Week 1 Intro

Definition

Distributed System:
A collection of independent computers that appears to its users as a single coherent system.

Middleware:
A distributed system is a middle wear layer between applications and local OSs of multiple individual machines and offer each applications the same interface.

Challenges

Scalability: the ability for the system to preserve some properties as system grows.
Consistency
Fault Tolerance: the ability for the system to recover from partial failures (so it continues to appear as single running system to users)

DNS example (from textbook)
Consistency example (week 4: how to make sure all computers in the system follows the same time.)

Week 1 Reading

Week 2 Tut

Thread lifecycle:

1. NEW: when instantiate a thread ~start()
2. RUNNABLE: executing its task
3. Waiting : ~wait() and sleep(). When other thread signals, this thread goes back to its runnable state.
4. DEAD: after the thread finishes its task.

Week 2 Reading & Lecture

Logical organization of components in distributed systems

- Components: modular unit with well-defined interfaces that is replicable
- Connector: mechanism that medicates communication, coordination among components.

Event-based architectures: communication through events, optionally carry data
Data-centred architecture: through a shared repository that contains data.

Centralised architectures

UDP: connectionless protocol (No need of resource between server and the client); TCP: connection oriented protocol.

Stateless and stateful server
Stateless: does not record the state of its clients.
Stateful: maintains persistent information about its clients.

Client-server model

- Thinking in terms of clients that request services from servers.
- A server is a process implementing a specific service
- A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.
- Connectionless protocol: efficient, but client doesn't know whether the message has been lost and if sending it twice will result in duplicate operation.

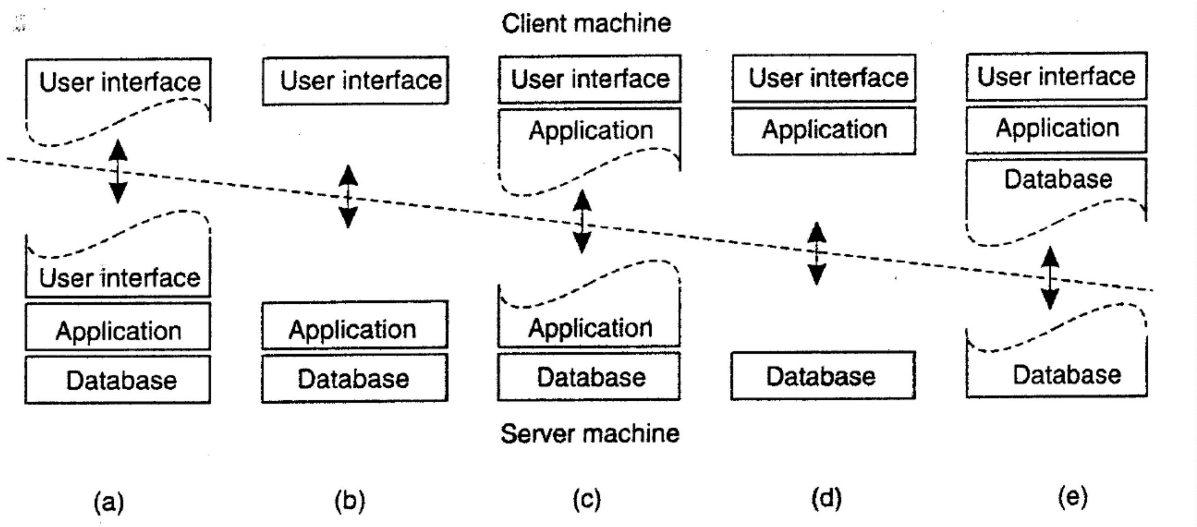
- Reliable connection oriented protocol: low performance, TCP/IP connection. When client requests a service, it first set up a connecting to the server before sending the request. Server uses that same connection to send the reply message. Cost is the setting up/ tearing down of that connection.

Application layering

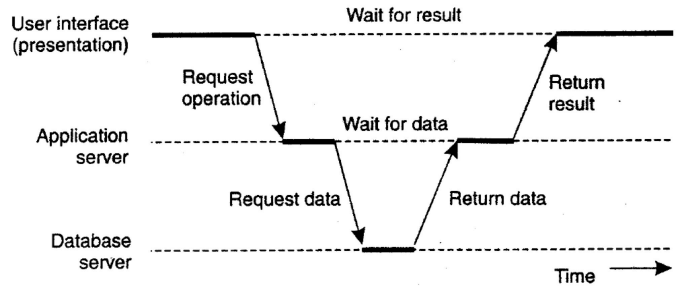
- Client server applications can be constructed from 3 different pieces: interaction layer, database layer, operation/logic layer.
- In the client-sever model, the data level is implemented at the server side. Usually relational database/object-oriented/object relational database

Multitiered architectures

- We could have many options where to put each application on the client side.



We want to make the client side as thin as possible so preferably move to a-c as a trend.
Server acting as a client: we may have application server separated from database server.



Structured P2P architectures (Peer 2 Peer)

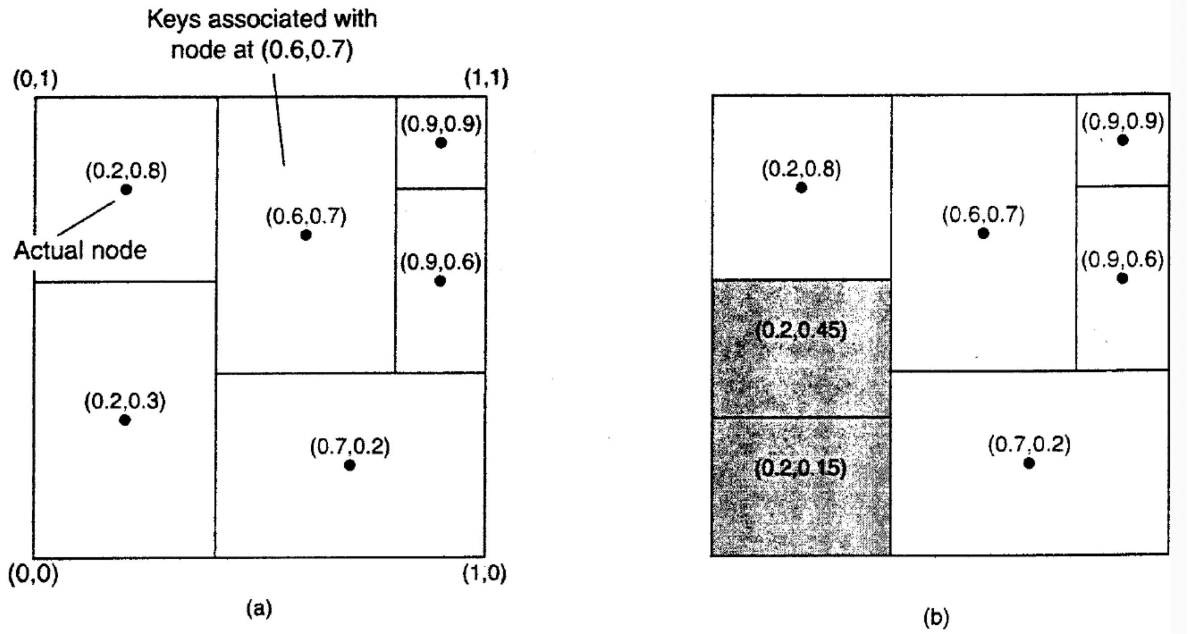
- Every machine can be a client and a server
- No centralised control: responsibility is distributed evenly.
- Overlay network is constructed using a deterministic procedure: distributed hash table.
- Data items are assigned a random key from a large identifier space such as a 128 bit or 160 bit identifier.
- Nodes in the system also has a key. Nodes: processors, links possible communication channels.

DHT
问自己两个问题: how to join, how to leave.

- Data item with key k is mapped to the node with the smallest id k, denoted as succ(k)
- To look up at the data item, application running on an aibitary node call function lookup(k), returns network address of succ(k)
- Membership management: how nodes organise themselves into an overlay network.
- Each node will maintain shortcuts to other nodes in O(logN) time. N is the number of nodes participating in the overlay.

CAN:

- D-dimensional cartesian cooridinate space.
- Each node has a coordinate and responsible for that data falls into that region.



- How a node joins:
- How a node leaves:

Justin doroth

Hybrid architectures: Edge server system

- Servers are placed at **the edge of the network** (补充 what is the edge of the network?)
- Edge servers serve content, after applying filtering and transcoding functions
- One edge server acts as an origin server for all content, while taking advantage of other servers for optimised delivery of content/replication.

Collaborative distributed system

How does BitTorrent work? Goal: quickly replicate large file to large number of clients.

- First, download bitten file. (You join in the distributed network)
- First establish by using a traditional client server models. Once joined fully, decentralised schemes can be used.

补充: 理解 bit torrent 的工作方式

- Client node -> webserver -> File server -> Trackers
- Why files are divided into chunks? (256kB-1Mb)
    - primary reason: **increase the availability of content** in the distributed environment.
    - secondary reason: prevent bottleneck of your speed.
    - Since there is no controller.

Conclusion:
Concept of the client and server
Client and server may run on: same machine/distinct machines with very different resources/with similar resources

Client-Server Blockchain. Please start working on it.
Soft deadlines: Task1: Friday of week 2; Task2: Sunday of week2; Task3: Sunday of Week 3/

Processes

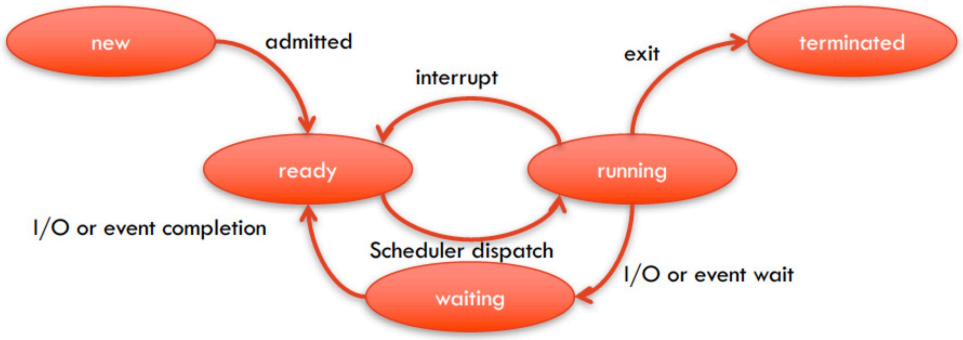
Uni-programming: one program execution at a time. No longer acceptable.
Multiprogramming: More than one program execution at a time: UNIX/Linux, Mac..

Concurrency:
How to give the illusion to the users of multiple CPUs?
Solution: **scheduling** program executions, one after the other during small fractions of the time. 比如每一个做 0.1 秒. 人察觉不出来的..

Network operating systems: machines provide resources to other machines.
Middleware: a layer over the network services providing

Process

一种抽象的程序执行过程.
defined as a program in execution. Operating system creates a virtual CPU for each process. OS makes sure that processes cannot maliciously or inadvertently affect each others behaviours. Concurrent transparency comes at a very high cost.
这个图要会画



Address space
Stack – Heap -Data section (global variable) - Text (program code)
a unit of management of a process's virtual memory.

Process management

- Creation:
  - First allocation: need to choose the host of process.
  - Execution environment: copies address space, initialized content and open files: static involves program text, heap and stack regions created from a list. Dynamic: Unix fork shares program text and copies stack & heap regions.

- **IPC (Interprocess communication): Pipe/Message Queue** / Requires costly context switches.

Switching from one process to another has become one of the most expensive operations due to context switches.

Light weight process: LWP
LWP does not block upon system calls.

Thread

Smallest unit of CPU utilization.

- POSIX: Portable OS Interface.
- Only copies current activity (program counter) & register/stack data. No data/code/files are copied.

Communication is through memory. No context switches needed.

User level threads library
Managed by User

1. Cheap to create and destroy
2. Blocking system call
3. Cheap context-switch: few instructions to store and reload CPU register values.

4. No need to change memory map in MMU(Memory Management Unit) and flush the TLB (Translation Lookaside Buffer)

Kernel-scheduled threads

Managed by OS

1. Costly to create and destroy
2. Do not block the current process upon blocking system calls.
3. Costly context switch. (Needs to change MMU and flush the TLB)

Process 和 thread 的优势和区别

Process	Thread
Isolated, Inefficient. Prevents one process from interfering another. Starting/terminating a process is costly.	Non- Isolated, efficient. Needs programmer to handle interference. Lighter weight than process. Generally faster than a process, but shares its data with other threads

Java Threads

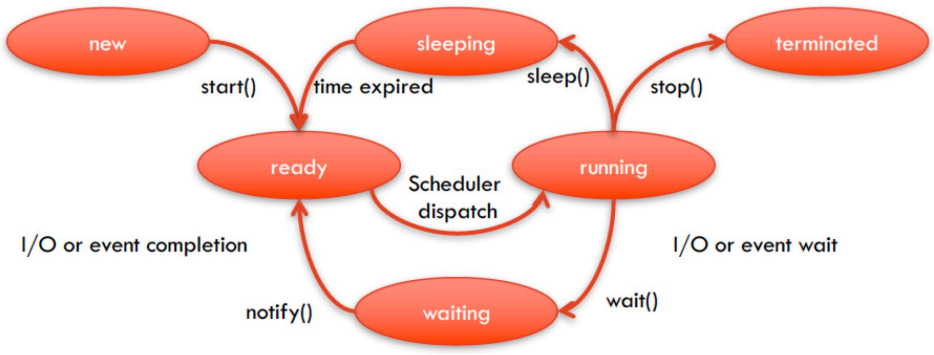
JVM: main thread. Responsible for executing VM operations
Periodic task thread: simulate timer interrupts
Garbage Collection thread: parallel and concurrent garbage collection
Compiler thread: runtime compilation of bytecode to native code.
Signal dispatcher thread: redirects process directed signal to Java level signal handling methods.
Graphical user interface: Java AWT and Swing
Java Servlets and RMI: pool of threads and invoke methods within these.

Thread API

Thread.join: blocking the calling thread until specified thread has terminated.
Thread.interrupt: cause it to return from a blocking method such as sleep()
Object.wait: blocking the calling thread until a call made to notify on objects wakes the tread.
Object.notify: wakes all threads that have called wait on object.

Lifecycle

new -> runnable -> blocked -> waiting -> sleeping -> terminated.



How to program thread:

- Write a class that implements runnable. So code within **run()** executes as a separate thread.

```
Class MyThread implements Runnable{
    public void run(){
        /* here goes my thread code*/
    }
}

Thread mt = new Thread(new MyThread()); //allocates a new thread.
mt.start(); //start it
```

- Also needs to .**start()** it which calls its **run()** method.

