以太坊的点对点网络协议栈 - DevP2P

戎佳磊
garyrong0905@gmail.com
rjl493456442

大纲

- DevP2P简介
- Kademlia协议
- 节点发现协议
- 常见的discovery协议攻击
- 传输层协议

1.DevP2P简介

1. DevP2P简介

- DevP2P是用来组成**以太坊点对点网络**所使用到的一系列网络协议栈
- DevP2P的目标不是仅服务区块链网络，而是服务所有需要点对点网络的应用层协议
  - ETH, LES(区块链)
  - Swarm(去中心化存储)
  - Whisper(去中心化消息系统)

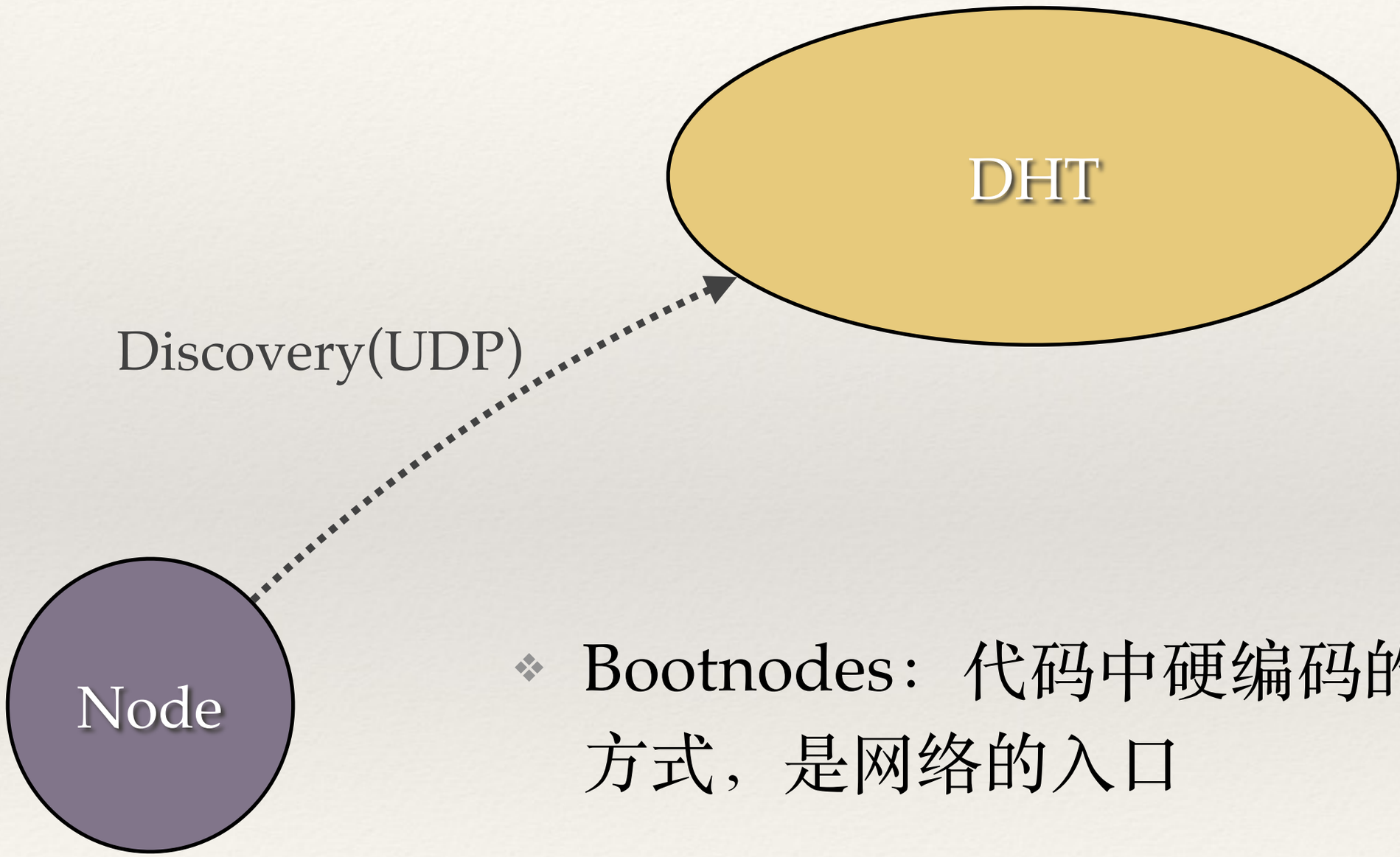
1. DevP2P简介

- DevP2P协议栈中包含以下几类子协议
  - 传输层协议 - **RLPx**
    - 网络参与方之间的加密通信
  - 节点发现协议 - **DiscV4**
    - 个体节点能够通过该协议寻找到网络中的其他节点
  - 基于Topic的节点发现协议 - **DiscV5**

1. DevP2P简介

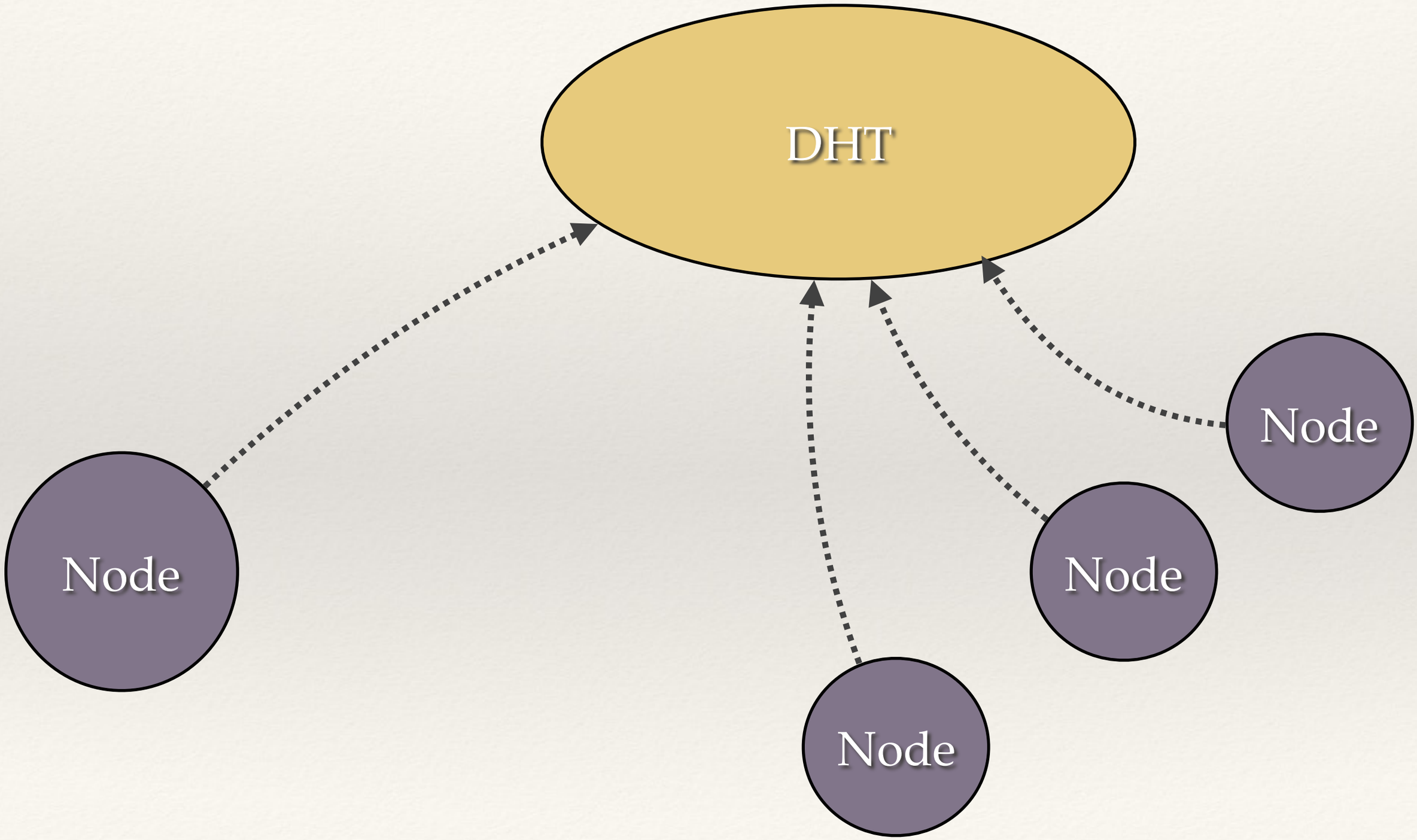
- C#: Nethermind https://github.com/tkstanczak/nethermind
- C++: Aleth https://github.com/ethereum/aleth
- C: Breadwallet https://github.com/breadwallet/breadwallet-core
- Elixir: Exthereum https://github.com/exthereum/ex_wire
- Go: go-ethereum/geth https://github.com/ethereum/go-ethereum
- Java: Cava RLPx library https://github.com/consensys/cava/tree/master/rlpx
- Java: EthereumJ https://github.com/ethereum/ethereumj
- Java: Pantheon https://github.com/PegaSysEng/pantheon
- JavaScript: EthereumJS https://github.com/ethereumjs/ethereumjs-devp2p
- Kotlin: Cava Discovery library https://github.com/consensys/cava/tree/master/devp2p
- Nim: Status eth_p2p https://github.com/status-im/nim-eth-p2p
- Python: Trinity https://github.com/ethereum/trinity
- Ruby: Ciri https://github.com/ciri-ethereum/ciri
- Ruby: ruby-devp2p https://github.com/cryptape/ruby-devp2p
- Rust: Parity Ethereum https://github.com/paritytech/parity-ethereum

