DHT へ、愛を込めて。

58th isa

I. 序

世間の人間というのは、それは人であったり物であったりあるいは形而上学的なものであったりはしますが、とにかく何らかのものを、常に探し求めているものなのだと、大昔に読んだ本に書いてありました。

ボールを追いかける少年、大きい夢を追いかける青年、女の尻を追いかける中年。これらはどれも、自分を見つめながら生きる人間の何とも言えない面白さを感じさせてくれる、とても素晴らしいものですね。

でも僕はロマンチストでは無いので、人々がそれぞれ探し求めている具体的な物や、あるいは人々が知らず知らずのうちに探求に駆られている理由などといったことには、残念ながら、全く興味がありません。

それよりも僕はプログラマーですので、人々がどのような手法を使って、目的物に辿り着こうとしているのかという事にとても興味があるわけでして、それを追求しようとするのも当然のことであるわけです。

ということでこれから、P2P ネットワークという世界の上で人々が各々望む情報を手に入れる事ができるためにどのような仕組みが働いているのかという事を、DHT という最新の情報科学の分野から解説します。

II. どうやったらお話、できるかな？ － ルーティング入門 －

DHT を学ぶ前に、あるノードに辿りつくための手法＝ルーティングについて、研究してみましょう。
さて、突然ですが、以下のルールが成立するような、不思議な街を想像してみてください。

① 住民は全員対人恐怖症で、直接会話は行わない。代わりに電話は同時に数人と行える。
② 住民は通信手段として無料の携帯電話を一人一台所有している。その他の遠隔通信手段はない。
③ 携帯電話の電話帳には、1000 人までの電話番号を記録できる。
④ 条例によって、電話帳外に電話番号を記録することは禁止されている。
⑤ 住民は、携帯の電源を夜間など応答できない時には切る。再起動すると電話番号が変わる。
⑥ 入居時及び電話帳の中身が 0 の時に限り、役所にランダムに選ばれた住民数人の電話番号を聞ける。
⑦ 住民の数は最低 1000 人で上限はない。常に緩やかに増加方向で変化している。街を抜ける人もいる。
⑧ 住民同士は互いに友人関係を作らない。
⑨ 街の真ん中には掲示板があって、それを介しても連絡が出来る。ただし、小さい。
⑩ 住民全員が、住民全員分の名前リストを保有している。

この街で、どの人がどの別の人に対して電話をしたいと考えた時でも、お互いに携帯電話の電源さえ ON になっている状況であれば電話が可能である、という状況にするにはどのようにすれば良いのでしょうか？

III. 超集中☆ルーティング

さて、前頁の問題に対しての答えですが、これにはまず、次のような極論が出てくると思います。

A1:「掲示板に住民全員が、自分の名前と自分の現在の電話番号の対応表を書く。」

これは全くもって単純な発想による回答ですが、案外馬鹿にしてはいけません。掲示板の前が混雑するなどといった問題が発生しないのであれば、これは間違いなく一番手っ取り早く分かりやすい手段です。

ですが、ここでルールを良く読み直してみるとルール⑦に、住民の数は 1000～1000 万と書いてあるのが分かりますね。利用者の規模が数十人とか数百人程度であれば、掲示板を使うというのは良いアイデアです。しかし、その規模が千とかウン万とかなったら全く話が変わってきます。ルール⑨に掲示版は小さいと規定してありますが、その小さい掲示板の前に千とか万単位で人が集まったら、一体どうなってしまうことでしょうか。

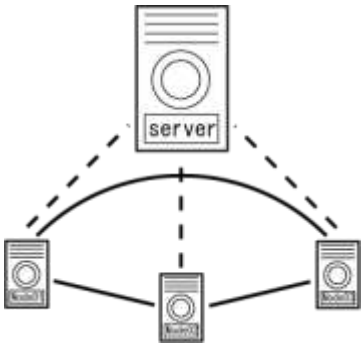
結果は、予想できますね。大混雑が起こって、かえって必要な情報を見るまでに大きい時間がかかります。