Multithreading

Thread safety: protecting data from uncontrolled concurrent access

- **Object state**: data stored in stat variables such as instances or static fields.
- **Shared variable**: variable that can be accessed by multiple threads
- **Mutable variable**: variable whose value may change.
- if more than one variable writes to a shared variable, synchronization needs to be handled, otherwise the program is broken.就是比如两个 thread 同时 x++ 但是没有 lock 的例子

三种解决线程安全问题的方法

1. Thread specific data

- Share 同一个 process 的 thread 才能 share data
- 在 java 里用 Threadlocal class, 设置成 static variable, static setter 和 getter
- `private static ThreadLocal errorCode = new ThreadLocal()`

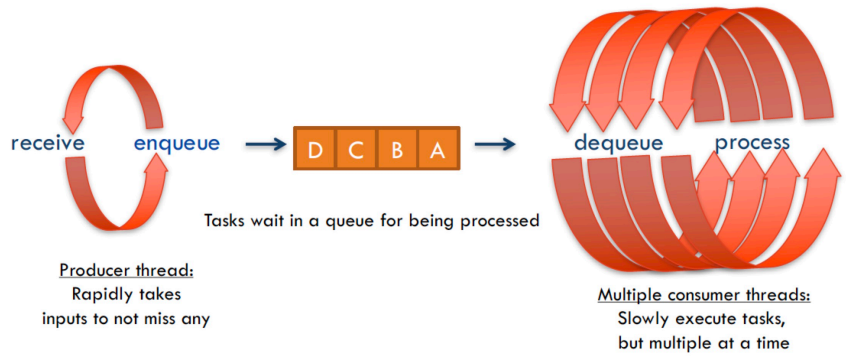
synchronization 技巧之 Producer/Consumer

适用范围:

1. 同时收到很多 task; 解决 task 需要一定时间; 结束之后需要解决所有 task; 在处理 task 的的时候需要记录在这期间收到的 task 请求
2. Asynchronous communication between threads.
3. Decouples the consuming and production

方法:

用一个 queue 和两个(多个)thread, 一个 thread 收 task 并放到 queue 里, 另一个(多个)thread 干活.



不用 messagequeue 的例子(用 sync 和 lock)

