Peer Discovery in Ethereum

py-libp2p
(Robert Zajac, Alex Haynes, et. al)

Who are we

- Four University of Pennsylvania (UPenn) CS undergrads (4th year)
- py-libp2p team
  - Python client of the libp2p standard

Robert Zajac

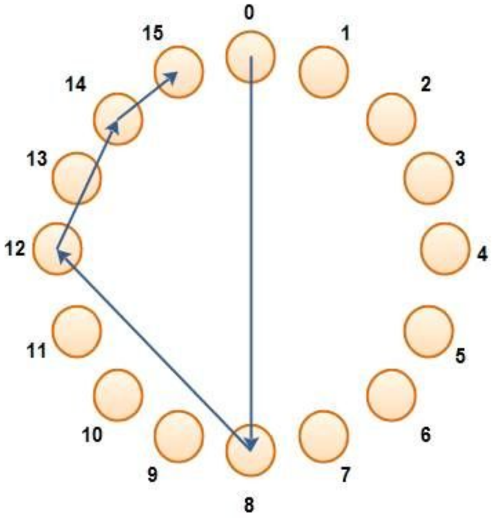
Alex Haynes

Aspyn Palatnick

ZX Zhang

Overview

1. State of peer discovery in Ethereum - ***discv4***
    a. Weaknesses & vulnerabilities
2. Current alternatives
    a. ***libp2p*** and its solutions (Kademlia DHT)
3. [Discussion] Peer discovery in Ethereum 2.0
    a. “discv5”
    b. Desirable properties & improvements
    c. Unifying libp2p + Eth 2.0

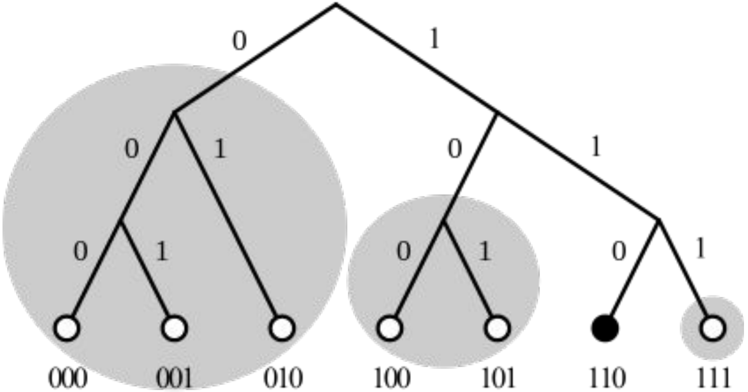


Disclaimer: information not canonical

Peer Discovery in Ethereum

discv4 Overview

- Discovery layer of *devp2p* (Ethereum networking stack)
  - Geth, Trinity implement discv4
- Kademlia-inspired DHT
  - Node ID: 64-byte elliptic curve public key
  - Kademlia ID (used in routing table): SHA256(node id)



discv4 RPCs (simplified)

1. *Ping*(**from**, **to**)
    a. Test connection to a peer. **from** + **to** contain ip/port details
2. *Pong*(**to**, **ping-hash**)
    a. Respond to *Ping*. Clients ignore *Pong* with invalid **ping-hash**.
3. *FindNode*(**id**)
    a. Ask for the *k* (*k* = 16) nodes closest to the node **id**.
4. *Neighbors*(**nodes**)
    a. Respond to *FindNode*. **nodes** = [(ip, port, node id)...]

All packets include *expiration timestamp* to prevent packet replay attacks

discv4 Details

- Initial peer discovery
  - Boot process queries *bootnodes* for closest neighbors (*FindNode*), repeats on result
- Routing table (Kad-compliant)
  - Consists of *k-buckets* (list of recently seen nodes)
  - For each $0 \leq i < 256$, node keeps a *k-bucket* of nodes at distance $(2^i, 2^{i+1})$
    - "Distance" = XOR of Kad IDs
  - *k-buckets* populated upon *FindNode*; only most recent *k* peers kept (time of last-successful *Ping/Pong* exchange)
- Bonding (endpoint proof)
  - Guarantee authenticity of incoming request
    - Avoid DoS attacks, where attacker spoofs *FindNode* from a peer and floods it with *Neighbors* response
  - Peers *Ping* each other and validate response (via hash) before issuing *FindNode*

discv4 Drawbacks

- Implementation difficulties around bonding; code complexity
- Bonding process is imprecise
    - Uncertain if *Pong* is processed, peer may consider other peer bonded when they are not
- Process for recovering full information about other nodes is not specified by discv4
    - Node IDs are found by looking at the public key used to sign UDP packets after the bonding process
- Multiple attacks (e.g. Eclipse) have been identified on Kademlia

Peer Discovery Alternatives

Peer discovery alternatives

- Kademlia DHT
    - Differences between Kademlia and discv4
        - Kademlia offers both finding nodes and storing/finding values (the latter is unused by discv4)
        - Discv4 additionally uses 64-byte public keys as node IDs in addition to having a Kad ID (SHA256(node ID))
- mDNS
    - Mostly used in for small/local networks
- Address Advertisement
- Many protocols implement custom peer discovery in-house (see BitTorrent)
- ...libp2p!

What is libp2p?

- *Not* devp2p
- Modular p2p networking library
- Original networking stack of IPFS
  - Backed by Protocol Labs
- Clients in Go and JS
  - Python (us), Rust, Java and Haskell coming soon

libp2p Peer discovery

- Modular design of library allows use of any module conforming to a “discovery interface”
  - Kademlia DHT (most common)
  - mDNS
  - “discv5” could easily be plugged in
- libp2p peer discovery solutions openly specified/documented
- py-libp2p developing a Kad DHT solution
  - Current goal: backwards compatibility with devp2p

Ethereum 2.0 Peer Discovery

Current work on “discv5”

- “discv5” is a misnomer; refers to next generation of peer discovery in Ethereum
- Geth upgraded to a “discv5” implementation as part of 1.5 release (2016)
    - Other clients on hold
- Specs are unofficial, and still in the works (why we’re here)

Future of peer discovery - *talking points*

Desirable Qualities of Spec

- Returning ENR Records?
  - *FindNodes* provide node metadata instead of simply address information
- Masking algorithm?
  - Avoid information leak via signed UDP packets
- Alternative to bonding process?
  - More precise, resilient to network

Integrating libp2p + Eth 2.0

- Leveraging rich libp2p specs + documentation?
- Code reuse between IPFS + Ethereum?
- Joint effort by communities?

Let’s discuss

Thank you!