p2p 网络 模块

文档说明

作者	liangdu
联系邮箱	liangdu@nscc-gz.cn或liangdu1992@gmail.com
发布日期	2020-07-09
版权所有	nscc-gz@copyright
备注	无

libp2p库说明与使用

libp2p 简介

背景知识:

IPFS:

“IPFS is a distributed system for storing and accessing files, websites, applications, and data.”

ipfs 具有特性如下:

- 去中心化
- 支持弹性网络
- 更难以进行文件内容审查
- 加速网络内容获取

ipfs构成核心:

- Content-address 内容标记
- 分布式hash表 DHT
- Merkle DAGS
- git
- bitswap
- IPLD
- IPNS
- Libp2p 网络层
- Self-certifying File system

参考:

1. https://docs.ipfs.io/
2. https://ipfs.io/
3. https://zh.wikipedia.org/wiki/%E6%98%9F%E9%99%85%E6%96%87%E4%BB%B6%E7%B3%BB%E7%BB%9F
4. https://github.com/ipfs/ipfs
5. https://en.wikipedia.org/wiki/Self-certifying_File_System

P2P网络:

"A peer-to-peer (p2p) network is one in which the participants (referred to as peers or nodes) on more or less “equal footing”.
This does not necessarily mean that all peers are identical; some may have different roles in
However, one of the defining characteristics of a peer-to-peer network is that they do not rec as is the case in the the predominant client / server model."

p2p网络特性:

- 去中心化
- 扩展性强
- 隐私性
- ...等

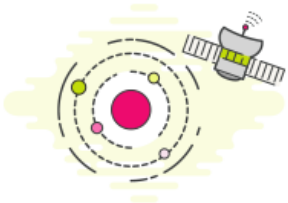
libp2p最初作为IPFS项目中的网络层实现，用以解决IPFS项目中IPFS-Node节点的网络联通、数据交换等问题.后来从IPFS项目中分离出来作为独立的网络库，用以解决现代网络环境下构建P2P网络应用的需求。 **libp2p来源于IPFS项目，目前已经从ipfs项目中独立出来作为一个专注于P2P网络的网络库**

> If you’re doing anything in the decentralized, peer-to-peer space, you’ve probably heard of libp2p - a modular networking stack for peer-to-peer applications. libp2p consists of many modular libraries from which p2p network developers can select and reuse just the protocols they need, while making it easy to upgrade and interoperate between applications. This helps web3 developers get up and running faster, build more resilient decentralized applications, and utilize advanced features like decentralized publish-subscribe and a distributed hash table.What makes libp2p different from the networking stack of today is its focus on transport agnosticism, modularity, and portable encodings (like multiaddr). These properties make libp2p the networking layer of choice for most new dweb projects, blockchains, and peer-to-peer applications. Read more about why projects are choosing to build on libp2p, or watch the recent talk from Tech Lead Raul Kripalani at DevCon5.

libp2p 特性和适用场景:

- [x] Use Serval Transports 适用于多种协议,TCP/UDP/QUIC/WebRTC等
- [x] Native Roaming 自适应网络，网络发生变动时程序、服务不需要做额外配置
- [x] Runtime Freedom 运行时无关，运行平台/软件不影响网络
- [x] Protocol Muxing 协议复用.网络连接复用 如: stream multiplexing
- [x] Work Offline 可自行发现节点，不需要中心服务器或注册服务 如: mdns 节点发现
- [x] Encrypted Connections 连接加密，通信链路加密和节点加密认证 如: peer identity
- [x] Upgrade Without Compromises 无感升级
- [x] Work In the brower 可浏览器中运行
- [x] Good For High Latency Scenarios 可应用于高延迟场景

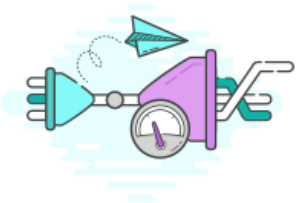
Features



Use Several Transports

Native Roaming

Runtime Freedom



Protocol Muxing

Work Offline

libp2p is capable of discovering other peers without resourcing to centralized registries, enabling apps to work disconnected from the backbone.