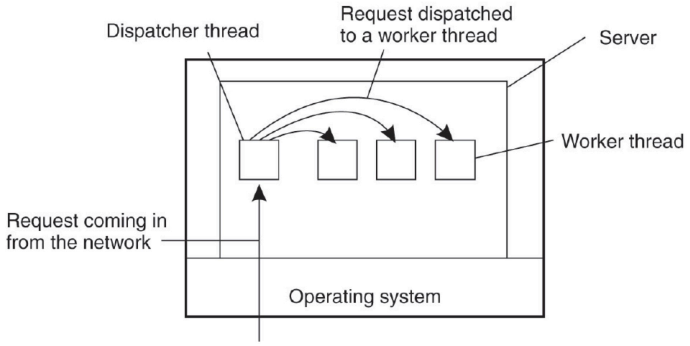
```
import java.util.ArrayList;
import java.util.List;
class Processor_PC{
    private List<Integer> list = new ArrayList<>();
    private final int LIMIT = 5; //start remove when hit 5.
    private final int BOTTOM = 0; //start to add when hit 0
    private final Object lock = new Object();
    private int value = 0;
    public void produce() throws InterruptedException{
        //add item
        synchronized (lock){
            while(true){
                if(list.size()==LIMIT){
                    System.out.println("Waiting for removing items from the list..");
                    lock.wait();//if we synchronised on the lock, not the class(this)
                    //then we need to call wait from the lock object.
                }else{
                    System.out.println("Adding: "+value);
```

```
                    //System.out.println("list size is "+list.size());
                    list.add(value);
                    value++;
                    lock.notify();
                }
                Thread.sleep(500);
            }
        }

    }
    public void consume() throws InterruptedException{
        //remove item
        synchronized (lock){
            while(true){
                if(list.size()==BOTTOM){
                    System.out.println("Waiting for adding items to the list...");
                    lock.wait();
                }else{
                    //System.out.println("list size is "+list.size());
                    System.out.println("Removed "+list.remove(--value));
                    lock.notify();

                }
                Thread.sleep(500);
            }
        }

    }
}


//使用者
public class ProducerConsumer {
    public static void main(String[] args) {
        Processor_PC processor = new Processor_PC();
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                try{
                    processor.produce();
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        });
        Thread t2 = new Thread(new Runnable() {
            @Override
            public void run() {
                try{
                    processor.consume();
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }
        });

        t1.start();
```

```
        t2.start();
        try{
            t1.join();
            t2.join();
        }catch (InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

Multithreaded server
三种手段

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite state machine	Parallelism, nonblocking system calls



逻辑和 consumer producer 差不多. 有一个专门的 dispatcher thread 在来找哪个 worker thread 是 idle 来分配

其他的不重要的实现方法:

1. Single threaded server: one loop iteration must treat a request before starting to treat another. Could have request loss.
2. Finite state machine: server treats request if requested info is in the cache. Otherwise doesn't wait for I/O to disk to complete but stores the state the request in a table. Uses non- blocking system calls.

两种利用 thread 的方法

1. Thread pool: Create a pool of threads/server can (re)use, 哪怕没有人用也开好
2. 类似 lazy loading 的思路, 来一个 request 才开一个 thread.

Week 3

两种类型的 communication 会考会考!!!

1. Persistent communication: 三个特点: 一个只管发 一个只管收, 东西存中间
    - Message is stored by communication middleware.
    - Sending application stop executing after submitting the message.
    - Receiving application also need not to be executing
2. Transient communication: 一个特点: 只有双方上线的时候才能传

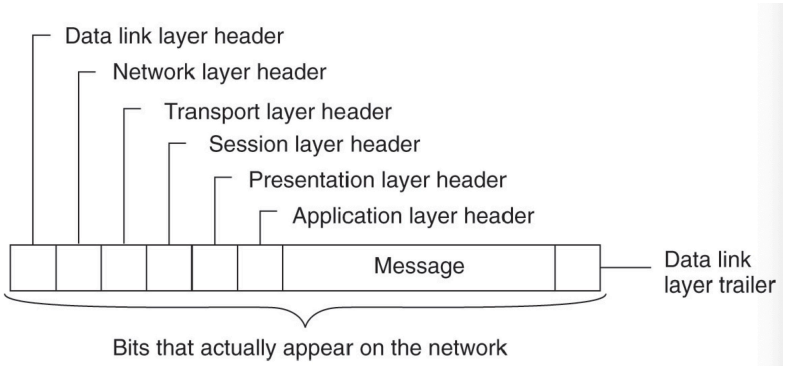
- Message is stored only for the durating when sending/receiving applications are executing. 只有双方都上线的时候才传文件.

记住七层 protocol

Connection Oriented	Connectionless
Establish explicitly a connection with a partner **before exchanging data**	No setup in advance needed
例子: TCP	例子:UDP
适用于: File Transfer, web browsing, email	适用于: Skype

Layers 重要

每一层都有自己需要解决的问题, 分开解决. 每一层都会给前一层的信息加上 header. 包住自己, 传给再下一层
Application
Presentation
Session
Transport
Network
Data link
Physical. (最底层)
APSTNDP



底层: PDN

Physical Layer: 0101010101
Data Link Layer:

- group bits into frames/assigns sequence numbers to frames;
- adds special bits at the beginning & end; adds a checksum;
- if receiver disagree with the checksum, asks the sender to resend.
- Example: VPN (virtual private network) LAN(local area network)

Network layer:

- Example: Internet protocol (IP), Wide Area Network (WAN)

中层: T(transportation)

Transport Layer:

- Split the message from the application layer
- Number them
- Add the amounts of sent and remaining packets

Reliable transport layer can be built on top pf:

- Connection-oriented protocol: TCP
- Connectionless protocol: UDP

TCP/IP 是 transport 和 network 的结合

高层 (Session, Presentation, Application

- Session: (DNS) dialog control, transfer check-pointing.
- Presentation: says how information is presented.
- Application: (HTTP, File Transfer Protocol, Telnet) Application has no clue of intermedinary steps.

例子

Email 和网页

Mail Application: post a letter with address and receiver reads it.
Web application: user types a URL and server send back the webpage.

Transport(Error control): Mail – if write a wrong address on letter, it will be send back to you; TCP – initialized a connection, check for potential errors and may resend.

IPV6 (Internet Protocol version 6)

Why IPV6?

1. Scalability in the number of addresses (IPV4 32 bits, IPV6 **128 bits)**
2. Scalability **of multicast address (组播地址)** : able to send a packet to any one of a group of nodes.
3. Efficient forwarding: changes in the way IP header options are encoded.
4. Greater flexibility for additional options.

Routing Problem

要会看 bellman-Ford algorithm, Dijkstra Algorithm

- **Routing is necessary** in all networks except for Local Area Networks (LAN) where ethernet provides direct communications.
- Routing is implemented in **network layer.**
- **Adaptive routing:** best route for communication between two points is **re-evaluated periodically** by the current traffic.

Graph node: internet routers; Edges: communication links
Dijkstra : shortest path algorithm

两种 algorithm

1. Distance Vector (RIP, bellman-Ford) neighbours exchange their routing table
2. Link-state(Dijkstra): neighbour only exchange connectivity information

Distance Vector

1. Router Information Protocol (RIP) – used up to 1979. The idea is to maintain a table representing the direction for each destination. Every 30 sec, or whenever local routing table changes, send T_L to all accessible neighbours.
2. Once recept a table T_R with a new destination or a lower cost route to an existing destination, update local table T_L.

Link State

- 用的是 Dijsktra: 一种带 weight 的 BFS

每次挑 queue 里 cost 最小的一个出来 explore 并加入 explored set.
如果邻居是第一次被发现, 加入 queue, 如果邻居已经出现过了, 按照是否更小来更新当前 node 邻居的 cost.
第一次出现/需要更新的话就需要记录这个 parent 节点. 然后找 path 就用从终点往前找的方式 trace 回去.

- "Open Shortest Path First": router floods OSPF link state advertisements to all other routers in entire AS. This message is directly covering IP.

Distance Vector (DV)	Link-state (LS)
Bellman-Ford	Dijkstra: single source shortest path to all other nodes.
Simle, Efficient in small networks	Efficient:复杂度 NlogN.

Costs is nor realistic: calculated by number of hops (edges). Doesn't take consider the brandwidth. **Inefficient in large network:** loop may occur before the convergence state.	Multiple same-cost paths allowed. (only 1 in RIP)
Message complexity: exchange between neighbours only	**Message complexity:** with n nodes, E links, O(nE) messages sent.
Speed of convergence: Varies. May be routing loops – might count to infinity.	**Speed of convergence:** May have ossilations. O(n^2) algorithm requires O(nE) messages.
Robutness: node can advertise incorrect **path cost** (一次可以错整个 path) each node's table used by others, so error propagates through network.	**Robutness:** node can advertise incorrect **link cost (一次只错一个 edge)** each node computes only its own table.
simple but not scalable.	**Complex but scalable.**

Socket

定义

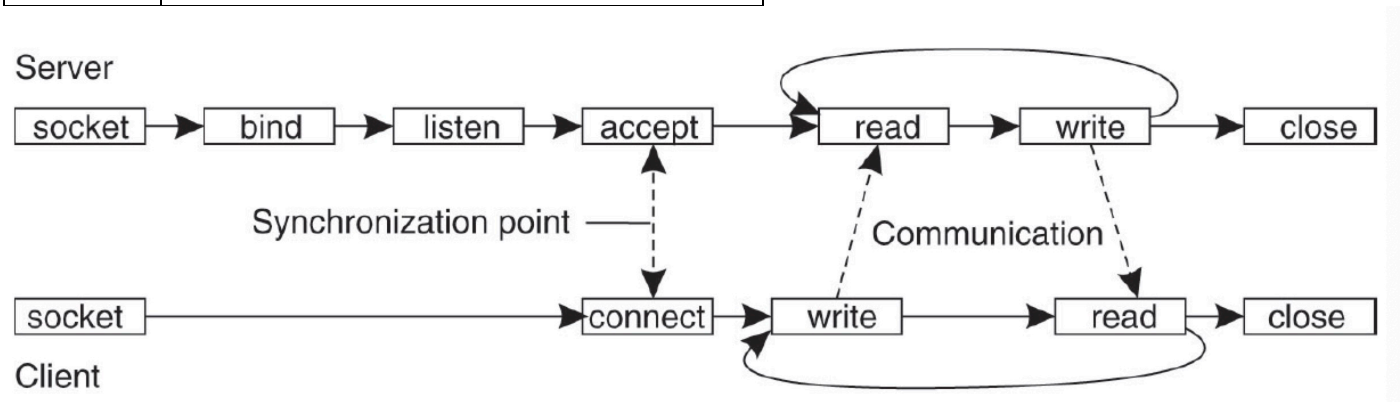
什么是 socket: communication endpoint where application can write data via network and read incoming data.

- 每一对 process 之间有一对 socket 通话. 也就是 one socket, one process.
- Socket 用 IP address 和 port number 定义

Socket 用 client-server model 形式

- Server **listens to i**ncoming client requests on a **port**. Once a request is received, the server **accepts** that connection from the client socket to complete the connection.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection



1. Server execute socket, bind, listen and blocks on accept.
2. **Bind** attach the new socket **to a local address and a port**.
3. Client connects to a specific address, **blocks until response**.
4. Once the connection is **accepted by the server**, the system creates a new socket with same property as the original one. Client and server can communicate with **send and receive.**

Port

- Port number <1024 is for specific service protocols
- Client initiating connection is assigned a free port > 1024.

Java socket

Datagram Socket	**UDP.** Lightweight rather than reliability.
Multicast Socket	Subclass of datagram, sends to multiple reciepients.
Socket	**TCP,** Connection oriented , reliable.

Message-Passing Interface (MPI)

- **Hardware and platform independent**
- Direct use of underlying **network (no multiple layers)**
- Assume **no failures**. Communication involves a group of processors.

Message-Oriented Persistent Communication (MOM)

Feature of message oriented persistent communication:

- Do not require sender/receiver to be active during communication
- Offering intermediate storage capacity for messages.
- Typically target long (> 1minute) message transfers.传个电影啥的

Principle:

- Each application maintains its local queue, sends messages to other application queues.
- Unaware of the reading of its messages. (recipient can be down at sending time)

做的事情也很典型:

1. 如果 sender 上线, 那就 sender 传东西到中间
2. 如果 receriver 上线, 那就从中间收东西
3. 如果 sender receiver 都上线, 那就 1+2
4. 如果都不上线: 东西放中间, 没有传输.

Interface

Primitive	Meaning
Put	Append a message to specified queue
Get	Block until queue is non-empty, remove first message
Poll	Non-blocking variant of get.
Notify	Install a handler to be called when a message is put into the queue.

Streaming oriented communication

