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

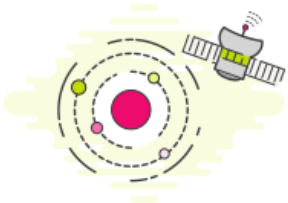
libp2p最初作为IPFS项目中的网络层实现，用以解决IPFS项目中IPFS-Node节点的网络联通、数据交换等问题.后来从IPFS项目中分离出来作为独立的网络库，用以解决现代网络环境下构建P2P网络应用的需求。 **libp2p来源于IPFS项目，目前已经从ipfs项目中独立出来作为一个专注于P2P网络的网络库**

> 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

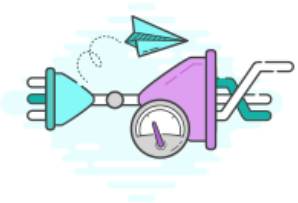
Features



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

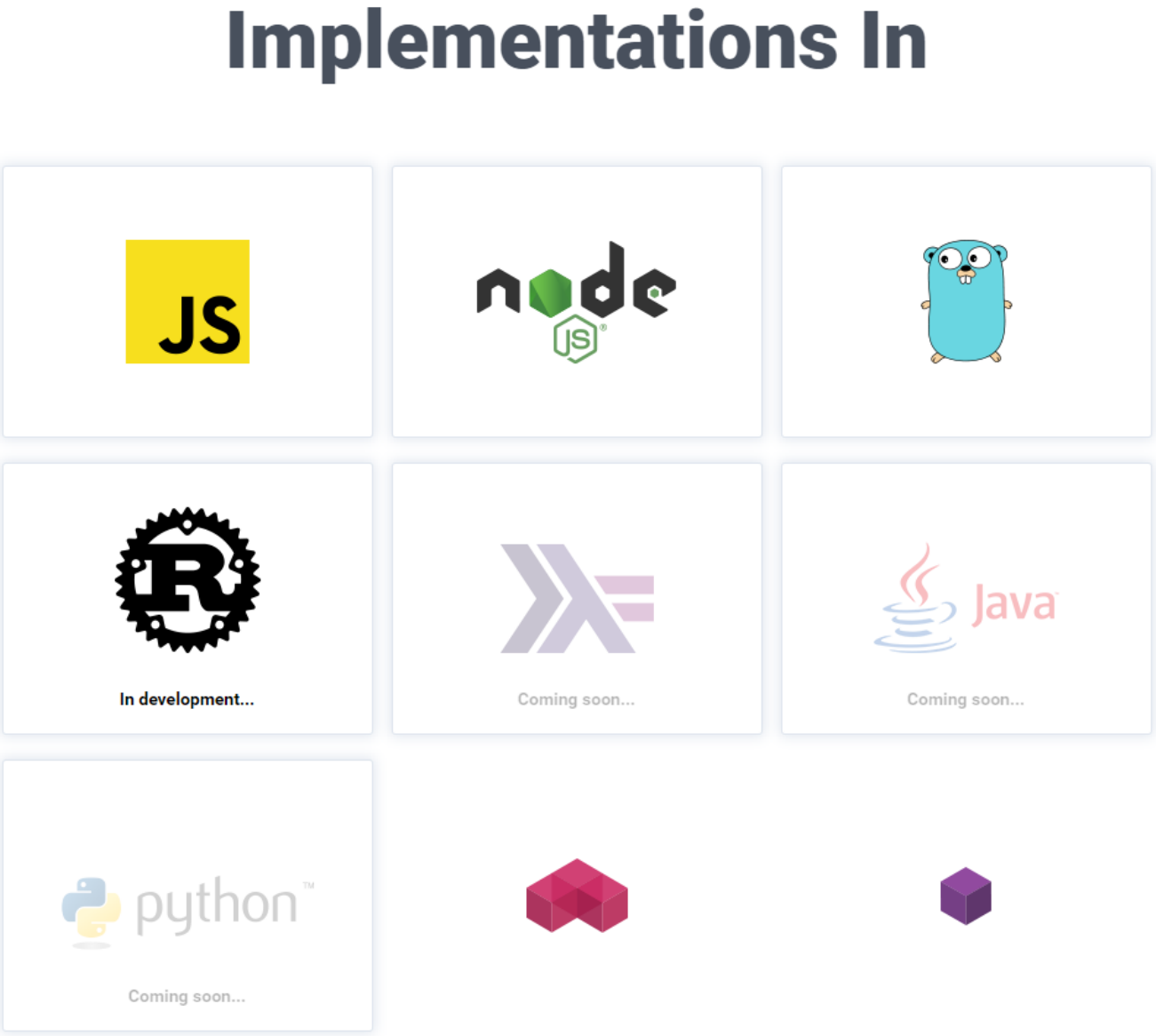
Upgrade Without Compromises

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

- Circuit Relay

> A means of establishing communication between peers who are unable to communicate directly, with the assistance of a third peer willing and able to act as an intermediary.
> In many real-world peer-to-peer networks, direct communication between all peers may be impossible for a variety of reasons. For example, one or more peers may be behind a firewall or have NAT traversal issues. Or maybe the peers don’t share any common transports.
> In such cases, it’s possible to “bridge the gap” between peers, so long as each of them are capable of establishing a connection to a willing relay peer. If I only speak TCP and you only speak websockets, we can still hang out with the help of a bilingual pal.

> Circuit relay is implemented in libp2p according to the relay spec, which defines a wire protocol and addressing scheme for relayed connections.

- Client / Server

> A network architecture defined by the presence of central “server” programs which provide services and resources to a (usually much larger) set of “client” programs. Typically clients do not communicate directly with one another, instead routing all communications through the server, which is inherently the most privileged member of the network.

- 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

> A distributed hash table whose contents are spread throughout a network of participating peers. Much like an in-process hash table, values are associated with a key and can be retrieved by key. Most DHTs assign a portion of the addressable key space to nodes in a deterministic manner, which allows for efficient routing to the node responsible for a given key.

*Connection

> A libp2p connection is a communication channel that allows peers to read and write data. Connections between peers are established via transports, which can be thought of as “connection factories”. For example, the TCP transport allows you to create connections that use TCP/IP as their underlying substrate.