Encrypted Connections

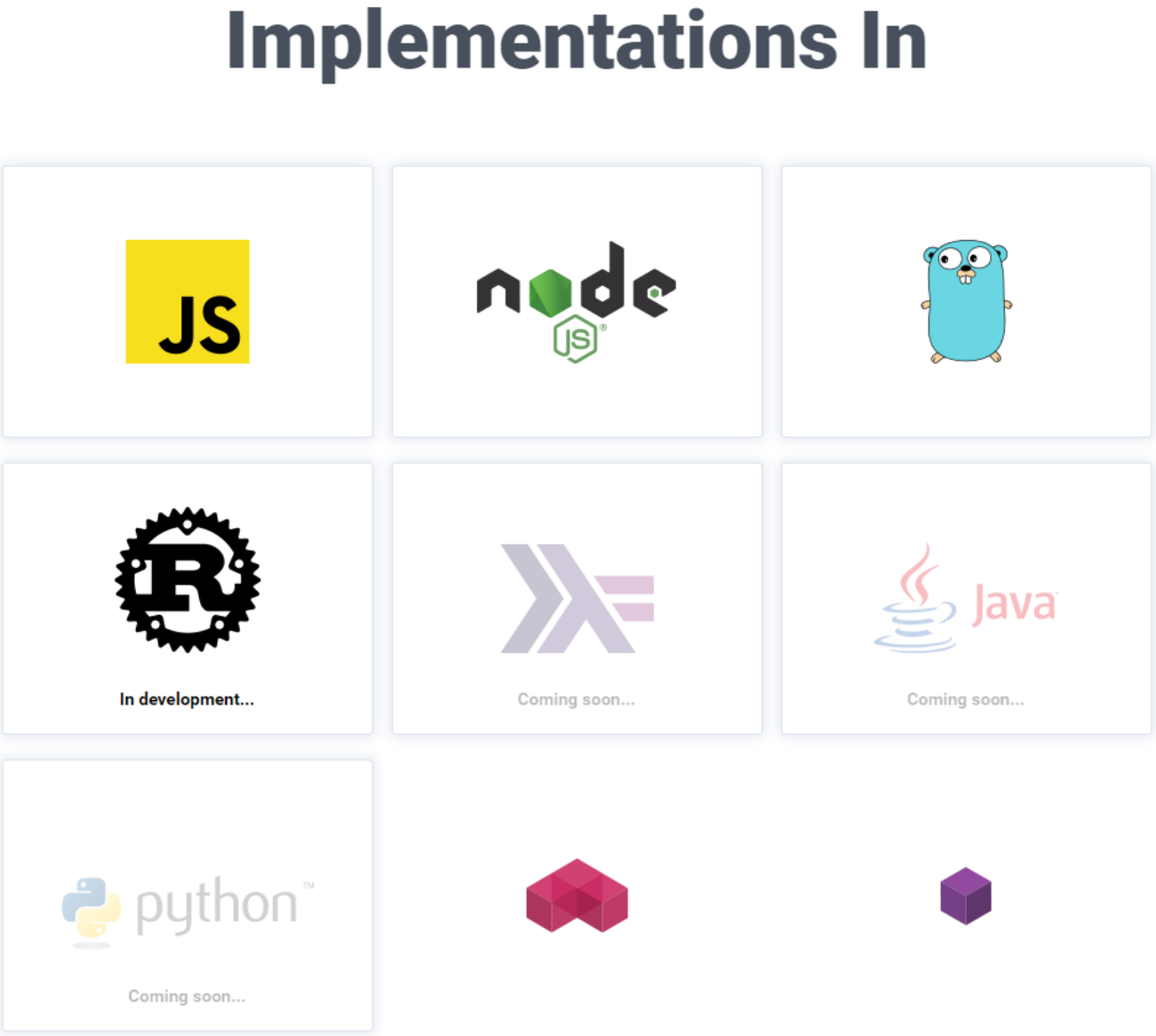
Upgrade Without Compromises

Work In The Browser

Good For High Latency Scenarios

libp2p 的实现版本

- [x] go-libp2p
- [x] js-libp2p
- [x] nodejs-libp2p
- [x] rust-libp2p



libp2p 中关键概念

该部分内容可参考:https://docs.libp2p.io/reference/glossary/

- Circuit Relay
  libp2p中可通过中继节点方式进行节点联通和数据通信,libp2p中中继模式的实现在"github.com/libp2p/go-libp2p-circuit",在通过中继节点连接目标节点时构造的目标点peer-address形式为:

  ```
  relayAddr, err := ma.NewMultiaddr(option.RelayID + "/p2p-circuit" + option.PeerID)
  ```
- DHT
  libp2p通过使用DHT来实现其peer routing方法并且通常使用DHT来存储和提供文件内容的元数据，用于内容发现和服务广播

- Connection
  libp2p中连接是在节点建立完成进行数据读写的通信通道,libp2p中的连接通常是host.Connect(ctx,targethost)方法进行创建的,底层的连接所使用的协议可以有多种,tcp/udp等
- Dial & Listen
  libp2p中用于建立和接收连接操作，尝试打开libp2p连接时叫dial ,监听和接收libp2p 连接时叫listening
- mDNS
  mDNS协议 multicast DNS , 使用5353端口，组播地址 224.0.0.251。在一个没有常规DNS服务器的小型网络内，可以使用mDNS来实现类似DNS的编程接口、包格式和操作语义。通常可用于局域网内节点发现，libp2p中使用mdns进行本地网络环境中的快速节点发现。
- multiaddr
  mutiaddr是libp2p中对节点地址的标识方法,通常形式为/ip4|6/ip/tcp|udp/port
  如"/ip4/127.0.0.1/udp/1234"
- Multihash
  Multihash是对多种hash算法组合的一种简要表示方法,在libp2p中使用mutihash的典型场景是节点id peer id,peerid 是包含生成该节点的加密算法的公钥的，此外另一个适用场景是CID (Content identifier)的生成。mutihash在libp2p中通常是做过base58加密的。在libp2p中mutihash的实现包为"github.com/multiformats/go-multihash"
- Multiplexing
  多路复用在libp2p中是指在单独的连接上适用多个数据流/网络流的通信方式,通过使用多路复用技术可以在一个连接上使用多个网络协议,从而降低网络连接成本。在libp2p中已经实现和支持的复用协议又yamux和spdy等。具体的libp2p 可以通过一个端口，如TCP或UDP端口，根据所使用的传输来执行其所有操作。libp2p 可以通过点到点连接来复用它的许多协议。这种复用是用于可靠的流和不可靠的数据报。
  libp2p 比较务实。它试图在尽可能多的配置中使用，以模块化和灵活的方式来适应各种用例，并尽可能少地选择。因此，libp2p 网络层提供了我们松散地称之为“多重多路复用”的内容:
  1. 多个网络接口的多路复用
  2. 多个传输协议的多路复用
  3. 多个对等连接的多路复用
  4. 可以复用多个客户端协议
  5. 每个协议/连接可以多路复用多个流（SPDY、HTTP2、QIC、SSH）
  6. 流量控制（背压，公平性）
  7. 用不同的临时密钥加密每个连接
- multistream
  libp2p使用mutistream来标识通信节点之间所使用的网络协议,也可以用来进行协议协商

> multistream is a lightweight convention for “tagging” streams of binary data with a short header that identifies the content of the stream.

> libp2p uses multistream to identify the protocols used for communication between peers, and a related project multistream-select is used for protocol negotiation.

- NAT & NAT Traversal
  NAT是在计算机网络中是一种在IP数据包通过路由器或防火墙时重写来源IP地址或目的IP地址的技术。这种技术被普遍使用在有多台主机但只通过一个公有IP地址访问互联网的私有网络中。它是一个方便且得到了广泛应用的技术。当然，NAT也让主机之间的通信变得复杂，导致了通信效率的降低
  NAT转换时内网向外网转换相对容易，外网向内转换则相对困难。在Client-server模式下服务端通常情况下具有足够的信息来完成外网向内网的NAT转换，但是在P2P网络模型下则相对困难。用于NAT转换的方法通常有根据端口转换,libp2p中适用NAT转换的包在"https://github.com/libp2p/go-libp2p-nat"中
- Peer
  p2p网络中的参与节点,给定的节点可能会支持多种协议,并且节点都会有独立的节点ID,通过该ID来在网络中标识自己。
- PeerId
  libp2p中节点的标识,且标识形式为mutihash形式,通常以节点公私钥加密算法形式进行生成，并且可以结合DHT进行节点身份的验证,通过此种方式可以将节点标识与节点所使用的底层协议分离。
- Peer store
  用于存储已知peer 节点peer-id的数据结构，结合已知的mutiaddress 可以用来建立与特定peer的连接
- Peer routing
  peer routing 是用来发现网络中节点peer的(routing)节点路由或节点地址的进程,也可以用来做本地节点的临近发现，比如通过 muticatst DNS方式。libp2p中的主要路由机制是通过实现了kademlia路由算法的分布式hash表dht来定位节点
- Peer-to-peer (p2p)
  p2p网络中网络参与节点可以直接与目标节点进行直接通信,但这种通信模式并不代表网络中的所有节点都是完全对等的。在p2p网络中的某些节点仍可能有不同的角色。
- Pubsub
  发布订阅模式在libp2p中的实现,具体实现可参考"github.com/libp2p/go-libp2p-pubsub",标准模式说明可参考:
  https://docs.libp2p.io/concepts/publish-subscribe/
- Protocol
  libp2p自带多种网络协议而且也可以使用操作系统或运行环境的有的网络协议,在libp2p中的协议通常是使用multistream头信息进行标注。
- Protocol Negotiation
  libp2p中的协议协商过程首先由包含头信息的mutistream进行协议选择,mutistream的头信息包含唯一协议名称和版本号，当节点建立连接后进行协议握手并选择所要使用的网络协议，这个过程叫做mutistream-select

- Stream
  libp2p中的流通常是指p2p节点之间所建立的网络流,节点之间的数据传输也是通过libp2p stream进行传输。对应的是数据结构为network network.Stream
- Switch
  switch是libp2p中将多个传输协议封装到一个接口的模块,通过switch模块可以使得应用在不指定特定传输协议的前提下与节点通信。除此之外switch模块还可以用来进行协议升级，比如将原始的传输层协议升级成应用支持的协议,并进行流多路复用和安全通信。switch最初在ipfs项目中时叫swarm

  > Swarm
  > Can refer to a collection of interconnected peers.
  > In the libp2p codebase, “swarm” may refer to a module that allows a peer to interact with its peers, although this component was later renamed “switch”.
  > See the discussion about the name change for context.

