤：KDC向B返回一个A的公开密钥证书以及信息［Na,Ks,IDB］，表明Ks是由KDC代表B产生并绑定到Na，将使A确保Ks是最新的。为了让B能证实该元组确实来自KDC，使用KDC的私有密钥对这个元组加密，也用B的公开密钥进行加密。

步骤6：B用KDC私有密钥加密的元组{Na,Ks,IDB}连同B生成的临时值Nb用A的公开密钥加密后再发送给A。

步骤7： A得到会话密钥Ks，用它对Nb加密后发回给B。使B确信A已获得了会话密钥。

一个基于临时值握手协议：WOO92b

1、$A \rightarrow KDC$：$ID_A||ID_B$

2、$KDC \rightarrow A$：$E_{KR_{auth}}[ID_B||KU_b]$

3、$A \rightarrow B$：$E_{KU_b}[N_a||ID_A]$

4、$B \rightarrow KDC$：$ID_B||ID_A || E_{KU_{auth}}[N_a]$

5、$KDC \rightarrow B$：$E_{KR_{auth}}[ID_A||KU_a]|| E_{KU_b}[E_{KR_{auth}}[N_a||K_s||ID_A||ID_B]]$

6、$B \rightarrow A$：$E_{KU_a}[E_{KR_{auth}}[N_a||K_s||ID_A||ID_B]||N_b]$

7、$A \rightarrow B$：$E_{K_s}[N_b]$

在步骤5和步骤6中，A的标识符ID_A被加到用KDC私有密钥加密的项目组中，这将会话密钥Ks与会话中通信双方的标识符绑定在一起。

内含IDA说明，可将现时Na看作所有由A产生的现时值中而不是由通信各方产生所有现时中惟一的现时，因此{IDA,Na}是A连接请求的惟一性标识符。

单向鉴别One-Way Authentication

- E-mail
  - 信封明文
  - 消息密文
  - 收发双方不同时在线
  - 要求具有认证

传统加密方法

1、$A \rightarrow KDC$：$ID_A||ID_B||N_1$

2、$KDC \rightarrow A$：$E_{K_a}[K_s \| ID_B \| N_1 \| E_{K_b}[K_s \| ID_A]]$

3、$A \rightarrow B$：$E_{K_b}[K_s \| ID_A] \| E_{K_s}[M]$

这个方法保证只有合法的接收者阅读到报文内容。它提供了发方是A这级鉴别。

正如所说明，该协议无法防止重放攻击。可以提供某些防御措施如在报文中入时间戳。然而，由于电子邮件潜在的时延，这样的时间戳作用可能有限。

公钥加密方法

1、 $A \rightarrow B$： $E_{KU_b}[K_s] \| E_{K_s}[M]$
保障机密性

2、 $A \rightarrow B$： $M \| E_{KR_a}[H(M)]$
满足需要的数字签名

3、 $A \rightarrow B$： $E_{KU_b}[M \| E_{KR_a}[H(M)]]$
满足机密性与数字签名

4、 $A \rightarrow B$： $M \| E_{KR_a}[H(M)] \| E_{KR_{as}}[T \| ID_A \| KU_a]$
带有AS的证书

基于票据的认证协议

- 一个典型的分布式认证协议
  - Kerberos协议
  - Kerberos由MIT发明。Kerkeros建立一个安全的密钥颁发中心(Key Distribution Center，KDC)。每一个用户使用自己的保密密钥与KDC进行保密通信。如果一个用户想跟另一个用户进行保密通信时，它会向KDC申请一个临时密钥。这个临时密钥只被使用在本次通信中。KDC在收到请求后，生成一个临时的密钥，并用安全通信，把这个密钥传给要与之进行通信的相应用户。

假设用户Alice想要使用服务器s,首先Alice应通过两个服务器获得向服务器**s**认证她自己的信任状（票据）。

$T_{A,B}$ = Barnum ||{Alice||Aaddr||validtime||$K_{A,B}$} K_B

代表票据颁发者对服务请求者身份的证明凭证。

其中：

- **K_B是Barnum与认证服务器共享的密钥**
- **$K_{A,B}$是Alice和Barnum共享的会话密钥**

- 当Alice希望向Barnum证明发送票据的一方与票据颁发的一方是同一方时，可以使用包含票据发送者身份的认证符：

$$A_{A,B} = \{Alice||t||kt\}\ K_{A,B}$$

其中：

- **kt是会话密钥**
- **$K_{A,B}$是Alice和Barnum共享的会话密钥**

Realm发出票据的领域（域）名，KDC只能在它自己的领域发行票据，所以也是服务器的领域名
Barnum需要验证$A_{A,G}$中的realm和$T_{A,B}$中的Aaddr是不是相符
如果$T_{A,B}$没过期，则不需要再向KDC认证，同理$T_{A,G}$。防止没需要服务一次都需要认证一次

- Alice 要使用Guttenberg服务打印一份文件。认证服务器是Cerberus,票据授权服务器是Barnum.
- $T_{A,B}$ = {Alice||Barnum||Aaddr||validtime||$K_{A,B}$} K_B
- $T_{A,G}$ = {Alice||Guttenberg||Aaddr||validtime||$K_{A,G}$} K_G
- $A_{A,G}$ ={Alice|| Guttenberg ||realm||t||}

1. Alice->Cerberus: Alice||Barnum||n
2. Cerberus->Alice: {$K_{A,B}$ ||n} K_{alice} ||$T_{A,B}$
3. Alice-> Barnum: Guttenberg||{$A_{A,G}$} $K_{A,B}$ ||$T_{A,B}$ ||n
4. Barnum->Alice: Alice||{$K_{A,G}$ ||n} $K_{A,B}$ ||$T_{A,G}$
5. Alice->Guttenberg: |{$A_{A,G}$} $K_{A,G}$ ||t || $T_{A,G}$
6. Guttenberg-> Alice: {t+1}$K_{A,G}$

- kerberos是基于对称加密方法的，它适用于一大批都属于同一个机构的用户群。互联网本身是一个很庞大、并无中心的网络。在这种环境下，建立一个KDC中心很难。所以，Kerberos在实际应用中还有很大的限制。

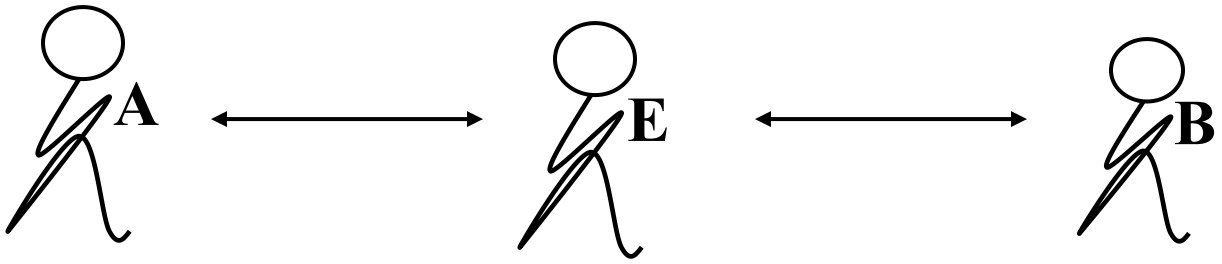
常见的攻击

常见的攻击和应对的手段

- 机密性：窃听
  - 加密
- 完整性：篡改
  - 加密、签名
- 可控性：中间人、重放
  - 先验知识、时间戳、临时值
- 不可抵赖性：发送方或接收方抵赖
  - 非否认协议

常见攻击和对策(一)

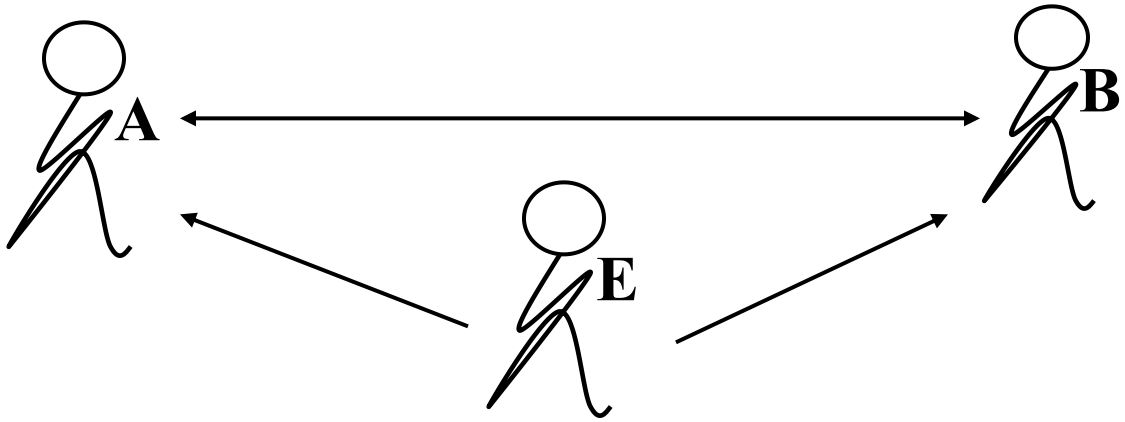
- 中间人攻击(MITM, man in the middle)



- 如果通讯双方没有任何先决条件，那么这种攻击总是存在的
- A和B的协商过程很容易受到这一类攻击
- 对策：
  - 增加A和B之间的先决知识

常见攻击和对策(二)

- 重放攻击(replay attacks)



- 偷听者可以记录下当前的通讯流量，以后在适当的时候重发给通讯的某一方，达到欺骗的目的
- 对策：
  - nonce
    - 保证通讯的唯一性
    - 增加时间戳

重放

- 常见的消息重放攻击形式有：

1、简单重放：攻击者简单复制一条消息，以后在重新发送它；
2、可被日志记录的重复：攻击者可以在一个合法有效的时间窗内重放一个带时间戳的消息；
3、不能被检测到的重复：这种情况可能出现，原因是原始信息已经被拦截，无法到达目的地，而只有重放的信息到达目的地。
4、反向重放，不做修改。向消息发送者重放。
   - 当采用传统对称加密方式时，这种攻击是可能的。
   - 因为消息发送者不能简单地识别发送的消息和收到的消息在内容上的区别。

非重复值的使用

- 1）序列号
  - 计数的策略：对付重放攻击的一种方法是在认证交换中使用一个序数来给每一个消息报文编号。仅当收到的消息序数顺序合法时才接受之。但这种方法的困难是要求双方必须保持上次消息的序号。
- 2）时间戳
  - A接受一个新消息仅当该消息包含一个时间戳，该时间戳在A看来，是足够接近A所知道的当前时间；这种方法要求不同参与者之间的时钟需要同步
- 3）验证者发送随机值（如询问）：不可预测、不重复

时间戳

- 在网络环境中，特别是在分布式网络环境中，时钟同步并不容易做到
- 一旦时钟同步失败
  - 要么协议不能正常服务，影响可用性(availability)，造成拒绝服务(DOS)
  - 要么放大时钟窗口，造成攻击的机会
- 时间窗大小的选择应根据消息的时效性来确定

询问/应答方式(Challenge/Response)

- A期望从B获得一个消息
  - 首先发给B一个随机值(challenge)
  - B收到这个值之后，对它作某种变换，并送回去
  - A收到B的response，希望包含这个随机值
- 在有的协议中，这个challenge也称为nonce
  - 可能明文传输，也可能密文传输
  - 这个条件可以是知道某个口令，也可能是其他的事情
  - 变换例子：用密钥加密，说明B知道这个密钥;
    简单运算，比如增一，说明B知道这个随机值
- 询问/应答方法不适应非连接性的应用，因为它要求在传输开始之前先有握手的额外开销，这就抵消了无连接通信的主要特点。

常见攻击和对策(三)
抵赖与非否认协议

- 需求
  - 如果通讯中的其中一方否认发送或接收过某个数据怎么办？
  - 不可抵赖性
- 要求
  - 发方非否认、收方非否认。
  - 公平性：如服务在任何一步终止，收、发双方都不能得到额外的利益
- 可信第三方
  - 仲裁
  - 发方非否认证据：EOO，给收方提供的不可抵赖证据
  - 收方非否认证据：EOR，给发方提供的不可抵赖证据

- A与B通讯，TTP做为可信第三方。

1. $A \rightarrow TTP$ ： A， B， N_a
2. $TTP \rightarrow A$ ： $\{N_a, N_{ttp1}, K_{ab}\}K_a$
3. $A \rightarrow B$： $\{M\}K_{ab}, EOO=\{f_{EOO}, N_{ttp1}, \{M\}K_{ab}\}\ K_a$
4. $B \rightarrow TTP$： EOO， $EOR=\{f_{EOR}, N_b, \{M\}K_{ab}\}\ K_b$
5. $TTP \rightarrow B$： $\{N_b, K_{ab}\}K_b$
6. $TTP \rightarrow A$： $Na, EOR=\{EOR标志, N_b, \{M\}K_{ab}\}\ K_b$

– 其中：f_{EOO}为EOO标志，f_{EOR}为EOR标志

- 安全电子商务中的可信第三方，三类TTP协议
  - Inline TTP：通讯双方间不直接交换任何消息，所有消息均经过TTP中转
  - Online TTP：需要TTP提供可信第三方服务时，TTP介入
  - Offline TTP：只有必须提供第三方服务时，TTP才介入

Offline TTP协议： Asokan-Shoup-Waidner协议

- A → B：N_a,EOO
  - IF B gives up ,Then quit.
- B → A：N_b,EOR
  - IF A gives up ,Then Abort
- A → B：m, N_a
  - IF B gives up ,Then Resolve_B
- B → A：N_b
  - IF A gives up ,Then Resolve_A

最终验证时 **Na,EOO**
Nb,EOR
TTP确认EOR，Na

- Abort：
  - A → TTP：Abort，EOO
    - IF B has Resolve_B, Then Resolve_A
    - Else TTP → A: abort_token
- Resolve_B：
  - B → TTP：EOO,EOR, N_b,未收到m
    - IF A has abort, then
    - TTP → B: abort token
    - Else TTP → B：m, N_a
- Resolve_A：
  - A → TTP： EOO,EOR, N_a,未收到N_b
    - IF B has abort, then TTP → A: abort token
    - Else TTP → A: TTP确认EOR

总结：各种威胁及防范方案

- 1、身份认证：一个主体（人、进程、计算机）验证另一个实体是它自己声称的那个人或事物。
  - 认证需要证书形式的证据。
  - 证据：口令、私钥、指纹
  - 认证模式
    - 基本认证、摘要认证、基本表单的认证、passport认证、windows认证、NTLM认证、Kerberos v5认证、X.509认证、IPSec认证（前三种既认证客户端又认证服务器）、RADIUS

- 基本认证：是一个简单的认证协议，是作为HTTP 协议的一部分定义的。
  - 在WEB服务器和浏览器都支持这个协议
  - 不安全，口令没有保护
  - 用户名和口令是64位编码的，容易解码
  - 对安全性要求高时，应使用SSL或IPSec保护连接

- 摘要认证：比基本认证安全，口令加密传输。可在IMAP、POP3、SMTP中考虑摘要认证。
- 基于表单的认证：在web页面显示用户时输入用户名和口令，然后单击提交或登陆按钮
  - 例如：ASP代码从表单中读取用户名和口令，然后作为认证数据

- Microsoft Passport：微软提供的集中式认证方式，用于多种服务,如Hotmail、MSN等。要在Web服务中包含Passport需要使用Passport SDK，在http://www.passport.com
- Windows认证：NTLM和Kerberos
  - NTLM：被多种windows服务应用，包括文件和打印服务，IIS，MS SQL,Exchange包括版本1和版本2。NTLM只向服务器认证客户机，它不为客户机校验服务器的真实性
  - Kerberos V5：由MIT设计的，Win2K以后版本都实现该协议，客户机和服务器都被校验，比NTLM更安全，速度更快

- X.509：现在最实际的用法是SSL，当用SSL连接Web服务器或邮件服务器时，应用程序会校验服务器的真实性。
  - 是通过将服务器的证书中的普通名字与应用程序正在连接的主机名相比较完成的，若两个名称不同，应用程序会发警告。
- IPSec：只认证服务器，不能认证用户。
  - 认证服务器，还提供数据完整性和保密性
- Radius：许多服务产品，包括Microsoft Internet认证服务(Internet Authentication Service,IAS)都支持Radius，它实际上是远程用户认证的标准协议，Win2K下的认证数据库是Active Directory

- 2、授权
  - ACL：由一系列ACE，ACE决定一个主体能够对一个资源做什么。如Blake对某个对象有读权限
  - 特权：是赋予用户的权利，用户可以执行系统范围内的操作，如调试程序、备份、远程关机
  - IP限制：是IIS的一项功能，限制Web站点的一部分，如一个目录，也可限制整个站点，使得只有专门IP、DNS、子网访问
  - 服务专用权限：服务的访问控制形式，保护自己专门的对象类型。如MS Server提供权限，管理员用它可以决定谁有权访问哪一个表、存储过程和视图。例COM+应有程序支持角色，可以为一组组件定义用户。

- 3、防篡改、增强保密性和可控性
  - SSL:在服务器和客户机之间传输的数据加密，并用MAC消息认证码提供数据完整性
  - IPSec：支持认证、数据保密需要的加密、数据完整性需要的MAC
  - DCOM和远程调用：支持认证、保密和完整性。对性能影响小
  - EFS加密文件系统：包含在Win2K以后版本中，是基于文件的加密方法，对文件加密并检查篡改

- 保护秘密或最好不要保存秘密
- 不要在第一个出现的地方存储秘密信息，最好是记忆
- 机密性confidentiality:通过加密完成
  - 散列：将数据传递给加密函数，生成一个小的值，该值唯一的标识数据，128位或160位，不提供数据信息
  - MAC：消息数据和秘密数据一起进行散列计算，只有受信任方知道这个数据
  - 数字签名：类似于MAC，数据被混编，用只有发送者知道的私有密钥来加密散列
- Windows提供了CryptoAPI，用户可以在应用程序中添加加密支持，包括加密、散列、MAC和数字签名

- 审核auditing：收集成功的和失败对象的访问、特权的使用、以及其他重要的安全活动信息，并存储在硬盘上，以便日后分析。Windows、IIS Web、SQL Server都提供
- 过滤filter：对接受到的数据进行检查
- 节流Throtting：限制对系统请求的数量，如只允许少量的匿名请求，同时允许更多的被认证的请求
- 服务质量：允许为专门的传输类型提供优先处理，如允许优先处理多媒体

- 访问或修改秘密的HTTP数据（T&I）
  - 使用SSL或IPSec；
  - 需要设置http服务器使用私有密钥和证书，配置ipsec对建立连接性能影响大
- 访问或修改秘密的RPC和DCOM （T&I）
  - 使用一致性和保密选项
  - 要求修改代码，对性能影响小
- 读或修改基于电子邮件的通信（T&I）
  - 使用PGP或S/MIME
  - PGP不容易使用， S/MIME很难配置
- 可能会丢失一台包含了秘密数据的设备（I）
  - 在设备上使用PIN个人身份码,有人多次尝试后锁定。不要忘记PIN。

- 服务建立太多的连接（D）
  - 提供基于IP地址的节流，要求认证
  - 通过代理的IP地址检查不能正确工作，需要给用户提供帐户和口令
- 攻击者试图猜口令（S,I,E）
  - 给每个无效口令增加延时，尝试多次后锁定，支持强口令
  - 攻击者可能会通过猜测的方法创建一个DDoS攻击，强制此账号锁定，使得有效的用户无法使用自己的账号
  - 改变锁定账号的时间设置为较短的时间，需要添加代码加强口令的强度
- 读私密的cookie （ I ）
  - 在服务器端加密cookie
  - 需要给web站点添加加密代码
- 篡改cookie（T）
  - 在服务器端对cookie作MAC或签名
  - 需要给web站点添加MAC或签名代码

- 访问私有的秘密数据（I）
  - 隐藏并妥善存储数据
  - 使用好的访问控制
- 攻击者欺骗服务器（S）
  - 使用支持服务器认证的认证
- 攻击者给你的站点发送html或脚本（D）
  - 使用正则表达式限制发送内容
  - 需要定义表达式并判断有效输入
- 攻击者打开多个连接，但什么也不做（D）
  - 管理连接不要超时
- 未被认证的连接将消耗内存（D）
  - 要求认证，不要给未知连接分配大量资源

- 数据重放（T,R,I,D）
  - 使用SSL等加密，包计数或超时，设置时间戳
  - 需要技巧
- 攻击者给你的进程附加调试器（T,D,I）
  - 限制使用SetDebugPrivilege特权帐户
- 在一个共享的工作站环境中，攻击者访问或使用被前一个用户缓存的数据（T,I）
  - 不要缓存敏感的数据
- 攻击者关闭你的进程（D）
  - 要求本地管理员组的成员才能关闭进程
- 攻击者修改配置数据（S,T,R,I,D,E）
  - 认证访问数据的所有连接，加强数据上的ACL，支持数据签名
  - 耗时难实现
- 恶意的用户访问或篡改web服务器上的查询数据（T&I）
  - 使用基于文件的加密，确保加密密钥安全

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

软件安全开发

- 3.1 软件安全的指导原则
- 3.2 软件安全需求
- 3.3 软件安全编码基本方法与技术
- 3.4 软件安全设计

面向网络攻击的软件安全设计

- 可用性安全
- 机密性安全
- 完整性安全
- 可控性安全
- 可审性安全

匿名通信技术

- 基本概念
- 经典匿名协议
- 经典匿名网络

Every man should know that his conversations, his correspondence, and his personal life are private.

-Lyndon B. Johnson, President of the United States, 1963-69

基本概念

- 匿名性（Anonymity）：在一定范围内无法确认的状态
  - 没有绝对的匿名
  - 不可关联性（Unlinkability）
    - 操作与执行该操作的主体的身份无关联关系
    - 比如：消息发送者和消息本身在系统中无法关联起来
  - 不可观测性（Unobservability）
    - 无法区分感兴趣的东西和其他东西

“Anonymity is bullshit”

基本概念

- 匿名（Anonymity）：
  - 隐藏发送者、接收者的身份信息，或者无法关联消息的发送者和接收者。
- 匿名分类：
  - 发送者匿名（Sender Anonymity）
  - 接收者匿名（Receiver Anonymity）
  - 发送者和接收者的无关联性（Unlinkability of Sender and Receiver)

匿名技术应用

- 隐私
  - 网上交易、WEB浏览等
- 数字现金（Digital cash）
  - 电子货币
- 匿名投票（Anonymous electronic voting）
- 不可审计性出版（Censorship-resistant publishing）
- 不可追踪电子邮件（Untraceable electronic mail）

密码员进餐问题

- 密码员进餐问题： Dining Cryptographers
- 是一个具有强安全属性的发送者匿名方案:即便攻击者具有无穷计算能力，也不能破坏其匿名性（Guarantees information-theoretic anonymity for message senders）

David Chaum. “The dining cryptographers problem: unconditional sender and recipient untraceability.” Journal of Cryptology, 1988

密码员进餐问题

密码员进餐问题：

A，B，C希望匿名支付帐单；可能一个密码员正在付帐，也可能是由NSA(国家安全局）付帐。A、B和C尊重彼此的匿名支付的权利，但是需要知道是否是NSA付帐。

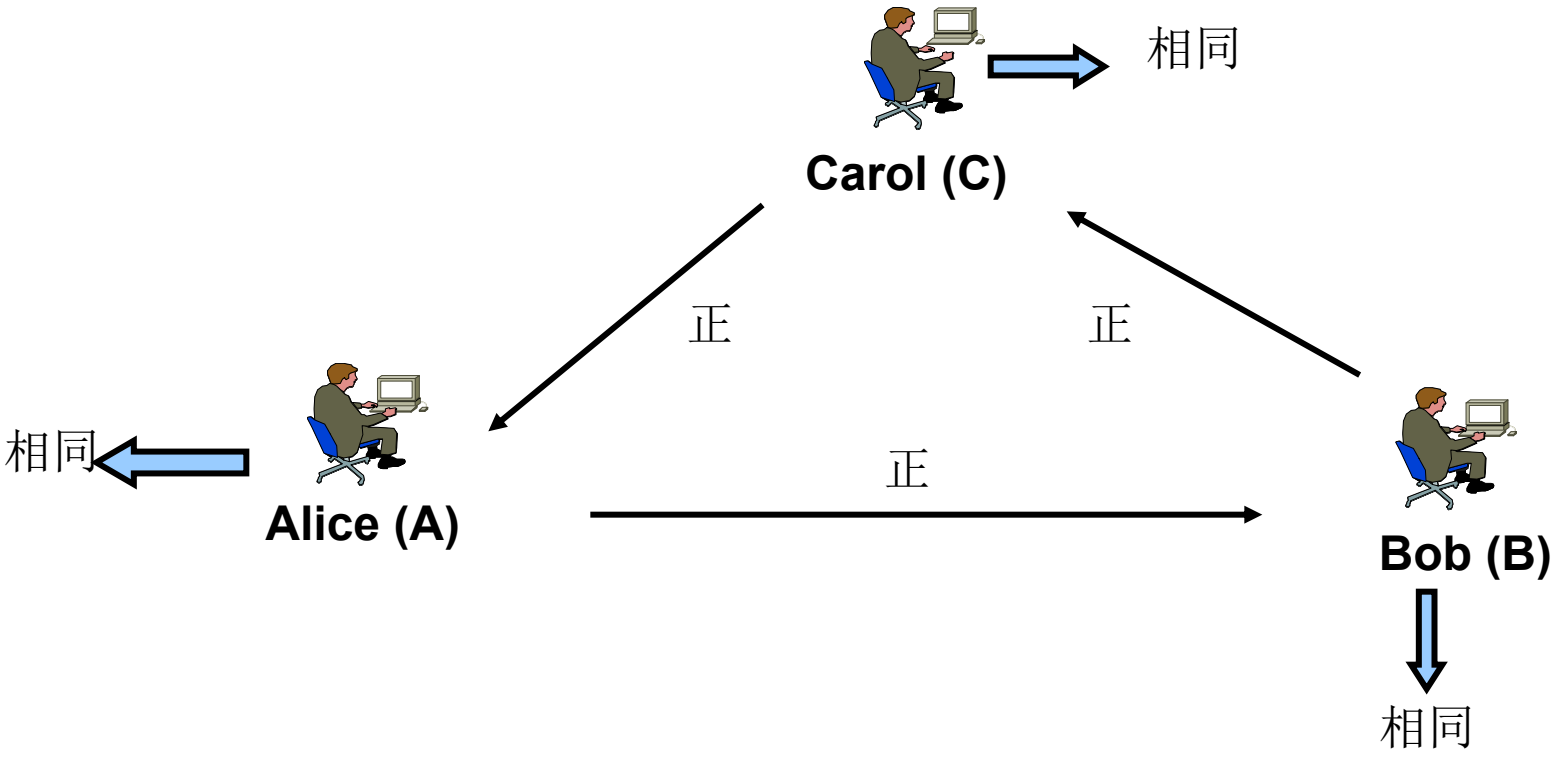
方法：每个密码员在他右边抛一个硬币（用正面和反面表示），结果只能被自己和自己右边的密码员看到。

比较左右两边的数，并报结果：相同和不相同。

规则：如果自己没有付帐，则报正确结果；如果自己正在支付，则报相反的结果

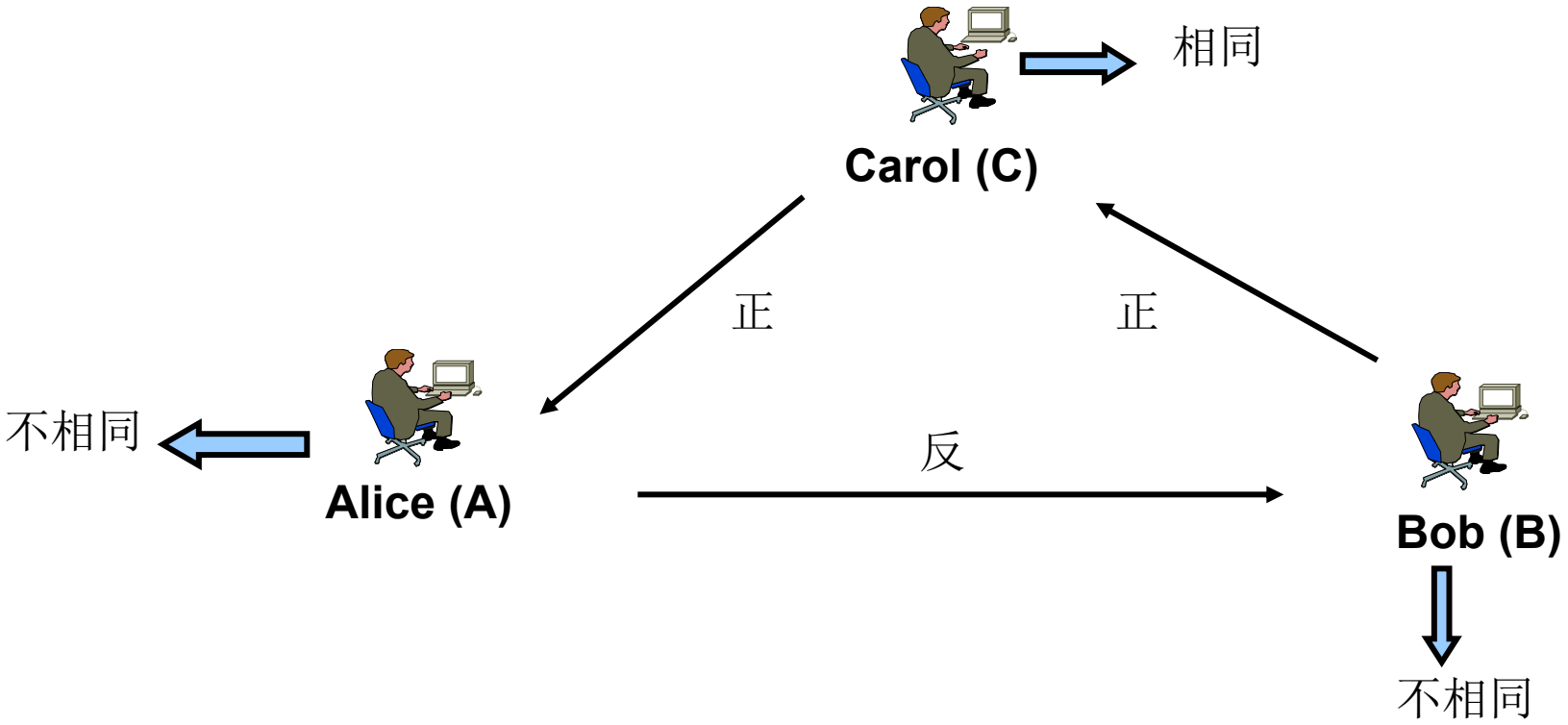
答案：如果报“不相同”的人数为奇数，则表明有一个密码员在付帐；如果为偶数，则是NSA在付帐。

密码员进餐问题



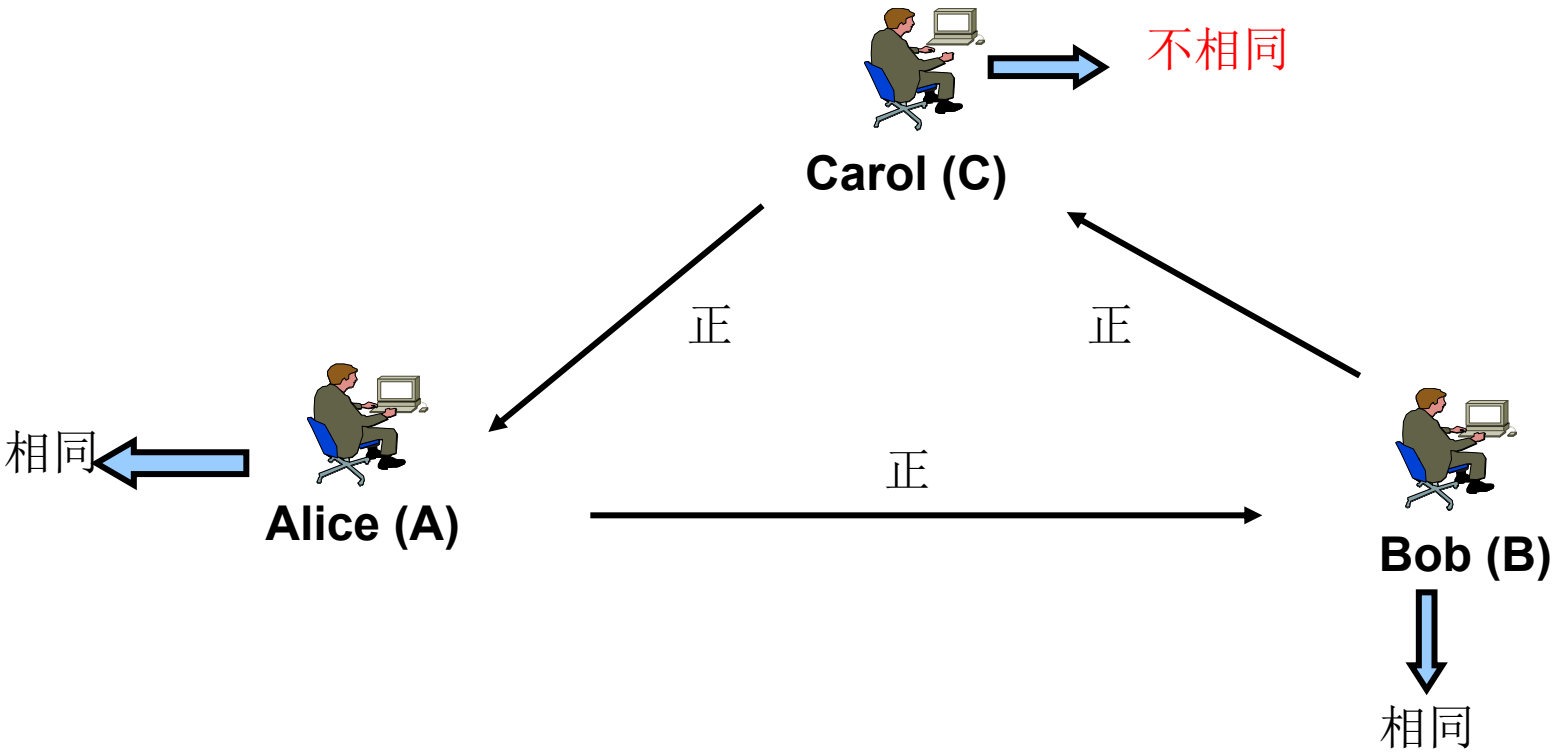
答案：报不相同的为偶数，NSA付帐。

密码员进餐问题



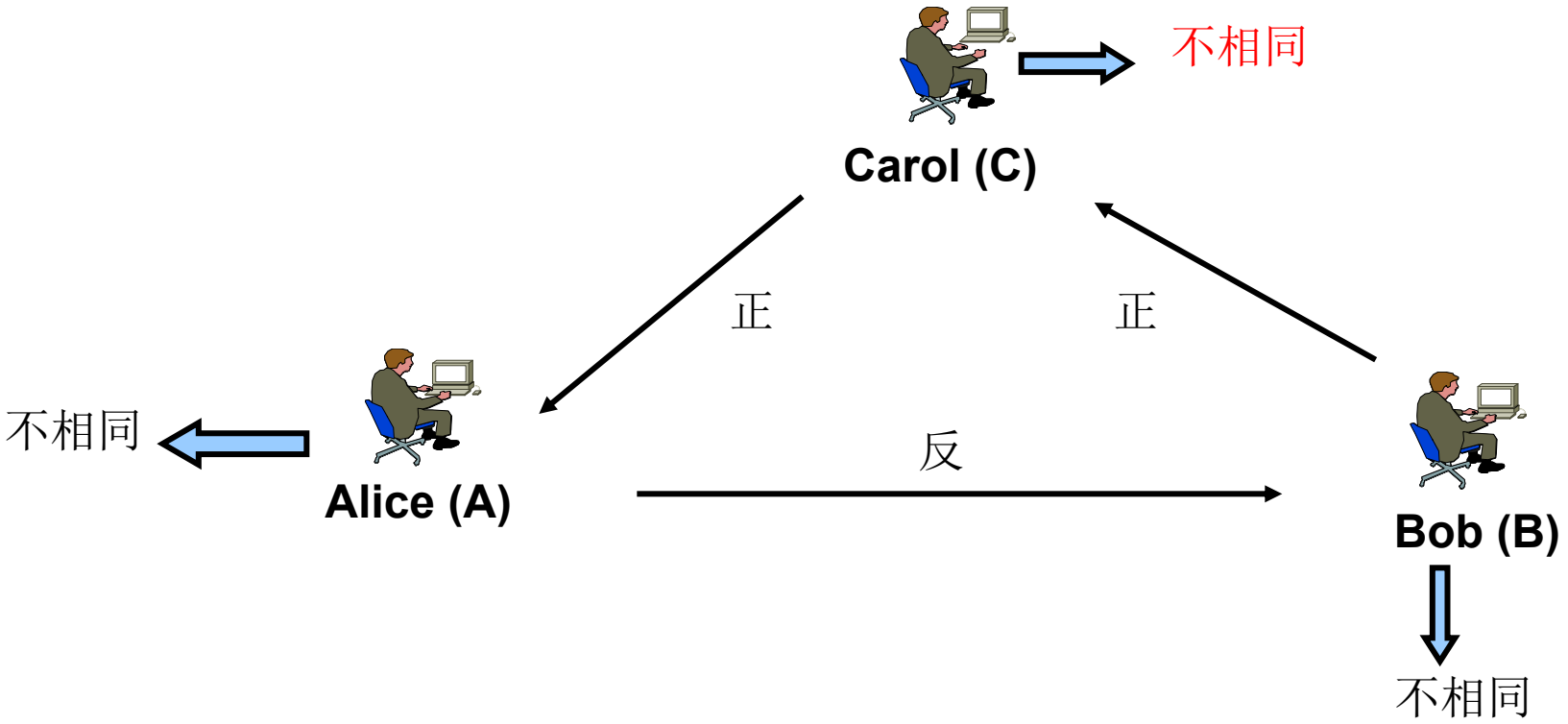
答案：报不相同的为偶数，NSA付帐。

密码员进餐问题



答案：报不相同的为奇数，一个密码员付帐。

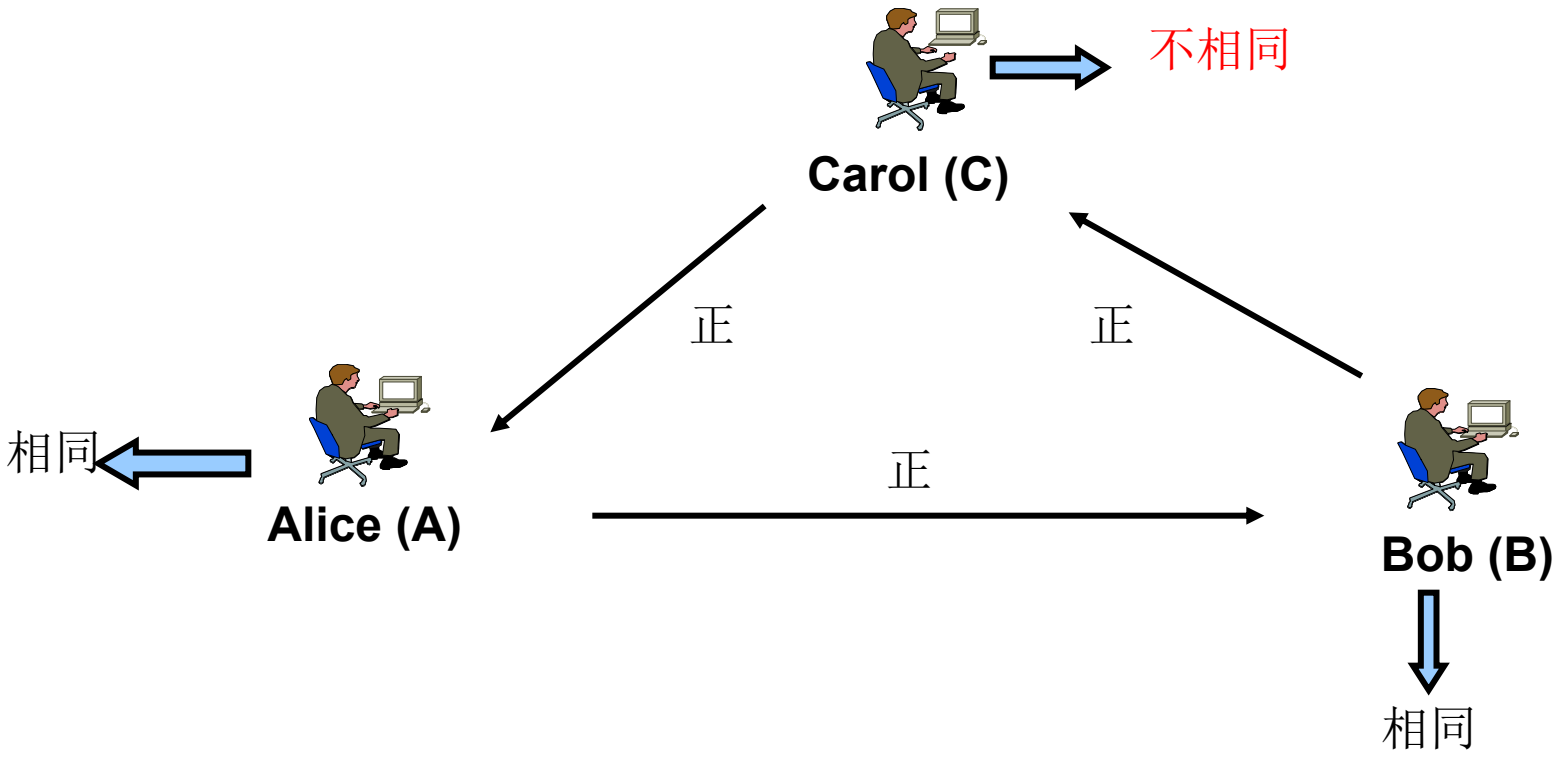
密码员进餐问题



答案：报不相同的为奇数，一个密码员付帐。

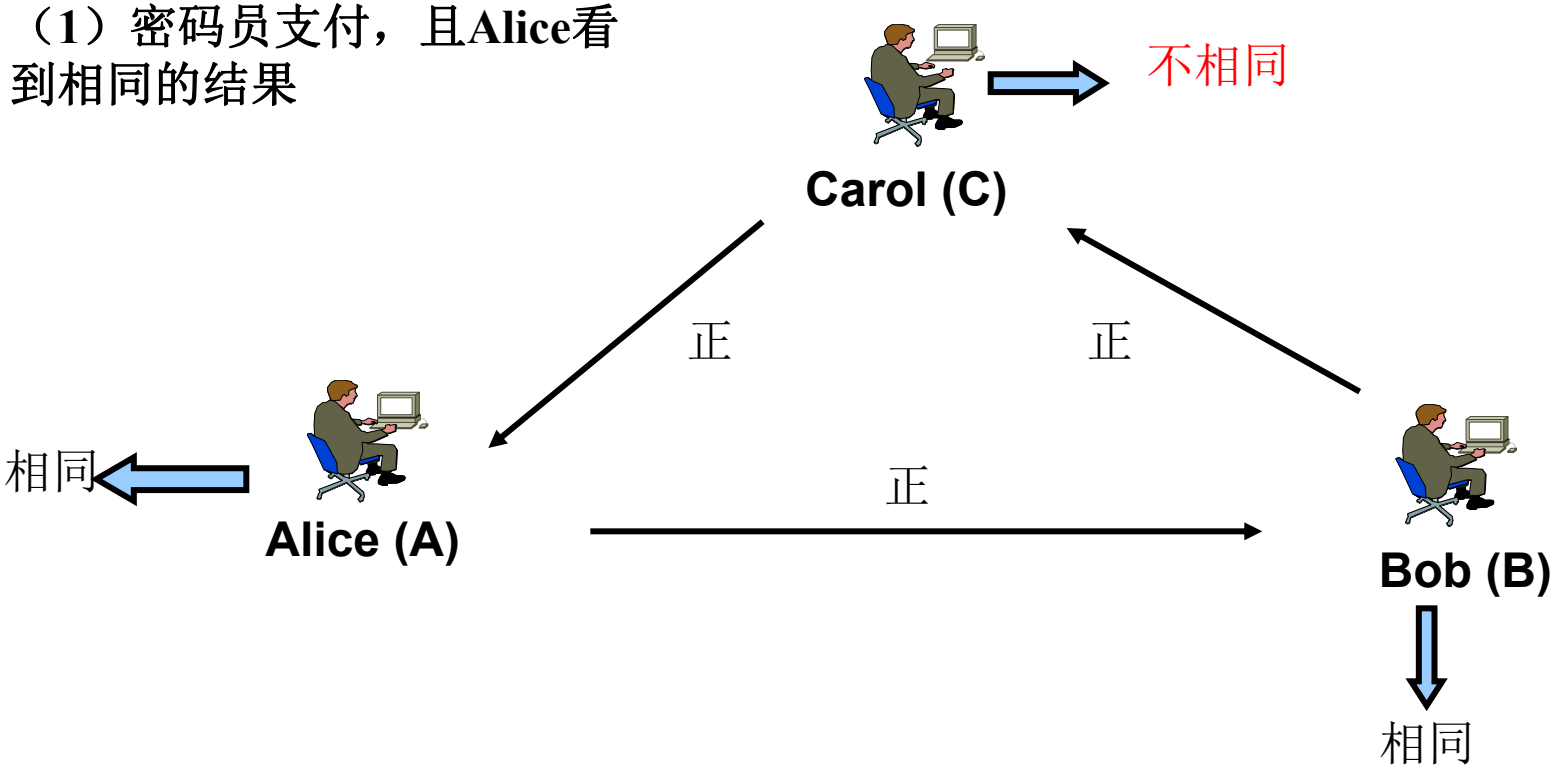
密码员进餐问题

问题：其他密码员能否知道是Carol付帐？



密码员进餐问题

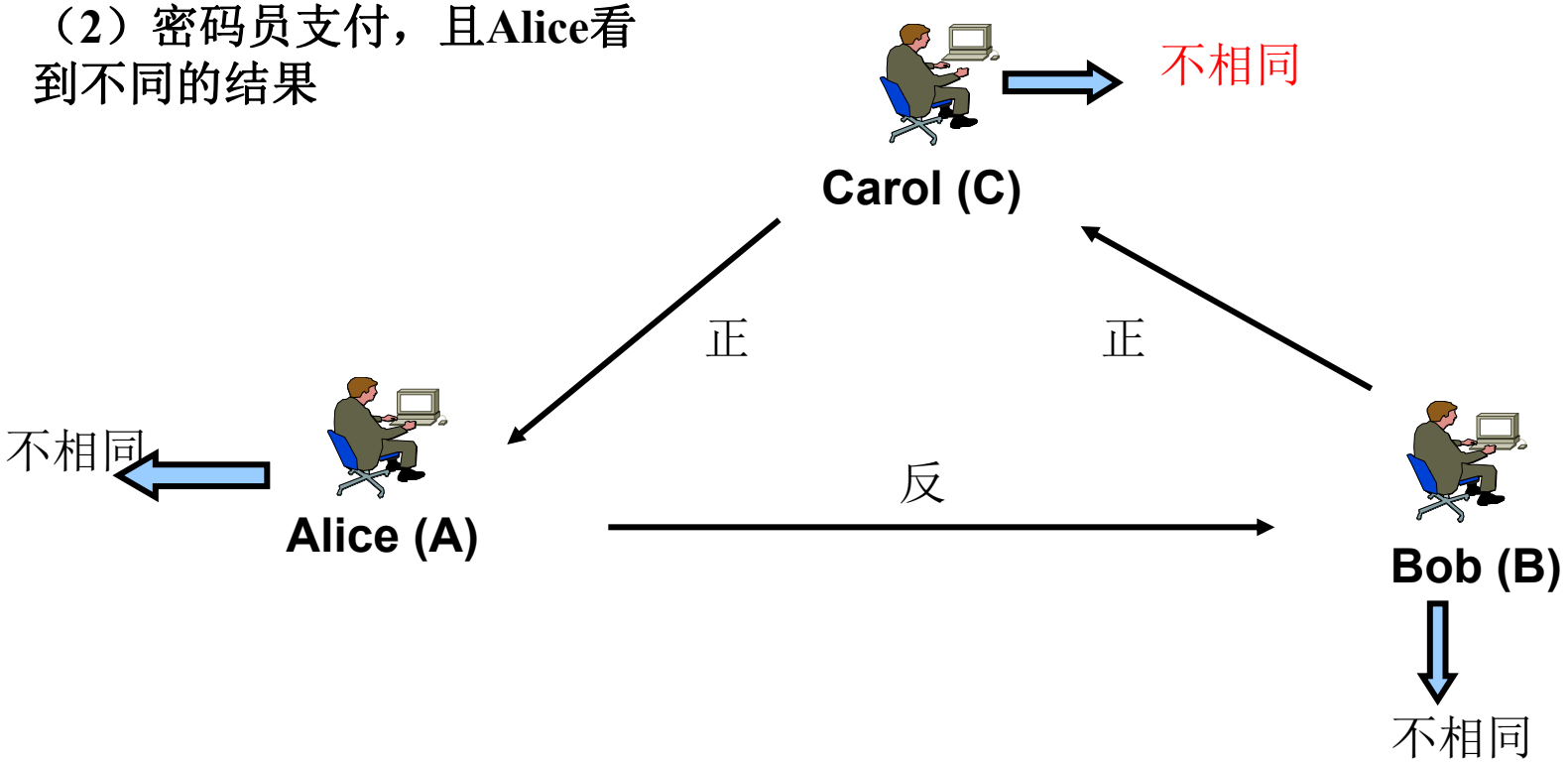
（1）密码员支付，且Alice看到相同的结果



要么Bob说相同，Carol说不同；要么Bob说不同，Carol说相同。
说不同的是支付者。但是由于概率相同，Alice无法区分。

密码员进餐问题

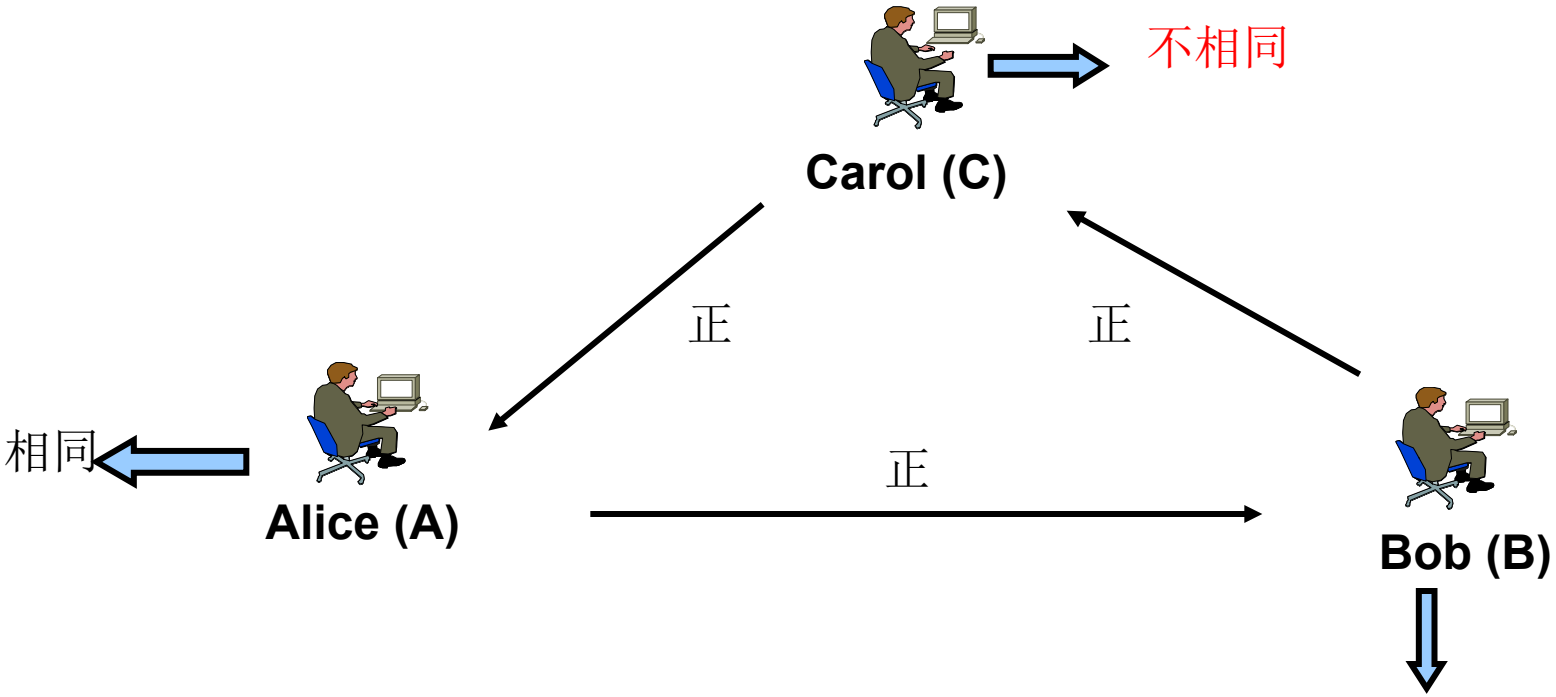
（2）密码员支付，且Alice看到不同的结果



1、要么Bob和Carol说不同（Carol付账）。但是由于概率相同，Alice无法区分。

2、要么Bob和Carol说相同（Bob付账）。但是由于概率相同，Alice无法区分。

密码员进餐问题



问题：其他密码员怎样才能知道是Carol付帐？

答案：比如Alice，如果要想知道是Carol付帐，则需要知道Carol和Bob之间的抛币情况。

密码员进餐问题

- 问题：如何通用化上述协议，并用来匿名传送消息？
- 方法：
  - 参与通信的用户排成一个逻辑圆
  - 在一定时间间隔内，相邻用户之间抛币（必须防止窃听）
  - 每次抛币后用户都说相同或者不同