*Dial

> The process of opening a libp2p connection to another peer is known as “dialing”, and accepting connections is known as “listening”. Together, an implementation of dialing and listening forms a transport.

*Listen

> The process of accepting incoming libp2p connections is known as “listening”, and it allows other peers to “dial” up and open network connections to your peer.

*mDNS

> Multicast DNS is a protocol for service discovery on a local network. One of libp2p’s peer routing implementations leverages mDNS to discover local peers quickly and efficiently.

- multiaddr

> A multiaddress (often abbreviated multiaddr), is a convention for encoding multiple layers of addressing information into a single “future-proof” path structure.
> For example: /ip4/127.0.0.1/udp/1234 encodes two protocols along with their essential addressing information. The /ip4/127.0.0.1 informs us that we want the 127.0.0.1 loopback address of the IPv4 protocol, and /udp/1234 tells us we want to send UDP packets to port 1234.
> Multiaddresses can be composed to describe multiple “layers” of addresses.
> For more detail, see Concepts > Addressing, or the multiaddr spec, which has links to many implementations.

- Multiaddress

> See multiaddr

- Multihash

> Multihash is a convention for representing the output of many different cryptographic hash functions in a compact, deterministic encoding that is accommodating of future changes.
> Hashes are central to many systems (git, for example), yet many systems store only the hash output itself, since the choice of hash function is an implicit design parameter of the system. This has the unfortunate effect of making it quite difficult to ever change your mind about what kind of hash function your system uses!
> A multihash encodes the type of hash function used to produce the output, as well as the length of the output in bytes. This is added as a two-byte header to the original hash output, and in return for those two bytes, the header allows current and future systems to easily identify and validate many hash functions by leveraging common libraries. As new functions are added, you can much more easily extend your application or protocol to support them, since the old and new hash outputs will be easily distinguishable from one another.