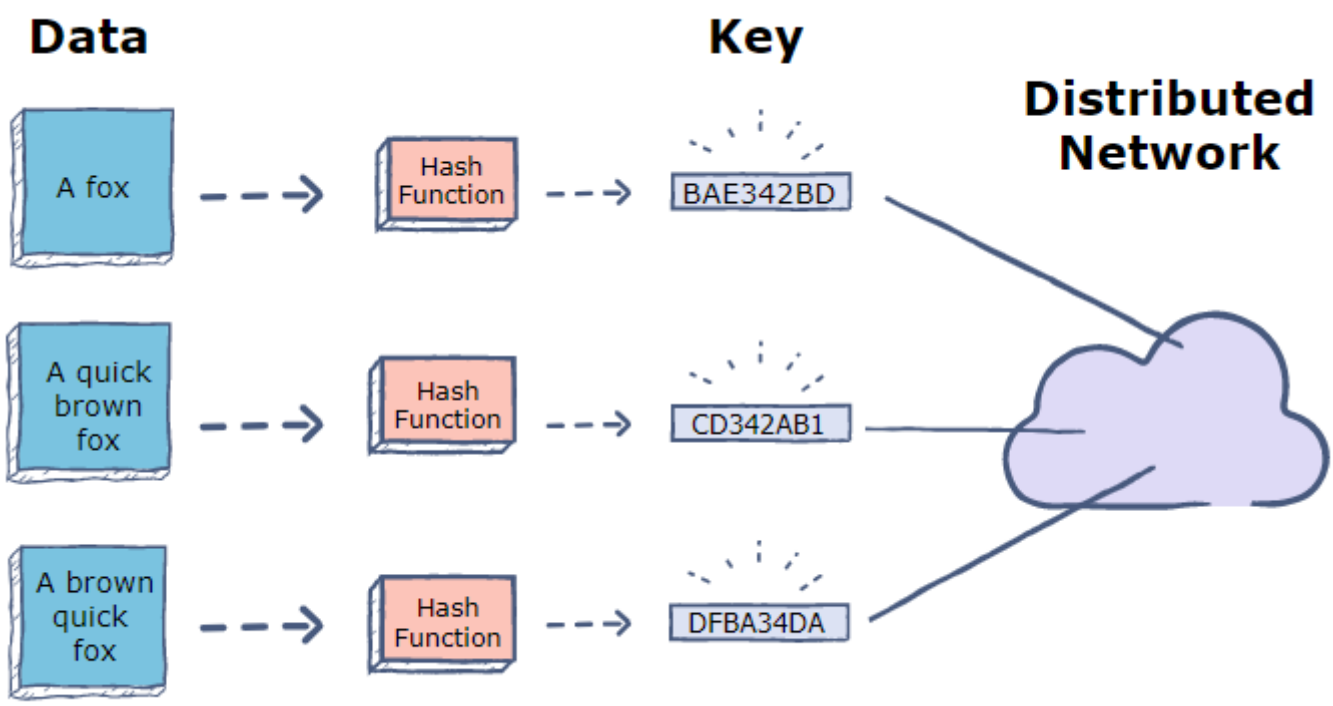
libp2p 中关键数据结构

- [x] 分布式hash表(Distributed Hash Tables|dht)
  libp2p中分布式hash表主要用于节点路由(peer routing)和内容发现(content routing)
  分布式散列表用来将一个关键值（key）的集合分散到所有在分布式系统中的节点，并且可以有效地将消息转送到唯一一个拥有查询者提供的关键值的节点（Peers）。这里的节点类似散列表中的存储位置。分布式散列表通常是为了拥有极大节点数量的系统，而且在系统的节点常常会加入或离开（例如网络断线）而设计的。在一个结构性的延展网络（overlay network）中，参加的节点需要与系统中一小部分的节点沟通，这也需要使用分布式散列表。分布式散列表可以用以创建更复杂的服务，例如分布式文件系统、点对点技术文件分享系统、合作的网页缓存、多播、任播、域名系统以及即时通信等。
  分布式散列表本质上强调以下特性：
  1.离散性：构成系统的节点并没有任何中央式的协调机制。
  2.伸缩性：即使有成千上万个节点，系统仍然应该十分有效率。
  3.容错性：即使节点不断地加入、离开或是停止工作，系统仍然必须达到一定的可靠度。

  > A DHT gives you a dictionary-like interface, but the nodes are distributed across the network.
  > The trick with DHTs is that the node that gets to store a particular key is found by hashing that key, so in effect your hash-table buckets are now independent nodes in a network.
  > This gives a lot of fault-tolerance and reliability, and possibly some performance benefit, but it also throws up a lot of headaches. For example, what happens when a node leaves the network, by failing or otherwise? And how do you redistribute keys when a node joins so that the load is roughly balanced. Come to think of it, how do you evenly distribute keys anyhow? And when a node joins, how do you avoid rehashing everything? (Remember you'd have to do this in a normal hash table if you increase the number of buckets).
  > One example DHT that tackles some of these problems is a logical ring of n nodes, each

> taking responsibility for 1/n of the keyspace. Once you add a node to the network, it finds a place on the ring to sit between two other nodes, and takes responsibility for some of the keys in its sibling nodes. The beauty of this approach is that none of the other nodes in the ring are affected; only the two sibling nodes have to redistribute keys.
> For example, say in a three node ring the first node has keys 0-10, the second 11-20 and the third 21-30. If a fourth node comes along and inserts itself between nodes 3 and 0 (remember, they're in a ring), it can take responsibility for say half of 3's keyspace, so now it deals with 26-30 and node 3 deals with 21-25.
> There are many other overlay structures such as this that use content-based routing to find the right node on which to store a key. Locating a key in a ring requires searching round the ring one node at a time (unless you keep a local look-up table, problematic in a DHT of thousands of nodes), which is O(n)-hop routing. Other structures - including augmented rings - guarantee O(log n)-hop routing, and some claim to O(1)-hop routing at the cost of more maintenance.

.