边下边播

- In SOC, the stream should not be interrupted; messages should kept being received at regular time intervals.
- Destination can start reading before the entire information has been transmitted
- Each pieve of data should be read in order at fixed rate.

Streaming 里需要注意的点

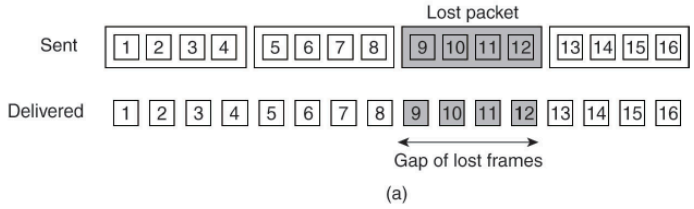
- Bit rate
- Maximum delay: time takes to set up a session.
- Maximum end to end delay: how long takes to transmit databetween sender and recipient
- Maximum delay variance (jitter)
- Maximum round-trip delay: sender-receiver-sender trip.

问题: IP(Internet protocol) could drop packets.IP only does Best effort but not garantee

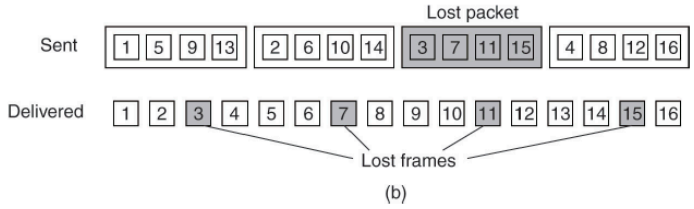
已知每个 packet 路上花的时间不同, 到达 receiver 的时间就有快有慢, 所以如何保证以恒定速率到达 receiver?

- 用 buffer. 但这也不是万全之策因为很难定 buffer 的窗口时间

如果一定要丢包了怎么办: 作弊, 发的时候把顺序打乱成四个不连续的 然后到 receiver 的时候再组合.
如果把一整个连续丢的 N 个包拆散混在别的完整包序列中间, 让你无法发现丢包了



(a)



(b)

DASH: Dynamic, Adaptive, Streaming over HTTP

Server: divides video file into multiple chunks. Each chunk stored encoded at different rates. Manifest file: provide URLs for different chunks.

Client: periodically measures server-2-client bandwidth. Consulting manifest, request one chunk at a time. Client determines 1. When to request chunk, 2. What encoding rate, 3. Where to request chunk.

How to stream simultaneously to thousands of users?

- Store/serve multiple copies of video at multiple geographically distributed sites.

Week 4 Communication

Message Loss

May impact the computation of a distributed system

Why message loss?

- Networks are unreliable
- Messages can be lost

Eg. Server receives too many requests simultaneously. / router drops the message

Coordinated attack problem: two **remote entities** can only communicate via **messages** who can be dropped before reaching destination; They want to do something **simultaneously.**

Fact: there is no protocol to ensure they can win.

TCP/IP

Connection oriented:

Connection: an end to end agreement to perform reliable data transmission

Before any data is transferred the sending and receiving process must cooperate in **establishing bidirectional communication.**

A TCP sending process divides the stream into **a sequence of data segments** and transmits them as **IP packets**.

A **sequence number** is attached to each segment as the first byte with a checksum attached. If receiver does not match checksum, it is dropped.

Packets might take different route to get to destination. So the receiver awaits all sequence number to come and reorder them before delivery at its higher layers.

Sequence numbers:

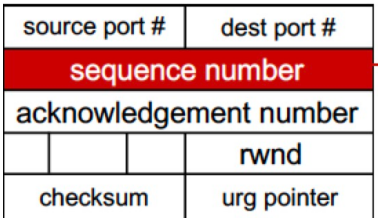
- byte stream "number" of first byte in segement's data

Acknowledgements:

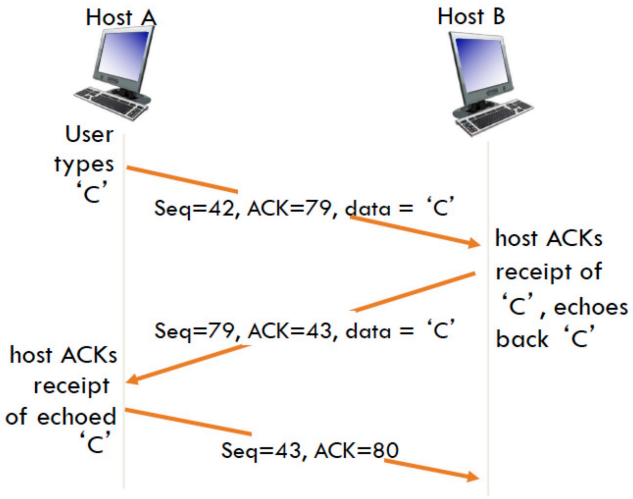
- Seq # of next byte expected from other side

Receiver handles out of order segments depends on the implementor. TCP spec doesn't say.

Packet segment 的组成:



每一次, 获得者发送的新 Seq 是收到的 Ack, 发送的新 Ack 是收到的 Seq 加上 data 的大小



1. A: Seq=42, ACK=79.
2. B: 先交换两数, Seq=79, Ack=42. Data 大小是一个 byte, Ack 加上 1. 发送 Seq=79, Ack=43
3. A: 先交换两数, Seq=43, Ack=79. Data 大小是一个 byte, Ack 加上 1. 发送 Seq=43, Ack=80.

TCP Header 种类:
URG, ACK, PSH, SYN, FIN(结束链接)

什么是 seg number: Byte stream number of first data byte in segment

Sender Events
- Data received from APP.
- Create segment with seq #, the byte-stream number of first data byte in segment.

With each segement sent, a timer is set. If an ack is not received before the timer expires, the segment is resent.
Why lost? It could be either the segment sent is dropped or the ACK segment is dropped.

TCP timeout value should be longer than RTT (rount trip time)
- What if too short? Unnecessary retransmissions
- What if too long? Slow reaction to segment loss
- **What do we use? EstimateRTT + 4*DevRTT (Safety Margin)**

How to find RTT? 难点:需要动态调整, 不能太高也不能太低

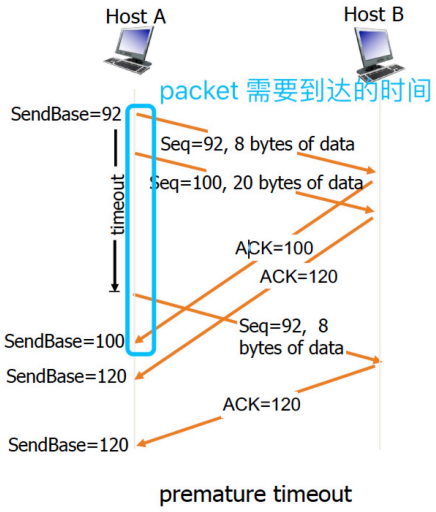
- Find EstimateRTT

Sample RTT is the **average** time from segement transmission until ACK receipt, ignoring retransmissions.
Since **SampleRTT varies** a lot, we want a very **SMOOTH estimated RTT**
$EstimatedRTT = (1 - alpha) * EstimatedRTT + alhpa * SampleRTT$
$alhpa = 0.125$ typically.
Influence of past sample decreases **exponentially fast**!

- Find DevRTT

$DevRTT = (1 - beta) * DevRTT + beta * |SampleRTT - EstimatedRTT|$
$beta = 0.25$ typically.

Premature timeout
原因: Timeout 设置时间太短, 或者本次 RTT 时间太久了



Fast retransmit

用 timeout 之外的 resend 模式

- Since sender often sends many segments back to back, if segment is lost, there will likely be many duplicate ACKs. (correctly sent ack should accumulate)
- TCP fast retransmit restricts that to: if sender received 3 ACKs for the same data, stop waiting (instead of waiting for timeout to finish) and resend unacked segment with smallest seq number.

CWND, RWND, MSS

1. CWND: Congestion Window 路上能发送多少
2. RWND: Receiver window. 收取者能一次获得多少
3. MSS: Maximum Segment Size

The **amount of data that can be transmitted through a TCP connection** is dependent on the **congestion window**, which is maintained by the source. The **receiver window** is maintained by the **destination.**

What is TCP flow control

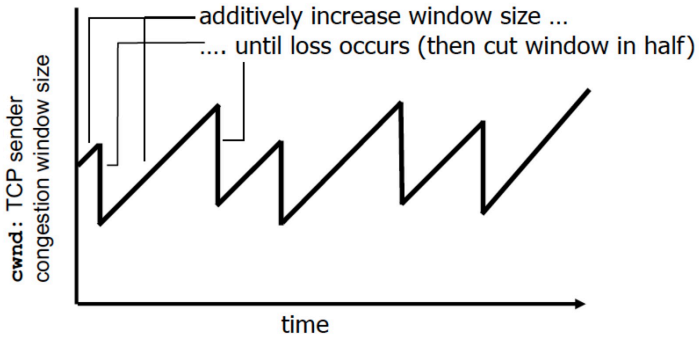
- Receiver controls the sender so sender won't overflow revceiver's buffer by transmitting too much/ too fast.
- Receiver-side buffering: receiver includes RWND value(indicates free buffer space) in TCP header. Sender then limits the amount of unacked data to receiver's RWND value.[类似是 receiver 告诉 sender 有多少空间]

Congestion control (关于 CWND)

- What is congestion: too many resources sending too much data for network to handle.
- 两类问题: Lost packets (buffer overflow at router 缓存爆了) /long delays (queuing in router buffers 在缓存区等待太久)

方法: Sender 逐渐越发越多直到 loss 发生(试探上限), 然后把发送速度减半.

- Additive increase: increase CWND by 1 MSS every RTT until detect loss.
- Multiple decrease: cut CWND in half after loss.



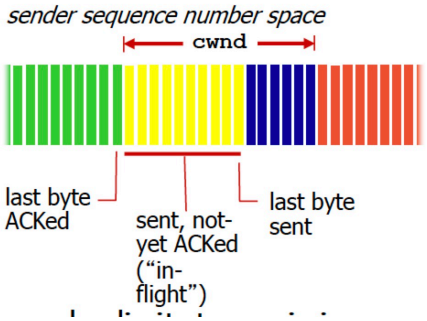
CWND 是一个动态的, 关于时间的 function of perceived network congestion, 单位是 Byte

TCP sending rate:

$$Rate \sim \frac{CWND}{RTT}\ bytes/sec$$

Sender limits transmission:

$$LastByteSend - LastByteAcked \leq CWND\ (Byte)$$



黄色不能蔓延到红色区域去, 一定要有一些蓝色.

TCP slow start

刚刚建立传输的是时候 CWND = 1 MSS, 每次 RTT 之后翻倍 ,直到出现 loss.
Initial rate is slow but ramps up exponentially fast.
第一次出现 loss 之后, cut by half, 然后每次就用之前说的 Additive increase (linearly each time increase by 1 MSS) & Multiple decrease 方法
复习一下: 如何发现 loss? 3 Duplicate Ack (TCP fast transmission)或者 1 次 timeout

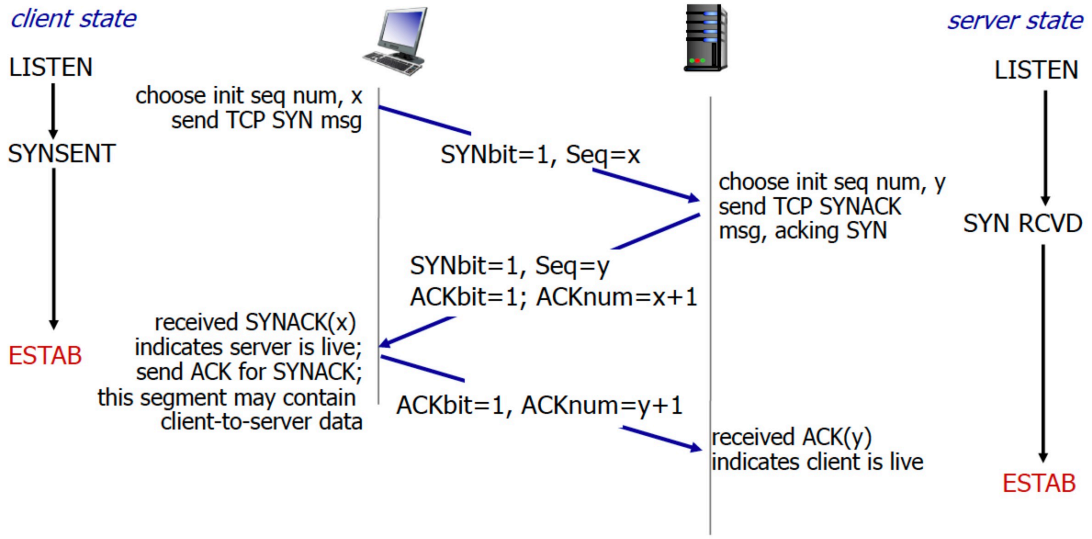
而在另外有一些协议里(TCP Tahoe), 每一次出现 loss 之后, CWND 都是从 1MSS 开始, 先用 exponential 直到 1/2 最大允许的 CWND, 然后转为 linear.

TCP connection management

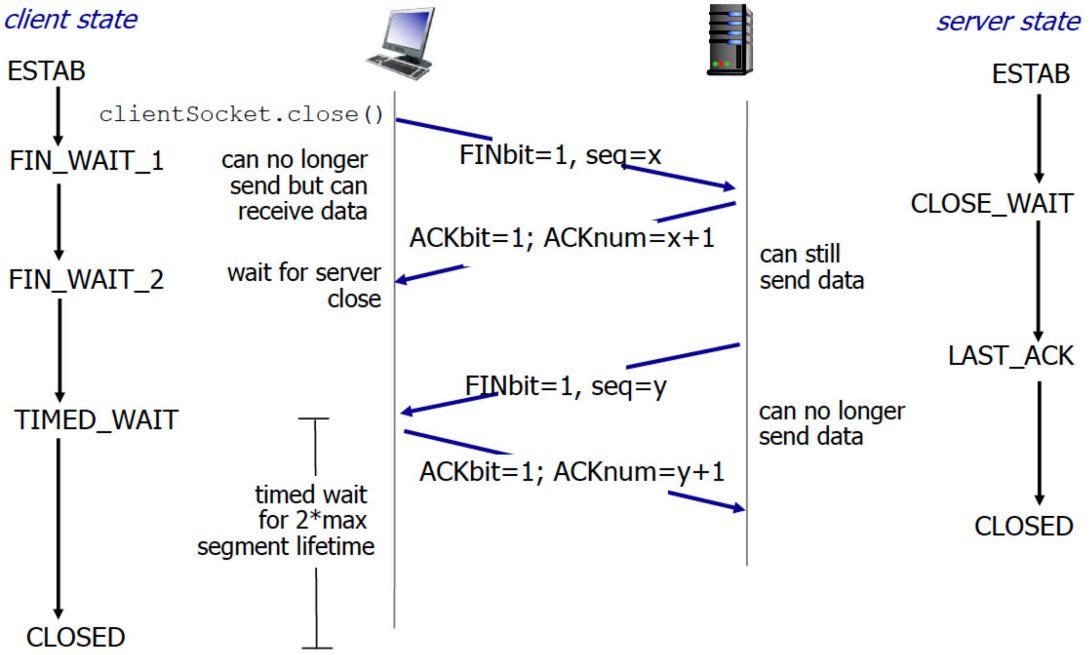
Establish Connection

为什么需要握手? 需要了解一些公共信息, 比如

- 彼此是否想要传输
- MSS 多大
- RWND 是多少(因为是 receiver dependent, 注意不是 CWND)



Closing Connection



HTTP: Hypertext transfer protocol

HTTP is an **application layer protocol,** uses **client/server model**

- Client: browsers requests web objects
- Server: web server sends objects in response to requests

HTTP is stateless.
Server maintains no information bout past client requests.

HTTP 和 TCP 关系: HTTP 在 TCP 之上,使用 TCP.
Client 从 port 80 开始 TCP connection; server accepts TCP from client. Then HTTP messages exchanged between client and this server.

HTTPS: HTTP over TLS(Transport layer security, eg. Over SSL)

(from symentic)
SSL stands for Secure Sockets Layer. it's the standard technology for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems, preventing criminals from reading and

modifying any information transferred, including potential personal details. It does this by making sure that any data transferred between users and sites, or between two systems remain impossible to read. It **uses encryption algorithms** to **scramble data in transit**, preventing hackers from reading it as it is sent over the connection. This information could be anything sensitive or personal which can include credit card numbers and other financial information, names and addresses.

TLS (Transport Layer Security) is just an **updated, more secure, version of SSL**. HTTPS (Hyper Text Transfer Protocol Secure) appears in the URL when a website is secured by an SSL/TLS certificate. The details of the certificate, including the issuing authority and the corporate name of the website owner, can be viewed by clicking on the lock symbol on the browse

RPC: Remote Procedure Call

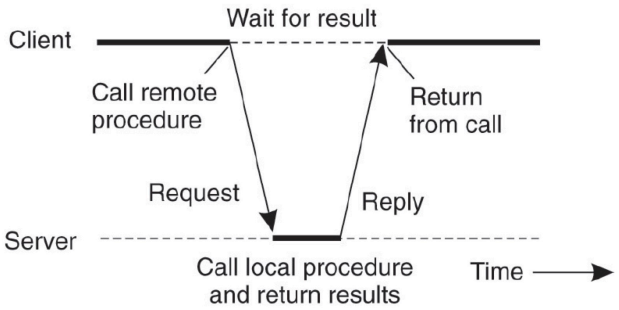
RPC and RMI are more transparent for the client

It allows programs to call procedures located on other computers. No message passing visible to the programmer.

Conventional Procedure call: stack pointer opens up new stackspace from main process.

Will modifications within the called procedure affect the value outside the procedure scope??

- Call-by-value: NO
- Call-by-reference: YES (pointer)



(这张图是 synchronised RPC)

Two sides: client stub and Server stubs

RPC Steps:

1. Client Procedure calls Client Stub (CP -> CS)
2. Client Stub builds message and call local OS (CS -> COS)
3. Client OS sends message to Server OS (COS -> SOS)
4. Remote OS gives messages to Server Stub (ROS -> SS)
5. Server Stub unpacks param and calls Server. (SS -> Server)
6. Server does work and return to Stub (Server -> SS)
7. Server stub pack it and calls Server OS (SS->SOS)
8. Remote OS send to COS (ROS -> COS)
9. Client OS give message to Client Stub (COS- > CS)
10. Client Stub unpacks and return to client (CS -> CP)

过程就是 CP-> CS -> COS -> SOS -> SS -> Server 得到结果之后原路返回.

什么是 stub: A stub in distributed computing is **a piece of code that converts parameters passed between client and serve**r during a RPC

Stub 的职责是什么: parameter marshalling. Packing params into a message

1. Client Stub **marshals** parameter and identifier of procedure operation to call. 比如一个 function call 是 ADD 1 2, 那两个 parameter 是 1 和 2, operation 是 ADD
2. Server Stub **unmarshals** message and calls Server Procedure.
3. 然后发送回去的时候, SS 的职责是 marshal 1+2 的结果, CS 的职责是 unmarshal 这个信息.

为什么我们发送的时候必须同时发送 type 的信息

- Different architectures may have different data representations
- big endian (SPARC) 大端, 低地址存高位/little endian difference. (x86) 小端, 低地址存低位

同样, 如果 procedure 是 pass by reference 的, client stub 需要 **marshall pointer and value it points to**.