> The most prominent use of multihashes in libp2p is in the PeerId, which contains a hash of a peer’s public key. However, systems built with libp2p, most notably IPFS, use multihashes for other purposes. In the IPFS case, multihashes are used both to identify content and other peers, since IPFS uses libp2p and shares the same PeerId conventions.

> In IPFS, multihashes are a key component of the CID, or content identifier, and the “v0” version of CID is a “raw” multihash of a piece of content. A “modern” CID combines a multihash of some content with some compact contextualizing metadata, allowing content-addressed systems like IPFS to create more meaningful links between hash-addressed data. For more on the subject of hash-linked data structures in p2p systems, see IPLD.
> Multihashes are often represented as base58-encoded strings, for example, QmYyQSo1c1Ym7orWxLYvCrM2EmxFTANf8wXmmE7DWjhx5N. The first two characters Qm are

> the multihash header for the SHA-256 hash algorithm with a length of 256 bits, and are common to all base58-encoded multihashes using SHA-256.

*Multiplexing

> Multiplexing (or “muxing”), refers to the process of combining multiple streams of communication over a single logical “medium”. For example, we can maintain multiple independent data streams over a single TCP network connection, which is itself of course being multiplexed over a single physical connection (ethernet, wifi, etc).
> Multiplexing allows peers to offer many protocols over a single connection, which reduces network overhead and makes NAT traversal more efficient and effective.
> libp2p supports several implementations of stream multiplexing. The mplex specification defines a simple protocol with implementations in several languages. Other supported multiplexing protocols include yamux and spdy.
> See Stream Muxer Implementations for status of multiplexing across libp2p language implementations.

- multistream

> 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

> Network address translation in general is the mapping of addresses from one address space to another, as often happens at the boundary of private networks with the global internet. It is especially essential in IPv4 networks (which are still the vast majority), as the address space of IPv4 is quite limited. Using NAT, a local, private network can have a vast range of addresses within the internal network, while only consuming one public IP address from the global pool.
> An unfortunate effect of NAT in practice is that it’s much easier to make outgoing connections from the private network to the public one than it is to call from outside in. This is because machines listening for connections on the internal network need to explicitly tell the router in charge of NAT that it should forward traffic for a given port (the multiplexing abstraction for the OS networking layer) to the listening machine.
> This is less of an issue in a client / server model, because outgoing connections to the server give the router enough information to route the response back to the client where it needs to go.

> In the peer-to-peer model, accepting connections from other peers is often just as important as initiating them, which means that we often need our peers to be publicly reachable from the global

> internet. There are many viable approaches to NAT Traversal, several of which are implemented in libp2p.

- NAT Traversal

> NAT traversal refers to the process of establishing connections with other machines across a NAT boundary. When crossing the boundary between IP networks (e.g. from a local network to the global internet), a Network Address Translation process occurs which maps addresses from one space to another.
> For example, my home network has an internal range of IP addresses (10.0.1.x), which is part of a range of addresses that are reserved for private networks. If I start a program on my computer that listens for connections on its internal address, a user from the public internet has no way of reaching me, even if they know my public IP address. This is because I haven't made my router aware of my program yet. When a connection comes in from the internet to my public IP address, the router needs to figure out which internal IP to route the request to, and to which port.

