btc-eclipse-sim

bhangra

2015/1/29

2月3月中のTODO

- 斉藤さんに以下の点を相談:
  - シミュレータのデザインに関して相談
  - 攻撃者が存在しないケースでもネットワークが個々のシードを起点に分断してしまう点
  - 実際のBitcoinネットワークを再度調べる手法
- osamuさんから賜った訓示なのですが
  - 3月中には何らかのデータが取れている事
- なのでそれに向けて
  - シミュレータの改善
  - 現実のネットワークの調査
  - それに則したシード等の挙動の実装

本日の発表の趣旨

- 本日は、半期卒業が伸びてしまったので・・・
- これを期にシミュレータのパフォーマンスとスケーラビリティを上げるために
- ちょっと現状のシミュレータのデザインというか実装に関してざっと説明させて頂いて
  - どこが問題か
  - どうデザインするのがいいのか
- プログラミングやデザインの経験が乏しいので皆様からコメント頂ければ幸いと存じます。

https://github.com/bhangra/btc-eclipse-sim/

ノード間のリンク　connection.h

```
struct links{
    unsigned int          group;
    unsigned int        miner_id;
    unsigned int          subnet;
    unsigned int          n_time;
    struct link        *new_comer;
    struct link           *link;
    struct links          *prev;
    struct links          *next;
};
```

```
struct link{
  pthread_mutex_t    rcv_mutex;
    unsigned int        num_msg
    unsigned int         read_pos;
     unsigned int        write_pos;
     unsigned char   buf[BUF_SIZE];
     unsigned char  sbuf[BUF_SIZE];
     unsigned char process_buf[BUF_SIZE];
     struct link        *dest;
     bool             fgetblock;
};
```

ノード間の通信 connection.c

- send_msg
- read_msg
- process_new
- process_msg

- ノード間の通信は、send_msg関数とread_msg関数により行われる
- 先ず送信者は自身の持つ link->sbuf に送信内容を作成し、send_msg関数で通信対象*dest の buf にmemcpy する
- 受信者は、num_msg でメッセージの存在を確認し、read_msg 関数で process_buf に memcpy し、ルーチンから呼ばれる
- process_msg か、process_new関数に渡す

メッセージ送受信の現状の問題

- sbuf　　送信内容作成バッファ
- buf　　受信バッファ
- process_buf　受信後の処理バッファ

と**3つのバッファに同じ内容**を**memcpy**しており、

- また3つのバッファが**メモリを食っている**ので

- どうメッセージか通信のシミュレートをデザインするのがいいのかご相談させて下さい。。。

リンクと通信の問題

現状の実装では:

- 全てのノード間の通信は、メッセージ数、サイズに**関係なく1秒(1ターン)**に行われている

- **通信の遅延**などもblockの**伝搬遅延**、そして**blockchain fork**に影響すると明らかになっているので反映したいが**デザインが思い浮かばない・・・**

一つ思いついた通信のデザイン

- ノードは未読メッセージのリスト構造体へのポインタを保持
- 各々の未読メッセージリスト構造体は当該メッセージの通信遅延値を内包し、毎回ルーチンで当該メッセージが処理出来る時間になったかをデクリメントしつつ確認
- このメッセージリスト構造体はペイロードへのポインタを持つ

blockに関して block.h block.c

```
struct block { //現状同じ内容の
    blockを個々にmallocしている・・
unsigned char
    hash[SHA256_DIGEST_LENGTH];

    unsigned int    height;
    double          time;
    unsigned int    miner_id;
    unsigned int    size;
    unsigned int    valid;
};

struct blocks{
    struct blocks   *prev;
    struct block    *block;
    struct blocks   *next;
};
```

- 現状全てのblockは同じサイズだけど、blockのサイズも伝搬遅延とblockchain forkに影響するようなので改変が必要、だけどデザインが思い浮かばない。。。
- 現在 block の連なりをSHA256ハッシュ関数で表しているが、パフォーマンス上よろしくないのは明らかなので、前のblockの採掘者のIDで判定を行うよう変更
- 現状全てのノードは同じ内容のblockをそれぞれ個々にmallocしてるのでそれは改変する予定