密码员进餐问题

- 比如Alice想匿名发送消息M
- 过程：
  - 将M二进制化：1001
  - 对于1，Alice采用颠倒说法；对于0，Alice采用正常说法：
    - 比如Alice看到的是“不同”、“相同”、“相同”、“相同”，则Alice说“相同”、“相同”、“相同”、“不同”。
  - 如果Alice发现协议的结果与自己所期望的不同，则表示其他人也在发送消息，因此她等待一段时间后再发送。

密码员进餐问题

Alice要发送1001

相同

Carol (C)

正

正

不相同

Alice (A)

正

Bob (B)

相同

第一轮：Alice发送1

由于报不同的人数为奇数，因此Bob和Carol知道有人说1

密码员进餐问题

Alice要发送1001

相同

Carol (C)

正

正

相同

正

Alice (A)

Bob (B)

相同

第二轮：Alice发送0

由于报不同的人数为偶数，因此Bob和Carol知道有人说0

密码员进餐问题

Alice要发送1001

相同

Carol (C)

正　　正

相同

Alice (A)

正

Bob (B)

相同

第三轮：Alice发送0

由于报不同的人数为偶数，因此Bob和Carol知道有人说0

密码员进餐问题

Alice要发送1001

相同

Carol (C)

正

正

不相同

Alice (A)

正

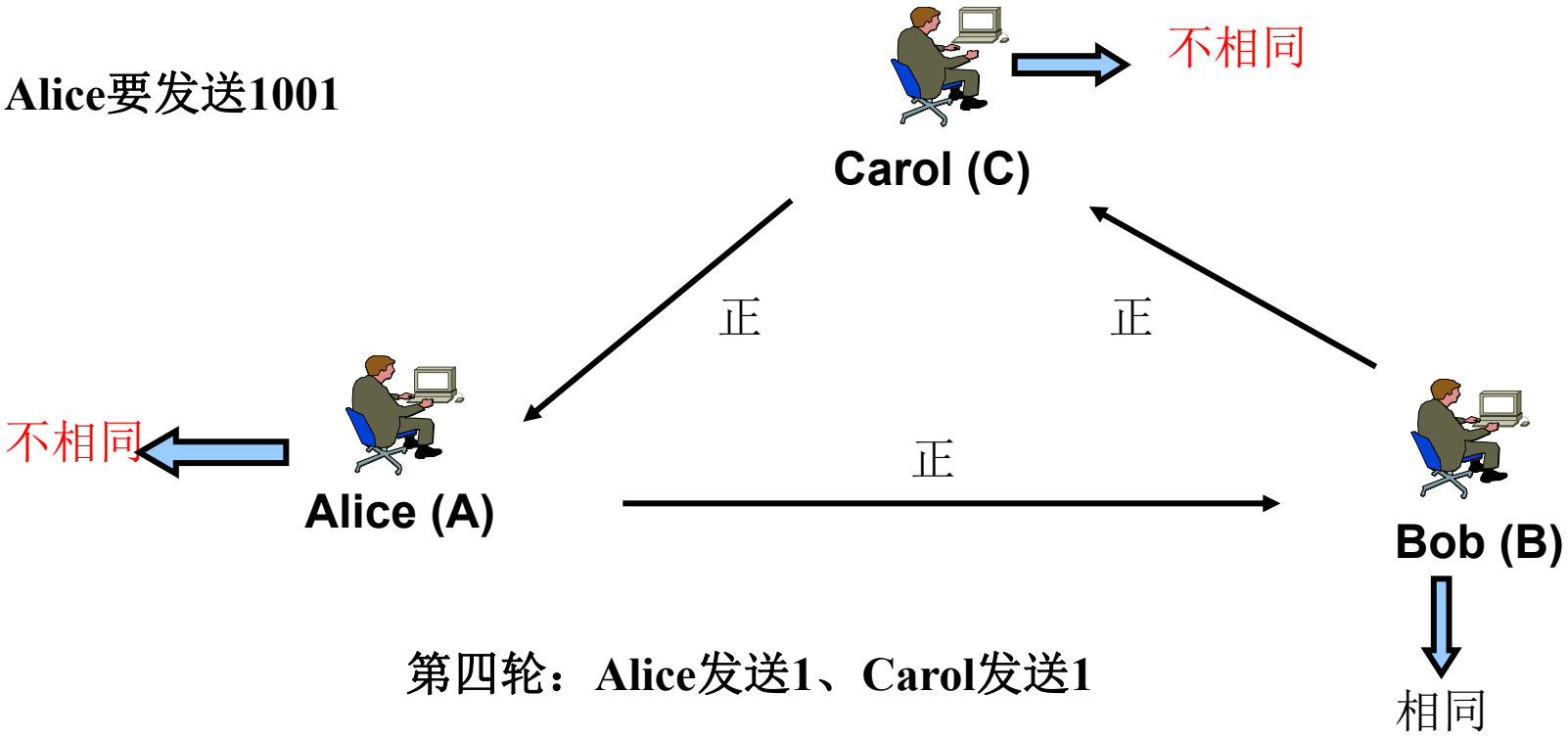
Bob (B)

相同

第四轮：Alice发送1

由于报不同的人数为奇数，因此Bob和Carol知道有人说1

密码员进餐问题



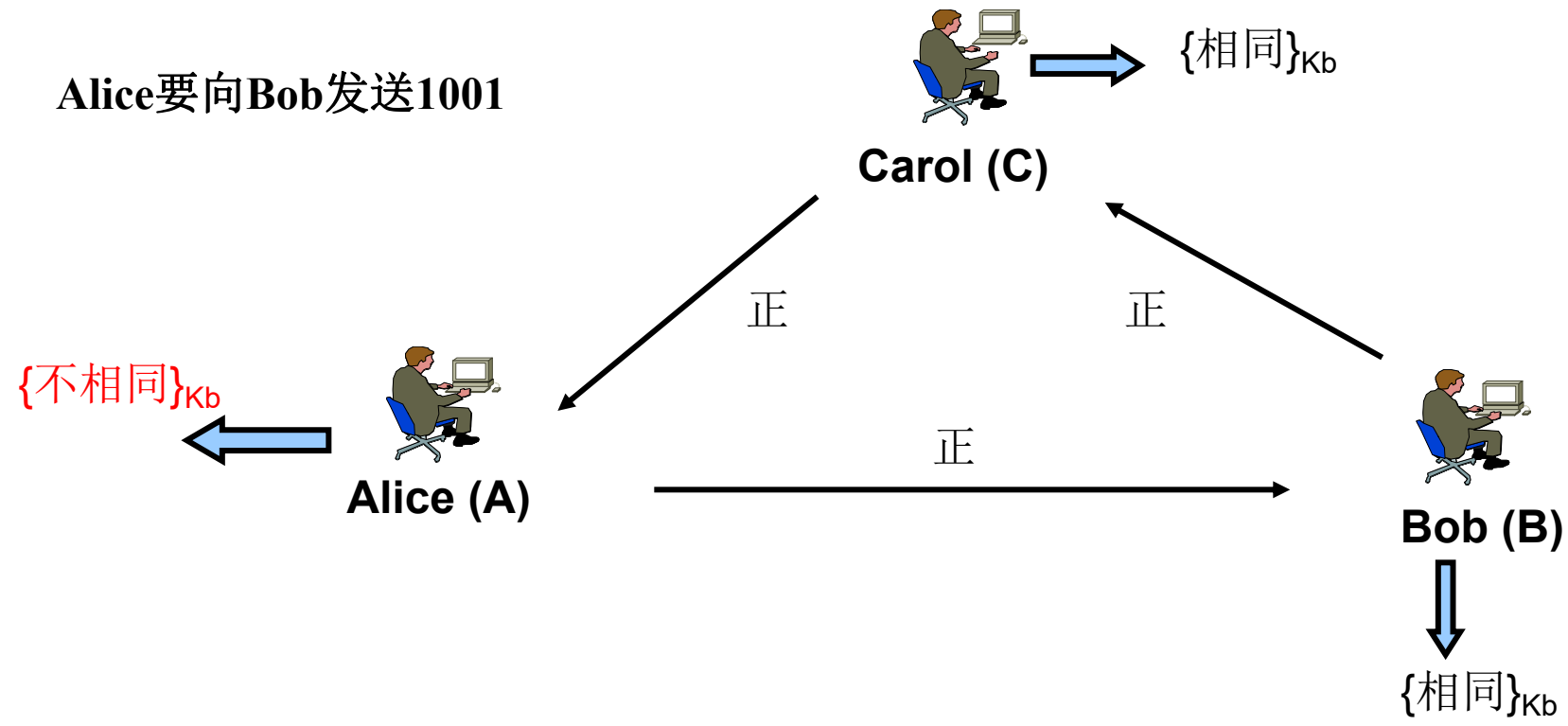
第四轮：Alice发送1、Carol发送1

由于报不同的人数为偶数，因此Alice、Bob和Carol知道有人说0

这与Alice要发送的消息不一致，因此Alice等待一段时间后重发

密码员进餐问题

Alice要向Bob发送1001



问题1：如何抛币？

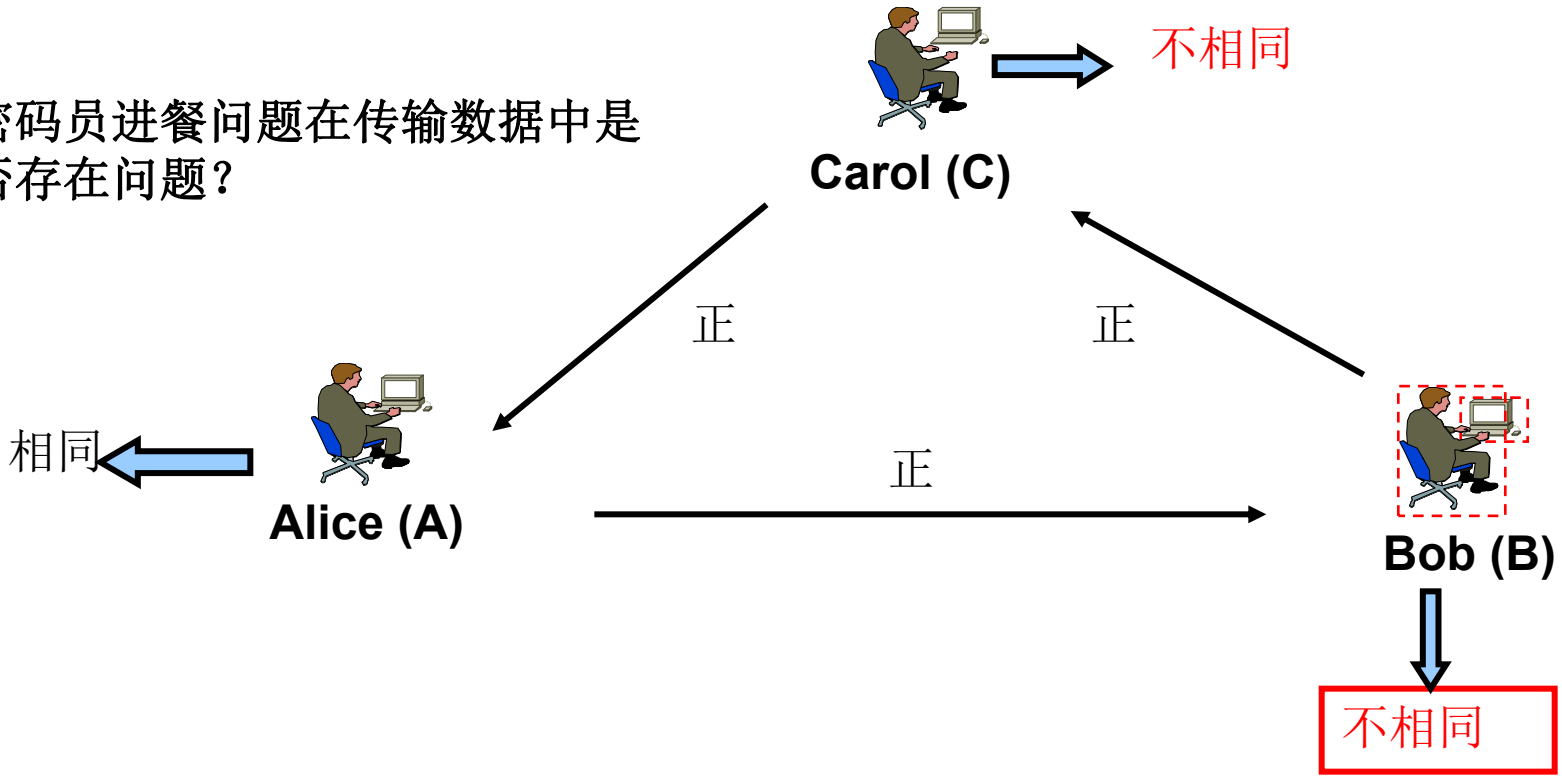
方法：公正抛币协议

问题2：如何将消息发送给指定的人（比如Bob）？

方法：用Bob的公钥加密结果

密码员进餐问题

密码员进餐问题在传输数据中是否存在问题？



Bob说谎，导致结果为NSA付帐，实际上是Carol在付帐。假设Carol传输1给Alice，Alice收到数据错误。

问题：能否检测这种恶意破坏？

密码员进餐问题

- 更一般的推广
  - 对每一位将要发送的消息，每个用户产生一个随机位，并发送给一个邻居节点
    - 任一节点i知道两位信息：自己选择的随机位b_i和一个邻居节点j的随机位b_j
  - 每一个节点（除发送节点外）通告自己所知道的的两个位的异或值结果：$C_i = b_i$ XOR b_j
  - 发送者x通告以下结果：$C_x = b_x$ XOR b_k XOR M_x
  - 所有通告结果的异或值就是消息位

$$M_x = C_1 \oplus C_2 \oplus \ldots \oplus C_n$$

原因：每一位出现两次，在异或操作中被抵消，最后只剩下b_1、b_n和M

密码员进餐问题

- 上述协议的优点：
  - 匿名性很健壮：即便部分主体作恶，也不能破坏匿名性
  - 每一个协议主体均很难通过分析协议过程中的知识来破坏匿名性
- 上述协议的缺点：
  - 成员间需要一个需要一个成对的信道（否则随机位信息无法被共享）
  - 需要大量的通信和大量的随机位
  - 需要可靠广播信道
  - 无法抵抗恶意破坏者

密码员进餐问题

- 基本思路：
  - （1）构造一个可靠广播信道：Byzantine General Problem
  - （2）去除可靠广播信道的要求
  - （3）信道预留技术
    - 两个阶段：第一阶段预留信道，第二阶段发送消息

Chaum's MIX

- 1981年提出的Chaum's MIX是最早的匿名电子邮件通信方案，也是现代匿名通信系统的基础之一
- 该方案采用公钥＋可信的邮件转发网络

David Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". Communications of the ACM, February 1981

Chaum's MIX

- Chaum's MIX有四种情况
  - 不带返回地址的单个MIX
  - 带返回地址的单个MIX
  - 不带返回地址的多个级连MIX
  - 带返回地址的多个级连MIX

Chaum's MIX

Alice (A)

$\{R_1, \{R_0, M_1\}_{K_b}, B\}_{K_1}$

发送者的身份不会被暴露

$\{R_4, M_3\}_{K_y}, Y$

$\{R_3, \{R_2, M_2\}_{K_x}, X\}_{K_1}$

Carol (C)

K_1

MIX

$\{R_0, M_1\}_{K_b}, B$

Bob (B)

Malice (M)

$\{R_5, \{R_4, M_3\}_{K_y}, Y\}_{K_1}$

$\{R_2, M_2\}_{K_x}, X$

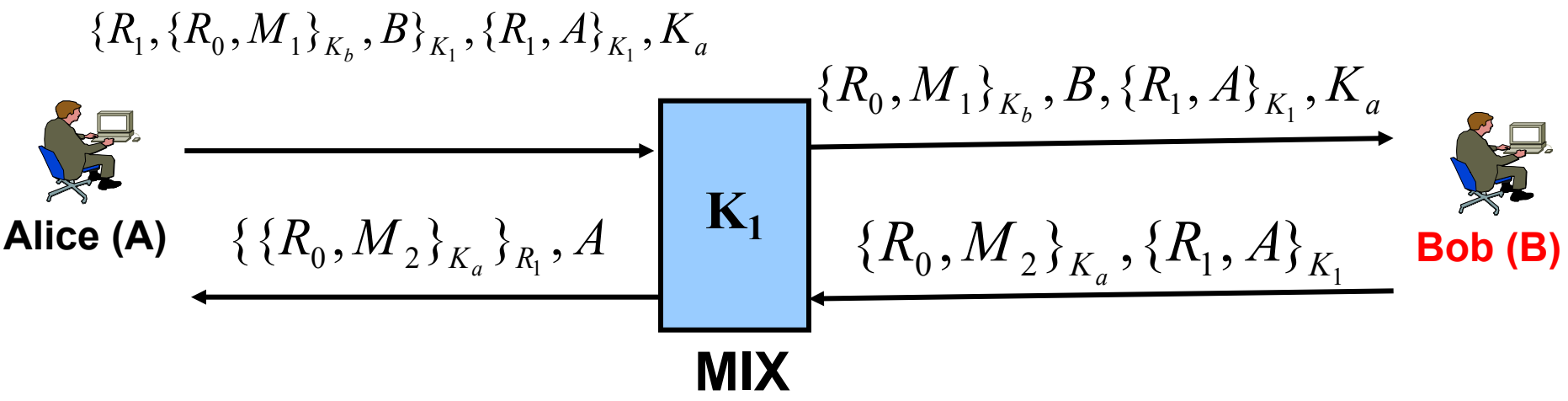
DC-MIX – 无返回地址

Chaum's MIX

不带返回地址的多个级连**MIX**：在这种模式下，消息通过多个**MIX**节点依次传递。每个节点去除一层加密，然后将消息发送到下一个节点。这提高了匿名性，因为跟踪消息变得更加困难。

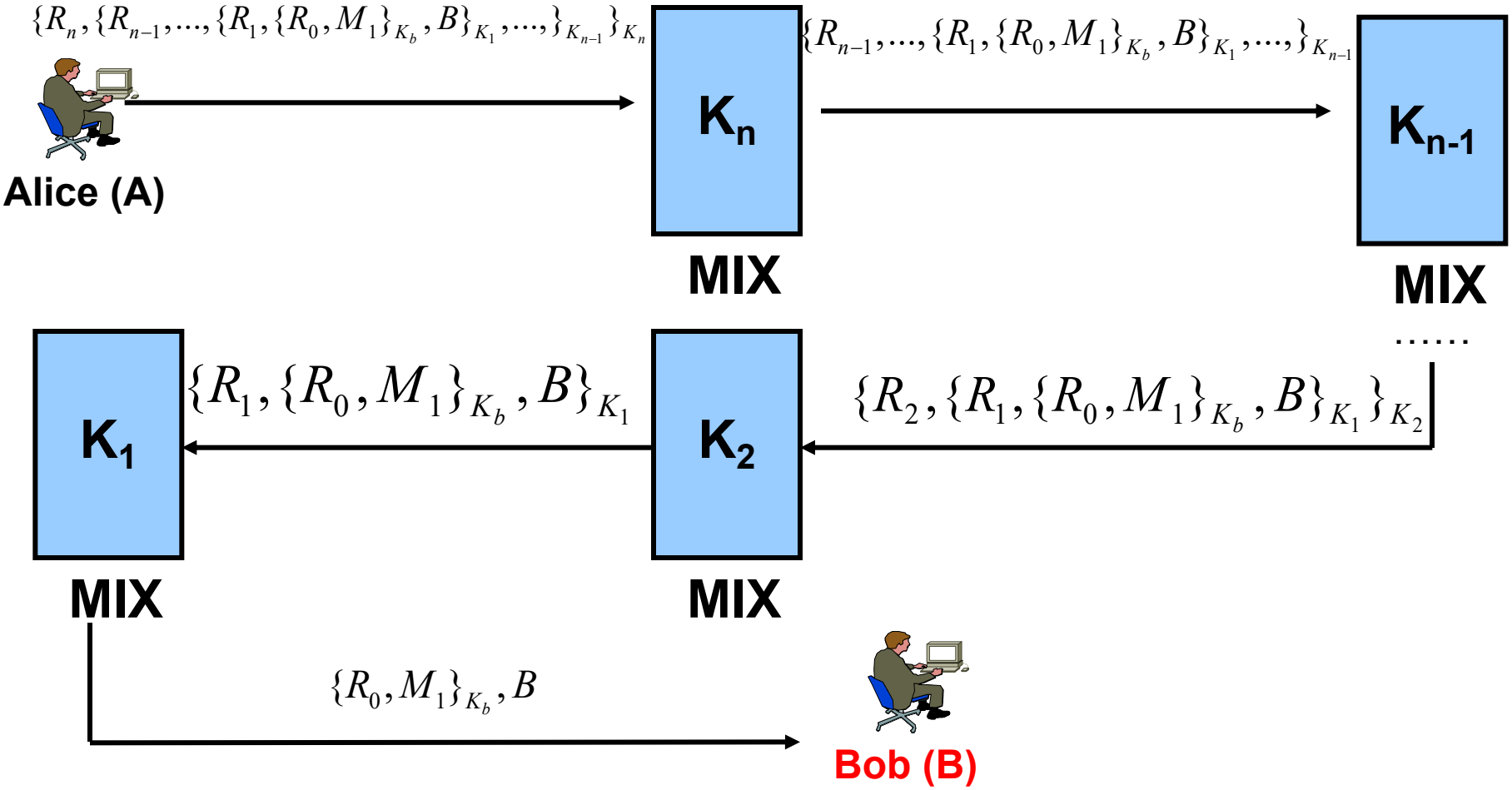
带返回地址的多个级连**MIX**：这种模式在多级连**MIX**的基础上添加了返回地址。这使得接收者可以通过同样匿名的方式回复消息，同时保持了高度的匿名性。

提供了一个方法使得**B**可以和**A**通信，具体为**MIX**同时发送了一个加密的地址



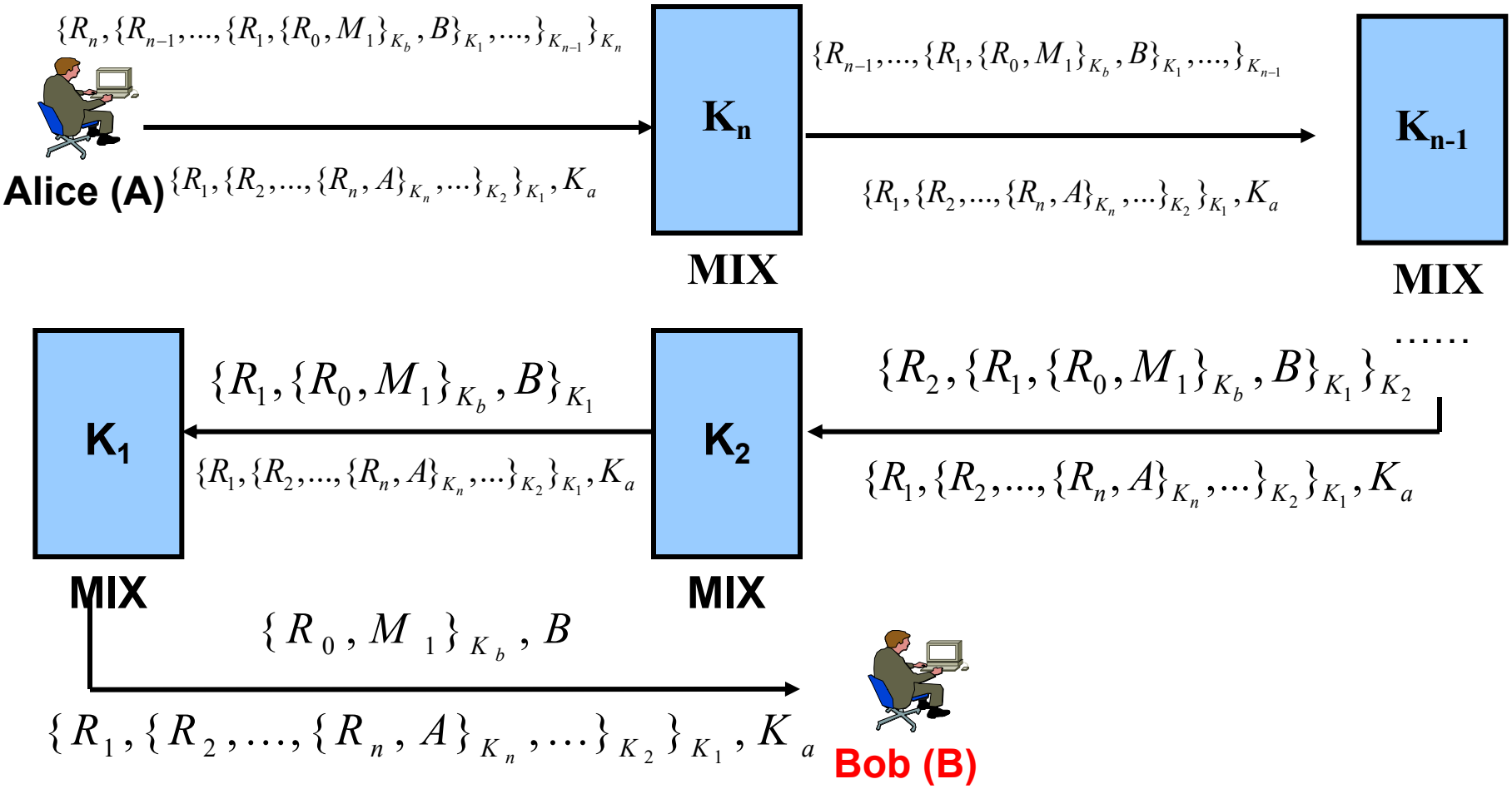
DC-MIX – 带返回地址

Chaum's MIX



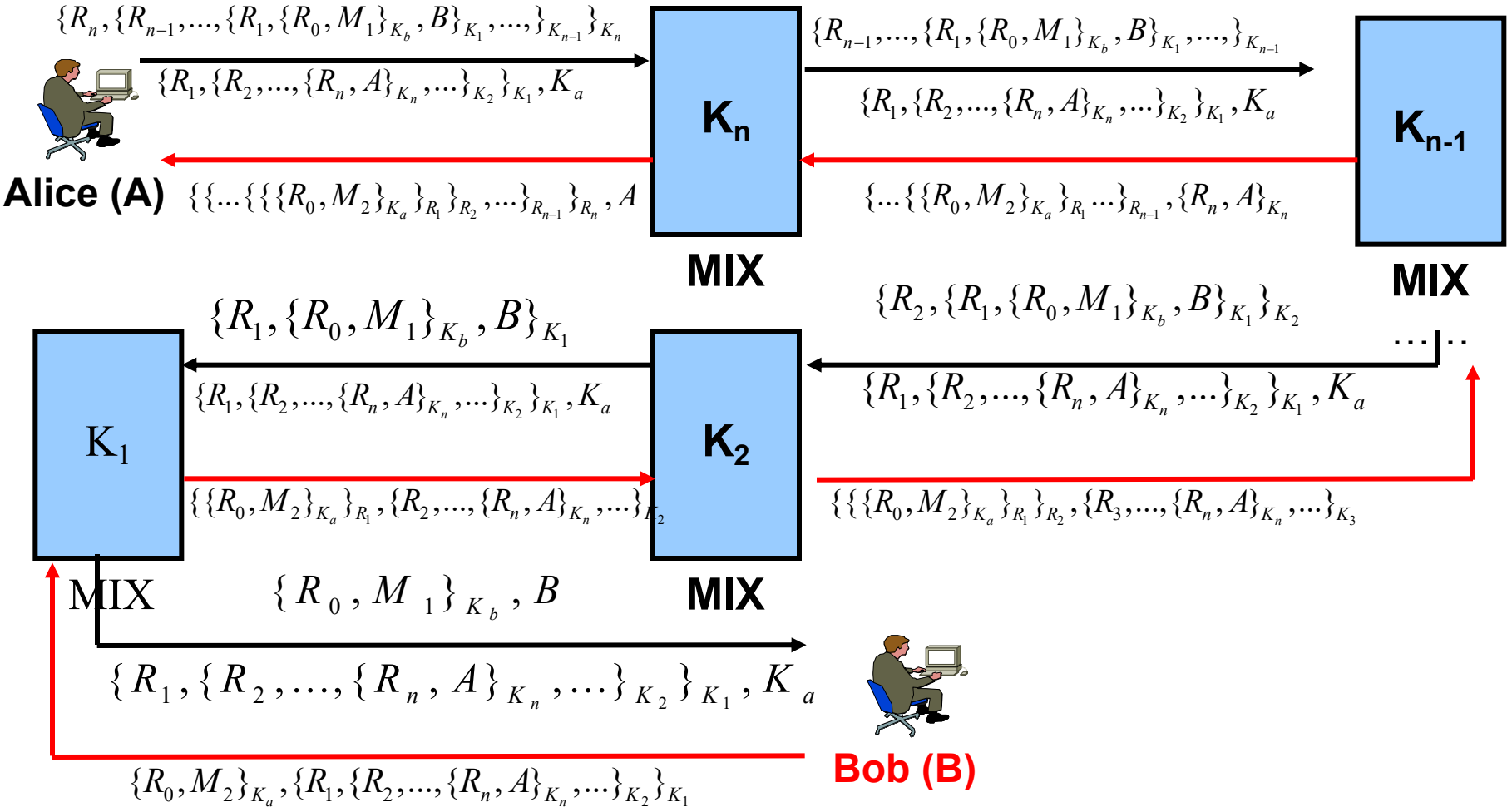
级连DC-MIX－无返回地址

Chaum's MIX



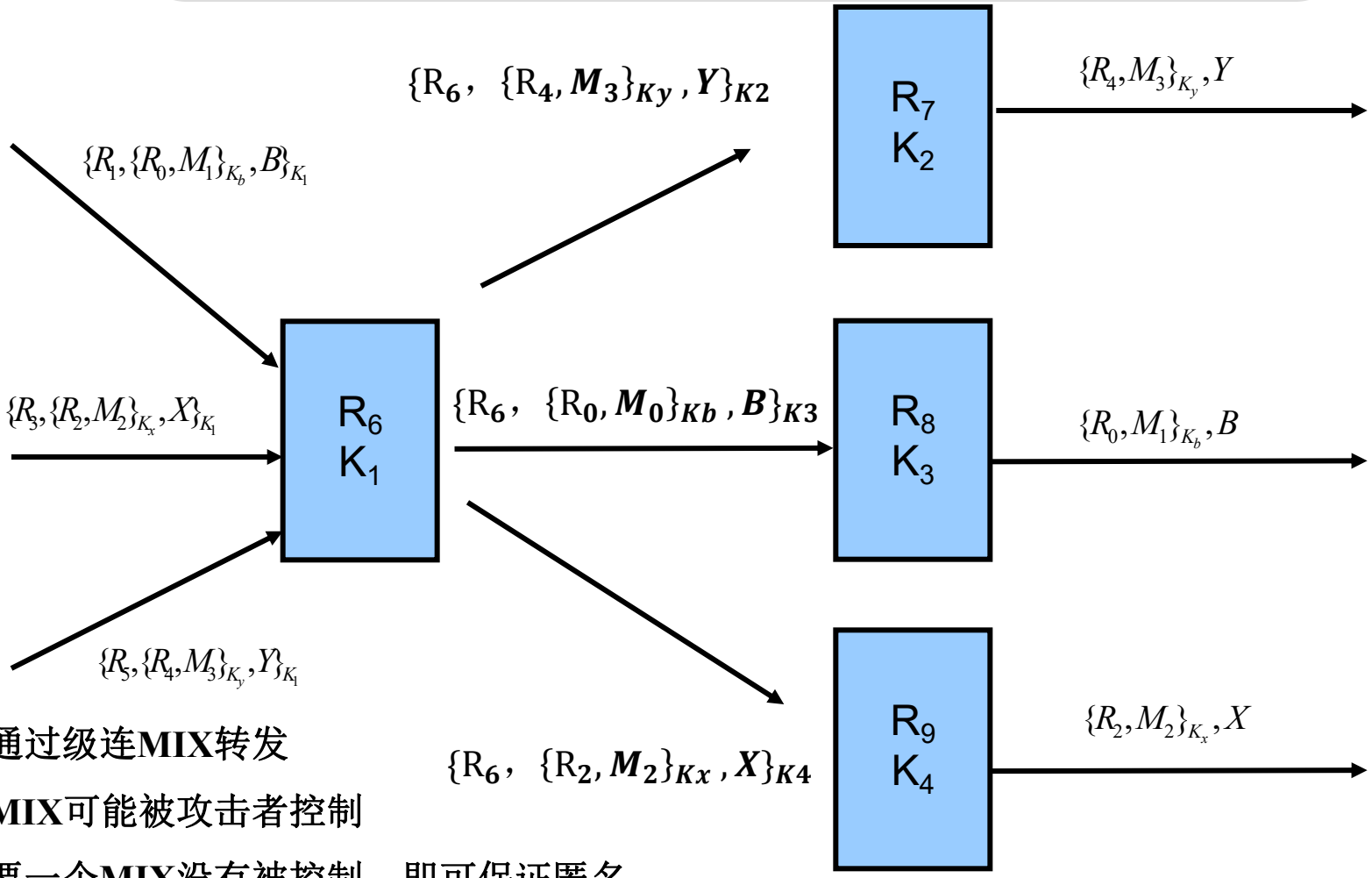
级连DC-MIX－带返回地址

Chaum's MIX



级连DC-MIX－带返回地址

Chaum's MIX



消息通过级连MIX转发

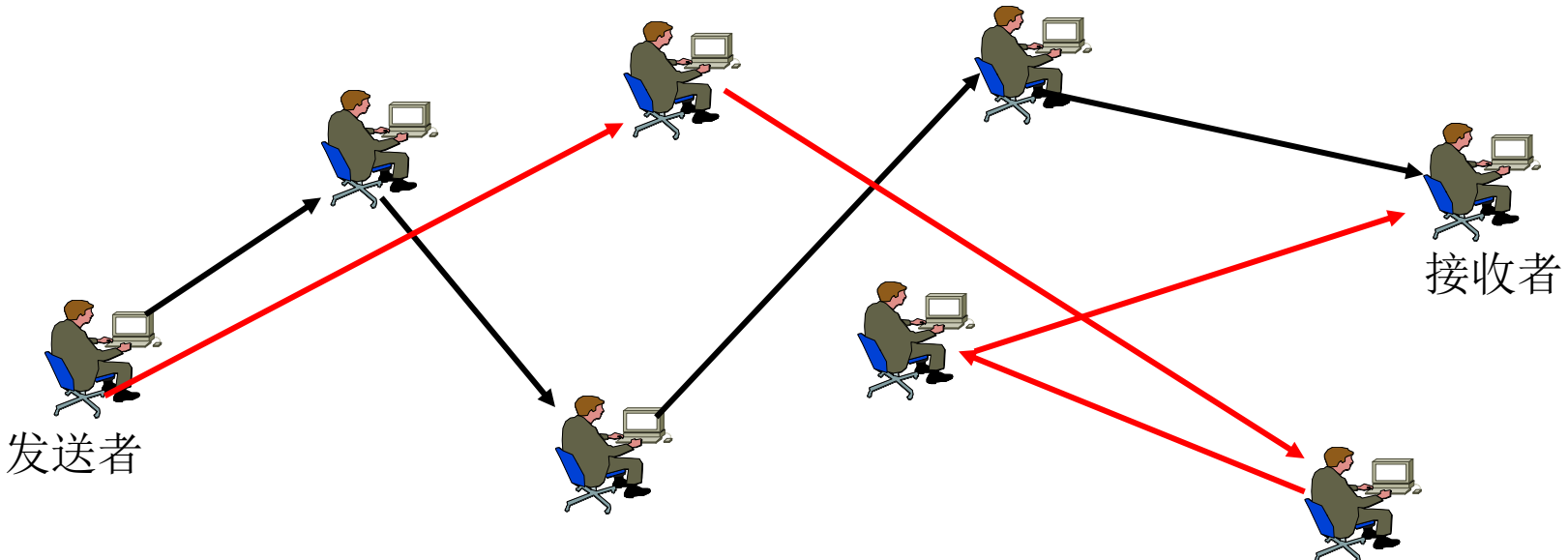
某些MIX可能被攻击者控制

只需要一个MIX没有被控制，即可保证匿名

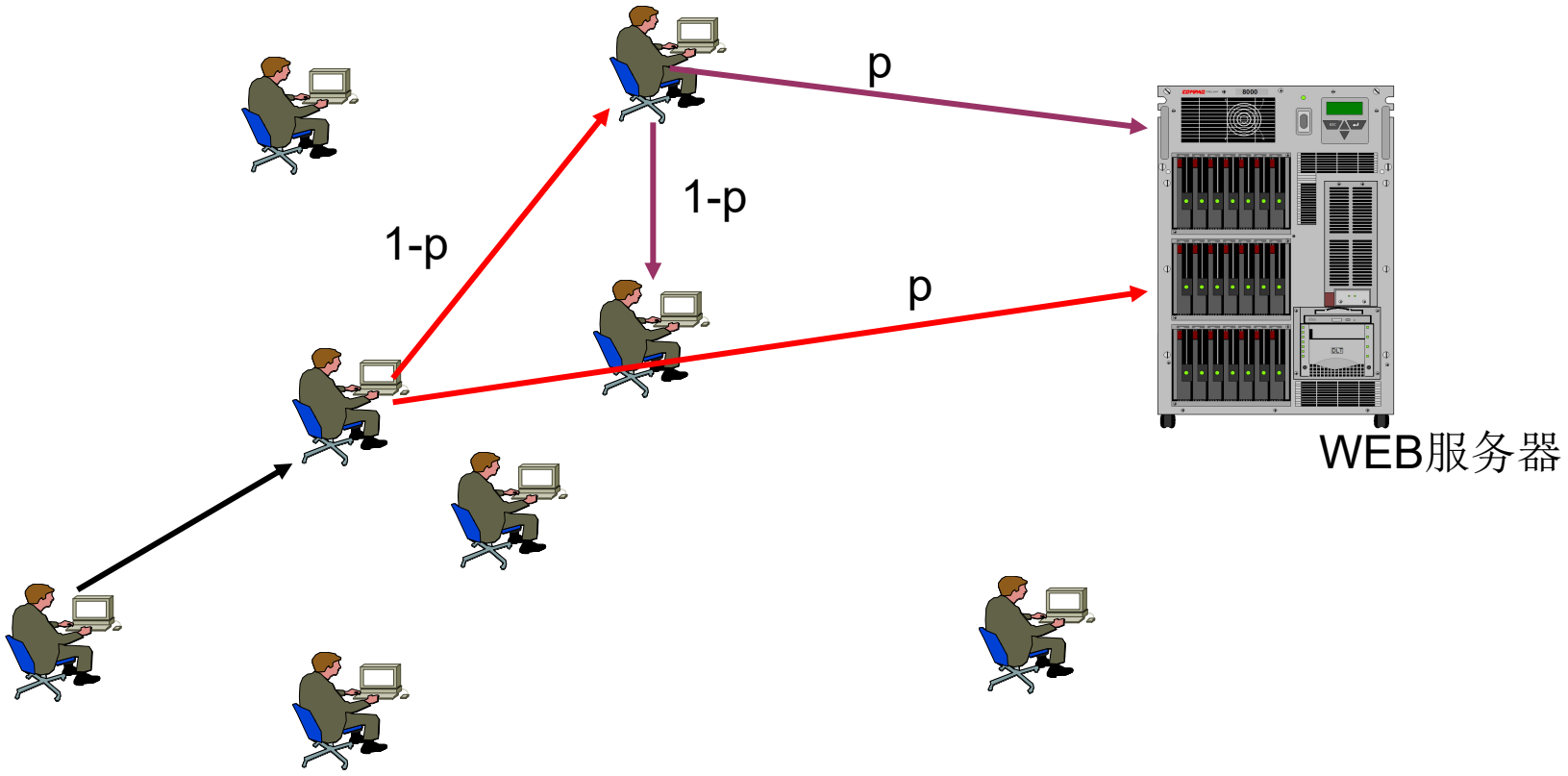
需要消息填充来提供时间关联攻击

匿名与随机路由

- 通过随机路由来隐藏消息源
  - Crowds, Freenet, Onion routing
- 路由节点无法区分所收到的消息是来自真实的发送者或者来自其他路由节点



Crowds System



（1）路由节点形成一个随机路由路径
（2）诚实的路由节点采用抛币的方式决定将请求发送到web服务器或继续转发Routers

Crowds System

- Crowds的匿名性：
  - 观察者无法确定提交web访问请求的用户是真正的发送者还是一个转发者
  - 观察者所看到的web请求的提交者是真正发送者的概率不会超过50%

概率清白（Probable innocence）：真正发送者被视作发送者和非发送者的概率相同

Crowds System

- Crowds协议

STEP 1：发送者随机选择一个Crowds中的节点，采用其加密后发送给该节点

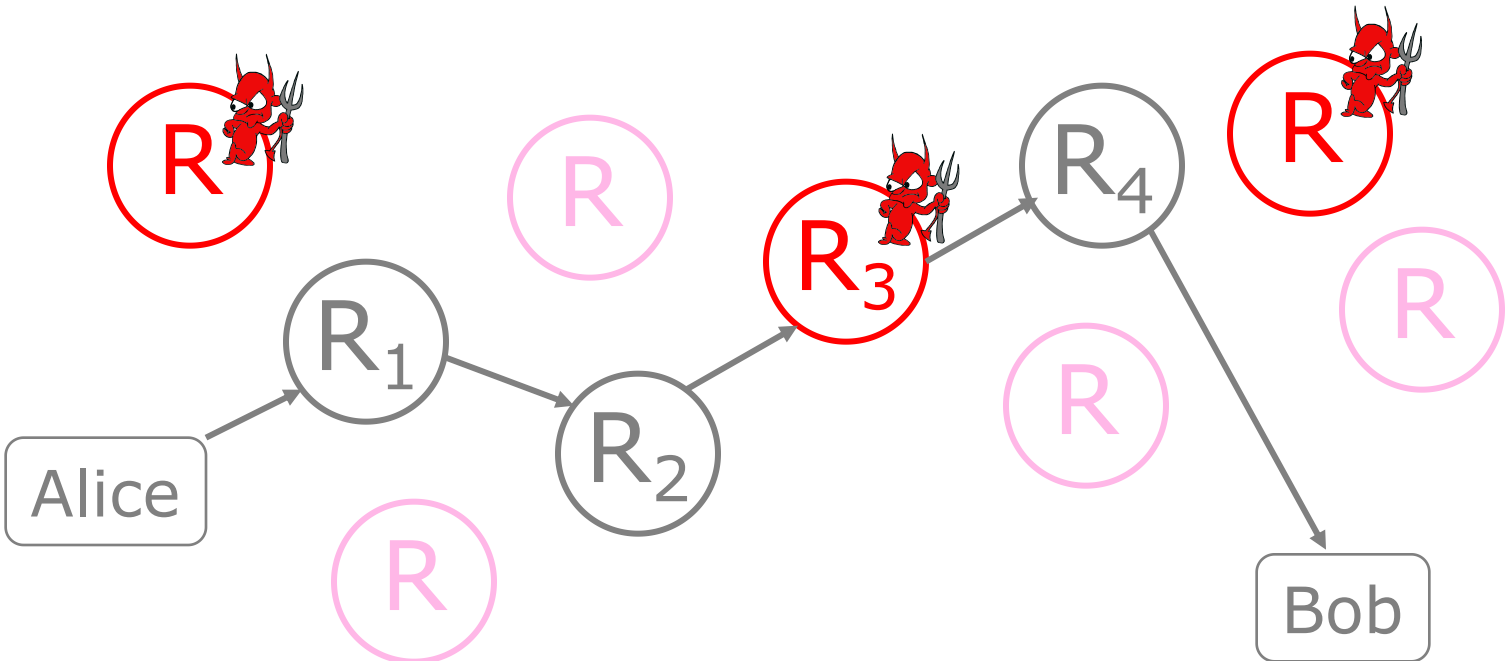
STEP 2：诚实节点收到消息后，利用公正抛币协议以概率p继续转发，则转STEP1；以概率1－p将消息直接提交给信宿。