1. DevP2P简介

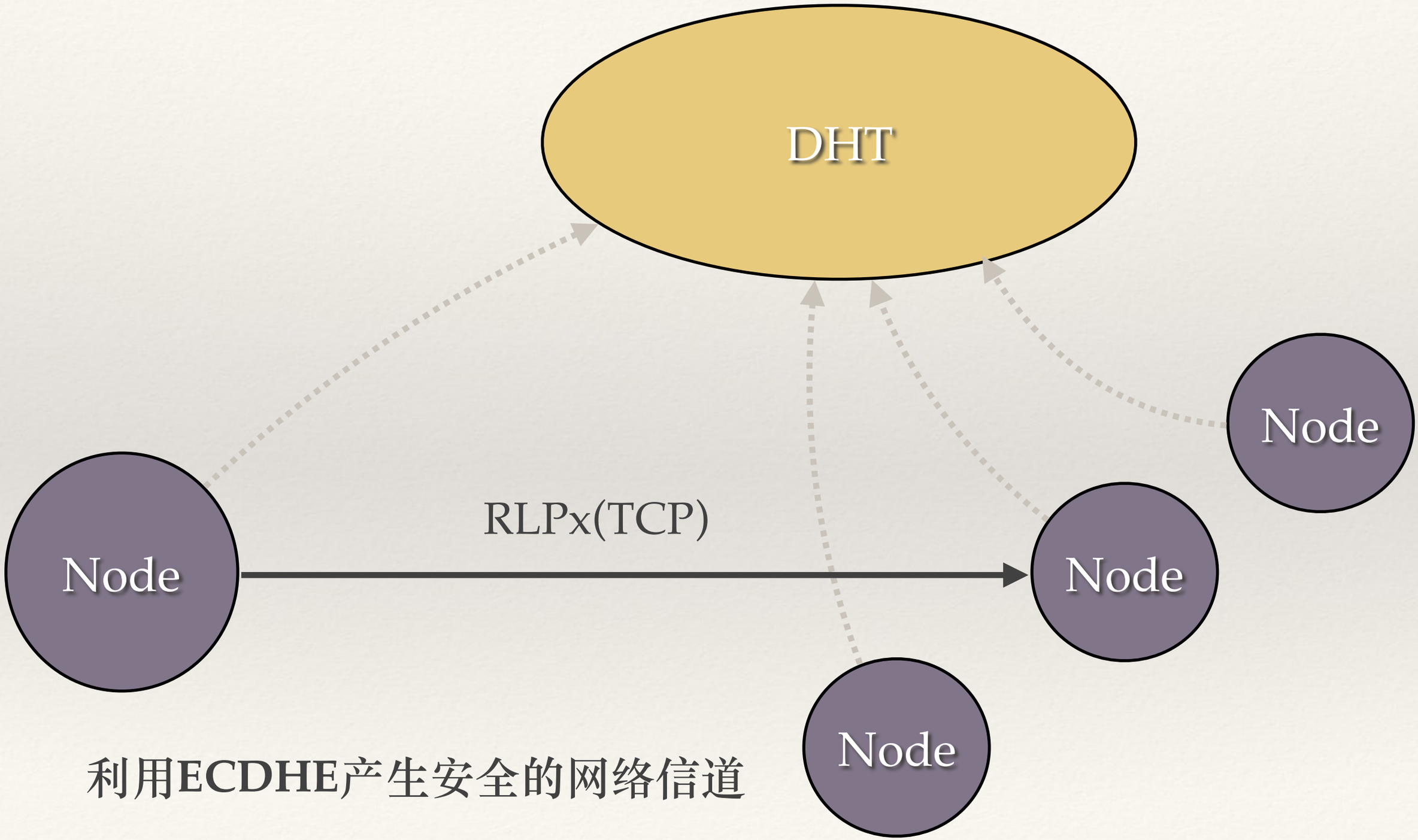


- Bootnodes：代码中硬编码的节点连接方式，是网络的入口

1. DevP2P简介

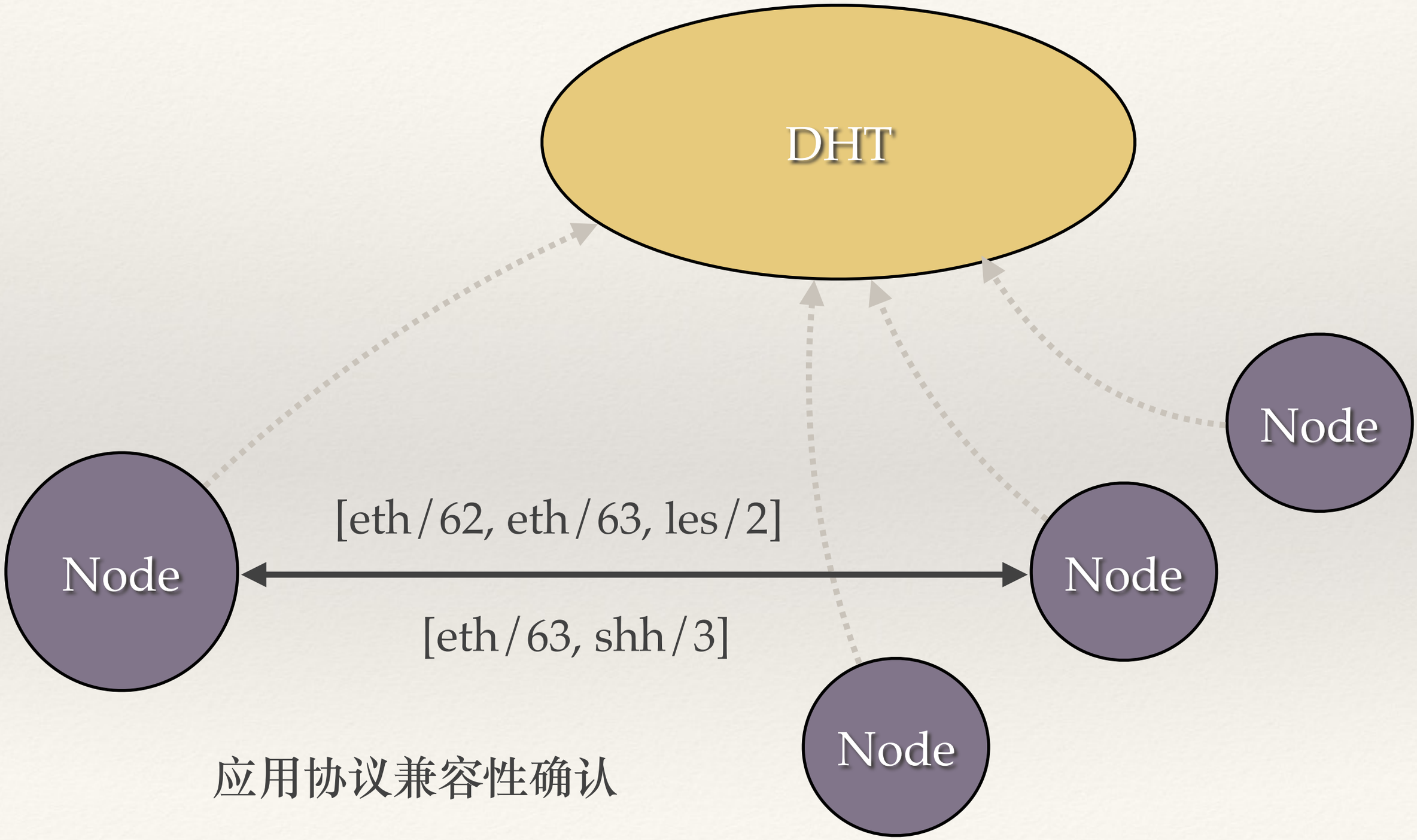


1. DevP2P简介



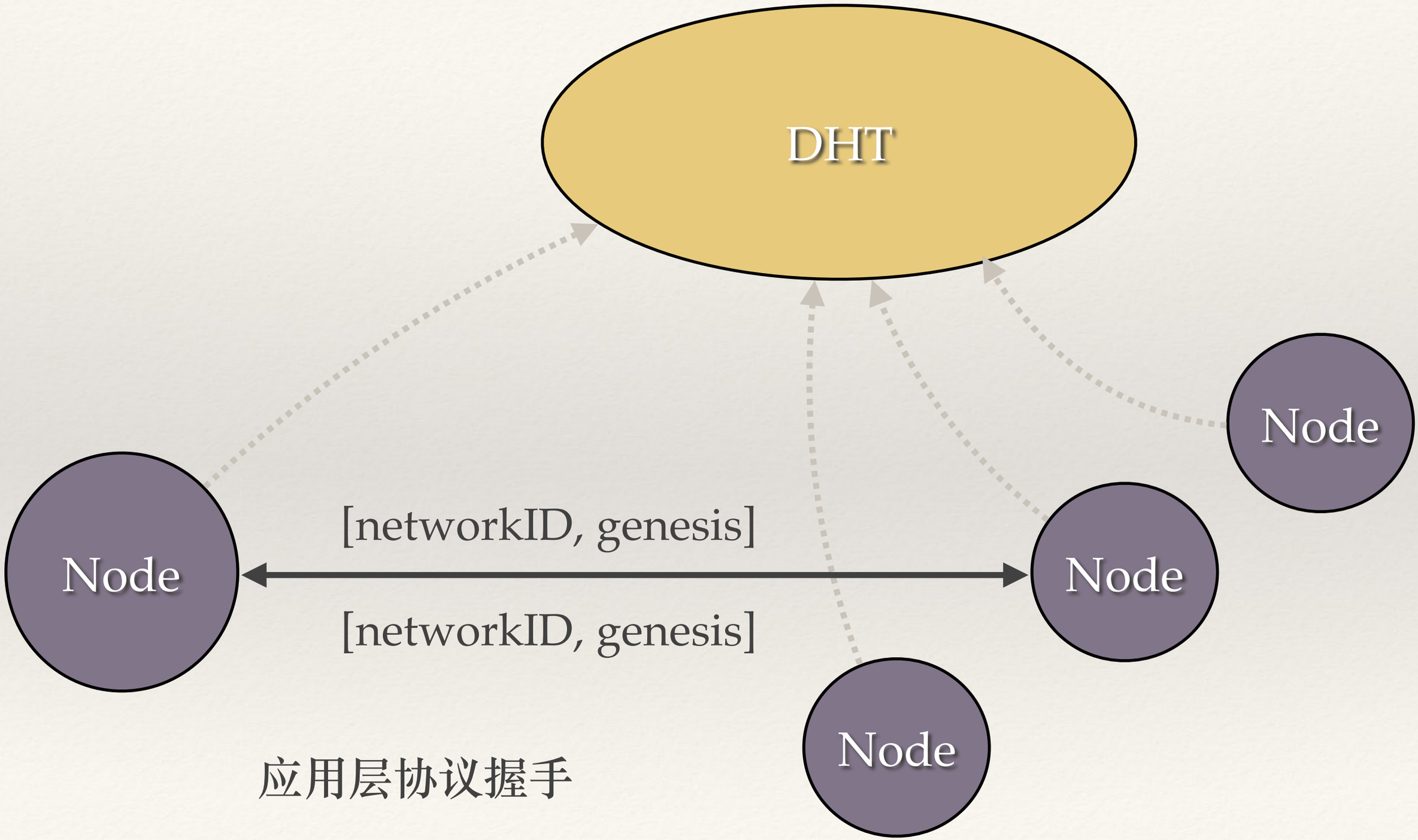
利用ECDHE产生安全的网络信道

1. DevP2P简介

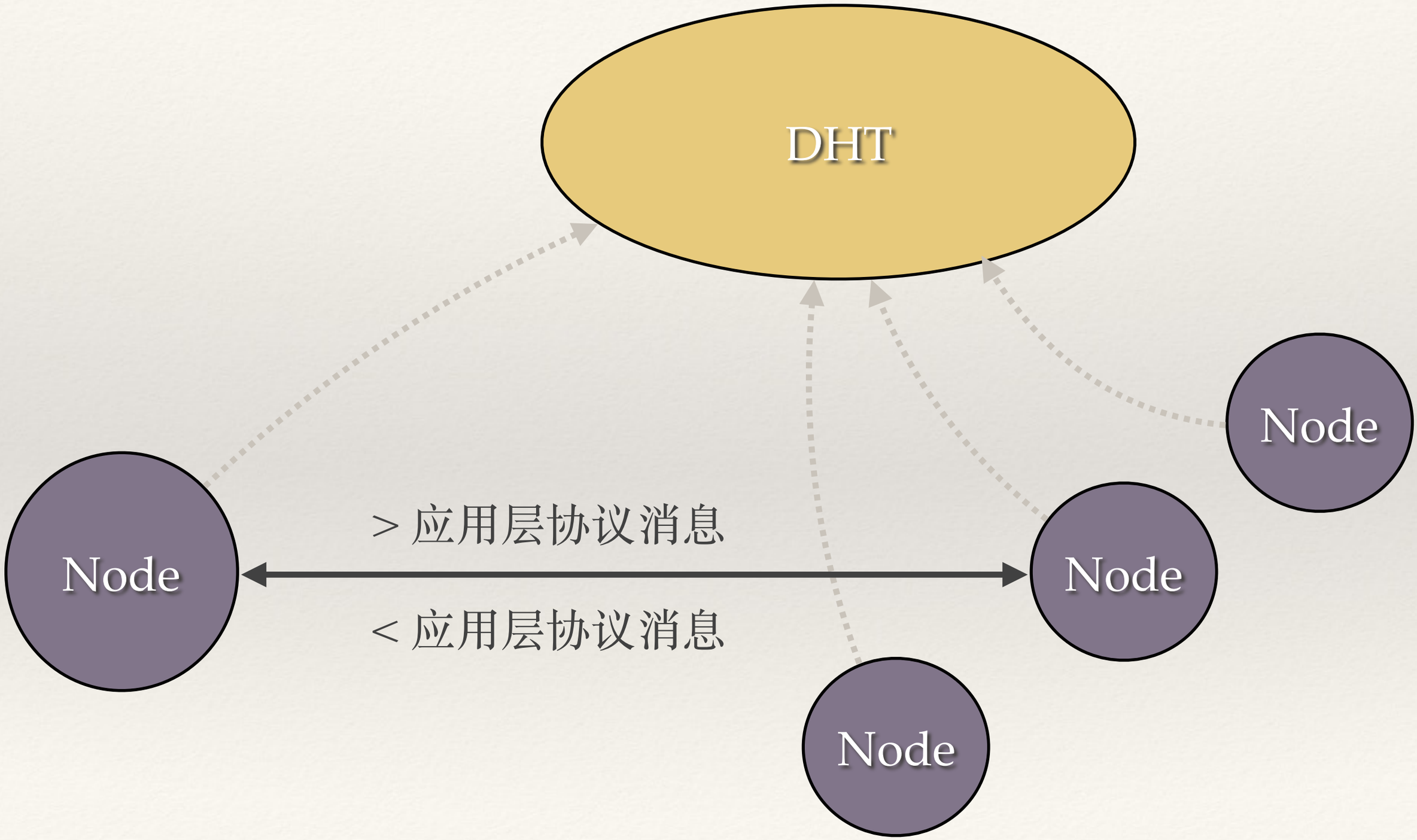


应用协议兼容性确认

1. DevP2P简介


DHT
Node
Node
Node
Node
[networkID, genesis]
[networkID, genesis]
应用层协议握手

1. DevP2P简介



2. Kademlia协议

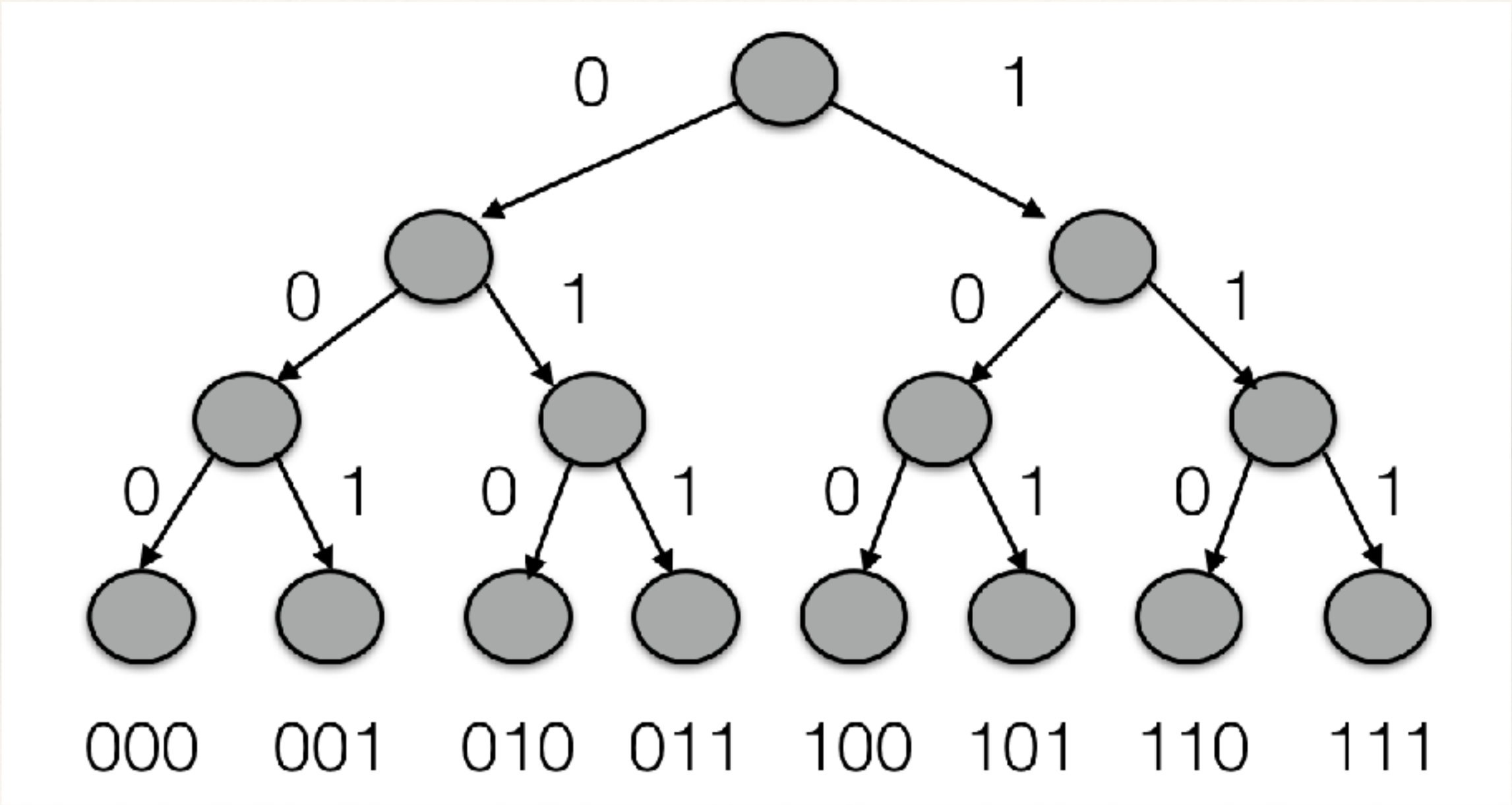
2. Kademlia协议

- Kademlia在2002年由美国纽约大学的Petar和David所提出
- Kademlia被电驴、BitTorrent等P2P存储应用所采用
- 以异或运算为距离度量公式
- Kademlia协议是一种高效的以分布式哈希表（DHT）的方式来进行节点管理的机制

2. Kademlia协议

- Kad协议将整个网络设计成了一个二叉树
- 每一个叶子结点代表一个网络节点
- 每个节点有长度为160Bit的唯一标识
- 二叉树的层高为160层
- 每个网络节点在二叉树中都有一个唯一的位置