A1'： P2P ネットワークとしての解釈

A1 は混雑したり時間がかかったりして全くダメダメだと言ったばかりですが、実は多くの P2P プロトコルでこれと非常に良く似たルーティング手法が採用され、実用化されています。

この手法の利点は何かというと、掲示板の管理者がノードリストを自由に管理できるというところで、つまりこれは、P2P の利点（http://paken.s1.hayasoft.com/files/down/denno2006_p2p.pdf 参照）を活かしつつも、サーバー側による相当なマネージメントが可能な手法である、という事なのです。

これはハイブリッド P2P などと呼ばれる P2P の種類の一つで、実用化の例としては、ファイル共有プログラムの Napste が典型的です。Napster プロトコル上では、ノードは欲しいファイルの名前などをサーバーに送信し、サーバーは保有しているデータおよびその所有者の一覧から、クエリに一致するものをリストにして送り返します。返信を受けたノードは、リストの中から欲しいものを選んで、そこに書いてあるノードに対してアクセスを行い、実データの転送を行います。一般化すれば、メタデータおよびノードテーブルの扱いだけをサーバーに依存し、コンテンツのやり取りはサーバーを介さない P2P ネットワークと言えます。



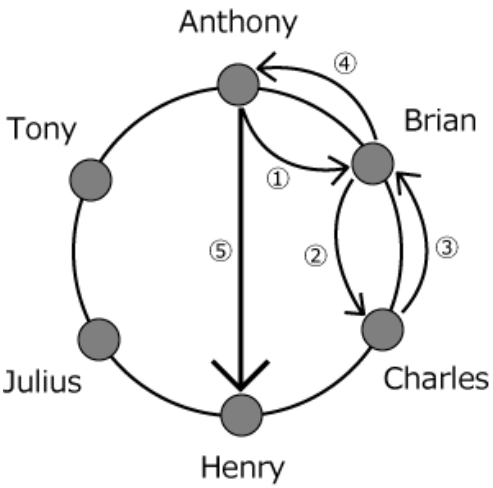
これらも非常に面白いテーマなのですが、しかし、今回のテーマは DHT という事で今後はピュア P2P、あるいはルーティング処理に於いてサーバーを使わないタイプの P2P だけを見ていきたいと思います。

IV. 超分散☆ルーティング

掲示板だけに依存した時に発生した問題を解決するために重要なポイントは、誰かの電話番号を知るために、別の誰かに電話をかけて聞くという手があるという事です。これをまた極端に利用して、一箇所に集まる負荷の大きさを最大限に分散させる手を考えると、次のようなものが思いつきます。

A2:「全員が名前順で自分の前後の人の名前と電話番号を知っておく」

これはどういう事かというと、全員が名前順で円形に並んでいると考えてください。そのある位置に *Anthony* さんがいます。彼の前には *Tony* さんが、彼の後ろには *Brian* さんがいて、二人の電話番号は既知です。



さて、ここで *Anthony* さんが *Henry* さんと電話をしたいとします。この時まず *Anthony* さんは *Brian* さんに、「*Henry* の電話番号を知っていますか？」と聞きます。しかし *Brian* さんは *Henry* の電話番号を知らないので、自分の次の *Charles* さんに聞かれたのと同じ質問をします。*Charles* さんの次は *Henry* さんなので、*Charles* さんは *Henry* さんの電話番号を知っており、これを *Brian* さんに教えます。*Brian* さんは、教えられた電話番号を *Anthony* さんに伝えます。こうして、*Anthony* さんは *Henry* さんの電話番号を知ることができました。

[Ant -> Bri -> Cha -> Bri -> Ant] という順番で情報がバケツリレーしたわけです。

この方法を使えば、掲示板などを使わなくても断絶さえ起こらなければ、必ず目当ての人に辿り着けます。しかも、一人当たりが持っているべき情報量が、1000 人の街でも 1000 万の国でもたったの 2 で済みます。