Crowds的结论：

Crowds中的节点数位n，其中有c个坏节点了；诚实节点转发消息的概率为Pf，则如果满足以下条件，Crowds满足概率清白：

$$n \geq \frac{P_f}{p_f - \frac{1}{2}}(c+1)$$

Onion Routing



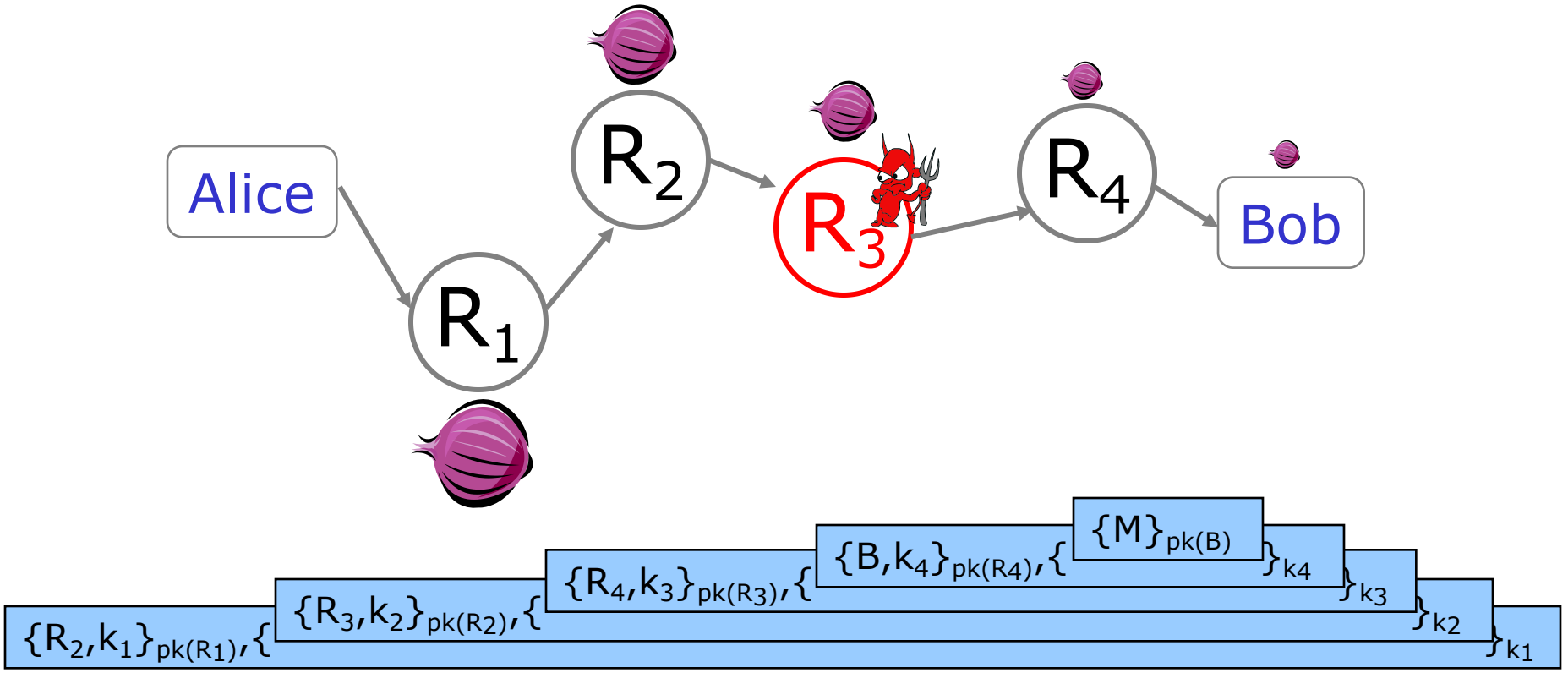
与Crowds的区别：

Crowds是随机选择路径

Onion是发送者控制路由路径的长度，并预先随机选择一条路由路径：某些路由节点是诚实的，某些可能是破坏者。或者每个路由节点在总条数的限制下随机选择下一跳路由节点，并总条数减1，总条数为0时发送给目标节点 。

来自Stanford课件

Onion Routing



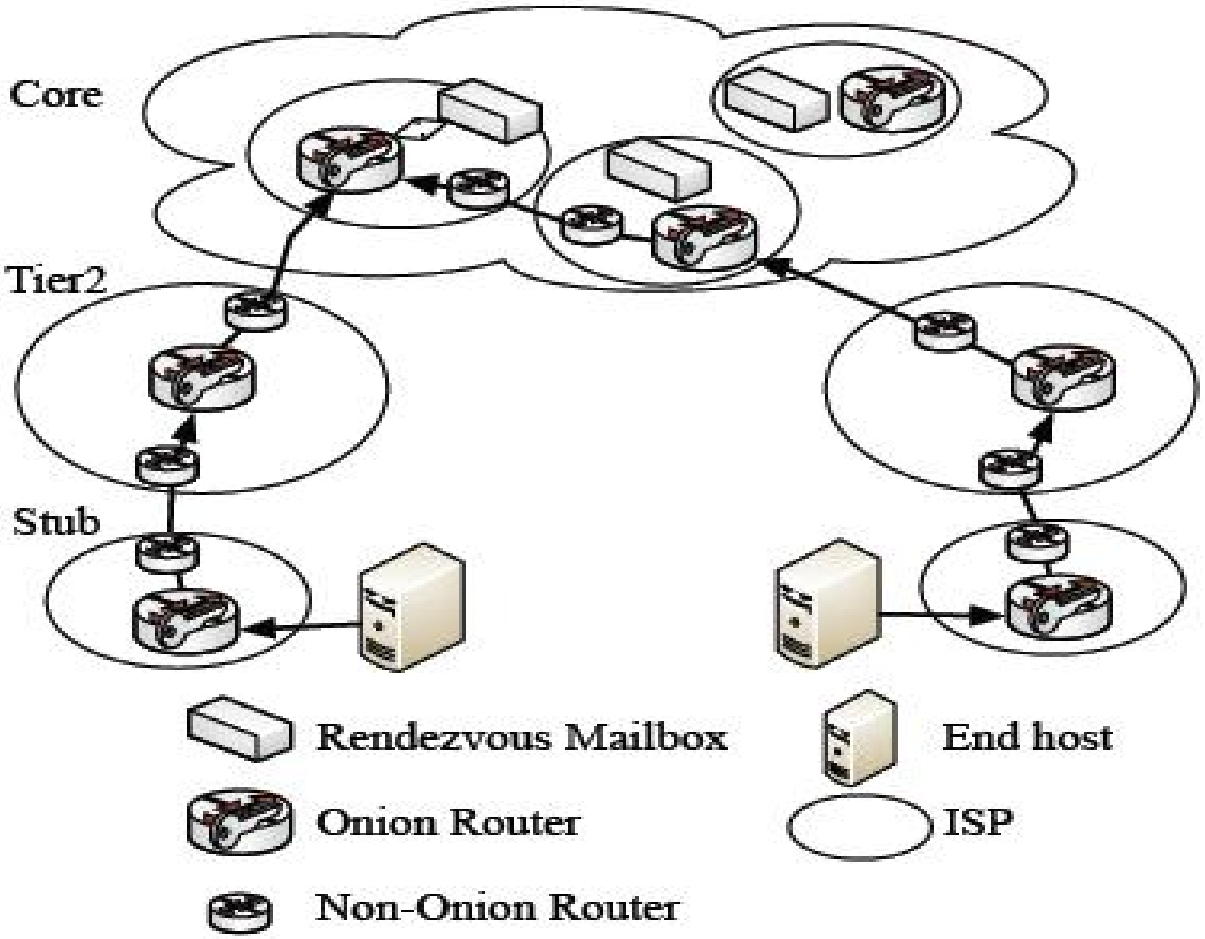
路由信息用路由节点的公钥加密

每个路由节点只能知道下一个路由节点的身份

Tor 网络

- Tor的工作原理如下：
  - 用户运行一个Tor server，用户的电脑就成为一个Tor节点，别人可以通过这个节点访问其它节点，用户也可以通过别人的节点进行访问。
  - 在Tor节点和节点之间的通信是完全加密的（SSL）。
  - 当要访问一个地址时，Tor利用一种路由算法在众多Tor节点中找到一条可达路径。数据经过几“跳”以后，最终能够到达一个可以访问目标资源的Tor节点。
  - Tor的选路过程并不是按照最优的原则，而是随机的。使用随机的路由就使得数据追踪几乎不可能。

Tor网络架构


Core
Tier2
Stub
Rendezvous Mailbox
Onion Router
Non-Onion Router
End host
ISP

- 系统的主要组成部分：洋葱路由器、普通路由器、交汇邮箱、终端用户、核心ISP。用户主要通过洋葱路由和交汇邮箱进行通信。

- 洋葱路由器：对数据进行加密。
  - 预先随机选定路径。中间节点不用知道源地址和目的地址就能够转发数据。
  - 在总跳数限定的情况下，每个节点随机选择下一跳的路由器节点。

- 交汇邮箱：交汇邮箱能共提供DoS攻击恢复能力，所有的包都需要转发给交汇邮箱。交汇邮箱的使用使得网络攻击只会破坏一部分流量。发送者发送数据时，发送者要把数据发送到交汇邮箱中；接收者接收数据时要从交汇邮箱中获得数据。交互邮箱的数目取决于带宽，并且每个交汇邮箱可以被多个终端用户使用。交汇邮箱位于核心ISP中，用户需要使用交汇邮箱时就需要跳转到核心ISP中。
- 核心ISP：核心ISP能够处理大量数据，并且应该有很多同等的核心ISP，这就需要很多一级和二级ISP位于核心ISP中。

实际匿名系统

- Free Haven project has an excellent anonymity bibliography
  - http://www.freehaven.net/anonbib/
- TOR (second-generation onion router)
  - Low-latency overlay network
  - http://www.freehaven.net/tor
- Mixminion
  - Type III anonymous remailer
  - http://www.mixminion.net
- Mixmaster
  - Type II anonymous remailer
  - http://mixmaster.sourceforge.net
- Cypherpunks
  - Assorted rants on crypto-anarchy
  - http://www.csua.berkeley.edu/cypherpunks/Home.html

算法2：建立失效函数*f*。
输入：转向函数*g*和部分的输出函数*output*。
输出：失效函数*f*和完整的输出函数*output*。
方法：

```
begin
    queue ← empty
    for each a such that g(0, a) = s ≠ 0 do
        begin
            queue ← queue ∪ {s}
            f(s) ← 0
        end
    while queue ≠ empty do
        begin
            let r be the next state in queue
            queue ← queue − {r}
            for each a such that g(r, a) = s ≠ fail do
                begin
                    queue ← queue ∪ {s}
                    state ← f(r)
                    while g(state, a) = fail do state ← f(state)
                    f(s) ← g(state, a)
                    output(s) ← output(s) ∪ output(f(s))
                end
        end
end
```

图3 建立失效函数*f*的伪代码

4. 转向函数的构建

图1中树型自动机的状态有0, 1, …, 9。某个状态（通常是0状态）被指定为起始状态。

转向函数把一个由状态和输入字符组成的二元组映射成另一个状态或者一条失败消息。

图1 a) 的状态图代表转向函数g。比如,从0到1标记着h 的边表示g(0, h) = 1，如果缺省箭头则表示失败。可见，对除e和i之外的其他输入字符，都有g(1, h) = fail。所有的树型有限自动机都有一个共同的特点，即对任何输入字符a, 都有g(0, a) != fail。我们将看到，转向函数在0状态上的这种性质确保每个输入字符都可以在状态机的一个操作循环内被处理。

举个例子，记树型有限自动机为状态机*M*。状态机*M*利用图1的函数去处理输入文本“*ushers*”，图4显示了*M*在处理文本串时产生的状态转移情况。

u	s	h	e	r	s
0	0	3	4	5	8	9
				2		

图4 扫描“*ushers*”时的状态转换序列

考虑M在状态4，且当前输入字符为e时的操作循环。由于g(4, e) = 5，状态机进入状态5，文本指针将前进到下一个输入字符，并且输出output(5)。这个输出表明状态机已经发现输入文本的第四个位置是“she”和“he”出现的结束位置。在状态5上输入字符r，状态机M在此次操作循环中将产生两次状态转移。由于g(5, r) = fail，M进入状态2 = f(5)。然后因为g(2, r) = 8，M进入状态8，同时前进到下一个输入字符。在这次操作循环中没有输出产生。

记s为状态机的当前状态，a为输入文本y的当前输入字符。树型有限自动机的一次操作循环可以定义如下：

1. 如果g(s, a) = s，，那么树型有限自动机将做一个转向动作。自动机进入状态s，而且y的下一个字符变成当前的输入字符。另外，如果output(s)不为空，那么状态机将输出与当前输入字符位置相对应的一组关键字。

2. 如果g(s, a) = fail，状态机将询问失效函数f并且进行失效转移。如果 f(s) = s’，那么状态机将以s’，作为当前状态，a为当前输入字符重复这个操作循环。

算法3：树型有限状态机。
输入：一个字符串$y=\{y_1y_2y_3\ldots y_n\}$（其中$y_i$是一个输入字符）；一台 包含上述转向函数$g$，失效函数$f$和输出函数$output$的树型有限自动机。
输出：关键字在y中出现的位置。

```
begin
    state ← 0
    for i ← 1 until n do
        begin
            while g(state, a_i) = fail do state ← f(state)
            state ← g(state, a_i)
            if output(state) ≠ empty then
                begin
                    print i
                    print output(state)
                end
        end
end
```

图5 建立树型有限自动机的算法伪代码

5. AC自动机 (匹配过程总结)

- 预处理阶段：

转向函数把一个由状态和输入字符组成的二元组映射成另一个状态或者一条失败消息。

失效函数把一个状态映射成另一个状态。当转向函数报告失效时，失效函数就会被询问。

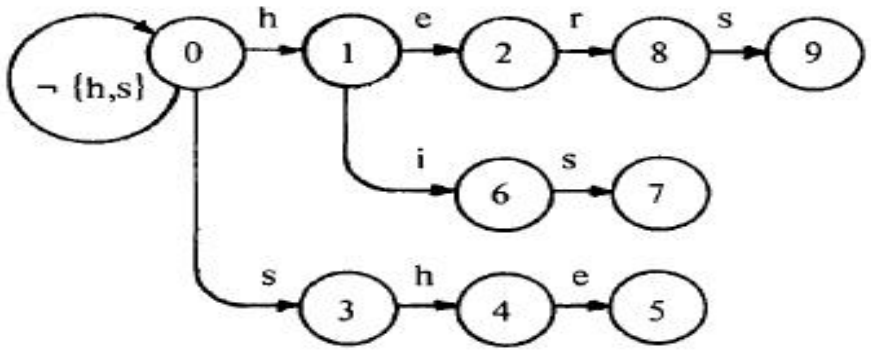
输出状态，它们表示已经有一组关键字被发现。输出函数通过把一组关键字集（可能是空集）和每个状态相联系的方法，使得这种输出状态的概念形式化。

- 搜索查找阶段：

文本扫描开始时，初始状态置为状态机的0状态，而输入文本*y*的首字符作为当前输入字符。然后，开始按照转向函数进行状态转移。如果转移失败则询问失效函数，自动机状态转为失效函数定义的状态。每次的状态转移都要检查输出函数。

- AC算法的特点
  - 自动机的构建是建立在深度优先的基础上
  - 算法构建不受模式集内容变化的影响
    - 模式集内容可动态变化
  - 扫描（检测）的时间复杂度与待测文本长度有关
    - 扫描的效率与模式的长度（扫描深度）有关
  - 算法构建与扫描时存在内存浪费

- AC算法的内存占用问题：
  - 转向函数

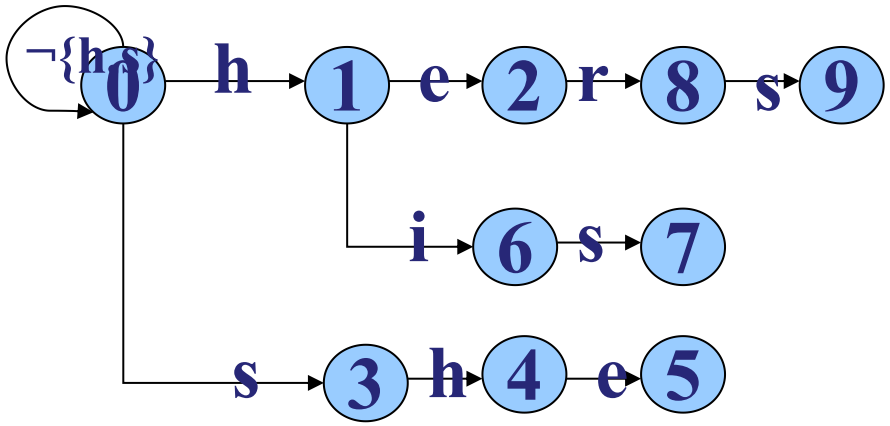


	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
• 0	NULL	NULL						1											3				
• 1	NULL	NULL	2					6															
• 2	NULL	NULL																	8				

- 内存的占用
  - ASCII码集合作为待检测文本的输入与模式集呢？
  - 每个状态的转向函数至少256×4字节=1K
    - 不包括failure，output
  - 实际上，大量的转向是指向Null的
  - 怎么减少无用转向的内存占用呢？

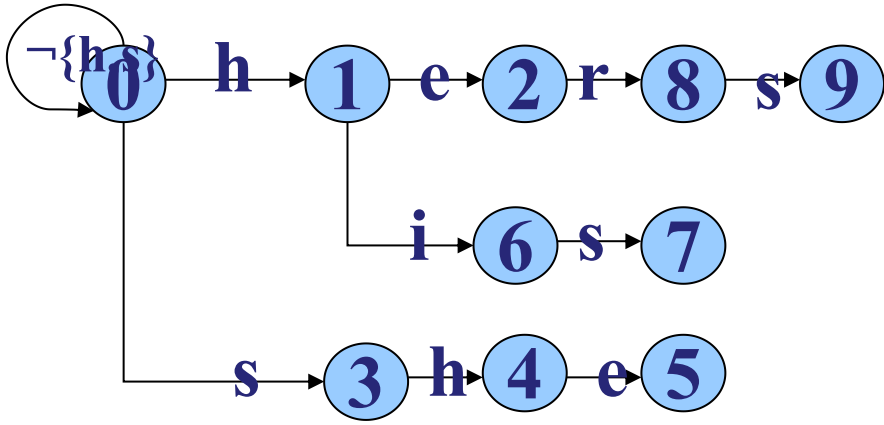
- 基于AC算法的优化方法
  - 行压缩方法
  - 位图方法

- 行压缩方法
  - 每个状态的goto函数只保存能发生转移的可能
  - 例如：状态0时，只有输入h或s时才有转向，那么，goto函数只保存h，s和1，3
  - 即：goto函数是一个数组、队列或链表，保存2，h，s，1，3



- 行压缩方法的特点
  - 状态少于字符集的一半时，能减少内存占用
  - 缺点：扫描时，会发生无用的状态查找
    - 若goto函数转向为Null时，其实查找了该状态的整个队列

- 位图方法
  - 每个状态的goto函数，首先利用位图表明哪个输入会有转向，然后指明转向的状态
  - 如果输入字符集为0-255，即单个字符
    - 则为每个状态建立一个256位的位图，如某位为1，则表明当前状态输入该位对应的字符时有转向。
    - 如0状态，其goto函数为：位图中第104位，第115位为1，其他位为0；然后h，1，s，3。

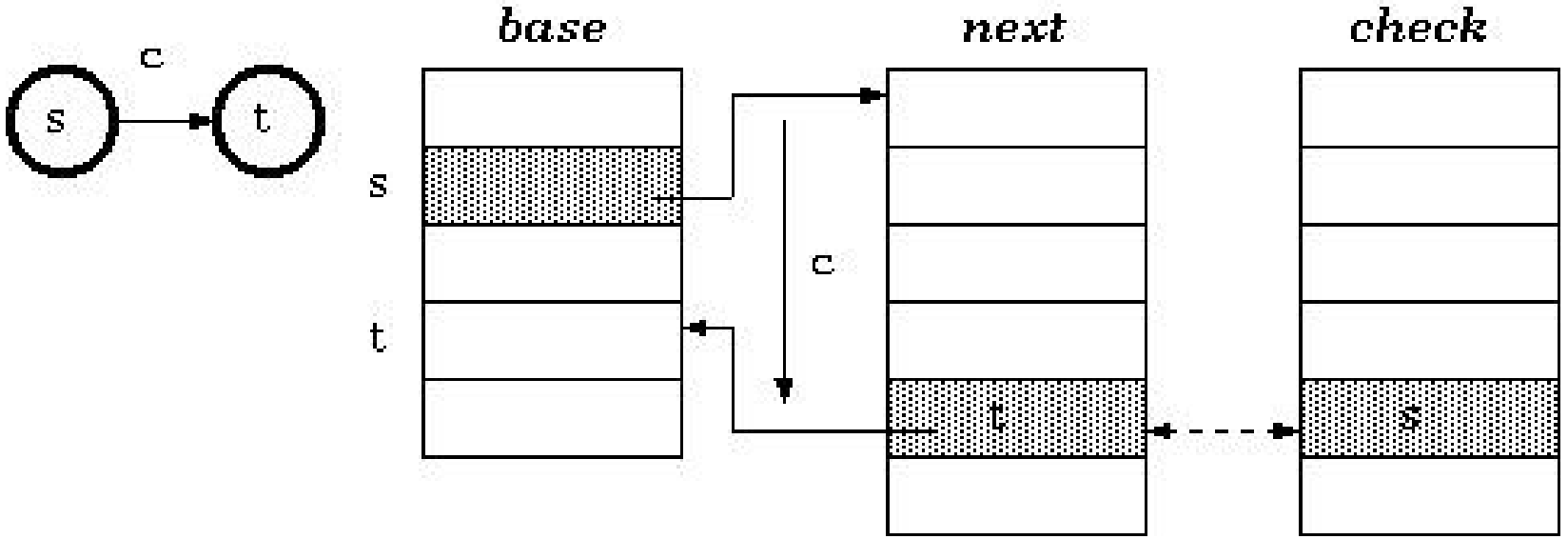


- 位图方法的特点
  - 与行压缩相似，可转向的状态越少，内存节约效果越好
  - 如转向为Null，可以立即通过位图发现
  - 缺点：位图会固定占用32字节（如果字符集是0-255的话，即输入为单个字符）

双数组改进方法

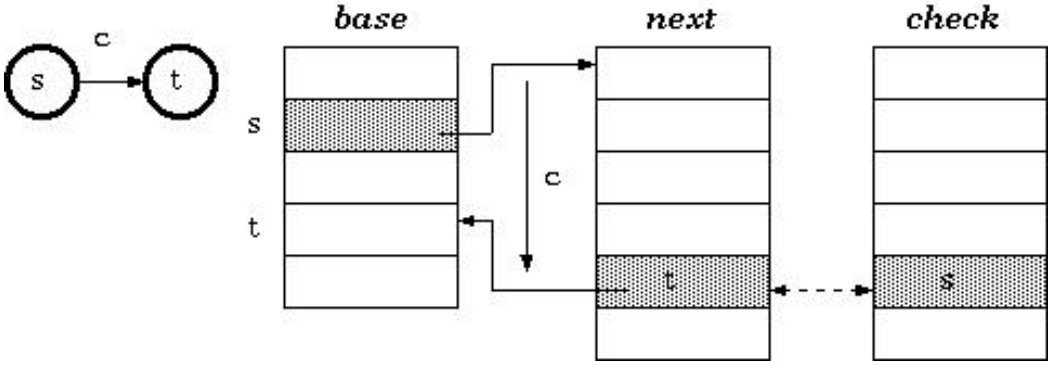
- AC算法改进
  - 转向函数中引入双数组：Base表，Check表
  - Base表：当前状态的Base值+ASCII输入=下一个状态的偏移。
  - Check表：当前状态的父状态信息
    - 父状态是唯一的
  - 自动机构建是建立在广度优先的基础上

- 算法的初始化：转向函数：Next表，Base表、Check表；output函数；Failure函数
  - 转向函数：广度优先
    - Next表
    - Base表
    - Check表
  - output函数与AC算法一样
  - Failure函数
    - 与转向函数一起构造



- Next为转向函数表（数组、链表），下标是位置偏移量，输出是状态值。
- Base表（数组），下标是状态值，输出是Base值。Next表中当前状态为s，输入为c时，假设应跳转为状态t，状态t在Next表中的位置=状态S的位置+状态S的Base值+输入c的ASCII码值。
- Check表（数组），下标是状态值，输出是下标状态的父状态的值。

- 转向函数： Next表，Base表、Check表
  - Next表初始化，模式集输入广度优先，模式集中所有模式的第一个输入作为第一层，考虑构建Next表。 ASCII码输入可能有256种，就申请256个存储单元。每个模式的第一个输入的字符按小到大排序，最小的输入字符，其在Next表中的偏移量为1，其他的输入字符按他的位置对应相应的偏移。



- 如0状态有3个输入c，f，g;
- Next表
- Base表： base[0]=-98
- Check表：Check[1]=0

Check[2]=0; Check[3]=0

0	1	2	3	4	5	6	7	8	9	10
0	1			2	3					
	c			f	g					

0	1	2	3	4	5	6	7	8	9	10
-98										

- 相应的查找算法:状态S，输入为C，求跳转状态next state
  - t := Next[&s+base[s] + c];
  - if check[t] = s

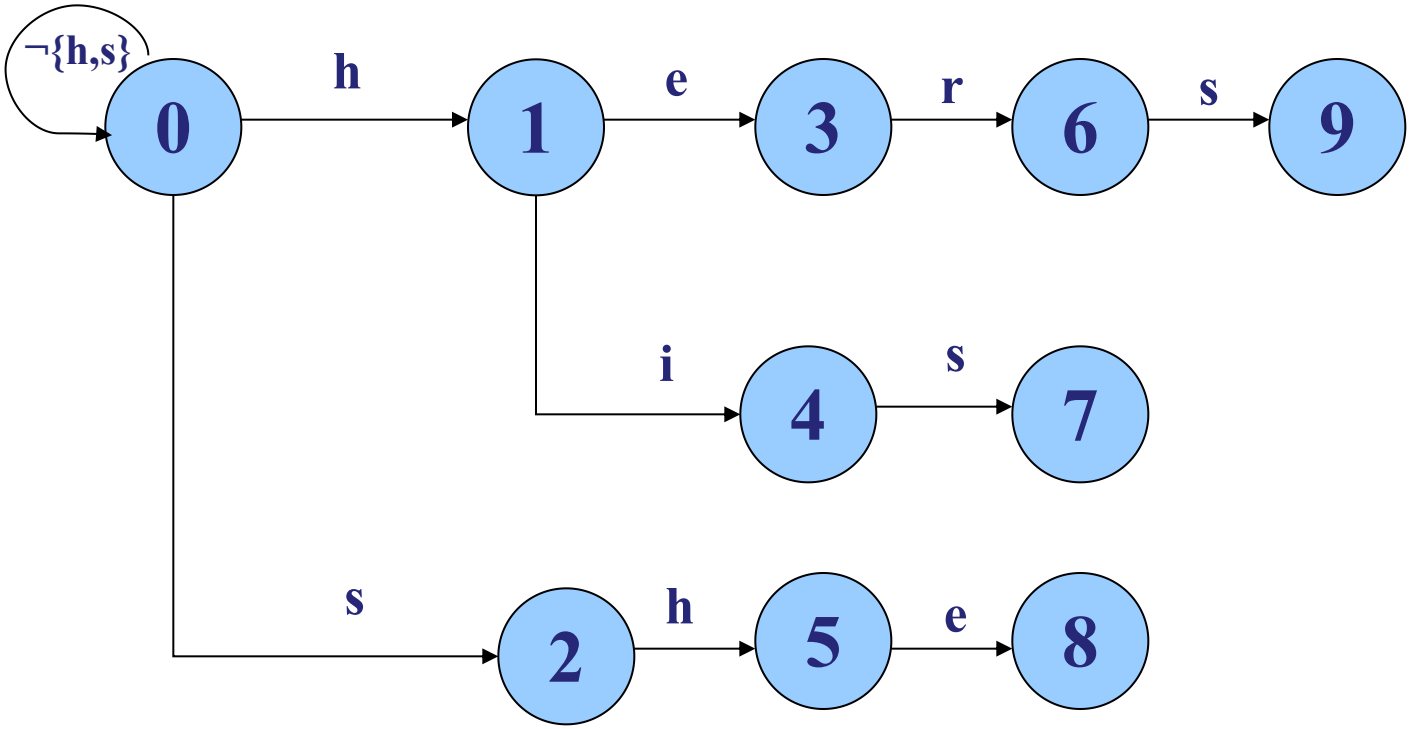
    then next state := t

    else fail

    endif

- 转向函数： Next表，Base表、Check表
  - Next表，Base表，Check表的构建重复前面的步骤。
    - 模式集中的所有第2个输入，第3个输入。。。。。。
    - 每次都从Next的开始部分查找空闲的空间，看是否够分，如果不够分，再申请256的空间。
- Output函数、 Failure函数构建与AC相同。

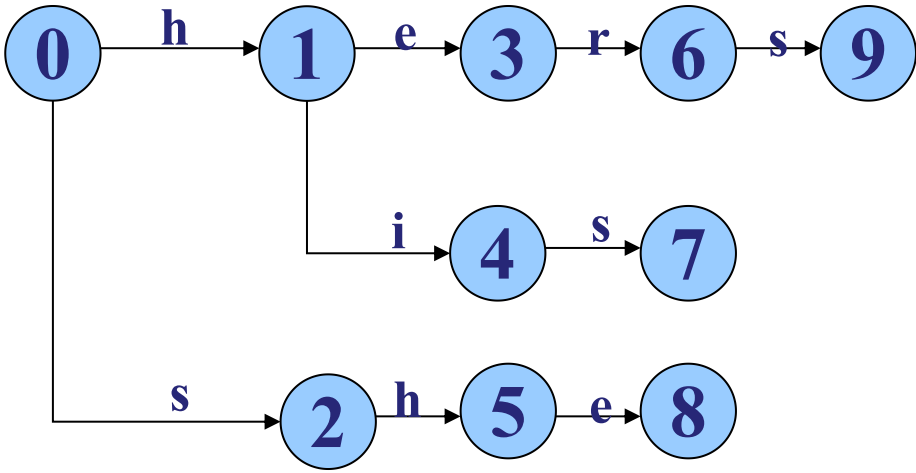
- 例子：构建模式集{*he, she, his, hers*}的有限自动机
  - 模式集第一层{h,s}
  - 模式集第二层{e,h,i}
  - 模式集第三层{e,s,r}
  - 模式集第四层{s}


{hers,his,she}
¬{h,s}
0
h
1
e
3
r
6
s
9
i
4
s
7
s
2
h
5
e
8

- ASCII码h=104;s=115;e=101;i=105;r=114
- 第一层{h,s}
- Next表:产生2个新状态，状态1，2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	1											2								
	h											s								

- Base表：
  - Base[0]=-103
- Check表：
  - Check[1]=0; Check[2]=0;

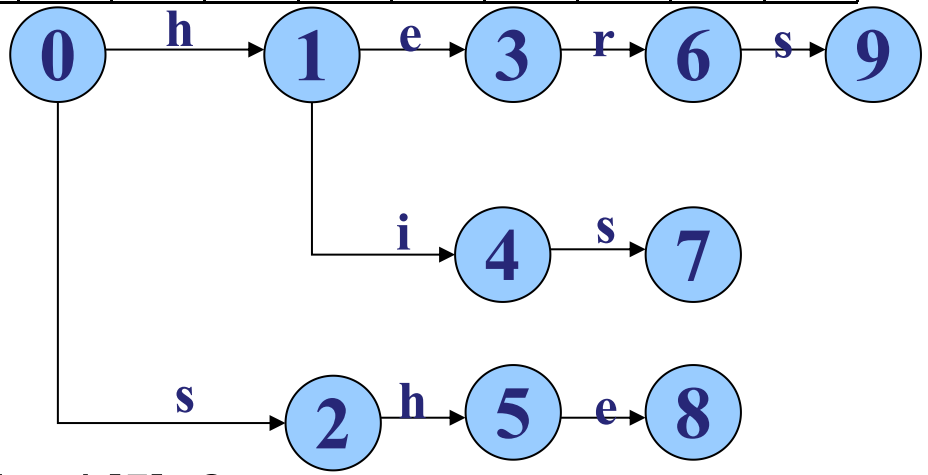


模式集{*he, she, his, hers*}

- ASCII码h=104;s=115;e=101;i=105;r=114
- 模式集第二层{e,h,i}
- Next表:产生3个新状态，状态3，4，5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	1	3	5			4						2								
	h	e	h			i						s								

- Base表：
  - 1状态：he,hi;
    - Base[1]=-100
  - 2状态： sh;
    - Base[2]=-113
- Check表：
  - Check[3]=1; Check[4]=1; Check[5]=2;

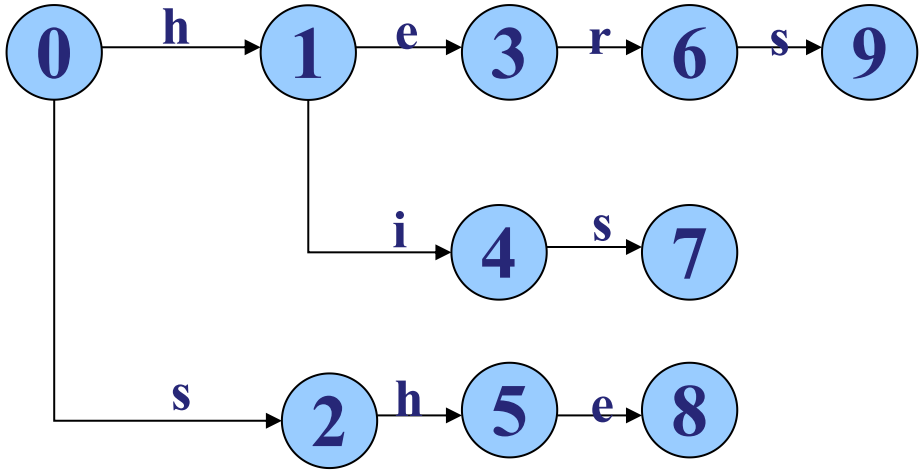


模式集{*he, she, his, hers*}

- ASCII码h=104;s=115;e=101;i=105;r=114
- 模式集第三层{e,s,r}
- Next表:产生3个新状态，状态6，7，8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	1	3	5	6	7	4	8					2								
	h	e	h	r	s	i	e					s								

- Base表：
  - 3状态：her；6状态
    - Base[3]=-112
  - 4状态： his；7状态
    - Base[4]=-116
  - 5状态： she；8状态
    - Base[5]=-97



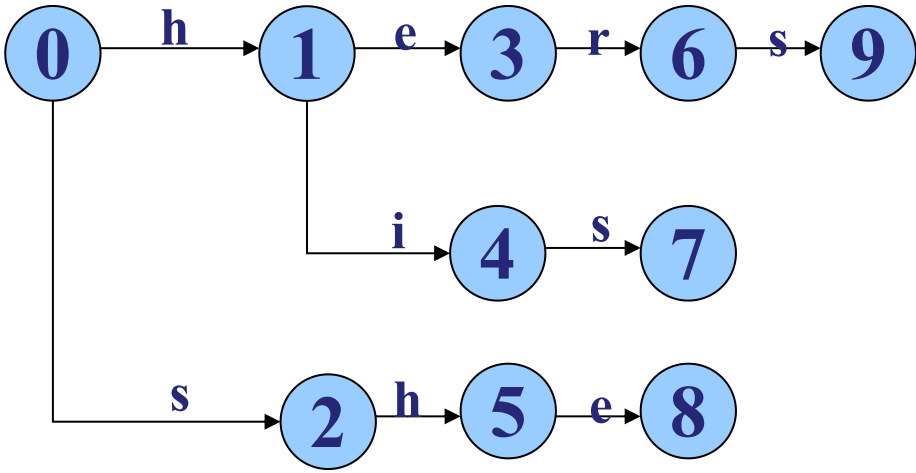
- Check表：
  - Check[6]=3; Check[7]=4; Check[8]=5;

模式集{*he, she, his, hers*}

- ASCII码h=104;s=115;e=101;i=105;r=114
- 模式集第四层{s}
- Next表:产生1个新状态，状态9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	
0	1	3	5	6	7	4	8	9				2								
	h	e	h	r	s	i	e	s				s								

- Base表：
  - 6状态：hers；9状态
    - Base[6]=-111
- Check表：
  - Check[9]=6;



- 双数组算法的特点
  - 内存利用率高
  - 检测(扫描)时间复杂度O(n),检测效率高
  - 缺点
    - 初始化时间长；
    - 模式的变化，无法动态重构

IP地址类信息的匹配？

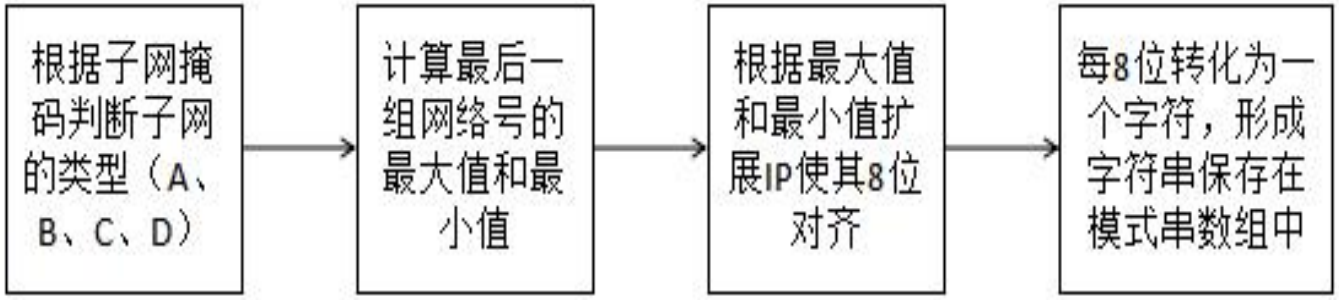
- IP地址匹配的特点和需求
  - 模式串为子网与子网掩码的组合。例如，子网为192.168.0.0，子网掩码为255.255.0.0的模式，可以表示为(192.168.0.0，255.255.0.0)或192.168.0.0/16。
  - 是基于IP地址前缀字符匹配。例如，当模式串为192.168.0.0/17时，只需匹配IP地址的前17位即可。

- 现有的多模式匹配算法针对IP地址匹配的问题
  - AC算法虽然是基于前缀的多模式匹配算法，但其无法很好地表示IP地址“子网+子网掩码”的模式
  - AC算法的失效函数在IP地址匹配中失去作用，且IP地址匹配需精确匹配前缀。
  - WM算法是基于字符后缀的匹配模式，与IP地址的前缀匹配思想不符；同时，其字符跳跃的特性在IP地址匹配中也无法很好的展现出来。

- 基于二进制trie树的IP地址前缀匹配方法
  - 每一个IP地址信息存储在二进制数的一个节点，且此转发信息对应的前缀长度与其所在的层数一致，其目的是查找最长前缀匹配。
  - 二进制trie树每次只比较一个比特位
  - 优化方法：前缀扩展、独立前缀、压缩
  - 缺点：最长前缀匹配（即单一命中），很难适用于多模式匹配的应用场景

IP地址类信息的匹配方法

- 基于AC双数组算法的IP地址匹配



  - A类地址的字符串长度为1个字节，B类地址为两个字节，C类地址为3个字节，D类地址4个字节
  - 模式串为“100.99.98.0/23”，将其转化为C类地址；使用前缀扩展，将其转化为{“192.168.98.0/24”，“192.168.99.0/24”}，则其对应的字符串集合是{“dcb”，“dcc”}。再根据转化后的字符串集合，构建AC自动机。以模式集{“100.99.98.0/23”，“99.0.0.0/8”，“99.97.97.97/32”}为例，预处理后形成的字符串集合为{“dcb”，“dcc”，“c”，“caaa”}

- 基于AC双数组算法的IP地址匹配方法的特点
  - 前缀扩展后的IP模式至多只有4个字节，故优化的DAT算法中，AC自动机最深层数为4，搜索速度快，效果好
  - 不用使用fail函数，所有失效都转向根节点
  - 采用了前缀扩展技术，造成了大量的数据冗余，大大占用了内存空间
  - 考虑到DAT算法的自动机是逐层构建的，模式较多地情况下增加状态节点时会可能会增加next表冲突的概率。

- 习题：利用双数组方法构建模式集 {*tar,the,her*} 的有限状态自动机，列出Next，Base，Check表。

ASCII码 a=97; e=101;h=104;r=114;t=116

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

恶意软件防范

- 特征代码法的优点是
  - 检测准确快速
  - 可识别病毒的名称
  - 误报警率低
  - 依据检测结果，可做解毒处理

- 特征代码法缺点
  - 不能检测未知病毒；
  - 查找已知病毒的特征代码，费用开销大；
  - 依赖于对已知病毒的精确了解，如果一个病毒的特征码有所变化，该方法就会失效。
  - 不能检查多态性病毒
  - 不能对付隐蔽性病毒

多态（形型）性病毒是指采用特殊加密技术编写的病毒，这种病毒在每感染一个对象时，采用随机方法对病毒主体进行加密。多形型病毒主要是针对查毒软件而设计的，所以随着这类病毒的增多，使得查毒软件的编写变得更困难，并还会带来许多的误报。

校验和法

- 计算正常文件的内容或正常的系统扇区的校验和，将该校验和写入文件中或写入数据库中保存。在文件使用/系统引导过程中，检查文件/系统扇区现在内容算出的校验和与原来保存的校验和是否一致，因而可以发现文件是否感染，这种方法叫校验和法。

- 优点：

  方法简单，能发现未知病毒，被查文件的细微变化也能发现。

- 缺点：

  发布通行记录正常态的校验和，会误报警，不能识别病毒名称

校验的内容

- 系统数据
  - 硬盘主引导扇区
  - 系统引导扇区
  - 中断向量表
  - FAT表
- 文件
  - 文件头部
  - 文件基本属性
  - 文件的内容

校验和法 —— 文件的校验和法

- 文件的头部
  - 大多数病毒寄生在宿主程序的头部
  - 寄生在尾部的病毒体也需要替换宿主文件头部的程序入口指针
  - 只需对文件头部的5-20个字节做完整性校验
  - 减少检验时间
  - 提高了误检率

- 文件属性
  - 文件长度
  - 文件的属性日期
  - 文件属性
  - 文件的特定内容

- 文件的内容

  对文件的内容做某种函数运算，这种运算产生的适当字节长度的结果就是校验和。
- CRC校验
- HASH校验
  - MD5
  - SHA

行为监测法

- 利用病毒的特有行为特征性来监测病毒的方法，称为行为监测法。
- 有一些行为是病毒的共同行为，而且比较特殊。在正常程序中，这些行为比较罕见。当程序运行时，监视其行为，如果发现了病毒行为，立即报警。

监测病毒的行为特征

- 改数据区的内存总量（为了防止系统将病毒覆盖）
- 修改内存控制块的段址
- 修改引导扇区
- 对COM、EXE文件做写入动作
- 病毒程序与宿主程序的切换
- 阻塞文件的执行

只有病毒才会同时具有数种此类行为，极少数正常程序也有类似行为

- 阻塞软件：可疑的行为包括
  - 试图打开、查看、删除或修改文件
  - 试图格式化磁盘或其他不可恢复的磁盘操作
  - 对可执行文件和程序代码进行修改
  - 对关键性的系统设置进行修改
  - 电子邮件的脚本程序
  - 网络通讯的初始化

- 优点：

  可发现未知病毒、可相当准确地预报未知的多数病毒。

- 缺点：

  可能误报警、不能识别病毒名称、实现时有一定难度。

病毒采用的反检测新技术 —— 病毒技巧

- 目的：提高病毒的生命力
- 与反病毒技术的对抗中成长
- 手段多样化
  - 隐藏
  - 加密
  - 花指令
  - 病毒代码优化
  - 异常处理

Windows病毒的隐藏技术

- 文件空隙内插入病毒代码
- 隐藏进程
- 创建服务进程
- 创建线程，插入到其他系统进程中

病毒技巧-病毒加密技术

- 对病毒主体代码采用加密技术
- 病毒代码包括
  - 解密算法
  - 病毒主体代码
  - 跳转
- 简单加密技术
  - 相同的加密算法
  - 密钥固定
  - 加密后的代码相同

新的加密技术 - 病毒的多态技术

- 同一个病毒的每个样本的病毒代码不相同，表现为多种形态
- 关键在于每次加密的密钥不同
- 密钥的生成
  - 随机
  - 固定的某些对象

新的加密技术 - 病毒的变形技术

- 高级的加密技术
- 使原始的病毒主体代码总是变化的：解密后的病毒主体代码是不相同的
- 方法
  - 加入垃圾代码
    - 垃圾代码不会破坏有用的寄存器
    - 垃圾代码不会改变存储器的内容
  - 加入变形代码，变换相同功能的代码

- 花指令
  - 反反编译，防破解
  - 干扰反汇编软件的判断
  - 采取的手段，人为的加入一些不影响功能的指令和代码，但在反汇编时产生怪异的代码
- 病毒代码的优化
  - 缩减代码的长度
  - 提高代码运行的效率

新的对抗技术

- 软件模拟法(虚拟机法)
- 启发式扫描
- 主动内核技术

软件模拟法 — 虚拟机技术

- 它是一种软件分析器，用软件方法来模拟和分析程序的运行。
- 作为特征代码法的补充。可用于检测加壳型的病毒。

虚拟机检测

- 查毒的虚拟机是一个软件模拟的CPU，它可以象真正CPU一样取指，译码，执行，它可以模拟一段代码在真正CPU上运行得到的结果。给定一组机器码序列，虚拟机会自动从中取出第一条指令操作码部分，判断操作码类型和寻址方式以确定该指令长度，然后在相应的函数中执行该指令，并根据执行后的结果确定下条指令的位置，如此循环反复直到某个特定情况发生以结束工作，这就是虚拟机的基本工作原理和简单流程。

虚拟机

- 被称为通用解密器
- 在于它不用事先知道病毒体的加密算法，而是通过跟踪病毒自身的解密过程来对其进行解密。
- 虚拟机技术主要用来分析未知病毒和查杀多态变形病毒.具体的思路是用程序代码虚拟CPU、各个寄存器甚至是硬件端口,将采集到的病毒样本放到该虚拟环境中执行,通过分析内存和寄存器以及端口的变化来了解程序的执行情况.当虚拟机技术加入病毒检测引擎中时,由于该技术动态分析程序的变化,因此对于多态变形病毒和未知病毒的发现准确性很高.对于多态病毒,无论如何变化代码和加密代码,最终执行时总会露出其真面目.

例如对正常PE文件最后一个字节不是执行代码片,而病毒一般把自己添加到正常文件的最后一节,并把执行的入口跳到最后一节. 启发式扫描发现这些代码异常之后再对文件进行特征代码扫描,这样会明显地提高扫描效率.

启发式扫描技术（一）

- 启发式扫描技术是在软件系统规模趋于庞大,病毒变种、变形技术多样化的情况下提出的更优化的特征扫描法.
- 启发式是指"自我发现的能力"或"运用某种方式或方法去判定事物的知识和技能"
- 基于专家系统的原理产生的,由于病毒程序和正常程序在执行行为上的不同,作为汇编级的代码分析人员可以很容易地分辨出这类非正常的程序.