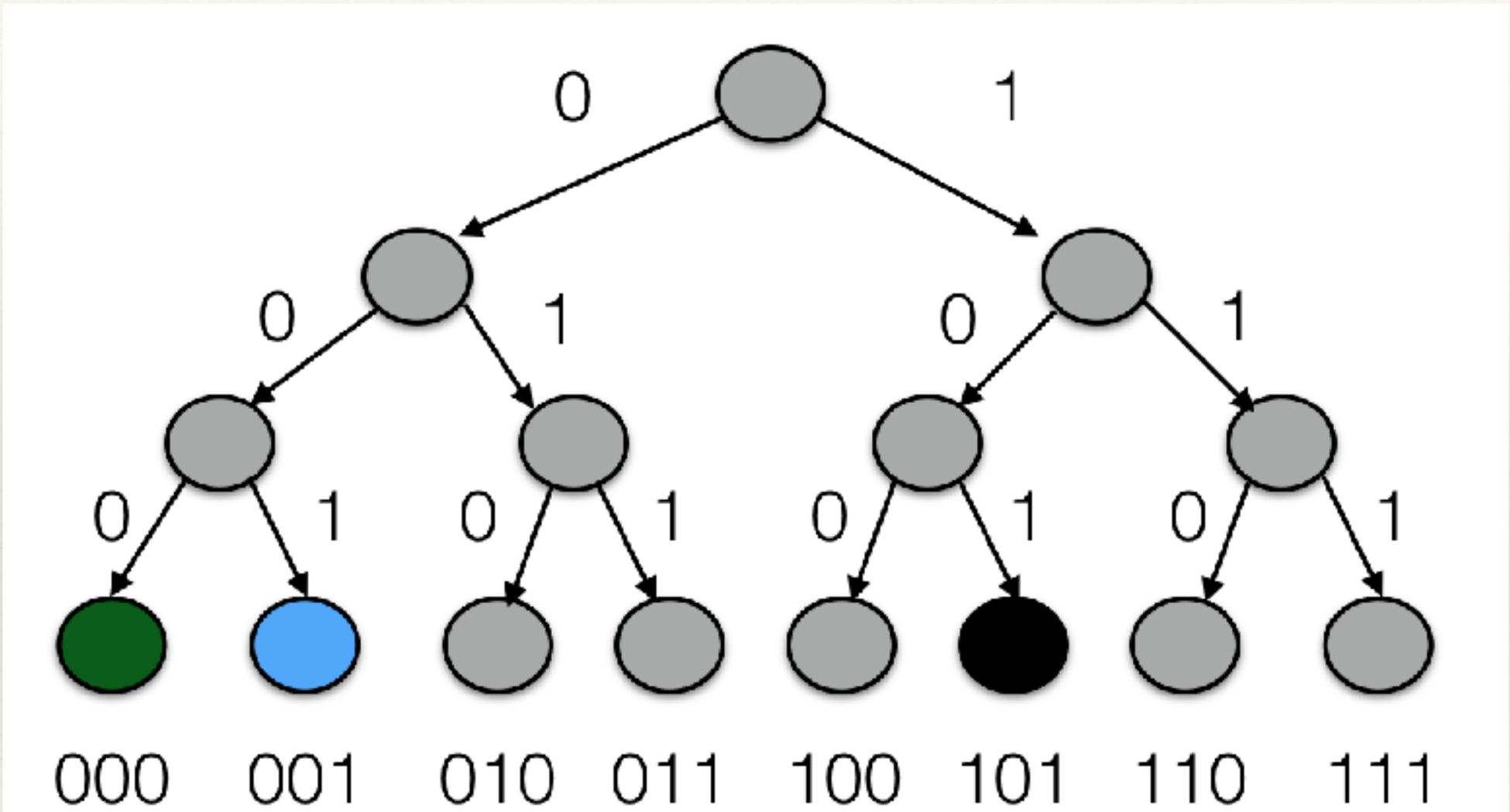
2. Kad中的网络二叉树



2. Kad中的节点距离计算公式

- Kademlia使用独特的异或运算来计算节点之间的距离
- X ^ X = 0（本地节点与本地节点之间的距离为0）
- X ^ Y > 0 （两个不同的节点之间必定存在距离）
- X ^ Y = Y ^ X （交换律，X到Y的距离等同于Y到X的距离）
- X ^ Y + Y ^ Z >= X ^ Z （两边之和大于第三边）
- 异或运算具有单向性（即给定一个节点与距离，存在唯一对应的节点）

2. Kad中的节点距离计算公式



000 ^ 001 = 1

000 ^ 101 = 4+1 = 5

2. 每个节点维护的“K桶”

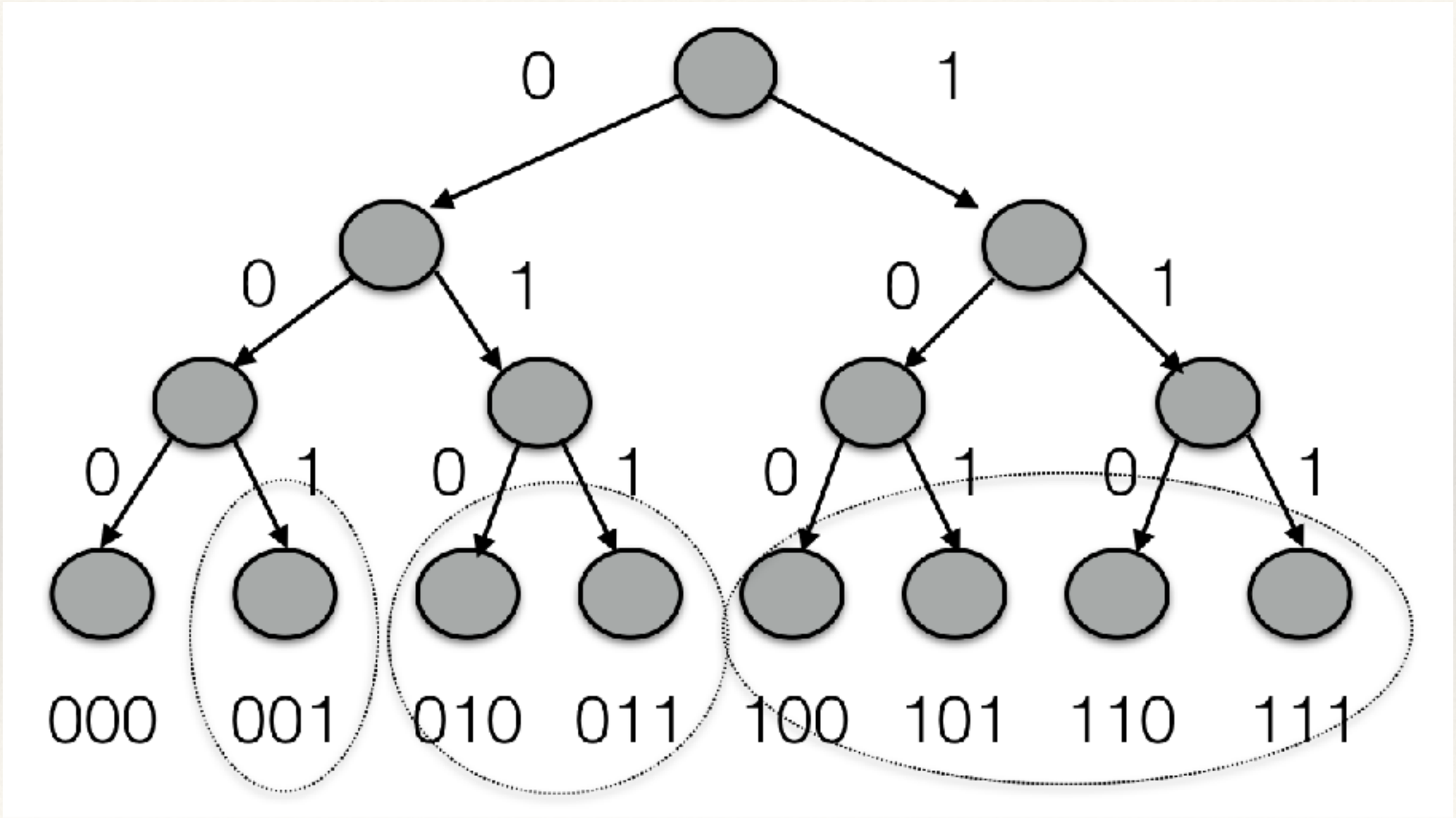
- Kad协议规定，每个网络节点都在本地维护一系列K桶
- 节点ID的位数等于K桶的个数
- 每个K桶都存储与本地节点的“距离”落在当前范围内的远端节点
- K桶i表示的距离范围为[2^i, 2^(i+1))
- 每个K桶内存储的节点个数为固定值P

2. 每个节点维护的“K桶”

K桶	存储的距离区间	存储比率	相同前缀
0	[1, 2)	100%	159
1	[2, 4)	100%	158
2	[4, 8)	100%	157
3	[8, 16)	100%	156
4	[16, 32)	50%	155
…			
i	[2^i, 2^(i+1))	8/2^i	159-i

每个K桶的容量取为8

2. 每个节点维护的“K桶”



2. K桶的更新

- 每当节点收到一条网络消息，就会更新一次K桶
- 根据计算的异或距离，将远端节点的信息（ID, IP, PORT） 存储到对应的K桶
- 若对应的K桶已满
  - 判断该桶中最不活跃节点的活性
  - 若该节点仍然具有活性，则放弃新节点
  - 实践证明，累积在线时间越长的节点越稳定
  - 当大量恶意节点涌入时可以抵御DHT污染
- 定时检查K桶内节点的活性

2. K桶的更新

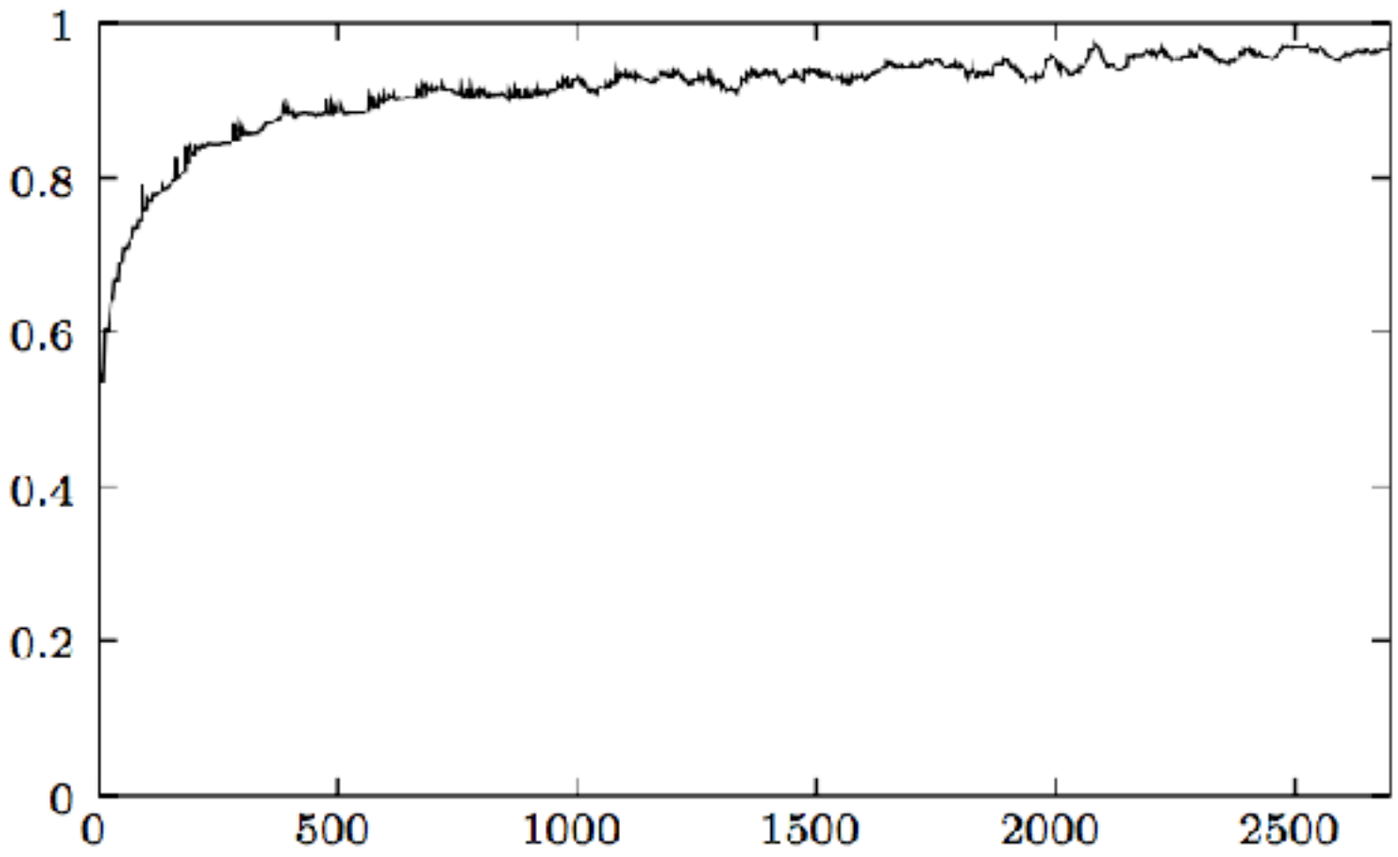
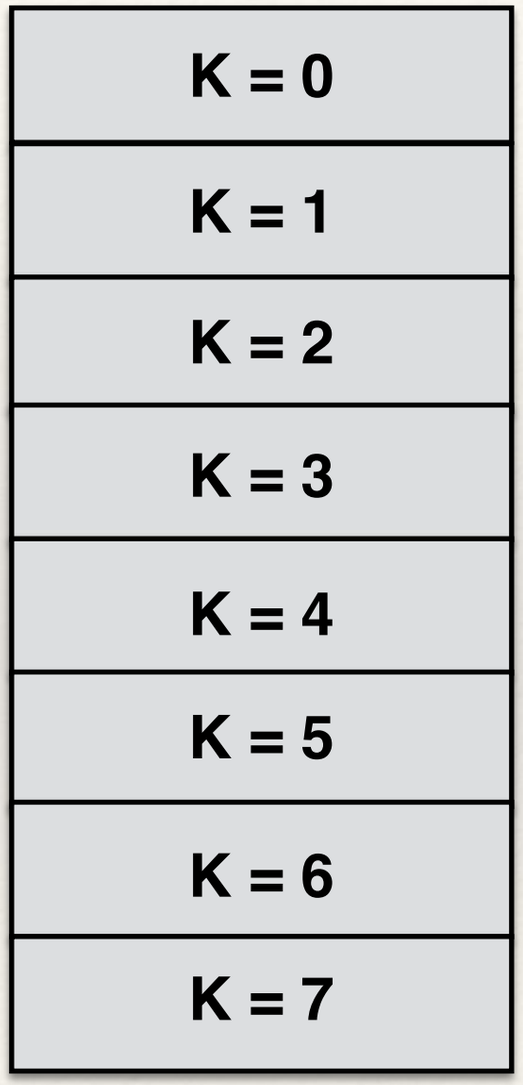


Fig. 3: Probability of remaining online another hour as a function of uptime. The x axis represents minutes. The y axis shows the the fraction of nodes that stayed online at least x minutes that also stayed online at least $x + 60$ minutes.