> There are many ways to inform one's router about services you want to expose. For consumer routers, there's likely an admin interface that can setup mappings for any range of TCP or UDP ports. In many cases, routers will allow automatic registration of ports using a protocol called upnp, which libp2p supports. If enabled, libp2p will try to register your service with the router for automatic NAT traversal.
> In some cases, automatic NAT traversal is impossible, often because multiple layers of NAT are involved. In such cases, we still want to be able to communicate, and we especially want to be reachable and allow other peers to dial in and use our services. This is the one of the motivations for Circuit Relay, which is a protocol involving a "relay" peer that is publicly reachable and can route traffic on behalf of others. Once a relay circuit is established, a peer behind an especially intractable NAT can advertise the relay circuit's multiaddr, and the relay will accept incoming connections on our behalf and send us traffic via the relay.

- Node

> The word "node" is quite overloaded in general programming contexts, and this is especially the case in peer-to-peer networking circles.
> One common usage is when "node" refers to a single instance of a peer-to-peer software system, running at some time and place in the universe. For example, I'm running an orbit-db node in AWS. I think it's on version 3.2.0. In this usage, "node" refers to the whole software program (the daemon in unix-speak) which participates in the network. In this documentation, we'll often use "peer" for this purpose instead, and the two terms are often used interchangeably in various p2p software discussions.
> Another quite different meaning is the node.js javascript runtime environment, which is one of the

> supported runtimes for the javscript libp2p implementation. In general it should be pretty clear from context when “node” is referring to node.js.

> Many members of our community are excited about graphs in many contexts, so the graph terminology of “nodes and edges” is often used when discussing various subjects. Some common contexts for graph-related discussions:
> When discussing the topology or structure of a peer-to-peer network, “node” is often used in the context of a graph of connected peers. Efficient construction and traversal of this graph is key to effective peer routing.
> When discussing data structures, “node” is often useful for referring to key elements of the structure. For example, a linked list consists of many “nodes” containing both a value and a link (or, in graph terms, an “edge”) connecting it to the next node. Since many useful and interesting data structures can be described as graphs, much of the terminology of graph theory applies when discussing their properties. In particular, IPFS is naturally well-suited to storing and manipulating data structures which form a Directed Acyclic Graph, or DAG.
> An especially interesting data structure for many in our community is IPLD, or Interplanetary Linked Data. Similar to libp2p, IPLD grew out of the real-world needs of IPFS, but is broadly useful and interesting in many contexts outside of IPFS. IPLD discussions often involve “nodes” of all the types discussed here.

- Overlay

> An “overlay network” or just “overlay” refers to the logical structure of a peer-to-peer network, which is “overlaid” on top of the underlying transport mechanisms used for lower-level network communication.
> Peer-to-peer systems are generally composed of one or more overlay networks, which determine how peers are identified and located, how messages are propagated throughout the system, and other key properties.
> Examples of overlay networks used in libp2p are the DHT implementation, which is based on Kademlia, and the networks formed by participants in the various pubsub implementations.

- 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

> A unique, verifiable identifier for a peer that is impossible for another peer to forge or impersonate without trivial detection. In libp2p, peers are identified by their PeerId, which is both globally unique and allows other peers to obtain the peer’s cryptographic public key.
> The most common form of PeerId is a multihash of a peer’s public key, which can be used to fetch the entire public key from the DHT for encryption or signature verification. There is also

> experimental support for embedding or “inlining” small public keys directly into the PeerId, however, this is an area of ongoing discussion and should be treated with caution in production systems until finalized.
> An important property of cryptographic peer identities is that they are decoupled from transport, allowing peers to verify the identity of other peers regardless of what underlying network they might use to communicate. This also gives them a much longer “shelf life” than location-based identifiers (for example, IP addresses), since identities remain stable across address changes.

- 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

> In general, refers to “publish / subscribe”, a communication pattern in which participants “subscribe” for updates “published” by other participants, often on a named “topic”.
> libp2p defines a pubsub spec, with links to several implementations in supported languages.
> Pubsub is an area of ongoing research and development, with multiple implementations optimized for different use cases and environments.

- Protocol

> In general, a set of rules and data structures used for network communication.
> libp2p is comprised of many protocols and makes use of many others provided by the operating system or runtime environment.
> Most core libp2p functionality is defined in terms of protocols, and libp2p protocols are identified using multistream headers.

