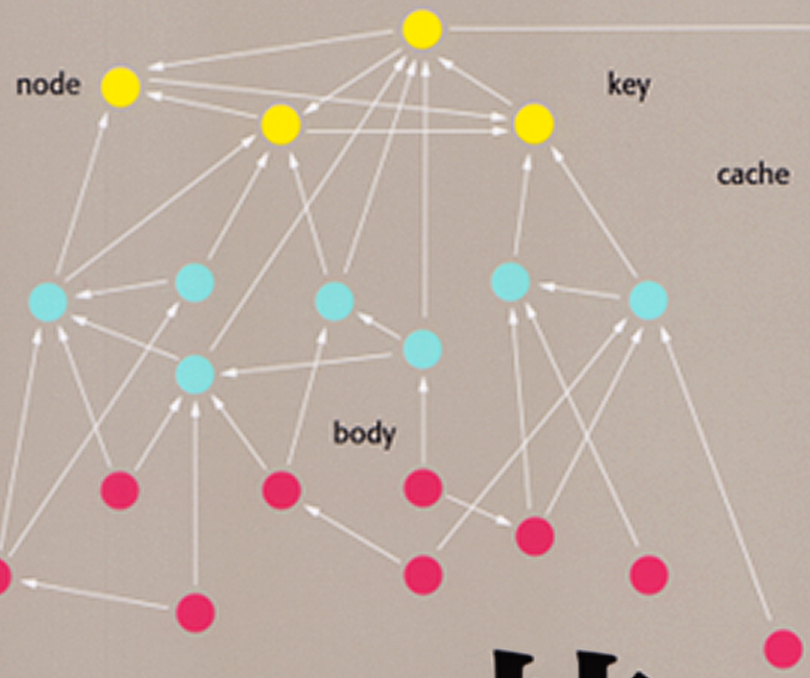


金子勇 著
Kaneko Isamu

Winnyの技術

The Technology of Winny

アスキー書籍編集部［編］

ASCII

金子 勇
著

Winnyの技術

The Technology of Winny

アスキー書籍編集部［編］

ASCII

Fingerprint:
16B6 7953 A7C1 2DCA 9DEF C7D1 2546 1CCE C03F 2BA3

まえがき

本書は、P2P 型のファイル共有ソフト Winny の動作と構造を詳細に解説したものです。

P2P 型のファイル共有ソフトというと、アンダーグラウンドな印象があるかもしれませんが、サーバーで集中的にサービスを提供する形態によらず、ユーザーが相互にコンピュータをつなぎあってサービスを実現しているという点でたいへん興味深いシステムであり、その技術面は確実に進化してきました。代表的なものとしては、ファイル共有ソフトとして非常に有名になった第一世代の Napster、サーバを使用せずに純粋にユーザーのコンピュータ間だけで情報を交換できるようにした第二世代の Gnutella、そして自由な発言と出版を目的として、情報発信者のプライバシーを護るために匿名性を実現した第三世代の Freenet を挙げることができます。Winny はこの Freenet を参考に、Freenet が持つキャッシュシステムをより大規模な利用にも耐えられるように効率を高めて開発したものです。

本書は読者対象として、Winny の動作や仕組みに関心のある方と、P2P 技術に興味があるソフトウェア技術者を想定しています。

1〜3 章では、Winny の技術背景となっている P2P 型システムについての基礎知識を説明し、ファイル共有ソフトの歴史を紹介したうえで、Winny の基本的な動作と設計コンセプトをわかりやすく解説しました。Winny の動作や仕組みに関心のある方はこちらをご覧ください。

4〜5 章では P2P 技術に興味がある開発者向けに、さらに詳細な実装を解説し、実際に P2P ソフトウェアを開発してみてわかった注意事項やヒント、今後 P2P ソフトウェアを開発される方へのメモをまとめました。

Winny は、ファイル共有ソフトとして Winny Version 1 が公開されたあと、P2P 型の大規模掲示板を目指した Winny Version 2 へと開発が続いていくわけですが、本書は Winny Version 1 に焦点を絞って解説し、Winny Version 2 に関しては展望のみにとどめることにします。また、Winny の利用方法やビジネス面、および法的な面は、別の場で議論されるべきことなので、本書では扱いません。

Winny は、ファイル共有ソフトの歴史の流れの途中にあるにすぎません。Winny

やほかのファイル共有ソフトによって確認された技術を基にして、これからさらに新しい技術が生み出されていくはずです。Winny の技術詳細を公開することにより、P2P 技術がより発展することを期待して本書を記します。

本書の執筆にあたっては、多くの方々からご意見やアドバイス、協力をいただきました。深く感謝いたします。

特に、本書を出版する機会を与えてくださった、東京大学の平木敬教授に感謝いたします。カバーのメインモチーフとなっているイラストの図案は、平木先生がノート PC の PowerPoint で描いてくださったものです。

裁判が続くなかでの執筆といういうこともあり、出版もスムーズというわけにはいきませんでした。編集担当の赤嶋映子さんには数多くのお手数をおかけしましたが、こうして無事に出版することができました。

執筆期間中、今井潔さん、清水了さん、中野克平さん、川崎晋二さんは、ときには説明の聞き手になりながら、草稿の数多くの問題点を指摘してくれました。篠田陽一さんからは貴重な資料を見せていただき、Winny を新鮮な目で見ることができました。

井上誠一郎さん、加藤朗さん、亀井聡さん、萩野純一郎さんにはテクニカルレビューをしていただき、鋭い意見を原稿に反映することができました。

本書とは直接のかかわりはありませんが、新井俊一さんをはじめとする支援団体の方々に多くの励ましをいただきました。Winny 開発のきっかけとなった、2 ちゃんねる関係者の皆さん、2 ちゃんねる管理人のひろゆき氏、ネットワーク上でご意見と励ましをいただいた多数の方々にもこの場をかりて感謝いたします。

また、事情で現在直接やり取りができない方、直接お会いしたことはありませんが Winny に多くの関心をはらっているたくさんの方々にも感謝したいと思います。

ネットワーク社会の未来が明るいものであることを祈ります。

2005 年 9 月

金子　勇

目　次

P2Pの基礎知識

1章

Winny は、どこかの会社や誰かが運用している特定のサーバを使わずに、利用者がインターネットを介して相互にコンピュータをつないでシステムを維持し、ファイルを共有します。いわば、ユーザーが「持ち合い」で運用するファイル共有システムだといえます。このようなシステムがどうしたらできるかは、ずっと以前から研究が続けられてきました。Winny は、先人達のファイル共有ソフトを参考にしながらいくつかの新しい試みを採り入れて開発したもので、(作成者の想像を越えることでしたが) 結果的に数十万から 100 万ともいわれるユーザーが参加するネットワークができあがりました。

この本は、Winny の技術面を解説することが目的ですが、それに先立って、Winny の基盤となっている P2P 技術、また Winny よりも以前に開発されてきたファイル共有ソフトの歴史を紹介し、Winny が開発された背景や継承した技術をなぞっていくことにします。

実は、P2P という用語の定義は一様ではなく、この言葉からどのような意味を感じ取るかも人さまざまです。また、私自身も P2P を専門に研究していたわけではないので、異論があるかもしれません。そこでここでは、私が考える P2P、あるいは私の目から見たファイル共有ソフトの歴史を述べることにします。

1.1 私的解説：P2P とは何か？

ファイル共有ソフトは、P2P 型のアプリケーションのなかでも最も普及したものでしょう。そのためか、「P2P ソフト」とはファイル共有ソフトのことだと勘違いをされているきらいがあります。しかし、P2P 型のアプリケーションはそれ以外にもいろいろあり、ファイル共有ソフトだけを指すわけではありません。

P2P とは“Peer to Peer”を短く約めたものです。Peer (ピア) とは「対等な対象」という意味で、ここではコンピュータを指します。つまり P2P とは、対等な動作をするコンピュータによる、主従関係のないシステムモデルです。ビジネスモデルの分野で P2P といえば、Person to Person――個人と個人のやり取り――を指す場合もありますが、ここでは通信モデルとしての Peer to Peer を説明していきます。

クライアント／サーバとは対照的な P2P

P2P とは対照的な形態が、クライアント／サーバ方式です。サーバとはサービス提供者であり、クライアントとはサービスを受け取る顧客です。つまり、サービスを提供する側と受け取る側がはっきり分かれているのがクライアント／サーバ方式です。

インターネットには、クライアント／サーバ方式で動作しているサービスが非常にたくさんあります。おそらく、最もよく知られているものは Web システムでしょう。Web システムのサーバ側アプリケーションとして代表的なものは、Apache の httpd です。また、クライアント側アプリケーションとなる Web ブラウザの代表的なものとして、Internet Explorer などを挙げられます。

Web システムにおける Web ページの参照は、

1. あらかじめ Web サーバに HTML 形式などのファイルを溜めておく。
2. クライアントの Web ブラウザが HTTP を使ってサーバに接続し、ファイルの送信を要求する。
3. サーバが送り返すファイルをクライアントが受け取る（ダウンロード）。

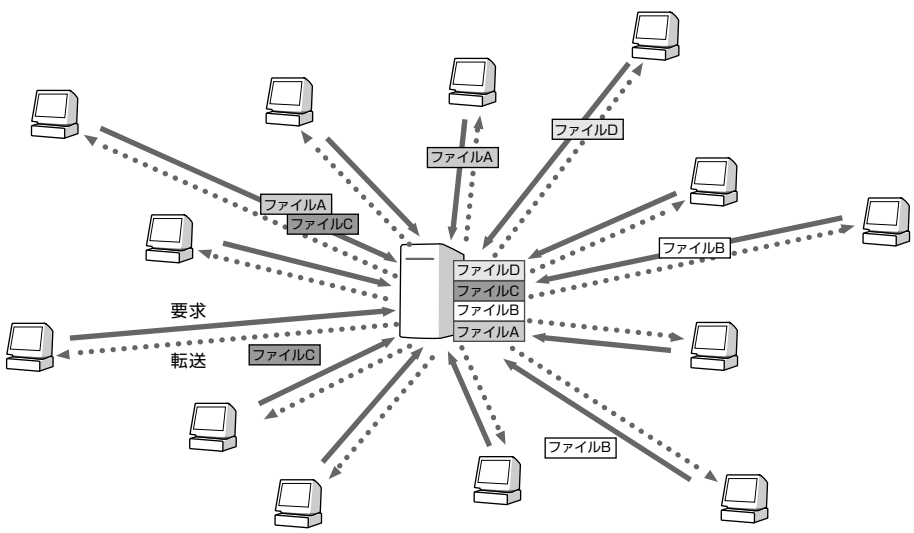


図 1-1　クライアント／サーバ方式のシステムでファイルを配布する

という方法で実現しています。

ここで注目すべきことは、サーバはクライアントからの接続を受け付けてもっぱらサービスを提供するのに対し、クライアントはサーバに接続して一方的にサービスを利用するだけで、自ら接続を受け付けることがないということです。サーバには、クライアントの接続がスター状に集中する形になります（図 1-1）。クライアント／サーバ方式のインターネットサービスとしては、ほかにも FTP、TELNET など、多くのものが存在します。

このクライアント／サーバ方式とは対照的な通信形態が P2P なのです。

P2P 方式の場合、サービス提供者となる特定のサーバはなく、対等な役割をはたす各ピアが状況に応じてサービスの提供者になったり利用者になったりします。実際の接続手順でも、ピアは受動的に接続を受け付ける一方で、能動的に他のノードへ接続を要求する側ともなります。

P2P システムの接続のようすはノード（節）とリンク（結線）による網目状の構造となるので（図 1-2）、このピアをノードと呼びます[*1]。また、クライアント／サーバ方式では、サービスの提供側をサーバ、利用側をクライアントと呼んでいたのに対し、P2P のピアはサーバとクライアントの両方の動作をすることから、サーバントと呼ぶことがあります。