启发式扫描技术（二）-提取的特征

- 具有可疑的文件操作
- 具有可疑的重定向操作
- 可疑的内存分配操作
- 错误的扩展名
- 包括搜索定位可执行程序
- 发现解码例程
- 变化的程序入口程序
- 直接写盘的操作
- 程序截获其他软件的加载

启发式扫描技术（二）- 提取的特征

- 可疑的内存驻留
- 可疑的跳转结构
- 无效的操作指令
- 在内存中搬移或改写程序的代码序列
- 执行文件（EXE,COM）辨认程序
- 返回程序入口
- 不合逻辑的错误的时间标记
- 非正常的堆栈

启发式扫描技术（三）

- 启发式扫描确定病毒的方法 – 概率方法
  - 定义可疑功能的调用集合
  - 确定每个特征的权值
  - 加强正常程序的识别能力
  - 自学习功能

主动内核技术（Active K）

- 从操作系统内核这一深度，给操作系统和网络系统本身打了一个补丁，而且是一个“主动”的补丁，这个补丁将从安全的角度对系统或网络进行管理和检查，对系统的漏洞进行修补；任何文件在进入系统之前，作为主动内核的反毒模块都将首先使用各种手段对文件进行检测处理。
- 也被称作主动反应装甲

- 这种理念最大的缺点在于将防治病毒的基础建立在病毒侵入操作系统或网络系统以后，作为上层应用软件的反病毒产品，才能帮助借助于操作系统或网络系统所提供的功能来被动地防治病毒。这种做法就给计算机系统的安全性、可靠性造成了很大的影响。

  能在操作系统和网络的内核中加入反病毒功能，使反病毒成为系统本身的底层模块，而不是一个系统外部的应用软件，一直是反病毒厂家追求的目标。

  嵌入操作系统和网络系统底层，实现各种反毒模块与操作系统和网络无缝连接的反病毒技术，实现起来难度极大。

  Active K（主动内核）技术的要点在于它采用了与“主动反应装甲”同样的概念，能够在病毒突破计算机系统软、硬件的瞬间发生作用。这种作用，一方面不会伤及计算机系统本身，另一方面却对企图入侵系统的病毒具有彻底拦截并杀除的作用。以往的反病毒技术，甚至连“被动反应”都称不上，因为它们本身不具备防护能力，病毒入侵系统时它们不会产生反应，它们之所以有存在的必然性，是因为系统被病毒感染后，能够用这些产品对系统进行反病毒检查与杀除工作。实时化的反病毒技术，可以被称为“主动反应”技术，因为这时反病毒技术能够在用户不关心的情况下，自动将病毒拦截在系统之外。

  但以上技术都不是深入到内核的技术。主动内核技术，用通俗的说法：是从操作系统内核这一深度，给操作系统和网络系统本身打了一个补丁，而且是一个“主动”的补丁，这个补丁将从安全的角度对系统或网络进行管理和检查，对系统的漏洞进行修补；任何文件在进入系统之前，作为主动内核的反毒模块都将首先使用各种手段对文件进行检测处理。

- “主动内核技术”能够在病毒突破计算机系统软、硬件前发生作用。这种作用，一方面不会伤及计算机系统本身，另一方面却对企图入侵系统的病毒具有彻底拦截并杀除的作用。
- “主动内核技术” 能够在用户不关心的情况下，自动将病毒拦截在系统之外。

如果安装一个反病毒软件，这个软件会使用自己编译的，加入反病毒功能代码的视窗程序替换你的视窗程序，你还有胆量安装这样一个杀毒软件吗？

- 这种源代码级的对操作系统的修改难度很大，难以实现
- 可能主动内核技术真正的价值在于和一种全局性的网络管理体系无缝连接。
  - 利用这种网络管理体系，主动内核技术可以自动地探测网络的每一个计算机是否都安装了主动内核，是否都已经升级到了最新的版本，如果有一个计算机没有做到，主动内核就可以对这个计算机进行安装或升级。

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

4. 恶意软件防范

- 恶意软件结构
- 基于特征检测的恶意软件识别
- 基于异常检测的恶意软件识别
- 恶意软件攻击遏制

- 网络病毒检测什么？
  - 在网络上传输的病毒体
    - 人为误用：一般都下载了包含病毒的文件
    - 网络攻击：获得执行权限后，shellcode一般会做下载网络病毒体的操作
    - 局域网访问、弱口令：也都会有传输病毒体的操作
  - 病毒体总要传播到被感染的目标上

网络级的病毒检测技术

- 面向数据包检测技术
  - 问题：由于互联网分组交换的特性，病毒的特征串很可能分布在多个连续的数据包中
  - 连续数据包缓存法
  - 特征码切割法

- 有些恶意软件并没有传播病毒体
  - 资源占用类攻击
  - 蠕虫扫描行为
  - 非法权限类网络攻击
- 蠕虫SYN扫描的预警
  - 大多数蠕虫扫描时会发出大量的SYN数据包
  - 通过对主机发出SYN数据包的频率和SYN数据包本身的规律来判断该主机是否正对网络进行扫描。
  - 当整个网络上的扫描主机达到一定的数量时，考虑当前是不是有蠕虫暴发。

网络蠕虫

- 定义
  - 计算机蠕虫可以独立运行，并能把自身的传播到另外的计算机上。
  - 蠕虫是一个程序，它进入计算机网络，利用空闲的处理器去测定网络中的计算机跨度。蠕虫程序由许多段构成，在其主段的控制下，蠕虫的某个段运行在单独的计算机上。蠕虫典型的传播方式是依靠网络的漏洞，利用网络或电子邮件方式由一台计算机传播到另一台计算机，靠将自身向其他计算机提交来实现再生，并不将自身寄生在另一个程序上。
- 两个基本特征
  - 可以从一台计算机移动到另一台计算机上
  - 可以自我复制

- 蠕虫这个词最早由Shoch和Hupp在1982年提出，他们用这个词来形容一个可以在局域网络自动传播并维护网络中主机的良性程序
- 本来蠕虫是作为分散式计算领域中研究的一部分而被编写的，没有破坏安全的意图，也不隐藏其出现或运作。一般而言，蠕虫本身并不被当作传统的计算机病毒。
- 现在，蠕虫被病毒的制造者们加以利用，很多带有蠕虫性质的计算机病毒被制造出来，它们实际上是蠕虫和病毒的混合体，既有蠕虫的在网络上繁殖的功能，又有病毒的寄生和破坏的功能。

  目前在流行的网络恶意软件，都具有蠕虫的某些性质。

- 蠕虫的行为特点
  - 主动攻击
  - 行踪隐蔽
  - 利用系统和服务的漏洞
  - 造成网络拥塞
  - 降低系统性能
  - 产生安全隐患
  - 反复性
  - 破坏性

蠕虫行为模式分类

- 蠕虫行为模式一般包括：扫描行为、攻击行为、命令控制行为、文件传输行为和激活行为
- **扫描行为(Scan):**蠕虫发出一系列数据报文来扫描目标主机是否在线，或者是否开启相应服务，或者是否存在漏洞，这样的一系列事件被称为蠕虫的扫描行为。扫描的顺序一般也是蠕虫复制、传播的顺序过程。
- **攻击行为(Attack):**蠕虫利用目标主机服务存在的漏洞，以进入目标主机并获得一定权限为目的的一系列事件被称为蠕虫的攻击行为。
- **命令控制行为(Control):**蠕虫利用攻击目标主机漏洞的结果和目标主机建立控制连接，从而可以在目标主机的shell环境下执行命令。蠕虫建立控制连接、向目标主机下达命令的过程被称为蠕虫的命令控制行为。

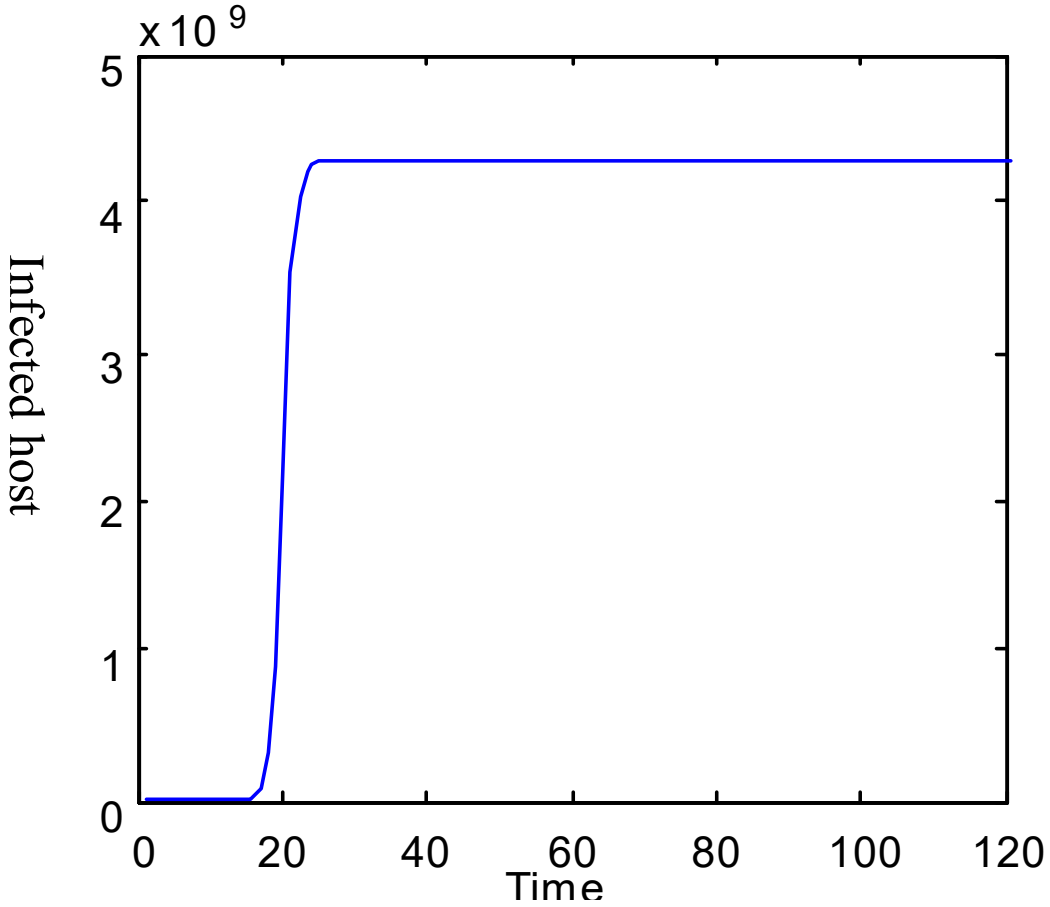
- **文件传输行为(Transmit)：**蠕虫利用TFTP等传输方式将蠕虫副本、攻击工具等传到目标主机上的过程称为文件传输行为。从传输协议的角度来看，文件传输行为可以是一个单独的传输连接。需要说明的是，蠕虫副本可以以文件传输的方式进行，也可以嵌入到攻击行内进行传输——即攻击成功完成的同时蠕虫副本传输也成功完成。比如Blaster利用TFTP向目标主机传输蠕虫副本，蠕虫CodeRed的蠕虫副本和攻击行为结合在一起进行传播。
- **激活行为(Provoke)：**是指目标主机已经获得蠕虫副本，攻击者通过命令控制或者攻击等方式激活目标主机上的蠕虫的过程称为激活行为。比如蠕虫Nimda通过执行类似“GET /scripts/Admin.dll HTTP/1.0”来激活目标主计上的蠕虫，蠕虫Sasser和Blaster通过控制连接激活目标主机上的蠕虫。

蠕虫的扫描（扩散）

- 随机扫描
- 本地优先扫描
- 顺序扫描
- 基于目标列表的扫描
- 基于路由信息的扫描
- 分治扫描策略
- 被动式扩散

随机扫描扩散

- 均匀随机扫描
  - 均匀随机扩散是指从扫描空间内随机抽取一个IP地址作为目标传播，扫描空间可以为整个IPv4地址空间
  - 算法简单、易实现，会产生大量异常的流量，容易在早期就被检测系统发现。
- 选择性随机扫描
  - 目标地址按照一定的算法随机生成，但对公网中不可能出现的地址块进行标识，避免扫描这些地址。
  - 算法简单、易实现的特点，若与本地优先原则结合，则能达到更好的传播效果。但选择性随机扫描容易引起网络阻塞，使得网络蠕虫在爆发之前易被发现，隐蔽性差
- CodeRed蠕虫、Slapper蠕虫、Slammer蠕虫



蠕虫传播趋势

本地优先扫描

- 主导思想：优先选择本地或者与本地相近的网络进行扫描。
  - 被感染主机上蠕虫的生成目标地址和所在主机的IP地址在同一个子网的概率比较大。如：Nimda蠕虫生成的目标地址有50%的概率在当前机器IP所在的B类地址范围内产生，有25%的概率在当前机器IP所在的A类范围内产生，另25%的概率是随机IP地址。这种扩散策略的支持者认为，一个主机被感染蠕虫之后，这个主机所在的子网的IP地址被分配出去并且被使用的概率比较大，从而能够增加发现漏洞主机的概率。
  - 将互联网地址空间中未分配的或者保留的地址块排除在扫描之列
  - 实现简单，传播速度与概率P的选取、地址空间数目相关
- 比如Blaster蠕虫和Nimda蠕虫，

顺序扫描

- 顺序扫描(sequential scan)：是指被感染主机上蠕虫会随机选择一个C类网络地址进行传播。根据本地优先原则，蠕虫一般会选择它所在网络内的IP地址。若蠕虫扫描的目标地址IP为A，则扫描的下一个地址IP为A+1或者A−1。一旦扫描到具有很多漏洞主机的网络时就会达到很好的传播效果。
- 该策略的不足是对同一台主机可能重复扫描，引起网络拥塞。所以尽量避免父节点与自节点的扫描空间相重合。多采用随机策略选择初始节点，扫描速度等价于随机策略
- W32.Blaster是典型的顺序扫描蠕虫。

基于目标列表的扫描扩散算法

- 基于目标列表的扩散是指网络蠕虫在寻找受感染的目标之前预先生成一份可能易传染的目标列表，然后对该列表进行攻击尝试和传播
- 目标列表生成方法有两种：①通过小规模的扫描或者互联网的共享信息产生目标列表；②通过分布式扫描可以生成全面的列表的数据库。
- 基于目标列表的扩散提高了蠕虫的传播速度，不足是网络蠕虫传播时必须携带一个IP地址库，使得蠕虫负载量增大。
- 分治扫描策略的另一个不足是存在“坏点”问题。在蠕虫传播的过程中，如果一台感染蠕虫的主机死机或崩溃，那么其所对应的剩余扫描列表中主机就会失去被感染的机会。并且，这个问题发生得越早，影响就越大。

基于路由表信息的扫描

- 网络蠕虫根据网络中的路由信息，对IP地址空间进行选择性扫描的一种方法。
- 采用随机扫描的网络蠕虫会对未分配的地址空间进行探测，而这些地址大部分在互联网上是无法路由的，因此会影响到蠕虫的传播速度。如果网络蠕虫能够知道哪些IP地址是可路由的，它就能够更快、更有效地进行传播，并能逃避一些对抗工具的检测。
- 网络蠕虫的设计者通常利用BGP路由表公开的信息获取互连网路由的IP地址前辍，然后来验证BGP数据库的可用性。
- 基于路由的扫描极大地提高了蠕虫的传播速度，以CodeRed为例，如果采用路由扫描，则蠕虫的感染率是采用随机扫描蠕虫感染率的3.5倍。

分治扫描

- 分治扫描(divide-conquer scan)：网络蠕虫之间相互协作、快速搜索易感染主机的一种策略。
- 网络蠕虫发送地址库的一部分给每台被感染的主机，然后每台主机再去扫描它所获得的地址。主机A感染了主机B以后，主机A将它自身携带的地址分出一部分给主机B，然后主机B开始扫描这一部分地址。
- 避免重复扫描，缺点：一旦节点失效，扫描中断

扫描策略评价

- 在理想情况下，当漏洞主机均匀分布在网络中的时候。
  - 蠕虫采用均匀随机扩散策略时传播速度比采用本地优先策略传播速度快
  - 基于目标列表的扩散策略当列表小于6M字节时，蠕虫传播速度比均匀随机扫描快
- 实际中仍然主要是随机扫描和顺序扫描两种方式，使用其它的传播策略的蠕虫还很少。通常蠕虫对目标列表的扫描，以及局域网内的顺序扫描都仅仅是在蠕虫传播的早期，大量的随机扫描则范围大，持续时间长，构成了蠕虫传播的主体，例如，曾经大肆泛滥造成巨大危害的：Code Red、Code Red II、Slammer、Blaster、Sasser、Welchia等蠕虫

被动式扩散算法

- 被动式传播蠕虫则不需要主动扫描就能够传播。被动式蠕虫通过潜伏在已感染主机中，等待潜在的攻击对象来主动接触它们，或者通过监听网络数据报文，获取其它用户活动信息，从而发现新的攻击目标。
- 由于它们需要目标触发，所以当目标数量少或者扫描频率低的时候传播速度很慢，但当目标造成的扫描频率很高，则传播速度会很快。
- 这类蠕虫在发现目标的过程中并不会引起通信异常，这使得它们自身有更强的安全性。Contagion和CRClean都是被动式蠕虫，Contagion通过监听正常的通信来发现新的攻击对象；CRClean则通过等待CodeRed II的扫描活动来发现新的攻击对象，当它发现有CodeRed II向自己所在主机扫描时，就发起一个反攻来回应该CodeRed，如果反攻成功，它就删除CodeRed II，并在目标主机上建立CRClean副本。

基于异常发现的检测

- 误用技术的不足
  - 不能检测新的病毒
  - 不能早期预警
- 异常检测：通过异常发现技术来发现蠕虫导致的网络异常
  - 异常检测是数据挖掘中一个重要方面，用来发现”小的模式”(相对于聚类)，即数据集中间显著不同于其它数据的对象。

什么是异常（outlier）？

- Hawkins(1980)给出了异常的本质性的定义：异常是在数据集中与众不同的数据，使人怀疑这些数据并非随机偏差，而是产生于完全不同的机制。
- 异常检测算法对异常的定义：异常是既不属于聚类也不属于背景噪声的点。他们的行为与正常的行为有很大不同。

什么是异常检测

- 前提：网络恶意代码产生的网络行为是异常活动的子集
- 用户轮廓(Profile): 通常定义为各种行为参数及其阈值的集合，用于描述正常行为范围
- 检测过程
- 修正
- 正常数据集 → 量化 → 建模 → 比较判定

异常检测方法

- 基于统计的检测方法
- 基于距离的检测方法
- 基于密度的检测方法
- 基于聚类的检测方法

异常检测系统的效率取决于用户轮廓的完备性

异常检测的方法 — 统计（一）

- SYN扫描的特性
  - 会发出大量的SYN包。
  - 如果扫描的目的地址不存在，那么发出的SYN包不会有回应。
  - 如果扫描的目的IP存在，但是没有开相应的端口，那么目的主机将返回ACK+RST包。
  - 如果发出的是大量UDP数据包，那么当目的主机不存在或者端口不开放时，会返回ICMP目的不可达报文。
- 正常的扫描
  - 会有大量的相应报文：ACK&SYN标志的数据包。
  - 具有ACK&SYN标志的数据包和只具有SYN标志的数据包之间数量差别是比较大的

- 恶意SYN扫描的特征
  - 某些主机对另外一些主机特定端口的扫描。蠕虫在扫描网络时，一般就是这种情况。它对网络危害是最大的，严重时常常造成网络堵塞。
  - 某些主机对同一台主机不同(特定)端口进行的扫描。这种情况最常见的是对于某个特定网站进行攻击（没有ACK数据包）。它造成的直接后果是被攻击主机崩溃，对于整个网络流量不会产生很大的影响。
  - 某台主机对另外一些主机不同端口的扫描。

- 基于统计的检测方法
  - 统计客户端扫描频率，当出现单位时间内扫描次数超过阀值时便报警。在蠕虫暴发时，这些机器很有可能都感染了蠕虫。
  - 统计扫描包的目的地址，并对回应数据包计数；如果发现有明显聚集现象，并且回应包的数量远远小于连接包数量，便可以报警。
  - 根据ACK&RST的数据包来判断，统计具有ACK&RST标志的目的IP和源端口信息，因为这就是攻击主机在发出SYN数据包时的源地址和目的端口，通过对聚集情况的分析确定出当前扫描客户端的IP和被攻击的端口。然后反过来统计这些IP的扫描情况来判断该客户端是不是同时也对其他机器有扫描行为。

面向恶意流检测的主动队列方法

- 路由器级别的常用控制方法
  - 丢尾算法
  - RED(Random Early Detection)算法
  - 优点：较低的时延、较高的吞吐量和较好的公平性
  - 缺点：不加区分的数据包丢弃，导致路由器FIFO队列被大量恶意数据流占用，控制不了恶意流

随机早期检测RED(Random Early Detection)算法是广泛应用于路由器中的IP层的拥塞控制算法之一,它的主要思想是在拥塞发生以前,通过计算队列中包的丢失概率,从而随机丢弃一 部分数据包,以达到实现网络拥塞控制的目的.

随机早检测算法通过计算TCP流的平均队列长度,进行适当的概率丢弃分组,从而有效地避免了由TCP流导致的网络拥塞.该算法因其具有较低的时延、较高的吞吐量和较好的公平性而被广泛采用。

- Clue（Compare and Limit Unresponsive flows）算法

```
for each packet( )
  if (packet来自恶意流)
      处理来自恶意流的packet，并处理该数据流
  else
      packet 进队；
  else
    判断packet是否是来自恶意流，并处理packet
      if packet不是恶意数据流
          以概率p_mark丢弃此包；
          更新全局丢包概率p_mark;
```

- 算法思想
  - Clue以瞬时队列作为拥塞程度的指示
  - 算法把网络上的数据流分成两大部分：恶意数据流和正常数据流。对于恶意数据流，根据每个恶意流的不同情况，算法维持不同的丢包概率； 对于其他的正常数据流，算法维持一个全局丢包概率p_mark
  - 这样既可以做到恶意流之间区别服务，也可保证正常流和恶意流之间的公平性。
- 算法关键：恶意流的识别！！！！

- 恶意流的识别（统计方法）
  - 对于当前到达的网络数据包，clue算法从对列里面随机挑选M个元素与其比较，设*saddr*为当前到达的网络数据包的用户，*rc*为比较结果值。则*rc*可以表示为：

$$rc = \sum_{i=1}^{M} saddr \Lambda x_i$$

    - 其中 Λ 的含义为：同或 $a \Lambda b = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$
  - *rc*大于一定阈值就认为是恶意流

- 识别算法

```
// rc初始化为0；
for each arriving packet
    从用户对列随机挑选M个元素x_i
    for i= 1:M
     if x_i 与当前网络用户相同
           rc++;
     end for
    if  rc的被怀疑次数大于阈值
     标记此网络用户为恶意攻击用户
     丢弃此数据包
end for
```

异常检测的方法 — 统计（二）：分布

- 假设给定的数据集服从一个随机分布（如正态分布等），用不一致性测试识别异常。
- 统计分析方法
  - 需要创建一个统计描述，归纳出一些测量属性（如：源地址、目的地址和目的端口的计数等），并记录正常使用时的这些属性的值。
  - 建立正常的行为模型，设定置信区间的宽度。统计的方法置信区间的宽度取决于用户设置不同的置信度，不需要以前关于异常活动的知识，从观测值中发现统计异常，而且置信区间随着测量数据的增多，自己改变。
  - 对网络上的传输数据进行测量，如果测量属性不在置信区间内时，那我们就认为有异常行为发生。

统计（二）：分布

- 模型假设：我们已经获得测量属性X，{x1， … xn，}，则根据它们可得：

$$sum = \sum_{i=1}^{n} x_i \qquad EX = sum / n \qquad \sigma = \sqrt{X^2 - (EX)^2}$$

- 置信区间由标准方差和参数d决定，其中参数d可以根据置信度的要求进行选择：

$$[EX - d * \sigma, \quad EX + d * \sigma]$$

- 根据Chebyshev不等式：

$$P(|X - EX| \geq \varepsilon) \leq \frac{DX}{\varepsilon^2} \qquad \varepsilon = d * \sigma$$

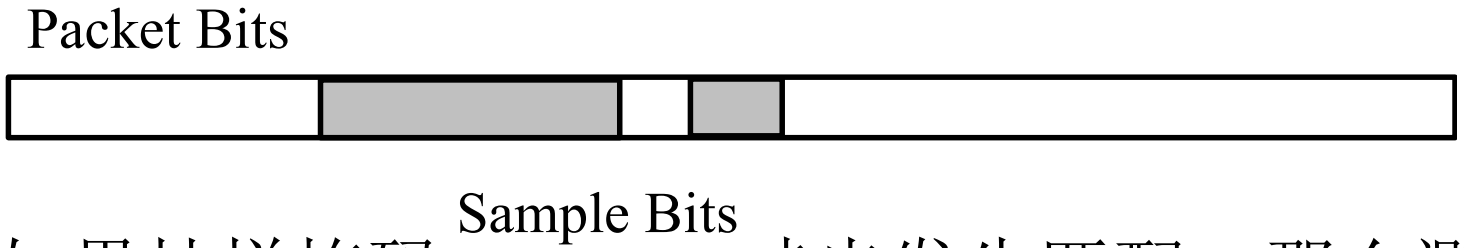
统计（二）：分布

- 落在这个区间外的概率最大为$1/d^2$，置信度为$1—1/d^2$。我们选取d= 4， 则误报的最大可能为0.0625。如果新的观测值X_n+1不落在置信区间内，被认为是异常。如Worm.Welchia暴发时，会产生大量的ICMP包。我们平时统计ICMP在每个时间段的数量的序列X，如果下一个时间段的ICMP包数目不在置信区间内，则认为有异常发生，可能感染蠕虫。
- 统计的优点是算法与实现非常简单、可检测到未知的和更为复杂的入侵；缺点是有误报、漏报率，敏感性不强。

统计（三）：抽样

- 对大规模骨干网络进行异常行为的实时检测会受到系统性能的约束，主要是：1、处理速度，包括网络适配器硬件、协议栈还原和检测算法的速度；2、存储容量，因为骨干网带宽很高，每秒的数据如果存储下来就是以G为单位。在这种情况下，统计方法难以胜任。
- 抽样检测：不需要使用传统方法测量所有的流量数据、记录所有的流量信息，为了能够实现实时需求，使用抽样报文测量代替测量所有流量信息。抽样可以采取定时抽样，或者根据一定规则(随机抽样、泊松抽样等)抽样。
- 利用抽样获取的数据，进行统计分析，即可进行异常发现

- 分布式抽样：特定网络数据流的抽样。
- 使用确定的掩码和标识字段的部分比特相匹配以实现报文抽样。



- 如果抽样掩码和匹配比特串发生匹配，那么测量器将抽样该报文。可以根据源IP、目的IP等等所在的位，相当于利用如下的表达式：

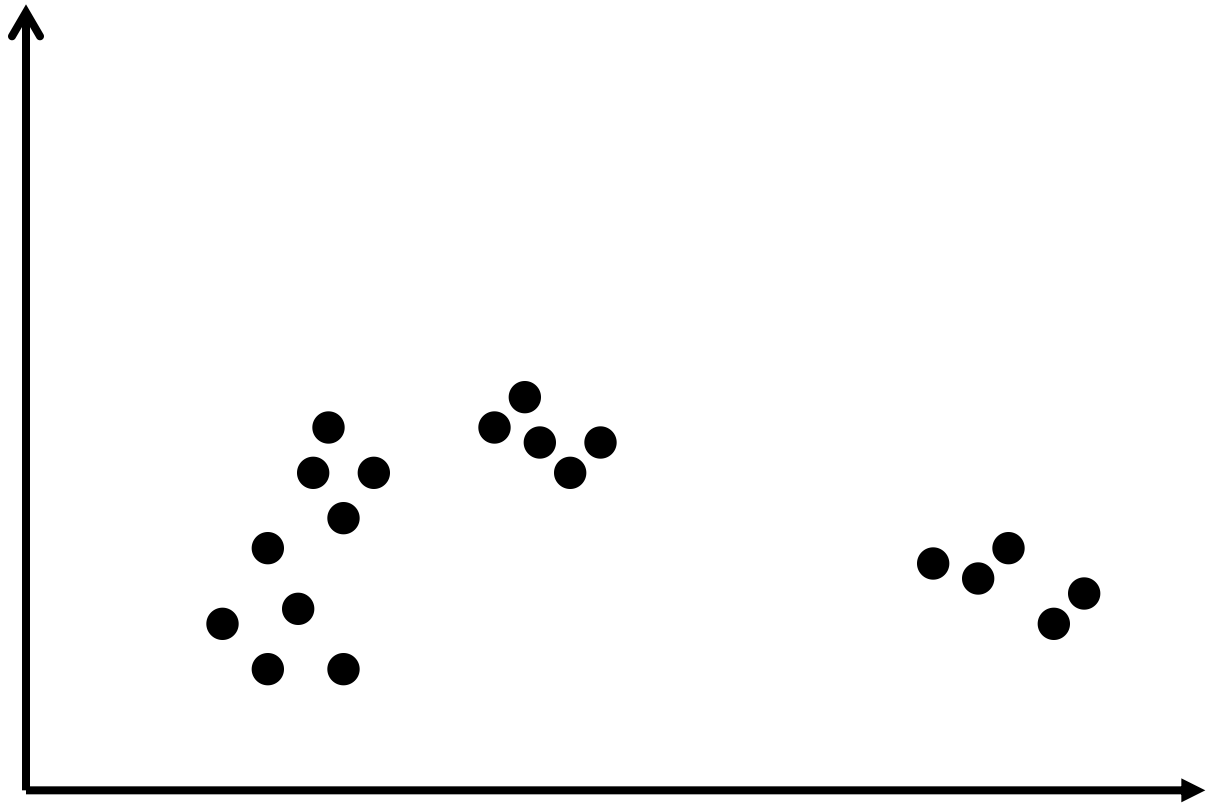
  （报文源IP&掩码）&指定IP

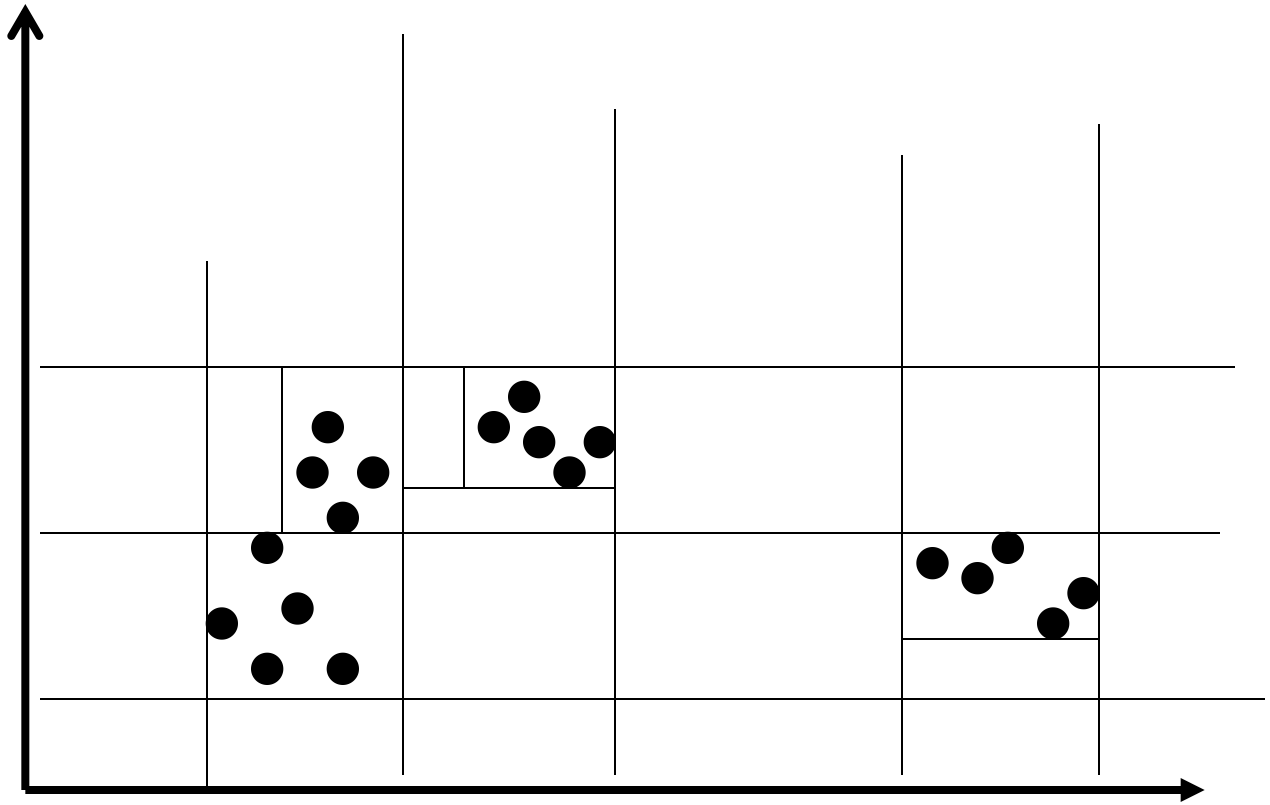
- 可以对指定的IP或者网段进行抽样，将获取的数据保存在历史窗口中

- 通过修改抽样参数使抽样所获得的样本能够尽量好的描述所有流量的统计属性。
- 抽样的优点，就在于可以更集中、更详细的处理分析样本；缺点就在于容易漏掉一些异常信息。

异常检测的方法 — 基于距离的检测方法

- 多维的输入向量通过计算欧氏距离来确定是否是异常
  - 海量的训练数据
  - 多输入变量之间的关联
  - 发现海量数据背后规律





基于距离的异常检测算法

- 欧式空间
  - 将网络数据以矢量的形式表示为特征矢量 $X=(X_1,X_2,\ldots X_k)$，x_i为每个网络数据特征向量的分矢量；特征矢量X抽象为k维特征空间$\mathbf{R_k}$的一个点，其中$\mathbf{R_k}$为欧式空间。
  - 用户正常轮廓则可看成是多维空间$\mathbf{R_k}$上的点集（网络数据的集合），待测网络数据则可以看成是多维空间$\mathbf{R_k}$上待测量的点。
- 欧氏距离
  - 待测数据与用户轮廓的差最终可以归结到多维空间上两点间的距离。 即欧氏距离。

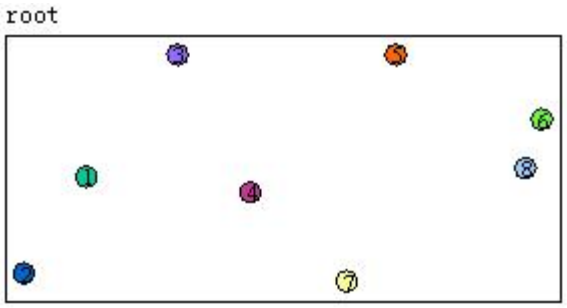
$$d(i,j)=\left\|x_i-y_j\right\|=\sqrt{\sum_{p=1}^{k}(x_{i,p}-y_{j,p})^2}$$

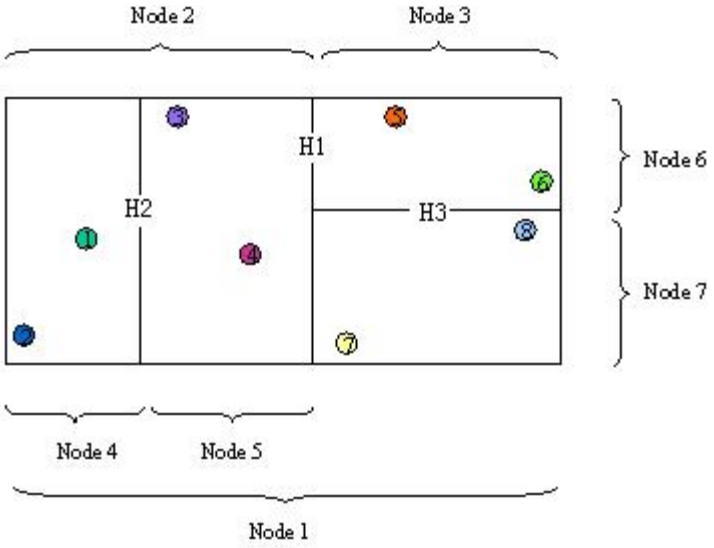
- 异常检测
  - 偏差
    - 设X为待测数据包对应的多维空间点，N为用户轮廓中点的个数，D_i为X与用户轮廓中每个点的距离($1 \le i \le N$)，且$\{D_1, D_2 \ldots D_i\}$为由小到大的递增序列，则$d=(D_1+D_2+\ldots+D_i)/i$ 代表待测数据包与用户轮廓的偏差，其中$1 \le i \le N$。
  - 如果待测点与任意一个用户轮廓的偏差都大于某个设定的阈值，则该X点所代表的数据为异常。说明X不符合任何一个用户轮廓。

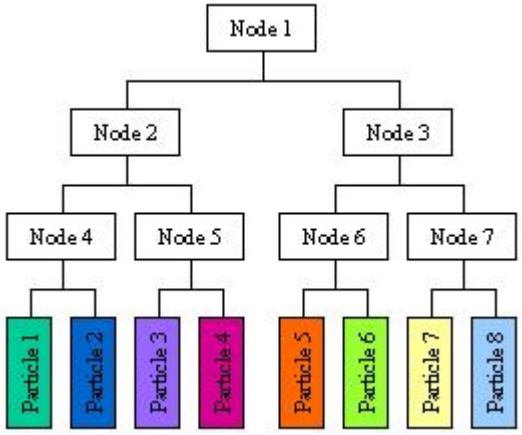
- 检测的关键
  - 怎样建立用户轮廓
  - 对于给定用户轮廓和待测的数据，怎样快速找到与待测数据最近的轮廓。
    - 因为不能将待测的数据与每个轮廓都进行计算偏差。

- KD树
  - kd树是做多维数据索引时候用到的一种数据结构：k-d树是二叉检索树的扩展，k-d树的每一层将空间分成两个。树的顶层结点按某一维进行划分，递归，每次划分均选择合适的维进行划分，各个维循环往复。划分要使得在每次划分后，大约一半的点落入一侧，而另一半落入另一侧。当一个结点中的点数少于给定的最大点数时，划分结束。
  - 检索效率O(lgn)

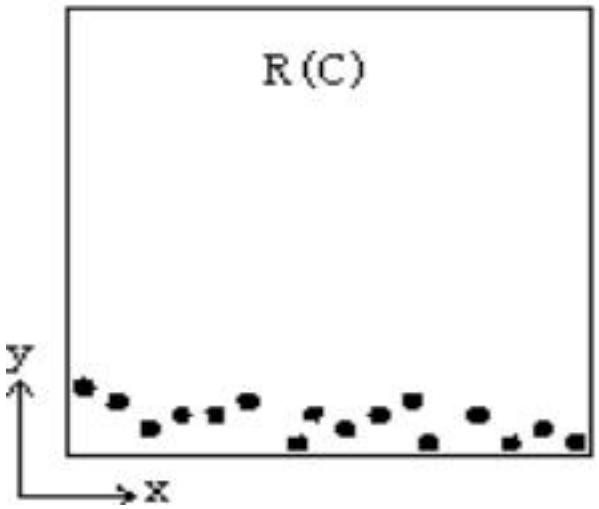
- KD树

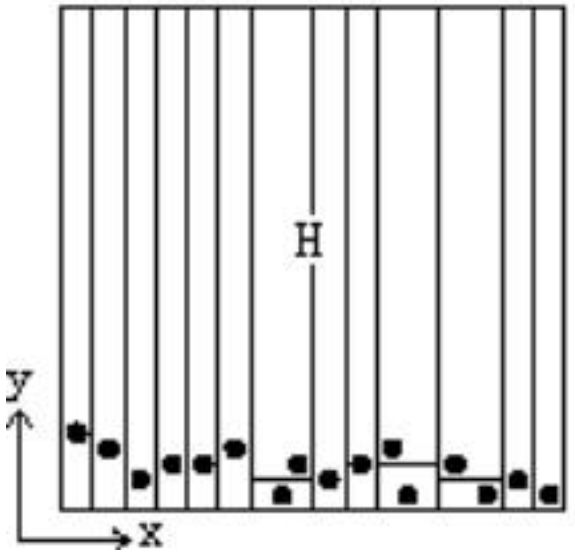

root


Node 2
Node 3
H1
H2
H3
Node 6
Node 7
Node 4
Node 5
Node 1


Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Particle 1
Particle 2
Particle 3
Particle 4
Particle 5
Particle 6
Particle 7
Particle 8

- 标准分割策略
  - 每次将点集中具有最大延展度的维定为分割维，将该维的中点定为分割点。然后继续递归分割，直到每个轮廓中包含的点的个数达到要求。
  - 优点：kd树的高度不超过log2n。缺点：可能使划分后得到的超矩形（轮廓）具有过高的长宽比。


R(C)
y
x


H
y
x

- 数据的规格化
  - 将有关属性数据按比例投射到特定小范围之中。是为防止在距离计算时某些取值范围过大的分量掩盖了其它分量的作用，使得算出的距离不能完全体现两个向量间的差异
- 最大最小规格化方法
  - 对被初始数据进行一种线性转换,保留原来数据中存在的关系。
  - 设minA 和 maxA 属性A的最大和最小值。
  - 最大最小规格化方法将属性A的一个值v映射为v’，且有 v’∈[new_minA,new_maxA]，具体映射计算公式如下：

$$v' = \frac{v - \min_A}{\max_A - \min_A}(new\_\max_A - new\_\min_A) + new\_\min_A$$

- 方法过程
  - 选择合适的体现网络行为的属性
  - 收集大量正常的网络样本，作为历史数据。
  - 统计、测量这些属性，属性数据的规格化
  - 采用相应的异常检测算法构建正常行为轮廓（异常检测模型）
    - 基于KD树的欧式距离算法
  - 使用该模型对新的数据进行检测

算法小结

- 统计的优点是算法与实现非常简单、可检测到未知的入侵；缺点是有误报、漏报率高，敏感性不强，难以进行更为复杂的检测；基于分布的方法局限在于必须事先知道数据的分布特征，多用于科研计算。
- 基于距离的算法跟基于统计的算法相比，不需要用户拥有任何领域知识。而且在概念上更加直观。更重要的是，距离异常更接近Hawkins的异常本质定义。

异常检测方法

- 基于分布（统计）的检测方法
- 基于距离的检测方法
- 基于密度的检测方法
- 基于聚类的检测方法

- 聚类

- 聚类技术将数据元组视为对象。它将对象划分为群或聚类，使得在一个聚类中的对象“类似”，但与其它聚类中的对象“不类似”。
- 绝大多数应用采用了以下两个比较流行的基于划分的方法，这些基于划分的聚类方法对在中小规模的数据库中发现球状簇很适用。
- （1）k-means算法，在该算法中，每个簇用该簇中对象的平均值来表示。
- （2）k-medoids算法，在该算法中，每个簇用接近聚类中心的一个对象来表示。

k-means算法如下所示：

begin initialize 样本数n，聚类数c，初始聚类中心m1, …, mc；

do 按照最近邻mi分类n个样本；

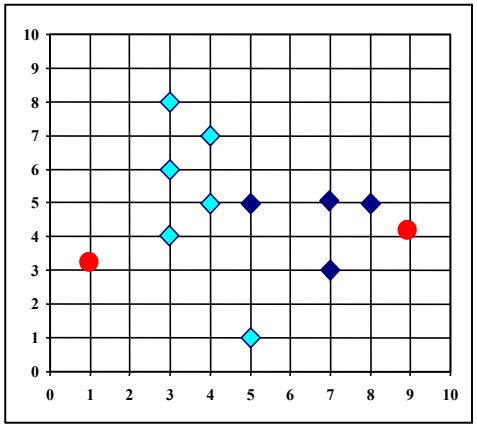
重新计算聚类中心m1, …, mc；

until mi不再改变；

return m1, …, mc;

end

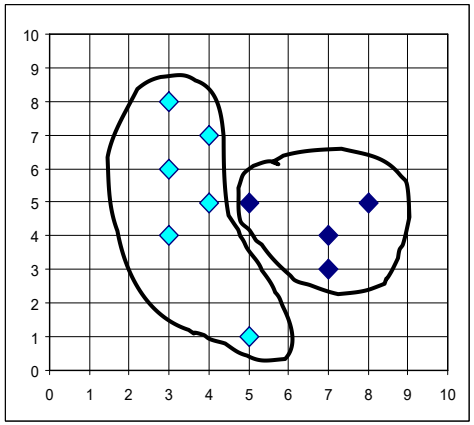
K-均值算法



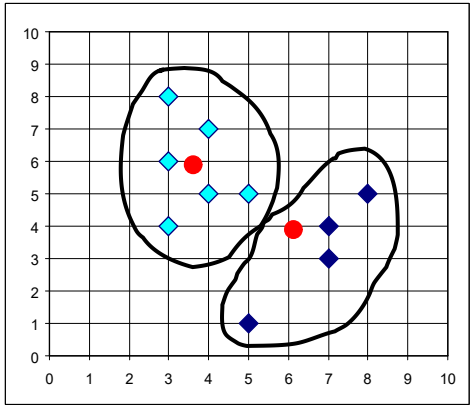
K=2

Arbitrarily choose K object as initial cluster center

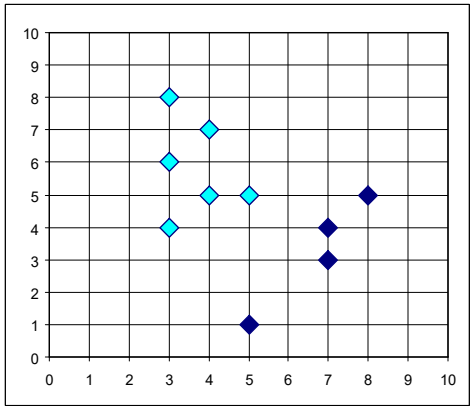
Assign each objects to most similar center



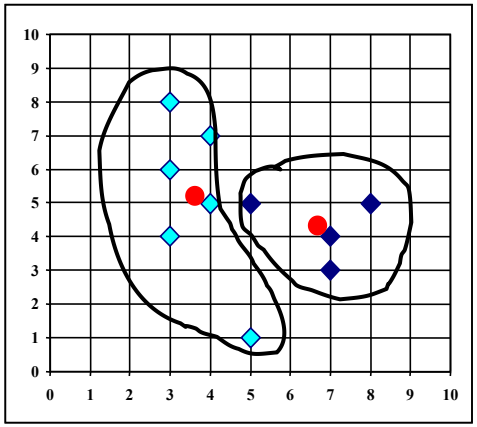
reassign



Update the cluster means



reassign



Update the cluster means

例:假设给定如下要进行聚类的元组:

{2，4，10，12，3，20，30，11，25}

假设要求的簇的数量为k=2。

应用K-means算法:

第一步:初始时用前两个数值作为簇的质心，这两个簇的质心是:ml=2;m2=4;

第二步:对剩余的每个对象，根据其与各个簇中心的距离，将它赋给最近的簇，可得:

K1=(2，3}:

K2={4，10，12，20，30，11，25}:

数值3与两个均值的距离相等，所以任意的选择凡作为其所属的簇。

第三步:计算新的质心:

ml=(2+3)/2=2.5;

m2=(4+10+12+20+30+11+25)/7=16;

重新对簇中的成员进行分配可得K_1={2，3，4}和K_2={10，12，20，30，11，25}，不断重复这个过程可得:

m_1	m_2	K_1	K_2
3	18	{2,3,4,10}	{12,20,30,11,25}
4.75	19.6	{2,3,4,10,11,12}	{20,30,25}
7	25	{2,3,4,10,11,12}	{20,30,25}