また、すべてのメンバーが対等な存在として配置されるので、一人あたりに必要となるダイヤルの回数も必ず均等に割り振られます。こういうことから、これはもっとも分散されたルーティング手法、と言えますね。

しかし、もうお気づきの方も多い事とは思いますが、この手法には非常に大きい問題点が存在します。それは、一回の電話を成立させるために必要な中継の回数が莫大なものになってしまうという点です。

住民数 N の街に於いて、その値は N/2 となります。具体的に言えば、1000 万人の街では平均すると一回のリクエストにつき 500 万回の電話が必要となるわけです。

500 万回という事は、一回の問い合わせに 10 秒必要と考えると、5000 万秒かかる計算になります。5000 万秒というのは、だいたい 579 日くらいになります。普通に考えて、一回の電話をかけるためだけに一年半もかかっていては、実用的とは言えませんね。そういうわけで、この手法は改善が必要とされる訳です。

V. あなたが知識人になれば、きっと人脈も増える。

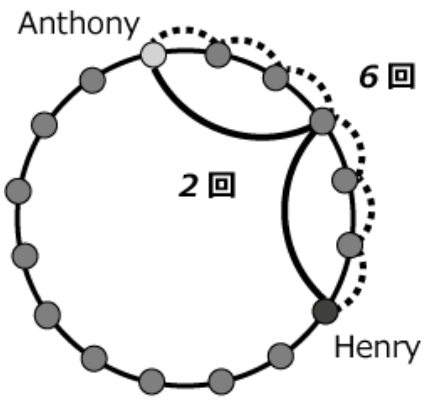
前頁で説明した方法では、とんでもなく電話番号探索に時間がかかってしまい、結局使い物にならないという事でしたが、実は、前頁の方法を採用しつつもその問題の解決する方法がいくつか確立されています。

では、果たしてそういう効率化というのは、どのようにして為されているのかという事を考えてみましょう。

とりあえず、皆が名前順で自分の両隣の人の電話番号しか知らないから、非常に情報や指令の伝達に時間がかかってしまったというのが、前頁における問題の要点でした。ということは、皆が両隣よりも遠くの人の事も知っていれば、もっとずっと速く電話番号検索ができると思いませんか？

ではまた具体化してみましょう。まず一番単純な効率化の例として、全ての住民が自分から名前順で前後 10 人ずつの電話番号も知っていたとします。そうすると、当然のことながらある人から別の人に辿り着くまでに必要な経路の数は、平均すれば 1/10 にまで減少するという事が分かりますね。

これはなかなかに良い効率化と言えます。一人当たり 1000 人分までの電話帳を保持できることを考えると、前後片側につき、500 人分もカバーできる訳ですから、単純計算で N 人の街では平均 N/1000 回の電話によって、目当ての人までたどり着くという事が分かります。最初の方法と比較すれば 500 倍の高速化です。左に模式図を書いてみました。



破線：知識深度 *1* の場合
実線：知識深度 *3* の場合

でも、ちょっと良く考えてみてください。先ほどの計算結果では、1000 万人の街（というか国）では、ある人から別のある人に辿り着くまでに 579 日掛かるという事でしたね。この莫大な値に、1/500 をかけたところで、その結果は 27.8 時間という値です。確かに、一年半から一日強へというのは大きい進歩かもしれませんが、それでも一回の電話に一日もかかっていたのではお話にならないと思いませんか。

結局のところ、自分よりちょっと遠くまで見るという手法では、多少工夫をしたところでその高速化は一次関数的にしか行われず、千万とか億とかいうところまで規模が大きくなって来ると、実用には無理が生じるということです。では、そういった場合でも安定した実用性を確保するためには、どうすればよいのでしょうか。

さらに探索回数を減らす方法の一つとして、円構造を幾つもの円に分割（たとえば名前の最初の音によって 50 分割）するという方法があります。各円の中で電話帳を目いっぱい使った上記の円を構成し、同時に全ての住民が別の円に所属している人の電話番号を、各円につき一人ずつ知っておくという方法によって、各円の中では N/50000、別の円へのアクセスに一回ということで、あわせても大体そのくらいで抑える事ができる、というやり方があります。流石にここまで行くと大概の場合で実用性を持ってきますが、結局一次関数オーダーでの高速化しかできないという事には変わりなく、根本的な解決とは言えないでしょう。