*1 ノードとリンクは、もともとグラフ理論という数学分野の用語です。ノードは連結対象の節を、リンクはそのあいだの連結線を意味します。

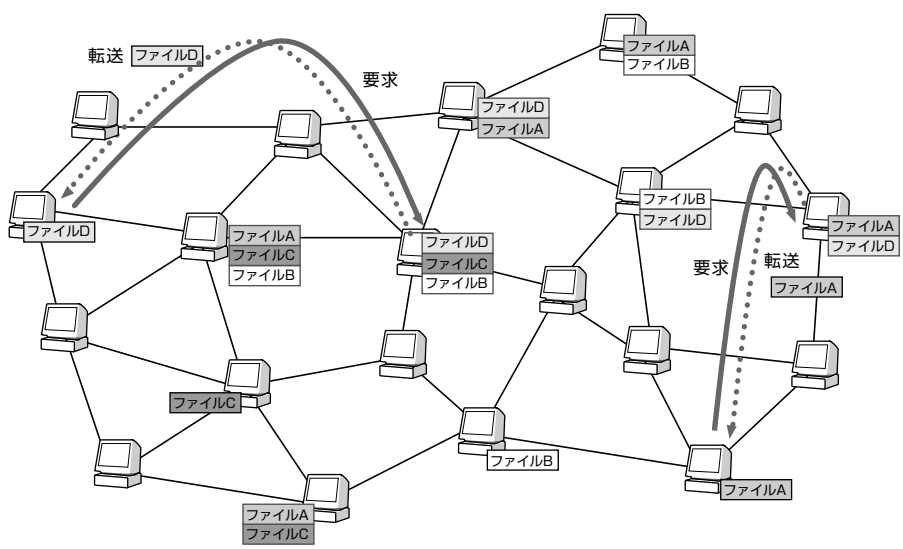


図 1-2　P2P 方式のシステムでファイルを配布する

P2P 型システムの特徴

P2P 型のシステムは、対称的なノードが協調して機能すると述べましたが、そのようなシステムを実現するためにはいくつかの要件があります。まず、サービスに必要なリソース――ファイルなどの資源は、各ノードが分散して持っていることになります。また、システム内にはリソースやノードの所在すべてを一括して把握しているものがないので、手に入れたいリソースがどのノードにあるかを調べるなんらかの仕組みが必要です。

ちょうど蟻が仲間の蟻から情報をもらって砂糖のありかにたどり着くように、P2P 型システムの各ノードも自律的にそれぞれが持っている情報を交換しながら目的のリソースを持つノードを探し、リソースを手に入れます。これは、P2P ソフトの開発者には関心の高い部分であり、アプリケーションによって設計が大きく異なります。

また P2P 型のシステムでは、ノードが新しく参加したりふいに切り離されてもサービスを継続できるように設計する必要があります。

以上のようなことから、P2P 型のシステムには次のような特徴があります。

- **システム規模の急拡大にも耐えられる**

  クライアント／サーバ型のシステムは、システムの利用者が増大してサーバにリクエストが集中すると、サーバやその周辺のネットワークにかかる負荷が高くなり、この部分がシステムの弱点になります。しかし P2P 型のシステムは、サーバの機能を各ノードが分散して維持しているので、システムの特定の部分に負荷が集中することがなく、利用者の急増にも耐えることができます。

- **コンピュータやネットワークの障害に強い**

  クライアント／サービス型のシステムは、サーバが停止するとシステム全体が利用できなくなりますが、P2P 型のシステムの場合、システム内の一部のノードが停止してもサービスを継続できるので、故障に強いといえます。

- **データの一元管理や短時間の同期は苦手**

  サービスに使用する情報がネットワーク内に分散しているので、データの一貫性を短時間で同期させるのが難しく、データの一元管理がしにくくなります。情報に生じた変更を短時間のうちに他の情報と同期させなければならないような用途には向きません。

● P2P ネットワーク全体の管理や監視が難しい

中心となるサーバがないので管理の必要がないという見方もできますが、システム内の情報やノードすべてを把握しているものがないので、システム管理の仕組みを作るのが難しいという面があります。多くの P2P 型システムは、いったん運用を開始すると、システム全体を停止することができません。

以上のような特徴から、P2P 方式がどのようなアプリケーション分野に向くかが見えてきます。

P2P で何ができるのか？

すこし乱暴なたとえですが、ネットワークアプリケーションを設計するときにクライアント／サーバ方式と P2P 方式のどちらを採用するかは、誰かとの連絡に掲示板と電話のどちらを使うかということと同じです。目的と状況に応じて、システムの効率やコスト、管理の手間を考え、適材適所の方法を選べばよいわけです。

現在、ネットワーク技術は、Web やファイル共有などの情報共有、電子掲示板などのコミュニケーション、あるいはオンラインゲームなどのアミューズメントなど、さまざまな分野に利用されています。その多くは、クライアント／サーバ方式で実現されていますが、P2P 型のシステムでも実現可能です。あとは向き不向きの問題です。

たとえば、映画や放送などの配信をクライアント／サーバ方式で実現しようとすれば、大量に集中するリクエストに応えて大容量のファイルをサービスできるように、膨大なストレージ（記憶装置）を備えた非常に大規模なサーバと高速な通信インフラが必要であり、きわめて高価なシステムになります。一方 P2P 型のシステムであれば、利用者のコンピュータやネットワークなどの遊休時間を有効活用できるので、低コストのシステムでサービスを構築可能であり、このような用途には効果のある方法だと考えられます。

逆に P2P 方式は、リソースの集中管理や利用制限を必要とするシステムには向きません。たとえば、ユーザーの認証情報のように一元管理が必要で、生じた変更を即座にシステムへ反映させなければならない場合、P2P 方式で情報を分散させていると、どのノードの情報を信用するかという問題が生じます。このようなシステムには、クライアント／サーバ方式のほうが向いています。

P2P ソフトいろいろ

このように、P2P 方式はいろいろなネットワークアプリケーションに応用可能ですが、現在 P2P 方式のネットワークアプリケーションは多くはありません。代表的な P2P ソフトといえば、やはりファイル共有ソフトですが、ほかにもグループウェアやインスタントメッセンジャー、インターネット電話のようなコミュニケーションツールが、実用的に使用されています。

また、オンラインゲームや掲示板、Web システムなど、従来はクライアント／サーバ方式が使われてきた分野にも、P2P 方式を採り入れる試みがあります。

P2P 型ファイル共有ソフト

代表的な P2P アプリケーションとして多くの人が思い浮かべるのは、ファイル共有ソフトでしょう。本書で扱う Winny も、このファイル共有ソフトに分類されます。詳しい歴史や技術的なトピックについては 2 章「Winny 紹介」で触れますが、ここではファイル共有ソフトがどのようなものなのかを簡単に説明します。

図 1-3 は代表的なファイル共有ソフトである Napster[*2]の画面です。

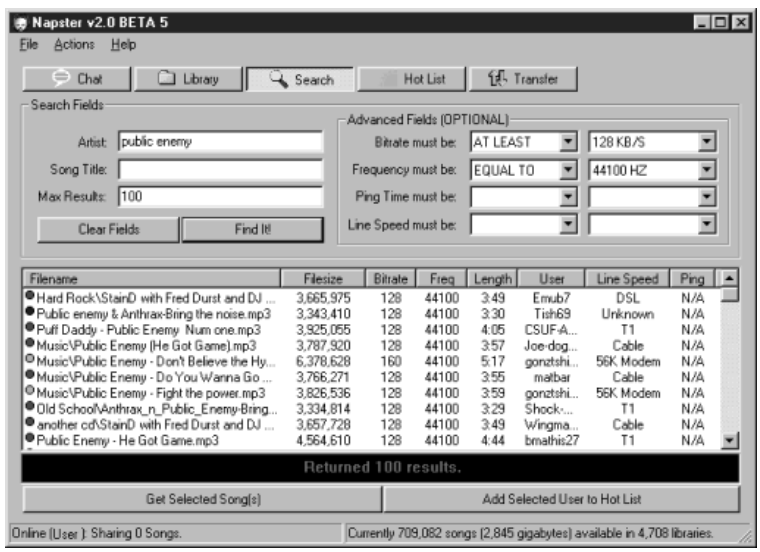


図 1-3　Napster 系ファイル共有ソフトの起動画面

*2　Napster は、いまでは商用サービスになっていますが、この画面はフリーソフトとして公開された世界初のファイル共有ソフトのものです。

ファイル共有ソフトというのは、その名のとおり、ユーザー各自が持っているファイルをインターネットを介して共有できるようにするソフトウェアです。Napsterが画期的だったのは、Napsterのユーザーそれぞれが自分のコンピュータに保有しているファイルを、まとめて検索できるようにしたことです。目当てのファイルがどこにあるのかを知らなくても、Napsterの検索機能で探して入手することが可能になったのです。

図1-3には検索キーワードを入力する部分がありますが、ここに関心のあるキーワードを入力して検索ボタンを押すと、他のユーザーがそれぞれのコンピュータで公開しているファイルを検索し、キーワードとファイル名がマッチするものを見つけ出すことができます。キーワードにはファイル名のすべてを指定する必要はなく、一部分だけでもかまいません。

検索で見つかったファイルは、転送するように指示をすると、自分のコンピュータにコピーされます。このことをダウンロードといいます。このように、ファイルをキーワードで検索してダウンロードするというのが、ファイル共有ソフトの基本的な利用方法です。

ファイルをダウンロードできるということは、どこかでそのファイルを送信（提供）している人がいるということです。これをアップロードといいます。ファイルをアップロードしているのは、ダウンロードと同じファイル共有ソフトを利用している別のユーザーです。これがファイル共有ソフトの面白いところであり、P2Pらしいところです。

多くの場合、ファイル共有ソフトにはアップロードフォルダを設けられるようになっています。アップロードフォルダに適当なファイルを置くと、ファイル共有ソフトを使用している他のユーザーは、そのファイルを検索してダウンロードできるようになります。こうして、ユーザーは自分のファイルを手軽に公開し、それをみんなで共有できるようになるわけです。

なお、情報の共有はどのアプリケーションにも共通する基本技術ですので、ファイル共有ソフトの技術は他のP2Pアプリケーションの基礎ともいえます。

1.2 P2P ファイル共有ソフト

次に、ファイル共有ソフトの進化のようすをもうすこし詳しく見ていきます。

Winny は、ファイル共有ソフトの進歩の過程で生まれたソフトウェアのひとつであり、第三世代の P2P ファイル共有ソフトに分類されます。それまでのファイル共有ソフトを理解することは、Winny の詳細を理解するうえで役立つでしょう。

P2P ファイル共有ソフトの技術ポイント

ここでは、技術的な特徴によって、P2P ファイル共有ソフトを第一世代、第二世代、第三世代というように分類することにします。

- **第一世代**

  P2P によるごく初期のファイル共有ソフト。ファイルを持っているノード同士が直接ファイルを送りあうが、どのノードがオンライン中でどのようなファイルを持っているかといったノード情報は、中央のサーバが集中管理する。代表的なものは Napster。

- **第二世代**

  特定のサーバを必要とせず、純粋にノード同士でファイルの検索と転送を可能にしたファイル共有ソフト。Gnutella が有名。

- **第三世代**

  第二世代のファイル共有ソフトにファイルのキャッシュ機構を備えている。Winny はこの第三世代に属する。キャッシュ機構を備えることによって、匿名性が高まるという効果も生じます。

第一世代と第二世代の違いははっきりしていますが、第三世代の特徴をキャッシュ機構としたことには、異なる意見があるかもしれません。これは、今後の P2P アプリケーションがどのように発展していくかにもよるでしょう。

技術的な面に注目すれば、これらの世代間では特に「ノードの発見」「ファイルの発見」「ファイルの転送」に大きな違いがあります。

● **ノードの発見**

P2P ファイル共有ソフトは、他者となんらかのファイルを共有するための仕組みですから、他のノードと接続しないことには始まりません。クライアント／サーバシステムであれば、いつも決まった場所にあるサーバに接続すればよいのですが、P2P 型のシステムの場合、ファイルの検索を始める前に、まずどのようにして他のノードを発見し、接続するのかが、ファイル共有ソフトの最初の問題となります。

ノードの発見に関する手法は大きく分けて 2 つあります。ひとつは、決まった場所になんらかのサーバを用意し、ノードの発見だけはクライアント／サーバ方式にするというものです。もうひとつは、なんらかの方法で他のノードをどれか発見し、そのあとはさらに別のノードを紹介してもらって順に接続していくという方法です。

● **ファイルの検索**

大規模に使われているファイル共有ソフトでは、ネットワークに参加しているノード数は最低でも数万ノード（数万人が同時に使用している）、維持しているファイル数も億単位に達します。これらのノード間で、いかに効率よくファイルを検索するかが重要なポイントになり、性能や使い勝手に大きく影響します。