2. kad协议中的四种操作

- Ping: 探测一个节点是否在线
- Store: 令对方存储一份数据
- Find Node: 根据节点ID查找一个节点
- Find Value: 根据键值查找一个文件块

2. Find Node过程



本地节点:01011011
目标节点:10111110

2. Find Node过程

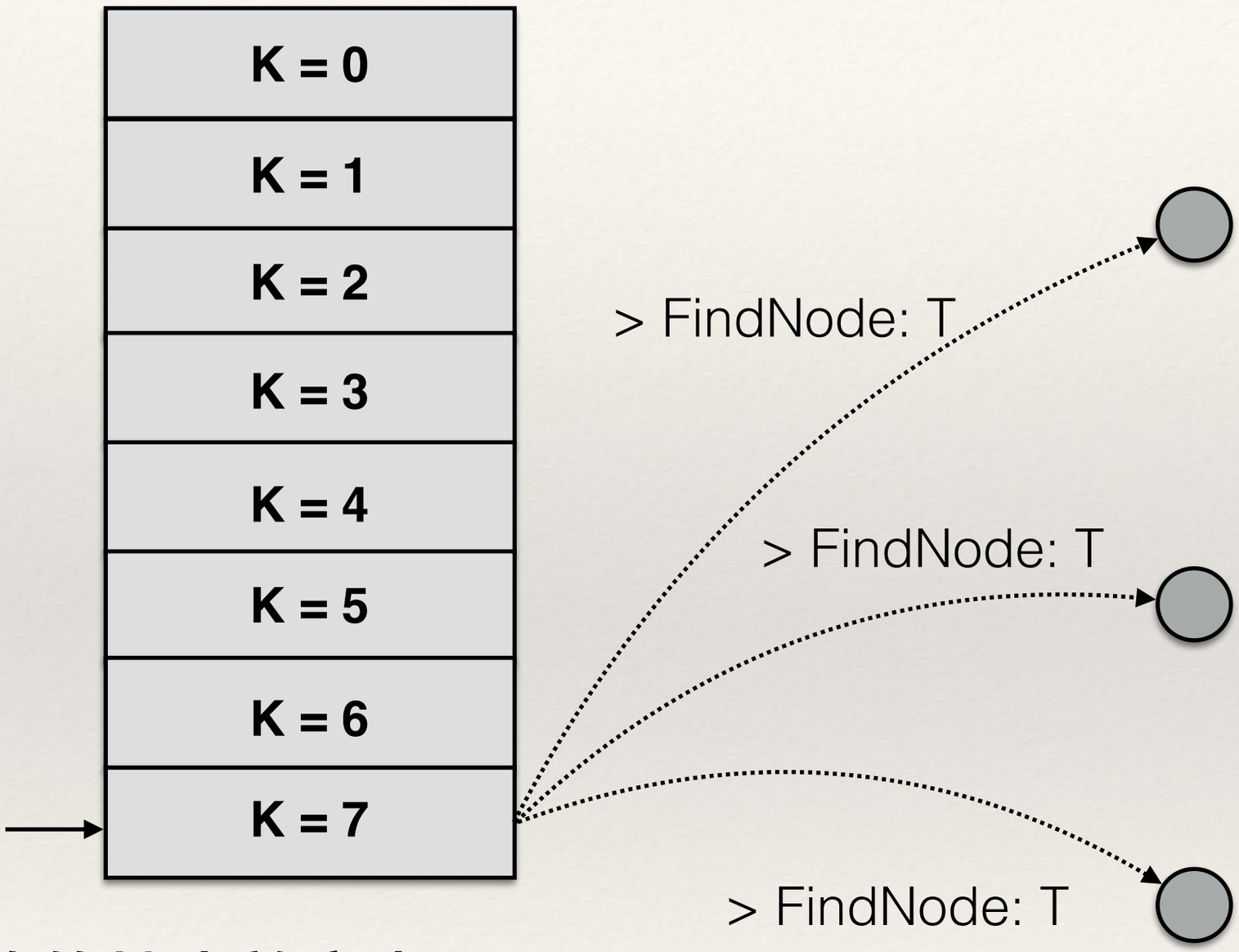
K = 0
K = 1
K = 2
K = 3
K = 4
K = 5
K = 6
K = 7

本地节点:01011011
目标节点:10111110

共享前缀长度为0

步骤	共享前缀长度	桶ID	节点
1	0	7	11000010 10101010 10001010
2			
3			
4			

2. Find Node过程



本地节点:11000010
目标节点:10111110
k桶：6

本地节点:10101010
目标节点:10111110
k桶：4

本地节点:10001010
目标节点:10111110
k桶：5

最终的搜索效率为Log(N),
N为节点总数

2. 节点发现过程

- 节点刚启动时，向P2P网络发起一次查找“自己”的FindNode过程 -》广播自己的信息
- 进行若干次查找“随机ID” 的FindNode过程 -> 填充自己的K桶

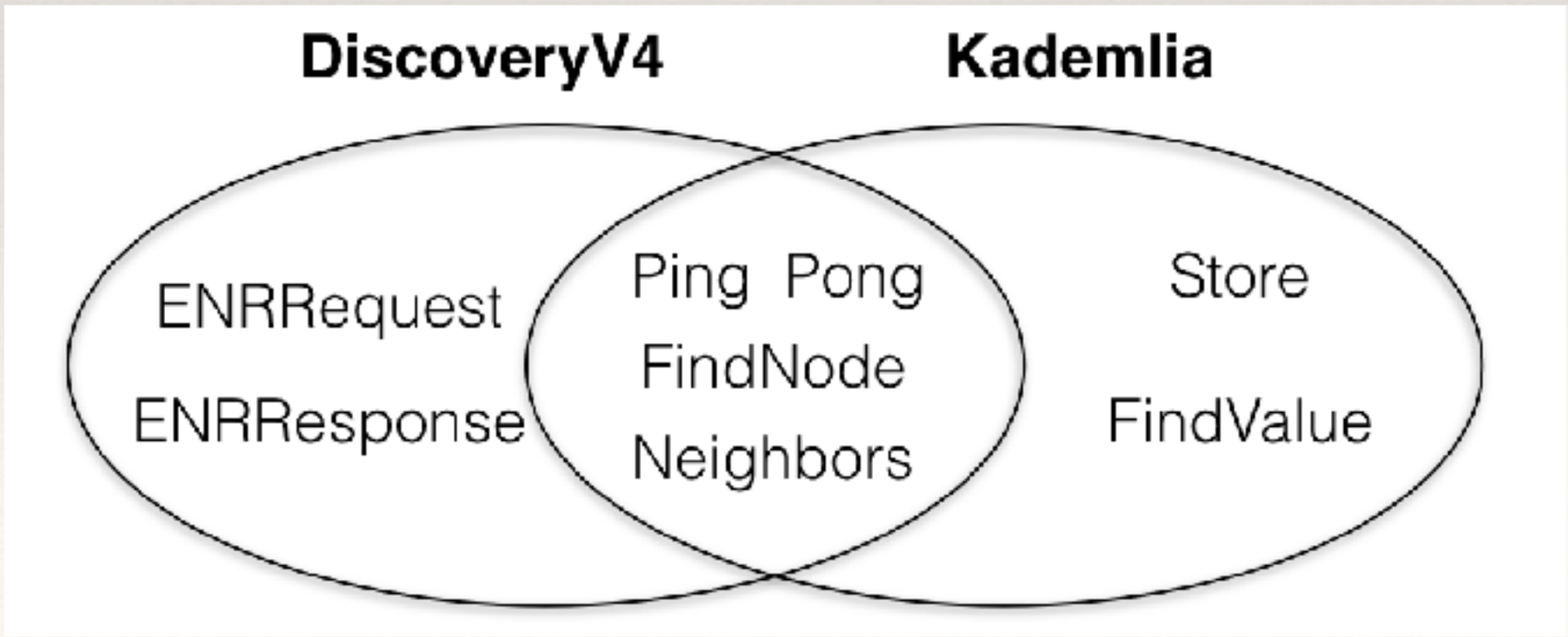
2. Kad的其他操作

- 存储文件：利用Find Node找到与文件块哈希逻辑距离最接近的n个节点，然后发送Store指令进行存储
- 存储数据的节点发现有其他节点与文件哈希的逻辑距离更接近时，进行转储
- 获取文件：利用Find Value进行查找，Find Value与Find Node基本一致，差别在于当节点存储目标哈希对应的文件时，直接返回文件
- 加入网络：对本地节点的ID进行Find Node，完成节点信息的广播
- 离开网络：无需任何操作，一段时间后本地节点的消息会从DHT中过期

3. 节点发现协议DiscV4

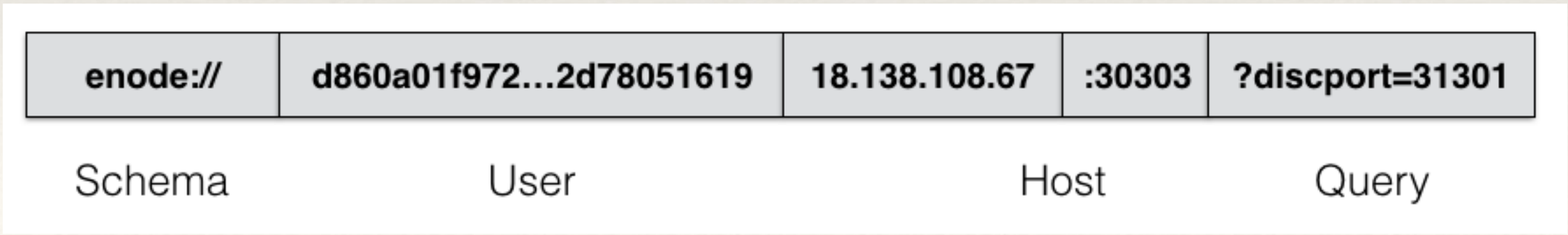
3. 节点发现协议

- 以太坊的节点发现协议DiscV4是Kademlia协议的变种
- 保持了Kad协议中的Ping, Pong, FindNode, Neighbors四种基本消息类型，删去了有关文件存储的消息类型
- 新增了ENRRequest, ENRResponse两种有关于ENR的消息类型



3. 节点发现协议 - 节点的URL表示

- Schema: 表明URL的模式，一般为enode
- User: 节点公钥序列化后的16进制表示
- Host: 节点IP与端口，在这里端口指传输层的TCP端口
- Query: 当UDP端口与TCP端口非共享时，在query字段指明用于discovery的端口



3. 节点发现协议 - Bootnode列表

```
// Ethereum Foundation Go Bootnodes

"enode://d860a01f9722d78051619d1e2351aba3f43f943f6f00718d1b9baa4101932a1f5011f16bb2b1bb35d
b20d6fe28fa0bf09636d26a87d31de9ec6203eeedb1f666@18.138.108.67:30303",   // bootnode-aws-
ap-southeast-1-001

"enode://22a8232c3abc76a16ae9d6c3b164f98775fe226f0917b0ca871128a74a8e9630b458460865bab4572
21f1d448dd9791d24c4e5d88786180ac185df813a68d4de@3.209.45.79:30303",     // bootnode-aws-
us-east-1-001

"enode://ca6de62fce278f96aea6ec5a2daadb877e51651247cb96ee310a318def462913b653963c155a0ef6c
7d50048bba6e6cea881130857413d9f50a621546b590758@34.255.23.113:30303",   // bootnode-aws-
eu-west-1-001

"enode://279944d8dcd428dffaa7436f25ca0ca43ae19e7bcf94a8fb7d1641651f92d121e972ac2e8f381414b
80cc8e5555811c2ec6e1a99bb009b3f53c4c69923e11bd8@35.158.244.151:30303",  // bootnode-aws-
eu-central-1-001
```

3. 节点发现协议 - 启动

- 当节点想要在启动时找到DHT中的邻居节点时，必须要向部分DHT中已知节点发起FindNode请求
- 已知节点包含两部分：Bootnodes和NodeDB中存储的历史节点

3. 节点发现协议 - 节点距离

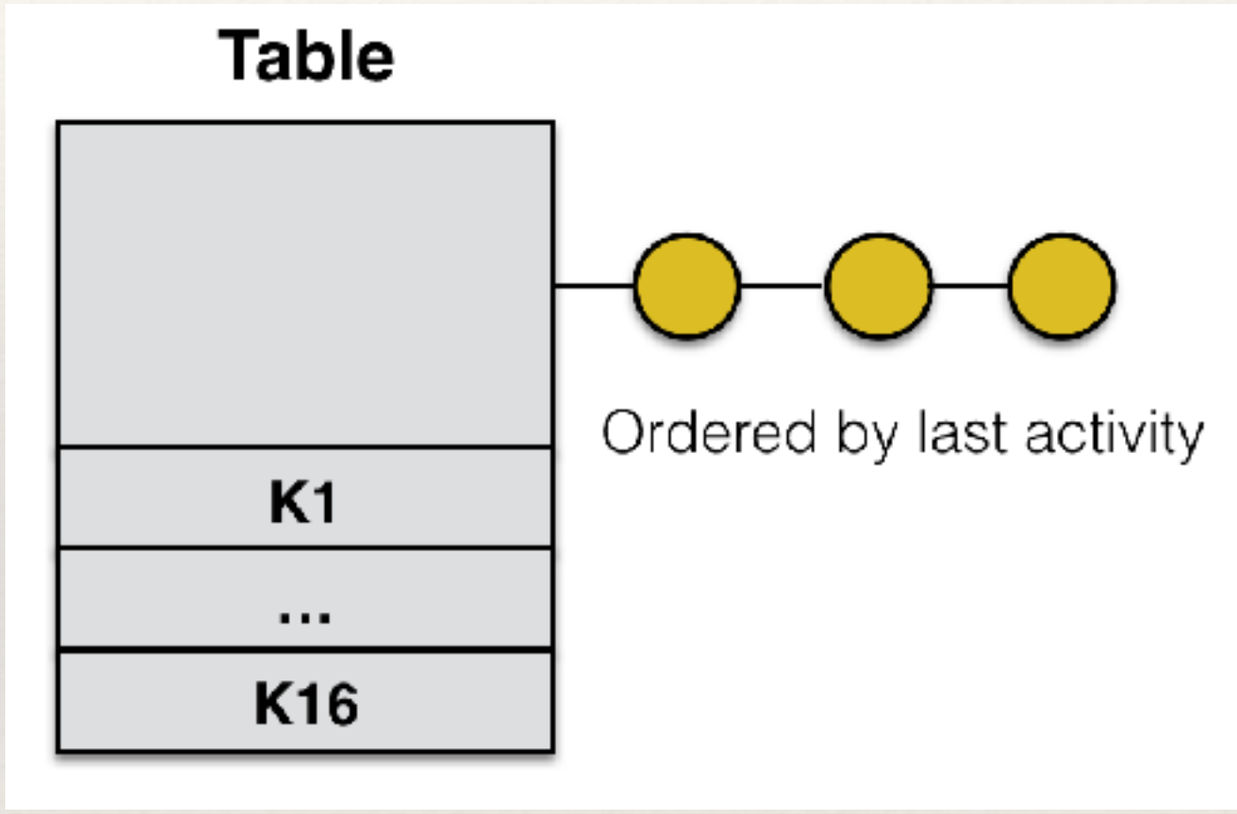
- 网络中的每个节点都有一个密码学上唯一的“标识”：一个secp256k1椭圆曲线上的一个Key
- 这个Key公钥的哈希信息为该节点的标识符，或者称为NodeID
- 两个节点之间的逻辑距离通过NodeID之间的异或距离表示：
  - distance(n_1, n_2) = n_1 XOR n_2
  - 实现时为了简化运算过程，使用log距离进行代替，即“NodeID哈希长度”减去“共享前缀的长度”

3. 节点发现协议 - K桶

- DiscV4中的K桶共有17个（非256个）
- 当相同前缀的长度大于等于16时，所有的节点都放在0号K桶
  - 现实工程中，产生两个具有16个相同前缀的节点几率是非常小的
- 当相同前缀的长度小于16时，剩余的节点依次放在i号K桶
- 每个桶内节点按照“最近活跃的时间”顺序排列