VI. 管理社会化が齎す不安と安心 – UUID –

さて、題名にあるようにこれから前頁で提議した高速なルーティング手法を解説する訳ですが、その前に一つ決めておくことがあります。それは、順番付けに重要である、各住民に与えられる番号の事です。

今まで、住民の順番というものは恣意的に決定される“名前”というファクターに基づいて決定していました。これは、例えば“ともや”は 20 35 36 （かなの順番）という数列で、“ふうこ”は 30 3 10 という数列で表現され、これらを上の桁から比べると、20 の方が 30 より早いので、ともやのほうがふうこよりも前、というものですね。

これは、ある名前の持つ“大きさ”は、

$$\mathrm{Value}(X) = \sum (\text{i番目の文字V} * \text{文字の種類M}^{(\text{文字数N}-\text{i})}$$

という数式で表現されてソートされる、とも表現できます。先例のひらがなであれば、M=50

$$Value(ともや) = 20 * 50^2 + 35 * 50^1 + 36 * 50^0 = 50000 + 1750 + 36 = 51786$$

$$Value(ふうこ) = 30 * 50\text{^}2 + 3 * 50\text{^}1 + 10 * 50\text{^}0 = 75000 + 150 + 10 = 75160$$

とすることができるわけです。

そしてこの手法によって、人の名前を数字として扱う事が出来たわけですが、しかしこれには非常に大きい問題がいくつもあります。まず一つめに、名前というのは人が決めるものである以上、同じ名前の人が複数人存在するケースが容易に発生し得ること。次に同様の理由から、例えば最初の文字が’ろ’で始まる名前が少なかったり、“あああ”を含む名前が普通は存在しなかったりなどといったように、その値の分布に偏りが出るという事、そして三つ目に、長さなどに統一性がなく扱いづらいという事があります。

こういったことから、ある人の電話番号を調べる際、その人の名前というのはキーワードとしては非常に不向きであるという事が分かります。そして、名前にとって代わる新しい表現方法が必要とされる訳です。

ここで出てくるのが、**UUID** です。これは *{C2A27EB5-11FF-4f90-835A-6A22B86011B0}* などといった風に、16 バイト=16 進 32 ケタの数値で表現される ID 値で、一つ一つが（原則として）必ずユニーク、唯一である事が保障されているというものです。原則として—というのは、全く同じ UUID が作り出される確率が、

1/2^128=1 / (3.40282367 × 10^{38}) と、気の遠くなるほど小さい値である事によるからです。

そして、これを住民一人一人が生成して自分の名前の代わりに使う事によって、問題点の一つめや二つ目のような、恣意性による問題点や、仕様の非共通性による問題点を解決する事ができる訳です。分布に関しては、どのくらいの等濃度性が保障されているのかがちょっとわからないので、より精密なものを求めるのであれば、後述の SHA ハッシュなどを使ってさらに濃度を等しくする必要があるかもしれません。

VII. 2^i 歩進んで 2 歩下がらずに、2^{i+1} 歩さらに進んだりしてみる。 － Chord －

さて、いよいよルーティングの話題も大詰めということで、一次関数オーダーの壁を一気に抜けだして、N の数によって速度が変化しないという、Chord というシステムで採用されているルーティング手法を解説します。

このルーティングにおいても、基本的な方針は超分散ルーティングと変わりません。各住民は自分の両隣（UUID によって定義される点に注意）の電話番号を知り、原則としてそれを使って目的地まで目指します。