ファイルの検索にも大きく分けて 2 つの手法があります。ひとつは、どこか決まった場所に用意した検索専用のサーバで集中的に検索を処理する方法です。つまり、ファイルの検索にも P2P ではなくクライアント／サーバ方式を使用するというものです。もうひとつは、ファイルの検索に P2P 的なメカニズムを採用し、複数のノードが協調してファイルを検索するというものです。

● **ファイルの転送**

検索して見つかったファイルは要求元のノードに転送することになります。P2P 方式のファイル共有ソフトでは、ノード同士が直接接続をしたあと、HTTP（Web システムにおけるファイルの転送プロトコル）のような手順を用いてファイルをやり取りするのが一般的でした。しかし、近年のファイル共有ソフトでは、ファイルのブロック転送や多重転送など、いろいろと新しい手法が導入されています。また、ファイル転送方式の工夫により新たに匿名性という要素が加わってきています。

表 1-1　ファイル共有ソフトの世代

備えている機能	第一世代	第二世代	第三世代
ノードの発見	クライアント／サーバ型	P2P 型	P2P 型
ファイルの検索	クライアント／サーバ型	P2P 型	P2P 型
ファイルの転送	P2P 型	P2P 型	P2P 型
ファイルのキャッシュ機構	なし	なし	あり

1.3 ファイル共有ソフトの歴史

以上の分類に沿って、技術的な面に注目しながら、各世代のファイル共有ソフトの進化を見ていきましょう。

第一世代ファイル共有ソフト

ファイル共有ソフトの歴史を語るうえではずすことができないのは、Napster です。Napster は 1999 年 1 月、米国マサチューセッツ州ボストンにある Northeastern University に通う学生だった Shawn Fanning が開発したソフトウェアで、ファイル共有ソフトという分野を確立した画期的なものでした。

Napster は、MP3 形式のファイルを共有しようというニーズから生み出されました。キーワードでファイルを検索し、見つかったファイルを指定してダウンロードするというファイル共有ソフトの基本スタイルは、この Napster が築いたものです。

あまりにも有名なソフトウェアなので、その詳しい歴史については省略します。Napster はファイル共有ソフトの第一世代に分類されるもので、ファイルの転送のみ P2P 方式とし、ノードの発見やファイルの検索にはクライアント／サーバ方式を使用しています。

ノードの発見

Napster に代表される第一世代ファイル共有ソフトでは、ユーザーすべてが共有するサーバが必ずどこかに動作しています。このサーバはファイル共有システムの管理者が運用するものです。ユーザーがファイル共有ソフトを起動すると、そのプ

ログラムはまずサーバに接続します。そのため、サーバ側はどのノードが動作中かを把握することができます（図 1-4）。

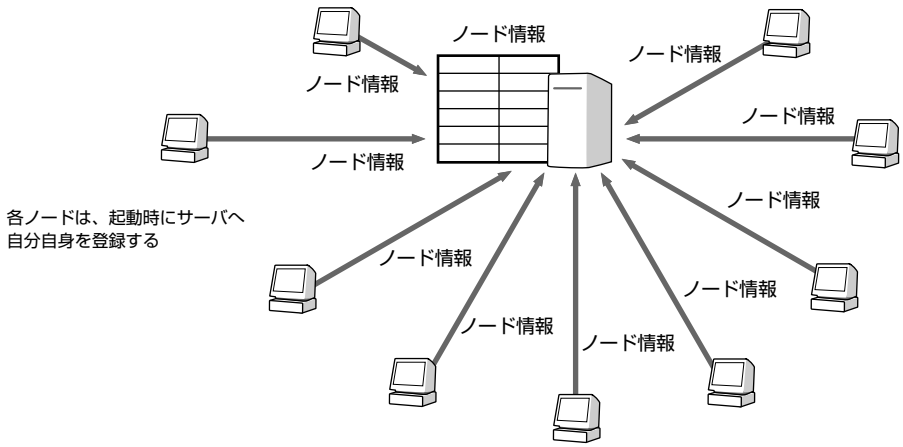


図 1-4　ノードの発見―第一世代ファイル共有ソフトの場合

このような中央集中型のサーバに接続するという形態は、従来のクライアント／サーバ方式と同じで、ファイルを発見するまでの流れは FTP を使ったファイル転送の場合と変わりません。また、サーバがダウンしてしまえば、システム全体が機能しなくなります。

コラム　第一世代のファイル共有ソフトと系

第二世代以降のファイル共有ソフトでは、あるノードがネットワーク内のすべてのノードの所在を知らなくても、直接知っているノードからさらに他のノードをたどるというノード発見の連鎖によって、自ノードをめぐるネットワーク（系）が形成されます。しかし、第一世代のファイル共有ソフトでは、ノードはまず特定のサーバに接続し、そのあとのファイル検索の結果、サーバに教えてもらったノードと直接通信を行うにすぎません。第二世代以降のファイル共有ソフトに見られるように、ノードどうしで自律的にネットワークを形成するわけではありません。

ファイルの検索

ユーザーがファイルを公開すると、そのファイルのインデックス情報はすべてサーバに集められ、サーバに問い合わせをすればファイルの検索が可能となります。インデックス情報には、ファイル名や大きさ、ファイルを持っているノードの情報などが含まれます。インデックス情報はすべて中央の検索サーバに集約されるので、サーバ側で検索を実行すれば、簡単に全ファイルを検索できます。

システムを利用しているユーザーはサーバに接続しているので、このサーバを使って他のユーザーが持つファイルも検索して発見することができます（図 1-5）。

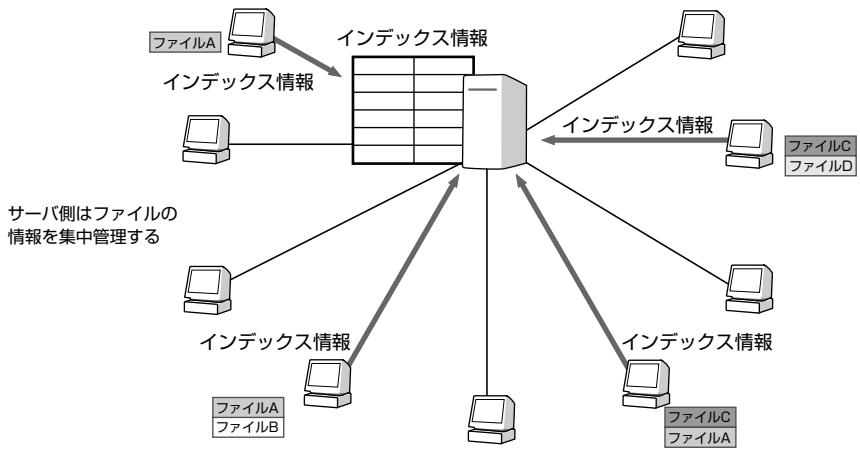


図 1-5　ファイルの検索―第一世代ファイル共有ソフトの場合

ファイルの転送

クライアント／サーバ方式の FTP というファイル転送プログラムでは、ファイルを得ようとするノードがサーバに接続し、そこに蓄えられているファイルをダウンロードします。これに対して P2P のファイル共有ソフトでは、ファイルをサービスするのは中央に位置するサーバではなく、ダウンロード側と対等な関係にある別のノード――アップロード側のノードです。ファイルを得ようとするノードは、サーバでの検索によって目的のファイル情報が見つかれば、どのノードがファイルを持っているのかがわかります。その情報を使って、アップロード側のノードに直接接続し、ファイルをダウンロードします（図 1-6）。

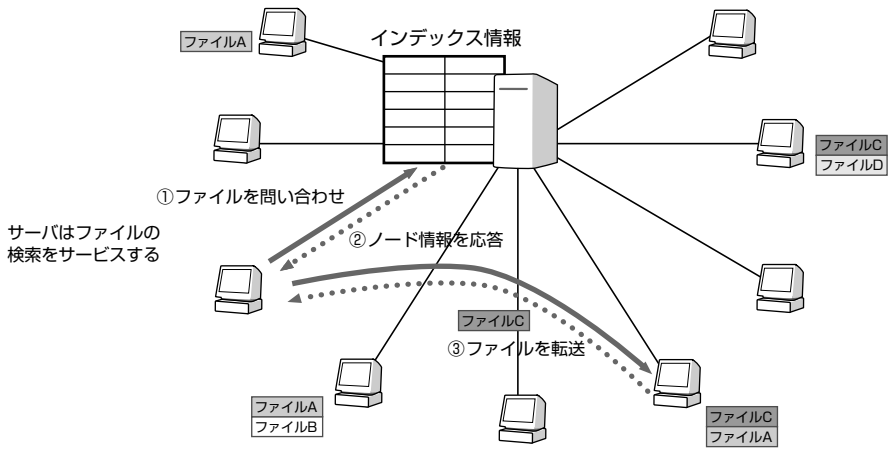


図 1-6　ファイルの転送―第一世代ファイル共有ソフトの場合

このように、P2P ファイル共有ソフトのノードはファイルをダウンロードするだけでなく、接続を受け付けてアップロードも行うことになり、ファイル転送機能を実現する一種のサーバとしても動作します。

第一世代ファイル共有ソフトでは、ノードの発見やファイルの検索にクライアント／サーバ方式を使用しており、基本的には FTP でファイルを転送するときと大差はありません。しかし、ファイルの転送にはサーバを介さず、ファイルを探しているノードがファイルを持っているノードと直接ファイルを受け渡す方式をとっており、この部分が P2P らしいところです。

第一世代のファイル共有ソフトのように、クライアント／サーバ方式と P2P 方式を組み合わた形態を、ハイブリッド型 P2P 方式と呼びます。

第一世代ファイル共有ソフトのその後

第一世代ファイル共有ソフトの代表格である Napster は、詳細が非公開のクローズドなシステムでした。しかし、概念やプロトコル自体は簡単なものだったため、Napster を基にしたオープンな第一世代ファイル共有ソフトがいろいろと開発されました。特に、Napster 互換のオープンなシステムである OpenNap というものが発展をして、互換性のあるさまざまなファイル共有ソフトが生まれました。WinMX も、この OpenNap プロトコルを用いる Napster 互換の第一世代ファイル共有ソフトウェアです。

これらの第一世代ファイル共有ソフトも時とともに技術が進み、特にファイルの転送については、あとの世代のファイル共有ソフトと同様なレジューム機能（ファイルの転送が失敗したときに途中から開始できる機能）や多重ダウンロード機能（同じファイルを複数のノードから同時にダウンロードする機能）などが付け加えられていきます。

第二世代ファイル共有ソフト

システムの堅牢性という点で考えると、第一世代のファイル共有ソフトはノードの発見とファイルの検索をクライアント／サーバ方式で行っているため、中央に位置するサーバがシステムの弱点となります。つまり、サーバがダウンするとシステム全体が動作不能になるという問題点があります。そこで次に登場するのが、Gnutellaに代表される第二世代のファイル共有ソフトです。

Gnutella は、Nullsoft の Justin Frankel と Tom Pepper により開発され、2000 年 3 月に公開されました。Gnutella が画期的だったのは、ノードの発見やファイルの検索部分にも完全に P2P 方式を採用していたことです。このように、固定的な特定のサーバの機能に頼ることなく、どのノードも完全に対等な形態を、ピュア P2P（純粋な P2P）と呼びます。

このソフトウェアはほんの一瞬（たった 1 日！）公開されただけでしたが、その画期的な方式により瞬く間に話題になり、すぐにプログラムが解析されて、互換プロトコルが広まっていったのです。Gnutella に代表される第二世代のファイル共有ソフトは、海外では 2005 年の現時点でも主流です。KaZaA というファイル共有ソフトもそのひとつです。

ノードの発見

第二世代のファイル共有ソフトは、ノードの発見にも P2P 方式を使用します。ただし、一番はじめだけはどこに接続するかという情報がないと、P2P ネットワークに参加しようがありません。動作していそうな、どこか別のノードのアドレスをシード情報（最初に接続するノードの情報）としてあらかじめ持つことになります。

第一世代ファイル共有ソフトも、最初に接続するサーバの情報をシード情報と考えれば同じようなものです。しかし第二世代は、各ノードの役割が完全に対等であり、特定のノードがすべてのノードの情報を持つことはありません。

ピュア P2P 方式の第二世代ファイル共有ソフトでは、どこかひとつのノードに接続できれば、そこから数珠つなぎ式に別のノードを教えてもらい、他ノードを発見することができます（図 1-7、図 1-8）。