Server Stub unmarshall the message, and **determin the pointer** (may be different) and call server procedure. If the server modified the value located at this pointer, then **modifications are visible from the stub**. Once the result message is sent back to the client stub, its copied to the client.

Client and server are implemented by **IDL: interface definition languages**

异步 RPC (Asynchronous RPC)

Client 在发送异步 RPC 之后继续做自己的事情, 等到结束进程之后用 future 来 interrupt client 的下一步. 可以达成 0 等待时间.

RPC 用在哪里: Middleware for Unix/Windows.
如何管理 client 和 server 呢? 除了 client 和 server 之外, 安排一个 directory server 作为中间人
How to locate a server?

- Client 在 DS 这里查找提供某个服务的 server; Server 在 DS 这里注册, 然后 Client 就知道 server 的存在.

How to locate a process?

- Server machine 管理这件事. Server machine 不仅仅有 server, 还内置了 DCE daemon maintain server/port table.

RMI

RMI: Remote method invocation, Java feature similar to RPC
RMI allows a thead to invoke a method on a remote object (an object on a different JVM)
RMI 和 RPC 区别:

1. **RPC is procedural; RMI is object absed**. (methods on remote objects)
2. RPC parameters are **data structures**; RMI parameters can be **primitive data types** & objects.

RMI 的步骤也和 RPC 比较像(有一些不同); 3 个专有名词

- Stub: **client-side** marshalling and unmarshallin proxy for remote object. **专指 client** 了
- Skeleton: server-side marshalling and unmarshalling proxy for the message. **专指 Server**
- Parcel: the information, data. marshalled parameters.

1. Stub creates parcel -> server.
2. Skeleton unmarshals param and computes
3. Skeleton marshals the return value and create returning parcel.
4. Stub unmarshals the info and pass back to client thread.

Multicast

使用的例子一般是 Delay Tolerant Services. (允许延迟的那种

这个词广义包含了 5 个类似术语:
Broadcast: 1 to ALL communication
Unicast: 1 to 1..
Anycast: 1 to random 1
Geocast: 1 to geographical neighbours
Multicast: 1 to many
Question to solve: how to set up all connection paths and how to maintain them??

Application level multicasting

- Organizing nodes into overlay network that **does not include routers.** (Hence routing might not be optimal)
- Overley network takes form of Tree Network and Mesh network.

Terms to measure the multicast structure

- Link Stress: How often a packet corers same link
- Relative Delay Penalt: ratio in delay between two nodes in the overlay and the delay those two nodes experiences in the underlying network
- Tree Cost: aggregated link costs.

Gossip Based Data Dissemination ***O(LogN) to all Nodes.***

特点: Efficient and scalable, but do not provide service guarantees

这种模式有三类 node

1. Suspectible – node not seen the data (没看到过
2. Infected – node holds data and willing to spread to others
3. Removed – updated node not welling to spread its data (曾经是 Suspectible, 不再参与

Anti-entropy spreading 算法:

1. P 随机 pick Q.
2. P only push updates to Q; only pulls new updates from Q. P & Q sends updates to each other.
3. PUSH: only when the other side is Suspectible (只向没看到过的发信息
4. PULL: only when the other side is Infected (有 data 的才能发 update
5. Round: time for every node exchange information at least once.
   缺点是会有很多重复的推送

Rumor spreading 算法:

1. P 随机 pick Q.
2. P 向 Q push, 如果 Q 已经知道这条信息内容, P 降低 spreading 的意愿 by 1/k

这个算法不能保证所有人都能 update 到, 应该 Anti-entropy spreading 和 rumor spreading 结合使用: network topology aware dissemination (directional gossiping)

directional gossiping 算法

1. 尽快向重要的 node 推送. Disseminate to important (high centrality) nodes as soon as possible to increase the delivery to as much as possible.
2. 如何评判重要: 3 个维度, Degree Centrality, Betweeness Centrality, Closeness Centrality.

Week 5 Naming

4 个 topic
什么是 name 以及为什么需要; Flat Naming; Structured Naming; Attribute-based Naming.

Name

就是用来找 resource 的. 就像 dict 需要 key 一样. 要知道某样东西在哪里, 需要地址; 要分享一个服务给别人, 也需要地址.
URL, email, 都是 name 的例子.
识别, 分享, 确认位置.

Name: used to **denote an entity** of a distributed system, usage includes:

- identify a resource out of many. (**URL**: **Uniform Resource Locator)**
- share a resource with other processors (printer)
- locate entites independent of their current location(email)

Address 一个个体可以由多个地址

name of an access point of an entity. One entity can have multiple address. Address can easily be changed. **Inflexible and not human-friendly.**

Identifier 一个个体只能有一个 ID

Uniquely identify an entity. Address!=(cannot be used as) Identifier. Doesn't need to be human friendly but can simply be a sequency of numbers. (Remote object references,

Human-friendly names

- Human-friendly names represents an entity in some form represents a meaning for human. 对人有意义的, 人会使用的就是 human friendly 的
- Eg. Domain name usyd.edu.au
- 用 character string 表示
- Resolve: Name is ***Resolved*** when it's translated into data about the resource to evoke action.
- Binding: 使 name 和 object 的一部分产生联系

Close relationship between **name resolution and message routing**.

Flat/Unstructured Naming

- **Not human-readable.** Doesn't contain any **info to locate the access point** of the object.

Phycical address (MAC address)

- **48 bits** of unique number, set up by manufacturer.
- One single machine may have multiple interfaces.
- Used in **Data Link Layer** to locate the receiver

Locating

How does LAN (Local Area Network) find MAC address?

- **Broadcast.** But only the entity with the correct access point can reply with the address.
- **ARP (Address Resolution Protocol)** returns the physical address given an IP in a network.

Howto resolve a MAC address on a LAN?

- Broadcast (1 to all): sending messages to all. 但是 broadcastingcost 太高, 不是大型网络里的好选择.
- Multicasting(1 to many, not all): sending to a specific group only. (所以有一组一组的 multicast group, 在哪一组的信息也必须包含在 IP address 里. Message to this address delivered to all group members

之前的 assumption 都是机器老老实实的待在一个地方. 如果东西会动怎么办? (Eg. Mobile)

- **Forwarding pointers:**

    - when entity **moves** from A to B, leaves behind in A **a reference to its new location** at B.
    - A client look up current address by following the chain of this pointer.路过留个 pointer,让别人跟着.
    - 显然不太可取, 你坐高铁玩手机怎么办? Long chains (expensive) for a highly mobile entity; all intermediate loations will have to maintain the chain of pointers needed. (复习一下: RPC 就是一个 forwarding pointer 的例子)
- Home-based approach:
- 别人访问初始地点, 而初始地点保留 mobile entity 的当前位置
- 问题也很明显, 如果离太远了:Increased communication delay; 而且容易是 home base 出问题:Home location must exist/contactable all time.

为什么需要 name server:
之前的例子里 broadcasting 给那么多人大家基本都是 ignore 的. 浪费资源.

Name Server:
- make servs responsible of maintaining the name-address mapping.
- Name resolution: clients contact such server to get the information.

Flat/Unstructured Naming – DHT (Distributed Hash Table)

What is hashing: **A deterministic function** that **gives same input** outputs the **same digest (output).** The digest is **typically smaller** than original message.
- One way function: can perform F(x)->Y but cannot give x from Y.
- Weak collision resistant: different input should rarely produce same digest.

例子: MD5

那如何 distributed hash table 呢!
Goal: provide a distributed lookup service returning the host of a resource.
Keys <-> resource | Names <-> objects
Locating key <-> Resolving a name

Distributed lookup:
- Iterative: node requested to look up a key **returns the address of the next node found.**
- Recursive: node requested to look up a key **forward the request**

DHT: Chord (Complexity of lookup: O(LogN)

A resource with key k is stored in the node with ID k. If there is no node that ID = k, the resource is stored in the first available higher ID node.

Node 的 Key 顺时针增大.

When Node N **joins network:**
- Node p calculate its harsh
- Contact arbiatry node, traverse until SUCC(p+1) and insert itself next to SUCC(p+1)

When Node N leaves the network:
- Inform its predecessor and successor, transfer its data items to its successor.

Update finger table:
- Each node regularly contacts **succ(P+1)** and requests to return **pred(succ(P+1)) 往后走一步往前走一步应该是我自己.**
- If p== pred(succ(P+1)), its OK.

How to insert data into the chord?
- Hash data to get key K.
- Routing to find node that also has key K. (or first higher than K) store the data.

How to search data:

- Same way, hash, find key, search using finger table.

DHT: Content Addressable Network (方形那个) (Complexity of lookup: O(d* n^1/d), 二维里就是 O(sqrt(N))

举个二维坐标的例子, 每个 node 和 file 都有 x y 两个坐标. Node 会管理它方形坐标范围内的 file.(实际上不止 2 维坐标)

- How to join: each node chose a random coordinate, lookups the node responsible for this point, and becomes responsible of half of its zone.
- How to leave: A node takes care of being responsible of the zone of leaving node.
- How to route: through abutting zones. Each node maintains the addresses of its neighbours: topmost zones are considered abutting bottommost zones. To route towards the destination, each node chooses **its closest neighbour** to the destination in terms of **Euclidean distances. N 维空间里 bfs 真的牛逼**
- **一个 node 需要 maintain 几个维度的邻居信息? 2 维里 4 个(上下左右), D 维里 D 个.**

DHT 的好处

- Fault Tolerance: fully distributed, no central point of control.
- Very small changes needed to adapt to change.
- Scalablility to large number of participants (CAN 是 O(d* n^1/d), Chord 是 Log(N))
- Routes are short compare to N. (if $N = 2^m$, route length can be $O(M)$)
- Do not consider network-proximity of nodes.

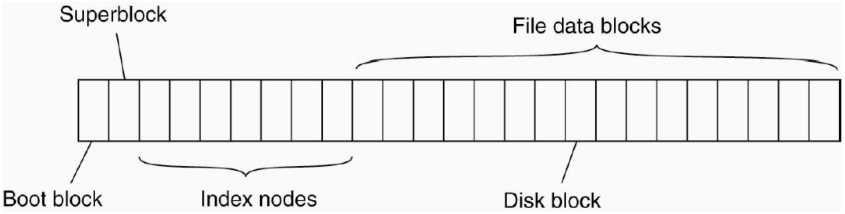
Structured Naming

有层级关系的 naming system

Labelled, directed graph with two types of nodes:
Leaf Node: no outgoing edges
Directory Node: a number of outgoing edges.

很多时候就是个 tree, 比如 Unix file system. Root 在"/".

Unix File System



Boot block: System boot time 加载;
Superblock: info of file system, 比如 size, which blocks are still unallocted, unused inodes.
Index nodes (Inodes): indes starting at 0 (root) and contain info about where to find file data.
Each directory is represented as a file.

Linking and Mounting (安装

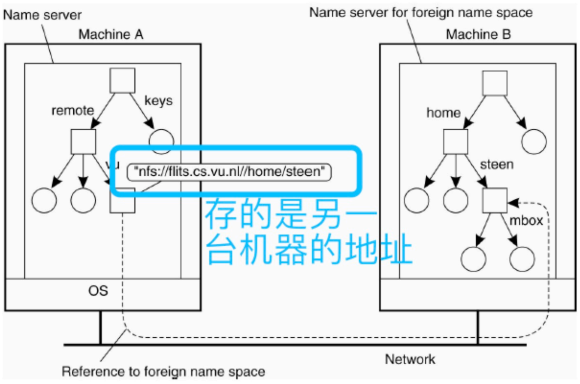
Alias

- Hard links: **multiple absolute paths** refer to the **same node** in a naming graph.
- Symbolic links: **a leaf node stores a path name**; resolving that name leads to another path name etc.

Mounted file system: It lets **a directory node** store **the identifier of a directory node from a different namespace**.

- **Mount point:** directory node storing node identifier. 自己的