そして、高速化を両隣以外の住民の電話番号も持っておくことで解決するというのもまた、V で示した手法と変わりません。しかし、この二つの間には、その選び方に大きい差があるのです。その選び方とは、

$$H \equiv ID(Me) + 2^n \bmod (2^{128}) \quad (0 \leq n < 128)$$

という式が成り立つような H をすべて選ぶという方法です。ID(Me)というのは、自分の ID を示します。

式だと分かりにくいかもしれませんから言葉で説明しますと、2^16 個の点が存在する円の上で自分の ID が示す場所から、[2, 4, 8, 16 …]と 2 の累乗個分離れた位置にある人をマークする、ということです。もしもその H を ID に持つ人がいない場合（普通はいない事が多い）は、その値より大きくかつ最も近い値の ID を持つ人をマークします。ちなみにこのマークしたリストを、ルーティングテーブルといいます。

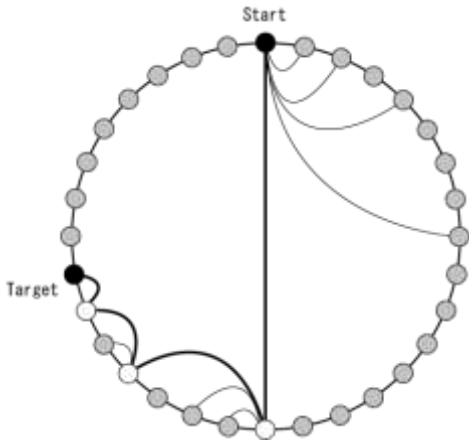
そして、ある人の電話番号を知りたいときには、そのマークされた人々の中から、ID が目的の人の ID よりも大きく最も近い人を選んで電話をかけて聞きます。聞かれた人は、その電話番号を知らなければ、やはり同様の手法でマークされた人々の中から、同様の手法で聞く人を選んで質問を伝達します。そして聞かれた人も・・・と、中継する事で目的地に辿り着くのです。どうでしょうか、基本的な方針は最初のものと同じでしたね。

では、本当にこのルーティング方法では住民数 N の値がいくつになっても、速度が変化しないのでしょうか。

情報伝達が、起点 A から X を目的地としてテーブル[m]の人 B に伝わる時の伝わり方を考えてみると、

$2^m \leq |ID(B) - ID(A)| < 2^{m+1}$ かつ $2^m \leq |ID(X) - ID(A)| < 2^{m+1}$とわかる。$(0 \leq m < 128)$

このとき、 $ID(B) < ID(X)$ という状況は、エラーがない限りは原則として発生し得ません。また、$ID(B) = ID(X)$ ならば目的地に届いたことになります。$ID(B) < ID(X)$の時には C へと引き継がれます。この時$ID(X) - ID(B) < 2^m$という式がわかり、そのように探索範囲が限定される以上、$|ID(C) - ID(B)| < 2^m$ が成立する事が分かります。すると、次の処理では 2 の指数は m−1 以下という事になり、つまり、転送処理が一度為されるたびに指数の値が 1 以上必ず減少するということになり、転送回数は最大で 128 である、という結論が導き出せました。以上より、Chord では住民数に関係なく小さいルーティング速度を維持できることが証明できました。



VIII. ジャガイモつぶして揚げたのを、ポテト抜きで頼む。 － DHT 概論 －

前頁までで基本的なルーティングについては大方全て解説が済みました。
ということで、いよいよここから、メインテーマである「DHT」の解説に移っていこうかと思います。
とはいっても、DHT の本質の多くはそのルーティングにあるので、解説する事はあまりないのですが。

まず”DHT”というのは、”Distributed Hash Table”を省略した単語で、日本語では「分散ハッシュテーブル」と呼ばれており、そのまま「ハッシュテーブル」を分散管理する技術の事です。テーブルというのはデータベースのようなもので、ハッシュというのは先ほど住民（＝ノード）に UUID を振り分けた様に、様々な情報・コンテンツに対して割り振る ID の一種です。ただしこれが UUID と異なるのは、UUID は同じ人が二回生成すればそれぞれ違ったものが生成されるのに対して、ハッシュは生成する元となるデータが等しければ必ず同じ ID が生成されるというところにあります。また、データがたった 1 ビットでも異なれば、全く違った ID が生成されたり、仕様が完全に統一されている UUID に対して、ハッシュには様々な仕様や生成方法があるという相違点があります。特に MD5、SHA-1 などといったものが有名ですが、それぞれ計算方法や衝突耐性（=同じハッシュが生成される確率の低さ）、生成されるハッシュの大きさなどに差があり、時々で最適な物を選んで使うことができます。