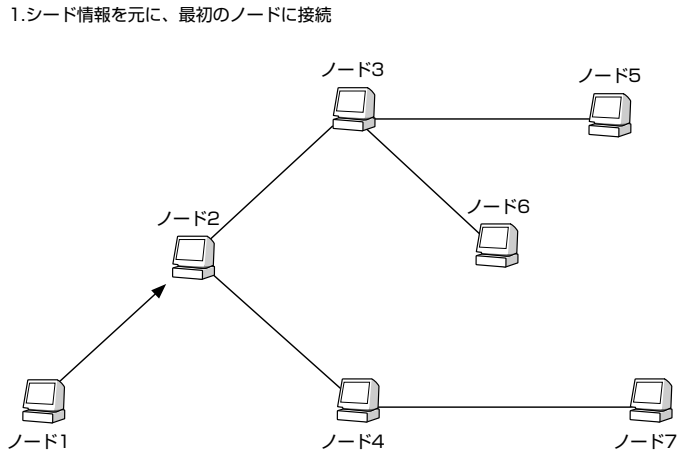


図 1-7　ノードの発見―第二世代ファイル共有ソフトの場合

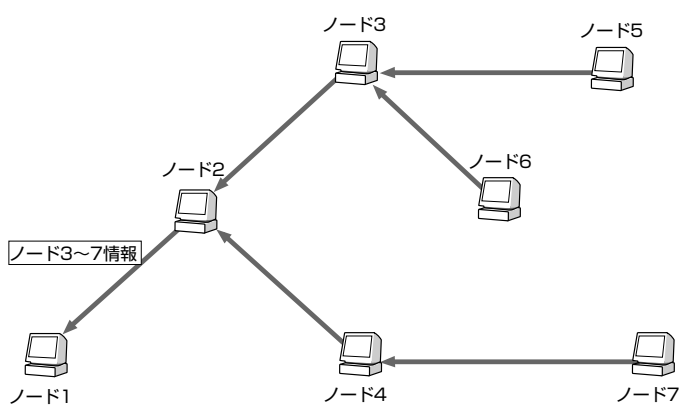


図 1-8　伝言ゲームのようにして離れたノードの情報を得る

ピュアP2Pでは、自ノードに他のノードが接続してきたとき、自分が知っているさらに別のノードの情報を教えることになります。この方法で、それぞれのノードは周辺のノードしか把握していなくても、知り合いの知り合いを追いかけてシステム内の広範囲なノードの情報を得ることができます。芋づる式にノードを順にたどるというこの方式は、ファイル検索機構でも最大限に利用され、第二世代ファイル共有ソフトの重要な考え方になっています。

もし1つのノードが多数のノードからの接続を受け付けてサービスをすることになれば、結果的にクライアント／サーバと同じような形態になってしまいます。そこでピュアP2Pの場合は、負荷分散の意味あいから他のノードからの接続数を制限し、なるべく特定のノードに負荷が集中しないように設計するのが一般的です。

また、中心となるサーバのない第二世代ファイル共有ソフトでは、他のノードがダウンしたことをどのように検出するか、またノード総数をどのように把握するのかが問題になります。これは多くの場合、次節で説明する擬似的なブロードキャスト――フラッディング（flooding）により実現されます。

ファイルの検索

第二世代ファイル共有ソフトは、ファイルの検索にもP2P的な方法を使用します。その基本的な実現方法は、擬似的なブロードキャストです。

ピュアP2Pでは、各ノードは網の目状につながっています。知り合いの知り合いはみな知り合いという考え方で、ファイルの検索を隣のノードに依頼し、隣のノードは検索をさらにその隣に依頼するということを繰り返していけば、いつかはネットワーク上のすべてのノードに対して検索を行うことが期待できます。このようにいくつものノードを介して、検索依頼が伝播していくことを、ここでは「擬似的なブロードキャスト」と呼ぶことにします。

図1-9はGnutellaの検索を例にとったものですが、あるノードがいくつかの隣接ノードに検索を依頼すると、その依頼はさらにその隣のノードへと伝播していきます。ファイルの検索には次のようなルールが適用されます。

- すでに検索時に経由したノードには検索をしない。
- いくつかのノードを経由したらそこで検索を打ち切る。

これらのルールをどのようにして実現しているかというと、検索依頼のそれぞれ

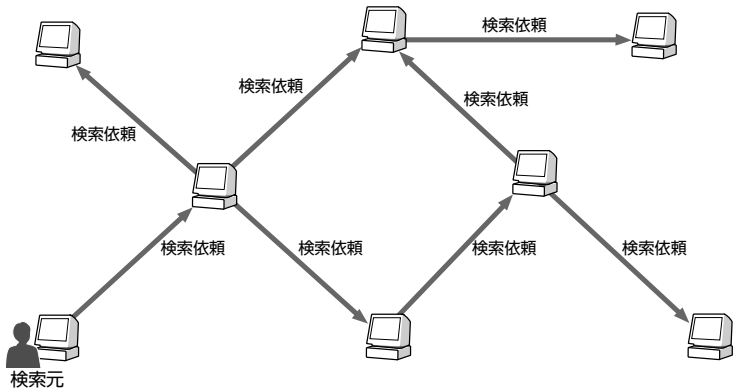


図 1-9　ファイルの検索―第二世代ファイル共有ソフトの場合

に重複のないように ID を割り当て、各ノードはその ID を見て同じ検索依頼が何度も同じ経路をたどらないように転送するのが一般的です。

1 番目のルールは、すでに扱ったことのある ID の検索依頼が再び到着したら、検索依頼がループしていると解釈し、検索依頼の送信を打ち切れば、実現できます（図 1-10）。

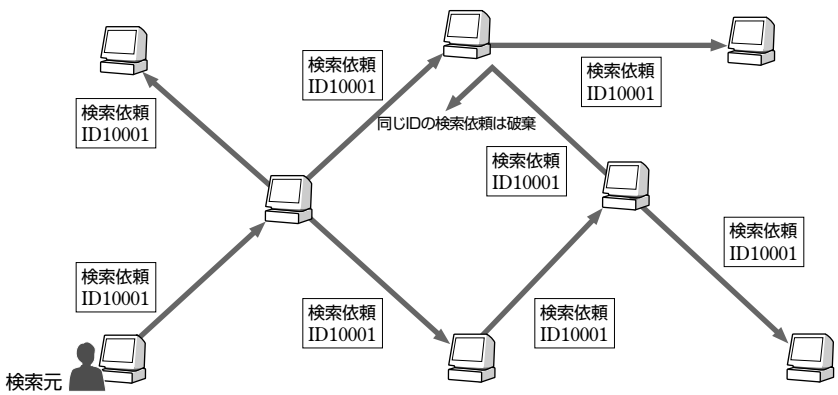


図 1-10　ID を付けて検索依頼を出す

2 番目のルールに関しては、検索依頼の中にカウンタ（ホップ数カウンタ）を設け、ノードを経由するたびに減少するようにすれば実現できます（図 1-11）。

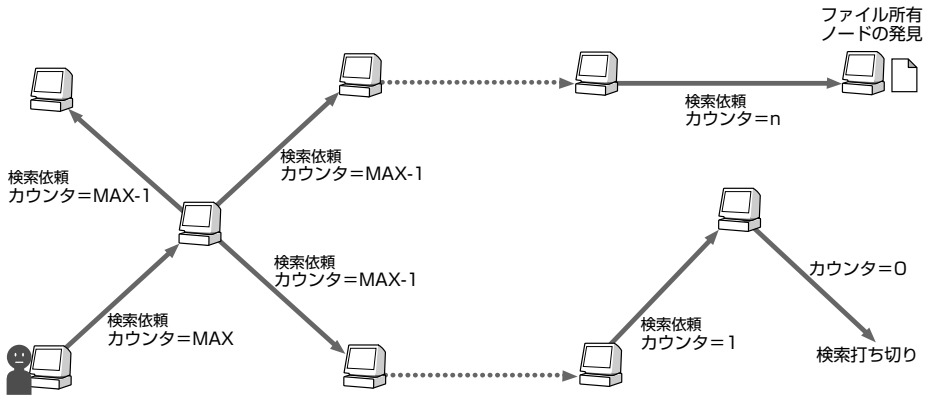


図 1-11　ホップ数カウンタが尽きると検索は打ち切り

検索の途中で目的のファイルを見つけたら、検索で通ってきたルートを遡って検索の依頼元に検索結果を戻します。ファイルの検索は、このような方法により実現されます。検索ルートを遡る際にも、各検索依頼に ID を添付して各ノードが検索依頼をどちらに送信したかを記憶しておけば、それを元にルートを遡ることができます。

ファイルの転送

このようにして、第二世代のファイル共有ソフトでは特定のサーバなしにファイルを検索しますが、その後の転送方法は第一世代と同じです。検索結果から目的のファイルの存在位置がわかるので、ファイルをアップロードしているノードとの間に転送用のコネクションを別に確立し、それを使ってファイルを転送します。Gnutella の場合は、HTTP のように、ファイルの転送が生じるたびにアップロードノードに接続し、転送が終了するとコネクションを切断します。

なお、初期の Gnutella にはレジューム機能（ファイルの途中から転送を開始する）はありませんでしたが、その後のファイル共有ソフトでは一般的になっています。

初期のころのレジューム機能は、転送が中断するとファイルの途中から転送を再開するという単純なものでしたが、近年のファイル共有ソフトが備えているものはもうすこし複雑になっています。ファイルをブロックに分割してブロック単位に転送できるようにしておき、ファイルの先頭から順番にダウンロードするだけでなく、いくつかのノードから並行してダウンロードできるように、任意のノードから任意

のブロックを任意の順番で転送できるようになっています。これは Winny も同様です。詳しくは 4 章「実装」で説明します。

第二世代ファイル共有ソフトのその後

第二世代ファイル共有ソフトの発端となった Gnutella も、ピュア P2P の形態を追求した結果、「ネットワーク規模が大きくなると破綻する」という問題が明らかになってきました。

Gnutella の検索は、検索依頼の伝播という擬似的なブロードキャストに頼っているため、あるノードが出した検索の問い合わせを全ノードに到達させる必要があります。そうでないと、検索できないファイルが Gnutella ネットワーク内に存在することになってしまうからです。しかし、すべてのノードからの問い合わせをすべてのノードに伝播させようとすれば、ノード数が大きく増えたときに、検索の伝播だけでネットワークは飽和してしまいます。

たとえば、P2P ネットワーク内に 8 つのノードがあるときにすべてのノードが検索依頼を送り出せば、他の 7 つのノードで検索が発生するので、検索依頼は 8×(8-1) 個発生します。ネットワーク内に 100 万ノードあったらどうなるでしょうか？　ノード数の増加に対して 2 乗規模で検索依頼が増えるので検索依頼の伝播だけでネットワーク帯域を使いはたしてしまいます。

また検索時間が長くなると、その間にオフラインになるノードもあるので、検索の失敗率が上がるという問題も生じます。だからといって、探索ホップ数を減らして遠方のノードを検索しないようにすれば、遠方のノードにあるファイルを検索できなくなります。基本的に全ノードに対して検索をしようとするピュア P2P の形態そのままでは、ネットワークが大きくなったときにどうしても問題が生じるのです。

この対策として、近年は P2P ネットワークを階層構造にするなど、いくつかの改良が採り入れられています。例として次のようなシステムがあります。

- 第一世代ファイル共有ソフトのモデルでありながら、検索サーバをピュア P2P の要領でつなげ、検索能力を強化する（WinMX の WPNP）。
- 第二世代ファイル共有ソフトのモデルにノードの親子関係を導入し、検索を上位の階層に任せる（KaZaA に代表される FastTrack 系）。

初期は非常に簡単な形態であった Gnutella にも同様の概念が導入され、Gnutella 2

として現在も発展しています。この擬似ブロードキャストによる大域検索は重要な技術であり、今後の第二世代ファイル共有ソフトでも改良が続けられていくでしょう。KaZaA の開発者が生み出した Skype のような P2P 方式の IP 電話は、この技術の応用だといえます。

第三世代ファイル共有ソフト

本書では、キャッシュ機構を備えている P2P ファイル共有ソフトを第三世代ファイル共有ソフトとして分類しています。キャッシュ機構を備えていることによって、匿名性も実現できます。第三世代ファイル共有ソフトの草分けとなった Freenet が提唱されたのは意外に古く、1999 年にまで遡ります。

Freenet の開発の発端となったのは、アイルランドの Ian Clarke が著した論文、“A Distributed Decentralised Information Storage and Retrieval System：非集中分散型の情報の保存と検索のシステム”であり、この概念を実際に動作する形で実現したのが Freenet のソフトウェア群です。

