Open Source HashGraph

Introduction

https://github.com/andrecronje/lachesis

cmd/lachesis	Upgraded dependencies; committing glide.lock; fixed some Go warnings	18 days ago
common	Merge branch 'blockchain' into dev	7 months ago
crypto	Lint application; replaced keyword-override variable; spelling	22 days ago
hashgraph	Merge branch 'master' of https://github.com/andrecronje/lachesis	11 days ago
net	SubmitTx Raw added	24 days ago
node	Upgraded dependencies; committing glide.lock; fixed some Go warnings	18 days ago
proxy	Lint application; replaced keyword-override variable; spelling	22 days ago
service	Update for headers	11 days ago
version	Stage 1 comparison between Babble HashGraph and Lachesis HashGraph im...	a month ago
.gitignore	Merge branch 'master' into martin-mobile	5 months ago
README.md	Stage 1 comparison between Babble HashGraph and Lachesis HashGraph im...	a month ago
glide.lock	Upgraded dependencies; committing glide.lock; fixed some Go warnings	18 days ago
glide.yaml	Upgraded dependencies; committing glide.lock; fixed some Go warnings	18 days ago
makefile	Stage 1 comparison between Babble HashGraph and Lachesis HashGraph im...	a month ago

Structure

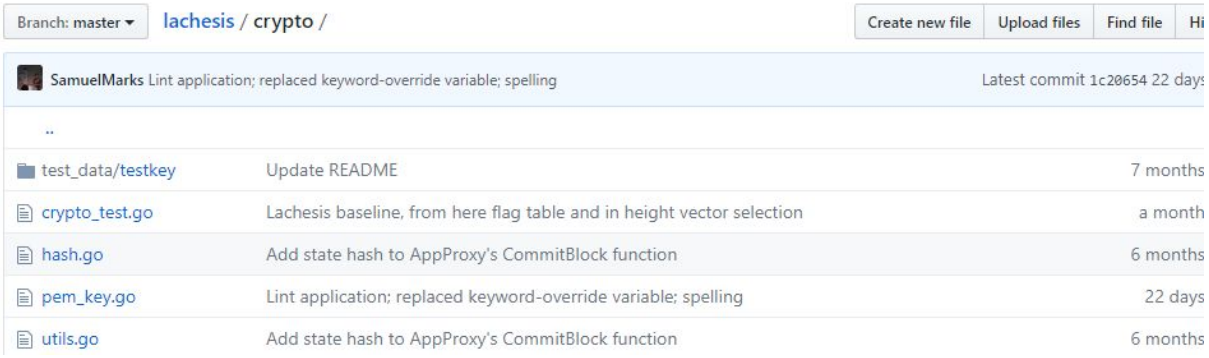
lachesis/cmd/lachesis/main.go is the entry point.