ハッシュを使う事によって様々なコンテンツに、住民に ID を振る時にあったような問題を解決できる名前付けを行う事ができます。そして、DHT というのは、このハッシュを前頁までに開設したルーティング技術と組み合わせて分散管理することで、莫大なコンテンツを容易に探索可能にする技術の事なのです。

実際の管理方法を、自分で描いた絵を売ろうとしている人を例に交えつつ、解説しましょう。

① 作者は絵のハッシュを生成する。この時、ハッシュは UUID と同じ桁数の物を使う。
② そのハッシュ値を、絵を展示してある画廊の場所情報とセットにして、そのハッシュと同じ UUID を持つ人に渡す。当然同じ UUID を持つ人は滅多にいないので、実際にはそのハッシュよりも大きく、かつ最も近い UUID を持つ人に渡す。こういう条件に該当する人 H を、Next(ハッシュ M)と定義。
③ 作者は絵のハッシュとメタデータを何らかの手段、例えば web などによって公開する。
④ メタデータを見て絵を欲しがった人は、Next(該当ハッシュ)の人にアクセスして、画廊の場所情報を貰う。
⑤ その情報に従って、画廊に行って絵を買う。

以上が大筋となります。このケースの場合は必要となる情報が住所など小さいものなので特別な必要性を感じないかもしれませんが、大きいコンテンツ自体を扱うときには重要性が大きくなります。

DHT を使う事で、あるノードが最大でもたった 128 ノードの情報を知っているだけで、ネットワーク上に存在する、ありとあらゆる情報を漏れなく探索する事が出来ます。

さらにコンテンツ自体の多人数で共有・分散を行えばネットワークトラフィックの減少・分散も見込めます。

DHT が実に画期的な情報分散技術であるという事、お分かりいただけましたでしょうか。

IX. 作る理系と使う文系

さて、これまで DHT 技術を大方解説し、それによって大規模な情報分散が可能になるという事が分かった訳ですが、実際に DHT を使ったネットワークプログラムの例をいくつか示そうかと思います。

① BitTorrent

これは非常に有名なファイル共有ソフト、あるいはそのプロトコルです。

ファイルを共有する際、一定のサイズごとにファイルをピースという単位に分割し、あるピースを受け取るときにはこちらからもピースを送信しなければならないというルールを設ける事によって効率的にファイル拡散を実現しています。これによって、一般的な固定ノードのみがコンテンツを保有しているタイプのファイル共有ソフトでは、ファイルが人気であればある程にダウンロード速度が落ちるのに対して、逆にファイルが人気であればある程により広くファイルが分散しダウンロード速度が上がります。

特に HTTP や FTP での配布ではサーバーに負荷掛かり過ぎてしまうような、Linux ディストリビューションイメージなどの巨大ファイルを配信する場合において BitTorrent はその本領を発揮します。

Azureus などの互換実装も多数存在し、それぞれでまた独自の DHT を実装している事があります。

② Perfect Dark

（おそらく）日本の匿名開発者によって開発されている、ファイル共有ソフトです。

コンセプトとしては Winny や Share に近いものですがネットワーク構造的には大きく異なり、前者は DHT を構築せず、ブロードキャストの範囲内でしか探索を行えなかったり、クラスタを構築する必要があったりするのに対して、Perfect Dark ではそのような問題点が解決されています。また、DHT 特有の「完全一致検索しかできない」という問題点も、distributed keyword table と呼ばれる技術によって克服されており、多種多様な P2P ファイル共有ソフトの中でも、DHT による分散と柔軟なファイル検索とを両立させたものとして、極めて革新的なものであると言えます。