Freenet の提案はたいへん画期的なもので、私にとっては「目から鱗」でした。そして、ここから着想を得て Winny を開発するに至るわけですが、Freenet の特徴は、なんといっても「技術的に匿名性を実現できる」と主張していることです。

ここでは、匿名性を情報の第一発信者がわからないという意味で使うことにします。新聞やラジオの番組でも情報提供者が匿名で発言することがありますが、これは新聞社や放送局が匿名を保障することによって成り立っています。匿名を保証する人や組織は一次情報源を知っているわけで、完全な匿名とはいえません。完全な匿名というのは実際にはありえないだろうと思われていました。そんななかで、Freenet は技術的に匿名性を実現できるはずだという可能性を示したのです。

Freenet そのものは、Ian Clarke の提案を形にしたソフトウェアの一例であり、その目的とするところは、誰にも規制されたり検閲されることのない自由な発言が可能な場を提供することです。Freenet はまだ開発途上にありますが、単なるファイル共有ソフトで実現したファイルシステムを基にして、BBS 機能や Web 機能を載せた複合システムとなっています。

ノードの発見とファイルの検索

第三世代ファイル共有ソフトにおけるノードの発見方法は、第二世代と同様です。

同様に、ファイルの検索にも第二世代のピュア P2P 方式が採用されます。隣のノードに検索を依頼すると、そのノードからさらにその隣のノードに検索が行われる、という具合に検索は進められます。

キャッシュ機構を活用したファイルの転送

第三世代ファイル共有ソフトの特徴となっているのは、ファイルの転送方式です。第三世代のソフトウェアはまだ実装例が少ないので、主流になるかどうかははっきりしませんが、ファイルの転送にキャッシュシステムを用いていることがこの世代の大きな特徴だと私は考えます。

キャッシュには一時的に情報を蓄える貯蔵所（Cache）という意味がありますが、ここでは送受信時に各ノードに蓄えられる共有ファイルの断片、あるいは断片を収容しているバッファという意味で使うことにします。

キャッシュシステムを利用すれば、P2P ネットワークは全体的に効率よく情報を共有できますし、匿名性を高めることができます。匿名性を実現する技術の本質は、つまるところ「たくさんの人を介して情報が渡されること」だと考えました。一次情報を発信しているノードから、その情報を入手しようとしているノードまでの間に多くのノードを経由するようにすれば、ファイルの受信者は情報発信元からの転送なのかキャッシュの転送なのかが区別できないので、匿名性が高まります。キャッシュシステムは、その情報（ファイル）の通過場所となるわけです。

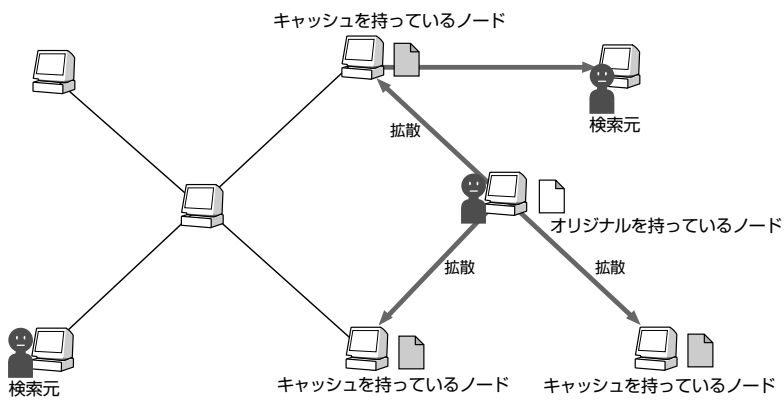


図 1-12　P2P ファイル共有ソフトとキャッシュ

第三世代のファイル共有ソフトでは、ファイルは適当な大きさの断片に分けて暗号化され、ノード間を転送されます。ファイルの断片は転送時の形式のまま各ノードに蓄えられるので、キャッシュがノード間を渡り歩いていると見ることができます。

Freenet も Winny も、P2P ネットワーク内にキャッシュを広めていくという点は同じです。しかし、そのコンセプトと実現方法は異なっています。

キャッシュという仕組みによって匿名性を実現するとき、たくさんのキャッシュファイルを P2P ネットワークに広めるほど匿名性が高くなります。Freenet は匿名性を重視し、ファイル断片を無作為にネットワーク内に拡散しますが、この方法はネットワークやノードへの負担が高く、利用者にはほしいデータがなかなか届かないように思えました。Winny の場合は、Freenet ほどの匿名性はありませんが、ネットワークやノードへの負担が少なく、効率がよい方法を採っています。

この設計コンセプトの違いについては、2.1「Winny の開発コンセプト」でもうすこしまとめて解説することにします。

1.4 まとめ

この章では、Winny の背景にある P2P 技術と P2P 方式のファイル共有ソフトについて見てきました。

- P2P とは、対称的な役割をはたすコンピュータの集まりでサービスを実現するようなネットワーク通信形態であり、クライアント／サーバとは対照的なモデルです。
- P2P 型のファイル共有ソフトは P2P アプリケーションのひとつですが、「P2P ＝ファイル共有ソフト」ではありません。
- ファイル共有ソフトは、技術的な特徴から第一世代、第二世代、第三世代と進化してきています。
- Winny は、Freenet を参考にして生まれた第三世代ファイル共有ソフトです。
- Winny が目指したのは、匿名性と情報共有効率の両立です。

Winny紹介

2章

1章「P2Pの基礎知識」では、Winnyを理解するうえで必要となるP2P技術やファイル共有ソフトについての基礎知識を解説しました。この章では、Winnyの設計コンセプトをまとめます。

2.1 Winnyの開発コンセプト

Winnyの開発コンセプトをまとめると、次の3つになります。

- **匿名性（プライバシーの保護）を実装したファイル共有ソフトであること**
- **ファイルの共有効率がよいこと**
- **Windowsネイティブプログラムであること**
  （Freenetは Javaで実装された）

Freenetを出発点として考えたので、上記の要件はそれを元にしたものになっています。1つ目は、Freenetのように匿名性を備えているということです。また、Freenetの共有効率は悪いと考えていたので、2つ目にその改善を技術的なテーマとして挙げています。3つ目にWindowsネイティブプログラムであることをわざわざ挙げているのは、利用者に受け入れられるソフトウェアの要件として、インストールが容易で、快適な実行速度で動作することも重要だと考えていたためです。それは、FreenetがJavaで書かれていることに対する私なりの考えでもあります。

それまでリアルタイムゲームやシミュレータのようなものばかり作ってきた経緯もあって、私はパフォーマンスを重視する傾向が強く、インタープリタ系や仮想マシン（VM）を用いる言語はあまり好きではありません。これは個人的な好みでしかないのですが、これに加えてもうひとつ、このあと述べる匿名性の仕組みと効率を両立させるためには、最終的にプログラムをカリカリとチューニングしていかざるをえないだろうという直感もあったからです。この2点から、WindowsネイティブなプログラムであることをWinnyの開発コンセプトに加えています。

なお、本書では私の理解にしたがって、匿名性は「情報の第一発信者を隠すことによりプライバシーを保護すること」、共有効率は「ほしい情報ができるだけ早く得られること」と定義することにします。

さて、以上のコンセプトに沿ってさらに基本設計を進めていくのですが、まず考えるべきことは、なぜ Freenet の情報共有効率が悪いのか、それを解決するにはどうすればよいかということです。

Freenet の匿名性はどのように実現されているか

Freenet の匿名性は、情報断片の拡散という手法によって実現されています。まず、公開する情報を細かいブロックに分断し、それぞれの中身がわからないように暗号化を行ったうえで、周辺のノードに送り出して拡散させておきます（図 2-1）。

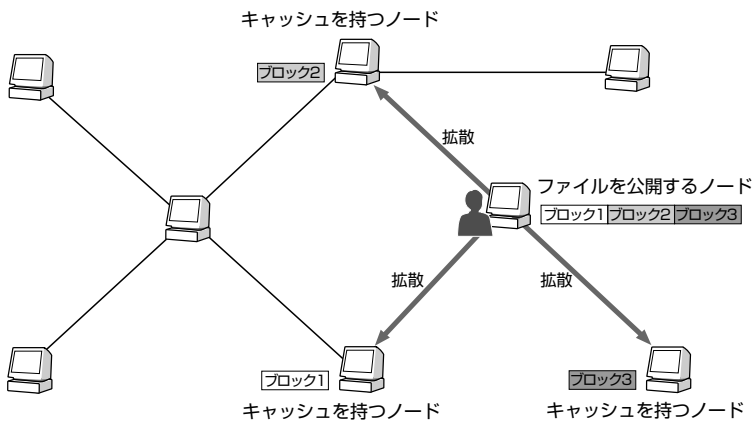
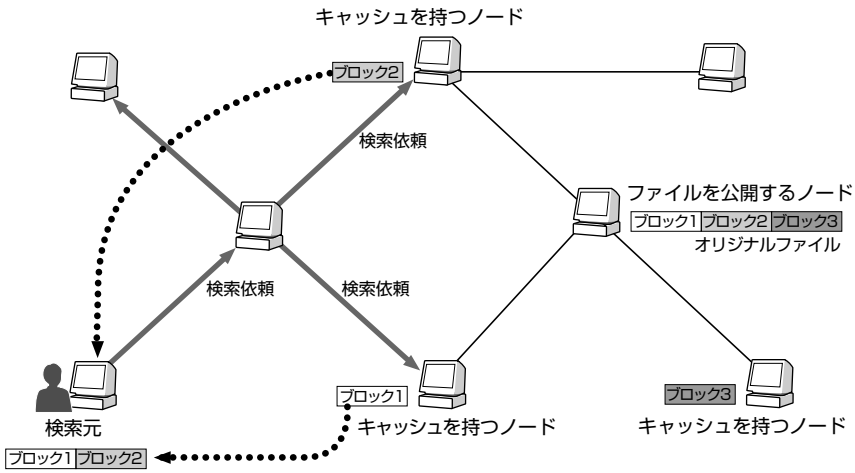


図 2-1　Freenet では公開する情報をあらかじめ断片化して拡散する

一方、情報を得るときは、ネットワーク内に検索をかけて検索範囲にたどり着いたブロックをかき集め、全部揃った時点で復号します（図 2-2）。情報の断片はノードからノードへと拡散していくので、断片を受け取るノードは送信者が情報の第一発信者かどうかを判断することはできず、断片をかき集めている最終的な受信者もむろん第一発信者が誰かを知ることはできません。

しかしこの方式の場合、ネットワークの規模が大きくなると、公開された情報を広いネットワーク内に行き渡らせるのは難しく、また検索元のノードが出した検索依頼を広いネットワーク内のすみずみに届かせることも現実的ではありません。ネットワークの通信速度や各ノードのディスク容量には限界があるためです。結局、公開されたファイルはほしい人の手元にはなかなか届かないでしょう。

この方法はいわば、メッセージをちぎって詰めたビンを広い海に向かってたくさん放ってやれば、そのうち誰かに匿名で情報が渡せるだろうというような気の長い方法です。匿名性は、ファイルの断片をたくさんのノードに広めるほど高くなるので、Freenet の方法はたしかに匿名性は非常に高いのですが、あまりにも効率が悪すぎるように思えました。



検索元は検索依頼の届く範囲にあるファイル断片を入手できる

図 2-2　検索をかけて断片化した情報をかき集める

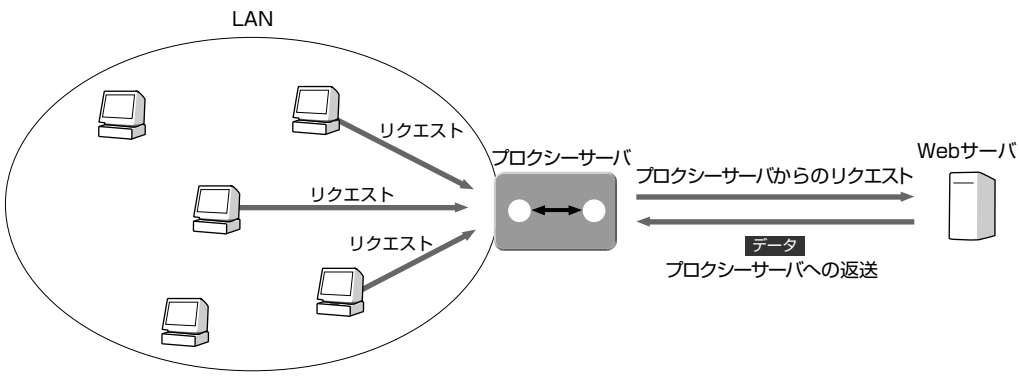
Freenet の匿名技術の本質を考える

Freenet の匿名性を実現しているものが何かを考えたとき、それは情報の拡散ではなく、情報の多段中継だという考えに私はたどり着きました。情報を直接やり取りするのではなく、間接的にやり取りすることこそが匿名技術の本質であり、Freenet で用いられている暗号化や情報拡散などの手法は本質ではないと考えたわけです。