- **Mounting point:** directory node in foreign namespace.外来的

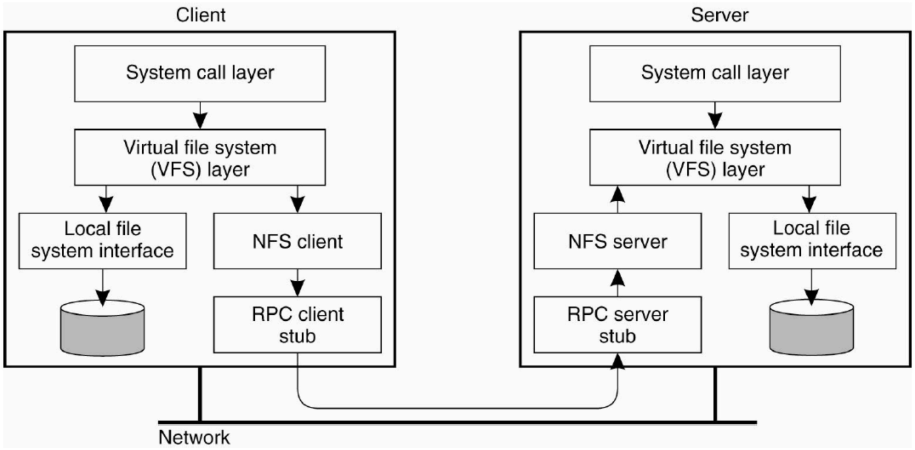
安装需要的条件: Access protocol name (NFS, 用来访问除了 local 之外); Server Name; Mounting point name.



NFS (Network File System)

特点: Stateless (up to V3) ; modifications are very slow, commited to servers's disk before return to client.

- 3 layer:
  - Unix File System Interface: open, read, write, close calls + file descriptors
  - VFS layer: distinguishes local from remote files.
  - NFS service layer: bottom layer. Implements NFS protocol
- 可以用它做的事: read/search, manipulate directories, accessing file attributes/reading & writing.



VFS 层给两边呈现大家都在 local 保存着的感觉. 实际上分布有 local 和 NFS

Caching

Why we need caching? Access remote entity takes time (go through network, disk..)
如果近期访问过 remote 的 entity, 会把 remote 放到 local 的 cache 里去. 短期内再次访问就会直接使用 cache 里的内容
Policy: 当 cache 满了的时候去掉哪些部分. 最经典就是 LRU 了, (least recently used block is overridden by the new access entry)
Name translations: mapping from paths -> inodes. Only when first time accessed the directory is parsed. Later on its map to a given inode.
Disk blocks: mapping from block address -> disk content. If two processes on the same machine access same disk block, the second one finds it in the memory.

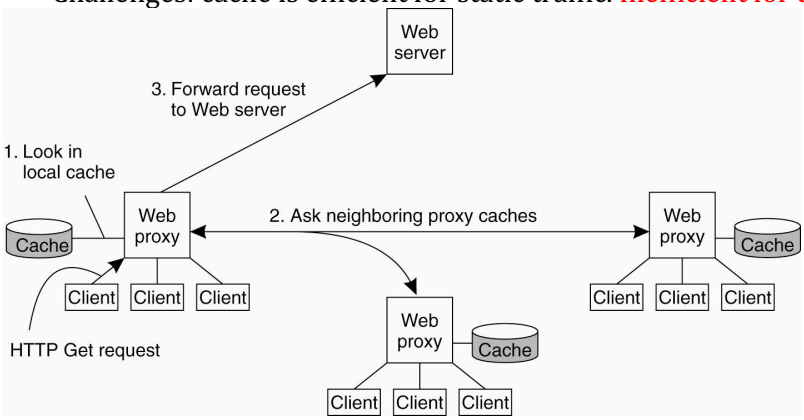
两种 file system caching

1. NFSv3: weak consistency
2. Andrew File System: stronger consistency

Web caching

- WWW: originally a graphical front end to RPC protocol; no cachineg; providing daily accuracy

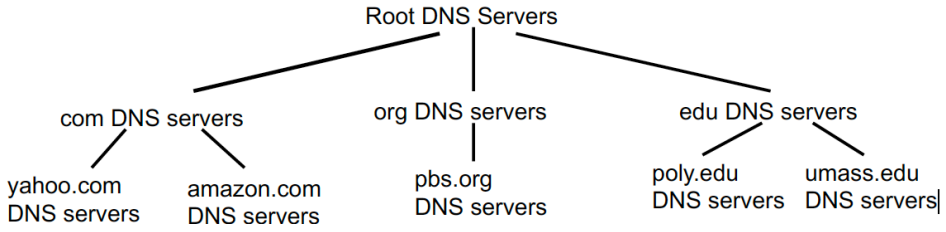
- Browser does **client side cache**: caches webpage for faster reaccess. Can generally be flushed or disabled in the browser settings. Update might not be apparent instantaneously. Time to live (TTL) is given by HTTP “expires” from server.
- Proxy provides **network cache:** decrease the load on server; hierarchical cache.
- Challenges: cache is efficient for static traffic. Inefficient for dynamic traffic (multimedia files)



本地没有看邻居 邻居也没有最后再看 remote web server.

DNS

URL: Uniform Resource Locator, uniquely locate and access a resource.
一般有两个部分, host name + path name. http://sydney.edu.abcedfg/somepic.jpg
URI: Uniform Resource Identifier, uniquely identify a resource
Coherent: 这个 resource 可以被许多 software 发现
Uniformly: 新的 resource 也可以以此类形式加入

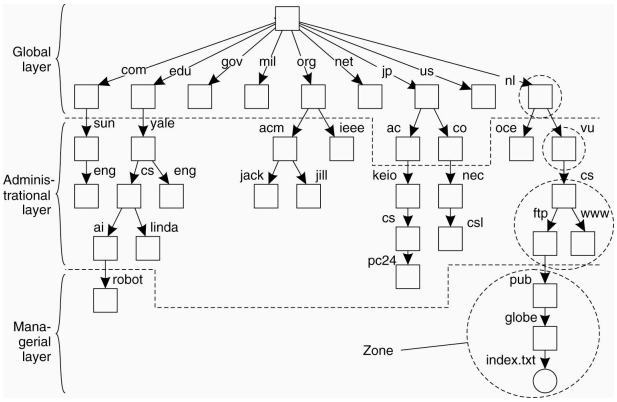


为什么需要 DNS 系统呢?

1. Load distribution. 不能让所有的 request 都找同一个区域的 server. Replicated Web servers: many IP addresses corresponds to one name.
2. Hostname to IP address translation. 人不可能记得 IP address.
3. Host aliasing.

Domain: a subtree. 比如.com 结尾的就是一个大 substree.
Domain name : a path to its root node

最上层的上面: root DNS server
最上层: Top level domain servers (TLD
中层: **authoritative DNS servers & Local DNS servers**
下层: files



Root name servers

13 root name servers world wide. Contacts authoritative name server if name mapping not known; gets mapping; **Returns mapping to local name server**. Each server **replicated many times**. (fault tolerance)

RR format

RR stands for resource records

RR format: (name, value, type, ttl)

Name 和 value 的关系肯定要考!!

If type==**A:** name is host name; value is IP address. 最标准的

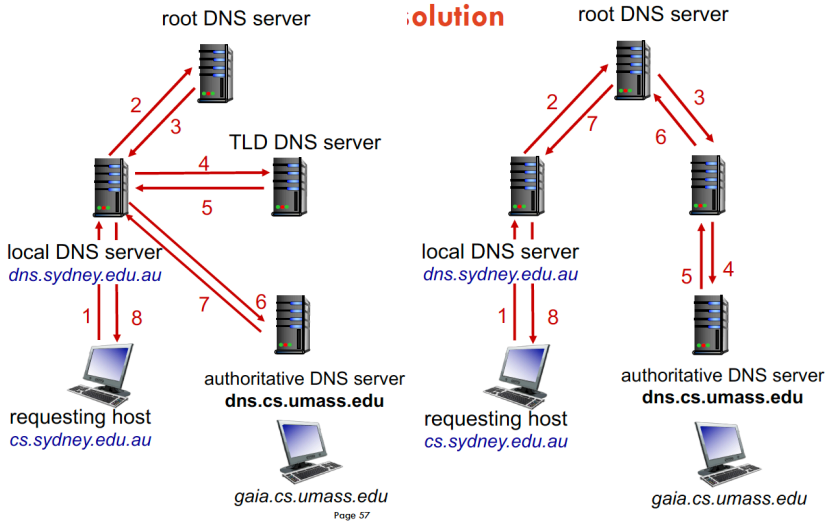
If type==**CNAME**: **name is alias name**, not real name. **value** is canonical(real) name 比如 www.ibm.com 其实是 servereast.backup2.ibm.com 存的是 Node-symbolic link with primary name of the represented node.

Type==NS: {name: domain, value: host name of Authoritative name server}

Type==MX: {name: name, value: mailserver's name}

同样的有两种 resolution 方法: iterative/recursive

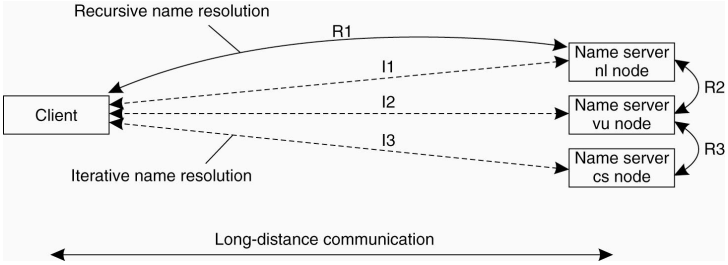
Iterative: 不方便 cache



左边 iterative, 右边 recursive (类似 DFS)

Recursive: 可以更好地利用 cache

从距离上来看也是 recursive 更好一些 更快一些



在 DNS 里是如何利用 cache 的呢?

一个 name server 之前 resolve 过一个 mapping 之后, 它会 cache 这个 mapping.

Time To Live (TTL)生存时间，就是一条域名解析记录在 DNS 服务器中的存留时间。当各地的 DNS 服务器接受到解析请求时，就会向域名指定的 NS 服务器(权威域名服务器）发出解析请求从而获得解析记录；在获得这个记录之后，记录会在 DNS 服务器(各地的缓存服务器，也叫递归域名服务器）中保存一段时间，这段时间内如果再接到这个域名的解析请求，DNS 服务器将不再向 NS 服务器发出请求，而是直接返回刚才获得的记录；而这个记录在 DNS 服务器上保留的时间，就是 TTL 值。

如何注册一个新的域名

需要在 DNS registar 注册, 在 Top Level Domain 这里插入两条记录

1. Type= A: 注册真名和 IP 的关系 RR = (dns1.networkutopia.com, 212.212.212.1, A)
2. Type=NS: 注册 domain 和真名的关系: RR = (networkutopia.com, dns1.networkutopia.com ,NS)

总结

Naming resolution 域名解析

为什么需要 Naming Hierarchy(树结构):

- Helps distribute name resolution. Allows DNS to scale very large

Naming helps **masking lookups**: allows NFS to represent local & remote directories identically. 装作云端资源在 local 这里

为什么需要 cache:

- Caching helps lowering communication: static information can be cached for efficient retrival.

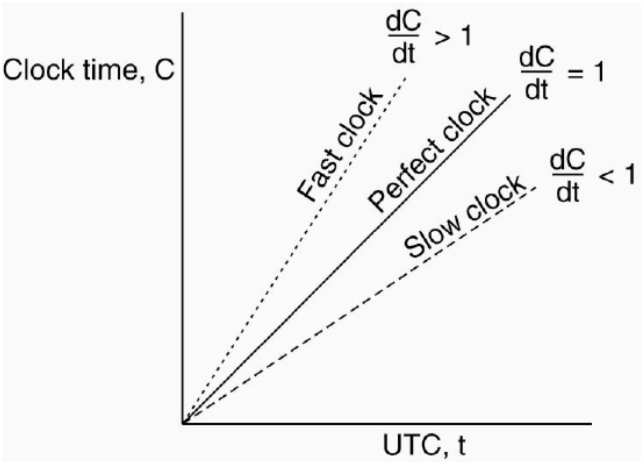
分布式域名解析的好处 (fully distributed naming solution): tolerate many failures: maintains only small amount of info per node and leveraging this information to route quickly.

Week 6 Synchronization

Physical Time

Time can be measured astronomically by calculating the difference in earth's transit angle with the sun.
Solar day: interval between two events of sun's reaching highest apparent point in the sky = 24 hours. Bases for GMT standard.
Mean solar second: average seconds length over # days
Atomic clock: 1 second = cesium 133 atom to take 9*10^9 transitions. It has a constant length.
Leap second: extra time to count on earth's slowing down. Operating Systems must be aware of this.
UTC: universal coordinated time

The **accurate time is broadcasted** because it's too expensive to have one on every computer. Its broadcasted by GPS on short radio wave, each second.
Computer clock **use quartz crystal.**
Hardware clock: stored in CMOS RAM. Even when CPU is off, the time is still incremented.
Software clock: starts at boot time and synchronize to hardware clock.
Clock skew: multiple CPU gives different time. **Skew = dC/dt -**1

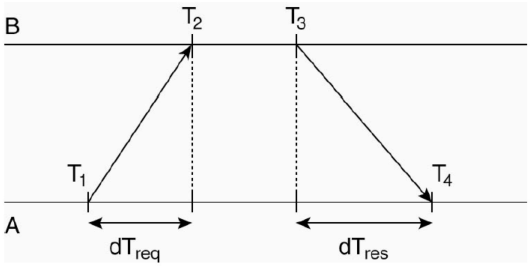


如何协调 physical time

1. *NTP (Network Time Protocol)*

Clients synchronise its time with a time server which have an accurate time.

- *Round-trip time* = $(T_2-T_1)+(T_4-T_3)$
- *Offset* of A relative to B = $T_4 - T_3 - RTT/2$
- Algorithm A can re-adjust its time to T3 + RTT/2
- This results is not accurate but gives an acceptable approximation



The University of Sydney

RTT, 在路上的时间: (T4-T3) + (T2-T1)
A 与 B 之间的时间差 = T4-T3 – RTT/2 = T4-T3 – (T4-T3)/2-(T2-T1)/2 = (T4-T3)/2 -(T2-T1)/2
A readjust to: T3+RTT/2
NTP is set up pairwise between servers
Strata: 根据本身计数器的精确程度划分. Strata 0 是 atomic clock.
如果某个机器和 stratum i 层进行 sync, 那这个机器就算是 stratum i+1

Berkeley Algorithm

适用于没有 strata 0 的情况. 某个 server 负责获取, 计算并返回所有 server 的平均时间. 适用于 relative sync, 绝对值结果与 UTC 可以差得很多很多.

1. Time daemon sends request to all nodes
2. Nodes answer with their local time
3. Deamon calculates average and tells all nodes to adjust their clock.

Pros & Cons of Physical Time

Pros: total order. All events made comparable.
Cons: hard to achieve. Difficult to synchronize. In large scale system, message delay is typically high, heterogenous, unpredictable.

Logical Time

Logical time keeps track of the order of events

- If two process do not interact, it is not necessary for their clocks to synchronise.

"happens before"

- A and b are events from the same process such that a occurs before b, or
- If a is the event of message being sent by one process; b is the event of the same message being received by another process;
- It exists some event c that a-> c and c-> b.

If two events x and y happen in different process that do not exchange messages, then you cannot tell x->y or y->x. x and y are concurrent.

Lamport's Logical Clock

1. Each process maintains a counter
2. Before executing an event (sending a message): a process increments its counter by 1
3. When a process sends a message m to another process, it sends the counter as the time stamp of the message.
4. When receiving the message, process adjust its own local counter to compare of the received timestamp.
5. After receiving and updating, increment its counter by 1.

If Pa -> Pb, then Ca<Cb. 但是 不是说 Ca<Cb 就有 Pa -> Pb, 还需要有 communication.

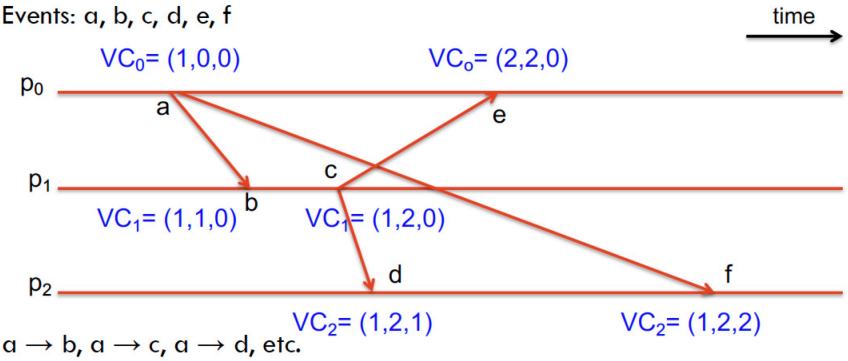
Vector Clock 重要

Vector clock can visualize both "happen before" and "did not happen before" relationship.
One is **lower than other** if each of its coordinates is lower than the corresponding coordinate of the other
VC(a)<VC(b) if for all i (0<=i <n) VC(a)[i] <VC(b)[i]每个维度上都比另一个要小的话 这个在另一个的前面

If VC(a) < VC(b), then a->b ; also if a-> b then VC(a)<VC(b) 这里是个 iff 的关系

在 counter-increment 上 和 lamports 大致一样.
和 lamport 不一样的地方时 vector clock 每一个点都存了它所知道的其他 process 的先后信息

- Processes: p_0, p_1, p_2
- Events: a, b, c, d, e, f



- $a \rightarrow b$, $a \rightarrow c$, $a \rightarrow d$, etc.
- $e \not\rightarrow f$ and $f \not\rightarrow e$

VC(i) < VC(j), if and only if $i \rightarrow j$

Logical time 总结

如果 event A has strictly lower logical clock than another event b, it is that: a & bare concurrent, OR, a happened before b.

Multicast

Ordering of messages:

1. Totally ordered: for all messages m1 and m2 and all processors pi and pj, if **pi delivers m1 before delivers m2**, then **m2 is not delivered at pj before m1** is.
2. Casually ordered: ..if m1 happens before m2, then m2 is not delivered. At pi before m1 is.

Mutual Exclusion

- When will we have mutual exclusion?

Concurrent access to the same resource may corrupt the resource or make it inconsistent.
Problem: **multiple processes in a distributed system** want **exclusive access** to some resource.

两种 algorithm 来解决这个问题

Centralized.

有一个 process 来管理. 比如 process1 和 2 同时需要用一个资源, 先来后到, 管理者记录目前在等待的程序, 然后用完资源的程序向管理者发送 release 信号, 管理者把资源给排队的下一个.
三种 message: request, granted, release
好处: simple, no starvation,
坏处: single point of failure.

Decentralised Algorithm

方法: 假设每个 resource 被复制了 n 次, 每一个复制都有自己的 coordinator
Assumption: 如果一个 coordinator 崩溃了那它会马上 recover 但是会忘记自己已经 grant 的 permission
Problem: coordinator 可能会忘记自己已经给出了资源, 导致重复 grant 资源

Richart and Agrawala algorithm

好处: N-point of failure.
坏处: More message exchange, getting permission from everyone is waste, starvation

Token ring algorithm

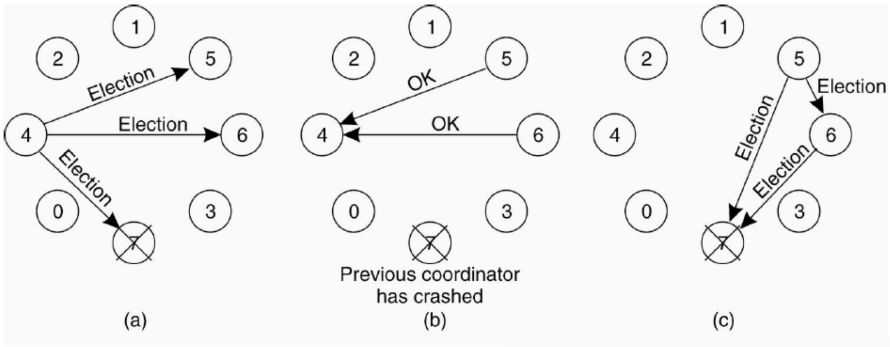
Algorithm	Message per entry/exit	Delay before entry	Problems
Centralised	3	2	Coordinator Crash
Decentralized	3mk, k = 1,2..	2m	Starvation, low efficiency