- Protocol Negotiation

> The process of reaching agreement on what protocol to use for a given stream of communication.
> In libp2p, protocols are identified using a convention called multistream, which adds a small header to the beginning of a stream containing a unique name, including a version identifier.
> When two peers first connect, they exchange a handshake to agree upon what protocols to use.

> The implementation of the libp2p handshake is called multistream-select.
> For details, see the protocol negotiation article.

- Stream

> TODO: Distinguish between the various types of “stream”. Could refer to
> raw tcp connection
> one component of a multistream connection
> node.js streams / pull-streams
> 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.

- Switch

> A libp2p component responsible for composing multiple transports into a single interface, allowing application code to dial peers without having to specify what transport to use.
> In addition to managing transports, the switch also coordinates the “connection upgrade” process, which promotes a “raw” connection from the transport layer into one that supports protocol negotiation, stream multiplexing, and secure communications.
> Sometimes called “swarm” for historical reasons.

- Topology

> In a peer-to-peer context, usually refers to the shape or structure of the overlay network formed by peers as they communicate with each other.

- Transport

> In libp2p, transport refers to the technology that lets us move bits from one machine to another. This may be a TCP network provided by the operating system, a websocket connection in a browser, or anything else capable of implementing the transport interface.
> Note that in some environments such as javascript running in the browser, not all transports will be available. In such cases, it may be possible to establish a Circuit Relay with the help of a peer that can support many common transports. Such a relay can act as a “transport adapter” of sorts, allowing peers that can’t communicate with each other directly to interact. For example, a peer in the browser that can only make websocket connections could relay through a peer able to make TCP connections, which would enable communication with a wider variety of peers.

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)

  > 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.

.