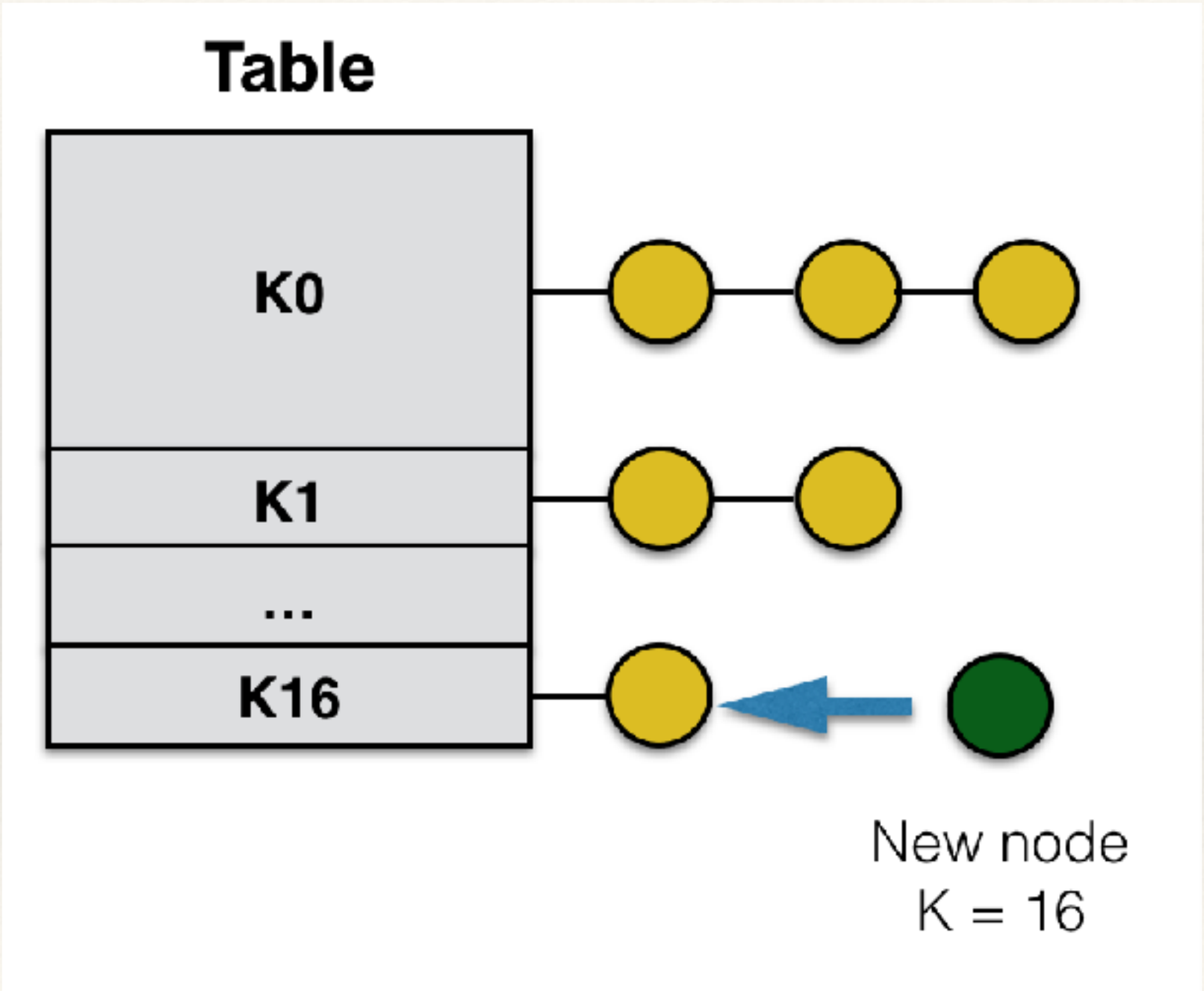
3. 节点发现协议 - K桶

K桶	相同前缀
0	16 - 255
1	15
2	14
3	13
4	12
…	
16	0

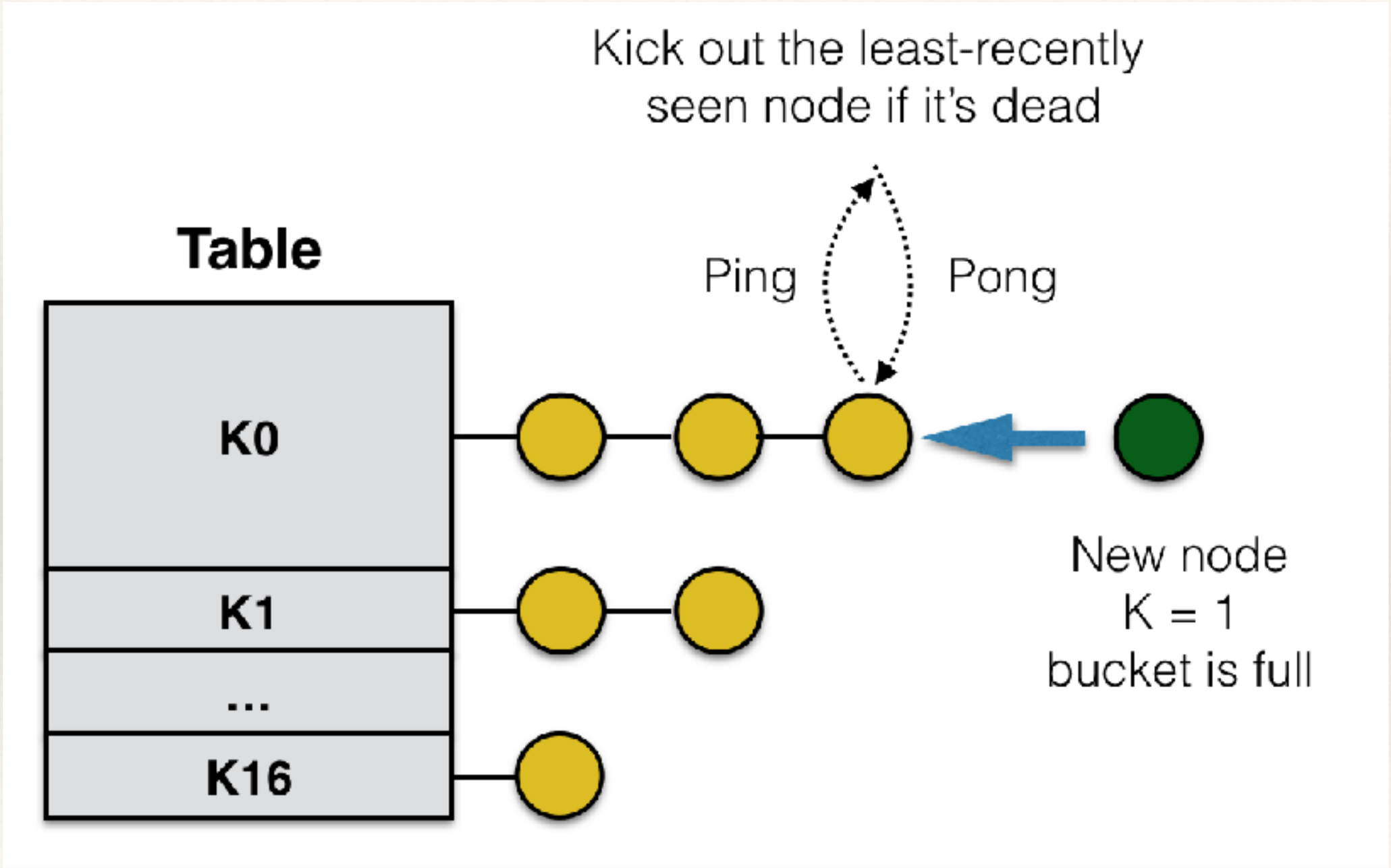


每个K桶的容量取为16

3. 节点发现协议 - K桶更新（1）



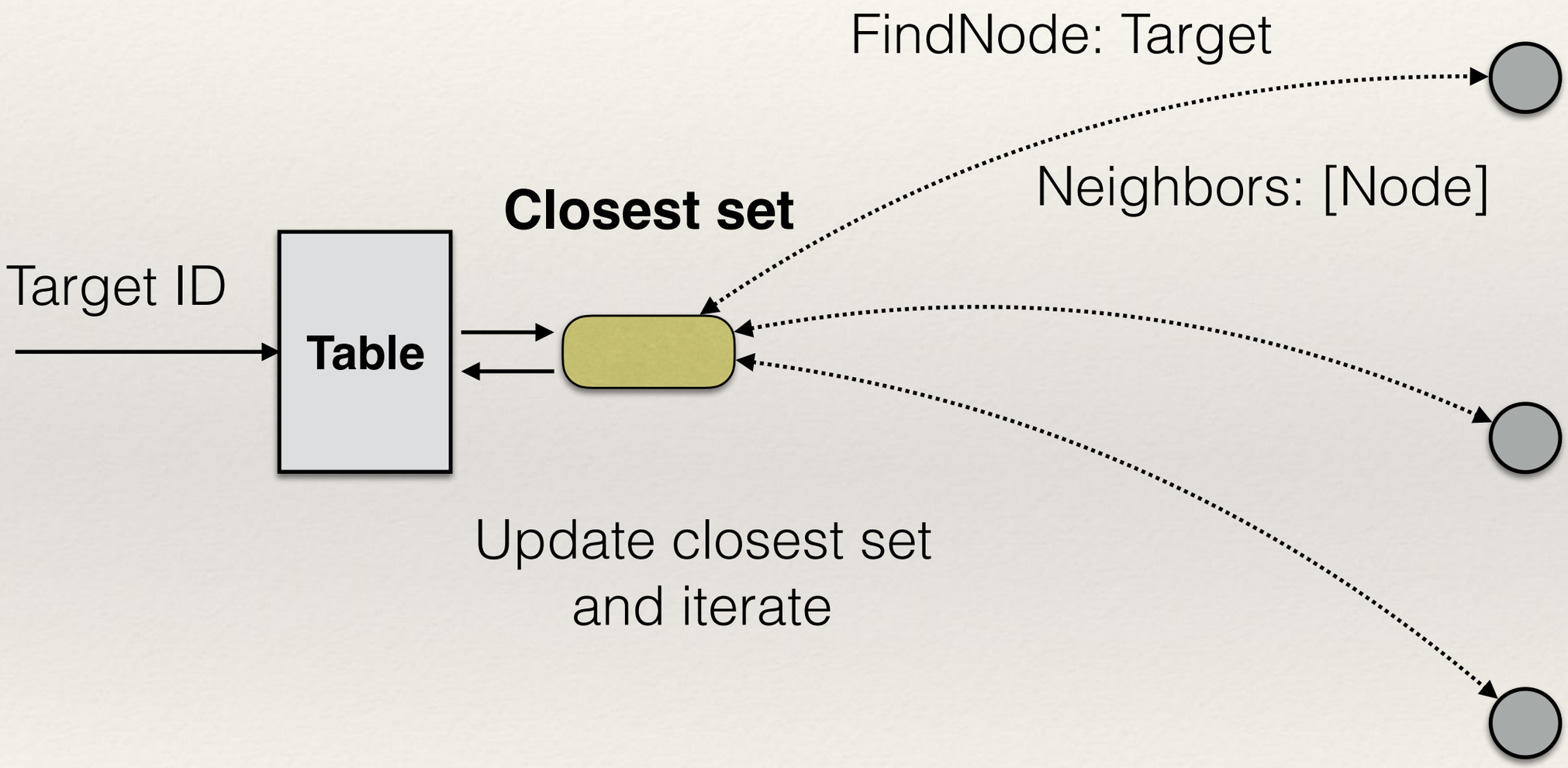
3. 节点发现协议 - K桶更新（2）



3. 节点发现协议 - 三个核心操作

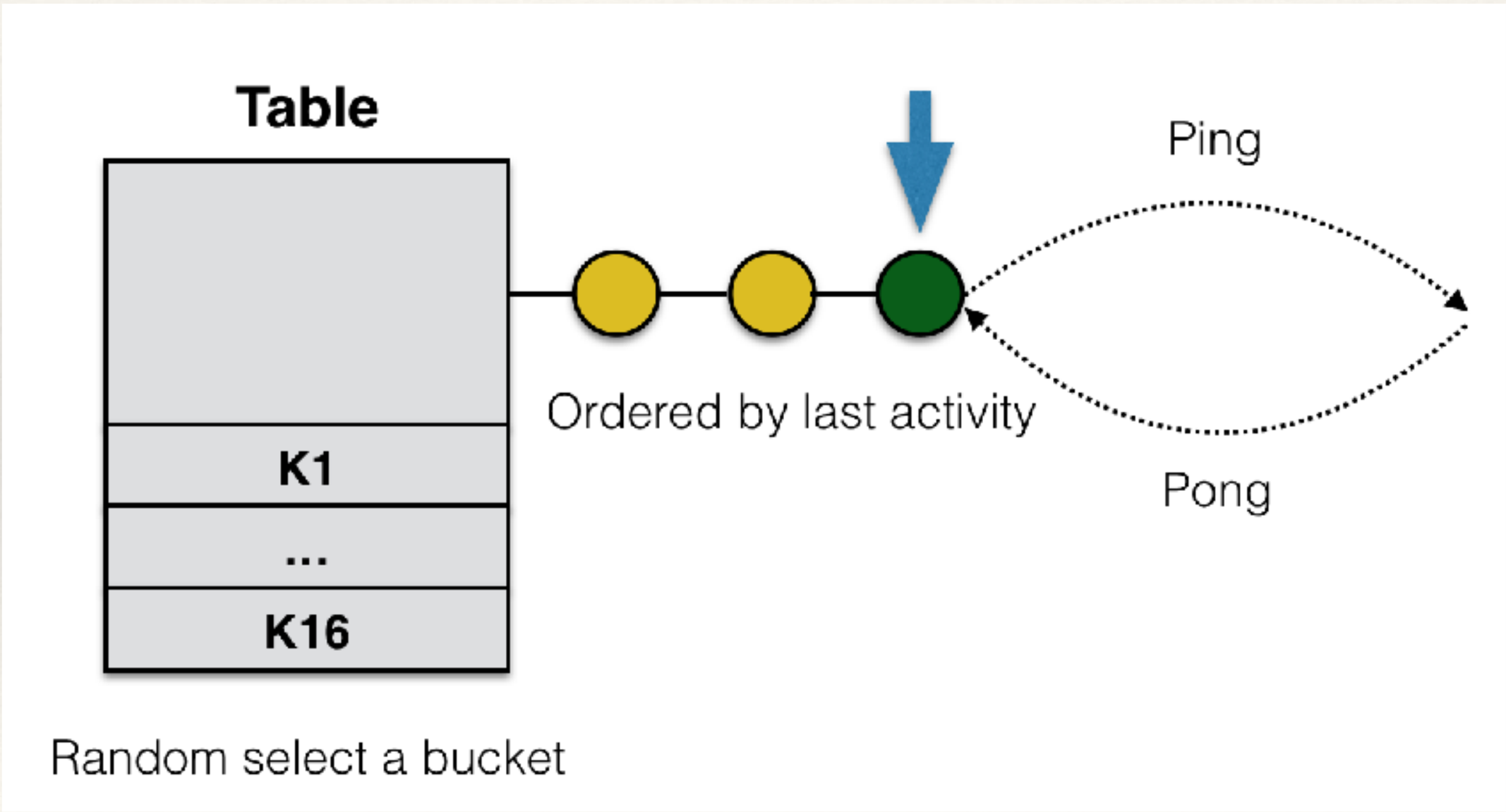
- Refresh: 选择一个随机的目标节点，对网络进行查询，使得本地的table尽可能多保存网络节点的信息
  - 一次对自身的查找（广播自身的网络信息）
  - 三次对随机目标节点的查找（填充“路由表”）
- Revalidate: 对本地table中的节点进行活性检查
- CopyLiveNode: 把具有活性的且连接时长较长的节点进行持久化