Distributed	2(N-1)	2(N-1)	Crash of any process
Token ring	1 to infinity	0 to N-1	Lost token, process crash

Leader election

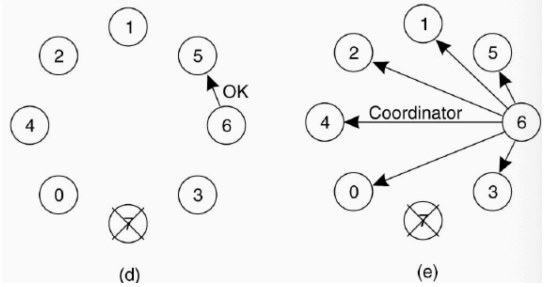
为了解决什么: 有的时候有一个 algorithm 需要一个 process 来当 coordinator, 但是如何选呢?
有的时候 coordinator 选出来就一直是了, 形成 centralized solutions -> single point of failure.
如果一个 distributed system 没有 coordinator, 就不一定比 centralized solution 更 robust.

Bully Algorithm

1. Process p sends ELECTION message to all processes with HIGHER numbers
2. If no one responds, p wins election and becomes coordinator
3. If one of the higher up answers, it takes over P's job.



- (a) Process 4 holds an election
- (b) Processes 5 and 6 respond, telling 4 to stop
- (c) Now 5 and 6 each hold an election
- (d) Process 6 tells 5 to stop
- (e) Process 6 wins and tells everyone



也就是目前活着的最高顺位 process 最后当 leader.

Leader Election

依旧是: 目前活着的最高顺位 process 最后当 leader

1. Process priority is obtained by organizing process into a logical ring. Process with highest priority should be elected as coordinator

任何活着的 process 都可以发 election 信息给它的顺位 successor, 如果顺位的死掉了, 发给下个顺位的. Loop 一圈之后获得所有活着的 process 的 list 记录. 选所有的活着的里面顺位最高的.

Week 7 consistency

Definitions

Why do we need consistency?

- Improve the reliability of a system.
- Scale in numbers and geographical area. 如果一个数据库炸了还有另一个可以用, 哪怕澳洲大陆沉到海里去了还有北美的数据库可以用.

Drawback of consistency:

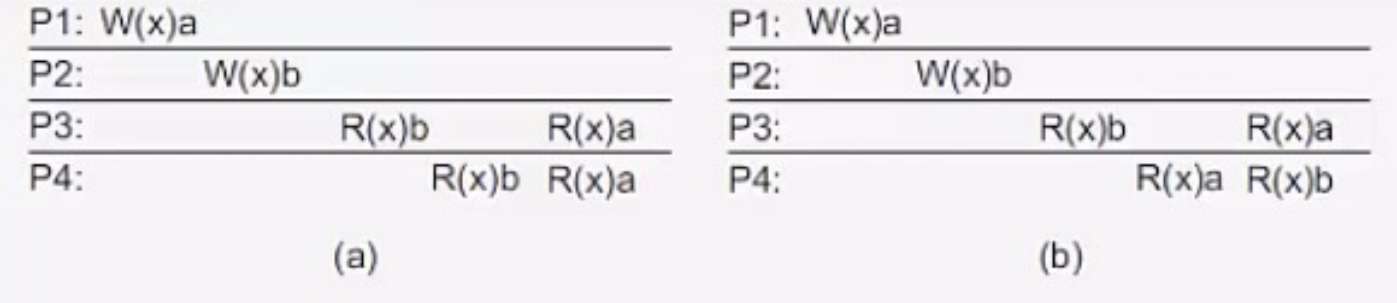
- All replica needs to maintain information so that: the "write" of a user needs to be sync to all replicas, and the "read" should the be up to date from ANY replica.
- 就像某个 thread 的写操作需要保证其对之后的所有 thread 的读操作都可见.
- Difficulty: propergation takes time; not all replica synced at the same time. 这样的话两个 client 可能会访问到不一样的 data.

Concurrent: each of two operations starts before the other ends;
Update: operation modifies the data
Conflict: two operations access the same data and at least one operation is "update".
Data store: a distributed collection of storages, a process has its local storage copy.
Consistency: a **consistency model** is a contract between a process and the data store - if process obey certain rules, the data store promises to work correctly.这句话太虚了, 实际上是: the data store specifies what the results of read and write operations are in the precense of concurrency.

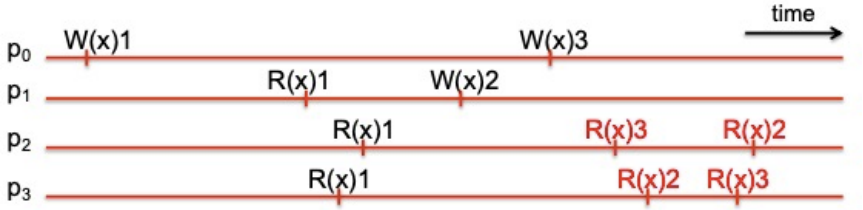
目标: ensure all conflicting operations are done in the same order everywhere.
Conflicting operations: read-write conflict; write-write conflict 同时发生的读写和写操作.

Sequential consistency

任何操作的结果就和单线程发生的一样
The result of any execution is the same as if the operation of all processes were executed in some sequential order, and the operations of each individual process appear in this sequence in the order specified by its program.



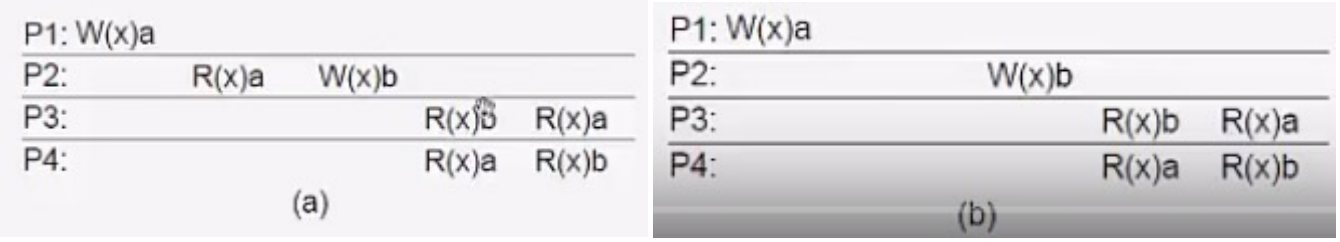
A 是 sequential consistent 的, 只要对于 p3 和 p4 上读顺序一致即可 – 因为并不能保证 p2 的写发生在 p1 的写之前; 而 b 不是 sequential consistent, 因为 p3 和 p4 的读结果有矛盾.



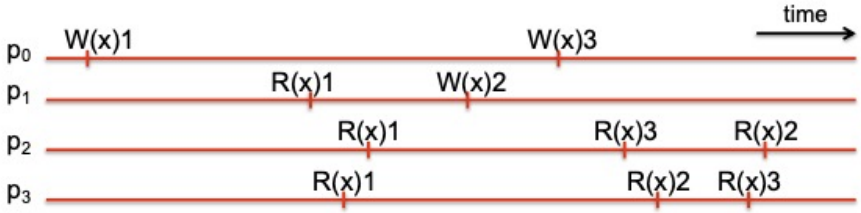
类似地这个也不是 sequential consistency.

Casual consistency

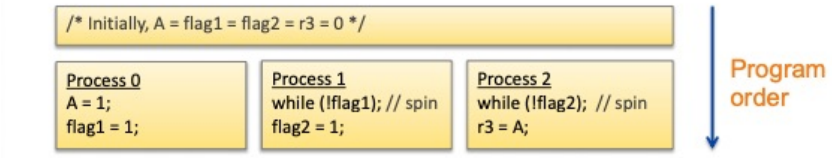
Writes that are potentially causally related must be seen by all processes in the same order. Concurret writes may be seen in a different order by different processes.



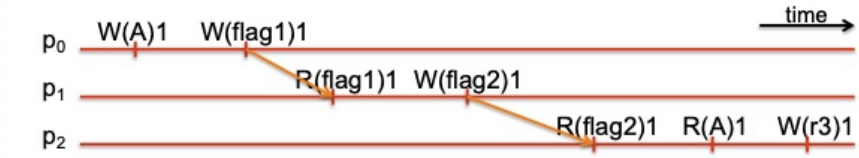
A is not casually consistent ; b is.
A: 因为 p2 于 writing b 前先 read 了 a, 证实了任何 read 都需要先看到 a 再看到 b. 所以 A 并不是 casual consistent.
B: 两个 write 之间没有 read 去验证先后, 所以视作 concurrent writes: Concurret writes may be seen in a different order by different processes.



这个里面 w1 和 w3 是 casually related , 因为它们发生在同一 thread 上并且有先后顺序. W2 和 w3 是 concurrent write, 所以不是 casually related.



- Causal consistency implies that at the end r3 = 1, because all the writes are causally related (we can order them with the *happen-before* relation)



考试问 program output 的话就写成时序图. 所有的 commercialized memory models guarantee casual consistency.

Replication

Replication 会遇到的问题: 我在悉尼传个文件上悉尼的 s3, 然后纽约的朋友马上就刷新去收, 这时候悉尼的 replica 还没有同步到美东的服务器, 就会产生 client-centric inconsistency.

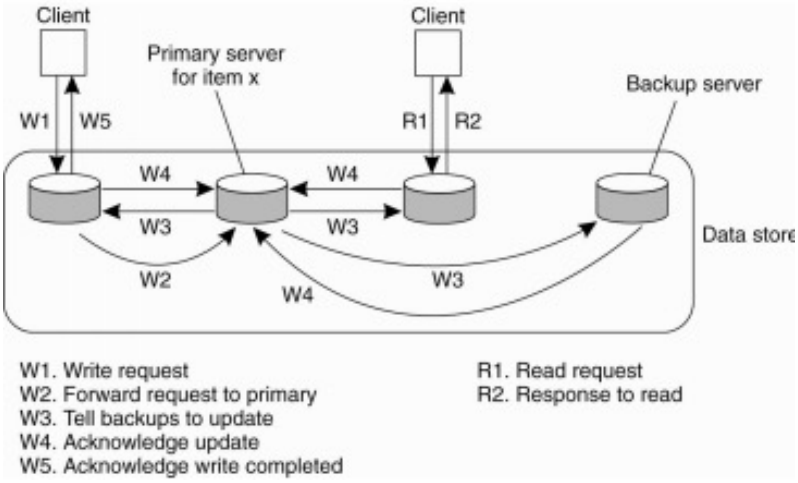
几种算法:

Primary backup

所有的 update 发生在 centre coordinator (primary server), 并且每次有 update 的时候就通知所有人需要同步, 并且这是一个 blocking execution – 在所有人完成同步之前没有人可以 read.

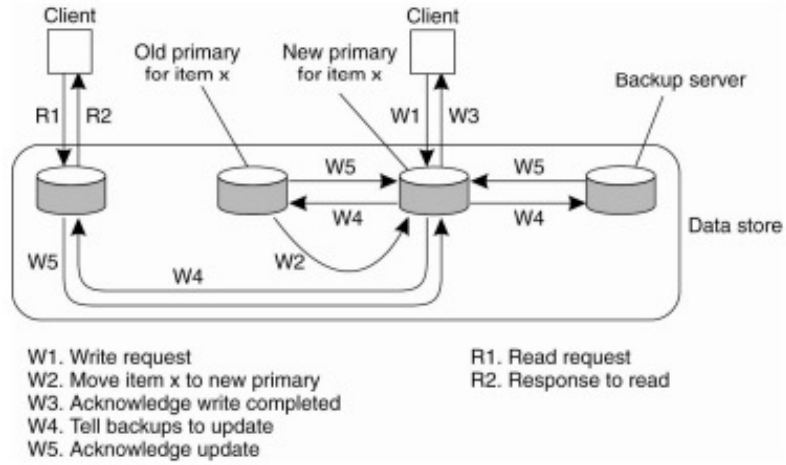
Pro: guarantees sequential consistency

Con: blocking execution, delayed answer to client.



Local-write primary backup.

和上面的原理一致, 但是不 block 了. 区别是上面的 w2 是 local forward 到 primary, 这里的 w2 是让 primary 变成 local.

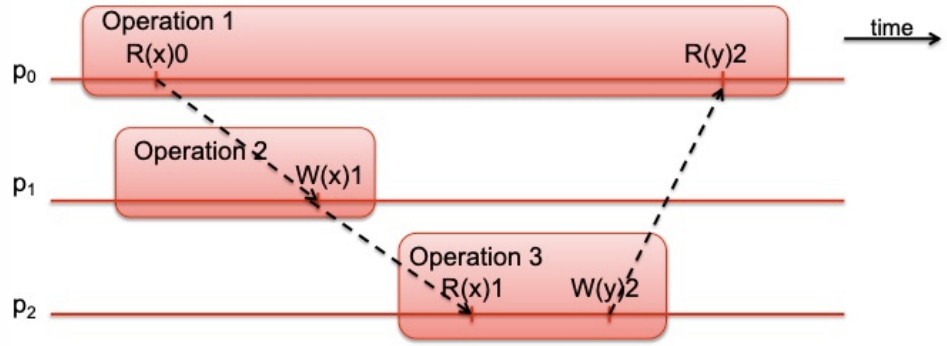


Multiple-access operations

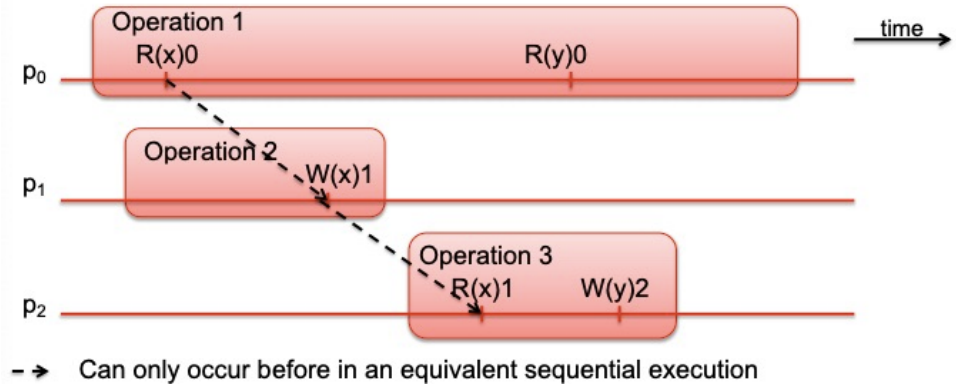
Definition: any single operation can now access multiple object (eg. Updating a single idex Multiple tables)
Data base transaction 的特性:

- Atomic: to the outside word, the transaction either completes or do not happen at all.
- Consistent: the transaction does not violate system invariants
- Isolated: concurrent transactions do not interfere with each other
- Durable: once a transaction commits, the changes are permanet.

The result of an execution is serializable if there exists an equivalent sequential execution. – 要求 process operation 之间没有环.



这个在 p0~p1, p1~p2, p2~p3 之间形成环, 所以无法 serialize

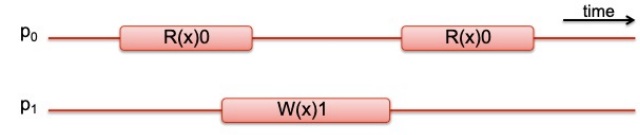


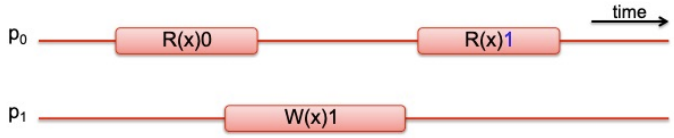
这个就没有问题.

Linearizabiltiy: real time precedence

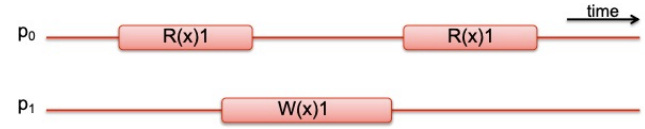
The result of each execution is equivalent to a sequential execution that respects the real time precedence: an operation returning before another is invoked is always ordered before.
The same as if the read&write operations by all processes on the same data store were executed in some sequential order and the operations of each individual process appear in this sequence in its program order.

如果一个操作在真实时间里已经 return 了, 就会对后续的操作可见.
例子 1: 假定一开始 x 都是 0.

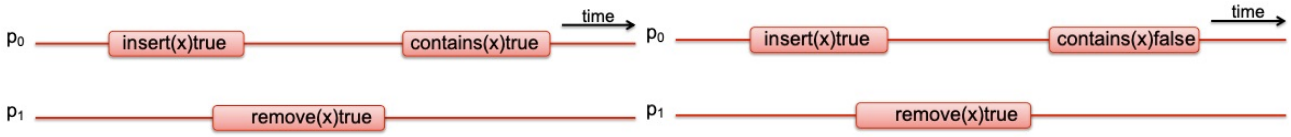




^ 这个例子 not linearizable. Process1 的 Write 1 在 process 0 的第二个 read 之前已经 return, 表明已经完成了写操作, 如果 linearlizable, 需要 process0 读到 1. 第二个例子就 linearizable.



这个例子也是 linearizable. 因为 r1 和 w1 有 overlap, 所以算作并行. P0 最开始 read 无论读到 1 还是 0 都符合 linearizable 条件. P0 第二次 read 必须读到 1 就可以了.

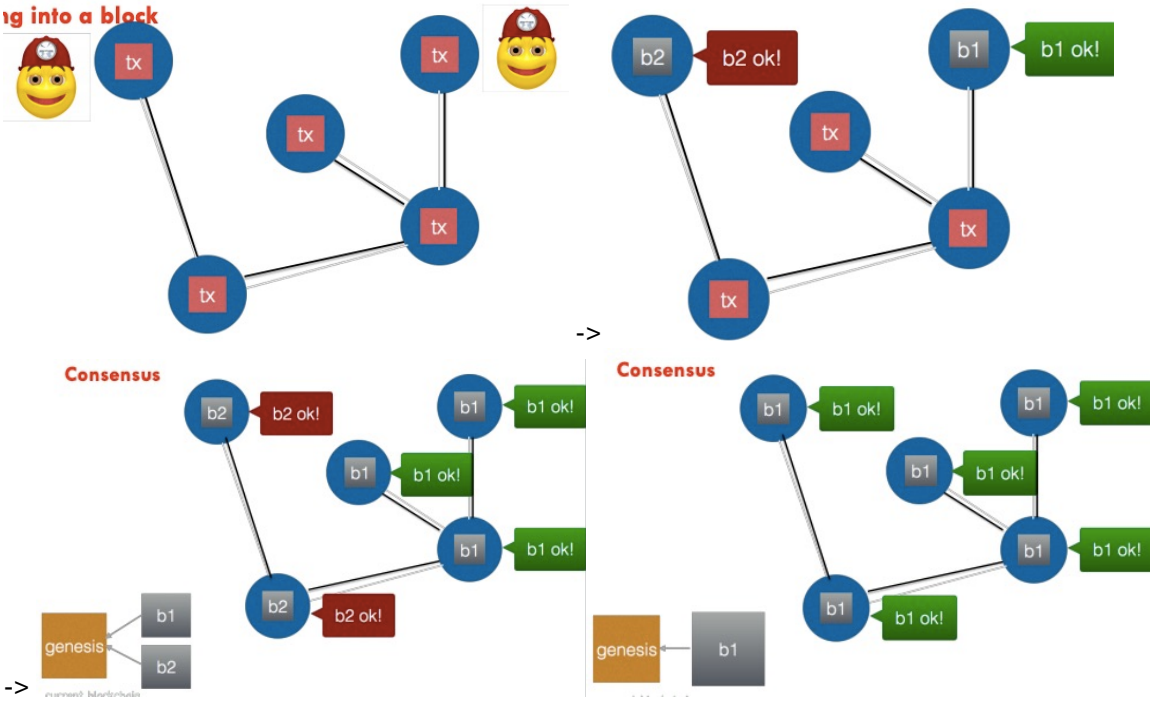


^ 这个例子 not linearizable. Process1 的 remove 操作在 process 0 的第二个 read 之前已经 return, 表明肯定 x 已经不在了. 如果 linearlizable, 需要 proecess0 第二次 contains x = false.

Week 8 falt-tolerence

Distributed implementation of blockchain

Block chain is a DAG with linked list. <b0, b1> is a pointer from current block b1 to previous block b0.
Pointer: representation of a hash of the destination block that the source block contains
Genesis block: special block known initially by all participants.
Distributed ledger: transactions can only be appended to the ledger (immutable)
Psudonymous: participants only identified by their address.
Gossip based protocol.
Peer to peer network: peers are both clients and servers (miners)
Public key cryptosystem: participant sign their transactions.
Consensus: for the distributed system to decide a block at index I.
如果一个新的 block 被加到 blockchain 里去所有人都会知道



这时候需要某种机制投票是加上 b2 还是 b1. 现在 b1 多, 我们加 b1 并让所有人知道最后我们加了 b1

一些定义

Failure: a system fails when it cannot meet its promises. 一个系统无法按照规定的期望产生结果就是 failure
An **error** is a part of a system state that may lead to a failure 一个系统会产生 failure 的某一部分
A **fault** is the cause of an error. 产生错误的原因

Transient fault: a fault that occues once then disappear. 只出现一次的 bug (??我希望我的 bug 都是这样的
Intermittent fault: a fault that occurs, vanishes, reappears. 一会儿有一会儿没有的 bug (???
Permanent fault: a fault that continues to exist until faulty component is replaced. 不 fix 就不会消失的 bug...

Availability: the probability that the system is operating correctly at any given moment. 如果一个系统一小时里有 1ms 不 available, 那 availability > 99.9999%
Reliabiltiy: the ability for a system to run continuously without failures. 如果这个系统一直在跑是否可以一直保证没有 failure?

Safety: a property of a system in which nothing bad happens if it fails. 如果 fail 了没有任何坏的副作用
Maintainabiltiy: the easiness for a system to be repaired.

Failure Models

Type of failure	Description
Crash failure 停机之前若可以保持工作那就是 crash failure	A server halts, but is working correctly until it halts
Omission failure 漏收/发信息	A server fails to respond to incoming requrests (if only receiving fails, called receive omission; if only sending fails, called send omission)
Timing failure 超出时间/过早反应	Server response outside the specified time interval
Response failure	A server's response is incorrect. (if thre **value of response** is wrong, called **value failure**; if server **deviates from the correct flow of control**, called **state transition failure**)
Arbitrary (Byzantine) failure – most serious.	Produce randoms responses at random times. – could be any type, any time!

What is byzantine failure: a server producing output it should never have produced, but which can not be detected as being incorrect; it might also maliciously working together with other servers to produce intentionally wrong answers

Consensus

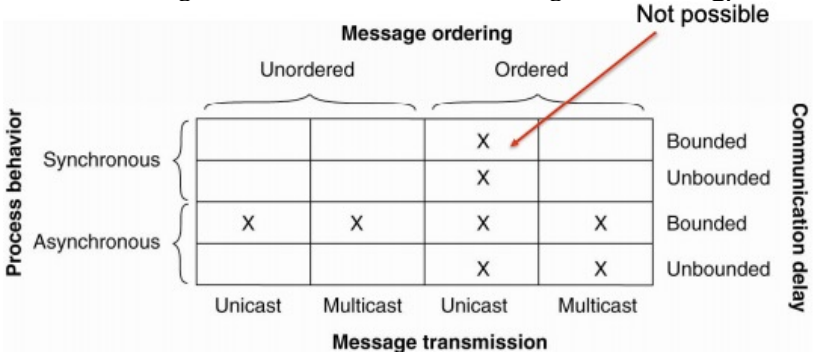
如何获得 fault-tolerance?

- Resilience by grouping: Organize identical processes into a group and achieve by replicating processes, and when a message is sent, all member receives it; when a member fails, other processes takes over.
- Group management: creates new groups and destroys old group. A process joins and leaves at any time; be a member of multiple groups.

K fault tolerant: A system is k fault tolerant if it **can survive faults in k componets** and still meets specifications.

为什么 distributed consensus 很难 achieve:
有四种情况(四个 boolean)

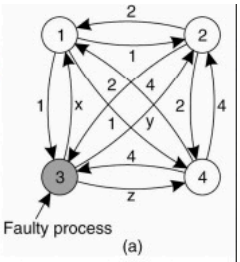
1. **Synchonous/asychonous:** a system is **synchronous** it there exist some constant c>=1 such that if any processor has taken c+1 steps, every other process has taken at least 1 step.
2. Communication delay is **bounded/not bounded:** if theres a system max time to get a message delivered.
3. Message delivery is **order/not ordered:** ordered – the sequence of delivered message is the same as the order they are sent.
4. Message transmission is done through **unicasting/multicasting**



Bazantine-tolerant synchronous solution:

Byzantine-tolerant synchronous solution

- Example of solution for n=4 and f=1
  - (a) every process sends its input to all others