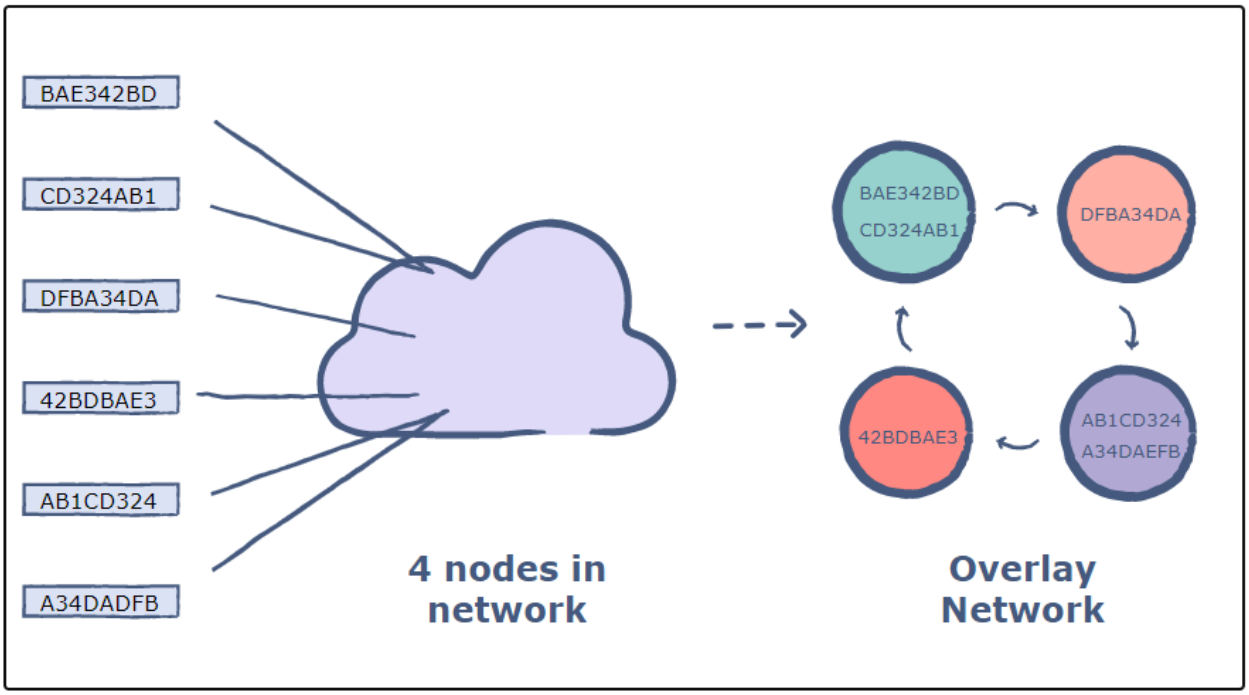
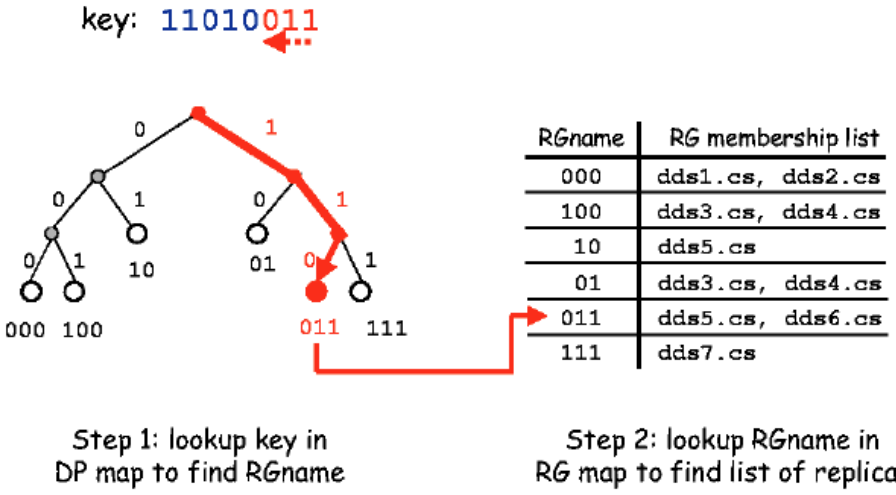


Figure: Distributed hash table metadata maps: this illustration highlights the steps taken to discover the set of replica groups which serve as the backing store for a specific hash table key. The key is used to traverse the DP map trie and retrieve the name of the key's replica group. The replica group name is then used looked up in the RG map to find the group's current membership.

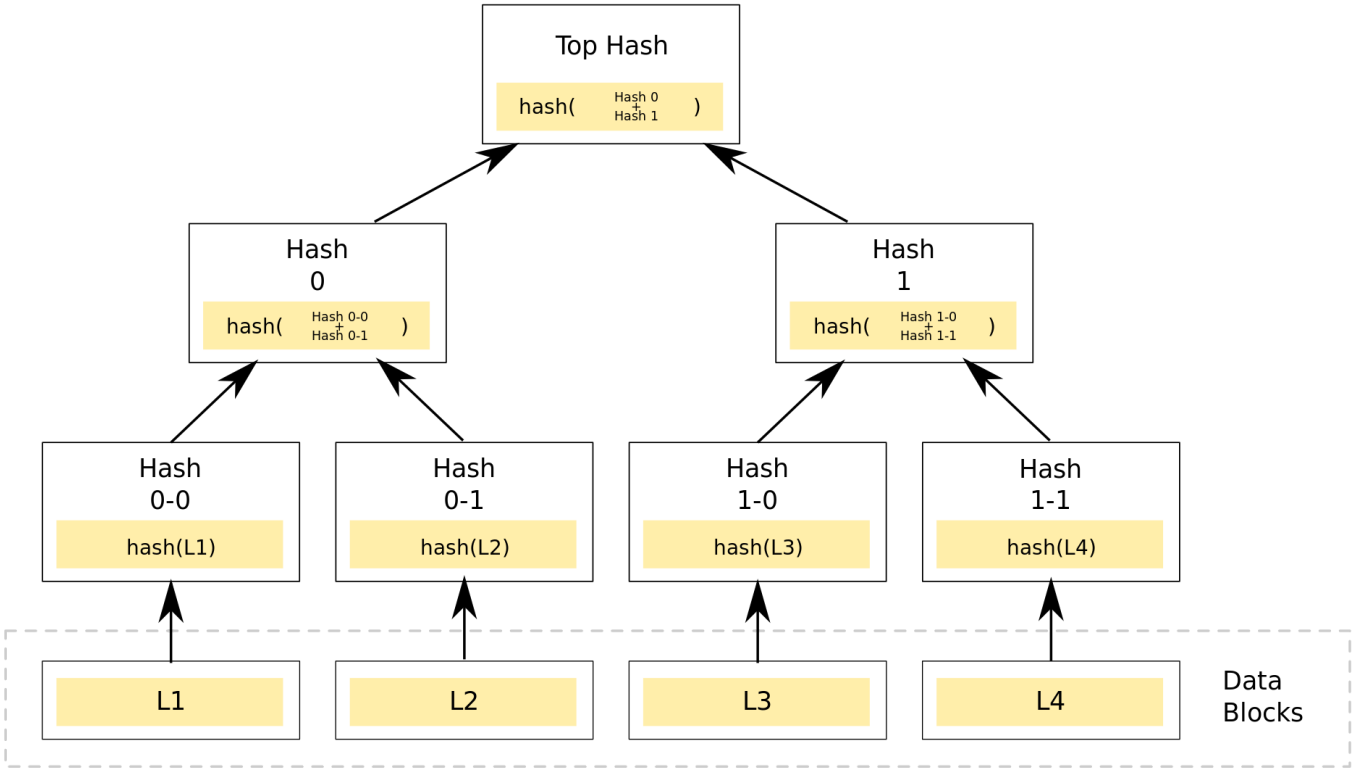


- [x] Merkle Tree

  哈希树（hash tree；Merkle tree），在密码学及计算机科学中是一种树形数据结构，每个叶节点均以数据块的哈希作为标签，而除了叶节点以外的节点则以其子节点标签的加密哈希作为标签 。哈希树能够高效、安全地验证大型数据结构的内容，是哈希链的推广形式。哈希树中，哈希值的求取通常使用诸如SHA-2的加密哈希函数，但如果只是用于防止非故意的数据破坏，也可以使用不安全的校验和获取，比如CRC。