しかし、開発者の意向によってプロトコルの詳細な仕様などは公開されず、また全体的にアンダーグラウンドな雰囲気を持っているために、技術的な意味での評価や注目をされない状態にあります。

いくつか解説すると言っておきながら、二つしか解説していませんがこれには理由があります。
それは、DHT を使用した著名なプログラムのほとんどがファイル共有ソフトである、という事です。
そもそもこれまで解説してきたように、情報の効率的な分散に向いた仕組みである以上、ファイル共有という分野に於いてその強さを発揮するのは当然のことなのですが、DHT というのはまだまだ発展途上の技術ですので、そのうちどんどんファイル共有以外の事にも生かされてくるのではないかと思います。

例えばファイル共有の延長線上として、web 的な広義な情報の共有・データベース化には間違いなく利用されるでしょう。グリッドとも繋がるかもしれません。いずれにしても今後が非常に楽しみです。

X. おわりに

さて、ここまで 10 ページもの長い間、DHT という技術に関して、基本的なところを解説してきたわけですが、如何でしたでしょうか。とりあえず DHT を使うとすごくたくさんの情報が簡単に引き出せるようになるんだよー、という点が理解していただけていれば、それで十分です。そして、もしも DHT を含めたこういった情報科学に興味を持ってもらえたりしていたら、とても喜びます。

P2P は、最近妙に悪い視線を浴びる事が多いような気もしますが、とにかく非常に面白い分野です。P2P ネットワーク上にいるのは、サーバーとクライアントという閉鎖されたペアではなく、何千何万という生きたノードです。そしてこれらは実にアトラクティブで予想のできない挙動をして、それがとても面白い。

ライフゲーム（人生ゲームじゃありませんよ）というものを知っている人もいるかもしれませんが、ああいった、小さなルール・挙動が合わさって、マクロな変化を起こす様子というのは、見ていて実に興奮するものです。

だから皆さんも P2P を研究しろ—とは言いませんが、普段コンピュータで色々なソフトを使っていれば、きっとそのうち一つは P2P を使ったものがあるはずです。そういったものを見たときに、ただ漠然と便利だなと使うのではなく、その後ろでどのようなネットワークが構成されて、自分のコンピュータはそのネットワーク上でどのような挙動をしているのかなぁという事を考えて使うようにすると、きっと少しだけ、毎日が豊かになるでしょう。

最後に昨年の部誌の終わりの部分で少しだけ書いた、P2P と SNS を融合させて新しいコミュニケーションを作るという構想について、プロジェクトの進行状況と概説を書かせてもらって終わりという事にします。

まず、昨年プラグインが使えたら面白いねーとか書いた覚えがありますが、その点に関して無駄にこだわったおかげで、自分で言うのもなんですが、非常に柔軟性の高いプラグインシステムが出来ました。その名もリンカーコア。これの設計と製作のためにだいたい 3 か月くらいかかりました。全体の構想と同じくらい。

最初は適当にデータを共有できてあとはそれぞれで独自のスレッドで動くだけーとか考えていたのですが、気づいたら TCP の上に新しい汎用プロトコルを一枚作っていました。プログラマーはこれを使えばソケットとか考えずにネットワークプログラミングができるようになったわけです。もちろん制限もありますが。

次に SNS を構築する部分ですが、これも基本的な部分は出来ました。コミュニティベースで動くので、そのあたりは保守も含めて大方完成したのですが、友人という概念は未だに存在していなかったり。今後ちゃんと実装します。あと、実はこれの制作のために Pastry という DHT の技術の一部を参考にしていまして、今回この記事で DHT をテーマに選んだのもそれが要因。今後は本格的な DHT も組み込んでデュアル DHT 構成にすることで、より高スケーラビリティで柔軟な SNS 機能を実現できればいいなと。

今大会に出品する事になっているのでテストをやっているのですが、そのあたりが一区切りついて色々と機能追加が済んだら本格的に一般公開しようと考えてたりするので乞うご期待。では、そういうことで。