Branch: master ▾ lachesis / common / Create new file Upload files Find file Hi

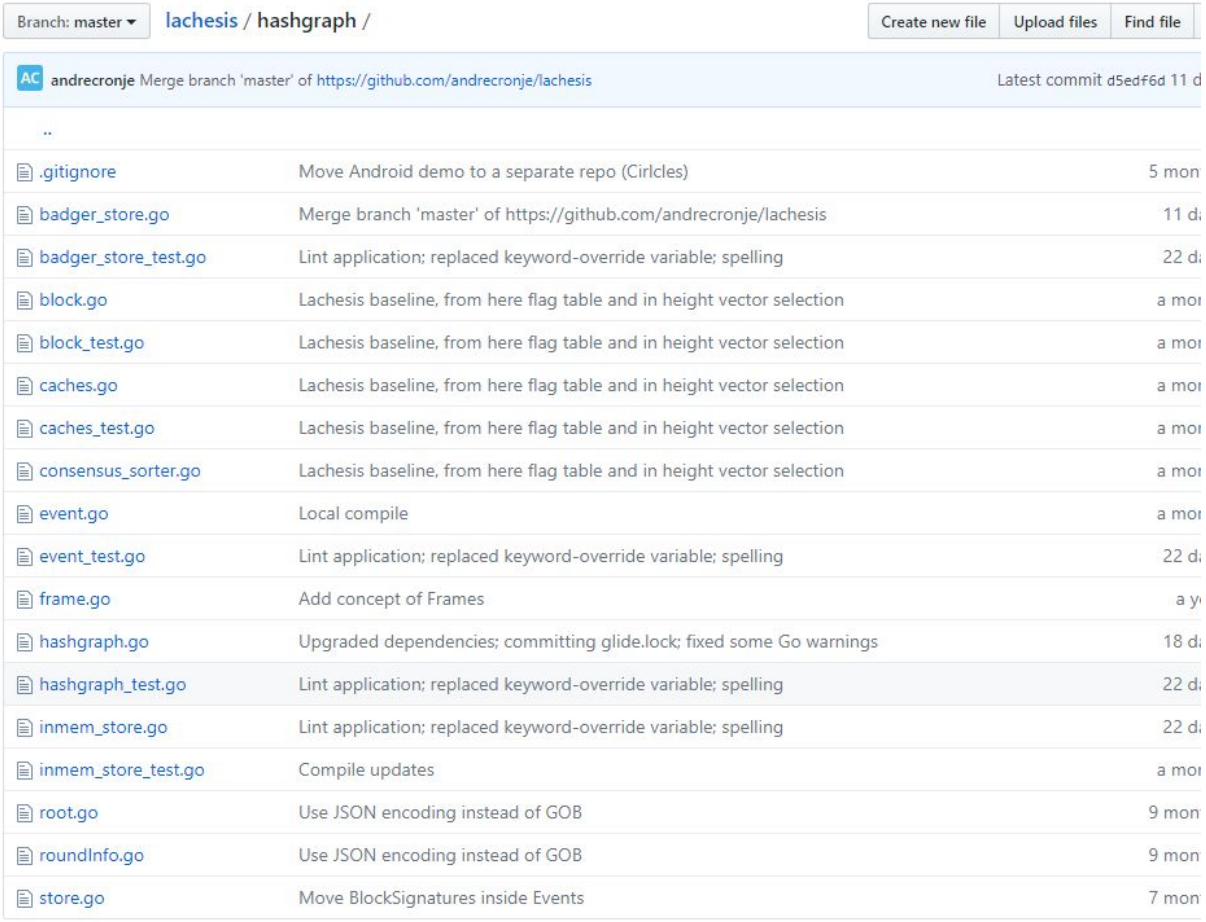
arrivets Merge branch 'blockchain' into dev Latest commit 81aad10 on Fe

..		
errors.go	Implement Block Signatures	7 months
lru.go	Remove license	a year
lru_test.go	Remove license	a year
rolling_index.go	Merge branch 'blockchain' into dev	7 months
rolling_index_map.go	Implement Block Signatures	7 months
rolling_index_test.go	Merge branch 'blockchain' into dev	7 months
test_logger.go	Update sirupsen and badger dependencies	9 months

/lachesis/common/ is shared memory cache implementations, don't worry about it.