この結論にたどり着けたのは、プロクシーサーバという匿名性と効率性とを兼ね備えた既存技術があったからです。プロクシー技術についてはこのあと触れますが、ともかく、Winny の初期設計ではこのプロクシーサーバの概念が参考になりました。

匿名機構としてのプロクシー技術

プロクシーサーバはおもに、ファイアウォールで隔てられたローカルネットワーク内のノードを、外部ネットワークと通信させるときに使用します。これは、ローカルネットワーク内のノードから外部のノードに宛てたコネクションをプロクシーサーバがいったん受け取り、新たに開いた別のコネクションに通信内容を中継して実現しています。このため、外部のノードからはプロクシーサーバが通信相手として見え、ローカルネットワーク内にある本来の通信相手はわかりません（図 2-3）。



プロクシーサーバはLAN内からのリクエストをいったん終端し、Webサーバへの新たなコネクションに中継する。接続先のWebサーバの通信相手はプロクシーサーバで、LAN内の本当の通信相手は見えない（匿名性が保たれている）。

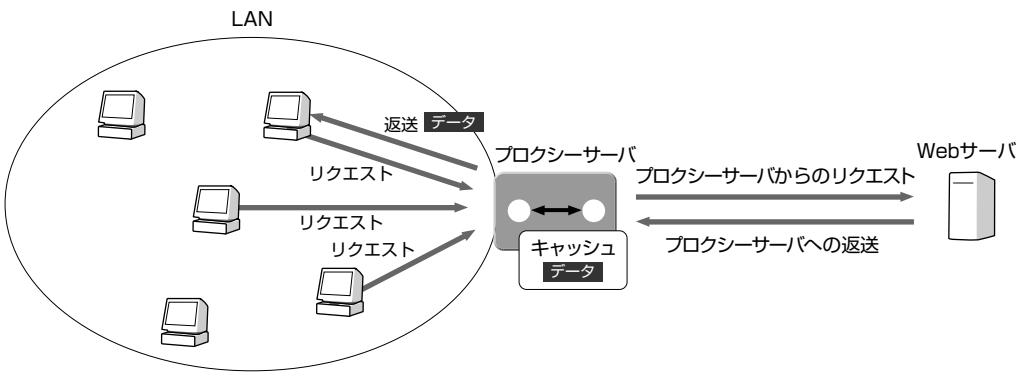
図 2-3　プロクシーの匿名化技術

インターネット通信の仕組みでは、相手ノードの IP アドレスがわかるようになっているので、相手に IP アドレスを隠して通信することはできません。しかし、双方のノードが中継ノードを介して通信するときには、中継ノードの向こう側にあるノードの IP アドレスを知ることはできなくなります。

プロクシー技術の利用目的のひとつは、セキュリティ上の理由から LAN 内のノードの IP アドレスを外部のネットワークから隠すことですが、結局のところ、この匿名性の本質は中継機能によるといえます。

キャッシュ機構としてのプロクシーサーバ

プロクシーサーバには、複数のノードからリクエストのあった情報を効率よく配送するというもうひとつの効果があります。プロクシーサーバは Web サーバへのアクセス中継によく利用されるので、ここではそれを例に説明しましょう。図 2-4 はプロクシーサーバのキャッシュ機能の概念を説明したものです。



リクエストしたデータがプロクシーサーバのキャッシュにあれば、Webサーバに問い合わせをせずにキャッシュのデータが返送される。このため、効率よくデータを配布できる。

図 2-4　プロクシーサーバのキャッシュ機能

一般的に、プロクシーサーバにはキャッシュ機構があり、クライアントがプロクシーサーバを経由してファイルをダウンロードすると、プロクシーサーバは中継時にファイルの内容をキャッシュに保存します。このため、別のクライアントから同じファイルを要求されたときには、キャッシュ内のファイルをクライアントに送信するだけですみます。同じファイルを Web サーバから何度も取り寄せなくてもよいので、キャッシングにより通信効率の向上が望めます。

プロクシー技術をファイル共有ソフトに応用する

Winny は、プロクシーサーバにアイデアを得て設計したものです。しかし、例として HTTP プロクシーサーバを引き合いに出したように、プロクシーサーバの概念はおもにクライアント／サーバ方式を前提としたものでした。それを P2P ファイル共有ソフトに持ち込むとどうなるのか？　それが、Winny の基本設計のかなめとなりました。すなわち、プロクシーサーバを内蔵したファイル共有ソフトというものを思いついたのです。この方法で、匿名性と効率を両立できるのではないかと考えました。

Winny は、第二世代ファイル共有ソフトにプロクシー機構を統合することによって実現しています。匿名性を実現する概念は Freenet から得ましたが、設計や実装は Freenet とは異なったものになっています。

Freenet と Winny の設計コンセプトを比較する

Freenet と Winny の設計コンセプトの違いをまとめると、Freenet では情報発信者のプライバシーを護るために匿名性を最重視し、効率にはあまり配慮していません。また、匿名性を実現するために拡散という方法をとっています。Winny ではどちらかというと匿名性よりも効率を重視し、匿名性を実現するために中継という方法をとっています。

Freenet の拡散方式

Freenet の拡散方式では、ファイルそのものを細分化し、周囲のノードにすこしずつ送り込んでネットワーク内に伝播させます。これは、ファイルが実際に要求されているかどうかとは関係なく行われます。ファイルを探すユーザーは、ネットワーク内に散らばっているファイルの断片自体を検索の対象とし、見つかった断片を集めて元のファイルを復元します。

Winny の中継方式

一方、Winny の中継方式の設計ポイントは 2 つあります。

まず 1 つ目は、ファイル自体の代わりにあらかじめサイズの小さいキーをネット

ワーク内に拡散させておくことです。キーはファイルの内容や存在場所を要約したもので、ちょうどWinnyネットワーク内のファイルのインデックスとして機能します。ユーザーはキーを検索し、見つかったキーに含まれている所在情報からファイルを持っているノードを知り、そのノードからファイルの本体を転送させます。

もうひとつのポイントは、ネットワーク内に中継方式でファイルのキャッシュを作ることです。これは、他のノードからの要求に応じてファイルを転送するとき、経路上の中継ノードに一定の確率でファイルのキャッシュができるというものです。さらに、ファイルをダウンロードしたノードも以降はそのファイルを公開するので、中継ノードとして機能します。プロクシーサーバのキャッシュと中継の概念は、このような形でWinnyに活きているのです。

Freenetの拡散は、周囲のノードに対してファイル断片を強制的に送りつけるプッシュ型の方法です。一方Winnyの中継方式では、需要があったときに要求に応じてファイルが取り寄せられ、ノードに蓄えられます。またキーの拡散も、周囲のノードが余裕のあるときにキーを要求して取り寄せており、Winnyはファイルの転送にもキーの拡散にもプル型の方法をとっています。このためFreenetと比べると、ネットワークやノードにかかる負担は少なく、効率はよいといえます。

この具体的な仕組みは、3章「Winnyの仕組み」で説明します。

開発の過程をたどる

Winny Version 1（Winny 1）の実装が進み、β 1として初めて公開したのは2002年5月6日のことです。簡単な開発履歴を**表2-1**にまとめます。

表2-1　Winny 1の開発履歴

2002年4月1日	Winnyの開発を宣言
2002年5月6日	Winny 1のβ 1初公開
2002年7月25日	Winny 1のβ 18.0で簡易BBS機能を追加
2002年12月30日	Winny 1の正式版公開（β 22.52がそのままバージョン1.0へ）
2003年4月7日	Winny公開サイトを閉鎖（最終版はバージョン1.14）

それから約1年後に、Winny 1はその開発と公開を終えます。

匿名性を備えた共有効率のよいファイル共有システムというものが技術的に可能かどうかは、やってみないとわからないことでした。P2P型のソフトウェアは、ユー

ザーの操作や接続ネットワーク環境がまちまちな多数のノードが協調して動作するものなので、システムの動作に影響する可変要素が多く、また Winny の場合は同様の設計事例もなかったので、予想が難しかったのです。

Winny の開発目的は技術的な検証にあったので、正式版をリリースしたところで、ファイル共有ソフトとしての Winny 1 の開発は終了しました。しかし Winny 1 の開発を終えたすぐあとに、Winny ネットワークを活用して 2 ちゃんねるのような大規模な掲示板を実現できるのではないかと考え、新しく Winny Version 2（Winny 2）プロジェクトが始まることになります。

コラム Freenet について

第三世代ファイル共有ソフトについては 1 章「P2P の基礎知識」でも簡単に紹介しましたが、ここでもうすこし詳しく説明します。

すでに述べたように、Freenet の開発はアイルランドの Ian Clarke の論文が発端となっています。言論弾圧への対抗手段として匿名出版システムというものを考え、その設計例として開発されました。Freenet は、論文で論じられた設計の一実装例にすぎませんし、開発には Ian 以外にも多くの方がかかわっています。

サービス基盤としての匿名情報共有システム

Freenet は、単なるファイル共有ソフトというよりも、さまざまなサービスのインフラとなる匿名システム基盤です。このため正確には、Freenet は基盤機構の部分を指し、他のサービスが Freenet のネットワーク上で動作する形になります。たとえば、Frost というクライアントソフトウェアは、Freenet 上で匿名 BBS 機能などを実現しています。また、Freeweb という匿名 Web システムを実現するソフトウェアも存在します。

Freenet には、標準ではファイルの検索機構がありません。利用者は、キーと呼ばれるファイルのインデックス情報（Winny でいうハッシュ値のようなもの）を知っていることが前提で、このキーを指定して情報の中身を取得します。そのため、キーを共有・検索するシステムが別に必要になります。Frost もそのひとつです。

Freenet の実装の特色

Freenet は、大きなファイルの内部に情報断片を暗号化して格納・管理します。ファイル断片を格納するためのストレージ機能を、ファイルシステムとは別に持っていることになります。このような方法によって、Freenet の実装は徹底的に匿名性を追求

しています。「匿名性はないよりあったほうが良い」という程度の Winny とは、コンセプトの異なるシステムだといえます。

現在の Freenet は Java 言語で記述され、Java 環境で動作します。しかし、これは実装例のひとつにすぎず、Java で実装しなくてはならないわけではありません。実際、C 言語で再構築したシステムも存在します。ドイツの第三世代ファイル共有ソフトである Entropy がそれです。

Freenet は Winny よりも多くの部分に暗号技術を使用していますが、匿名性を実現する基本的な仕組みは、すでに述べたとおりです。ファイルを断片化し、それを暗号化してキー情報を知っている利用者のみファイルを復元できるようにしたうえで、各ノードのストレージに拡散させます。これは、初期情報公開者が暗号化したファイル断片を周辺のノードに押し込むという方法で実現されています。

執筆時点で、Freenet の最新バージョンはテスト版の Version 0.7 です。しかし、いまもオープンソースで開発が進んでおり、効率はこれから良くなっていくと思われます。今後も注目すべきソフトウェアでしょう。

基盤技術としての P2P ファイル共有システム

このテーマについては、3 章「Winny の仕組み」でもうすこし掘り下げるつもりですが、P2P ファイル共有システムは巨大な分散ファイルストレージと考えることができます。つまり、1 つのファイル格納庫を多数のノードが協調して維持しているというイメージです。ファイルストレージはすべてのコンピュータサービスの基本的なインフラであり、そのインフラを利用すれば BBS システムを比較的容易に作成することができます。事実、簡易なものでしたが、Winny 1 でも BBS 機能を実験的に作り込んでいましたし、これを発展させれば大規模 BBS も実現できるように思われました。

日本製の P2P ファイル共有システムで、ある程度の規模のノードを確保しているものは Winny しかない状況だったので、Winny のファイル共有ネットワークを利用して大規模 BBS を実現しようと、Winny 2 プロジェクトは始まったのです。