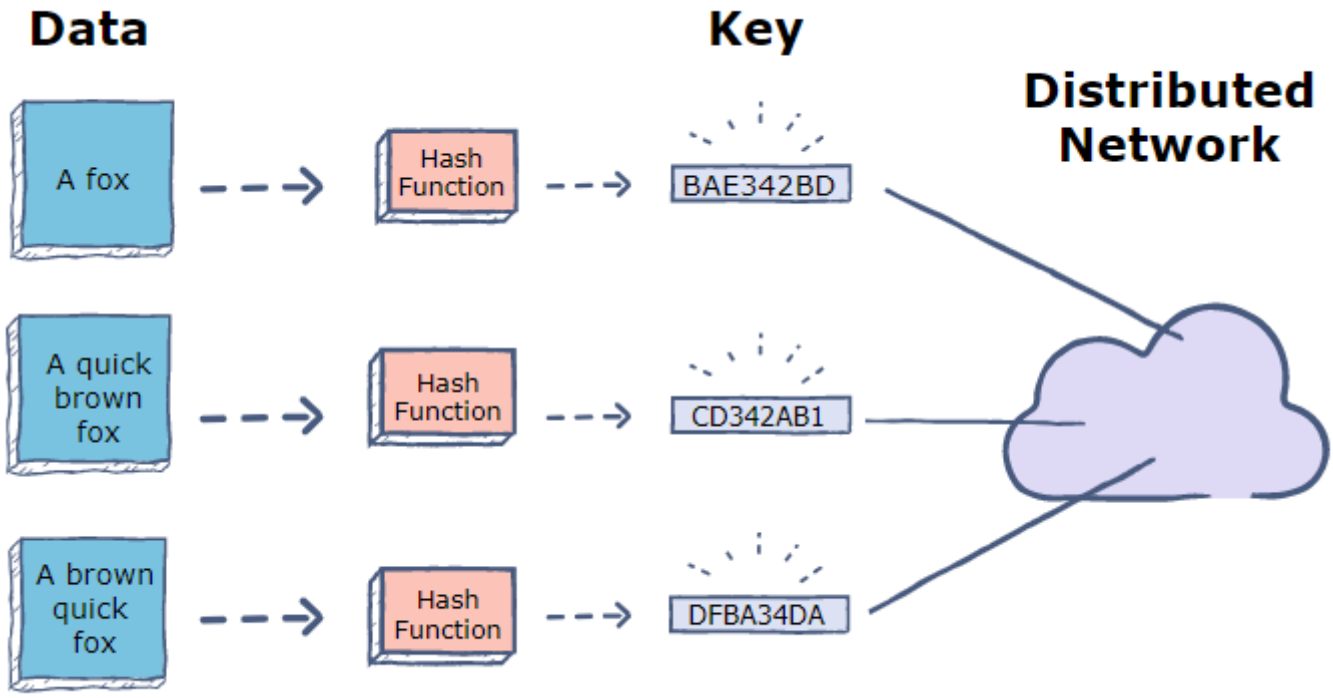


Figure: Distributed hash table architecture: each box in the diagram represents a software process. In the simplest case, each process runs on its own physical machine, however there is nothing preventing processes from sharing machines.

