# StreamNet-Go 概要设计文档

|  |  |  |
| --- | --- | --- |
| **版本** | **时间** | **修改** |
| V0.0.1 | 2020年6月4日 | 初稿 |
|  |  |  |
|  |  |  |

## 一、需求简介

我们这一版实现的StreamNet在功能上与Java版本的StreamNet保持一致，区别是使用Go语言实现，并且是要自己搭建整个平台，去除冗余代码。

也就是说，我们这一版的目标是自主实现StreamNet应用，包括所有的模块。其中的重点是：实现一个go版本的DAG结构，并且实现go版本的TipSelection算法、StreamNet-Conflux算法。

之前设想的账户模型、经济模型、防女巫攻击机制、ABCI等，放在下一个版本继续考虑。一个版本中做太多修改，难度会较大，在实践中也难以开展工作。

我们在自主开发的同时也有所参考，如go语言实现的tendermint、ethereum、fabric等。这样子或许能够降低我们的开发难度，提高我们的设计和开发水平。

其中重点是参考tendermint，因为后续我们有可能会参考ABCI这套框架来实现搭建在StreamNet之上的应用。

总结来说，这一个版本的目标是实现完整的DAG功能。

在完成工作的过程中，积攒Go语言的使用经验，积攒对tendermint 的理解，思考新功能的加入，为下一步更加完善的功能做好准备。

## 二、流程设计

整体流程如下所示：



### 1.命令行参数解析

main函数里面首先进行命令行参数解析。目前暂时有2个子命令：init 和 start：

init负责生成默认的目录结构和配置；

start启动http server，能够接收rpc请求，包括发送交易和查询；并启动genesis forward线程，用来监控DAG的状态。



### 2.RPC处理

Rpc接收请求之后，会进行对应的处理，目前的RPC请求只有2个：AddNode和QueryNodes。分别是用来存储投票信息和查询排名。



**这里注意一点：**

在Java 版的StreamNet中，PageRank模块是单独出来的，我们这一版的实现中，期望将PageRank模块融合进来，在QueryNodes返回的结果中，将已经PageRank过的内容返回。

* 1. AddNode

流程如下所示



AddNode 会根据当前DAG的状态进行TipSelection操作，然后计算POW形成Nonce，将这些内容形成一条Transaction，然后进行Store和Broadcast。

* 1. QueryNodes

流程图如下所示。

QueryNodes根据当前的DAG，经过Conflux算法得到一条链，再经过PageRank算法得到排名的结果。



### 3.Genesis Forward

流程图如下所示：



Genesis Forward会启动一个线程，或者 go routine，会将DAG的早期历史节点固化成链。它会清减DAG的内存状态，也会将固化的链的信息存储到数据库中。

### 4.网络广播和同步

目前我们的gossip广播机制是：如果是本节点从RPC收到的交易，会将完整的内容进行广播；如果是从网络中获得的交易，会将交易的Hash进行广播，然后依据Hash进行请求和响应。

节点之间会同步状态，当自己发现没有相应的交易时，会像邻居节点请求；当收到邻居节点的请求时，会从数据库中读出相应的内容进行返回，如果自己也没有相应的内容，会继续发行请求。

请求是点对点的。如果你收到了一条交易的Hash，发现本身没有这条交易，就向发送者请求这条交易。

如果接收到请求，发现没有这条交易内容，会向自己的所有的邻居发送请求。

## 三、模块设计

### 命令行解析模块

代码目录：main.go 和 commands目录。

命令行参数解析有很多种方法，go语言标准库里就有os.Args和 flag库，另外还有第三方的cli和cobra等等。

Tendermint和Fabric等都是采用了spf13/cobra。我们这里也使用cobra的框架。

* 1. init

init负责生成基本目录结构和配置，默认在~/.sng目录下生成config目录和data目录。data用于保存数据库，config目录下生成config.toml和genesis.json文件。

可以参考tendermint的实现。

* 1. start

start用于启动一个StreamNet节点。最重要的就是解析配置、解析genesis文件、初始化或者读取数据库、跟网络节点同步、启动http server服务器、启动genesis forward线程、初始化DAG内存中的结构等。

这里有几个问题需要在实现的时候考虑：

1. http server 的选择
2. 这些服务的启动顺序
3. 锁的问题：在启动http服务器的过程中是否需要阻止RPC请求的处理

参考tendermint的实现。

### 配置模块

代码目录：config目录和commands目录。

配置文件的解析有固定的处理方式，tendermint是配合spf13/viper和spf13/cobra来进行解析的。

这个地方也参照tendermint的实现。

### 日志模块

代码目录：libs/log目录

形成统一的日志输出样式。

参照 tendermint的实现。

### 加密模块

代码目录：crypto目录

1. Hash

交易等都需要一个唯一的hash值来指代。这里暂用tendermint的hash方式。Tendermint里面使用的是[]byte字节切片的方式。

### Dag模块

代码目录：dag目录

这里面是最核心也是最复杂的数据结构DAG的实现，里面有各种graph关系的描述。

需要参照StreamNet里面的实现。

### Tipselction算法模块

代码目录：tipselection

这里需要实现TipSelector接口。里面只有一个方法：getTransactionsToApprove

参考StreamNet的实现。

### Conflux算法模块

代码目录：conflux

需要实现conflux接口。

参考StreamNet-java的实现。

### PageRank模块

之前已经实现了PageRank模块，现在将它融合到这里来。

### Storage模块

代码目录：store。

由于StreamNet-java使用了RocksDB并且使用了comumn family 的特性，所以我们这里也使用RocksDB。

存储模型也需要参考StreamNet的实现。

### Network模块

代码目录：network目录

这里需要实现peer接口。

参考StreamNet-java 和 Tendermint 的实现。

### Genesis forward 模块

代码目录：forward目录

参考StreamNet-java的实现。

## 四、数据结构和接口

我们在实现上参考tendermint。为了方便起见，我们StreamNet-Go生成的二进制文件名为sng，取3个首字母。下面统一以sng来表示。

### Config

下面是目录路径：



下面是具体的配置：



RPCConfig 指定RPC监听地址，P2PConfig指定节点间通信端口和邻居节点的信息，ConsensusConfig指定了 pow 的困难度。

### Hash



### DAG

最大的数据结构就是整个DAG结构，里面包含了各个关系的graph，以及score、degree、level等。



### Transaction



### TipSelection



### Store

存储的接口如下：

RocksDB的实现如下：



### Network