Winny 1 から Winny 2 へ

Winny 2 は Winny 1 の上に大規模 BBS を実現させることが目的でしたので、Winny 1 のファイル共有機能はそのままにし、プラスアルファの機能で大規模 BBS を実現させる予定でした。しかし、Winny 1 で後回しにしていた GUI（Graphical User Interface）の強化などもあり、結局半分以上のプログラムソースコードを放棄して、Winny 2 の開発は始まったのです。

その簡単な開発履歴を**表 2-2** にまとめます。

表 2-2　Winny 2 の開発履歴

2003 年 4 月 9 日	Winny 2 の開発を宣言
2003 年 5 月 5 日	Winny 2 の β 1 を公開
2003 年 11 月 27 日	Winny 2 の BBS 上で著作物の放流を予告し、Winny 2 で公開した 2 名が逮捕される
	警察の家宅捜査により開発停止。公開していた最終バージョンは Winny 2 β 7.1
2004 年 5 月 10 日	Winny 作者（私）が著作権法違反幇助の疑いで逮捕される

Winny 2 は、BBS のブラウザ機能を備えていることが Winny 1 との大きな違いです。また、大規模な BBS に発展したときにもレスポンスが低下しないように、BBS にスレッドを立てた公開者を BBS の閲覧者とは別のネットワークとして分離するようになっています。補足的なことですが、Winny 2 では GUI の変更を目的として、開発環境を Visual C++から C++Builder へと変えています。

Winny 1 と Winny 2 にはネットワーク的な互換性がなく、Winny 1 のユーザーと Winny 2 のユーザーが直接ファイルを共有したり掲示板を構成することはできません。ファイル共有には Winny 1 を使用し、分散 BBS の開発に興味があるテスト協力者だけが Winny 2 の β を使用して開発に参加することを意図していたのです。Winny 2 では、最終的にシステムの管理を可能とすることも目指していましたが、現在は開発を続けることができません。ウィルス対策ができないこととあわせて、残念な点です。

コラム Winny と 2 ちゃんねる

Winny は、大規模掲示板である 2 ちゃんねるのユーザーをテスト協力者として開発されました。

2 ちゃんねるは、誰もが匿名で発言できる掲示板で、ときには怪しい解説や不愉快な発言が飛び出すこともありますが、所属や立場を越えて自由に（そして愉快に）意見を交換できる貴重な場でもあります。なんといっても 2 ちゃんねるには圧倒的な数の人がいますから、発言の量も膨大です。実際、どうでもいいようなものも多いのですが、しかしそのなかには非常に貴重な意見も含まれています。Winny の開発でも、技術的に質の高い意見を 2 ちゃんねるの住人（2 ちゃんねらー）から多数いただきました。また、P2P 型のソフトウェアは、利用者のコンピュータが相互に維持するネットワークがあってはじめて機能するものなので、開発したソフトウェアを実際に検証するのは難しいという面がありますが、最初の Winny ネットワークはまず 2 ちゃんねらーの協力により構築され、テストされてきたのです。

2.2 まとめ

この章では、それまでのファイル共有ソフトと比較しながら Winny の技術的な特色を見てきました。

- Winny は、Freenet に感銘を受けたことから作成した第三世代ファイル共有ソフトです。
- Freenet の設計コンセプトでは匿名性を最重視していますが、Winny は匿名性よりも情報の共有効率を重視しています。
- ファイル共有ソフトとして作られたのは Winny 1 です。これは、2003 年に開発・公開を終了しています。
- Winny 2 は、大規模匿名 BBS を目指して開発が始められた、Winny 1 とは別の P2P アプリケーションです。

［ユーザーの目から見たWinny］

ここでは、Winny を使ったことのない読者のために、Winny の基本的な機能を使ってファイルを公開したり、目当てのファイルを探し出してダウンロードするまでのようすを紹介します。例示する画面や操作手順は Winny 2 のものです[*1]。

以下の画面は、Winny を起動したところです。上部にあるのは、ファイルを検索するときにキーワードを入力するためのテキストボックスです。その下には表示モードを切り替えるタブが並んでおり、上段では検索で見つかったファイルを表示する［ファイル検索］画面が、下段では他のノードへの接続状況を示す［ノード接続］画面が選択されています。

*1 Winny を利用するときにはアンチウィルスソフトなどが動作していることを確認してください。また、Winny を利用した結果生じたいかなる障害にも、著者およびアスキーは責任を負いかねます :-)。

Winny の画面

1. **Winny のネットワークにつなぐ**

   Winny を起動すると、Winny はまず Winny ネットワーク内でオンライン中の他のノードに接続します。初回起動時だけは、ユーザーが接続先のノード

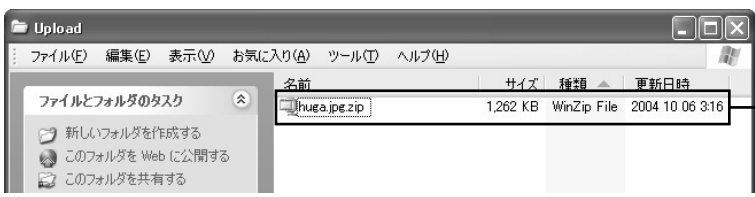
情報（初期ノード情報）をインターネット上の掲示板や情報サイトなどから入手して、Winny に教えてやります。接続相手のノードは、ユーザーがどんなジャンルに関心があるか、また Winny ネットワーク内のノードの稼動状況に応じて自動的に変わっていきますが、Winny の終了時にはその時点までに収集したノード情報が保存されるので、2 度目以降の起動時には自動的にオンライン中の他のノードを見つけて接続してくれます。

ノードへ接続したところ

2. **ファイルを公開する**

ファイルを公開するには、まず空のフォルダを用意し、アップロードフォルダとして指定しておきます。このアップロードフォルダに公開するファイルをコピーし*2、［フォルダ情報］タブにある［再チェック］ボタンをクリックします。公開の手順は、これでおしまいです。

*2 Winny をインストールしただけではアップロードフォルダは用意されないので、あらかじめ手作業でアップロードフォルダを指定してやります。



アップロードフォルダにファイルを置いたところ

3. **目当てのファイルを探す**

ファイルを検索するには、［検索単語］テキストボックスに探しているファイルの名前（一部でもよい）を入力します。［▲検索］あるいは［▼検索］ボタンをクリックすると、Winny は Winny ネットワーク内を検索し、指定した条件に合うファイルを一覧表示します。

ファイルの検索

4. **ファイルをダウンロードする**

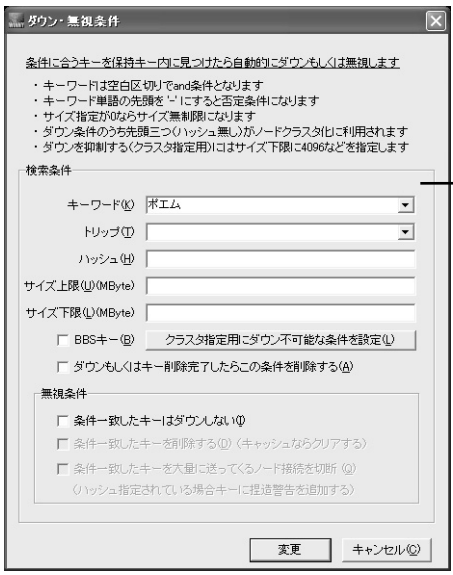
ファイルの一覧からダウンロードしたいものを選択してダブルクリックすると、そのファイルはダウンリストに登録され、ダウンロードが始まります。

ファイルは実際には、ブロックに分割して暗号化したキャッシュ形式でダウンロードされます。キャッシュのダウンロードが完了すると、ブロックは元の形に組み立てられて復元され、ダウンロードしたファイルの収納庫となっているダウンロードフォルダに保存されます。ユーザーは、ダウンロードフォルダ内のファイルを利用可能になります。

ファイルのダウンロード

以上が Winny によるファイル共有の基本的なステップですが、ユーザーは Winny の特色となっている自動ダウンロード機能を使って、特定の条件にマッチするファイルのダウンロードをまとめて Winny に任せることもできます。

自ノードからの検索が届く範囲にほしい情報があるかどうかは、刻々と変わっていきますし、1 回目の検索で目当ての情報が見つからなくても、2 回目には検索ルートが変わって見つかるかもしれません。手動であれば、目当てのファイルを入手できるまで（あるいはあきらめるまで）一連の検索操作を繰り返すことになりますが、これを Winny が代行してくれるので便利です。条件を指定して、コンピュータを Winny ネットワークについだまま放っておくと、時間をかけてすこしずつ条件にあうファイルが溜まっていくという使い方が、Winny の身上です。



検索条件や除外条件を指定して放っておくと、検索とダウンロードが自動的に繰り返され、関心のあるファイルが徐々に溜まっていく

自動ダウンロードを行うキーワードの指定

Winnyの仕組み

3章

Winny は、複数のノードが協調しながらファイル共有を実現しています。この章では、最も単純なファイルの流通を例に、公開されたファイルが他のユーザーにダウンロードされるまでの仕組みを追い、どのようにして匿名性のある効率のよいファイル配布が可能になったかを解説します。

3.1 Winny ネットワークの概観

Winny はピュア P2P 型のシステムなので、中心的な役割をはたす特定のサーバは存在しません。また、P2P ネットワーク内のノードやファイルすべてを把握しているノードもありません。近隣のいくつかのノードだけを知っているノードがインターネットを使って多数接続し、他のノードの情報やファイルの所在を交換しながらファイルの検索や転送を可能にしているのです（図 3-1）。

各ノードがインターネットを介して接続し、形成しているネットワークを、Winny ネットワークと呼ぶことにします。Winny ネットワークは、「インターネット上に Winny ノード相互が築いたアプリケーションによるネットワーク」だということができます。各ノードは利用者の都合によって起動したり停止したりするので、Winny ネットワークは全体としてノードの参加と離脱がかなり頻繁な、動的なネットワークになります。このため Winny の設計も、ノードがいつ離脱してもネットワーク全体の機能に大きく影響しないようになっています。

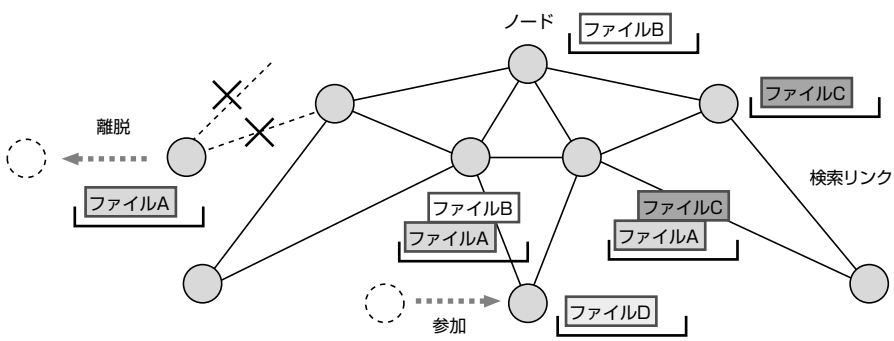


図 3-1　Winny ネットワーク

以降では、図 3-2 のような簡単な Winny ネットワークを使って説明します。

Winny を起動すると、そのノードは Winny ネットワークに接続し、終了するまで常に接続を維持します*1。Winny は、このノード間の接続（コネクション）を通じて、他のノード情報やファイルのキーを交換したりファイルを検索したりするので、この接続のことを「検索リンク」と呼びます。なお、ファイルを転送するときには、ファイルの転送元ノードと転送先ノードとの間でこれとは別の接続を開きます。転送に使用する接続を「転送リンク」と呼びます。

検索や配送の効率を高めるため、Winny は高速な回線に接続しているノードにより多くのキーやファイルを集めるようになっています。このために、Winny ネットワークには、高速な回線に接続するノードを上流、低速な回線に接続するノードを下流とする、特有の概念があります。Winny ネットワーク内のノードを大まかに分類すれば、FTTH に接続している高速回線のノード、ADSL に接続しているノード、ISDN に接続した低速回線のノードがあり、高速回線ほど他のノードにサービスする能力が高いと考えられます。この概念にどのような効果があるかは、以降で明らかになっていきます。

*1 Winny ノードは、起動時に上流（なければ自分と同等）のノード 2 つに接続を要求し、Winny ネットワークに参加したあとは、下流（もしくは自分と同等）のノード 5 つまでの接続要求を受け付けます。詳しくは 4 章「実装」をご覧ください。