注意在最后两步中簇的成员是一致的。由于均值不再变化，所以均值已经收敛了。

因此该问题的答案为K_1={2,3,4,10,11,12}和K_2={20,30,25}

K-MEDOIDS 算法

- k-means算法缺点
  - 受脏数据影响较大，产生的分类不合理
  - 例如 {2,3,4,10,20,25,30,80}
  - k-means算法分类的结果: {2,3,4,10,20,25,30}{80}
- k-medoids算法
  - 中心点的选取:从当前分类中选取这样一个点——它到其他所有（当前分类中的）点的距离之和最小——作为中心点。
  - 再用这个点分别与所有点

k-medoids算法如下所示：

1）任意选取K个对象作为medoids（O1,O2,…Oi…Ok）。

2）将余下的对象分到各个类中去（根据与medoid点距离最近的原则）；

3）对于每个类（Oi）中，顺序选取一个Or，计算Or到类中每个点的距离之和 — E（Or）。选择E最小的那个Or来代替Oi。这样K个medoids就改变了。

4）重复2、3步直到K个medoids固定下来。

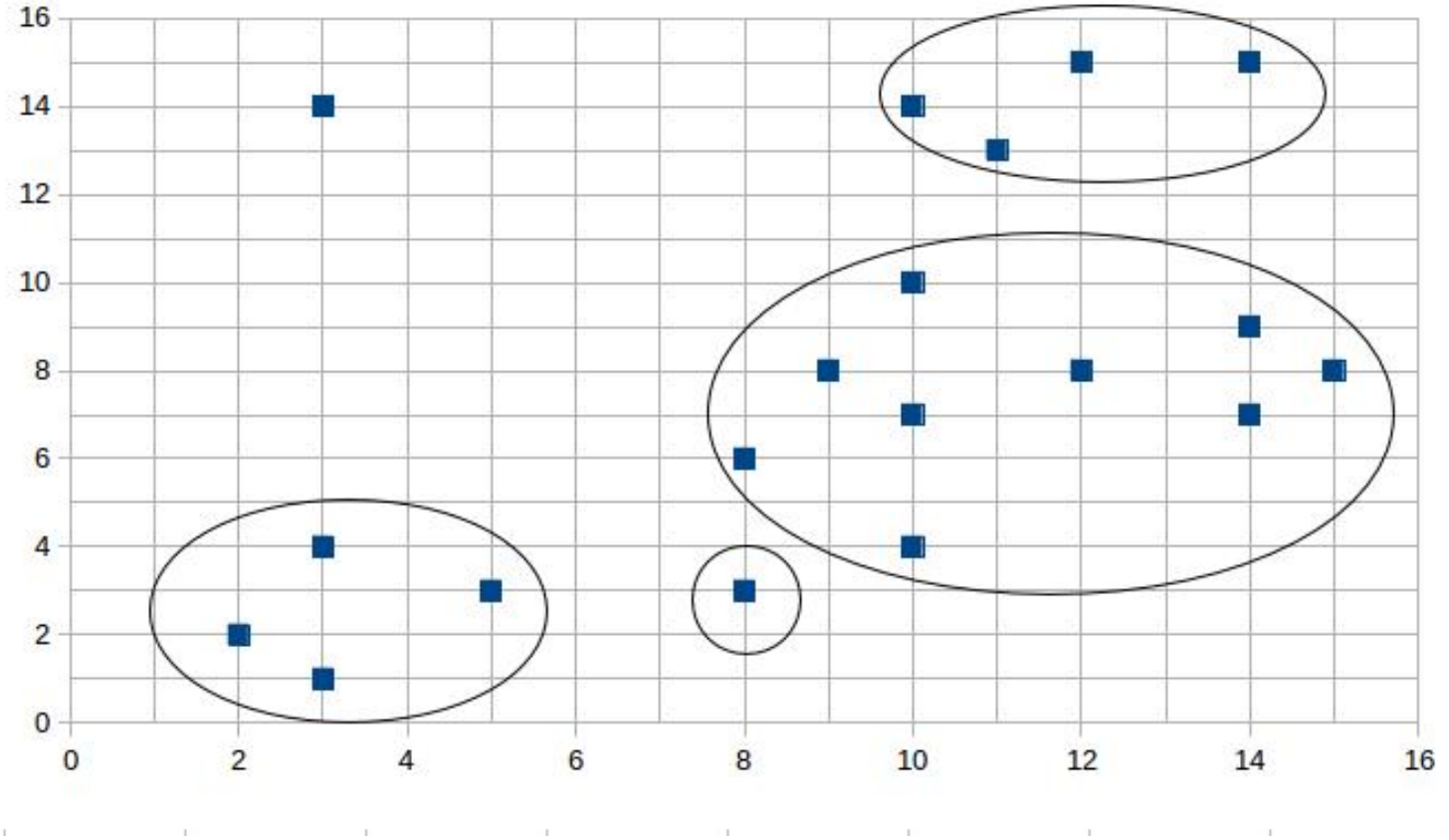
- k-medoids算法
  - 不容易受到那些由于误差之类的原因所产生的脏数据的影响
  - 计算量显然要比K-means要大
  - 一般只适合小数据量。

- 基于划分的聚类方法的限制
  - 数据聚类的形状
  - 需要指明聚类的簇数
  - 噪声点

异常检测方法

- 基于分布（统计）的检测方法
- 基于距离的检测方法
- 基于密度的检测方法
- 基于聚类的检测方法

- 基于密度的聚类方法
  - 密度通常用邻近度定义。一种常用的定义密度的方法是，定义密度为到k个最近邻的平均距离的倒数。如果该距离小，则密度高，反之亦然。
  - 给定半径内的点计数：一个对象周围的密度等于该对象指定距离d内对象的个数。
  - 需要小心地选择参数d。如果d太小，则许多正常点可能具有低密度，从而具有高离群点得分。如果d太大，则许多离群点可能具有与正常点类似的密度(和离群点得分)。


16
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16

- DBSCAN算法
  - DBSCAN是一种利用样本密度进行测量比较的聚类算法。算法认为若样本点在以它为圆心Eps距离为半径的区域内的少数类样本数大于阈值MinPts，那么该样本点即为核心点。核心点与其Eps范围内的邻居形成簇。如果簇间存在重合，则这些簇需要合并。

- 概念
  - **邻域**：给定对象半径内的区域称为该对象的邻域。
  - **核心对象（核心点）**：如果给定对象邻域内的样本点数大于等于MinPts，则称该对象为核心对象。**边界点、噪声点。**
  - **直接密度可达**：给定一个对象集合D，如果p在q的邻域内，且q是一个核心对象，则我们说对象p从对象q出发是直接密度可达的
  - **密度可达**：对于样本集合D，如果存在一个对象链 $p_1,p_2,\cdots,p_n$ ，$p_1=q$，$p_n=p$，对于$p_i \in D\ (1 \le i \le n)$，$p_{i+1}$是从$p_i$关于 MinPts直接密度可达，则对象p是从对象q关于MinPts密度可达的。
  - **密度相连**：如果存在对象o，使对象p和q都是从o关于和 MinPts密度可达的，那么对象p到q是关于和 MinPts密度相连的。
  - 密度可达是直接密度可达的传递闭包，只有核心对象之间相互密度可达。密度相连是对称关系，**DBSCAN目的是找到密度相连对象的最大集合。**

- **DBSCAN算法的聚类思想**
  - DBSCAN算法基于一个事实：**一个聚类可以由其中的任何核心对象唯一确定**。等价可以表述为：任一满足核心对象条件的数据对象p，数据库D中所有从**p密度可达**的数据对象o所组成的集合构成了一个完整的聚类C，且p属于C。
  - 算法的具体聚类过程如下：
    - 扫描整个数据集，找到任意一个核心点，对该核心点进行扩充。扩充的方法是寻找从该核心点出发的所有密度相连的数据点。遍历该核心点的邻域内的所有核心点（因为边界点是无法扩充的），寻找与这些数据点密度相连的点，直到没有可以扩充的数据点为止。最后聚类成的簇的边界节点都是非核心数据点
    - 之后就是重新扫描数据集（不包括之前寻找到的簇中的任何数据点），寻找没有被聚类的核心点，再重复上面的步骤，对该核心点进行扩充直到数据集中没有新的核心点为止。数据集中没有包含在任何簇中的数据点就构成异常点。

- **DBSCAN算法实现**
  - 采用迭代查找的方法，通过迭代地查找所有直接密度可达的对象，找到各个所有密度可达的对象，具体算法描述如下：
    - 1）检测数据库中尚未检查过的对象***p***，如果***p未被***处理(归为某个簇或者标记为噪声)，则检查其邻域，若包含的对象数不小于，建立新簇***C***，将中所有点加入***C***;
    - 2）对***C***中所有尚未被处理的对象***q***，检查其邻域，若中至少包含个对象，则将中未归入任何一个簇的对象加入***C***;
    - 3）重复步骤2)，继续检查***C***中未处理的对象，知道没有新的对象加入当前簇***C***;
    - 4）重复步骤1)~3)，直到所有对象都归入了某个簇或标记为噪声。

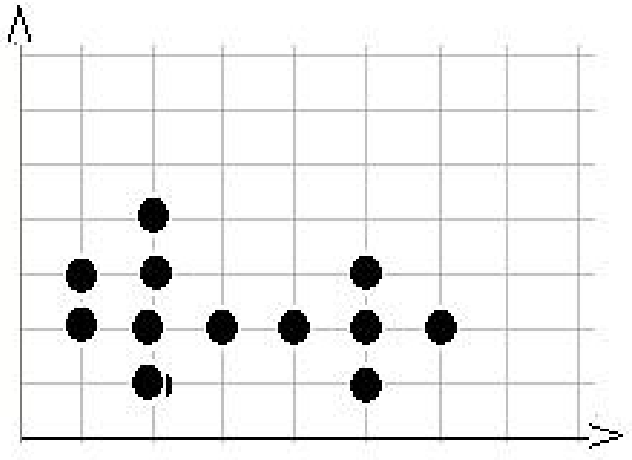
基于密度方法- DBSCAN

下面给出一个样本事务数据库（见下表），对它实施DBSCAN算法。

根据所给的数据通过对其进行DBSCAN算法，以下为算法的步骤（设n=12，用户输入ε=1，MinPts=4）

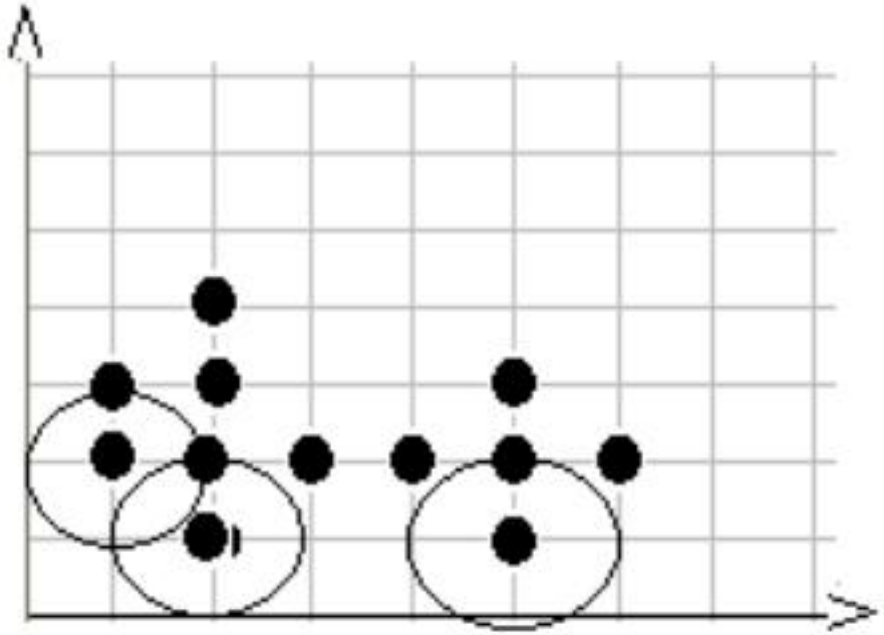
样本事务数据库

序号	属性 1	属性 2
1	2	1
2	5	1
3	1	2
4	2	2
5	3	2
6	4	2
7	5	2
8	6	2
9	1	3
10	2	3
11	5	3
12	2	4



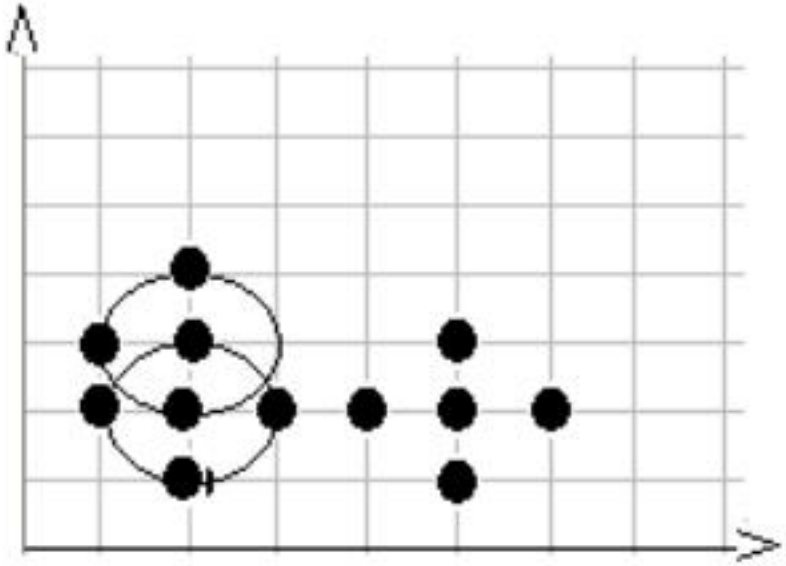
DBSCAN聚类过程

- 第1步，在数据库中选择一点1，由于在以它为圆心的，以1为半径的圆内包含2个点（小于4），因此它不是核心点，选择下一个点。
- 第2步，在数据库中选择一点2，由于在以它为圆心的，以1为半径的圆内包含2个点，因此它不是核心点，选择下一个点。
- 第3步，在数据库中选择一点3，由于在以它为圆心的，以1为半径的圆内包含3个点，因此它不是核心点，选择下一个点。



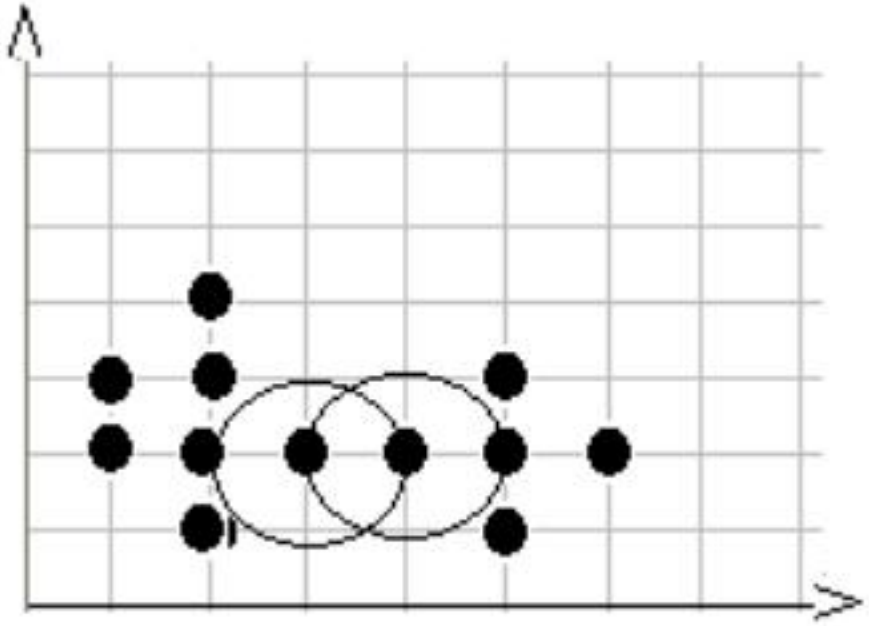
DBSCAN聚类过程

- 第4步，在数据库中选择一点4，由于在以它为圆心的，以1为半径的圆内包含5个点，因此它是核心点，寻找从它出发可达的点（直接可达4个，间接可达3个），聚出的新类{1，3，4，5，9，10，12}，选择下一个点。



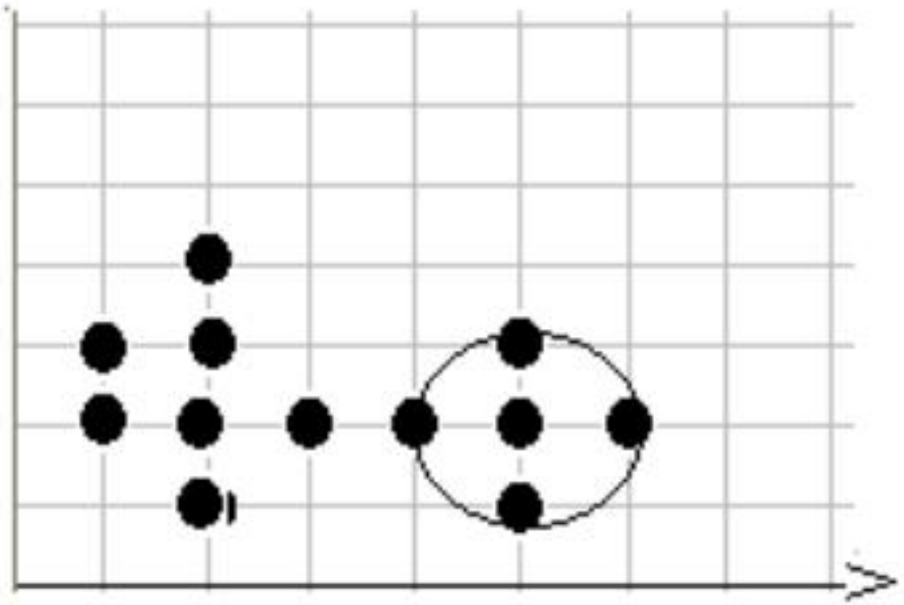
DBSCAN聚类过程

- 第5步，在数据库中选择一点5，已经在簇1中，选择下一个点。
- 第6步，在数据库中选择一点6，由于在以它为圆心的，以1为半径的圆内包含3个点，因此它不是核心点，选择下一个点。



DBSCAN聚类过程

- 第7步，在数据库中选择一点7，由于在以它为圆心的，以1为半径的圆内包含5个点，因此它是核心点，寻找从它出发可达的点，聚出的新类{2，6，7，8，11}，选择下一个点。



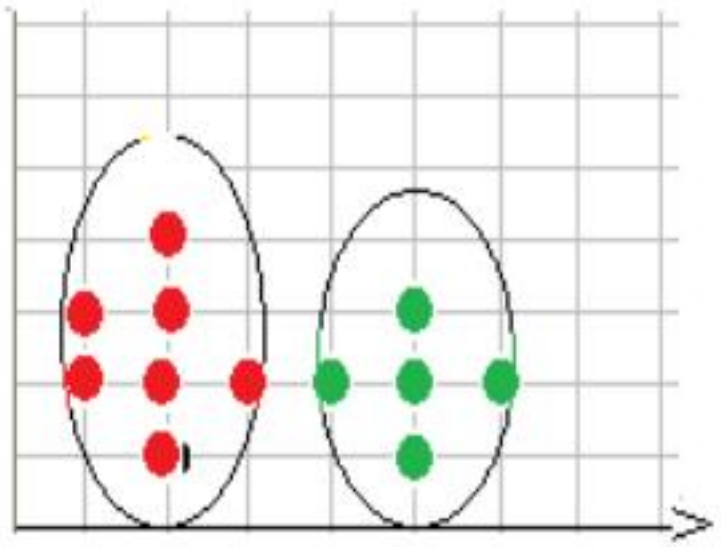
DBSCAN聚类过程

- 第8步，在数据库中选择一点8，已经在簇2中，选择下一个点。
- 第9步，在数据库中选择一点9，已经在簇1中，选择下一个点。
- 第10步，在数据库中选择一点10，已经在簇1中，选择下一个点。
- 第11步，在数据库中选择一点11，已经在簇2中，选择下一个点。
- 第12步，选择12点，已经在簇1中，由于这已经是最后一点所有点都以处理，程序终止。

基于密度方法的聚类- DBSCAN

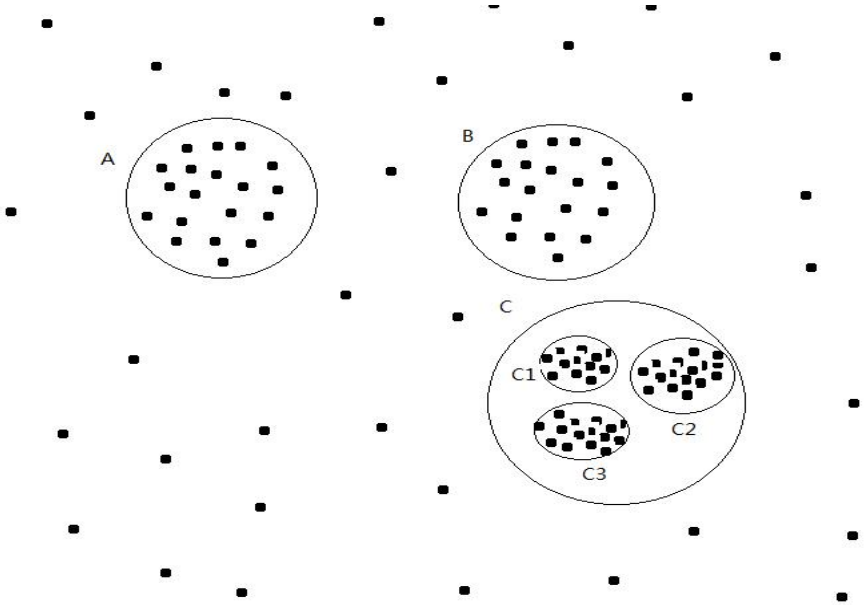
算法执行过程：

步骤	选择的点	在ε中点的个数	通过计算可达点而找到的新簇
1	1	2	无
2	2	2	无
3	3	3	无
4	4	5	簇C_1：{1，3，4，5，9，10，12}
5	5	3	已在一个簇C_1中
6	6	3	无
7	7	5	簇C_2：{2，6，7，8，11}
8	8	2	已在一个簇C_2中
9	9	3	已在一个簇C_1中
10	10	4	已在一个簇C_1中，
11	11	2	已在一个簇C_2中
12	12	2	已在一个簇C_1中



- 算法小结
  - DBSCAN是一种基于密度的空间聚类算法。该算法将具有足够密度的区域划分为簇，并在具有噪声的空间数据库中发现任意形状的簇。
  - 优点：与K-MEANS比较起来，不需要输入要划分的聚类个数；聚类速度快且能够有效处理噪声点和发现任意形状的空间聚类。
  - 弱点：当数据量增大时，要求较大的内存支持；当空间聚类的密度不均匀、聚类间距差相差很大时，聚类质量不够好。

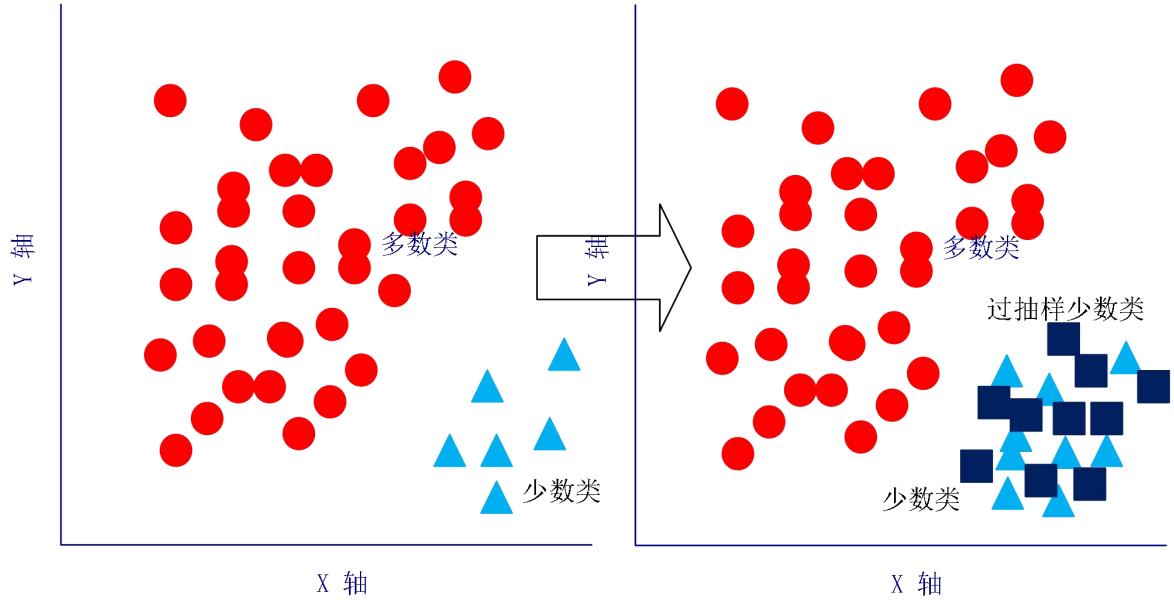
下图中所描述的数据集不能通过一个全局密度参数同时区分出簇A 、B 、C、C1、C2和C3，只能得到A 、B 、C或C1、C2和C3，对于C1、C2和C3而言A 、B 、C都是噪声。



对于固定的MinPts 值，和两个 *ε1＜ε2，关于ε1的*MinPts簇C一定是关于*ε2和*MinPts簇C’的子集，这就意味着。如果两个对象在同一个基于密度的簇中，则它们也是在同一个具有较低密度要求的簇中。

- 样本不平衡问题
  - 噪声属于小样本集合
  - **过采样**方法
    - **随机过采样：**从样本少的类别中随机抽样，再将抽样得来的样本添加到数据集中。然而这种方法如今已经不大使用了，因为重复采样往往会导致严重的过拟合。
    - 人工合成一些少数类样本：在少数类样本之间进行插值来产生额外的样本。

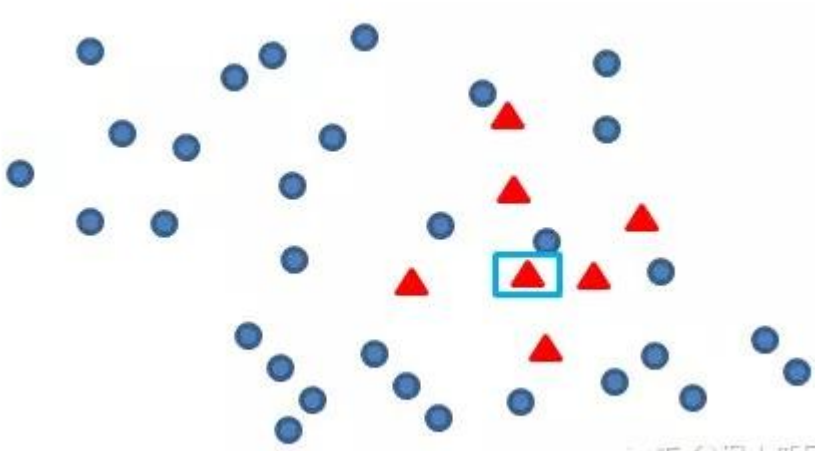
- SMOTE算法
  - 通过在寻找每个少数类样本的*k*最近邻并在它与这*k*个邻近样本间随机线性生成新的样本，使小类样本数量增多，样本集合趋于平衡，改善流量数据集的分类性能

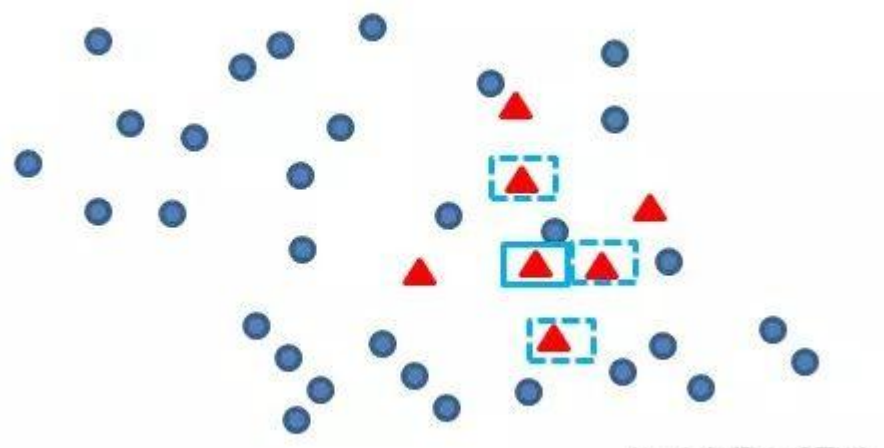


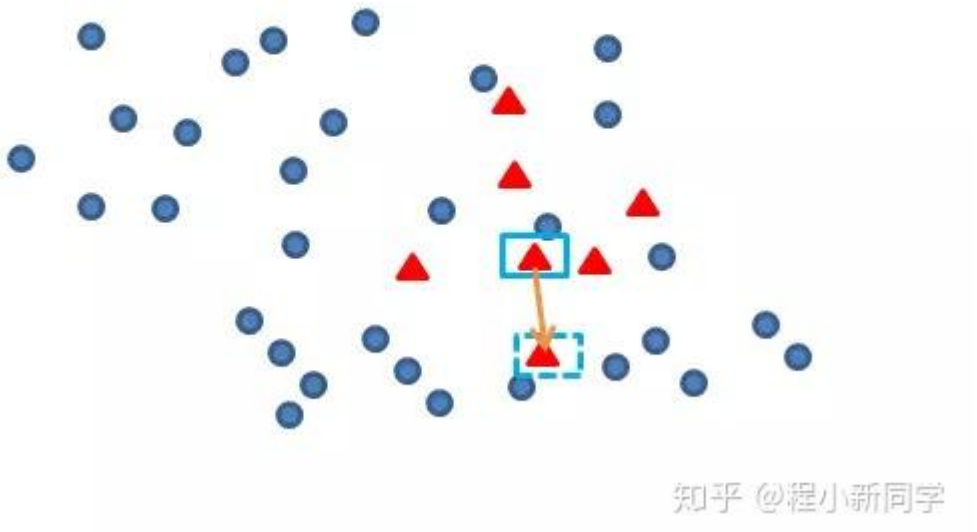
- SMOTE算法
  - 在所有的少数类样本中，对其中的每个样本*x*算出其到其它少数类样本的距离，获得其*k*个最近邻。
  - 根据多数类与少数类的倍率来设置过采样的倍率*n*，并从*x*的*k*个最近邻样本中选择合适的个数（*Y*）。$y \in Y$，在*x*与*y*之间进行随机线性插值。
  - 对每一个插值，按如下公式构造新的少数类样本：

$$x_{new} = x + rand\ (0,1) \times (y - x)$$

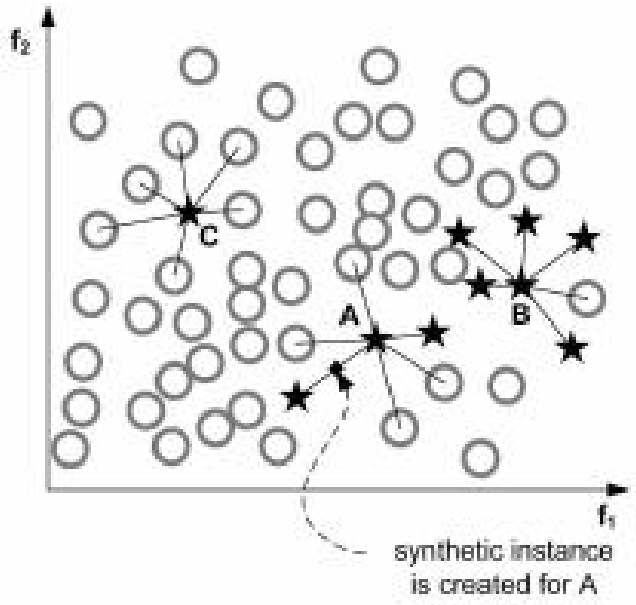
  - 将新构造的样本加入样本集，再利用分类器对其进行学习。


知乎 @程小新同学


知乎 @程小新同学


知乎 @程小新同学

- Border-line SMOTE算法
  - "safe"： 超过一半的k近邻样本属于少数类
  - "danger" ： 超过一半的k近邻样本属于多数类
  - "noise" ： 所有的k近邻个样本都属于多数类



Consider 6-nearest neighbor: m=6

For A: number of minority instance: 2
number of majority instance: 4 → "DANGER" instance

For B: number of minority instance: 5
number of majority instance: 1 → "SAFE" instance

For C: number of minority instance: 6
number of majority instance: 0 → "NOISE" instance

- Border-line SMOTE算法
  - 算法只会从处于”**danger**“状态的样本中随机选择，然后用SMOTE算法产生新的样本。
  - 处于”**danger**“状态的样本代表靠近”边界“附近的少数类样本，而处于边界附近的样本往往更容易被误分类。
  - Border-line SMOTE 只对那些靠近”边界“的少数类样本进行人工合成样本。

软件安全

主讲人：余翔湛
yxz@hit.edu.cn

课程内容

1. 软件安全需求
2. 软件安全面临的威胁
3. 软件安全开发
4. 恶意软件防范
5. 程序安全性测试

4. 恶意软件防范

- 恶意软件结构
- 基于特征检测的恶意软件识别
- 基于异常检测的恶意软件识别
- 恶意软件攻击遏制

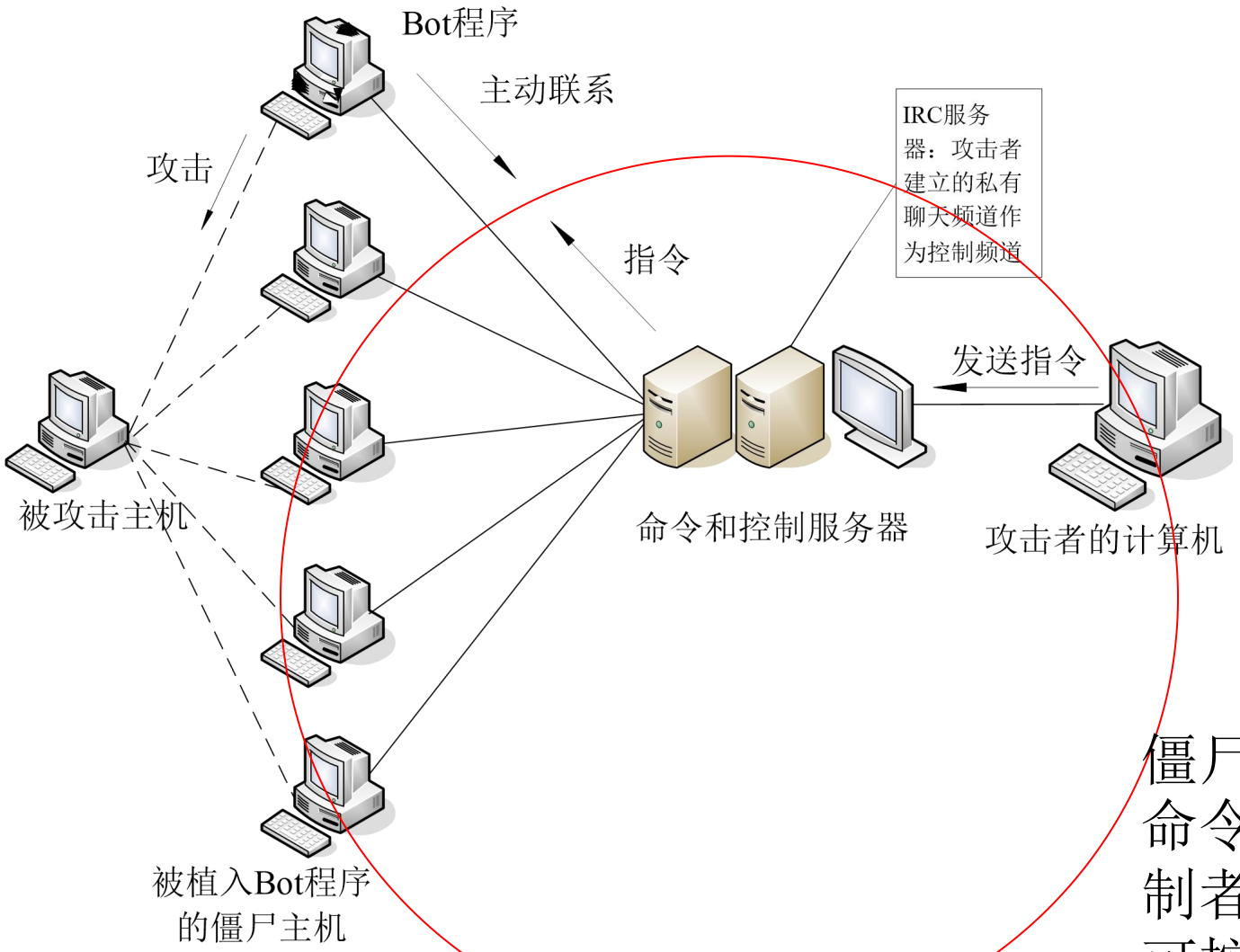
恶意软件攻击遏制

网络安全事件应急响应

僵尸网络（Botnet）

- 定义
  - 僵尸网络是可被攻击者远程控制的被攻陷主机所组成的网络。
  - CNCERT/CC的定义：僵尸网络指的是攻击者利用互联网秘密建立的可以集中控制的计算机群。其组成通常包括被植入“僵尸”程序的计算机群，一个或多个控制服务器，控制者的控制终端等。
  - 僵尸网络是攻击者处于恶意目标，传播僵尸程序将大量主机感染成僵尸主机，并通过一对多的命令和控制信道所组成的网络。

• 僵尸网络典型结构



僵尸网络:是由Bot、命令控制服务器和控制者组成的可通信、可控制的网络。

- 僵尸网络的特点：
  - 僵尸网络是一个可控的网络，这个网络并不是物理意义上具有网络拓扑结构的网络，而是具有一定分布性，随着Bot程序不断被传播有新的计算机加入的一个动态变化的网络。
  - 僵尸网络是通过一定的传播手段形成的，如主动的漏洞攻击、电子邮件病毒和蠕虫等传播手段。
  - 僵尸网络可以执行一对多的控制关系，发起恶意行为。比如说，指挥bots对某一个目标站点发起DDos攻击。
  - 隐蔽性

正是这种一对多的控制关系，使得攻击者可以极低的代价高效地控制大量的资源为其服务。这也是僵尸网络近年来受到黑客欢迎的根本原因。和普通的病毒和蠕虫不同的是，僵尸网络充当的是一个攻击的平台，比普通的病毒和蠕虫具有更大的破坏性。

僵尸网络生命周期

- 僵尸网络的产生
  - 蠕虫创建、木马创建和僵尸网络创建。
- 僵尸网络的等待
  - Bot程序定期连接到控制服务器，直到接收到特定命令后才会有所活动。
- 僵尸网络的扩散
  - 一般的Bot程序本身没有扩散行为。而是利用木马或者蠕虫来传播。
- 僵尸网络的迁移
  - 控制服务器可以使用新的域名或IP地址的方式实现物理转移，也可以使用动态DNS技术，每次使用不用的IP地址。

- 僵尸网络的升级
  - Bot接收到更新命令后提取参数到特定网站去更新下载程序。
- 僵尸网络的攻击
  - Bot程序接收到控制者的攻击命令，进行DDoS攻击或者扫描监视控制被害主机。
- 僵尸网络的消亡
  - 僵尸网络可能消亡的因素主要有：控制服务器不可解析或无法连接；大量Bot客户端被删除；僵尸网络间彼此争夺控制权；控制者主动放弃控制网络。

- **僵尸网络控制机制**
  - 基于IRC协议的命令与控制机制
  - 基于HTTP协议的命令与控制机制
  - 以及基于P2P协议的命令与控制机制

- **基于IRC协议的僵尸网络命令与控制机制**
  - IRC服务器为控制者所建
  - IRC客户端连接IRC服务器，然后发送Mick Nickname，加入一个指定的频道
  - 控制者可设置频道主题(TOPIC)命令，当僵尸程序登录到频道后立即接收并执行这条频道主题命令；
  - Bot程序在连接到控制服务器加入特定频道后会一直保持在线状态，用PING、PONG命令维持连接，直到接收到特定命令后才会有所活动。
  - 使用频道发送NOTICE或PRIVMSG消息，即通过IRC协议的群聊和私聊方式向频道内所有僵尸程序或指定僵尸程序发布命令；

- **基于HTTP协议的命令与控制机制**
  - 控制服务器是一个web server。Bot定期访问其URL，获得相关指令。
  - 优点：
    - IRC协议已经是僵尸网络主流控制协议，安全业界非常关注监测IRC通信以检测其中隐藏的僵尸网络活动，使用HTTP协议构建控制信道则可以让僵尸网络控制流量淹没在大量的因特网Web通信中，从而使得基于HTTP协议的僵尸网络活动更难以被检测；
    - 大多数组织机构在网关上部署了防火墙，在很多情况下，防火墙过滤掉了非期望端口上的网络通信，IRC协议使用的端口通常也会被过滤，而使用HTTP协议构建控制信道一般都可以绕过防火墙。

基于IRC协议和HTTP协议的命令与控制机制均具有集中控制点，这使得这种基于客户端-服务器架构的僵尸网络容易被跟踪、检测和反制，一旦防御者获得僵尸程序，他们就能很容易地发现僵尸网络控制器的位置，并使用监测和跟踪手段掌握僵尸网络的全局信息，通过关闭这些集中的僵尸网络控制器也能够比较容易地消除僵尸网络所带来的威胁。为了让僵尸网络更具韧性和隐蔽性，一些新出现的僵尸程序开始使用P2P协议构建其命令与控制机制。

- **基于P2P协议的命令与控制机制**
  - 新兴的僵尸网络控制机制
  - 采用P2P协议构建命令与控制机制
    - 第一个构建P2P控制信道的僵尸网络恶意代码在网络传播过程中对每个受感染主机都建立了一个完整的已感染节点列表
    - 通过建立种子主机列表

- 僵尸网络检测方法
  - 基于蜜罐蜜网技术
    - 蜜罐技术，捕获Bot程序，发现控制服务器
  - 基于终端信息检测
    - 防病毒软件
  - 基于流量特征检测
    - IRC流量检测区分正常IRC和非正常IRC
  - 基于DNS域名数据的检测

僵尸网络检测技术

- 基于IRC协议的僵尸网络检测
  - 通过监视IRC协议的标准端口(TCP 6667)来检测僵尸网络
  - 基于僵尸终端和僵尸频道昵称评估的检测
    - Nickname是Bot程序登录IRC服务器的唯一身份标示。需由Bot程序自动生成，需要一定的机制产生。如固定串+特殊串+随机串？ 那么就会具有一定的特征。

- 基于命令序列相似性的检测方法
  - 僵尸程序登录IRC服务器后，命令序列几乎是一致的：登录、发送Nickname、加入频道、修改自身模式和频道模式、接受命令、发送PING/PONG命令维持连接
  - IRC僵尸网络程序的命令序列相似度一定很高。
  - 比较2个IRC用户的命令序列可知其是否是僵尸程序

- web服务器的命令序列相似性分析的检测
- 基于网络流异常挖掘的僵尸网络检测
  - 网络中部分终端动作在某特殊服务端口动作相似并与其他大部分终端动作明显区别。

基于域名的僵尸网络检测与控制

- 僵尸网络的DNS查询特征
  - 发出对僵尸网络控制服务器的DNS查询的IP地址和数目是固定的，而正常的服务器的域名通常是被一些随机的，匿名的主机查询。
  - 僵尸网络的僵尸主机经常会同时发出请求，同时从一个僵尸网络控制服务器跳转到另一个僵尸网络控制服务器。因此，当僵尸网络活动时，会产生一个瞬时的DNS查询数据流。而正常的DNS查询数据流通常是连续的且不可能同时产生。
  - 僵尸网络经常会使用DDNS

什么是DDNS？DDNS（Dynamic Domain Name Server）是动态域名服务的缩写。DDNS是将用户的动态IP地址映射到一个固定的域名解析服务上，用户每次连接网络的时候客户端程序就会通过信息传递把该主机的动态IP地址传送给位于服务商主机上的服务器程序，服务程序负责提供DNS服务并实现动态域名解析。就是说DDNS捕获用户每次变化的IP地址，然后将其与域名相对应，这样其他上网用户就可以通过域名来进行交流了。 了 动态域名服务的对象是指IP是动态的，是变动的。普通的DNS都是基于静态IP的，有可能是一对多或多对多，但IP都是固定的一个或多个。但DDNS的IP是变动的、随机的。

DDNS在监控行业中的应用？

目前ISP大多为我们提供动态IP（如ADSL拨号上网），而很多网络视频服务器和网络摄像机通过远程访问时需要一个固定的IP，而固定IP的费用很难让客户接受。所以DDNS为大家提出了一种全新的解决方案，它可以捕获用户每次变化的IP，然后将其与域名相对应，这样客户就可以通过域名来进行远程监控了。

因这各家公司的产品和实力的不同，DDNS的解决方案也不一样

1、路由器外挂 具体的说路由器外挂就是采用集成DDNS的路由器，通过申请其域名和服务，把申请所得用户名密码填入路由器DDNS模块相关项，再由路由器上作端口映射指向所需访问的监控设备即可，远程监控端通过访问域名即可访问到当前路由器，根据不同的端口来判断并指向所需访问的监控设备。

2、集成DDNS的监控设备 对于无人值守或不方便外挂路由器的状况下，视频监控也可采用集成DDNS的网络摄像机，同样把申请DDNS服务得到的用户名密码填入相关项，通过一条ADSL等宽带线路直接相连。远程监控端通过域名直接访问。

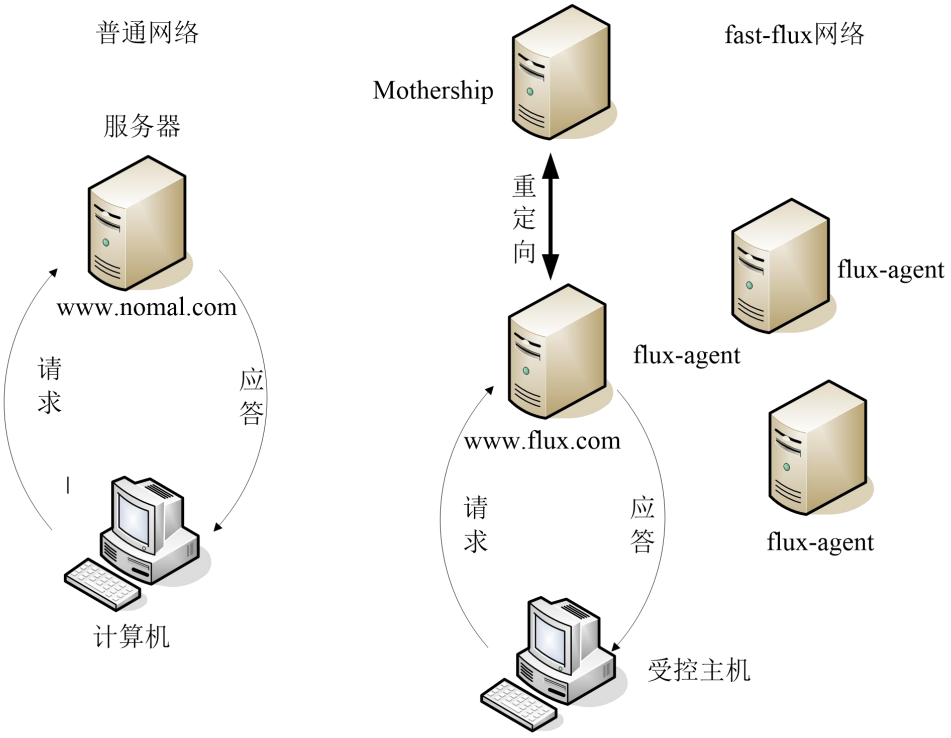
3、运行DDNS客户端软件 在局域网内部的任一PC或服务器上运行到DDNS客户端，此时域名解析到的IP地址是局域网网关出口处的公网IP地址，再在网关处作端口映射指向监控设备即可。

- 僵尸网络DNS查询检测算法
  - 计算僵尸主机的群体DNS查询的相似度。
  - 在t1和t2时间段内，分别建立查询相同域名的两个IP列表，将这两个IP列表定为A和B，计算两个列表中的IP地址集合的相似度。与阈值相比较，确定是否很相似，可怀疑是否是僵尸网络。
- 跃迁僵尸网络检测算法
  - 在僵尸网络跃迁期间，僵尸主机使用两台不同的控制服务器。因此，通过对比不同域名的大小相似的IP列表来确定。