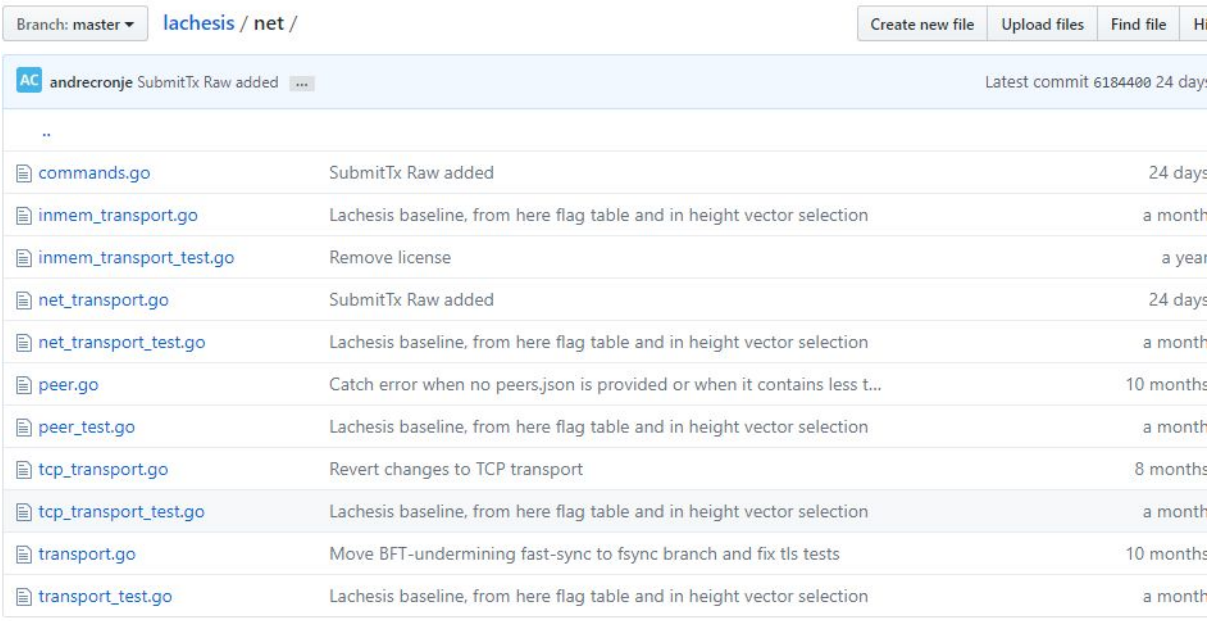


/lachesis/crypto/ is used for key generation, don't worry about it.

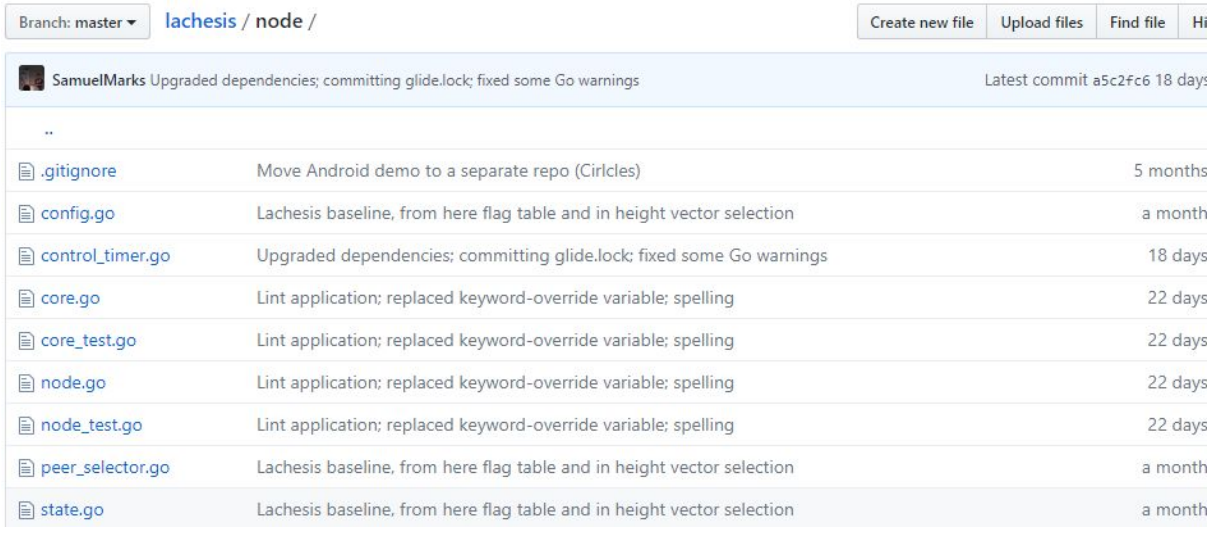


/lachesis/hashgraph/ is the hashgraph itself, few important pieces:

- block.go - the output after finality (2n/3 known) has been reached
- event.go - the graph event, defined by self signature and other parent
- hashgraph.go - the structure of events and gossip knowledge



/lachesis/net/ is used for p2p synchronization, don't worry about it.



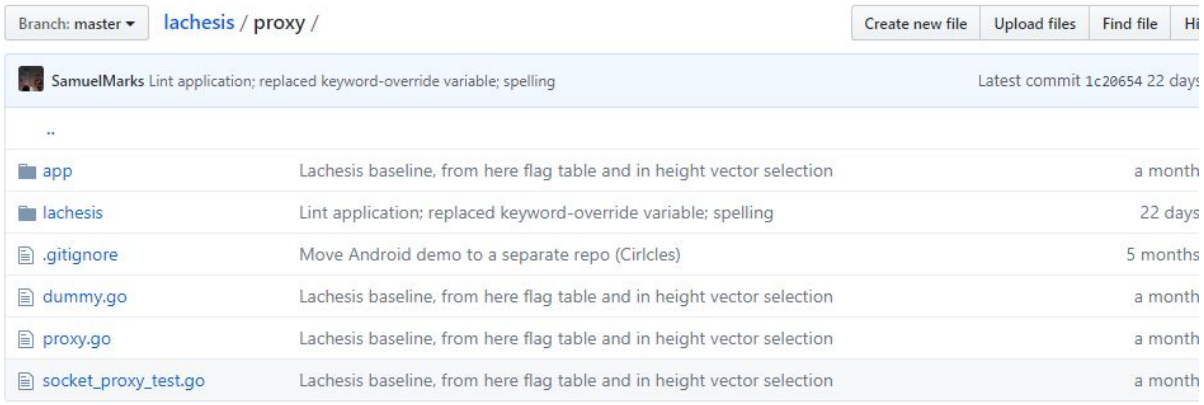
/lachesis/node/ is the core node software, this is instantiated by cmd (will do a trace further below). Only important part for now:

- node.go

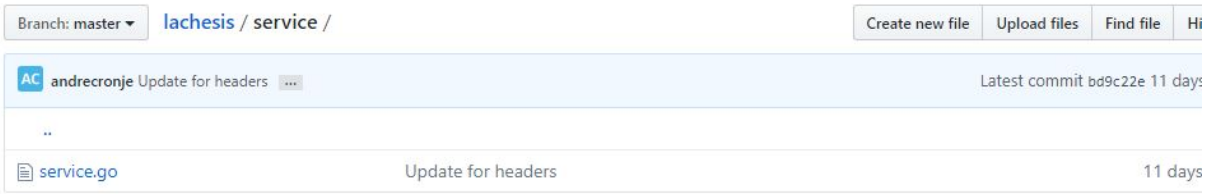
```go
618  func (n *Node) GetParticipants() (map[string]int, error) {
619          return n.core.hg.Store.Participants()
620  }
621
622  func (n *Node) GetEvent(event string) (hg.Event, error) {
623          return n.core.hg.Store.GetEvent(event)
624  }
625
626  func (n *Node) GetLastEventFrom(participant string) (string, bool, error) {
627          return n.core.hg.Store.LastEventFrom(participant)
628  }
629
630  func (n *Node) GetKnownEvents() map[int]int {
631          return n.core.hg.Store.KnownEvents()
632  }
633
634  func (n *Node) GetConsensusEvents() []string {
635          return n.core.hg.Store.ConsensusEvents()
636  }
637
638  func (n *Node) GetRound(roundIndex int) (hg.RoundInfo, error) {
639          return n.core.hg.Store.GetRound(roundIndex)
640  }
641
642  func (n *Node) GetLastRound() int {
643          return n.core.hg.Store.LastRound()
644  }
645
646  func (n *Node) GetRoundWitnesses(roundIndex int) []string {
647          return n.core.hg.Store.RoundWitnesses(roundIndex)
648  }
649
650  func (n *Node) GetRoundEvents(roundIndex int) int {
651          return n.core.hg.Store.RoundEvents(roundIndex)
652  }
653
654  func (n *Node) GetRoot(rootIndex string) (hg.Root, error) {
655          return n.core.hg.Store.GetRoot(rootIndex)
656  }
```

Getter functions for hashgraph data, quick explanation

- n = node
- n.core = core.go
- n.core.hg = hashgraph.go
- n.core.hg.Store = inmem or badger



/lachesis/proxy/ is important for when the EVM is coupled, for now can ignore.



/lachesis/service/ exposes http function for getters to retrieve data

```
28  func (s *Service) Serve() {
29          s.logger.WithField("bind_address", s.bindAddress).Debug("Service serving")
30          http.Handle("/stats", corsHandler(s.GetStats))
31          http.Handle("/participants/", corsHandler(s.GetParticipants))
32          http.Handle("/event/", corsHandler(s.GetEvent))
33          http.Handle("/lasteventfrom/", corsHandler(s.GetLastEventFrom))
34          http.Handle("/events/", corsHandler(s.GetKnownEvents))
35          http.Handle("/consensusevents/", corsHandler(s.GetConsensusEvents))
36          http.Handle("/round/", corsHandler(s.GetRound))
37          http.Handle("/lastround/", corsHandler(s.GetLastRound))
38          http.Handle("/roundwitnesses/", corsHandler(s.GetRoundWitnesses))
39          http.Handle("/roundevents/", corsHandler(s.GetRoundEvents))
40          http.Handle("/root/", corsHandler(s.GetRoot))
41          http.Handle("/block/", corsHandler(s.GetBlock))
42          err := http.ListenAndServe(s.bindAddress, nil)
43          if err != nil {
44                  s.logger.WithField("error", err).Error("Service failed")
45          }
46  }
```