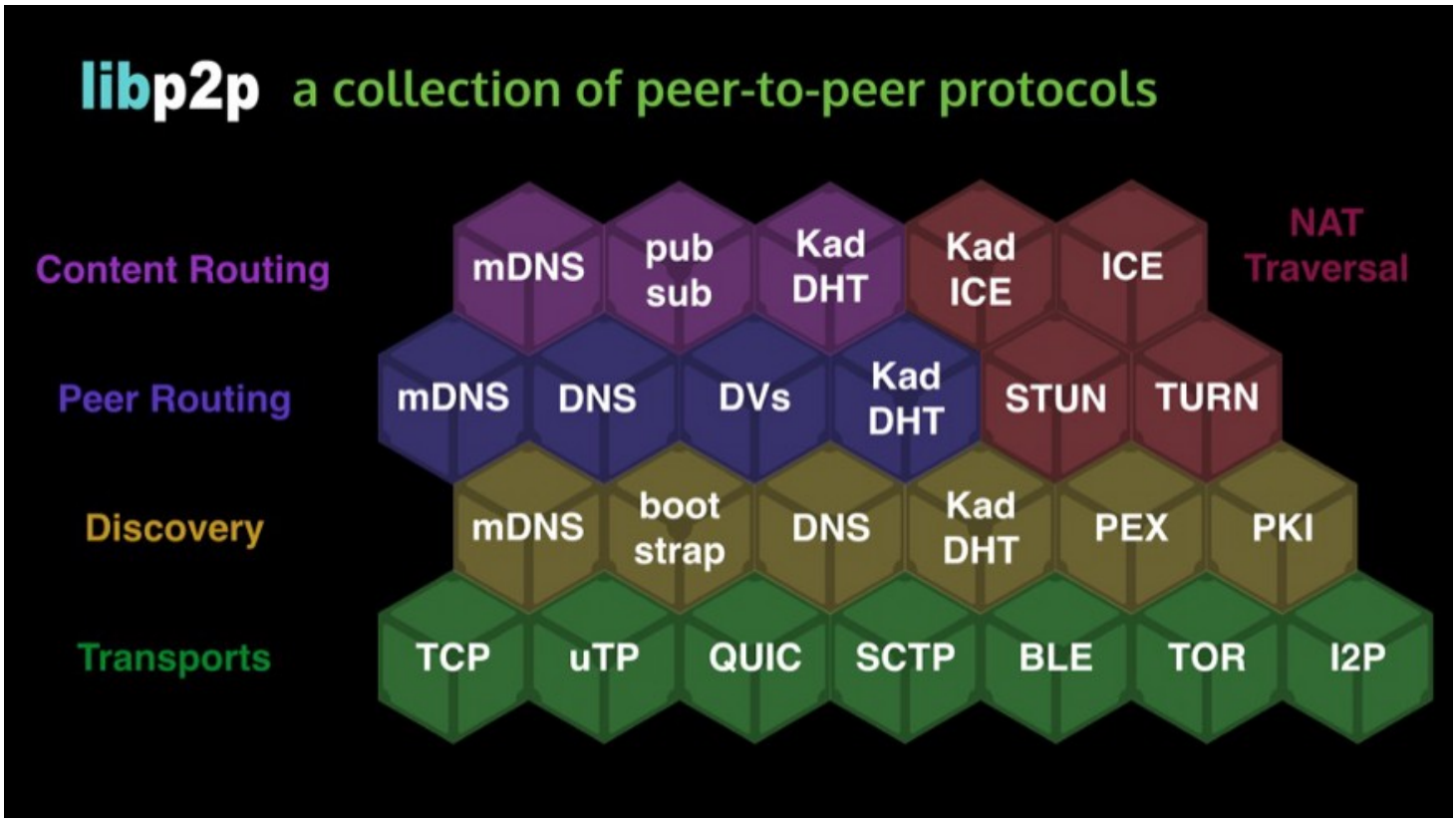
  哈希树的顶部为顶部哈希（top hash），亦称根哈希（root hash）或主哈希（master hash）。以从 P2P 网络下载文件为例：通常先从可信的来源获取顶部哈希，如朋友告知、网站分享等。得到

顶部哈希后，则整棵哈希树就可以通过 P2P 网络中的非受信来源获取。下载得到哈希树后，即可根据可信的顶部哈希对其进行校验，验证数据是否完整、是否遭受破坏。



- [x] Merkle DAGS(Merkle directed acyclic graph，默克尔有向无环图)

libp2p核心概念



- 传输(Transport)
  参考:
  - https://docs.libp2p.io/concepts/transport/
- NAT转换(NAT Traversal)
  参考:
  - https://docs.libp2p.io/concepts/nat/
- 通信安全(Secure Communication)
  参考:
  - https://docs.libp2p.io/concepts/nat/#automatic-router-configuration
- 中继传输(Circuit Relay)
  参考:
  - https://docs.libp2p.io/concepts/circuit-relay/
- 传输协议(Protocols)
  libp2p的核心协议有:
  1. ping 协议
  2. Identify 协议
  3. secio协议
  4. kad-dht 协议
  5. Circuit Relay 协议
     参考:
  - https://docs.libp2p.io/concepts/protocols/

- 节点标识(Peer Identify)
  参考:
  - https://docs.libp2p.io/concepts/peer-id/
- 内容路由 (Content Routing)
- 节点路由 (Peer Routing)
- 地址标识(Addressing)
  参考:
  - https://docs.libp2p.io/concepts/addressing/
- 安全性考虑(Security Considerations)
  参考:
  - https://docs.libp2p.io/concepts/security-considerations/
- 发布订阅模式 (Publish/Subscribe)
  参考:
  - https://docs.libp2p.io/concepts/publish-subscribe/
- 流多路复用 (Stream Mutiplexing)
  参考:
  - https://docs.libp2p.io/concepts/stream-multiplexing/

libp2p 使用

- [x] libp2p 库使用通用流程
  libp2p的使用通常分为以下几个步骤:

1. 构建libp2p 节点 construct p2p worknode

```
ctx = context.Background()
node, err = libp2p.New(ctx, option)
if err != nil {
       return ctx, host.Host(nil), err
}
```

2. 发布libp2p 节点 publish p2p worknode

```
ctx = context.Background()
 node, err = libp2p.New(ctx, nodeOptions)
 if err != nil {
       return ctx, host.Host(nil), err
 }
address, err := peerstore.AddrInfoToP2pAddrs(&peerInfo)
```

3. 连接到libp2p 节点 connect to p2p worknode

```
"github.com/libp2p/go-libp2p-core/host"
 var host host.Host
 peerInfo := peerstore.AddrInfo{
        ID:    node.ID(),
        Addrs: node.Addrs(),
 }

 host.Connect(ctx,peerinfo)
```

4. 处理节点之间网络流 handle p2p network stream

核心方法为:

```
"github.com/libp2p/go-libp2p-core/host"
var host host.Host
host.SetStreamHander(procolID,streamHandlerMethod)
func streamHandlerMethod(stream network.Stream){
//todo 业务处理方法
}
//示例 streamhandler

//todo 默认流处理函数
func Defaulp2pNetworkStreamHandler(stream network.Stream) {
   // stream.Stat().Direction 判断数据流向
   //stream.Conn()  获取连接信息
   //stream.Conn().GetStreams()
   //stream.Conn().NewStream()
   //stream.Conn().LocalPeer()
   //stream.Conn().RemotePeer()
   //...
   //stream.Protocol() 获取协议
   //stream.SetDeadline( ) // 设置过期时间
   //stream.Reset() //重置网络流

   fmt.Printf("got a new stream....")
   rw := bufio.NewReadWriter(bufio.NewReader(stream), bufio.NewWriter(stream))
   str, err := rw.ReadString('\n')
   if err != nil {
        panic(err)
   }
   fmt.Printf("str read from stream is %s \n:", str)

   _, err = rw.Write([]byte(str))
   if err!= nil{
        panic(err)
   }
}
```

5. 节点使用完成后关闭libp2p节点(可选) shutdown p2p worknode
   核心方法为:

```
"github.com/libp2p/go-libp2p-core/host"
var host host.Host
host.Close()
```

libp2p 关键库说明

- github.com/libp2p/go-libp2p-core/peer
  > 该包中包含libp2p 中网络节点peer 的表示和描述信息
  > Package peer implements an object used to represent peers in the libp2p network.