既知ノードリスト

既知ノードリストの実装

シミュレーションでは既知ノードリスト、どのノードがどう接続対象を選ぶかに関して公式実装の挙動をシミュレートする事に関して厳密性が必要なので

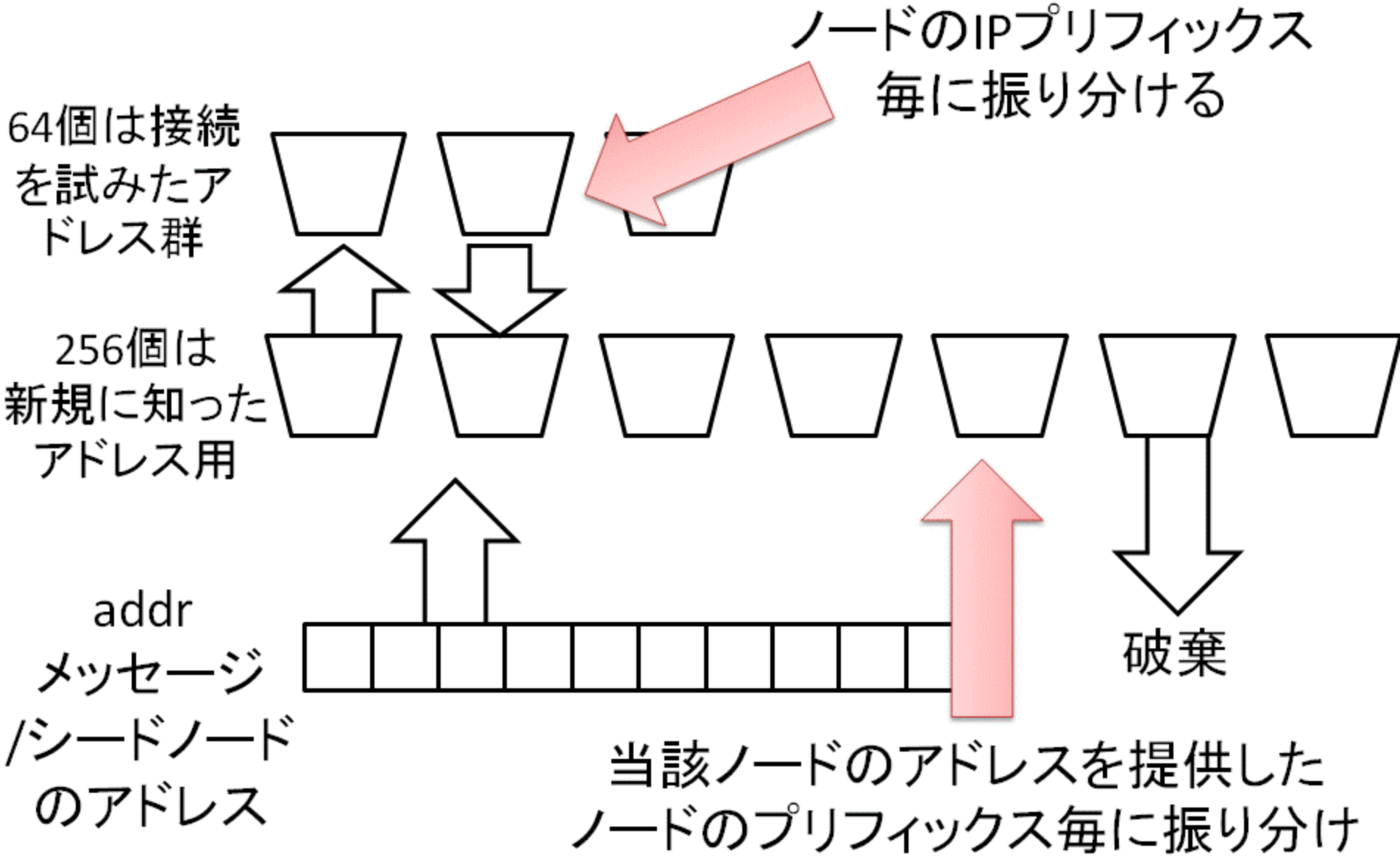
- https://github.com/theuni/bitcoin/blob/master/src/addrman.h
- https://github.com/theuni/bitcoin/blob/master/src/addrman.cpp