Serve() creates the routes for each getter. Calling http://node_ip/stats will call service.go GetStats

```
69  func (s *Service) GetStats(w http.ResponseWriter, r *http.Request) {
70          stats := s.node.GetStats()
71
72          w.Header().Set("Content-Type", "application/json")
73          json.NewEncoder(w).Encode(stats)
74  }
```

Which is simply a proxy for calling s.node.GetStats (from node.go)

Trace startup

/lachesis/cmd/lachesis/main.go -> run

```
155  func run(c *cli.Context) error {
```

```
188         conf := node.NewConfig(time.Duration(heartbeat)*time.Millisecond,
189                 time.Duration(tcpTimeout)*time.Millisecond,
190                 cacheSize, syncLimit, storeType, storePath, logger)
191
192         // Create the PEM key
193         pemKey := crypto.NewPemKey(datadir)
194
195         // Try a read
196         key, err := pemKey.ReadKey()
197         if err != nil {
198                 return cli.NewExitError(err, 1)
199         }
200
201         // Create the peer store
202         peerStore := net.NewJSONPeers(datadir)
203
204         // Try a read
205         peers, err := peerStore.Peers()
206         if err != nil {
207                 return cli.NewExitError(err, 1)
208         }
209
210         // There should be at least two peers
211         if len(peers) < 2 {
212                 return cli.NewExitError("peers.json should define at least two peers", 1)
213         }
214
215         //Sort peers by public key and assign them an int ID
216         //Every participant in the network will run this and assign the same IDs
217         sort.Sort(net.ByPubKey(peers))
218         pmap := make(map[string]int)
219         for i, p := range peers {
220                 pmap[p.PubKeyHex] = i
221         }
```

```
287             node_.Run(true)
```

That easy.

Examples

http://ip:port/stats
{"consensus_events":"3","consensus_transactions":"0","events_per_second":"0.00","id":"0","last_block_index":"-1","last_consensus_round":"1","num_peers":"2","round_events":"3","rounds_per_second":"0.00","state":"Gossiping","sync_rate":"1.00","transaction_pool":"0","undetermined_events":"8"
http://ip:port/block/0
http://ip:port/events/
{"0":5,"1":4}
http://ip:port/consensusevents/
["0xA2A41F7900AFC498B7A63A41FECD27891906AA5A7B3F1DA1C93FF3516714233D","0x4B94971926D41CE6FF09D4250C115033E9E515BA261387EBAC5F071A899E6E39","0x30A36D9AA6B9EAEC0A315B989BB479C71A9B6528EB61876B3CADAA836E1EB6AC"]
http://ip:port/participants/
{"0x0499D1A9BD84FDA510FA9F41D5DC45566DB3FD38504ABC0AFBE068471BA9E82CB3D20BE76A7F6A6A6246BAE878FCC25C1CFFCDBB323F86E5FAF8459E2E1A9F8336":0,"0x04EC501B7E7FC9682FDAA4EA23E965CAA8BCC98CBCE5F08DEDDF69140E17B63DC184693F214C0ACF4460D5DC43B0568B57A22D5D0E89E46D0A28DE0B25D9662001":1}

http://ip:port/event/0x4B94971926D41CE6FF09D4250C115033E9E515BA261387EBAC5F071A899E6E39
{"Body":{"Transactions":[],"Parents":["0x30A36D9AA6B9EAEC0A315B989BB479C71A9B6528EB61876B3CADAA836E1EB6AC","0xA2A41F7900AFC498B7A63A41FECD27891906AA5A7B3F1DA1C93FF3516714233D"],"Creator":"BJnRqb2E/aUQ+p9B1dxFVm2z/ThQSrwK++BoRxup6Cyz0gvnan9qamJGuuh4/MJcHP/NuzI/huX6+EWeLhqfgzY=","Timestamp":"2018-08-10T06:54:32.237819411Z","Index":1,"BlockSignatures":[]},"Signature":"1ke1suw7c17vkm2h6x3g4wgd0q79dascoimxco0tnep4t4m19p|4f1ggc5y89ewi8nc5cv2foof84qaqq9s05n0pxee6374wu5dos"}

http://ip:port/roundwitnesses/1
["0xCE01A083FC99A6E328D5A1C1D19CBBEB8529088F8F8ED1A2AD4D24882C9D8E34","0x6E584D0EE474217A1516897B85DDE158EACC84B37400FD15AE38ECFC643D81C0"]

http://ip:port/roundevents/0
3

Startup

Step 1 Installing Go (Ubuntu)

sudo apt-get update
sudo apt-get -y upgrade
sudo curl -O https://storage.googleapis.com/golang/go1.9.1.linux-amd64.tar.gz
sudo tar -xvf go1.9.1.linux-amd64.tar.gz
sudo mv go /usr/local
sudo nano ~/.profile
export PATH=$PATH:/usr/local/go/bin
source ~/.profile