- github.com/libp2p/go-libp2p-core/host
  > 使用该包来创建libp2p中网络节点
  > Package host provides the core Host interface for libp2p.Host represents a single libp2p node in a peer-to-peer network.

- github.com/libp2p/go-libp2p/p2p/host/routed
  > RoutedHost's Connect differs in that if the host has no addresses for a given peer, it will use its routing system to try to find some.

- github.com/libp2p/go-tcp-transport
  > 该包是libp2p中对tcp传输协议的实现
  > A libp2p transport implementation for tcp, including reuseport socket options.

- github.com/libp2p/go-ws-transport
  > 该包是libp2p中对web-socket传输协议的实现
  > A libp2p transport implementation using WebSockets.

- github.com/libp2p/go-libp2p-yamux
  > libp2p 中对yamux multiplex多路复用的实现
  > An adapter to integrate the yamux multiplexer into libp2p as a stream muxer.

- github.com/libp2p/go-libp2p-mplex
  > libp2p中多路复用的实现
  > This is an implementation of the go-stream-muxer interface for multiplex

- github.com/libp2p/go-libp2p-secio
  > libp2p中安全传输模块
  > go-libp2p-secio is a component of the libp2p project, a modular networking stack for

> developing peer-to-peer applications. It provides a secure transport channel for go-libp2p. Following an initial plaintext handshake, all data exchanged between peers using go-libp2p-secio is encrypted and protected from eavesdropping.libp2p supports multiple transport protocols, many of which lack native channel security. go-libp2p-secio is designed to work with go-libp2p's "transport upgrader", which applies security modules (like go-libp2p-secio) to an insecure channel. go-libp2p-secio implements the SecureTransport interface, which allows the upgrader to secure any underlying connection

- github.com/libp2p/go-libp2p-kad-dht
  > 该包中包含DHT方法的实现，使用分布式hash表进行路由发现和内容发现时需要使用该包
  > dht implements a distributed hash table that satisfies the ipfs routing interface. This DHT is modeled after kademlia with S/Kademlia modifications.

- github.com/libp2p/go-libp2p-circuit
  > libp2p中提供中继连接

- github.com/ipfs/go-datastore
  > 抽线数据存储和操作层，当使用DHT相关功能时需要使用，如content-routing
  > Datastore is a generic layer of abstraction for data store and database access. It is a simple API with the aim to enable application development in a datastore-agnostic way, allowing datastores to be swapped seamlessly without changing application code. Thus, one can leverage different datastores with different strengths without committing the application to one datastore throughout its lifetime.

In addition, grouped datastores significantly simplify interesting data access patterns (such as caching and sharding).

- github.com/libp2p/go-libp2p-discovery
  > lib2p中对节点发现功能的实现，通常结合content-routing等功能实现时使用
  > 在libp2p中进行节点发现通常有以下几种方法mdns、randomwalk和bootstraplist等
  > 下面对这三种发现方式进行说明:
  > 1.mDNS-发现 是一种在局域网上使用 mDNS 的发现协议。它发射了mDNS信标来查找是否有更多的对等体可用。局域网节点对于对等协议是非常有用的，因为它们的低延迟链路.mDNS-发现是一种独立的协议，不依赖于任何其他的 libp2p 协议。在不依赖其他基础设施的情况下，mDNS-发现 可以产生局域网中可用的对等点. 这在内联网、与互联网主干断开的网络以及暂时失去链路的网络中尤其有用.mDNS-discovery 可以针对每个服务进行配置(i.e. 即仅发现参与特定协议的对等体，如IPFS), 还有私有网络(发现属于专用网络的对等体).
  > 隐私注意：mDNS 在局域网中进行广告，在同一本地网络中向听众显示IP地址。不推荐使用隐私敏感的应用程序或太公开的路由协议.
  > 2.随机游走是DHTS（具有路由表的其他协议）的发现协议。它进行随机DHT查询，以便快速了解大量的对等体。这导致DHT（或其他协议）收敛得更快，而在初始阶段需要承担一定负载

> 开销.
> 3.Bootstrap列表是一种发现协议，它使用本地存储来缓存网络中可用的高度稳定的（和一些可信的）对等点的地址。这允许协议“找到网络的其余部分”。这基本上与DNS自举的方式相同（尽管注意到，通过设计改变DNS引导列表——“点域”地址——不容易做到）.该列表应该存储在长期本地存储中，无论这意味着本地节点（例如磁盘）。协议可以将默认列表硬编码或采用标准代码分发机制（如DNS）进行传送。在大多数情况下（当然在IPFS的情况下），引导列表应该是用户可配置的，因为用户可能希望建立单独的网络，(or place their reliance and trust in specific nodes)或者将它们的信任和信任放在特定的节点中.

> This package contains interfaces and utilities for active peer discovery. Peers providing a service use the interface to advertise their presence in some namespace. Vice versa, peers seeking a service use the interface to discover peers that have previously advertised as service providers. The package also includes a baseline implementation for discovery through Content Routing.

- github.com/multiformats/go-multiaddr
  > mutiaddr的go-libp2p实现
  > Multiaddr is a standard way to represent addresses that:
  > Support any standard network protocols.
  > Self-describe (include protocols).
  > Have a binary packed format.
  > Have a nice string representation.
  > Encapsulate well

- github.com/libp2p/go-libp2p-pubsub
  > go-libp2p中发现订阅模式实现
  > This repo contains the canonical pubsub implementation for libp2p. We currently provide three message router options:
  > Floodsub, which is the baseline flooding protocol.Randomsub, which is a simple probabilistic router that propagates to random subsets of peers.
  > Gossipsub, which is a more advanced router with mesh formation and gossip propagation.
  > See spec and implementation for more details.

libp2p库中各模块使用示例demo

- 传输(Transport)
- NAT转换(NAT Traversal)
  libp2p中NAT转换可分为手动方式和自动方式,手动方式使用"github.com/libp2p/go-libp2p-na"包进行NAT-Discovery和nat.NewMapping()进行端口映射。而自动方式可使用"github,com/go-libp2p-

autonat"进行自动端口映射。需要注意的是在NAT转换时需要注意MAP映射事件的处理,如映射失败,映射端口发生变化等。

- 通信安全(Secure Communication)

- [x] 启用节点公私钥认证和传输流量加密

```
        security := libp2p.Security(secio.ID, secio.New) // 传输加密
        if options.NodeIdentifyPrivStr != "" { //节点身份ID设置
                priv, _, err := utils.GenSecurekeysByStr(options.NodeIdentifyPrivStr)
                p2pnetworkhostOptions = append(p2pnetworkhostOptions, libp2p.Identity(priv))
        }
        //设置节点启动配置 basic node host
        p2pnetworkhostOptions = append(p2pnetworkhostOptions, security, muxers, transports, listenAddrOptions)
        networknode, err = p2pnetwork.CreatenetworkNodehost(ctx, dhtObj, p2pnetworkhostOptions)
}
```