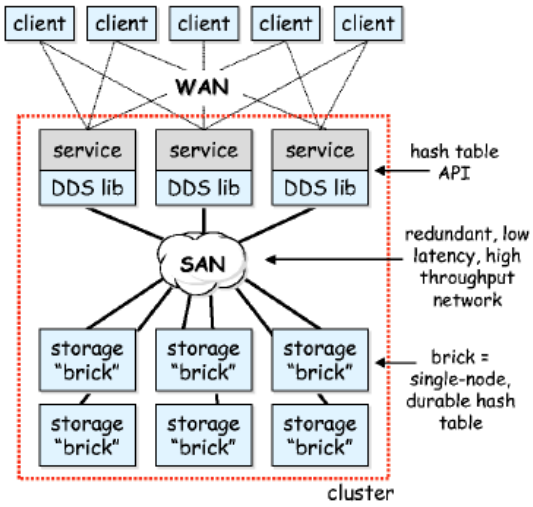
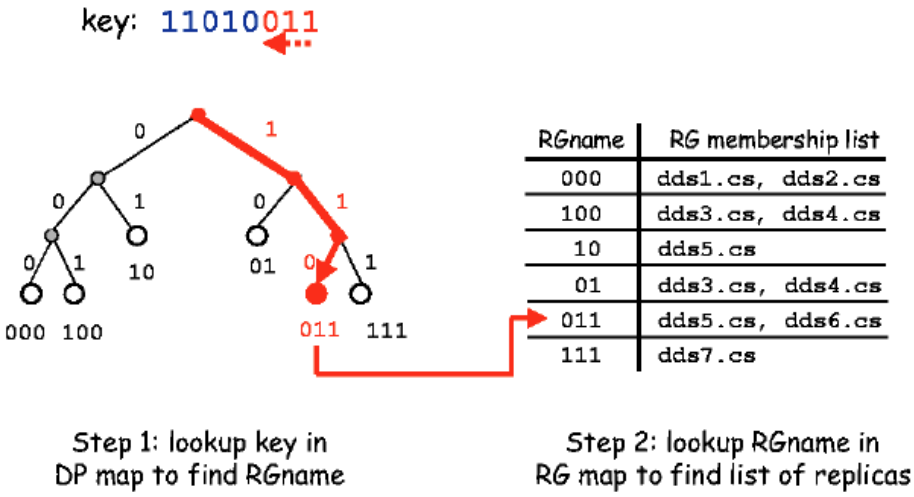
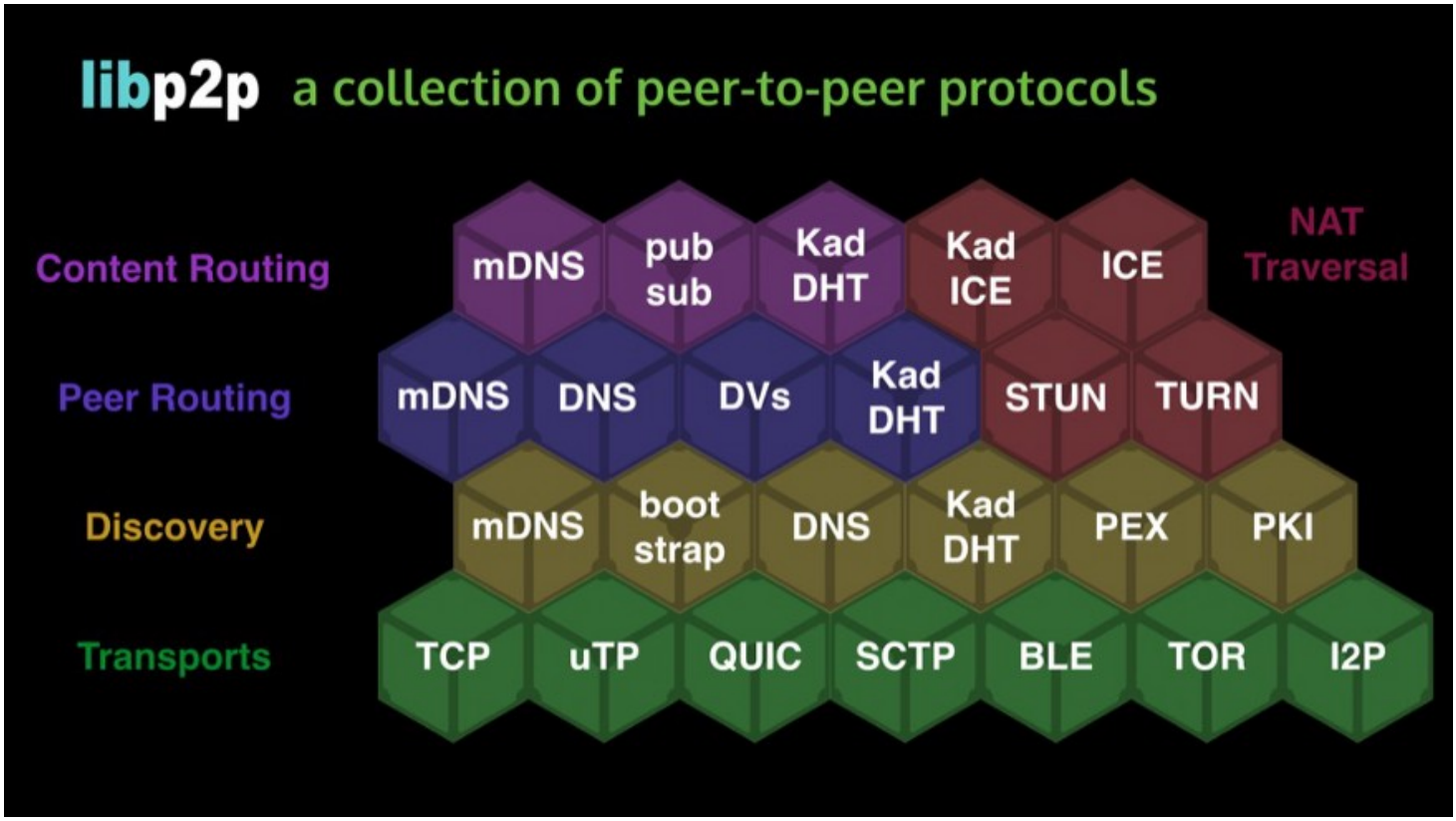


Figure: Distributed hash table metadata maps: this illustration highlights the steps taken to discover the set of replica groups which serve as the backing store for a specific hash table key. The key is used to traverse the DP map trie and retrieve the name of the key's replica group. The replica group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



- [x] Merkle DAGS

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mdns、content-routing等功能实现时使用
  > 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)
- 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.Protocol


   // 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)
```

- 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]