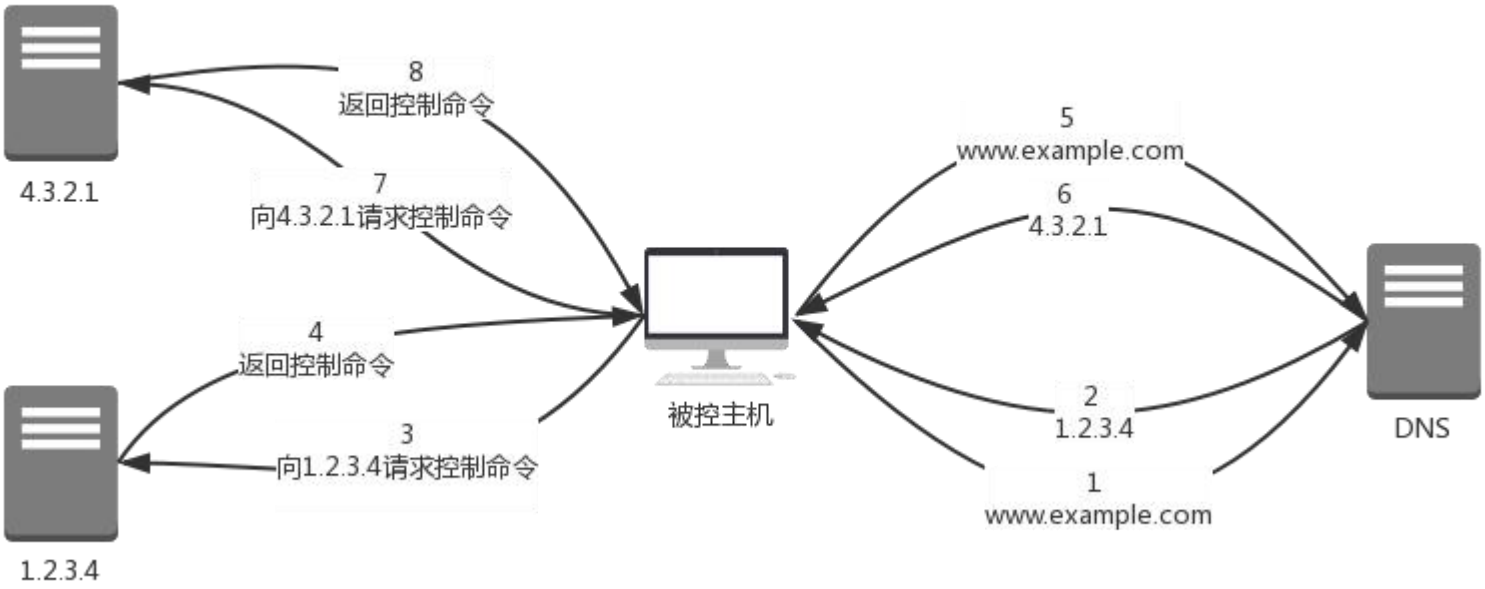
- 僵尸网络主动防御
  - 僵尸网络追踪：蜜罐、蜜网
  - ISP控制
  - DNS控制
  - 路由控制
- 控制的关键在阻断控制服务器与僵尸程序的联系

在正常的DNS服务器中，用户对同一个域名做DNS查询，在较长的一段时间内，无论查询多少次返回的结果基本上是不会改变的。Fast-flux技术是指不断改变域名和IP地址映射关系的一种技术，也就是说在短时间内查询使用Fast-flux技术部署的域名，会得到不同的结果。

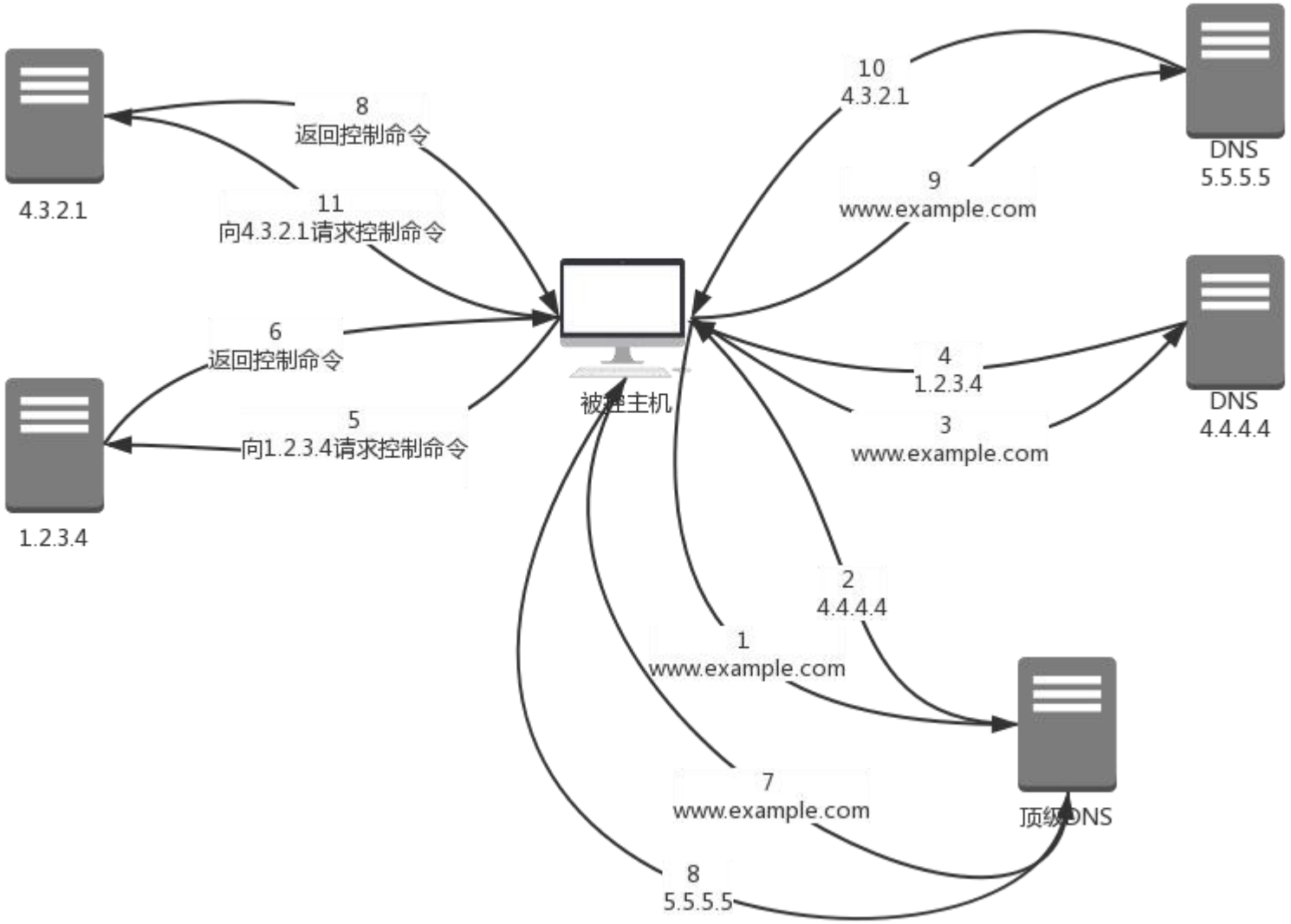
- 基于fast-flux技术的僵尸网络



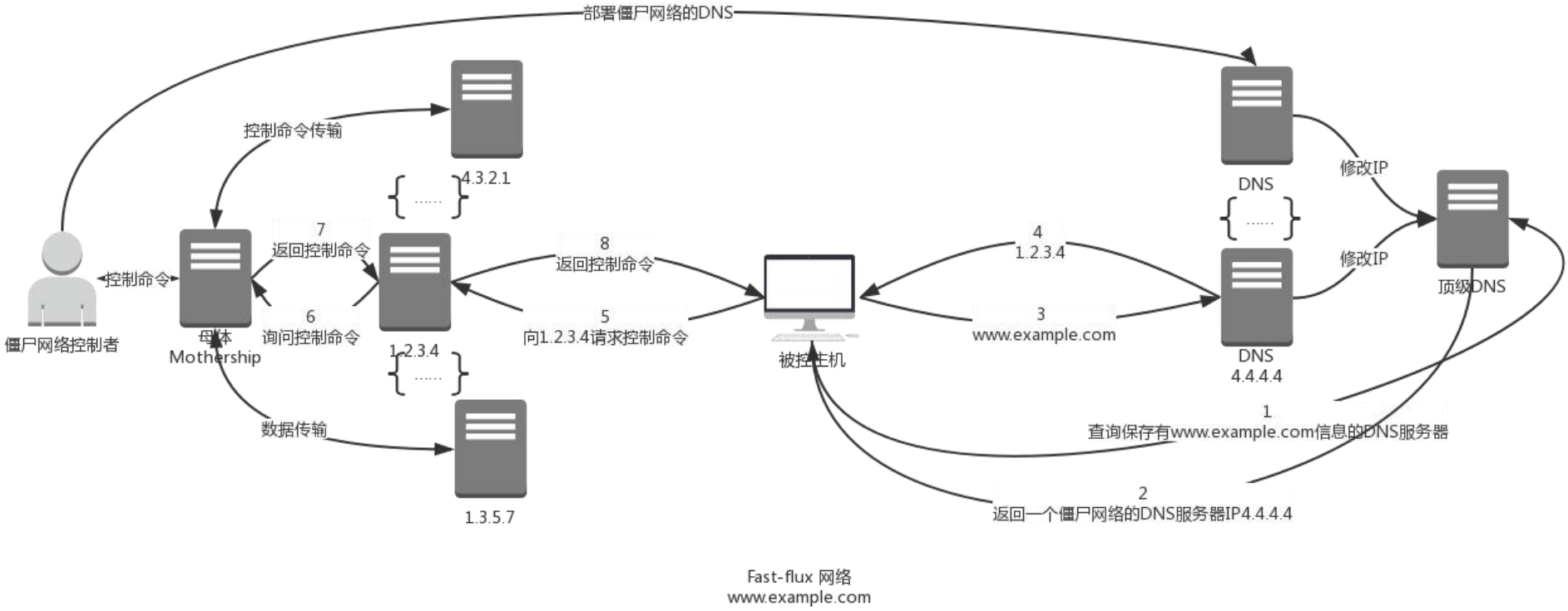
- Single-Flux


8
返回控制命令
4.3.2.1
7
向4.3.2.1请求控制命令
5
www.example.com
6
4.3.2.1
4
返回控制命令
被控主机
2
1.2.3.4
DNS
3
向1.2.3.4请求控制命令
1
www.example.com
1.2.3.4

- Double-Flux


4.3.2.1
8
返回控制命令
11
向4.3.2.1请求控制命令
10
4.3.2.1
DNS
5.5.5.5
9
www.example.com
6
返回控制命令
5
向1.2.3.4请求控制命令
被控主机
4
1.2.3.4
3
www.example.com
DNS
4.4.4.4
1.2.3.4
2
4.4.4.4
1
www.example.com
7
www.example.com
顶级DNS
8
5.5.5.5

- Double-Flux流程图



Fast-flux 网络
www.example.com

- Fast-flux技术优势
  - Fast-flux技术是合法的，在正常的运营中扮演调节服务器压力的角色，是每个网站都必备的功能。检测非法的Fast-flux较为困难。
  - 僵尸网络中，仅仅需要少量的C&C服务器---高性能母体（mothership）主机，用于提供C&C服务。而僵尸主机的数据请求连接的是Fast-flux网络，隐藏了实际的C&C服务器。在追踪僵尸网络时，通常只能追踪到Fast-flux中的部分节点，在这些节点中没有真正的控制命令，这使得对僵尸网络的调查变得更加困难。
  - Fast-flux延长了C&C服务器的生命期，由于多层的跳板，需要更长的时间用于识别和关闭C&C服务器。

- Fast-flux僵尸网络检测：异常检测方法

分类	标号	描述
域名	F1	域名时间
网络可达性	F2	A记录的IP个数
	F3	生存时间
代理的分布	F4	国家分布数
	F5	自治域分布数
	F6	组织的分布数

- 基于P2P协议的僵尸网络的检测
  - 是个难点
  - 蜜罐捕获样本分析是目前唯一有效的方法

- 基于p2p的僵尸网络
  - 僵尸网络传播时携带地址列表
  - 构建种子文件，通过DHT协议查找Bot程序
- 行为特点
  - 僵尸网络的不稳定导致TCP连接成功率低于正常P2P节点
  - 检测目前只能是一事一议

网络蠕虫

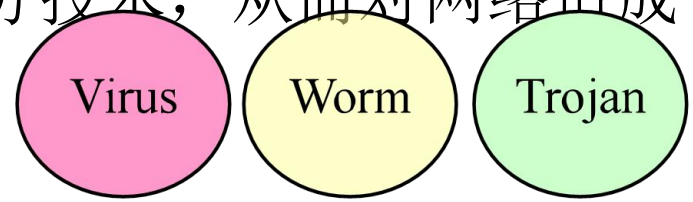
- 定义
  - 计算机蠕虫可以独立运行，并能把自身的传播到另外的计算机上。
  - 蠕虫是一个程序，它进入计算机网络，利用空闲的处理器去测定网络中的计算机跨度。蠕虫程序由许多段构成，在其主段的控制下，蠕虫的某个段运行在单独的计算机上。蠕虫典型的传播方式是依靠网络的漏洞，利用网络或电子邮件方式由一台计算机传播到另一台计算机，靠将自身向其他计算机提交来实现再生，并不将自身寄生在另一个程序上。
- 两个基本特征
  - 可以从一台计算机移动到另一台计算机上
  - 可以自我复制

- 蠕虫这个词最早由Shoch和Hupp在1982年提出，他们用这个词来形容一个可以在局域网络自动传播并维护网络中主机的良性程序
- 本来蠕虫是作为分散式计算领域中研究的一部分而被编写的，没有破坏安全的意图，也不隐藏其出现或运作。一般而言，蠕虫本身并不被当作传统的计算机病毒。
- 现在，蠕虫被病毒的制造者们加以利用，很多带有蠕虫性质的计算机病毒被制造出来，它们实际上是蠕虫和病毒的混合体，既有蠕虫的在网络上繁殖的功能，又有病毒的寄生和破坏的功能。

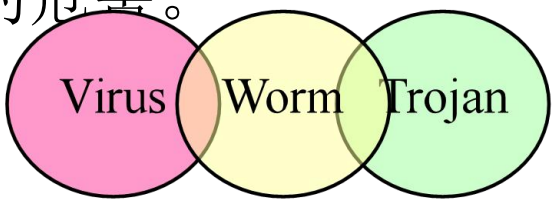
  目前在流行的网络恶意软件，都具有蠕虫的某些性质。

病毒、蠕虫和木马程序的区别与联系

- 计算机病毒、网络蠕虫、木马程序都具有传染性和复制能力。这两个主要特性上的一致，导致三者之间是非常难区分的。
- 但一个明显的不同是病毒和木马在网络中的的扩散速度远远小于蠕虫。
- 所以近年来，越来越多的病毒和木马程序结合了蠕虫的部分技术，从而对网络造成了更严重的危害。



（a）过去式



（b）现在式

- 蠕虫的行为特点
  - 主动攻击
  - 行踪隐蔽
  - 利用系统和服务的漏洞
  - 造成网络拥塞
  - 降低系统性能
  - 产生安全隐患
  - 反复性
  - 破坏性

蠕虫行为模式分类

- 蠕虫行为模式一般包括：扫描行为、攻击行为、命令控制行为、文件传输行为和激活行为
- **扫描行为(Scan):**蠕虫发出一系列数据报文来扫描目标主机是否在线，或者是否开启相应服务，或者是否存在漏洞，这样的一系列事件被称为蠕虫的扫描行为。扫描的顺序一般也是蠕虫复制、传播的顺序过程。
- **攻击行为(Attack):**蠕虫利用目标主机服务存在的漏洞，以进入目标主机并获得一定权限为目的的一系列事件被称为蠕虫的攻击行为。
- **命令控制行为(Control):**蠕虫利用攻击目标主机漏洞的结果和目标主机建立控制连接，从而可以在目标主机的shell环境下执行命令。蠕虫建立控制连接、向目标主机下达命令的过程被称为蠕虫的命令控制行为。

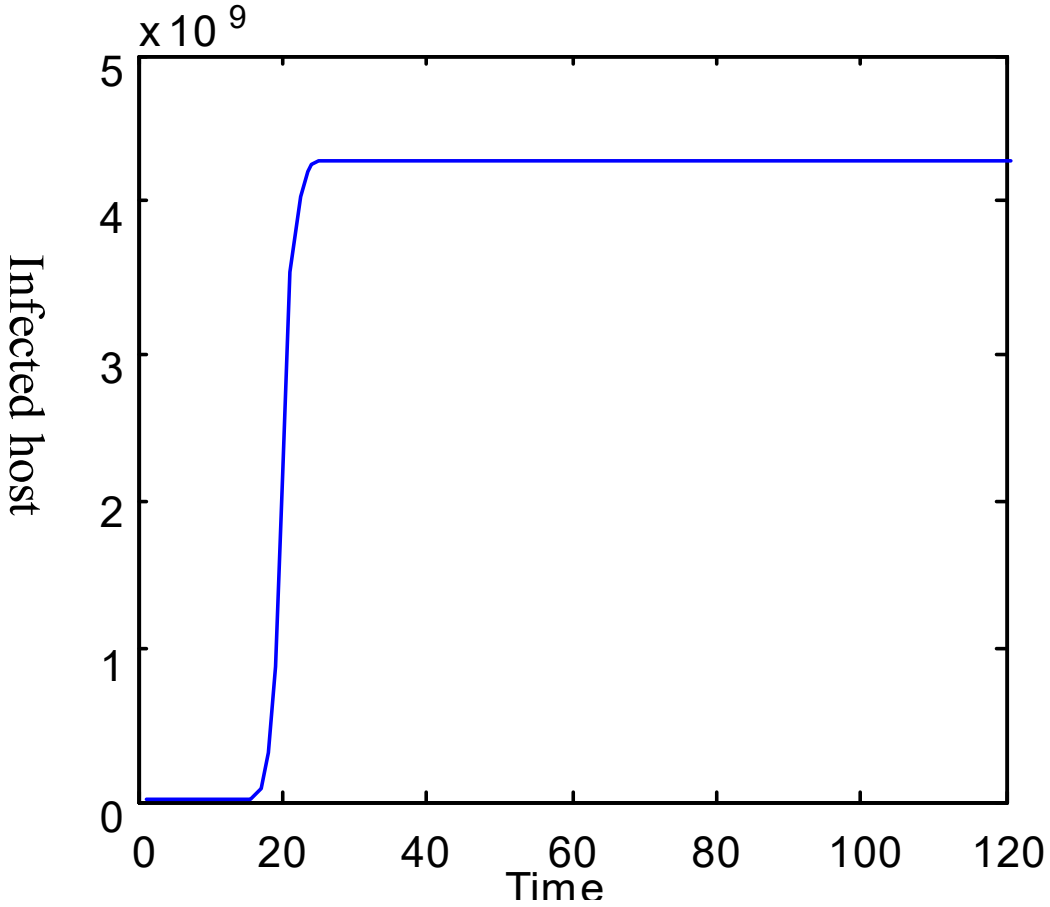
- **文件传输行为(Transmit)：**蠕虫利用TFTP等传输方式将蠕虫副本、攻击工具等传到目标主机上的过程称为文件传输行为。从传输协议的角度来看，文件传输行为可以是一个单独的传输连接。需要说明的是，蠕虫副本可以以文件传输的方式进行，也可以嵌入到攻击行内进行传输——即攻击成功完成的同时蠕虫副本传输也成功完成。比如Blaster利用TFTP向目标主机传输蠕虫副本，蠕虫CodeRed的蠕虫副本和攻击行为结合在一起进行传播。
- **激活行为(Provoke)：**是指目标主机已经获得蠕虫副本，攻击者通过命令控制或者攻击等方式激活目标主机上的蠕虫的过程称为激活行为。比如蠕虫Nimda通过执行类似“GET /scripts/Admin.dll HTTP/1.0”来激活目标主计上的蠕虫，蠕虫Sasser和Blaster通过控制连接激活目标主机上的蠕虫。

蠕虫的扫描（扩散）

- 随机扫描
- 本地优先扫描
- 顺序扫描
- 基于目标列表的扫描
- 基于路由信息的扫描
- 分治扫描策略
- 被动式扩散

随机扫描扩散

- 均匀随机扫描
  - 均匀随机扩散是指从扫描空间内随机抽取一个IP地址作为目标传播，扫描空间可以为整个IPv4地址空间
  - 算法简单、易实现，会产生大量异常的流量，容易在早期就被检测系统发现。
- 选择性随机扫描
  - 目标地址按照一定的算法随机生成，但对公网中不可能出现的地址块进行标识，避免扫描这些地址。
  - 算法简单、易实现的特点，若与本地优先原则结合，则能达到更好的传播效果。但选择性随机扫描容易引起网络阻塞，使得网络蠕虫在爆发之前易被发现，隐蔽性差
- CodeRed蠕虫、Slapper蠕虫、Slammer蠕虫



蠕虫传播趋势

本地优先扫描

- 主导思想：优先选择本地或者与本地相近的网络进行扫描。
  - 被感染主机上蠕虫的生成目标地址和所在主机的IP地址在同一个子网的概率比较大。如：Nimda蠕虫生成的目标地址有50%的概率在当前机器IP所在的B类地址范围内产生，有25%的概率在当前机器IP所在的A类范围内产生，另25%的概率是随机IP地址。这种扩散策略的支持者认为，一个主机被感染蠕虫之后，这个主机所在的子网的IP地址被分配出去并且被使用的概率比较大，从而能够增加发现漏洞主机的概率。
  - 将互联网地址空间中未分配的或者保留的地址块排除在扫描之列
  - 实现简单，传播速度与概率P的选取、地址空间数目相关
- 比如Blaster蠕虫和Nimda蠕虫，

顺序扫描

- 顺序扫描(sequential scan)：是指被感染主机上蠕虫会随机选择一个C类网络地址进行传播。根据本地优先原则，蠕虫一般会选择它所在网络内的IP地址。若蠕虫扫描的目标地址IP为A，则扫描的下一个地址IP为A+1或者A−1。一旦扫描到具有很多漏洞主机的网络时就会达到很好的传播效果。
- 该策略的不足是对同一台主机可能重复扫描，引起网络拥塞。所以尽量避免父节点与自节点的扫描空间相重合。多采用随机策略选择初始节点，扫描速度等价于随机策略
- W32.Blaster是典型的顺序扫描蠕虫。

基于目标列表的扫描扩散算法

- 基于目标列表的扩散是指网络蠕虫在寻找受感染的目标之前预先生成一份可能易传染的目标列表，然后对该列表进行攻击尝试和传播
- 目标列表生成方法有两种：①通过小规模的扫描或者互联网的共享信息产生目标列表；②通过分布式扫描可以生成全面的列表的数据库。
- 基于目标列表的扩散提高了蠕虫的传播速度，不足是网络蠕虫传播时必须携带一个IP地址库，使得蠕虫负载量增大。
- 分治扫描策略的另一个不足是存在“坏点”问题。在蠕虫传播的过程中，如果一台感染蠕虫的主机死机或崩溃，那么其所对应的剩余扫描列表中主机就会失去被感染的机会。并且，这个问题发生得越早，影响就越大。

基于路由表信息的扫描

- 网络蠕虫根据网络中的路由信息，对IP地址空间进行选择性扫描的一种方法。
- 采用随机扫描的网络蠕虫会对未分配的地址空间进行探测，而这些地址大部分在互联网上是无法路由的，因此会影响到蠕虫的传播速度。如果网络蠕虫能够知道哪些IP地址是可路由的，它就能够更快、更有效地进行传播，并能逃避一些对抗工具的检测。
- 网络蠕虫的设计者通常利用BGP路由表公开的信息获取互连网路由的IP地址前辍，然后来验证BGP数据库的可用性。
- 基于路由的扫描极大地提高了蠕虫的传播速度，以CodeRed为例，如果采用路由扫描，则蠕虫的感染率是采用随机扫描蠕虫感染率的3.5倍。

分治扫描

- 分治扫描(divide-conquer scan)：网络蠕虫之间相互协作、快速搜索易感染主机的一种策略。
- 网络蠕虫发送地址库的一部分给每台被感染的主机，然后每台主机再去扫描它所获得的地址。主机A感染了主机B以后，主机A将它自身携带的地址分出一部分给主机B，然后主机B开始扫描这一部分地址。
- 避免重复扫描，缺点：一旦节点失效，扫描中断

扫描策略评价

- 在理想情况下，当漏洞主机均匀分布在网络中的时候。
  - 蠕虫采用均匀随机扩散策略时传播速度比采用本地优先策略传播速度快
  - 基于目标列表的扩散策略当列表小于6M字节时，蠕虫传播速度比均匀随机扫描快
- 实际中仍然主要是随机扫描和顺序扫描两种方式，使用其它的传播策略的蠕虫还很少。通常蠕虫对目标列表的扫描，以及局域网内的顺序扫描都仅仅是在蠕虫传播的早期，大量的随机扫描则范围大，持续时间长，构成了蠕虫传播的主体，例如，曾经大肆泛滥造成巨大危害的：Code Red、Code Red II、Slammer、Blaster、Sasser、Welchia等蠕虫

被动式扩散算法

- 被动式传播蠕虫则不需要主动扫描就能够传播。被动式蠕虫通过潜伏在已感染主机中，等待潜在的攻击对象来主动接触它们，或者通过监听网络数据报文，获取其它用户活动信息，从而发现新的攻击目标。
- 由于它们需要目标触发，所以当目标数量少或者扫描频率低的时候传播速度很慢，但当目标造成的扫描频率很高，则传播速度会很快。
- 这类蠕虫在发现目标的过程中并不会引起通信异常，这使得它们自身有更强的安全性。Contagion和CRClean都是被动式蠕虫，Contagion通过监听正常的通信来发现新的攻击对象；CRClean则通过等待CodeRed II的扫描活动来发现新的攻击对象，当它发现有CodeRed II向自己所在主机扫描时，就发起一个反攻来回应该CodeRed，如果反攻成功，它就删除CodeRed II，并在目标主机上建立CRClean副本。

网络安全事件应急响应

- 应急响应体系与机制
- 应急响应的支持技术

概念

- 应急响应（Incident Response/Emergency Response）
  - 通常是指一个组织为了应对各种意外事件的发生所做的准备以及在事件发生后所采取的措施，其目的是避免、降低危害和损失，以及从危害和损失中恢复。
  - 计算机及网络攻击应急响应就是针对计算机攻击及网络攻击事件所采取的应急措施。

事件种类举例

- 计算机入侵
  —系统被破 、网页涂改
- 病毒、蠕虫爆发
- 发现非授权应用
  —发现加密软件、隐秘通信软件
  —发现黑客工具
- 拒绝服务攻击
- 知识产权被窃
- 内部或外部对系统的非法使用
  —赌博或色情网站
  —被作为攻击的跳板

应急响应做什么？

- 应急响应的活动应该主要包括两个方面：
  1. 未雨绸缪
  2. 亡羊补牢
  - 以上两个方面的工作是相互补充的

应急响应做什么？

- 未雨绸缪
  - 即在事件发生前事先做好准备，比如：
    - 风险评估
    - 制定安全计划
    - 安全意识的培训
    - 以发布安全通告的方式进行的预警
    - 各种检测与防范措施。

应急响应做什么？

- 亡羊补牢
  - 即在事件发生后采取的措施，其目的在于把事件造成的损失降到最小。
    - 这些行动措施可能来自于人，也可能来自系统。
      - 比如发现事件发生后，病毒检测、后门检测、清除病毒或后门、隔离、系统恢复、调查与追踪、入侵者取证等一系列操作。

应急组织

- 美国国防部资助卡内基梅隆大学(CMU)的软件工程研究所成立了计算机紧急响应组协调中心(CERT/CC)
  - 1988年的莫里斯蠕虫事件之后的一个星期内建立的
  - 通常被认为是第一个应急响应组

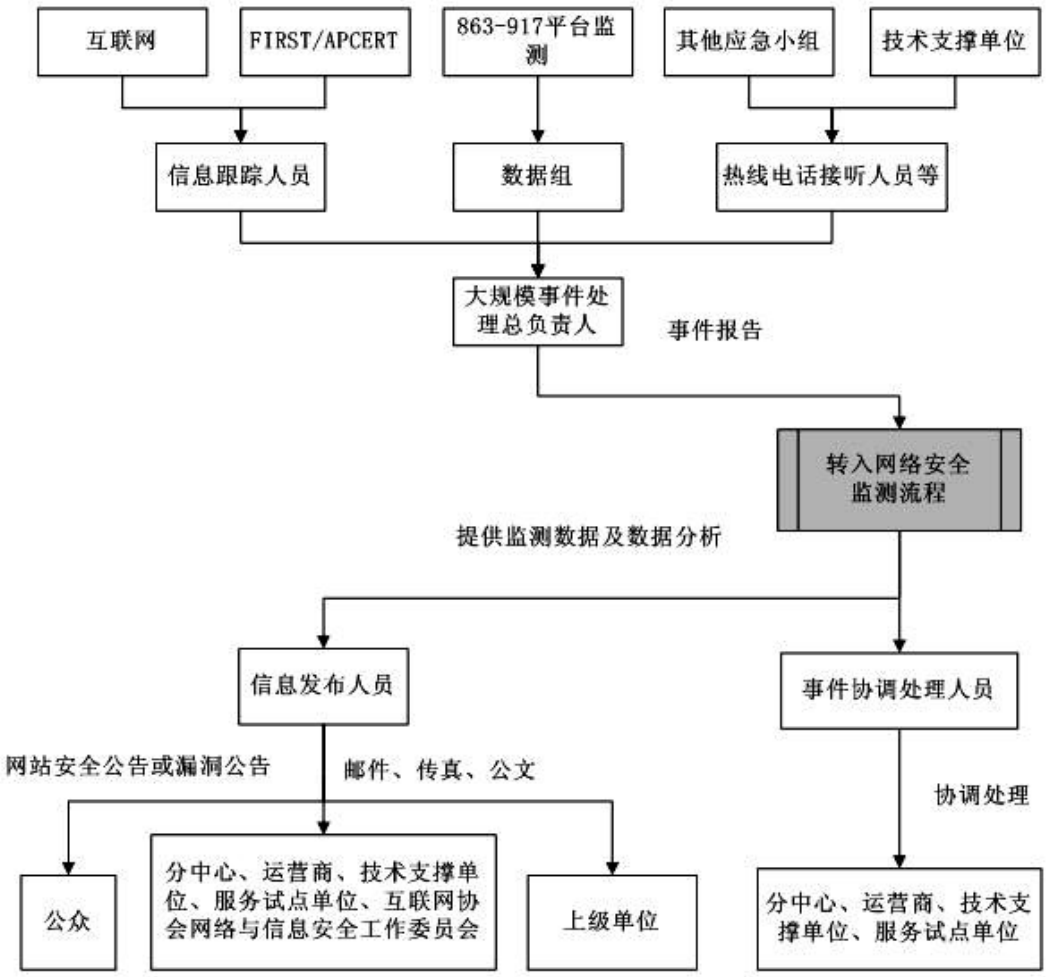
应急组织

- CERT/CC成立后，世界各地应急响应组雨后春笋般地出现:
  - 美国联邦FedCIRC
  - 澳大利亚的AusCERT
  - 德国的DFN-CERT，日本的JPCERT/CC
  - 亚太地区APCERTF
    - （Asia Pacific Computer Emergency Response Task Force）
  - 欧洲EuroCERT，
  - FIRST（Forum of Incident Response and Security Teams）
    - 论坛旨在促进全球CIRST组织之间协调与合作。

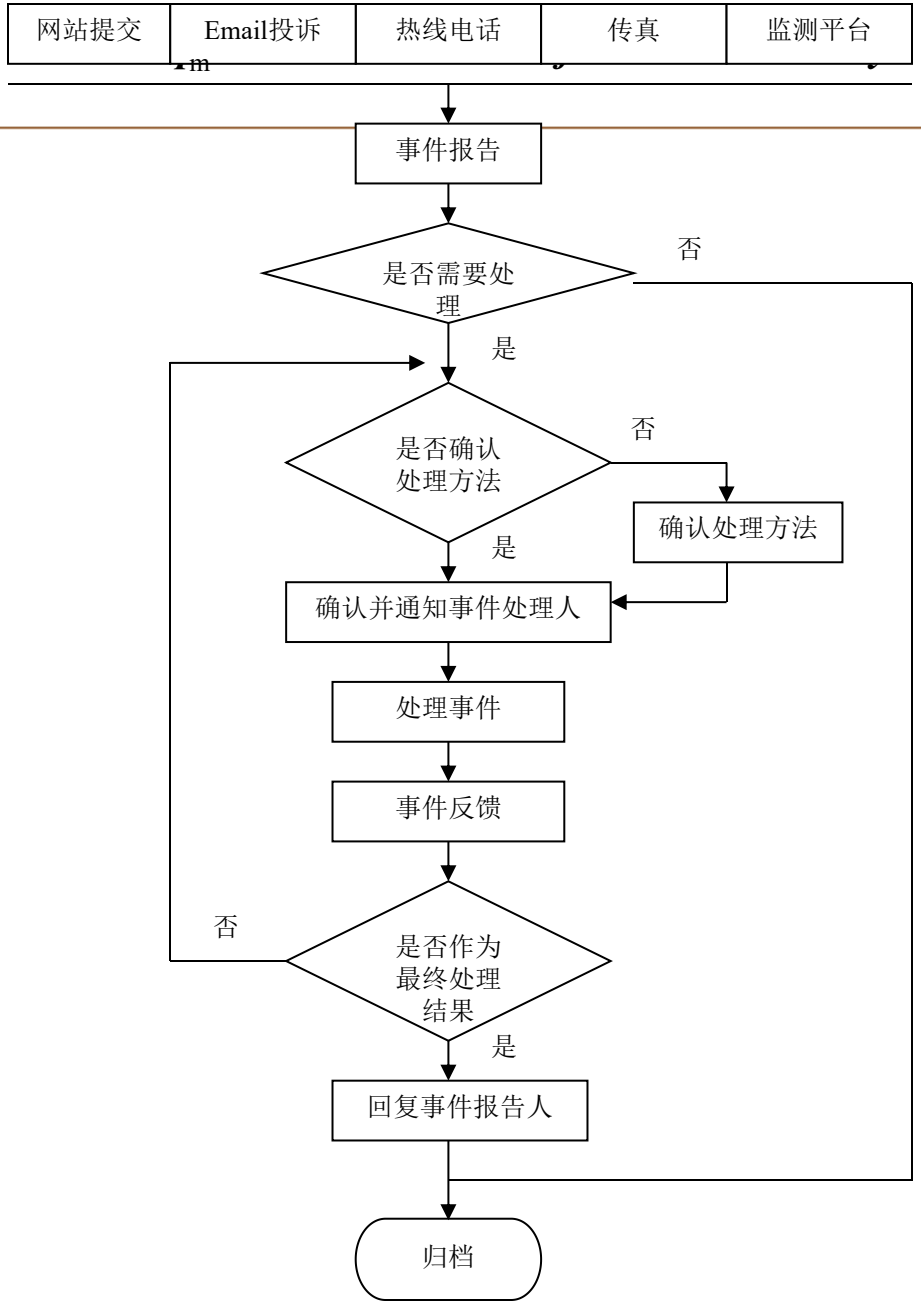
中国的应急组织

- CCERT
  - CERNET应急响应组
  - 中国教育与科研计算机网络（CERNET）于1999年在清华大学成立
  - 为中国教育和科研行业的用户提供应急响应服务。
- CNCERT
  - 中国计算机安全事件应急响应小组/协调办公室
  - 国家计算机网络与信息安全管理中心成立。
  - 负责国内各部门、行业等应急响应小组的协调工作。

CNCERT大规模事件处理流程



CNCERT个案事件处理流程



预防服务

- 发布安全公告
- 安全技术追踪
- 安全审计与安全评估
- 配置与维护安全工具、安全应用和安全基础设施
- 开发安全工具
- 检测入侵行为

响应服务

- 安全警告
- 事件处理 ：
  (1) 事件分析；
  (2) 事件现场响应；
  (3) 事件响应支持；
  (4) 事件响应配合
- 漏洞处理：
  (1) 漏洞分析；
  (2) 漏洞响应；
  (3) 漏洞响应配合
- 可疑物处理：
  (1) 可疑物分析；
  (2) 可疑物响应；
  (3) 可疑物响应配合

基础设施

- 分布式入侵检测设施
- 分析、取证设施
- 早期发现与预警设施
- 协同事件处理系统
- 安全资源管理系统（WWW、FTP）
- 安全通信系统

CCERT应急响应案例分析

- 口令蠕虫（DvlDr32.worm）
  - 2003年3月7日

监测与报告

- 2003/03/07，各地用户报告报告网络变慢 甚至中断
- 网络监测发现流量异常：
  - TCP/445 扫描

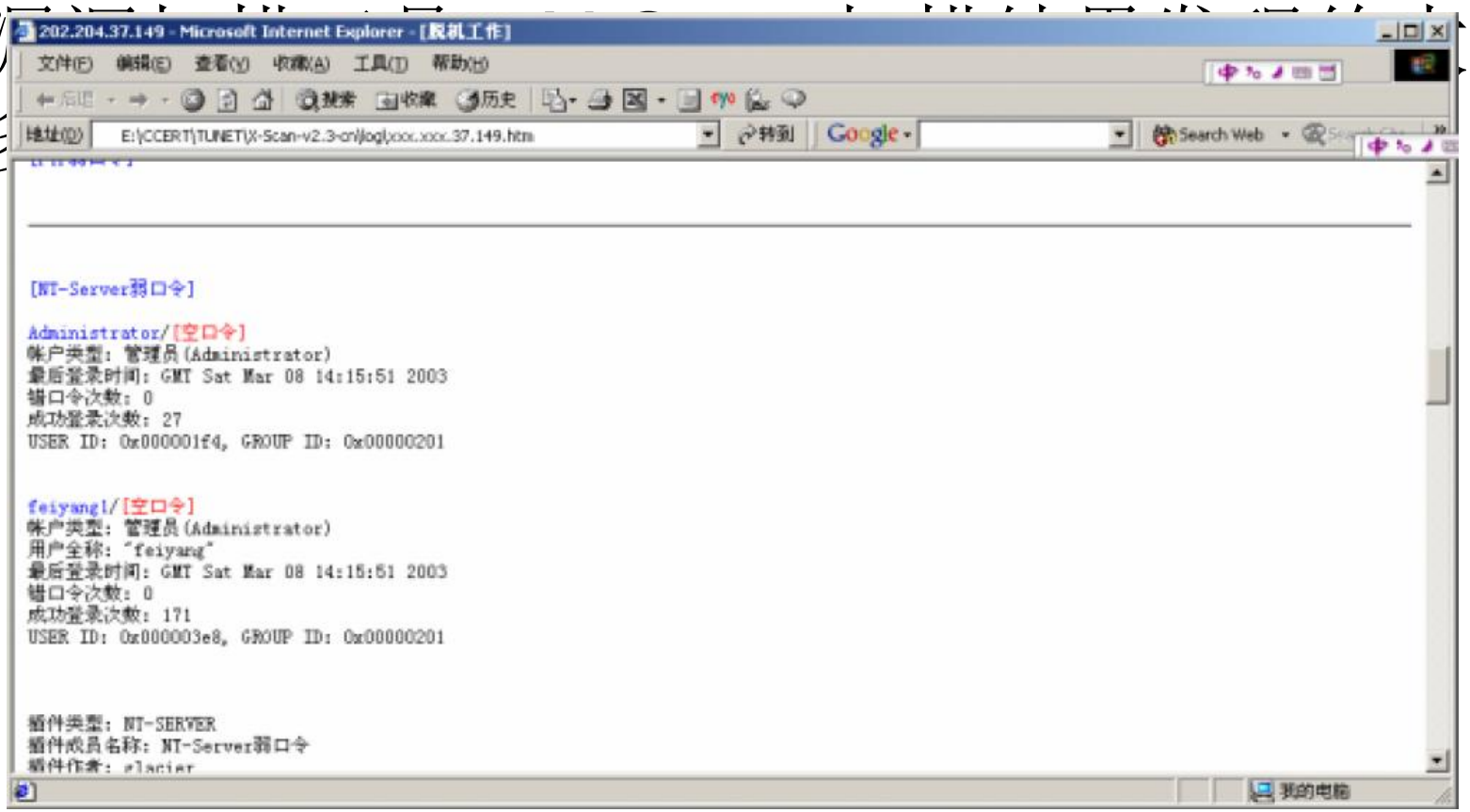
Protocol	Packets%	Bytes %
TCP	14.96%	9.57%
UDP	0.91%	1.14%
ICMP	1.69%	1.18%
OTHERS	82.44%	88.11%

调查与分析：远程端口扫描

[root@scan inprotect]# nmap -sS -O a.a.38.35
Starting nmap 3.27 (www.insecure.org/nmap/) at 2003-07-04 10:24 CST Interesting ports on a.a.28.35:
(The 1613 ports scanned but not shown below are in state: closed) Port State Service
21/tcp open ftp
135/tcp open loc-srv
139/tcp open netbios-ssn
445/tcp open microsoft-ds
1433/tcp filtered ms-sql-s
1434/tcp filtered ms-sql-m
1483/tcp filtered afs
1998/tcp open x25-svc-port
5800/tcp open vnc-http
5900/tcp open vnc
Remote operating system guess: Windows Millennium Edition (Me), Win
2000, or WinXP
Nmap run completed -- 1 IP address (1 host up) scanned in 4.728 seconds

调查与分析：远程漏洞扫描

- 漏洞扫描：最[illegible]



调查与分析：TCP 连接

C:\>netstat -an
Active Connections

Proto	Local Address	Foreign Address	State
……			
……			
TCP	0.0.0.0:5800	0.0.0.0:0	LISTENING
TCP	0.0.0.0:5900	0.0.0.0:0	LISTENING
TCP	127.0.0.1:43958	0.0.0.0:0	LISTENING
TCP	202.112.100.20:139	0.0.0.0:0	LISTENING
TCP	202.112.100.20:1667	155.186.170.0:445	SYN_SENT
TCP	202.112.100.20:1673	155.186.170.6:445	SYN_SENT
.....			
TCP	202.112.100.20:1674	63.226.170.0:445	SYN_SENT
TCP	202.112.100.20:1680	63.226.170.3:445	SYN_SENT
……			
TCP	202.112.100.20:2181	63.226.170.255:445	SYN_SENT
TCP	202.112.100.20:2738	61.182.210.26:83	ESTABLISHED
TCP	202.112.100.20:4899	166.111.232.177:3332	ESTABLISHED
TCP	202.112.100.20:4948	166.111.232.177:139	TIME_WAIT
TCP	x.x.x.x:4505	210.159.30.250:6667	CLOSE_WAIT
….			
TCP	x.x.x.x:4811	198.65.147.245:6667	CLOSE_WAIT
TCP	x.x.x.x:4887	149.156.91.2:6667	CLOSE_WAIT
TCP	x.x.x.x:4976	198.65.147.245:6667	CLOSE_WAIT

调查与分析: 后门程序

检查注册表, 发现增加了如下键值:

[HKEY_LOCAL_MACHINE\Software\Microsoft\Win dows\CurrentVersion\Run]

"TaskMan"="C:\WINNT\Fonts\rundll32.exe"

"Explorer"="C:\WINNT\Fonts\explorer.exe"

"messnger"="C:\WINNT\system32\Dvldr32.exe"

调查与分析：蠕虫程序分析

- Dvldr32.exe程序运行后，在一定范围内随机选择IP地址，通过网络邻居（TCP 445端口）连接目标主机，猜测帐号口令，成功后将自身复制到目标系统中。并留下后门，同时通过6667端口把被攻击的主机列表放到一系列聊天服务器上。

报告与通告

- 报告CERNET管理层，与网络管理部门（NOC）讨论控制解决方案
- 紧急报告公安部、 CNCERT/CC
- 通过网站和邮件列表发布安全通告


CERNET 网络应急响应组 - Microsoft Internet Explorer
文件(F) 编辑(E) 查看(V) 收藏(A) 工具(T) 帮助(H)
后退 搜索 收藏夹 媒体
地址(D) http://www.ccert.edu.cn/notice/zhow.php?handle=47
转到
Google
Search Web
47 blocked
AutoFill
Options
CERNET COMPUTER EMERGENCY RESPONSE TEAM
CCERT 中国教育和科研计算机网紧急响应组
关于我们 | 系统补丁 | 安全漏洞 | 常用工具 | 相关文档 | 安全论坛 | 教育培训 | 安全公告 | 垃圾邮件 | 安全咨询 | 相关链接
-首页
|-安全公告
|-具体信息
English
CCERT 关于 Dvldr32 蠕虫防范的紧急公告
CCERT 关于 Dvldr32 蠕虫防范的紧急公告
1.检测
2003年3月7日以来，CERNET主干网流量出现异常，主要表现在TCP 445, tcp 6667 端口流量、ip protocol =255 的流量异常增大。3月 8日12:30左右，ip protocol 255 、TCP 445 流量急剧上升，严重影响了网络性能。
2.攻击源分析
分析这些来源主机，主要有以下特征：
1) 运行 MS Windows 系统（NT/2000/ 或 XP）；
2) 开放 tcp 445 , 5800，5900 端口；
3) administrator 口令为空，另外还有一个普通用户账号口令为空；
在被感染的主机上可以看到
C:\>netstat.exe -n
Active Connections
Proto Local Address Foreign Address State
TCP x.x.x.x:1043 149.156.91.2:6667 CLOSE_WAIT
TCP x.x.x.x:1045 198.65.147.245:6667 CLOSE_WAIT
TCP x.x.x.x:4505 210.159.30.250:6667 CLOSE_WAIT
TCP x.x.x.x:4599 203.31.191.207:6667 CLOSE_WAIT
TCP x.x.x.x:4710 209.114.200.8:6667 CLOSE_WAIT
TCP x.x.x.x:4811 198.65.147.245:6667 CLOSE_WAIT
TCP x.x.x.x:4887 149.156.91.2:6667 CLOSE_WAIT
完毕
Internet

网络控制与隔离

- NOC 与各地区网络中心配合，在主干网上采取了控制措施，控制措施是在边界路由器上配置了如下ACL:

```
access-list 110 deny tcp any any eq 445
access-list 110 deny udp any any eq 445
access-list 110 deny 255 any any
access-list 110 deny 0 any any
access-list 110 permit ip any any
```

网络控制与隔离

- CERNET主干网在3月8日20：00 左右恢复正常，从检 测出爆发到控制经历了3个小时。

事后评估

- 尽管网络得到了控制，但是安全隐患并没有消失
- 没有口令或弱口令的计算机大量存在
- 大量的计算机被安装远程控制软件

应急响应的支撑技术

- 应急响应支撑技术
  - 漏洞检测技术及相关工具
  - 阻断技术
  - 路由控制技术
  - 网络追踪技术
  - 取证技术
  - 分布式检测技术
  - 网络陷阱及诱骗技术

漏洞检测技术及相关工具

- 已知漏洞的检测
  - 基于主机、基于网络
  - 按照扫描过程来分
    - Ping扫描技术
    - 端口扫描技术
    - 操作系统探测扫描技术
    - 已知漏洞的扫描技术
- 未知漏洞的检测
  - 静态漏洞检测技术
    - 源代码扫描
    - 反汇编扫描
  - 动态的漏洞检测技术
    - 环境错误注入
- 常用扫描工具
  - X-Scan
  - Nessus

阻断技术

- 3种基于IP的阻断方式
  - ICMP不可达响应
    - 通过向被攻击主机或攻击源发送ICMP端口或目的不可达报文来阻断攻击；
  - TCP—RST响应
    - 也称阻断会话响应，通过阻断攻击者和受害者之间的TCP会话来阻断攻击。
    - 是目前使用最多的主动响应机制；
  - 防火墙联动响应
    - 当入侵检测系统检测到攻击事件后向防火墙发送规则，由防火墙阻断当前以及后续攻击。
- 基于域名的阻断
  - 域名欺骗

路由控制技术

- 路由控制
  - 利用路由器自带的访问控制列表（ACL）功能进行安全防护和阻断攻击的方法。
  - 可以说路由器的访问控制列表是网络安全保障的第一道关卡。

路由控制技术

- 路由器的访问控制不同于普通的个人版防火墙，它多用来抵制大规模，集中式的攻击。例如蠕虫病毒爆发。
  - 以slammer蠕虫为例，我们知道slammer蠕虫利用udp 1434端口进行传播。这时我们就可以在边界路由上添加以下规则：