- 中继传输(Circuit Relay)

- [x] 使用中继节点方式进行节点联通与传输

```
    if option.PeerID != "" { //从命令行参数中获取中转节点PEER
        if option.RelayID != "" { //存在中继节点，则通过中继节点连接
                rawPeerIDWithIP := strings.Split(option.PeerID, "/")
                rawPeerID := rawPeerIDWithIP[len(rawPeerIDWithIP)-1]
                ID, err := peerstore.Decode(rawPeerID)
                logger.Printf("relayID is %s,and peerID is %s and targeID is %s\n", option.RelayID, option.PeerID
                relayAddr, err := ma.NewMultiaddr(option.RelayID + "/p2p-circuit" + option.PeerID)
                //根据地址信息拿到
                //peer, err := peerstore.AddrInfoFromP2pAddr(relayAddr)
                peer := peerstore.AddrInfo{
                        ID:    ID,
                        Addrs: []ma.Multiaddr{relayAddr},
                }
                if err = p2pNode.Connect(option.ctx, peer); err != nil {
                        logger.Errorf("node %s connect to peer node %s with relayaddr  %s failed ,error is %s ...
                }
        }
```

- 传输协议(Protocols)
  - [x] 传输协议标识方法

```
transports := libp2p.ChainOptions(
     libp2p.Transport(tcp.NewTCPTransport),
     libp2p.Transport(ws.New),
)
listenAddr := libp2p.ListenAddrStrings(
     "/ip4/172.16.171.94/tcp/0", "/ip6/::/tcp/0/ws")
p2pOptions = append(p2pOptions, muxers, security, listenAddr, transports)

basicHost, err := libp2p.New(ctx, p2pOptions...)
```

- 节点标识(Peer Identify)
  - [x] 采用自定义公私钥方式进行节点标识

```
// Generate a key pair for this host
priv, _, err := crypto.GenerateKeyPair(crypto.RSA, 2048)
if err != nil {
     return nil, nil, err
}

ctx := context.Background()

opts := []libp2p.Option{
     libp2p.ListenAddrStrings(fmt.Sprintf("/ip4/0.0.0.0/tcp/%d", listenPort)),
     libp2p.Identity(priv),
     libp2p.DefaultTransports,
     libp2p.DefaultMuxers,
     libp2p.DefaultSecurity,
     libp2p.NATPortMap(),
}

basicHost, err := libp2p.New(ctx, opts...)
if err != nil {
     return nil, nil, err
}
```

- 内容路由 (Content Routing)
- [x] 基于DHT数据结构的内容发现

```
dstore := dsync.MutexWrap(ds.NewMapDatastore())
dhtObj, err := dht.New(ctx, basicHost, dht.Datastore(dstore), dht.Mode(dht.ModeServer))

// step 3构建 需要存储在分布式dht 数据结果中的数据信息
//dhtObj := dht.NewDHT(ctx, basicHost, dstore)
data := []byte("this is some test content")
hash, _ := mh.Sum(data, mh.SHA2_256, -1)
contentId := cid.NewCidV1(cid.DagCBOR, hash)
// step4 发布/共享该数据
if err = dhtObj.Provide(ctx, contentId, false); err != nil {
     log.Error(err)
}
// Make the routed host
//构造 routed host ，routedhost 为包含dht信息的basichost
routedHost := rhost.Wrap(basicHost, dhtObj)
// Bootstrap the host
dhtObj.Bootstrap(ctx)
// Build host multiaddress
hostAddr, _ := ma.NewMultiaddr(fmt.Sprintf("/ipfs/%s", routedHost.ID().Pretty()))

//dht_1 := dht.New(ctx, basicHost_1, dstore)
basicHost_addressinfo := peer.AddrInfo{
     ID:    routedHost.ID(),
     Addrs: routedHost.Addrs(),
}
//works both 创建相同的dht 对象
//dhtobj1, err := dht.New(ctx, basicHost_1, dht.Datastore(dstore_1), dht.Mode(dht.ModeServer), dht.BootstrapPe
dhtobj1, err := dht.New(ctx, basicHost_1, dht.Datastore(dstore_1), dht.Mode(dht.ModeServer))

//dtbatch,err := dstore.Batch()
//dhtobj1  = dht.NewDHTClient(ctx, basicHost_1, dtb)
routedhost1 := rhost.Wrap(basicHost_1, dhtobj1) //关键点

if err = dhtobj1.Bootstrap(ctx); err != nil { //关键点
     log.Error(err)
}
// 将两个节点进行连接，注意此处是目标节点信息 可通过节点发现、固定参数等形式获取
if err = routedhost1.Connect(ctx, basicHost_addressinfo); err != nil { //关键点
     log.Error(err)
}



for {
     time.Sleep(5 * time.Second)
     //查询提供该内容的节点对象
     peers, err := dhtobj1.FindProviders(ctx, contentId) //关键点
     if err != nil {
             log.Error(err)
     }
     if len(peers) <= 0 {
             log.Println("found zero peers....", peers)
```

```
        }
        for _, peer := range peers {
                fmt.Printf("found peer %s provider contedt %s\n", peer, contentId.String())
        }

    }
```

- 节点路由 (Peer Routing)

- [x] 基于mdns的节点发现与节点路由(原始)

```
type discoveryNotifee struct {
        PeerChan chan peer.AddrInfo
}

//interface to be called when new  peer is found
func (n *discoveryNotifee) HandlePeerFound(pi peer.AddrInfo) {
        n.PeerChan <- pi
}

//Initialize the MDNS service
func initMDNS(ctx context.Context, peerhost host.Host, rendezvous string) chan peer.AddrInfo {
        // An hour might be a long long period in practical applications. But this is fine for us
        ser, err := discovery.NewMdnsService(ctx, peerhost, time.Hour, rendezvous)
        if err != nil {
                panic(err)
        }

        //register with service so that we get notified about peer discovery
        n := &discoveryNotifee{}
        n.PeerChan = make(chan peer.AddrInfo)

        ser.RegisterNotifee(n)
        return n.PeerChan
}
```

- [x] 基于mdns的节点发现与节点路由(dht)

```
// Start a DHT, for use in peer discovery. We can't just make a new DHT
        // client because we want each peer to maintain its own local copy of the
        // DHT, so that the bootstrapping node of the DHT can go down without
        // inhibiting future peer discovery.
        //kademliaDHT, err := dht.New(ctx, host)
        kademliaDHT, err := dht.New(option.ctx, p2pNode, dht.Mode(dht.ModeServer))
        //kademliaDHT, err :=dht.New(ctx, host,dht.Mode(dht.ModeServer),dht.ProtocolPrefix(protocol.ID(config.Prc


        // Bootstrap the DHT. In the default configuration, this spawns a Background
        // thread that will refresh the peer table every five minutes.
        logger.Debug("Bootstrapping the DHT")
        if err = kademliaDHT.Bootstrap(ctx); err != nil {
                panic(err)
        }
        // We use a rendezvous point "meet me here" to announce our location.
        // This is like telling your friends to meet you at the Eiffel Tower.
        logger.Info("Announcing ourselves...")
        routingDiscovery := discovery.NewRoutingDiscovery(kademliaDHT)

        discovery.Advertise(ctx, routingDiscovery, config.RendezvousString)
        peerChan, err := routingDiscovery.FindPeers(ctx, config.RendezvousString)
```