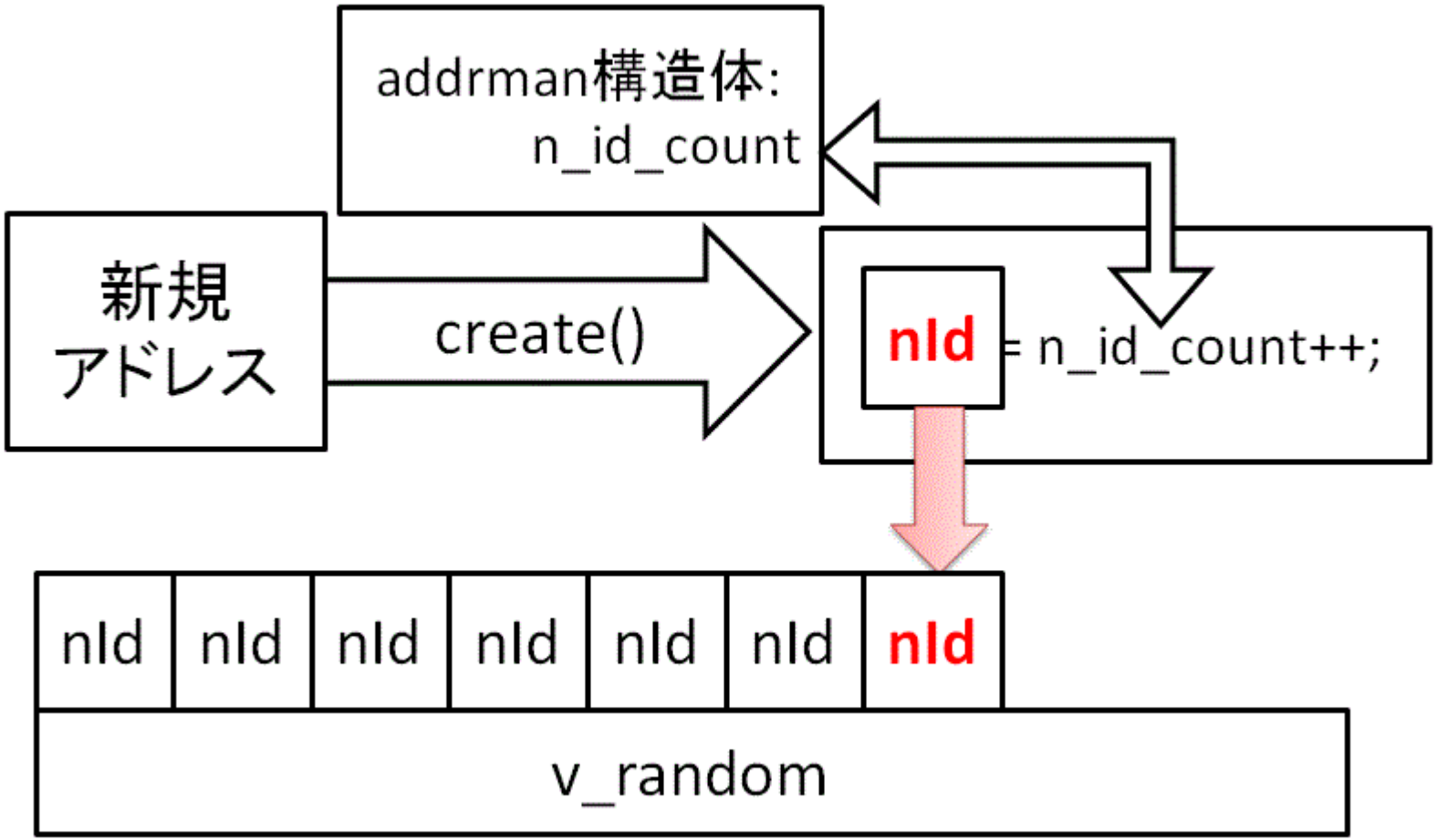
自身の既知ノードリストの実装

- 公式実装をc言語に書き換えつつ、ほぼ写経に近い形で実装しております
- https://github.com/bhangra/btc-eclipse-sim/blob/master/addrman.h
- https://github.com/bhangra/btc-eclipse-sim/blob/master/addrman.c

```
struct addrman{
    char                *n_key;
    unsigned int        n_tries;
    struct caddrinfo    *caddrinfo;
    unsigned int
    v_random[256+64][64];
    unsigned int        v_random_size;
    unsigned int        n_id_count;
    unsigned int        n_tried;
    unsigned int        n_new;
    unsigned int        vv_new[256][64];
    unsigned int        vv_tried[64][64];
};
```

```
struct caddrinfo{
    struct caddrinfo   *next;
    struct caddrinfo   *prev;
    struct link                 *new_comer;
    unsigned int                subnet;
    struct link                 *source; //will fix
    unsigned int                miner_id;
    unsigned int                nid;
    unsigned int                n_last_success;
    unsigned int                n_attempts;
    unsigned int                n_ref_count;
    bool                                f_in_tried;
    unsigned int                n_random_pos;
    unsigned int                n_last_try;
    unsigned int                n_time;
};
```


ノードのIPプリフィックス
毎に振り分ける
64個は接続
を試みたア
ドレス群
256個は
新規に知った
アドレス用
addr
メッセージ
/シードノード
のアドレス
破棄
当該ノードのアドレスを提供した
ノードのプリフィックス毎に振り分け


addrman構造体:
n_id_count
新規
アドレス
create()
nId = n_id_count++;
nId
nId
nId
nId
nId
nId
nId
v_random

既知ノードリストの実装の問題

- デタラメなアドレスを多数送りつけるという攻撃手法を試している時に**シミュレータの挙動が著しく低速化**して発露したのですが

おそらく原因としては:

- 個々のノードのアドレスのエントリーがリスト構造体となっており**管理に使われてるバケツにある個々のエントリを表すnId値**から当該エントリをリストから洗い出すのに時間がかかる事だと思われます
- nId値を整数では無く、**直接当該エントリへのポインタ**にする事でパフォーマンスの改善を図ろうかと思います。