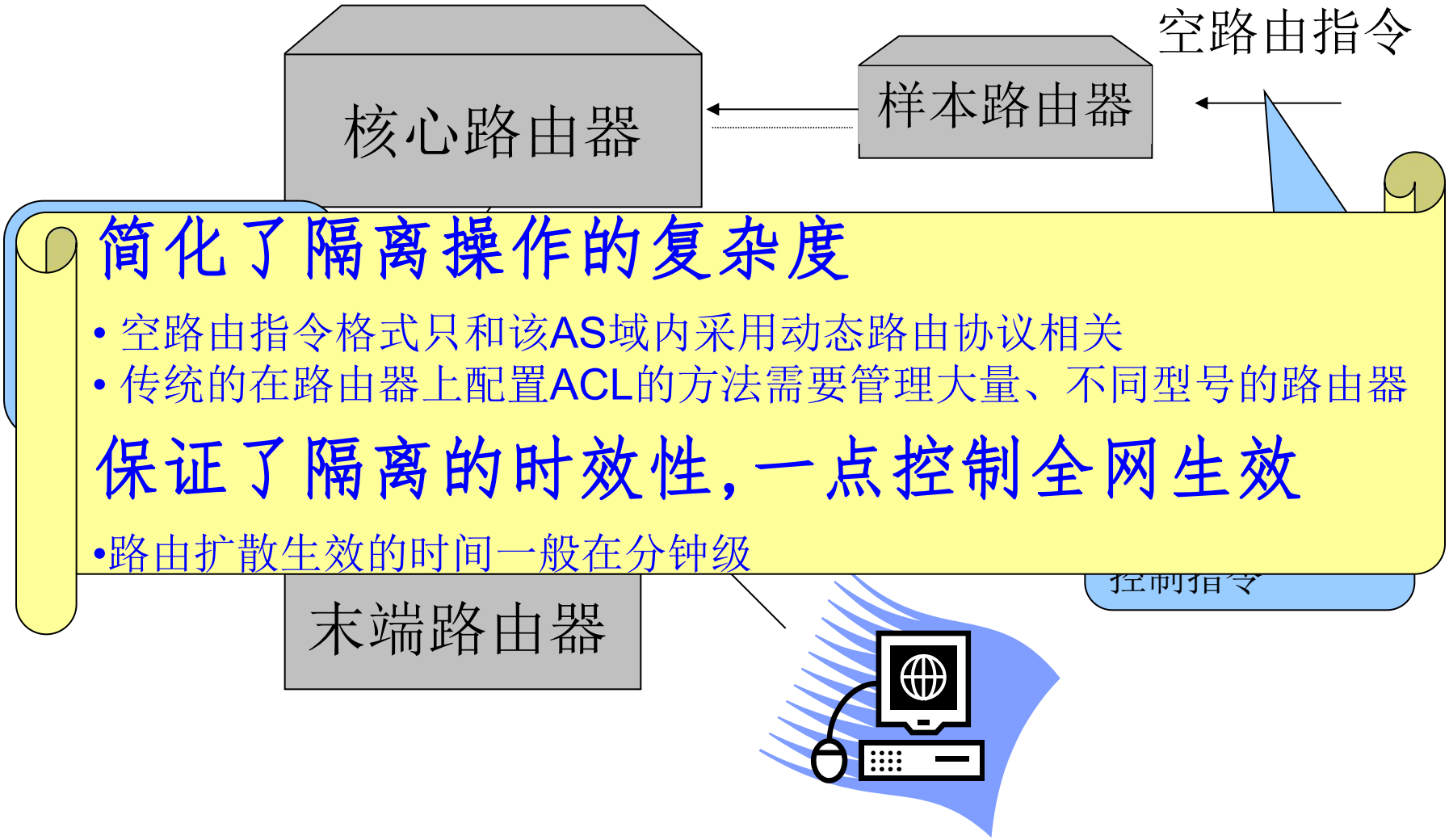
    access-list 110 deny udp any any eq 1434
  - 这条访问控制规则使得所有发往udp 1434端口的数据包都无法通过路由器，有效的阻断了slammer蠕虫的传播途径，从而达到集中控制的效果。
- 缺点：过多的访问控制列表会大大增加路由器的负荷，因此在主干路由器上请尽可能少的添加ACL控制规则。

路由控制技术

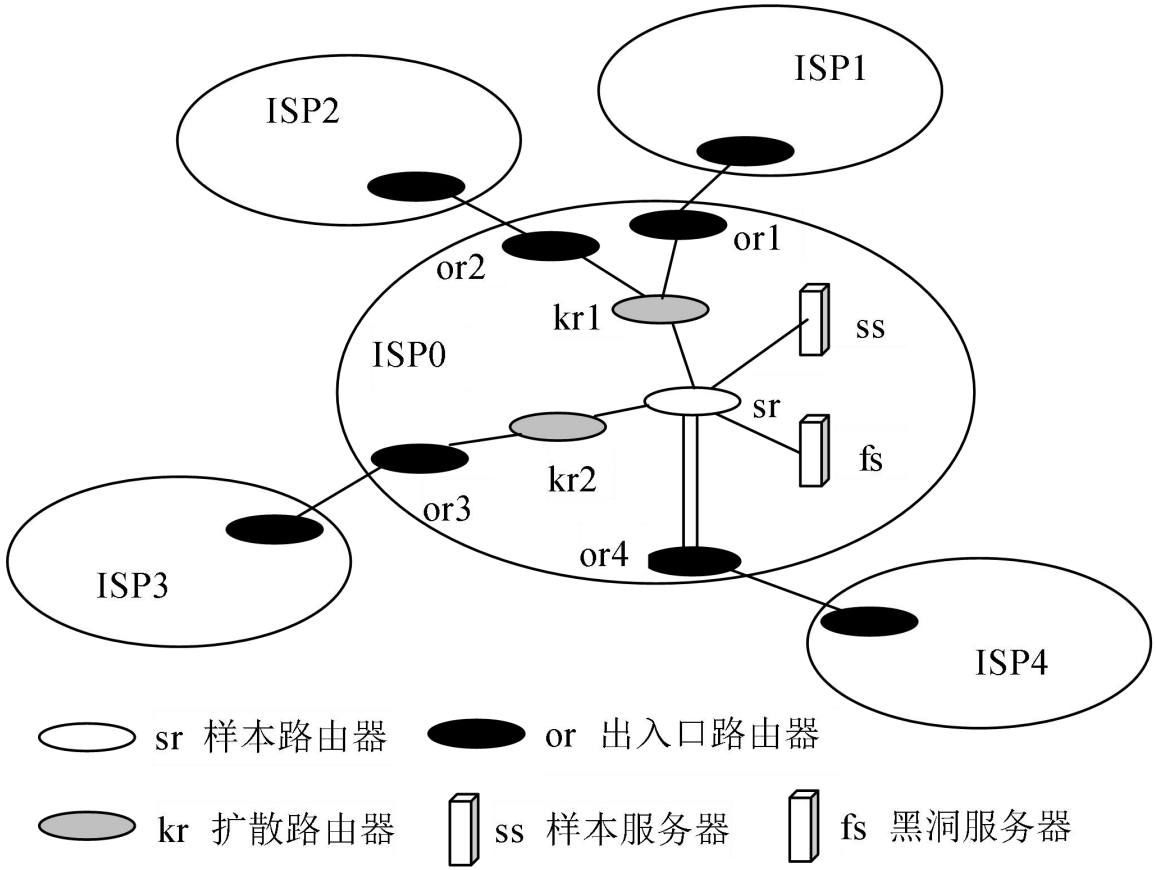
- 空路由规则控制
  - 在边界路由器内配置黑洞路由（空路由）对目标地址进行访问控制：

    add route 192.168.2.1 255.255.255.255
  - 这条访问控制规则使得所有发往192.168.2.1的数据包都被直接丢弃了，无法通过路由器。
- 优点：规则数目相对ACL可以更多，效率更高。可以通过动态路由协议扩散全域。

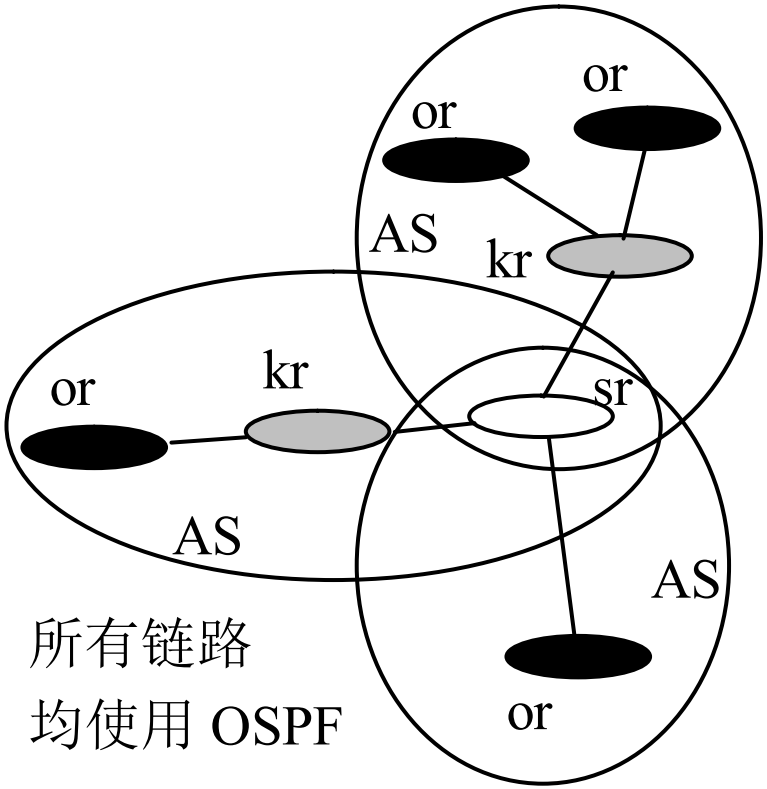
基于路由扩散的隔离控制技术



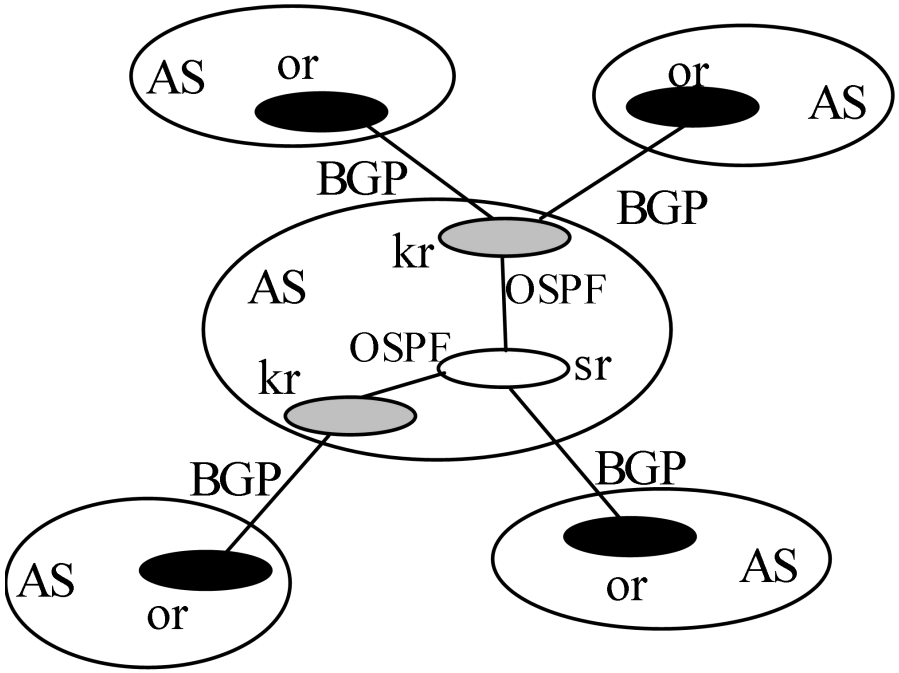
基于路由扩散的隔离控制技术



基于路由扩散的隔离控制技术



所有链路
均使用 OSPF



网络反向追踪技术

- 反向追踪技术分为
  - 包跟踪
  - 物理线路跟踪

反向追踪技术及相关工具

- 包跟踪
  - 黑客在攻击时为了掩盖自己的踪迹会通过一个或者几个跳板（被控制的傀儡机器）来进行攻击。
  - 包跟踪就是学习一个包从哪来的过程。
  - 包跟踪可以实时发生，这时攻击者和追踪者必须同时保持在线，追踪者可以通过监听包的信息来发现攻击者的来源。
  - 通过读取相关的日志文件来执行包跟踪也是可能的，这种情况可以发生在攻击过程中或者攻击后。

反向追踪技术及相关工具

- 物理线路跟踪
  - 通过物理连接设备来追踪攻击来源的一种技术，目前这种技术并不成熟，常用的是路由回朔技术，就是通过追踪物理路由的连接端口来确定攻击来源。
    - 由于这种方法牵涉到太多的物理设备，因此必然会耗费大量的人力和物力。
    - 这种技术适用于追踪那些无法确定来源地址的攻击，例如伪造源发地址的DOS攻击。

- 基于路由器日志的反向追踪
  - 分布式数据库，记录路由器数据包的源、目的地址及上一跳的路由器地址
    - 针对目的地址进行哈希存储
  - 反向追踪
    - 查找最近的路由器
    - 匹配目的地址
    - 追踪下一个路由器

取证技术

- 对存储在计算机系统或网络设备中潜在电子证据的识别、收集、保护、检查和分析以及法庭出示的过程，通常是对存储介质、日志的检查和分析。
- 计算机取证包括物理证据获取和信息发现两个阶段。

在应急响应中，收集黑客入侵的证据是一项非常重要的工作。取证技术不但可以为打击计算机、网络犯罪提供重要支撑手段，还可为司法鉴定提供强有力的证据。

按需取证技术

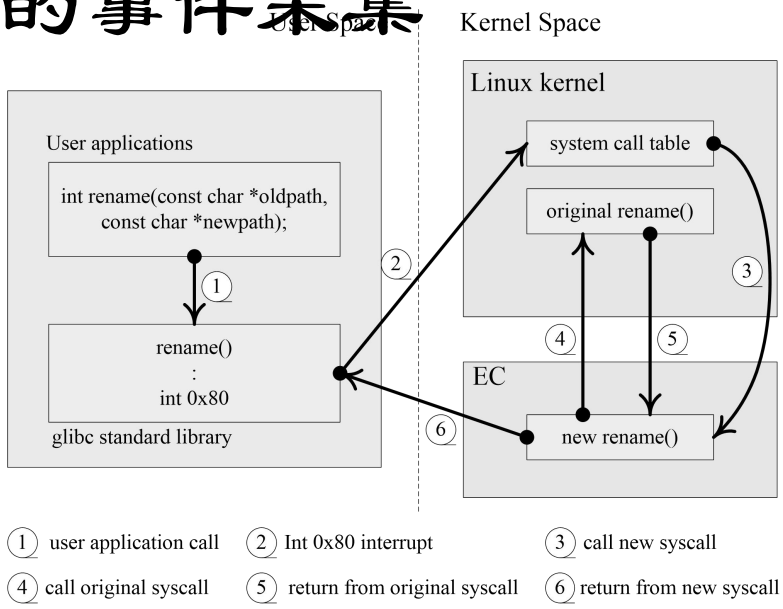
- 按需取证技术
  - 基于实时取证的需求，按需取证的必要性
  - 按需取证的概念
    - 在取证之初基于不同取证环境合理设置取证方法及对象，从而达到缩小处理范围、缩短调查取证时间、提高证据有效性的目的

DFR2的核心关键技术

- 应用无关的事件采集
- 按需取证机制
- 基于对象依赖的取证分析
- 多源证据融合

应用无关的事件采集

- DFR2通过系统调用劫持实现了细粒度的、应用无关的事件采集



类别	系统调用
文件系统	chmod, chown, chown32, fchmod, fchown, fchown32, lchown, lchown32, link, mknod, mount, open, rename, symlink, unlink, close, creat, dup2, flock, ftruncate, ftruncate64, ioctl, mkdir, nfsservctl, quotactl, rmdir, truncate, truncate64, umount, umount2, chdir, chroot, dup, fchdir, fcntl, fcntl64, fsync, llseek, lseek, newselect, poll, pread, putpmsg, pwrite, read, readv, select, sendfile, umask, utime, afs_syscall, write, writev, access, bdflush, fdatasync, fstat, fstat64, fstatfs, getcwd, getdents, getdents64, getpmsg, lstat, lstat64, oldfstat, oldlstat, oldolduname, oldstat, olduname, pipe, readahead, readdir, readlink, stat, stat64, statfs, sync, sysfs, ustat, mmap, mmap2, munmap, msync
网络	accept, bind, connect, listen, socket, socketpair, recv, recvfrom, recvmsg, sedmsg, send, sendto, setsocketopt, shutdown, getpeername, getsocketname, getsocketopt,
进程	execve, setfsgid, setfsgid32, setfsuid, setfsuid32, setgid, setgid32, setgroups, setgroups32, setregid, setregid32, setresgid, setresgid32, setresuid, setresuid32, setreuid, setreuid32, setuid, setuid32, vfork, adjtimex, brk, clone, exit, fork, ioperm, iopl, kill, modifyldt, nice, ptrace, reboot, vhangup, sched_setparam, sched_setscheduler, sched_yield, setpriority, setrlimit, vm86, vm86old, capset, personality, prctl, setpgid, setsid, uselib, wait4, waitpid, acct, capget, getegid, getegid32, geteuid, geteuid32, getgid, getgid32, getgroups, getgroups32, getpgid, getpgrp, getpid, getppid, getpriority, getresgid, getresgid32, getresuid, getresuid32, getrlimit, getrusage, sched_getscheduler, sched_get_priority_max, sched_get_priority_min, sched_getparam, sched_rr_get_interval, getsid, getuid, getuid32,
信号	rt_sigaction, rt_sigpending, rt_sigprocmask, rt_sigqueueinfo, rt_sigreturn, rt_sigsuspend, rt_sigtimedwait, sgetmask, sigaction, sigaltstack, signal, sigpending, sigreturn, sigsuspend, sigprocmask, ssetmask

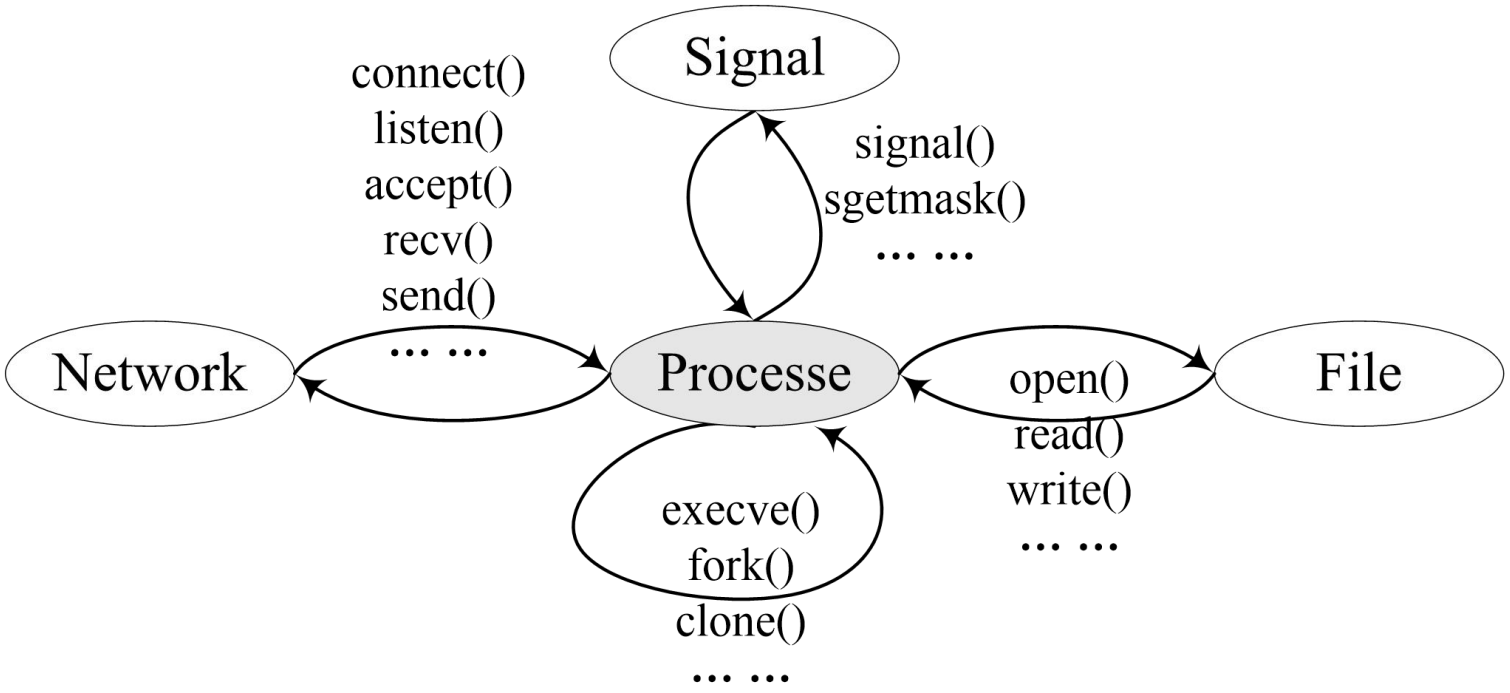
按需取证机制

- 在DFR2中，按需取证将取证需求以事件和对象的形式进行定义
  - 对象是取证过程中最基本的可选取证需求单位
  - 事件则是取证需求的具体表现形式，由对象的属性描述及各对象间的逻辑关系构成
- 按需取证API
  - 为了方便上层应用软件的升级和移植，按需取证组件屏蔽了底层的实现细节，将事件和对象的接口以API函数的形式提供给用户，通过API函数调用，用户可以很容易地实现取证策略的按需定制

类别		API	功能描述
事件管理		DFR2_Create	创建一个取证需求
		DFR2_addevent	增加一个取证事件
		DFR2_Delete	删除一个取证需求
逻辑运算		DFR2_and	对象或事件间的逻辑与操作
		DFR2_or	对象或事件间的逻辑或操作
		DFR2_no	对象或事件间的逻辑非操作
对象操作	进程	DFR2_Pid	对指定 PID 的进程对象进行取证
		DFR2_Childof	对指定进程的子进程对象进行取证
		DFR2_Uid	对指定 UID 的进程对象进行取证
	文件系统	DFR2_File	对指定名称的文件对象进行取证
		DFR2_Dir	对指定的目录进行取证
		DFR2_Subdir	对指定的目录的子目录进行取证
	网络	DFR2_Protocol	对指定的协议进行取证
		DFR2_Sip	对指定源 IP 的网络对象进行取证
		DFR2_Dip	对指定目的 IP 的网络对象进行取证
		DFR2_Sport	对指定源端口的网络对象进行取证
		DFR2_Dport	对指定目的端口的网络对象进行取证
	信号	DFR2_Sig	对指定的信号对象进行取证

基于对象依赖的取证分析

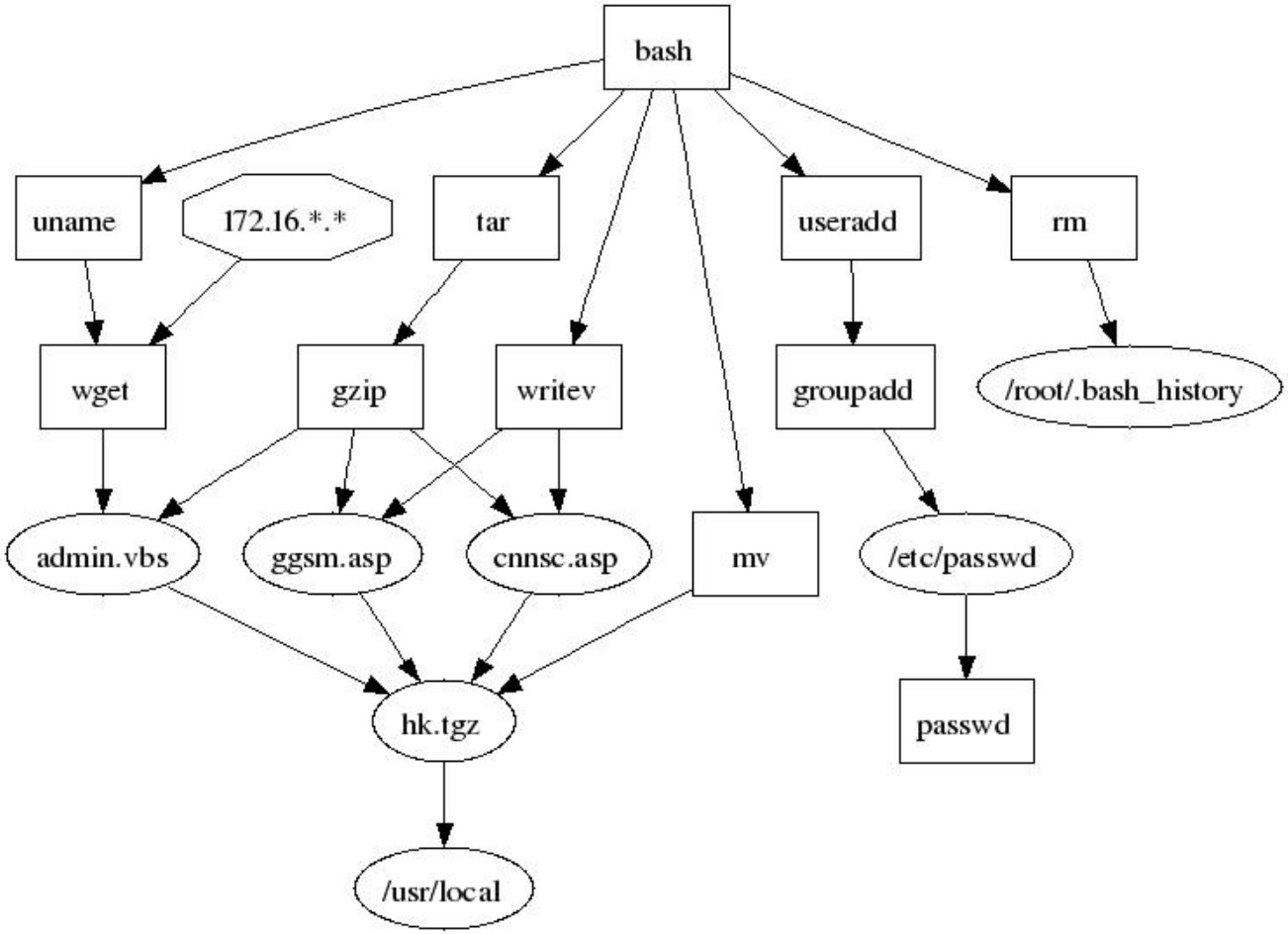
- 对象依赖关系转换图



多源证据融合

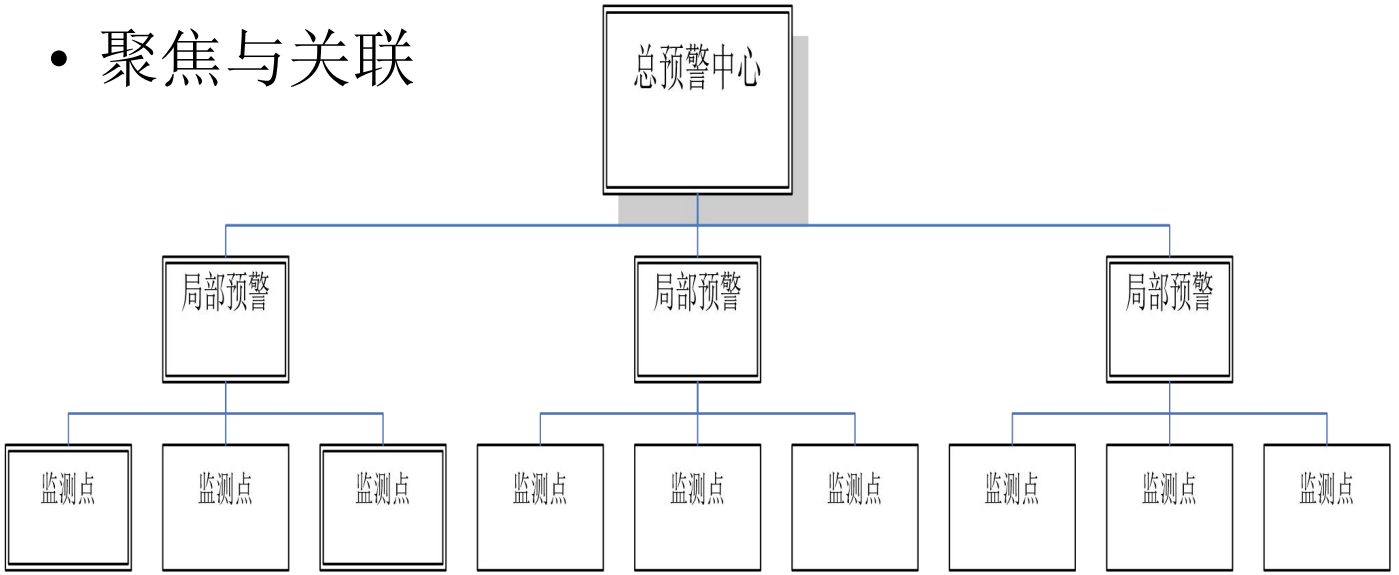
- 跳板攻击
  - 为了更好地隐蔽自己，入侵者通常不直接向目标主机发动攻击，而是采用过渡手段，先侵入若干中间系统并以之为跳板，最后通过跳板发动对目标主机的攻击。单凭一台机器的取证分析结果无法提取出完整的攻击流程，降低了证据的说服力
- 多源融合
  - 取证前后端独立运行的模式，使DFR2具有良好的可扩展性。针对跳板攻击的取证需要，只需在跳板上分别添加取证前端即可，各取证前端所产生的证据向量则由id字段加以区分。此外，取证分析服务组件FA还支持多源证据向量的自动融合，实现对跳板攻击类型进行取证的目的

- 取证分析结果测试


bash
uname
172.16.*.*
tar
useradd
rm
wget
gzip
writev
groupadd
/root/.bash_history
admin.vbs
ggsm.asp
cnnsc.asp
mv
/etc/passwd
hk.tgz
passwd
/usr/local

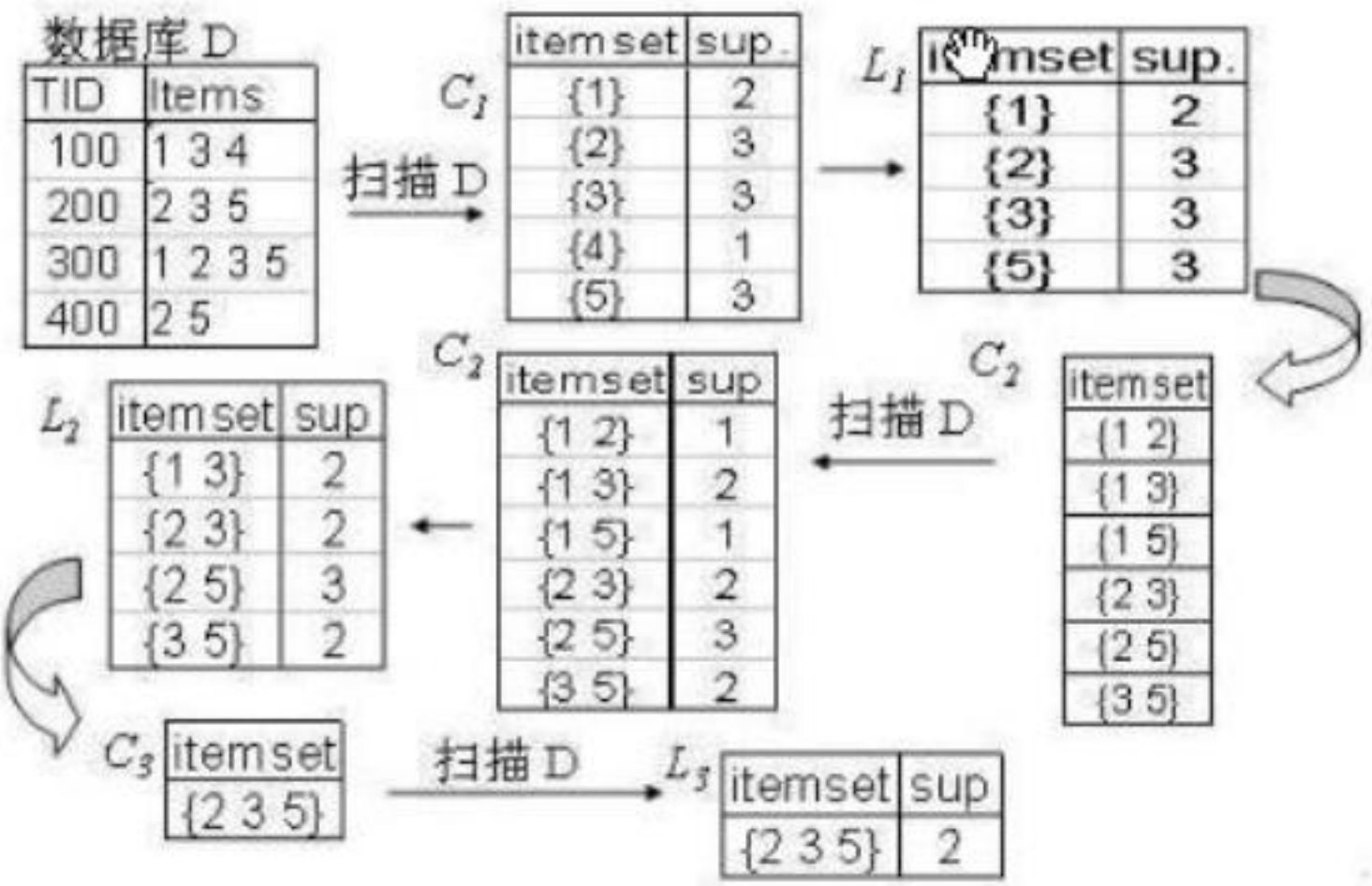
分布式检测技术

- 分布式检测技术
  - 关键在于对报警信息进行关联分析，将相似的报警通过融合的方法删除冗余，同时关联独立的报警事件。
    - 聚焦与关联



- 聚焦识别方法
  - DDOS, PROBE, R2L,U2R,DATA
  - 聚焦识别算法需要两个参数，一个滑动时间窗口参数T，当两条报警的时间差大于T时，认为这两条报警没有任何关系；当这两条报警时间差小于等于T时，在来判断目的IP是否相同；一个攻击阀值N，当报警中相同的目的IP出现多次时，而对应的源IP的数目超过N时，就认为目的IP可能受到DDOS攻击。

- 关联
  - 从大量的数据中挖掘出有价值的描述数据项之间相互联系的有关知识
- 关联规则算法
  - 基本算法Apriori思想：
    - 找到所有频繁项目集
    - 由频繁项集产生强关联规则：根据定义，这些规则必须满足最小支持和最小置信度。
  - 为生成所有频繁项集，该算法采用层次顺序搜索的循环方法，由k项集来产生(k+1)项集。具体做法就是：首先找出频繁1项集，记为L1；然后L1利用来L2挖掘，即频繁2项集；不断如此循环下去直到无法发现更多的频繁k项集为止。每挖掘一层就需要扫描整个数据库一遍。


数据库 D
TID Items
100 1 3 4
200 2 3 5
300 1 2 3 5
400 2 5
扫描 D
C_1
itemset sup.
{1} 2
{2} 3
{3} 3
{4} 1
{5} 3
L_1
itemset sup.
{1} 2
{2} 3
{3} 3
{5} 3
C_2
itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}
扫描 D
C_2
itemset sup
{1 2} 1
{1 3} 2
{1 5} 1
{2 3} 2
{2 5} 3
{3 5} 2
L_2
itemset sup
{1 3} 2
{2 3} 2
{2 5} 3
{3 5} 2
C_3 itemset
{2 3 5}
扫描 D
L_3
itemset sup
{2 3 5} 2

网络陷阱及诱骗技术

- 可以用来研究和防止黑客攻击行为，增加攻击者的工作量和攻击复杂度，为真实系统做好防御准备赢得宝贵时间。
- 可为打击计算机犯罪提供举证。
- 主要形式
  - 蜜罐
  - 蜜网
  - 蜜场
- 蜜罐蜜网等并没有向外界提供真正有价值的服务，那么所有对蜜罐蜜网进行连接的尝试都被视为可疑的。

- 蜜罐分类：按类型
  - 产品型蜜罐
    - 用来进行保护特定的目标
    - HoneyBrowser
  - 研究型蜜罐
    - 更多的用来学习
- 蜜罐分类：按交互模式
  - 低交互型
  - 中交互型
  - 高交互型

产品型蜜罐可以被看作是一个替身，职责是为另一台机器或者网络提供安全保护。它被设计成有缺陷的系统放置在网络中用于吸引攻击者的目光，能有效的牵制攻击者、赢得宝贵时间来采取防御措施，从而保护真实有用的系统。

研究型蜜罐则像一个陷阱，它吸引攻击者到来，然后不动生色的配合攻击者，其实暗中观察和记录攻击者的一切举动。它的目的是为了更好的搜集攻击者的信息、研究攻击者行为，可以发现新的攻击类型、手段以及新的黑客工具，为研究者提供了大量有价值的数据，也为IDS等技术提供了新的规则信息。

蜜罐按照物理实现方式还可以分为物理蜜罐和虚拟蜜罐。所谓物理蜜罐就是说蜜罐系统除了不具有任何业务功能，其它方面和平时使用的真实系统一样；所谓虚拟蜜罐就是一个对外提供虚拟服务的程序或利用虚拟机实现虚拟操作系统或网络环境。

- Honeyd
  - Honeyd是蜜罐技术的优秀代表，属于虚拟的低交互蜜罐。在实际应用中常作为产品型蜜罐或作为高交互蜜罐的补充来进行信息搜集。
  - 在一台主机上就能模拟多个虚拟网络主机的后台程序。通过一些简单的配置，这些虚拟主机对外就好像是在运行不同的操作系统并且提供各种不同的服务。
  - Honeyd允许为每一台虚拟主机配置IP地址，并且外界可以通过ping、traceroute等操作来确定这些虚拟主机的存在。它可以提供任意的路由拓扑，模拟延迟、丢包等操作。
  - 在充当网络诱饵，蠕虫监测，遏制垃圾邮件等方面都有很好的应用。但是它只提供网络级别上的模拟，不能提供真正的操作系统作为交互环境，所以搜集到的信息有限。

- 蜜罐采用的主要技术有：
  - 网络欺骗技术
  - 端口重定向技术
  - 数据控制技术
  - 数据捕获分析技术

- 网络欺骗技术
  - 是指在一个严格控制的环境中，利用各种手段，诱骗攻击者对虚构的系统进行攻击，并为攻击者提供其认为可信的对话信息，从而保护实际运行的系统免受攻击，并实现数据收集功能。

- 端口重定向技术
  - 在工作系统中模拟一个非工作的服务功能，将恶意的、未经授权的活动重定向到蜜罐系统中。
    - 如在Web服务器上，模拟并将FTP服务重定向到蜜罐系统中去，从而在网络中虚拟出该工作系统对外提供FTP服务。
  - 当发现非预期流量、已知攻击或发现被监控主机上有未授权的活动时，便将有关流量重定向到对应的蜜罐中。

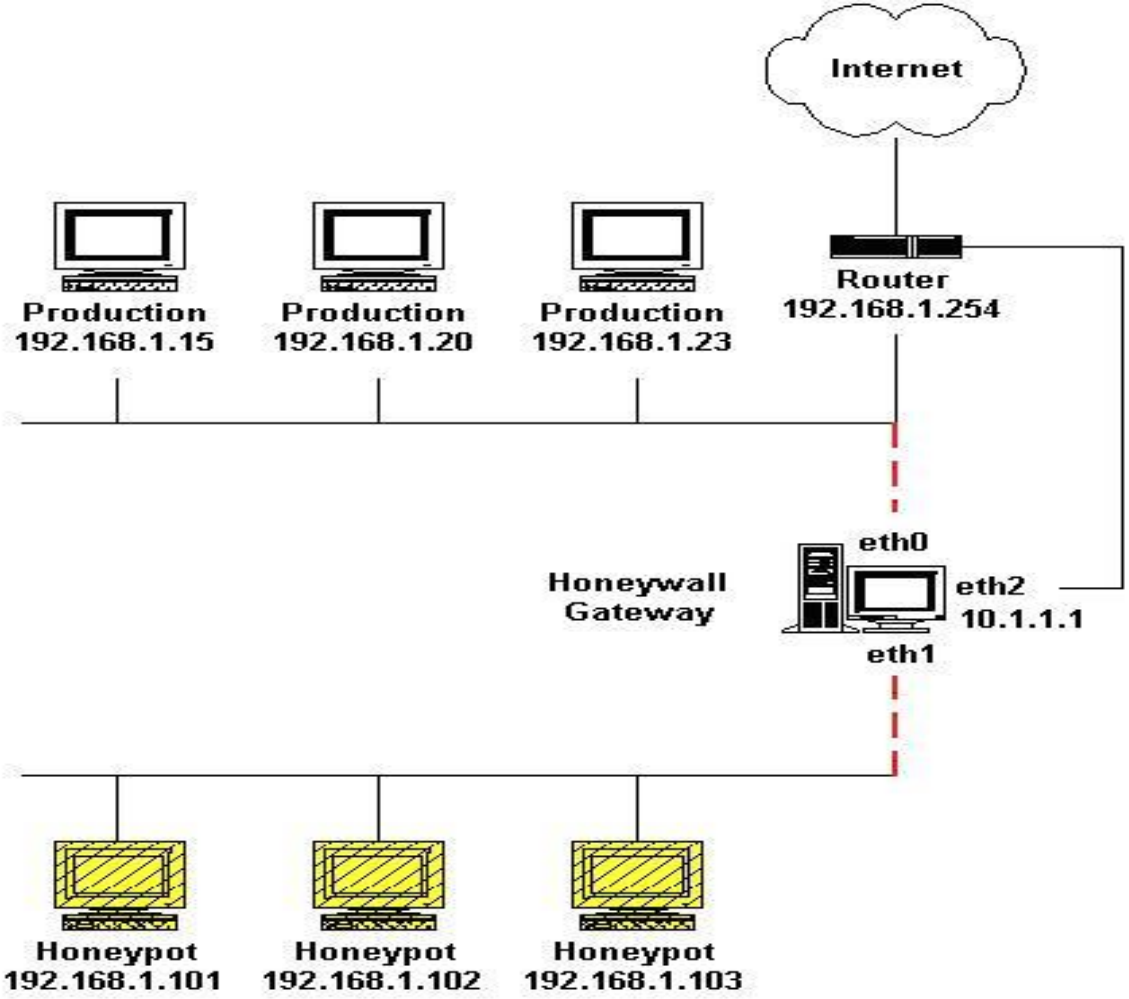
- 数据控制技术
  - 是指对蜜罐系统的连接控制和路由控制。
  - 防火墙实现连接控制
    - 允许所有外部数据包进入蜜罐，但对蜜罐主机的对外连接进行追踪限制；
  - 路由器实现路由控制
    - 防止基于蜜罐主机IP的跳转攻击。

- 蜜罐技术作为一种检测和信息搜集手段与传统技术相比具有如下优点：
  - 较小的数据量
  - 低误报率
  - 低漏报率
  - 适用于加密环境
  - 适用于IPv6
  - 高度灵活的实现方式
  - 资源最小化
  - 技术简单
- 所有其他的技术一样，蜜罐也有自身的缺点，因此人们常常将蜜罐与IDS、防火墙等技术配合使用
  - 风险
  - 视野受限

- 蜜罐技术作为一种检测和信息搜集手段与传统技术相比具有如下优点：
- （1）较小的数据量　传统的检测技术采用旁路监听方式，每天都要产生庞大的数据记录。蜜罐系统只记录访问到自身的数据信息，产生的数据量不大，而且这些数据具有很高的真实性和研究价值。
- （2）低误报率　传统的检测技术最大的问题就是产生误报。由于蜜罐系统不对外提供任何服务，对蜜罐的一切访问都一定是可疑的，因此蜜罐系统大大的降低了误报率，提高了检测攻击的效率。
- （3）低漏报率　传统的检测技术很难检测到未知类型的攻击。而蜜罐的与误用检测(misuse detection)和异常检测(anomaly detection)等IDS检测手段都有本质的区别，它不依赖于已知的攻击类型和网络状态，它可以在第一时间发现新的攻击类型。
- （4）适用于加密环境　目前网络上越来越多的数据采用加密传输（如SSH、IPsec、SSL），传统的检测手段无法处理这样的加密数据。蜜罐一般作为终端与攻击者进行交互，所以可以获得解密后的数据。
- （5）适用于IPv6　当前许多安全技术都是针对IPv4设计的，因此不能处理IPv6数据。蜜罐可以用于所有IP环境，不受IP协议影响。
- （6）高度灵活的实现方式　蜜罐的实现形式多种多样，甚至可以是一个信用卡号码或数据库的一条数据，所以蜜罐具有很高的适应性，在许多环境中可以使用。
- （7）资源最小化　使用虚拟方式实现的蜜罐只需要很少的资源，例如著名的虚拟蜜罐Honeyd，在一台奔腾机就可以管理上万个IP地址。
- （8）技术简单　蜜罐系统的实现相对简单，不需要复杂的算法和特殊的技术，便于安装和部署[10,18,20]。
- 和所有其他的技术一样，蜜罐也有自身的缺点，因此人们常常将蜜罐与IDS、防火墙等技术配合使用，蜜罐技术的缺点如下：
- （1）风险　蜜罐需要与攻击者进行交互，一旦蜜罐被攻克它很可能会被作为跳板去攻击其他的非蜜罐系统。风险的高低取决于蜜罐与外界的交互程度及蜜罐的实现方式。
- （2）视野受限　由于蜜罐只能记录访问到它的信息，不像IDS可以对整个网络的流量进行检测，因此单一的蜜罐存在视野狭小的缺点[18,20]。

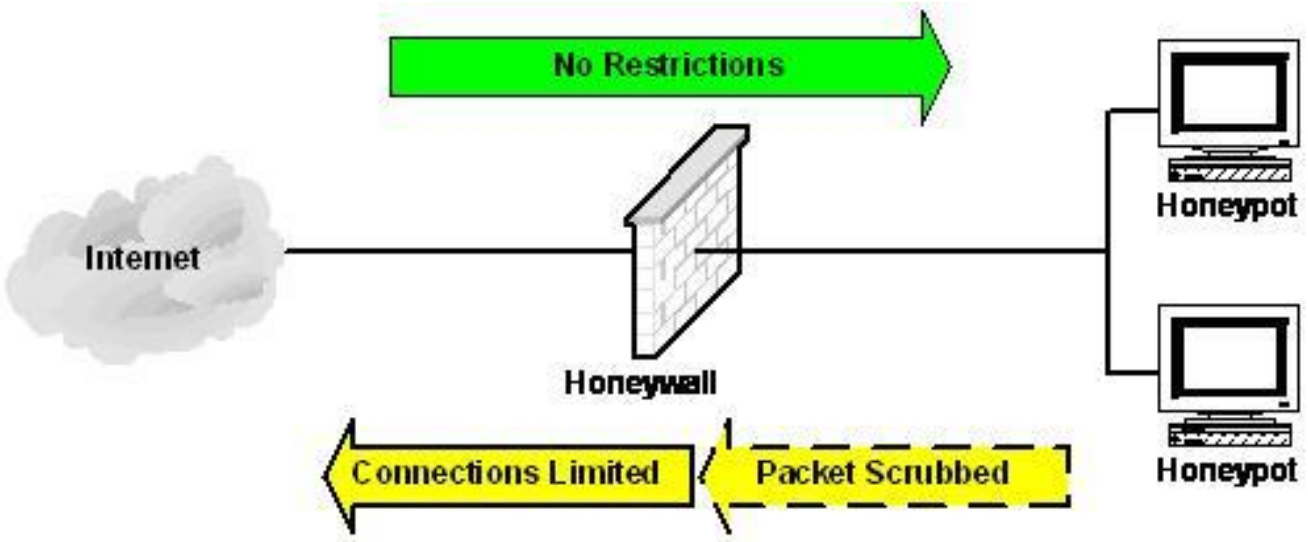
- 蜜网技术
  - 技术实质上仍是一种蜜罐技术
    - 蜜网是一种高交互型的用来获取广泛的安全威胁信息的蜜罐，高交互意味着蜜网是用真实的系统、应用程序以及服务来与攻击者进行交互
    - 蜜网是由多个蜜罐以及防火墙、入侵防御系统、系统行为记录、自动报警、辅助分析等一系列系统和工具所组成的一整套体系结构，这种体系结构创建了一个高度可控的网络，使得安全研究人员可以控制和监视其中的所有攻击活动，从而去了解攻击者的攻击工具、方法和动机。