3. 节点发现协议 - Refresh

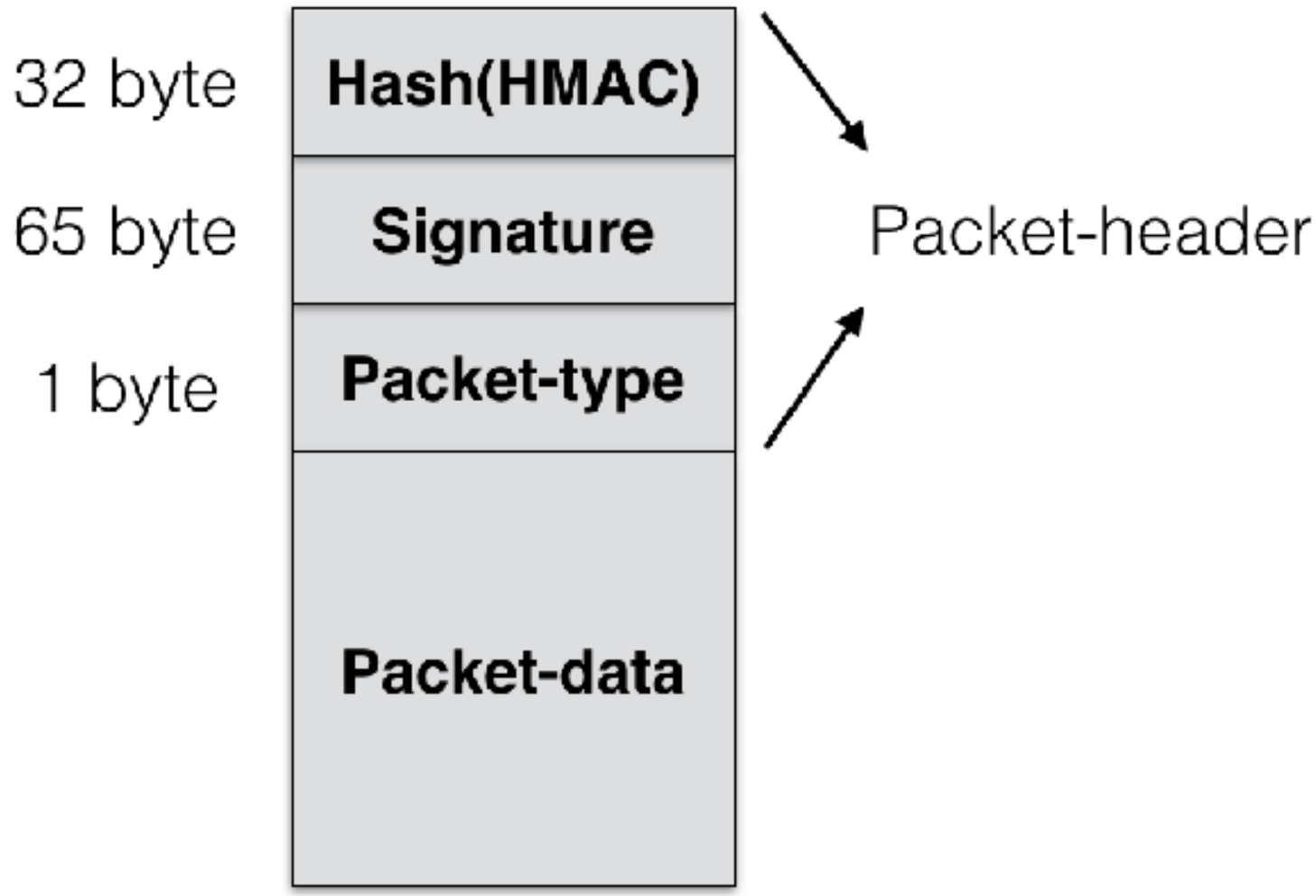


Fill table with all replies

3. 节点发现协议 - Revalidate



3. 节点发现协议 - 网络包



hash = keccak256(signature || packet-type || packet-data)

signature = sign(packet-type || packet-data)

4. 常见的discovery协议攻击

4. 常见的p2p网络攻击

- Eclipse攻击（月食攻击）
- Amplification攻击（放大攻击）

4. 月食攻击

- 选定一个目标攻击者，使得该攻击者所连接的节点都被某一个“恶意攻击者”所控制

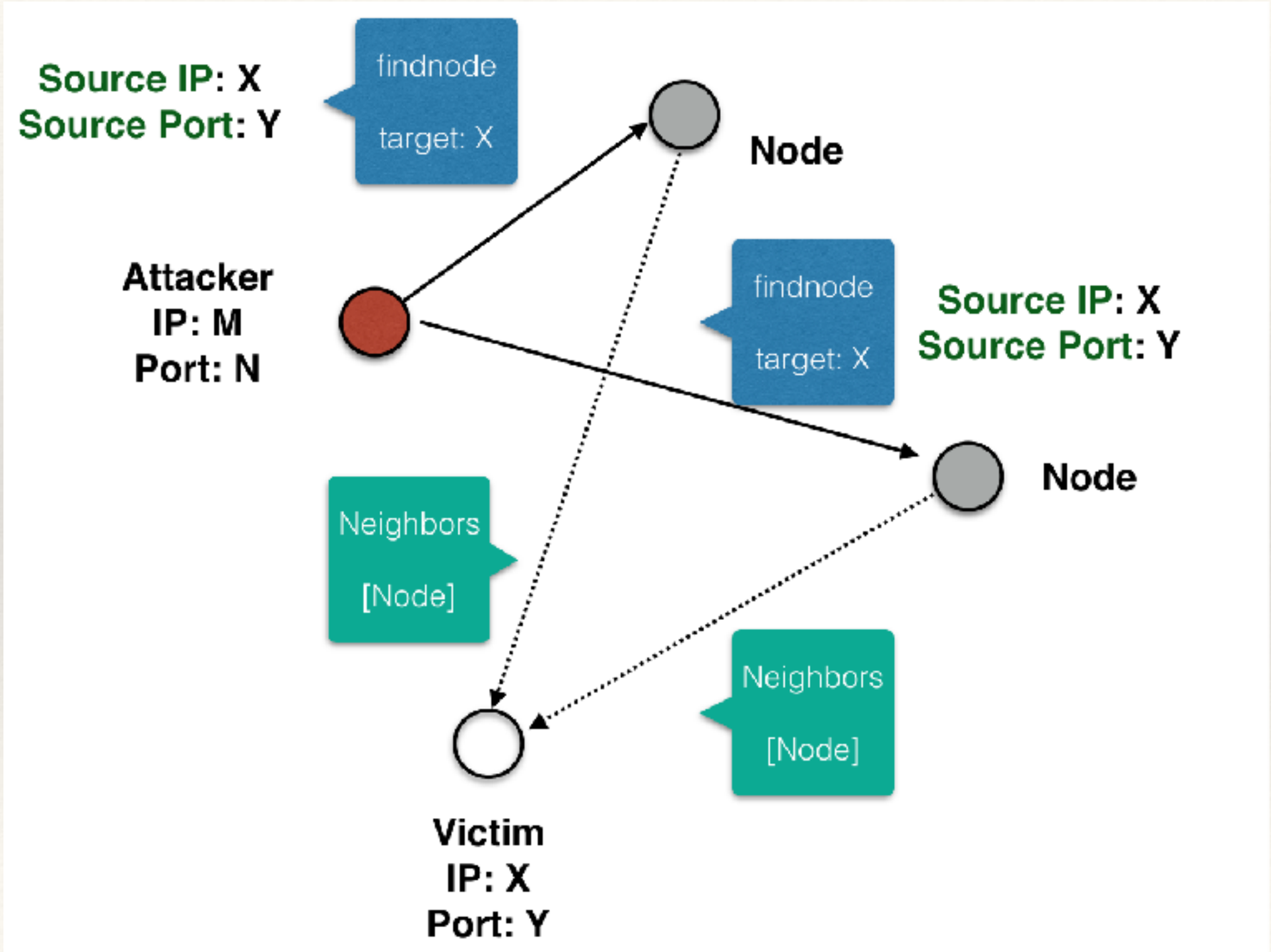
4. 限制incoming连接数

- Geth节点最多允许一半的连接来源自“外部主动连接”
- 每个Geth节点都会通过discovery去寻找到另外一半节点
- 确保节点的连接节点中，最大概率能有至少一个诚实节点

4. 放大攻击

- 攻击者向网络发送N个byte的数据，那么“被攻击者”将会接收到来自网络的M*N个byte的数据

4. 放大攻击


Source IP: X
Source Port: Y
findnode
target: X
Node
Attacker
IP: M
Port: N
findnode
target: X
Source IP: X
Source Port: Y
Node
Neighbors
[Node]
Neighbors
[Node]
Victim
IP: X
Port: Y

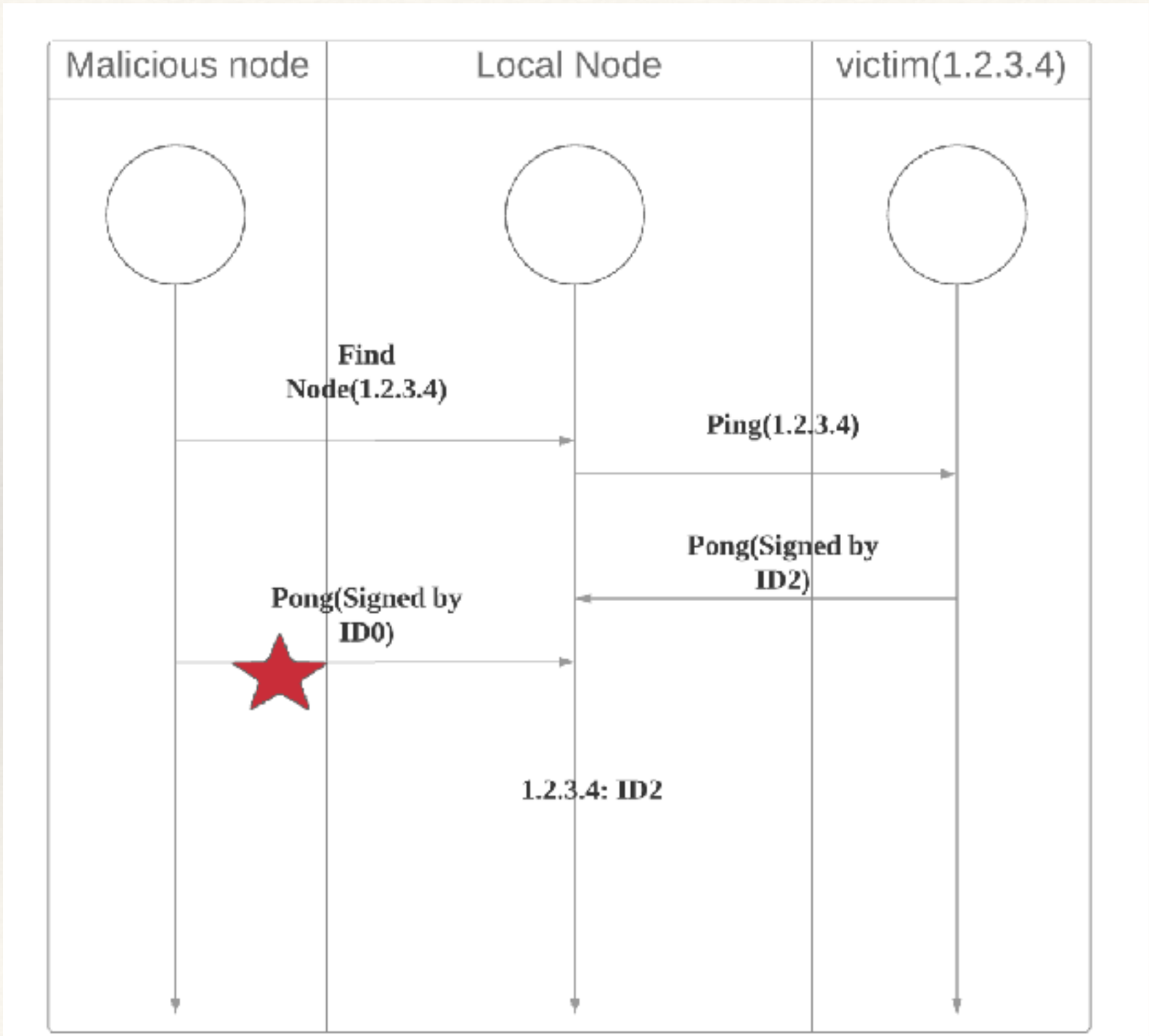
4. 节点发现协议 - BondCheck

- BondCheck是DiscV4对Kad协议的一个扩展
- Kad协议中存在一个DDoS的攻击方式：恶意攻击者将FindNode请求的来源地址改为“被攻击对象”的IP，端口，然后将该请求大量广播给不同的节点
- 收到请求的节点会把Neighbors节点发还给“被攻击对象”
- Neighbors的网络包大小远远大于FindNode的大小
- “被攻击对象”被Neighbors网络包所DDoS

4. 节点发现协议 - BondCheck

- 为了防止上述攻击方式，节点在处理FindNode请求之前必须确认请求者的endpoint必须是合理的
  - => We need endpoint proof!!
- 检查方式“本地节点”向“远端节点”发送一个Ping，如果能收到来自“远端节点”的合法的Pong，则视为改“远端节点的”endpoint是合法的。
- 换言之，我们需要绑定远端节点的公钥信息与网络信息
- endpoint的合法有效期为12小时

4. 节点发现协议 - BondCheck

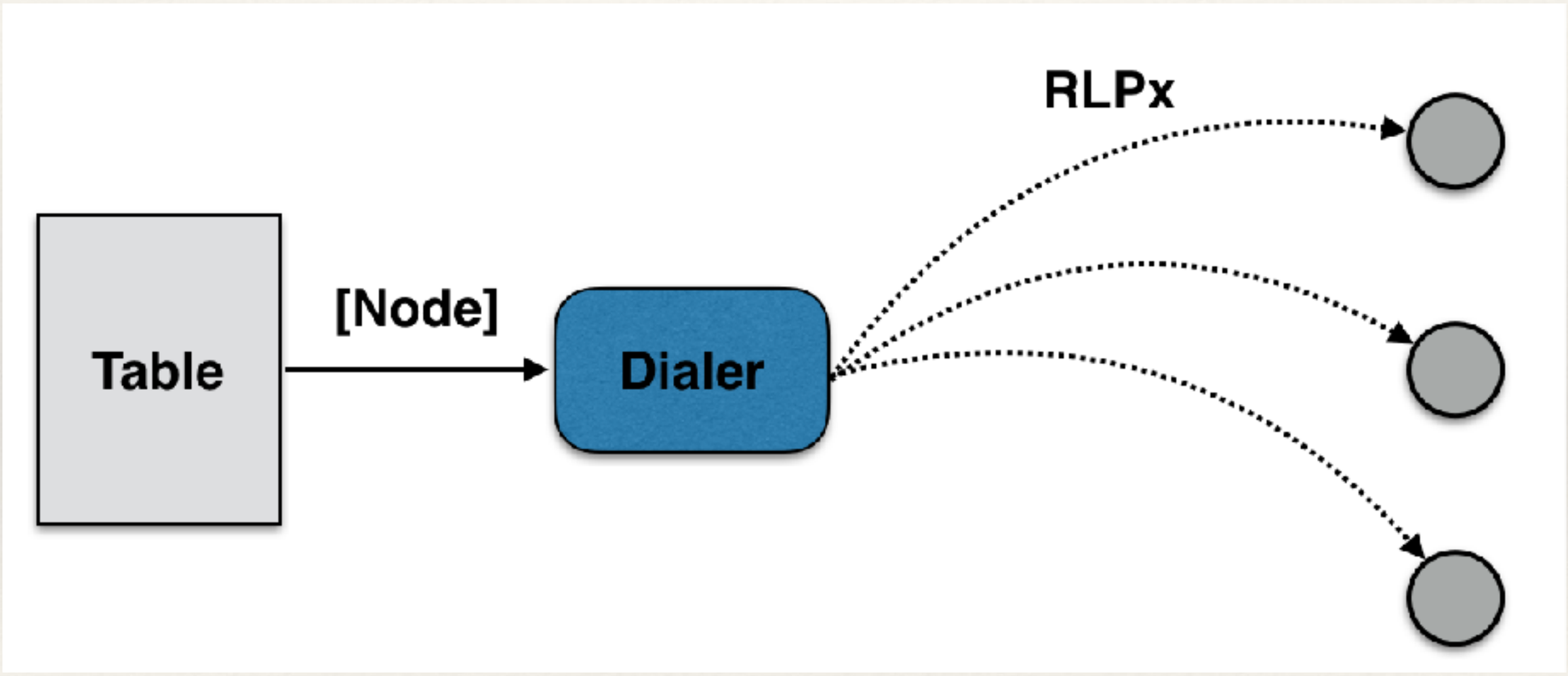


5. 传输层协议RLPx

5. 传输层协议RLPx

- 基于TCP的传输层协议，用于节点之间的通信
- 通信信道是基于对称加密算法进行加密的
- 对称秘钥是根据ECDHE算法，利用通信双方的公私钥协商得出
- RLPx协议的命名来源于其采用了RLP(Recursize Length Prefix)算法来进行网络包序列化、反序列化