Step 2 Clone Repo

```
mkdir -p $GOPATH/src/github.com/andrecronje/
cd $GOPATH/src/github.com/andrecronje
git clone https://github.com/andrecronje/lachesis.git
cd $GOPATH/src/github.com/andrecronje/lachesis
curl https://glide.sh/get | sh
glide install
```

Step 3 (Optional, hard mode)

```
sudo apt-get update
sudo apt-get -y upgrade
sudo curl -O https://storage.googleapis.com/golang/go1.9.1.linux-amd64.tar.gz
sudo tar -xvf go1.9.1.linux-amd64.tar.gz
sudo mv go /usr/local
sudo nano ~/.profile
export PATH=$PATH:/usr/local/go/bin
source ~/.profile
mkdir $HOME/work
export GOPATH=$HOME/work
mkdir -p $HOME/work/src/github.com/user/
cd $HOME/work/src/github.com/user/
git clone https://github.com/andrecronje/lachesis.git
apt-get install -y build-essential

#Lachesis
go get github.com/dgraph-io/badger
go get github.com/sirupsen/logrus
go get gopkg.in/urfave/cli.v1

make build

#Lachesis
./build/lachesis keygen
mkdir -p /root/.lachesis/
vi /root/.lachesis/priv_key.pem
vi /root/.lachesis/peers.json

[
  {
        "NetAddr":"18.191.184.199:12000",

"PubKeyHex":"0x0448C9212873C76DE0086DA680F9329735C40674B5FA05105886548B217273B0AFA0
2D73157F4D96DACFE1D9E44DBB2608F5C60D037743DB18567B82D077CBAE40"
  },
```

```
  {
      "NetAddr":"52.14.2.237:12000",

"PubKeyHex":"0x04C503A238046D9095B548E61939CA58BE926C6809F7205CD2D671A88C4E8369754
ADE343FBB4FBE9A46118EB549753B76B18243369E0475A319989F06879CFE19"
  },
  {
      "NetAddr":"18.222.125.244:12000",

"PubKeyHex":"0x046A584F2FBDF61A36B8D30461C9285B817F60271D41B7B7EC445EF73DD2B363B1F
782D5FE96D3D08B71750F4D07CC137AE7AD9A95574791737E6E407880015B3A"
  }
]
```


Running the node

The default data dir is currently;

$HOME/.lachesis/

In this folder it expects two files;

priv_key.pem
peers.json

peers.json is the current node list (completely permissioned system currently, it is defined as per below, the PubKeyHex is the public key (with 0x) that corresponds to the private key found in priv_key.pem


```
[
  {
      "NetAddr":"18.191.184.199:12000",

"PubKeyHex":"0x0448C9212873C76DE0086DA680F9329735C40674B5FA05105886548B217273B0AFA0
2D73157F4D96DACFE1D9E44DBB2608F5C60D037743DB18567B82D077CBAE40"
  },
  {
      "NetAddr":"52.14.2.237:12000",

"PubKeyHex":"0x04C503A238046D9095B548E61939CA58BE926C6809F7205CD2D671A88C4E8369754
ADE343FBB4FBE9A46118EB549753B76B18243369E0475A319989F06879CFE19"
  },
  {
      "NetAddr":"18.222.125.244:12000",

"PubKeyHex":"0x046A584F2FBDF61A36B8D30461C9285B817F60271D41B7B7EC445EF73DD2B363B1F
782D5FE96D3D08B71750F4D07CC137AE7AD9A95574791737E6E407880015B3A"
```

```
    }
]
```

To run the nodes you execute;

```
#service node
./build/lachesis run -node_addr="ip:port" -service_addr="ip:port"

#proxy node
 ./build/lachesis run -node_addr="ip:port" -proxy_addr="ip:port" -client_addr="ip:port"
```

Example Commands

```
#service node
./build/lachesis run -node_addr="172.31.17.156:12000" -service_addr="172.31.17.156:8000"

#proxy node
 ./build/lachesis run -node_addr="172.31.32.51:12000" -proxy_addr="172.31.32.51:9000"
-client_addr="18.221.128.6:9000"
```

You can subscribe to service_addr for http requests, so in above example
http://172.31.17.156:8000/stats