- 蜜网的体系结构 （Honeynet ）

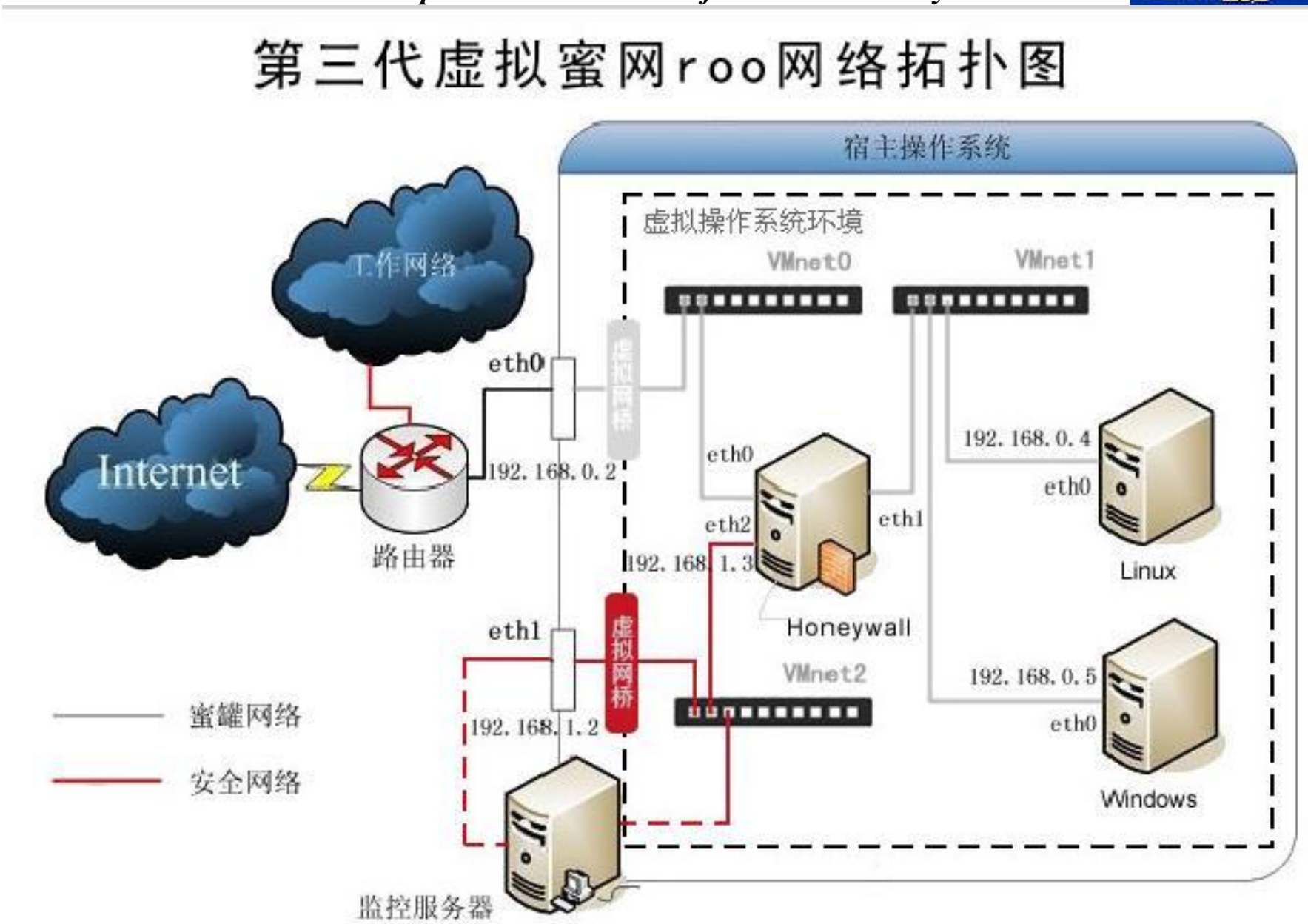


- 蜜网的关键是蜜网网关（ Honeywall ）
  - 蜜网网关将蜜罐所在的网络与正常网络相隔离。
  - 所有进出Honeynet的通信必须通过这个Honeywall，这样Honeywall就变成Honeynet的命令和控制中心。
  - 通常Honeywall是一个两层的网桥
- 蜜网技术
  - 蜜网的数据控制
  - 蜜网的数据捕获
  - 蜜网的数据分析

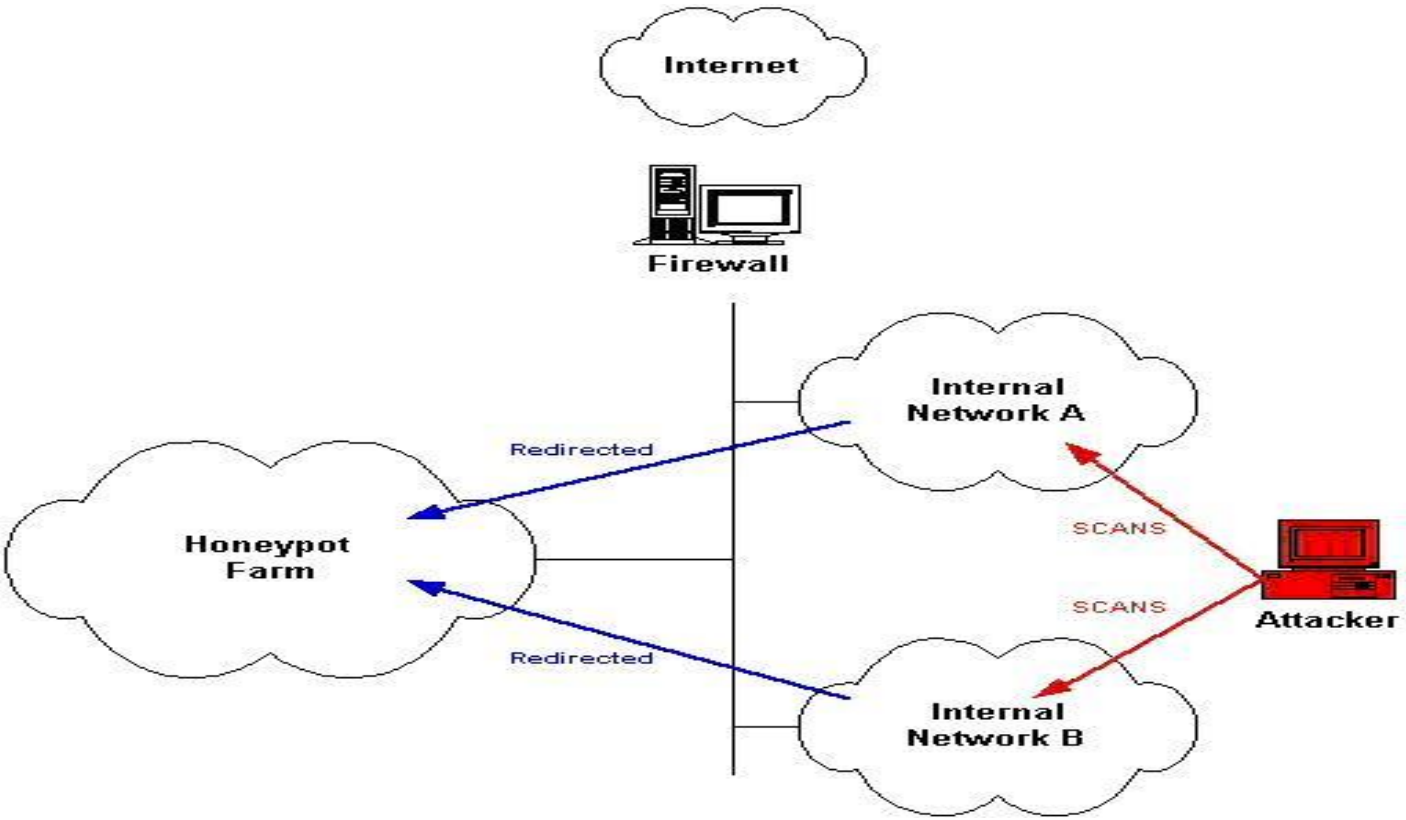
- 蜜网的数据控制
  - 连接计数：限制攻击者可以从Honeynet发起的外出连接次数
  - NIPS：网络信息防御系统，识别和阻挡已知的攻击
  - 防火墙：黑名单、白名单、防护名单等功能



- 蜜网的数据捕获
  - 目的是记录攻击者所有的活动，收集信息是蜜网系统的最终目的。
  - 数据捕获的关键是在尽可能多的层次收集信息。
    - 快速数据通道
      - 系统将来自网络和系统的信息融合成统一的格式化的形式保存到数据库中；
    - 而慢速数据通道
      - 是指将最原始的未经人工分析提取过的网络数据包进行保存。


第三代虚拟蜜网roo网络拓扑图
宿主操作系统
虚拟操作系统环境
VMnet0
VMnet1
VMnet2
工作网络
Internet
路由器
eth0
192.168.0.2
虚拟网桥
eth0
eth2
eth1
192.168.1.3
Honeywall
192.168.0.4
eth0
Linux
eth1
虚拟网桥
192.168.1.2
192.168.0.5
eth0
Windows
监控服务器
蜜罐网络
安全网络

- 蜜场技术
  - 起源于分布式蜜网
    - 它的工作原理是：将所有的蜜罐都集中的部署在一个独立的网络中，这个网络成为蜜场的中心；
    - 在每个需要进行监控的子网中布置一个重定向器（Redirector），重定向器会将本网络中发生的攻击信息通过某种保密的方式重定向到蜜场中心；
    - 蜜场中心选择某台蜜罐对攻击信息进行响应，并且利用一些手段对攻击信息进行收集和分析。


Internet
Firewall
Internal Network A
Redirected
Honeypot Farm
SCANS
SCANS
Attacker
Redirected
Internal Network B

- 关键问题
  - 数据收集：所有捕获到的数据收集到蜜场中心。数据的关联融合问题，以及数据收集机制的安全性问题。
  - 重定向器（Redirector）的设计
    - 什么样的数据需要重定向？
    - 重定向的数据与蜜场中的蜜罐如何对应？
    - 怎么样保证将攻击数据发送给他正好想攻击的类型的机器上？
    - 如何透明的将攻击数据传送到蜜场中？

- 蜜罐的应用
  - HoneyRobot僵尸网络
    - 低交互方式蜜罐利用程序模拟僵尸网路的命令通信协议，伪装成僵尸程序联入僵尸网络。
    - 高交互方式蜜罐直接在蜜罐上种植相应的僵尸程序，通过监控网络流量以及程序行为，来跟踪僵尸网络

基于良性蠕虫的主动遏制技术

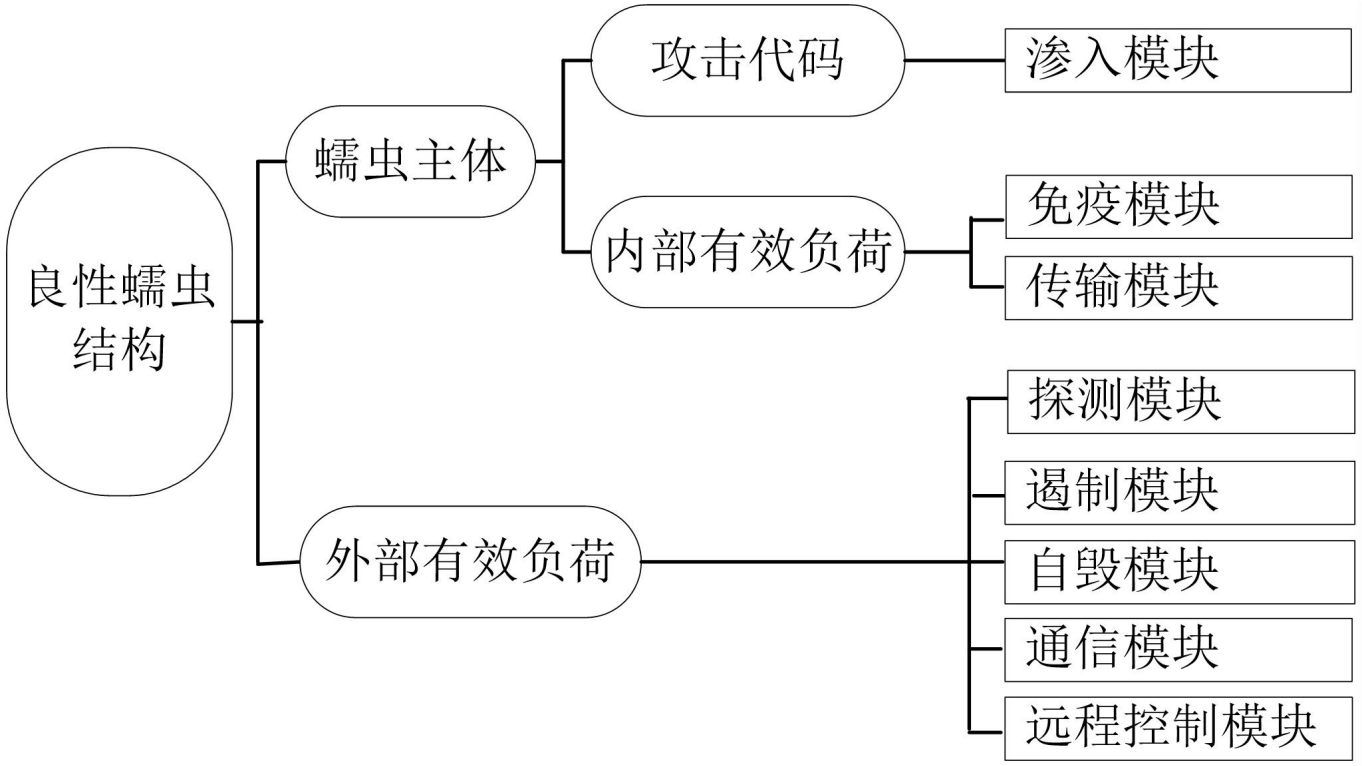
- 良性蠕虫主动遏制模型
- 良性蠕虫传播模型的研究

良性蠕虫主动遏制模型

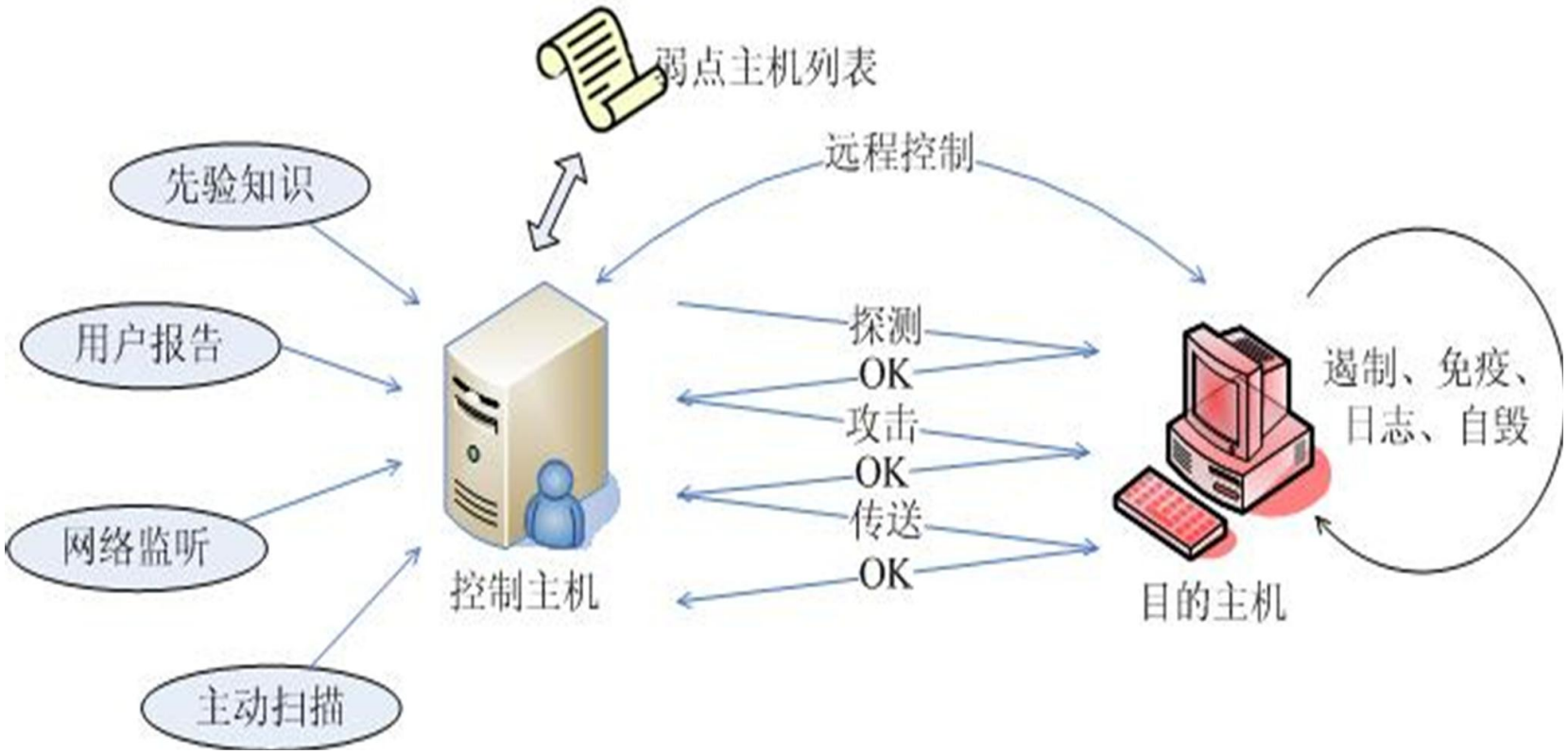
- 良性蠕虫拥有和恶性蠕虫相同的特点——代码的自动传播性，它们的最大区别在于良性蠕虫是用来对已经感染恶性蠕虫的主机进行修复、清除恶性蠕虫，对可能受到感染的漏洞主机进行免疫，并且在扩散的过程中要保障主机和网络资源能够正常被用户使用，而恶性蠕虫却可以肆无忌惮地耗费网络资源。
- **定义：良性蠕虫**是一段可控的无须计算机使用者干预即可运行的独立程序，它通过获得网络中存在漏洞的计算机上的部分或全部控制权，然后利用传输功能获得辅助工具，来完成免疫、修复、清除蠕虫和关闭后门等任务；最后能够安全自毁。

- 良性蠕虫具有如下性质：
  - 对抗性：能够遏制已感染蠕虫主机对网络带来的危害（如终止蠕虫进程、关闭后门、删除副本和修复注册表等），能够免疫、防护易感染主机存在的隐患（如主动修补漏洞、设置蠕虫免疫体和向用户预警等）。
  - 安全性：包括代码的安全性、扩散过程的安全性。前者主要表现在代码本身不应存在用于对抗恶性蠕虫以外的其他操作，如窃取信息、种植后门等，良性蠕虫在修复主机漏洞前应经过完备的测试，对可预计的恶性影响制定响应策略。后者主要表现在良性蠕虫代码不会被恶意利用，例如良性蠕虫在完成既定的任务后应该自毁等。
  - 可管理性：蠕虫的设计者在一定程度上可以对其产生的效果进行干预，包括对蠕虫的传播范围、影响时间进行预先的限定，在传播过程中能够进行异常安全终止等。
  - 合理开销：渗入主机后应尽量少地占用系统资源，如CPU、内存等；在扩散的过程中，应尽量避免对网络流量造成冲击。

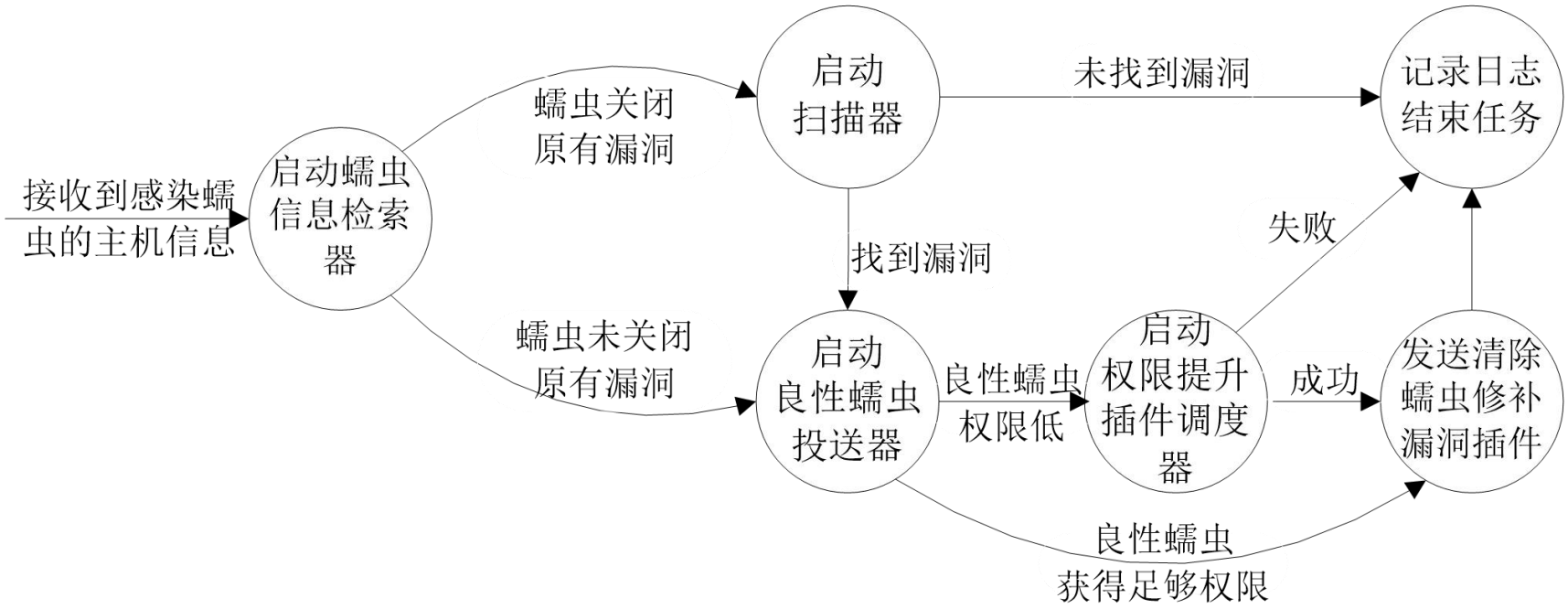
- 良性蠕虫的功能结构


良性蠕虫结构
蠕虫主体
攻击代码
渗入模块
内部有效负荷
免疫模块
传输模块
外部有效负荷
探测模块
遏制模块
自毁模块
通信模块
远程控制模块

- 良性蠕虫的工作机制


弱点主机列表
远程控制
先验知识
用户报告
网络监听
主动扫描
控制主机
探测
OK
攻击
OK
传送
OK
目的主机
遏制、免疫、
日志、自毁

- 漏洞与后门
  - 权限提升——强特权良性蠕虫
- 强特权良性蠕虫投送方法



- 通过建立权限提升插件关联库，可以针对某一初始权限生成攻击路径，从而可以很方便地组建比蠕虫攻击能力更强的良性蠕虫——强特权蠕虫
  - 主机权限更高
  - 传播权限更高

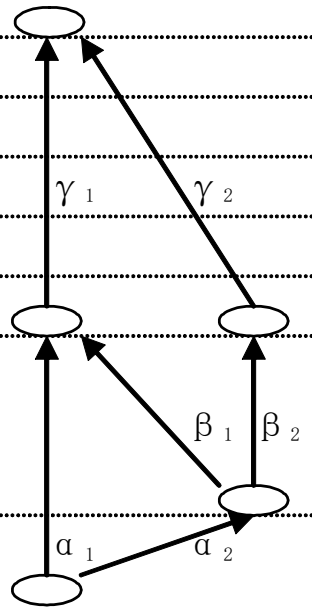
- 以MSBlaster和Welchia为例。

MSBlaster 的权限提升路径 $\pi_1 = (p_0, \alpha_1, p_2, \gamma_1, p_3)$

Welchia 的权限提升路径 $\pi_2 = (p_0, \alpha_2, p_1, \beta_1, p_2, \gamma_1, p_3)$

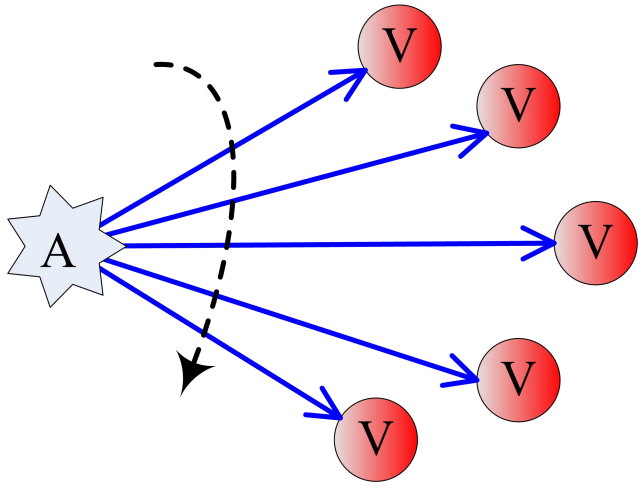
Welchia 的权限提升路径 $\pi_3 = (p_0, \alpha_2, p_1, \beta_2, p_2, \gamma_2, p_3)$

权限层次	
8 root plus	
7 root	p_3 = 获得 root 权限
6 某特权拥有者	
5 其它普通用户	
4 本地用户	γ_1 = Exploit DCOM RPC vulnerability 插件
3 无 shell 帐号用户	γ_2 = Exploit WebDav vulnerability 插件
2 获得端口开放状态	p_2 = 获得端口开放状态
	β_1 = TCP135 端口探测插件
	β_2 = TCP80 端口探测插件
1 获得主机存活状态	p_1 = 获得主机存活状态
0 Internet 公开服务访问者	α_1 = TCP135 端口探测插件
	α_2 = ICMP 请求插件
	p_0 = 起始特权层



良性蠕虫的主动对抗模型

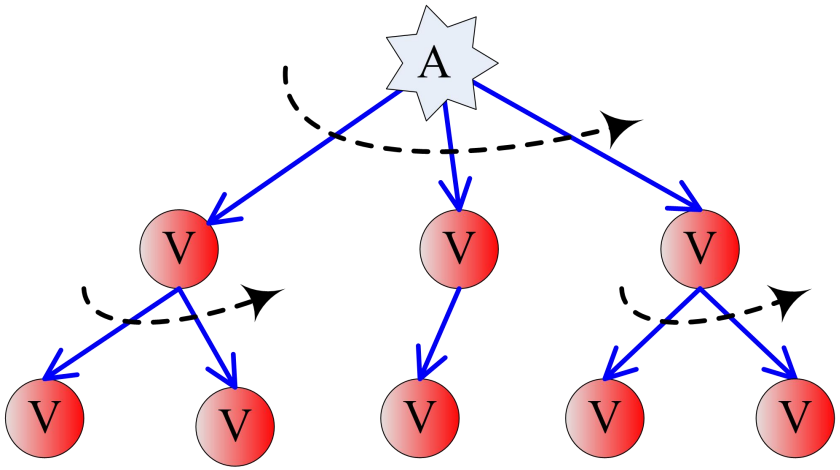
- 集中式对抗模型
  - 一个网络中存在一个或者多个蠕虫主动遏制服务器，通过主动扫描的方式对管理范围内的主机依次进行探测，当探测到漏洞主机后，向目标主机投放良性蠕虫。



  - 可控性强 、可追踪性强、灵活性强
  - 网络管理范围有限、扫描带来额外网络负载、漏洞主机搜索速度慢

- 监测驱动式遏制模型
  - 利用网络旁路侦听等技术对网络中的流量进行监测，从而根据蠕虫的特征发现感染的主机和漏洞主机的IP地址，实施遏制。
  - 监听网络，截获恶性蠕虫，替换恶性蠕虫的攻击模块为免疫模块，放出修改后的蠕虫
  - 无须主动扫描，可以获得网络中感染蠕虫主机的信息，从而避免了对网络流量冲击；②使良性蠕虫工作更有针对性，提高了良性蠕虫传播的可追踪性以及可控性
  - 所管理的范围受限于网络监测器所侦听到的网络流量，无法管理整个Internet。

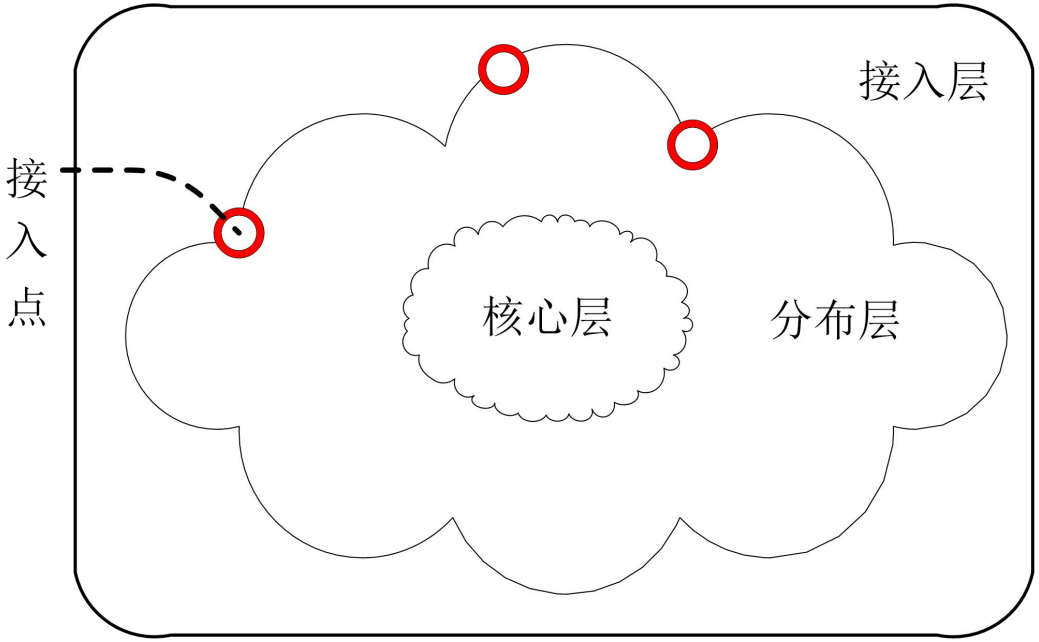
- 主动扩散式对抗模型
  - 像蠕虫一样在网络中自动寻找漏洞主机、渗入目标主机、传播副本及修复工具



  - 利用蠕虫自动传播的优势，迅速遍历网络中主机，从而使在蠕虫爆发初期或蠕虫爆发前和恶性蠕虫进行对抗成为可能，从而将蠕虫疫情控制在最小范围内。
  - 也会给网络带来流量冲击。

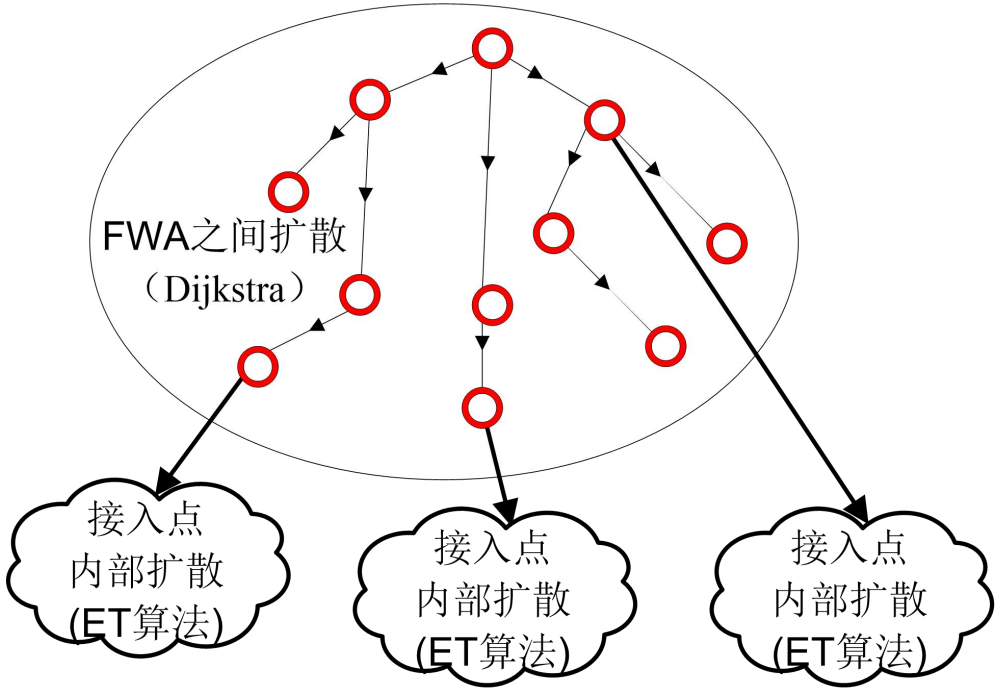
良性蠕虫的主动扩散策略

- 多级分层的拓扑结构
- 一般包括核心骨干层、汇聚层和接入层
- 以往蠕虫一旦爆发，接入点的出入口以及核心层、分布层都将受到严重的流量冲击



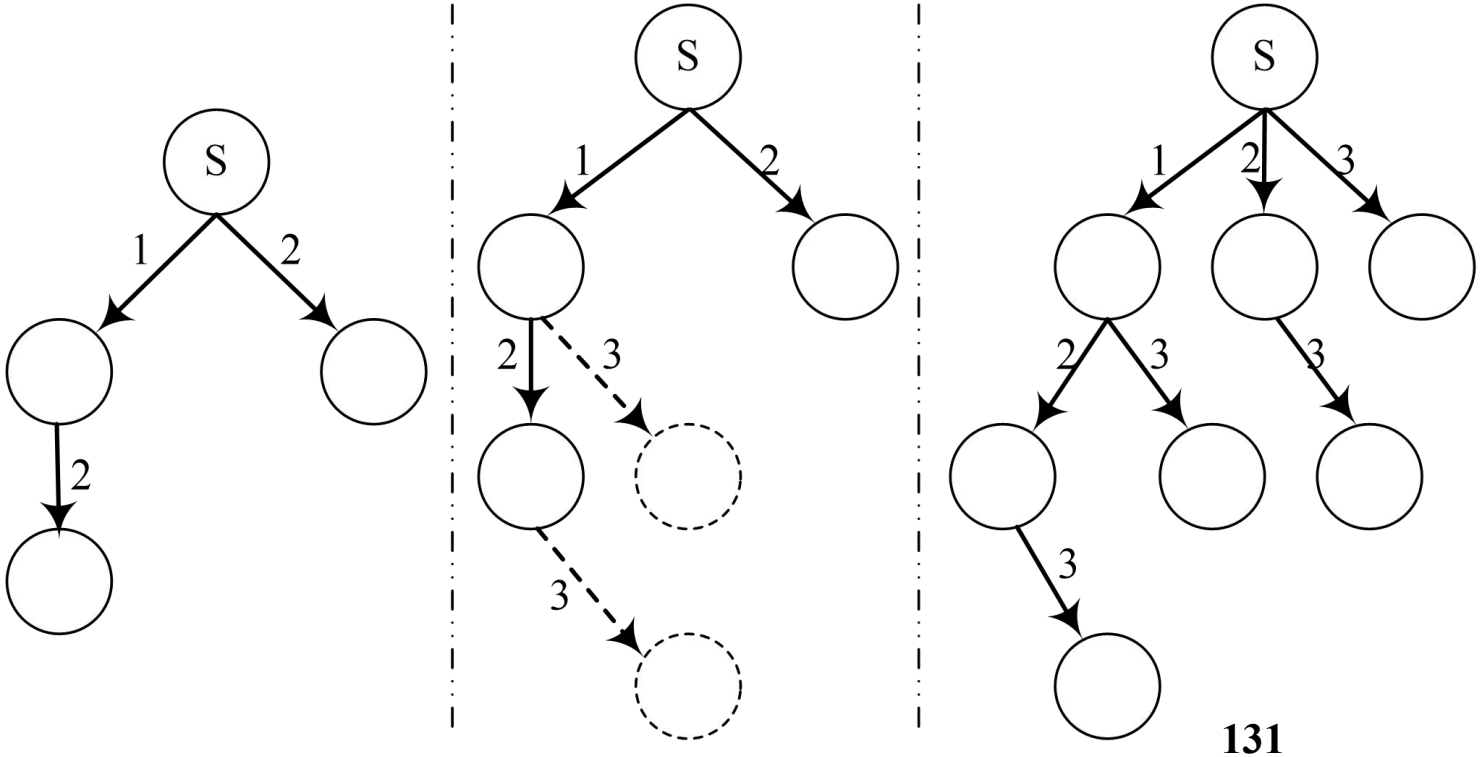
良性蠕虫的主动扩散

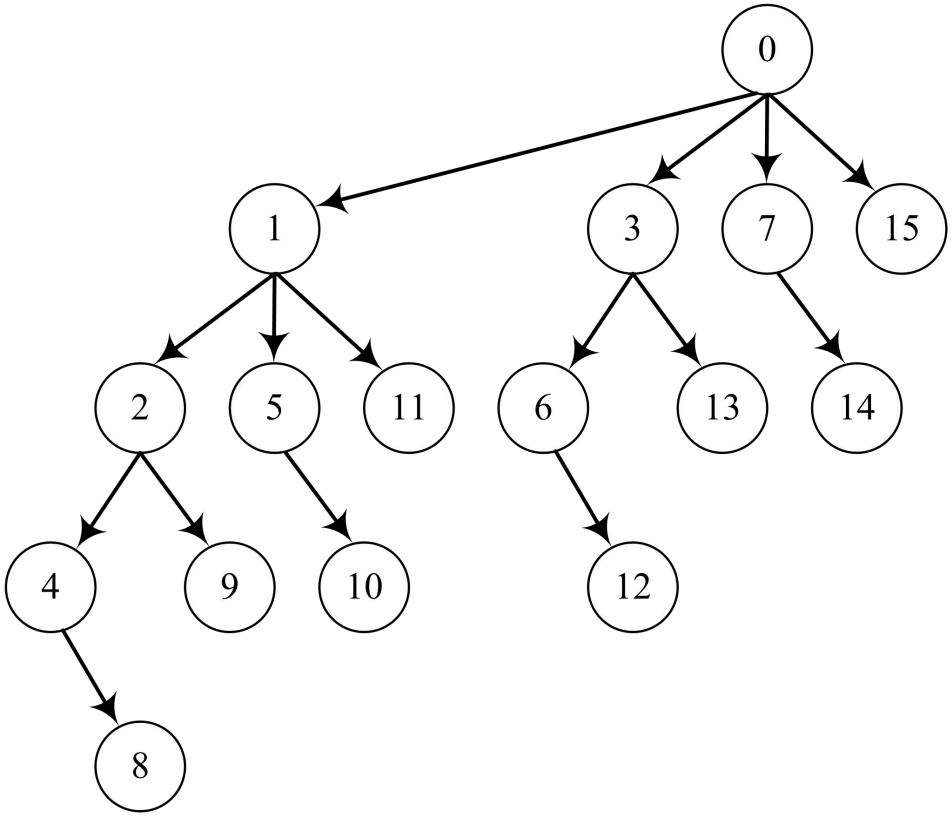
- 根据网络拓扑结构层次建立分级扩散机制
- 对骨干网络和接入层出口影响最小



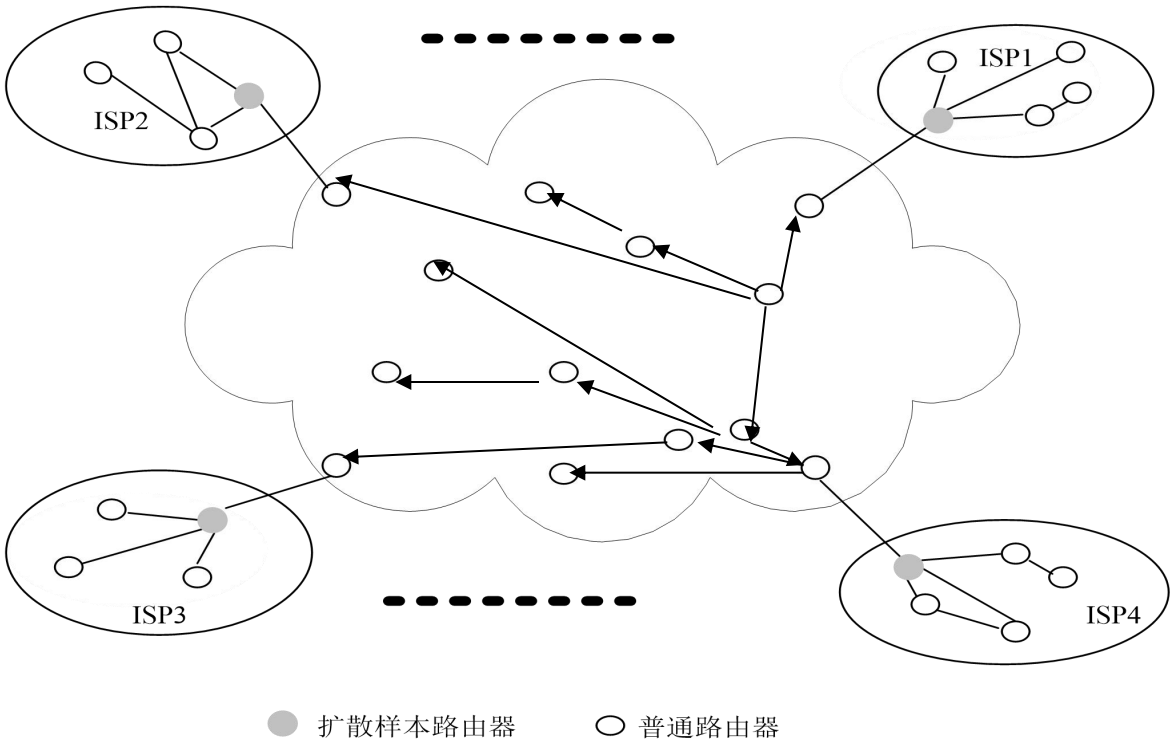
指数树的扩散方法

- 扩散延迟为*N*+1。 N为扩散树的阶数



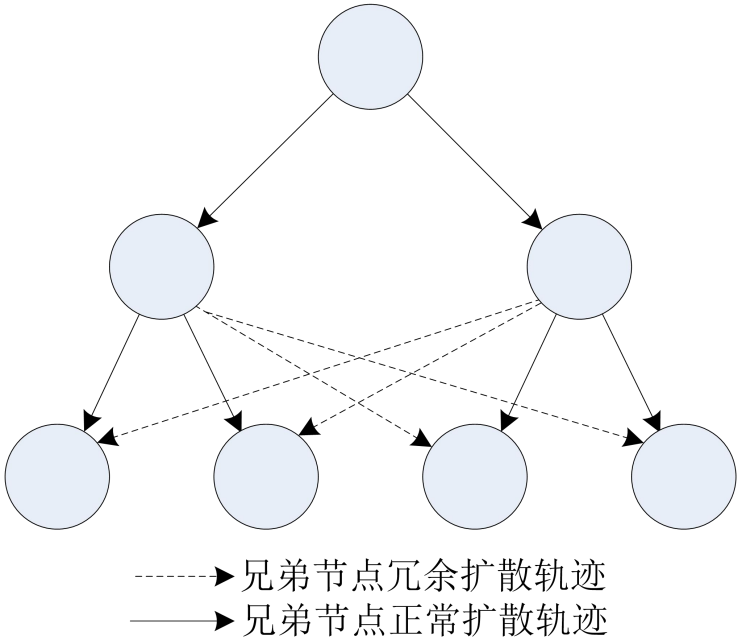

0
1
3
7
15
2
5
11
6
13
14
4
9
10
12
8

轻负载扩散算法

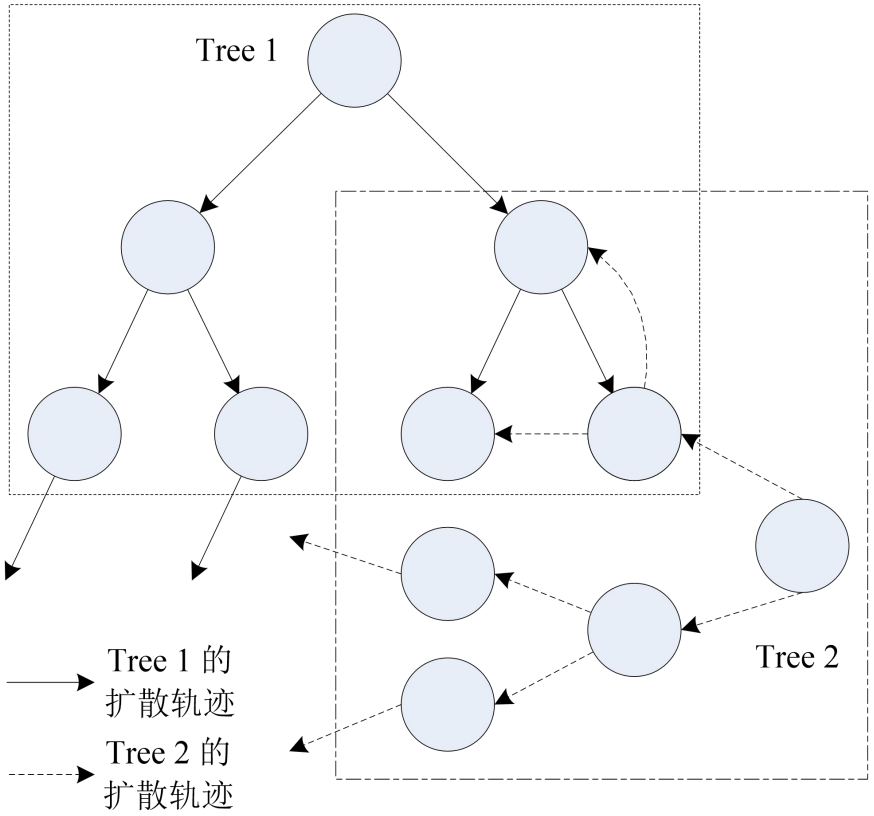


扩散树的稳定性问题

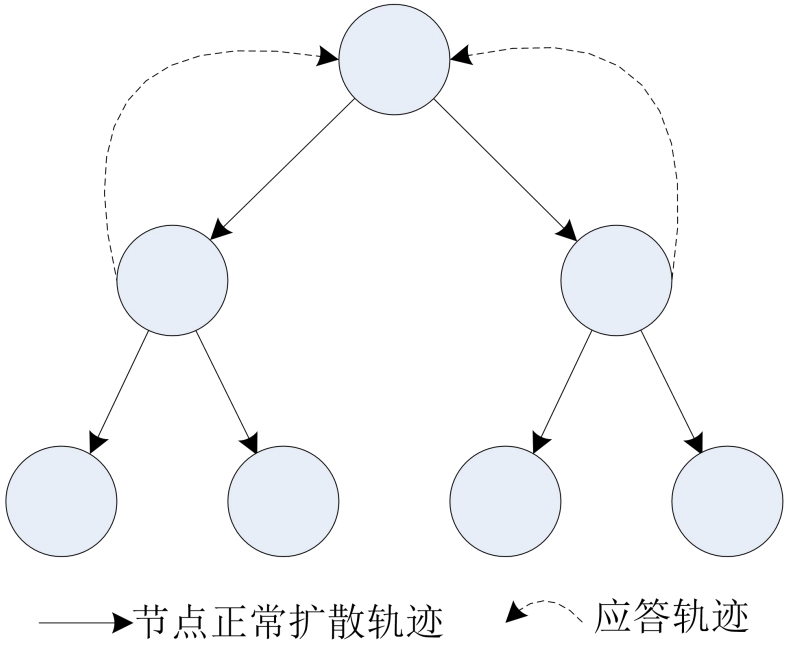
- 冗余扩散方法
  - 在扩散树中使每个节点获得重复的感染机会，从而当其中一个父节点成为坏点，其仍然可以有机会被感染
- 应答检测方法
  - 通过在扩散树中添加应答策略，使每个节点认识到它的后代节点是否被感染成功，如果发现未成功，则可以采取一定的补救措施



兄弟节点互助法



多树交错法



应答检测法

END