- [x] 基于Rendezvous 协议的节点发现和节点路由
  注意:通常使用bootstrap节点作为Rendezvous节点

```
// Start a DHT, for use in peer discovery. We can't just make a new DHT
  // client because we want each peer to maintain its own local copy of the
  // DHT, so that the bootstrapping node of the DHT can go down without
  // inhibiting future peer discovery.
  kademliaDHT, err := dht.New(ctx, host)
  if err != nil {
        panic(err)
  }

  // Bootstrap the DHT. In the default configuration, this spawns a Background
  // thread that will refresh the peer table every five minutes.
  logger.Debug("Bootstrapping the DHT")
  if err = kademliaDHT.Bootstrap(ctx); err != nil {
        panic(err)
  }

  // Let's connect to the bootstrap nodes first. They will tell us about the
  // other nodes in the network.
  var wg sync.WaitGroup
  for  _, peerAddr := range config.BootstrapPeers {
        peerinfo, _ := peer.AddrInfoFromP2pAddr(peerAddr)
        wg.Add(1)
        go func() {
                defer wg.Done()
                if err := host.Connect(ctx, *peerinfo); err != nil {
                        logger.Warning(err)
                } else {
                        logger.Info("Connection established with bootstrap node:", *peerinfo)
                }
        }()
  }
  wg.Wait()

  // We use a rendezvous point "meet me here" to announce our location.
  // This is like telling your friends to meet you at the Eiffel Tower.
  logger.Info("Announcing ourselves...")
  routingDiscovery := discovery.NewRoutingDiscovery(kademliaDHT)
  discovery.Advertise(ctx, routingDiscovery, config.RendezvousString)
  logger.Debug("Successfully announced!")

  // Now, look for others who have announced
  // This is like your friend telling you the location to meet you.
  logger.Debug("Searching for other peers...")
  peerChan, err := routingDiscovery.FindPeers(ctx, config.RendezvousString)
```

- 地址标识(Addressing)

> libp2p中节点标识形式为:/ip4或ip6/监听IP/协议名称/监听端口/p2p/节点标记HASH
> 示例:
> /ip4/7.7.7.7/tcp/4242/p2p/QmYyQSo1c1Ym7orWxLYvCrM2EmxFTANf8wXmmE7DWjhx5N
> /ip6/7.7.7.7/udp/4242/p2p/QmYyQSo1c1Ym7orWxLYvCrM2EmxFTANf8wXmmE7DWjhx5N
> /ip6/7.7.7.7/ws/4242/p2p/QmYyQSo1c1Ym7orWxLYvCrM2EmxFTANf8wXmmE7DWjhx5N

- [x] 标识节点监听地址

```
// Generate a key pair for this host
priv, _, err := crypto.GenerateKeyPair(crypto.RSA, 2048)
ctx := context.Background()
opts := []libp2p.Option{
        libp2p.ListenAddrStrings(fmt.Sprintf("/ip4/0.0.0.0/tcp/%d", listenPort)),
        libp2p.Identity(priv),
}
        basicHost, err := libp2p.New(ctx, opts...)
```

- 安全性考虑(Security Considerations)

  > libp2p makes it simple to establish encrypted, authenticated communication channels between two peers, but there are other important security issues to consider when building robust peer-to-peer systems.

libp2p中对于安全性相关的设置主要体现在以下两点:
1.节点之间构建加密的安全通信链路对流量加密
2.采用公私钥的方式进行节点加密认证

- 发布订阅模式 (Publish/Subscribe)

- [x] 发布订阅模式使用实例

```
// 发布订阅模式 ， 该处可选
   //   pubs ,err := pubsub.NewGossipSub(ctx,p2pnetwork.NetworkBasicHost)
   //pubs, err := pubsub.NewGossipSub(ctx, p2pnetwork.NetworkBasicHost)
   pubs, err := PubsubgossipGen(ctx, networknode.BasicNodeHost)
   if err != nil {
        log.Error(err)
   }
   sub, topsub, err := PubsubtopicsJoin(pubs, Pubsub_Default_Topic)
   if err != nil {
        log.Error(err)
   }

   // 设置mdns发现处理方法
   err = MdnsDiscoverySetup(ctx, networknode.BasicNodeHost, DiscoveryInterval, DiscoveryServiceTag)
   if err != nil {
        log.Error(err)
   }

   go func() {
        for {
                msg := new(PubsubMessage)
                msg.SenderPeer = networknode.BasicNodeHost.ID().Pretty()
                msg.PMessageStr = "hello world"
                msg.SenderFrom = "from localhost"
                err = PubsubTopicPubish(ctx, *msg, topsub, nil)
                if err != nil {
                        log.Error(err)
                }
                time.Sleep(3 * time.Second)
        }

   }()
   msgChan := make(chan interface{})

   go PubsubMsgHandler(sub, ctx, networknode.BasicNodeHost, msgChan)

   signalChan := make(chan os.Signal, 1)
   errChan := make(chan error, 1)
   signal.Notify(signalChan, syscall.SIGINT, syscall.SIGTERM)
   for {
        select {
        case msg := <-msgChan:
                fmt.Println("msg from msg chan is:", msg)
                for _, peersFind := range PubsubPeersList(pubs, Pubsub_Default_Topic) {
                        peerinfo, err := dhtObj.FindPeer(ctx, peersFind)
                        if err != nil {
                                continue
                        }
                        if err = networknode.BasicNodeHost.Connect(ctx, peerinfo); err != nil {
                                fmt.Print("connect b")
                        }
```

```
                        fmt.Printf("peer found by dht is %s\n", peerinfo.String())
                }
        }
    }
}
```

- 流多路复用 (Stream Mutiplexing)

核心示例代码如下:
`

`

参考资料

1. https://libp2p.io/ [官方网站]
2. https://github.com/libp2p [官方github]
3. https://github.com/libp2p/specs [libp2p 标准规范]
4. https://github.com/libp2p/go-libp2p [libp2p golang 语言实现]
5. https://github.com/libp2p/go-libp2p-examples [go-libp2p 官方example 库]
6. https://docs.ipfs.io/concepts/what-is-ipfs/ [ipfs 官方网站]
7. https://en.wikipedia.org/wiki/Distributed_hash_table [分布式hash表 dht]
8. https://docs.ipfs.io/concepts/dht/ [dht-concepts]
9. https://docs.ipfs.io/ [ipfs]
10. https://www.ietf.org/proceedings/65/slides/plenaryt-2.pdf [dht -slide]
11. https://www.tecposter.cn/article/libp2p-spec-simple
12. http://www.wjblog.top/articles/f80288a7/ [NAT
13. https://www.usenix.org/legacy/publications/library/proceedings/osdi2000/full_papers/gribble/gribble_html/node4.html [hash table]
14. https://www.coursera.org/lecture/data-structures/distributed-hash-tables-tvH8H
15. https://colobu.com/2018/03/26/distributed-hash-table/[分布式hash表]
16. https://en.wikipedia.org/wiki/Key-based_routing
17. https://program-think.blogspot.com/2017/09/Introduction-DHT-Kademlia-Chord.html[分布式hash表和KAD算法]