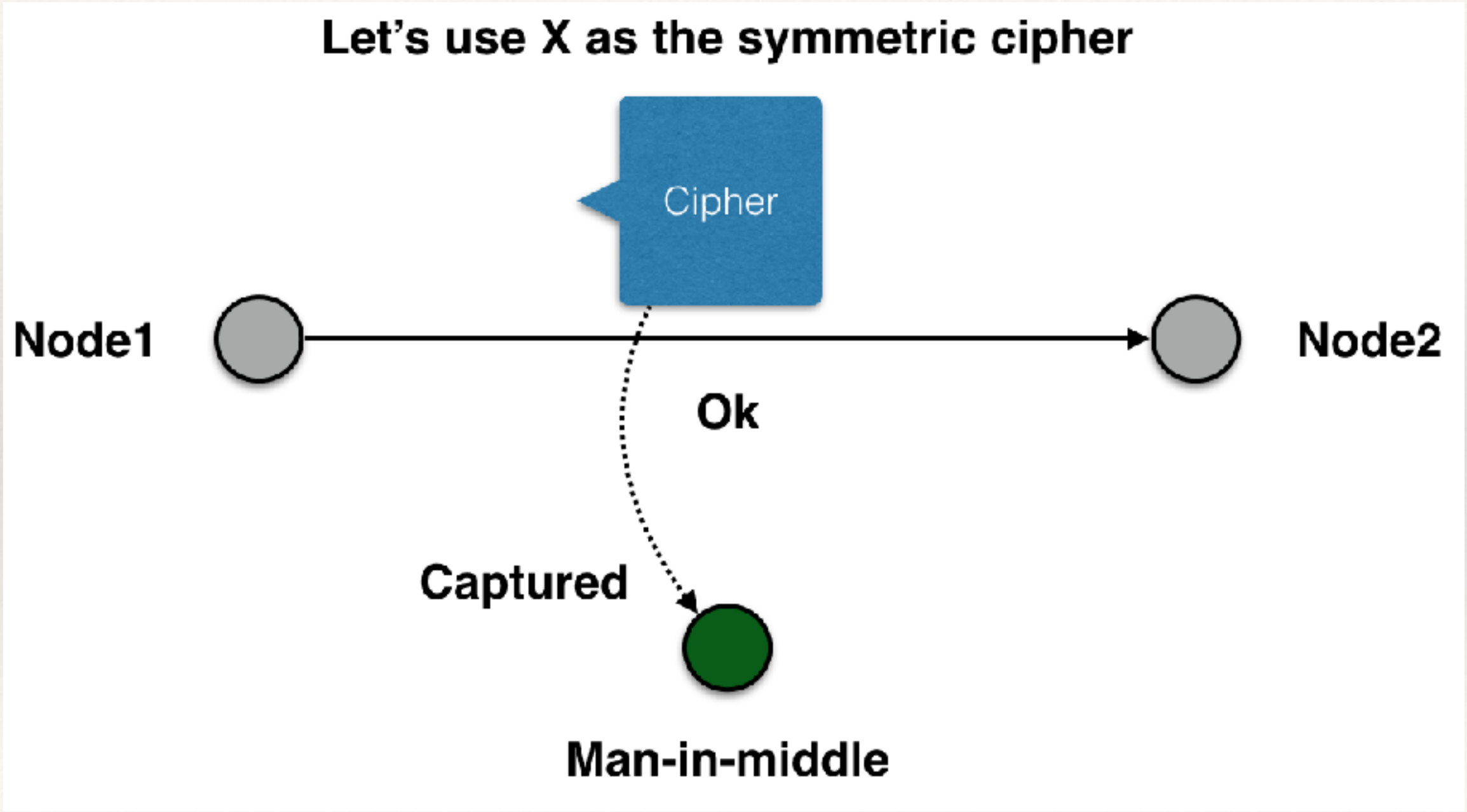
5. 传输层协议RLPx



5. 传输层协议RLPx

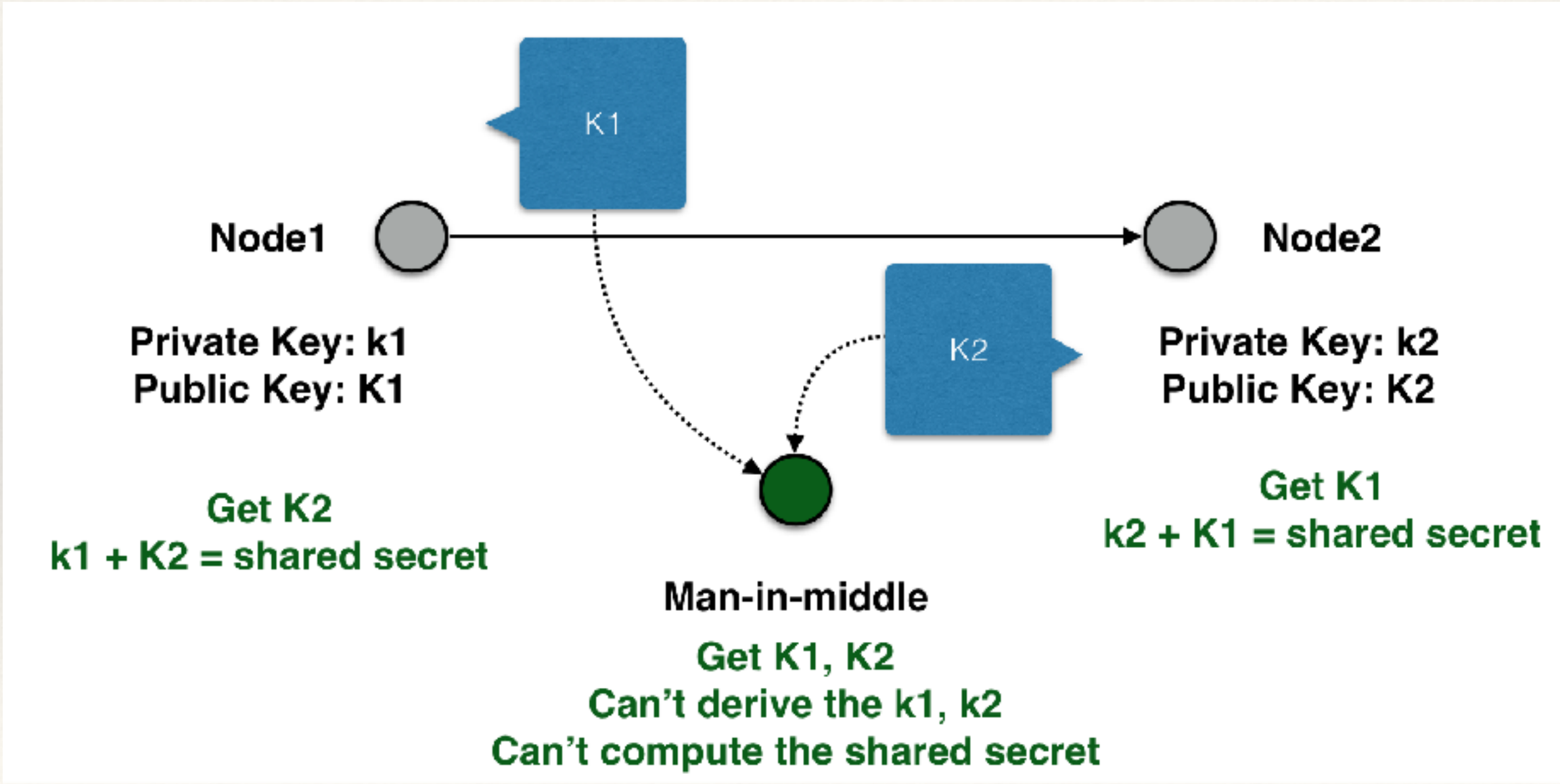
- RLPx除了建立底层的TCP连接以外，还有两次额外的握手操作
- 第一次握手为EncHandshake，通过这次握手双方协商出一个用于消息加密的共享秘钥
- 第二次握手为ProtoHandshake，通过这次握手双方协商出兼容的“应用层协议”以及对应的“协议版本”，若双方无兼容的应用层协议，则断开连接

5.共享秘钥协商（对称加密算法）



5. 共享秘钥协商（ECDH）

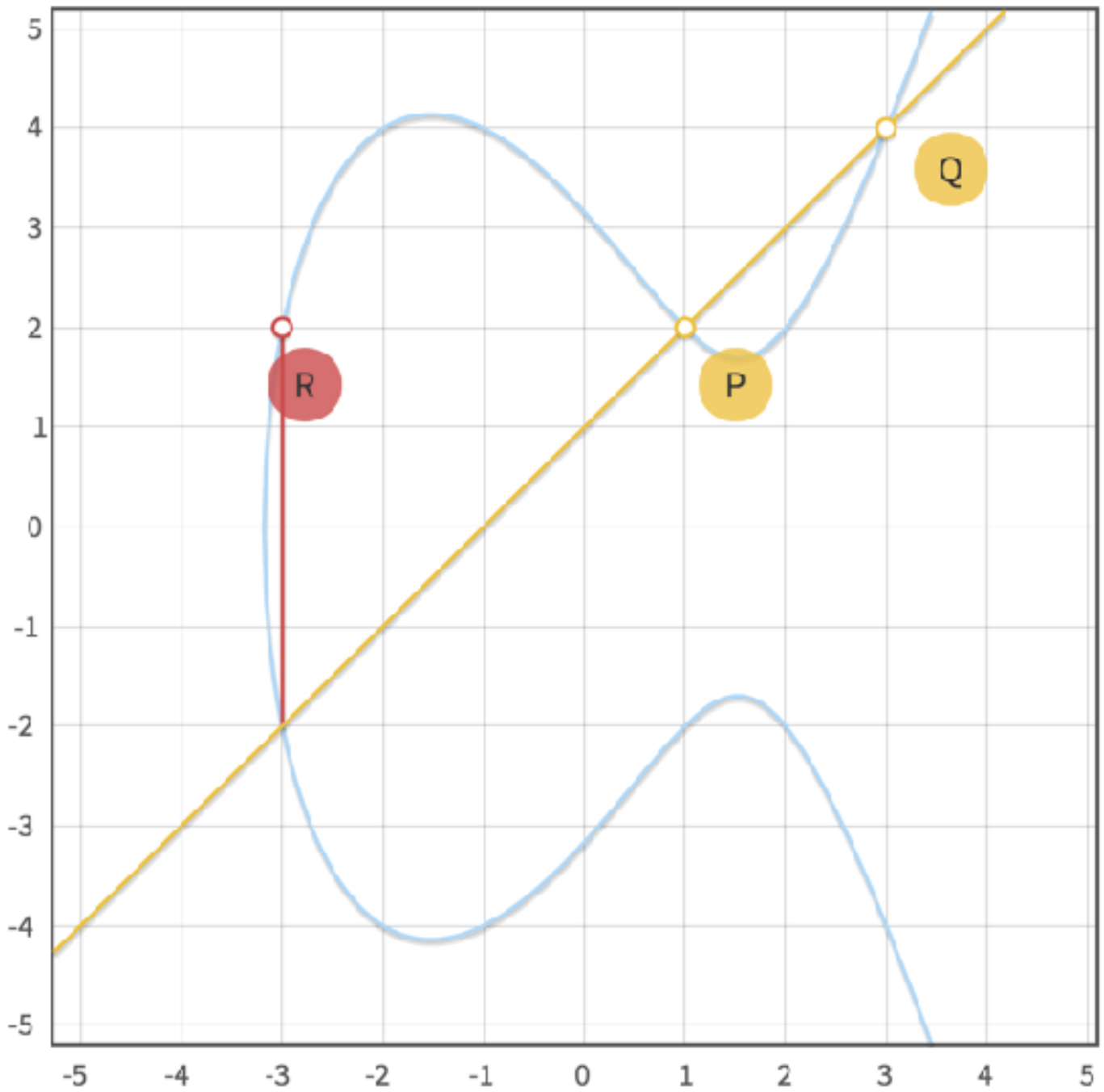
Elliptic-curve Diffie–Hellman



5. 共享秘钥协商（ECDH）

- ECDH是建立在椭圆曲线密码学上的一种具体算法
- 网络的每一个参与者都有一对公私钥
- 私钥是一个大整数k
- 公钥是在该椭圆曲线上基于G点进行k次标量乘法后得到的椭圆曲线上的另一点P，改点可以在二维空间内表示为(X, Y)。
- 椭圆曲线背后的安全性保证为：基于G点进行标量乘法很简单，但是基于P去计算k，不能够在多项式时间内完成
- 离散对数问题是椭圆曲线加密系统的关键

5. 实数域上椭圆曲线上的点加运算



Curve: a -7 b 10

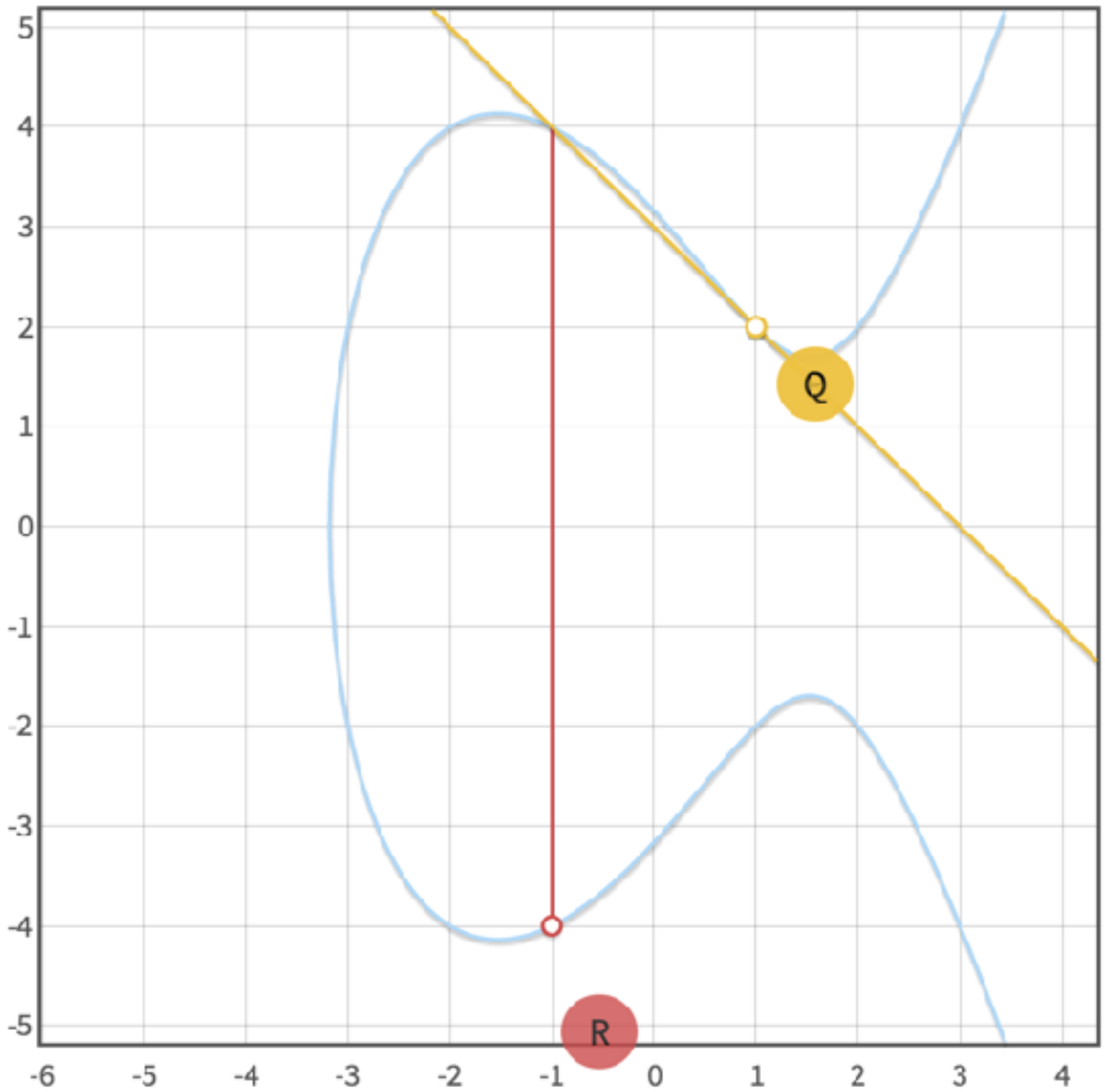
P: x 1 y 2

Q: x 3 y 4

R = *P* + *Q*: x -3 y 2

Point addition over the elliptic curve $y^2 = x^3 - 7x + 10$ in $\mathbb{R}$.