  - (b) the collected vectors (i^{th} coordinate received from
  - (c) every process sends the collected vector and obtains these 3 new vectors
  - A majority vector is computed using for each coordinate the one appearing a majority of times (or ⊥ if there is none)
  - (1, 2, ⊥, 4)



		1 Got	2 Got	4 Got
1	Got(1, 2, x, 4)			
2	Got(1, 2, y, 4)	(1, 2, y, 4)	(1, 2, x, 4)	(1, 2, x, 4)
3	Got(1, 2, 3, 4)	(a, b, c, d)	(e, f, g, h)	(1, 2, y, 4)
4	Got(1, 2, z, 4)	(1, 2, z, 4)	(1, 2, z, 4)	(i, j, k, l)

For a system with k faulty processes, agreement can be achieved only if 2k+1 correctly functioning process are present.
如果需要找 asynchronous communication 里同样的答案: no solution.见上表.

Week 9

Blockchain nodes:

- Wallet; Miner; Full Blockchain; Network routing node.
- Miner: compete to create new blocks.

Mining

Mining is the process of new bitcoins is added to the money supply. It secures the bitcoin system against fraudulent transactions.
Mining gets rewards:

1. new coins are created with each block. (打包奖励)
2. Transaction fees from all the transactions are included in the block.(手续费)

Mining reward cut in half every 4 years, precisely every 210000 blocks: Fixed monetary supply, resists inflation; tend to be inherently deflationary.

Miners compete to solve proof of work from cryptopuzzle.

Miners:

1. every bitcoin node including neighbours first verify the transaction
2. add them into a memory pool (transactions wait until they can be included into a block)
3. Miners receiving a new block means someone else has won the current competition. Time to find the next block.

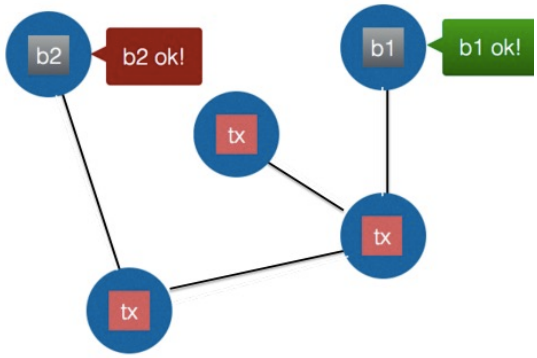
Make a new block

- Transactions that are old and high value gets selected into current block. First 50 KB is for high priority blocks, the rest filled up with lower priority ones. The rest remain there into the next block.
- First transaction in a block is the mining reward coinbase transaction. 打包奖励.
- Block header 里有什么: version number, **Previous block hash, merkle root,** timestamp, **difficulty target, nonce.**
  - **Previous block hash:** hash of the block header of pervious block known to the miner.
  - **Merkle root:** binary hash tree, contains summary info of all transaction selected to be included in the block. 把两个 transaction 计算 SHA256 两两合并, 直到只剩下最后的一个值.相当于一个高效的 hash. Can check if any transaction is included in the tree with at most $2\log_2(N)$ calculations.
  - **Difficulty target**: mining is the process of hashing the block header repeatedly changing the nounce until the resulting hash is less than a threadsold (difficulty target)

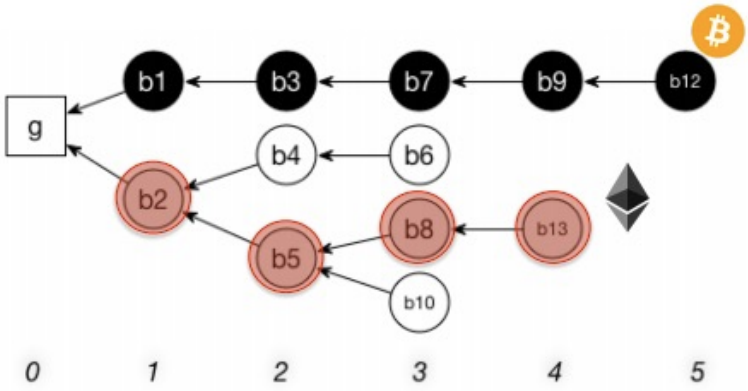
    - **Nonce**: the proof of work for the computation trying to find the nounce. Finding the nonce takes time, but validating the nonce is easy. Smaller the threshold, harder the computation is.
- **Why bitcoin threasold is adjusted every 2016 blocks (2 blocks)?**
    - To keep 10 minutes block interval.
    - New difficulty = (old difficulty * actual time of last 2016 blocks) / 20160 minutes.

Once a miner finds a matching nounce, miner immediately transmits the block to all peers; each peer validates th block with a series of tests and propagate the block to the network. (validation is easy; miners do not act dishonestly)

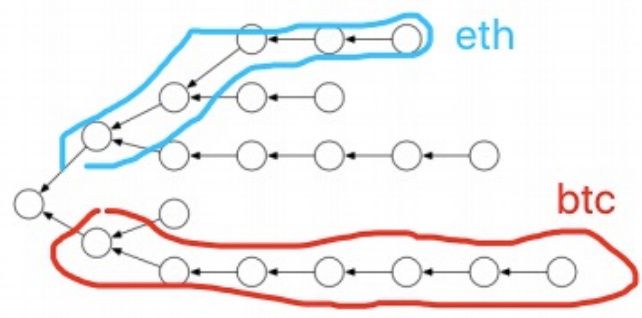
Resolving fork



当这种情况出现的时候就是有 fork 了 – 两个 miner 在很接近的时间里打包了新的 block.都说下一个要加自己的 block, 这时候需要一种机制比较应该听谁的.



Bitcoin 会选最长的 node.
Etherum 会每次选择 child subtree 最多的 node. (下图 eth 就是 ghost protocol)



A block is *commited* if it belongs to a decided block.

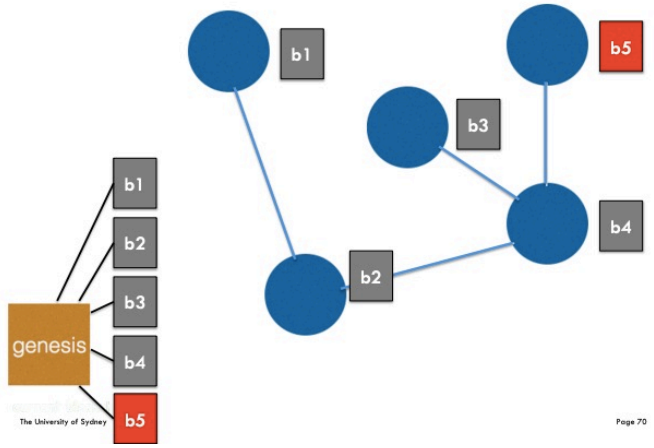
Q7: What is the PoW mechanism used for, and what is its major drawback?
PoW: you can stop Sybil attack with PoW. – can't fake identity easily.
Drawback: Inorder for no one can take control over the network, the difficulty to find a proof of work becomes increasingly difficult. It is energywise expensive.

Impact of mining power

51 % attack :
因为 bitcoin 认定选择更长的 chain, 如果黑恶势力有了很高的算力就能高速计算, 产出恶意的 block 来让 bitcoin 选择自己这条恶意的 chain. 理论上如果 51%的算力被掌控你就可以为所欲为.

Impact of communication delay

(假如)大家都下线去 mining 然后突然上线, 就会产生一大堆 fork – 大家长度都一样, 听谁的.
实际上是这样的 – propergation takes time



Q8: why can't we increase the size of each block to speed up blockchain?
You need time to transmit the block information without too much delay. If the size of block gets too big the above delay related attack can happen. It also will cause great network load.
Networking wise bitcoin is not very efficient.

Q10: in private blockchain, without changing Ethereum protocol, how could we speedup the creation of blocks?
If you know all the nodes and chains (private chain)
You can relax the difficulty – since you know there are no malicious ones.

Q11: Why can't we reduct the difficulty of a crypto-puzzle to speedup bitcoin or Ethereum (public)?
There will be many forks -> leads to arguments.

Week 11

安全常用的一些名词
Security policy: actions entities of the system are allowed to take, and which ones prohibited.
Security mechanisms:

- Encryption: trasfering data into something an attacker cannot understand.
- Authentication: verifying the claimed identity of an entity – 某个人是否是 xxx?
- Authorization: verifying whether the entity has the rights to perform the action it requires 给与一些用户行为权
- Auditing: action of monitoring which entity access what and how 监控谁可以看什么, 怎么看

Cryptography simple example
$C = \mathrm{E_k}(P), P = D_k(C)$
Bob encrypts plaintext P into ciphertext C, and Alice must decrypt C to read original P.

Passive/Active intruder

- Passive intruder only listens to C.
- Active intruder can alter/insert messages.

为什么需要 cryptography: **prevent interruption, modification, fabrication**

Encryption

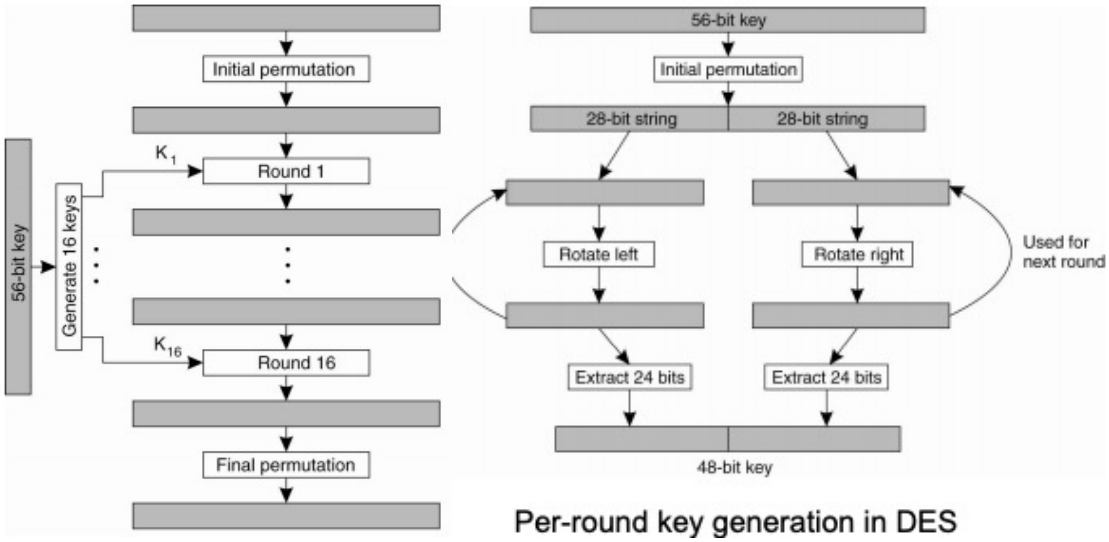
- Secret key cryptosystem (symmetric): 私钥加密: 一把钥匙加密与解密. 仅有发件与收件人知道这个钥匙. 代表算法: DES, triple DES.
  the same K is used to both encrypt and decrypt the message. P = $D_k(E_k(P))$

- Public key(asymmetric) cryptosystem: 公钥加密: 加密人与解密人用不同的两个 key, 其中某个 key 是公开的.

two separate keys K_E for encryption and K_D for decryption.
P = D_{K_D} ($E_{K_E}(P)$) one of the key is public while another one is private.

私钥加密: DES (data encryption standard)

- 用于加密 64bits block data. 首先数据被分成 64bit 块, 先重排列一次, 加密 16 次, 最后再反向重排列一次. 每次加密用一个不同的 48bitkey.
- 这个 48bit key 是从一个 56 bit masterkey 上每次通过分割+array shifting 产生的. 首先把 56 bit master key 分成两半, 左边一半 left rotate 1-2 个 bit, 右边一半 right rotate 1-2 个 bit (rotate 的意思就是移动 x 个 bit). 然后合并在一起产生一个 48 bit 的 key



Per-round key generation in DES

DES 好处: simple to be implemented on smartcard; resistant to analytical methods. DES 坏处: can be broken using brute-force attack.
Triple DES: 重复 DES 三次 encrypt-decrypt-encrypt, 广泛使用.

公钥加密 RSA

1. Choose large prime number p and q
2. Compute n = pq and z = (p-1)(q-1)
3. Choose a number d that releatively prime to z
4. Compute the number e such ed = 1 mod z.

只有 e 或者 d 里的一个数用作 public key.
Sender encrypts message into fixed length blocks m_i. Each block m_i into $c_i = m_i^e (mod\ n)$;
Receiver decrypts each block $m_i = c_i^d (mod\ n)$

RSA 好处: 比 DES 计算上更复杂. 加密时间更长. 用于加密传递"key"的信息, 而不是真正的数据. Used essentially to exchange encrypted keys.

Authentication

Definition: authentication is the process whereby one party is assured of the identity of a second party involved in a protocol, and the second has actually participated.

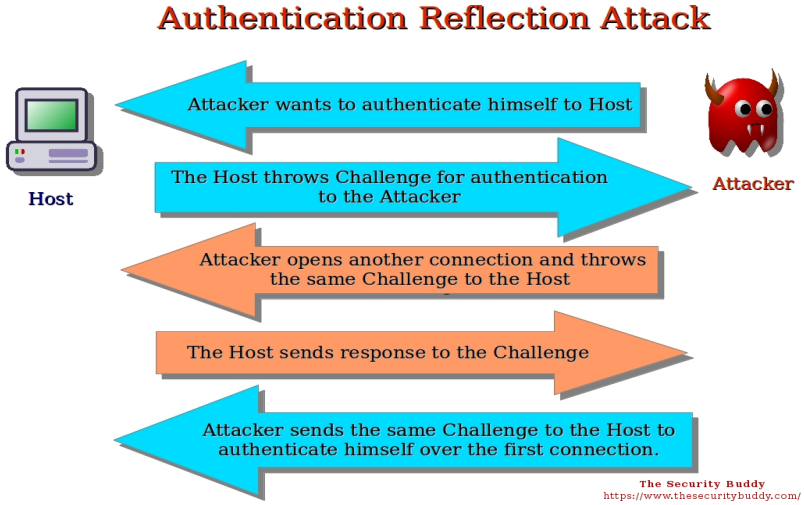
Corroborative evidence.

- **Possession:** something the user has – Physical key
  Possession 存在的问题:
  容易丢. 如何界定 identification.
- **Inherence:** something the user is – face scan
  Inherence 存在的问题: cannot be changed, if compromised lost forever. 如果一旦指纹被盗, 没法换个手.
- **Knowledge:** something the user knows – password
  Knowledge 存在的问题:

人很懒不改密码.
密码容易猜. (prone to brute force attack).
Attackers can observe typing passwords in many different ways (keystrokes)
Social media helps collect information about a person.

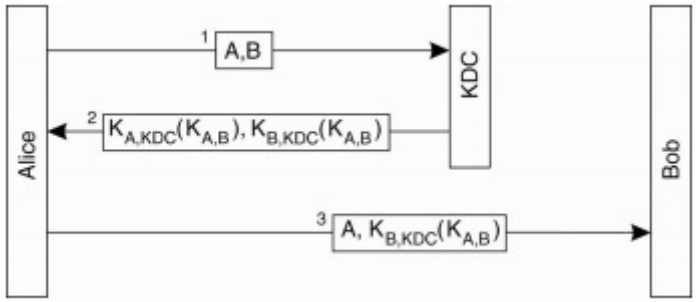
Process between 2+ parties.
Involvement and participation of the second party.

为什么单纯的 secret key authentication 不行:
Reflection attack 弱点在于彼此用同一个 protocal 太容易泄露信息.



Secret key Authentication 最好用 KDC 方法:

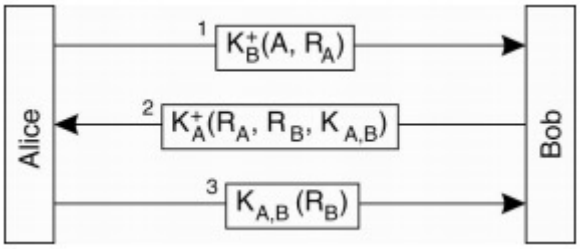
- 需要交换的次数最少. 用到的 key 最少.
  1. alice sends a message to KDC with a challenge
  2. The KDC responds with a ticket (2)
  3. Alice sends A and $K_{B,KDC}(K_{A,B})$ of this ticket to Bob who decrypts it.



Public key authentication – 非对称加密, 公钥加密, 是最好的!

1. Alice 用 Bob 的公钥加密自己的 identity
2. Bob 收到后用 bob 的私钥解密, 获得 Alice 的 identity, 用自己的 Alice 的公钥加密一个 session key 并返回给 Alice
3. Alice 收到后用 alice 的私钥解密, 用 bob 给的 session key 开始之后的通信.

好处是不需要像私钥加密那样需要 centralized KDC



HTTPS: HTTP over TLS

- 先做 TCP handshake, 再 TLS handshake:
  - client sends it capabilities (TLS version) and a random number R1.
  - Server has a certificate containing public key and corresponding private key.
  - Server replies with a selected capability… another random number R2.
  - Server also sends the certificate to the client. Client verifies the certificate.
  - Client generates pre-master key and encrypts using server's public key, sends it to the server using client key exchange message.
  - Both server and client compute the master key – a combination of the pre-master key, r1, r2.
  - Client sends a change cipher spec notification

什么是 certificate: a certificate is a text file containing some information such as company name, domain name and public key. A **certificate Authority** proves your claimed identity.

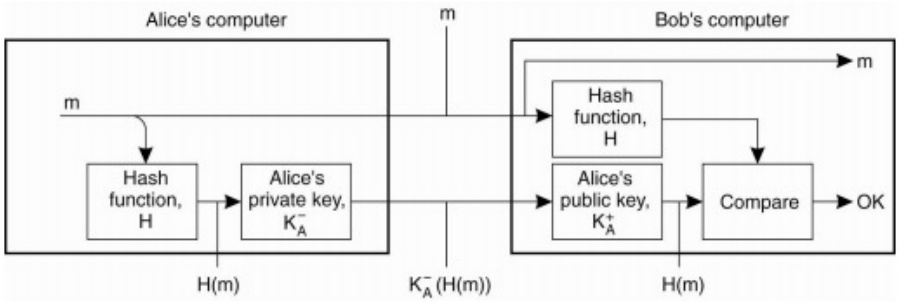
Integrity

用 hash function 来保证信息不被篡改.

Hash function are used to produce a hash H of fixed length given a message m: h = H(m).
One-way function: H = F(P) is easy. Find P from H is hard.
Weak collision resistant: not easy to find two different inputs producing the same hashed output.

Sender 在发送信息的同时 send 一个 checksum.(也就是 hash 信息, H(m))
Receiver 收到信息的同时 hash 信息, 对照检查 checksum 看看信息有没有被改变.

Public key 与 hashfunction signature: 更安全



Alice 在发送信息的时候要用 private key 保护自己的 checksum.

Privacy

Personal information: any information that identifies you or could reasonably be used to identify you.
E.g. Name, address, financial details, photos, health information, criminal record… Device ids, MAC address, contact list, call history, location, installed apps.