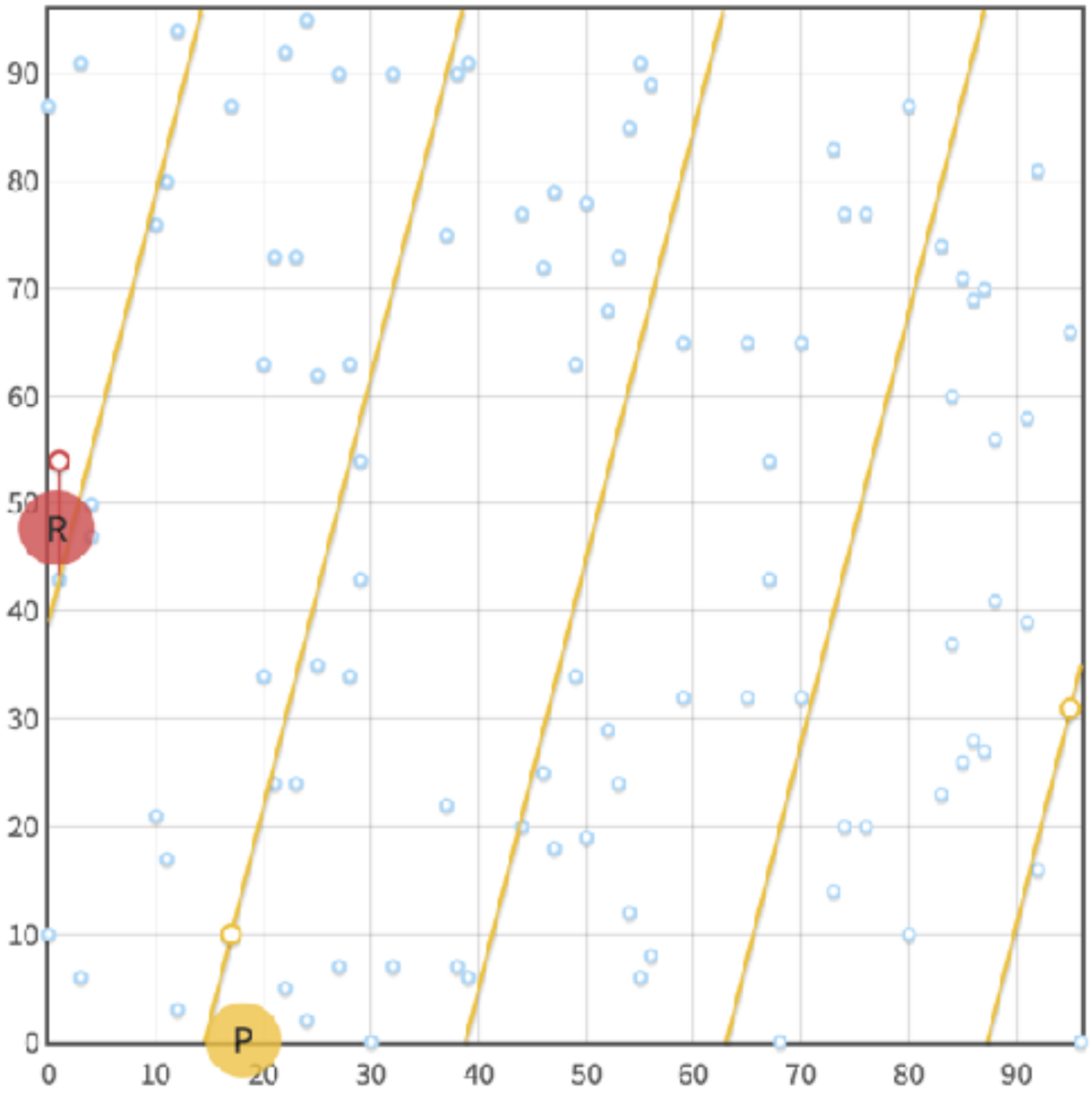
5. 实数域上椭圆曲线上的点加运算



Curve:	a	-7	b	10
P:	x	1	y	2
Q:	x	1	y	2
$R = P + Q$:	x	-1	y	-4

Point addition over the elliptic curve $y^2 = x^3 - 7x + 10$ in $\mathbb{R}$.

5. 有限域上椭圆曲线上的点加运算



Curve: a 2 b 3

Field: p 97

P: x 17 y 10

Q: x 95 y 31

$R = P + Q$: x 1 y 54

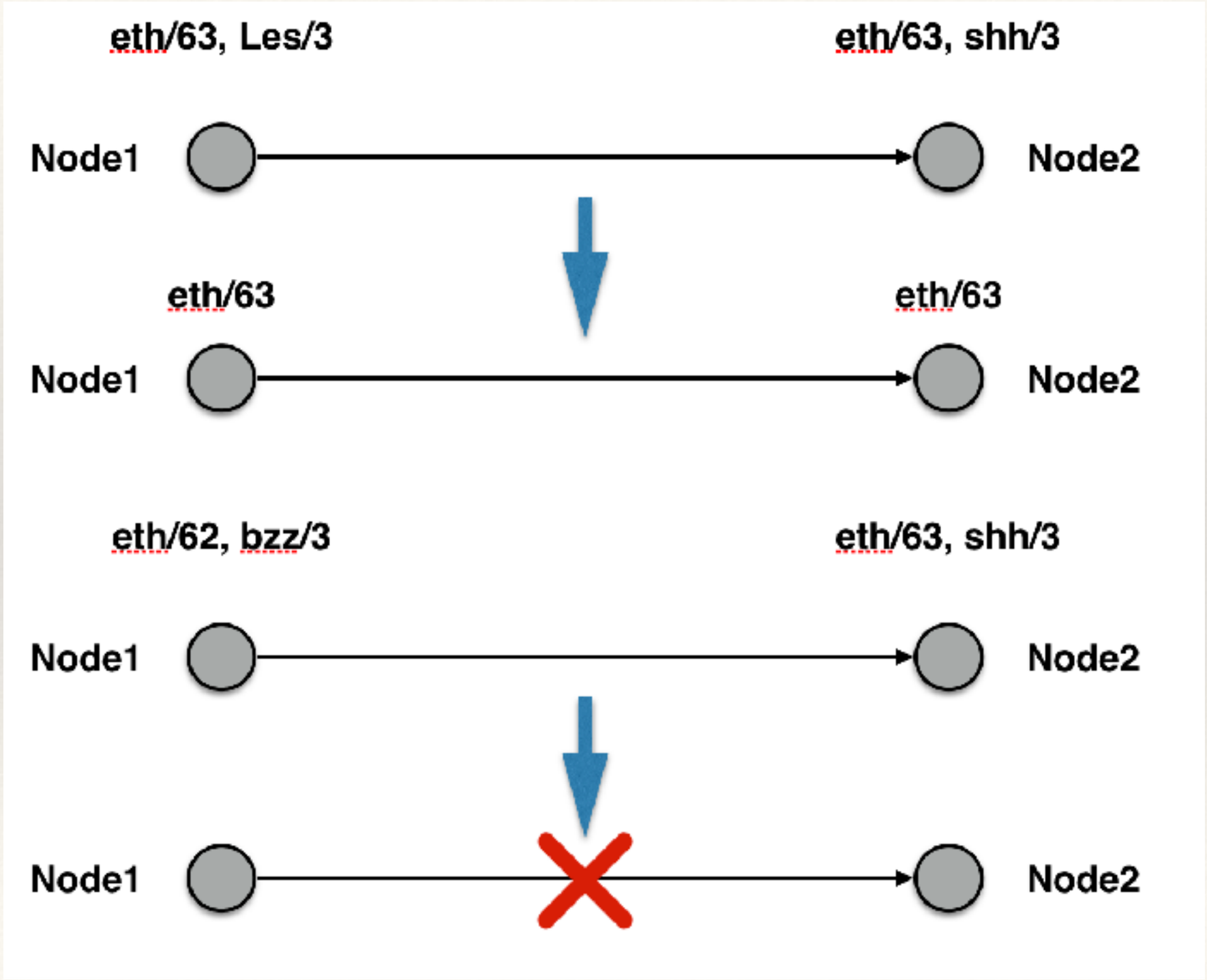
Point **addition** over the elliptic curve $y^2 = x^3 + 2x + 3$ in F_{97}.

The curve has 100 points (including the point at infinity).

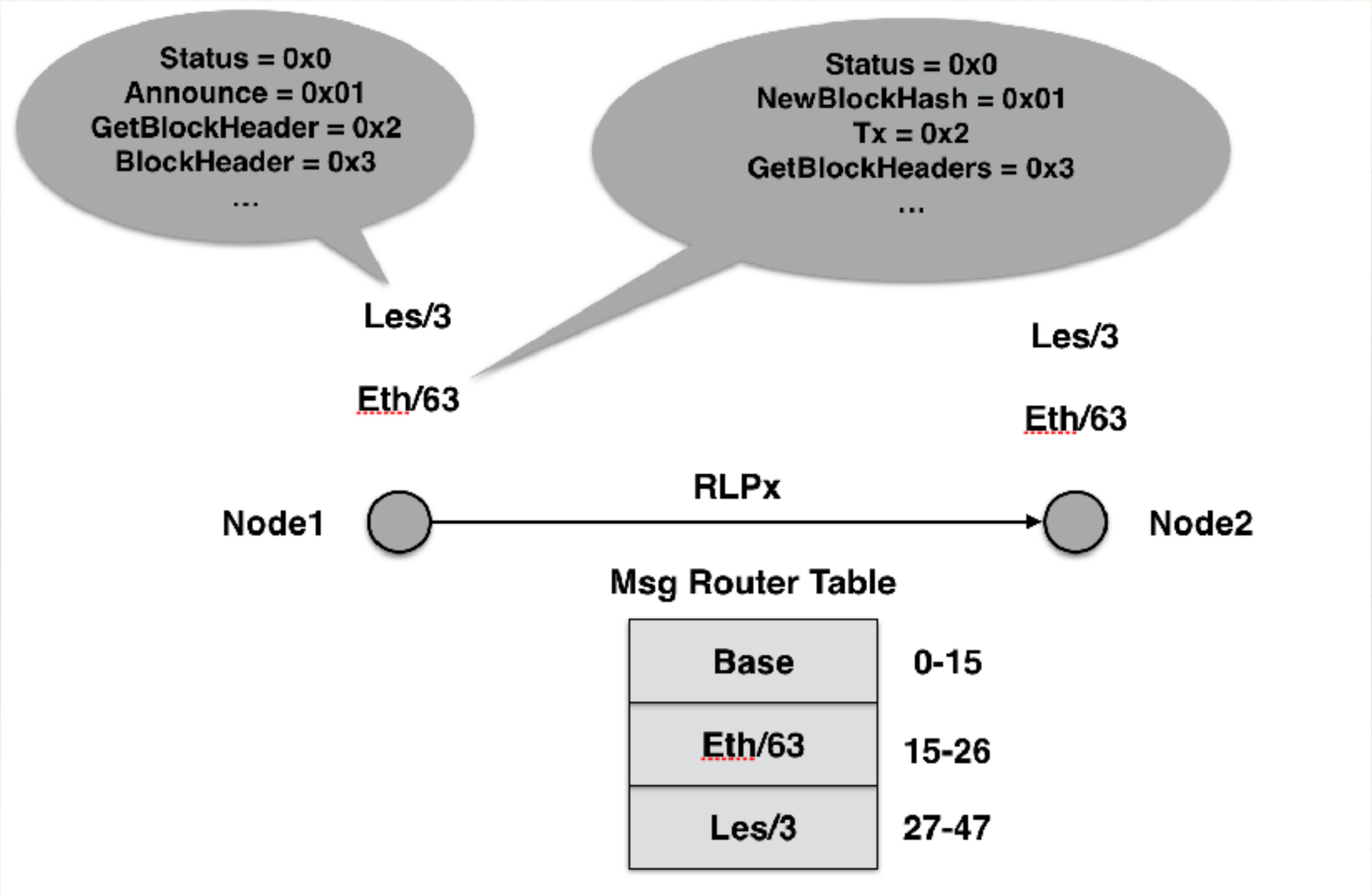
5. 协议握手

- 在RLPx至上，节点可以运行不同的应用层协议，例如Ethereum（ETH），Light Ethereum SubProtocol（LES），Swarm（bzz），Whipser（shh）
- 每个应用层协议都有对应的版本信息，例如Eth/62, Les/3
- 倘若通信双方的应用层协议无法兼容，则无法完成应用层的信息交换

5. 协议握手


eth/63, Les/3
eth/63, shh/3
Node1
Node2
eth/63
eth/63
Node1
Node2
eth/62, bzz/3
eth/63, shh/3
Node1
Node2
Node1
Node2

5. 消息路由


Status = 0x0
Announce = 0x01
GetBlockHeader = 0x2
BlockHeader = 0x3
...
Les/3
Eth/63
Status = 0x0
NewBlockHash = 0x01
Tx = 0x2
GetBlockHeaders = 0x3
...
Les/3
Eth/63
RLPx
Node1
Node2
Msg Router Table
Base
0-15
Eth/63
15-26
Les/3
27-47

扩展阅读

- Devp2p: https://github.com/ethereum/devp2p
- Kademlia: https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf
- 椭圆曲线密码学介绍：https://zhuanlan.zhihu.com/p/55758642

Thanks