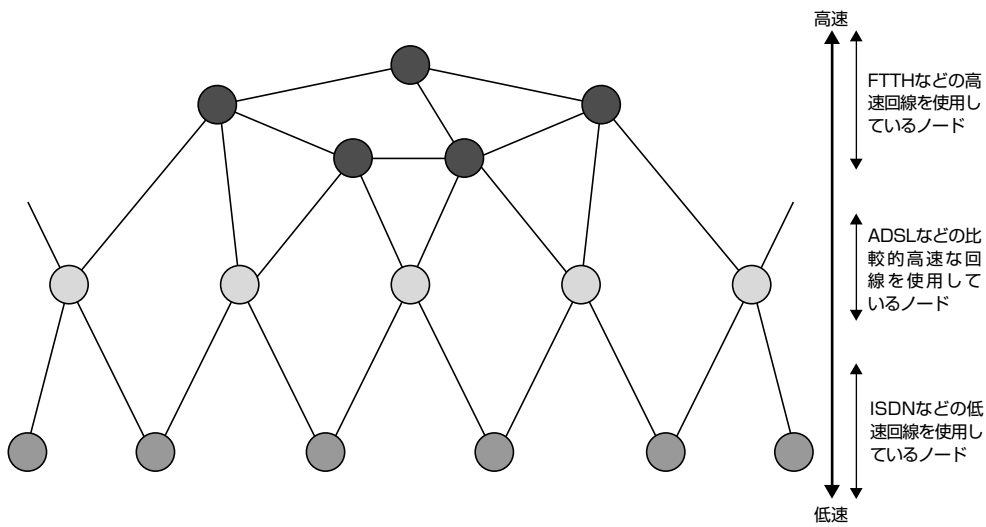


図 3-2　この章で使う Winny ネットワークの例

3.2 ファイルの公開からダウンロードまで

以降では、公開されたファイルがファイルを探しているユーザーに届けられるまでの過程を、順に追っていきます。

ファイルの公開

ユーザーが、公開するファイルをアップロードフォルダに置くと、Winny はそのファイルからキーと本体（ボディ）を作成します。キーは、いわばファイルの要約情報であり、ファイルの名前や大きさ、更新時刻、ファイルの識別に使うファイル ID（ハッシュ値）、ファイル本体の位置情報を示す IP アドレスとポート番号などが含まれています。ファイル本体のほうはファイルの内容そのものです。

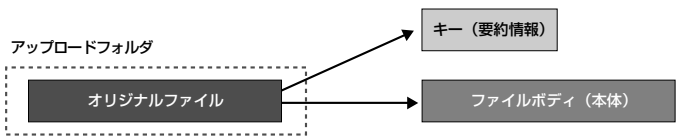


図 3-3　ファイルのキーと本体

Winny は、検索リンクでつながっている隣接ノードとのあいだで、持っているキーを定期的に交換するので、アップロードフォルダのファイルのキーはしだいに Winny ネットワークに拡散していきます。

コラム　ハッシュ値

ファイル ID となっているハッシュ値は、ファイルの内容全体から数学的なアルゴリズムで算出した一定サイズのデータです。ハッシュ値には、元のデータが異なればおおむね重複のない一意な数値になるという性質があるので、ファイルの内容の要約情報と考えることができます。Winny ではこれを利用して、ファイルの内容に応じた一意の ID を付けています。つまり、たとえファイル名が異なっていてもファイル ID が同じであれば同じ内容であり、ファイル ID が異なれば別の内容として扱います[*2]。

*2　ファイルの内容からハッシュ値への計算に使う手順をハッシュ関数と呼びます。Winny では、128 ビットの MD5 をハッシュ関数として使っています。

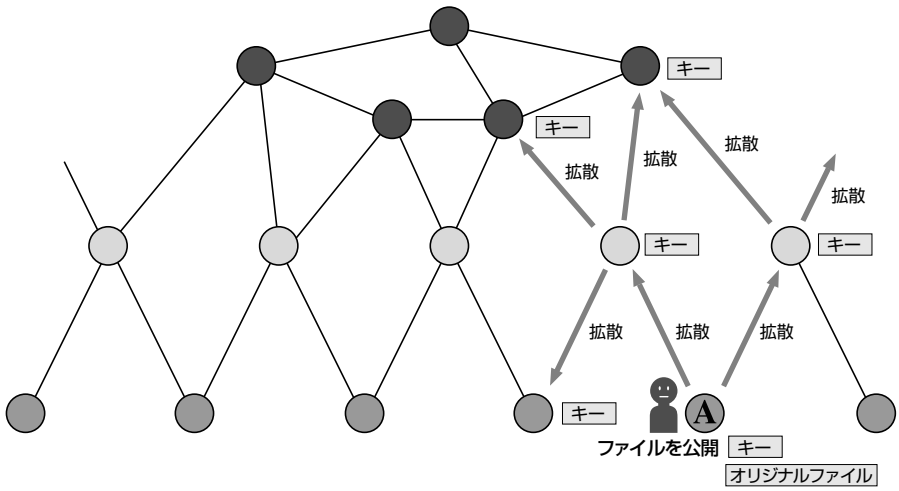


図 3-4　ファイルの公開とキーの拡散

図 3-4 は、ノード A がファイルを公開したときのようすです。

各ノードが保有するキーは、ノード間の定期的な交換と、次節で説明する検索によって、時間の経過とともに更新されていきます。

キーの寿命とリフレッシュ

このように、Winny の各ノードにはキーが集まってきますが、キーには寿命があります。他のノードから運ばれてきたキーは、一定の時間がたつと有効期限が切れ、キーのリストから削除されます。ファイルを公開しているノードが稼動しているかぎり、そのファイルのキーは定期的に運ばれてくるので、たとえ古いキーが消滅しても新しいキーの到着でキーは延命します。しかし、ファイルを公開しているノードがネットワークから切り離されれば、対応するキーは一定時間を経過したあと各ノードのキーのリストから削除されます。つまり、ファイルを持つノードがなくなればキーも消滅し、他のノードが検索をしても見つからなくなるというわけです。

ファイルの検索

公開ファイルのキーが拡散したところで、別のノードがこのファイルに関連するキーワードを指定して検索したとしましょう。Winny は、検索用のクエリ（問い合

わせ）パケットに検索条件の文字列を詰め、上流のノードに送り出します（図 3-5）。

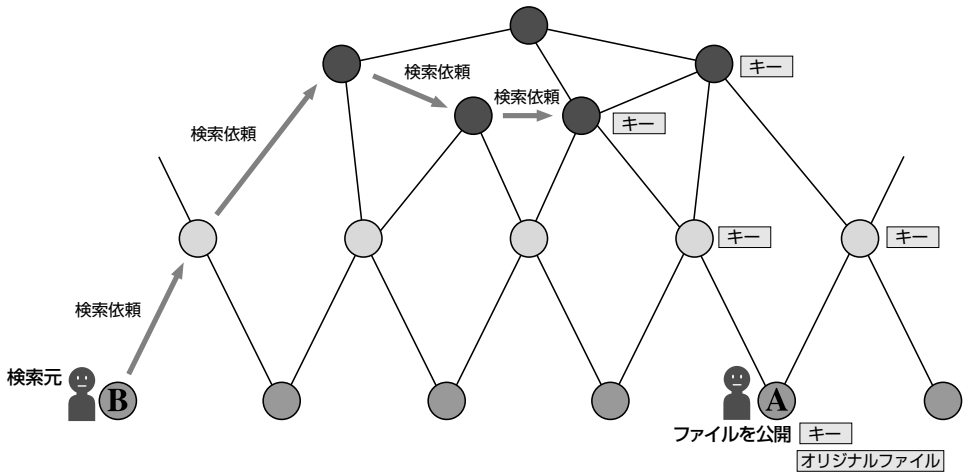


図 3-5　ファイルの検索

クエリは、転送先のノードで条件に合うキーがあるとそれを詰め込まれ、さらに隣接するノードへと転送されます。何度か転送を繰り返したところで、往路を逆にたどって検索元に送り返されます（図 3-6）。

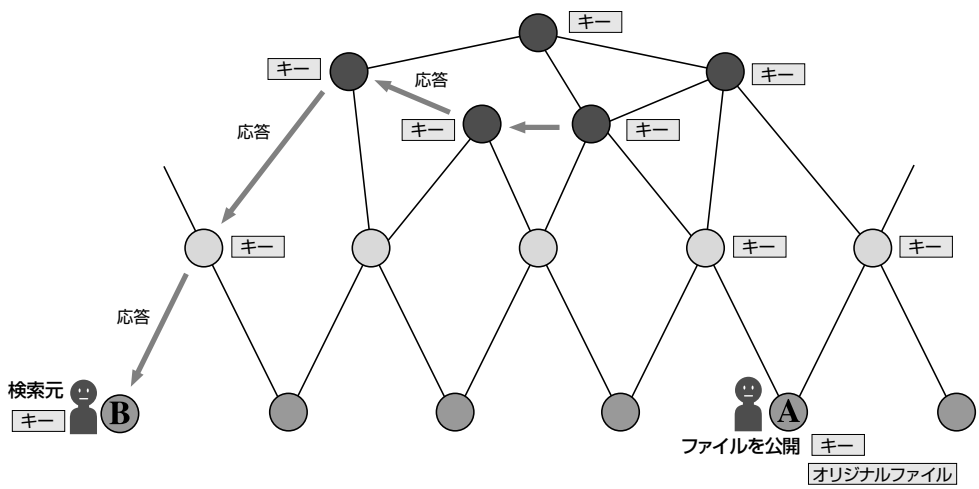


図 3-6　キーの入手

検索元ノードは、経由したノードで見つけたキーのリストを受け取ります（もちろん、不幸にして条件に合うキーが見つからない場合もあるでしょう）。このようにして検索元のノードは、隣接ノードとの定期的なキーの交換を待たずにクエリという積極的な方法を使って、遠方のノードにあるキーを検索し、キーを手に入れることができます。

検索を上流に向かって転送することには、重要な意味があります。まず、クエリの経路となる上流ノードにキーが拡散されるので、キーはしだいに上流ノードに集まっていきます。こうなったあとは、上流方向を検索するだけで、効率よく目当てのキーを見つけることができるようになります。

検索にはもうひとつ、クエリの通り道となったノードにも収集したキーが複製されるという効果があります。すなわち、キーの拡散は、隣接ノードとの定期的なキー交換と検索の２つのタイミングで行われます。

ファイルの転送

さて、定期的なキーの受信と検索操作でノードに集まったキーは、Winny の検索画面に一覧表示されています。ユーザーがそのなかから適当なものを選んでダウン

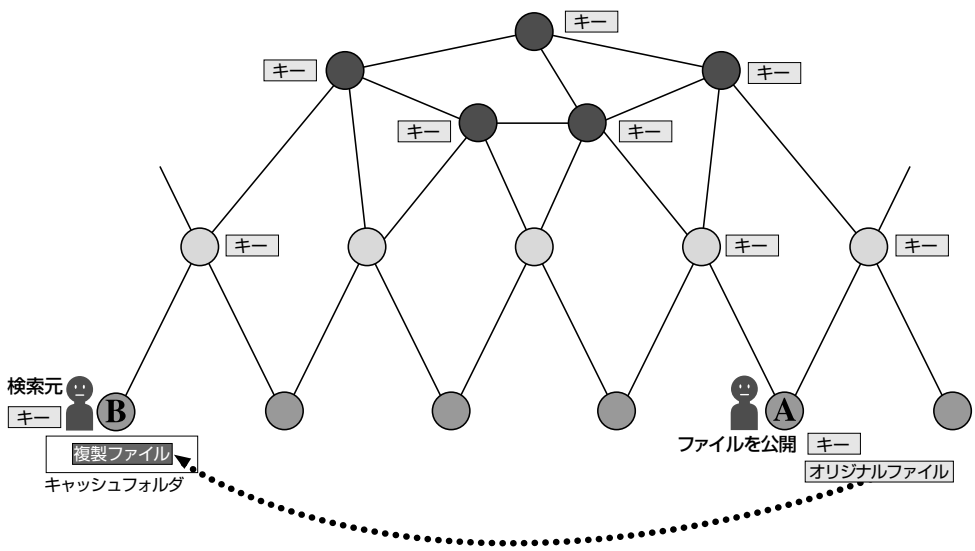


図 3-7　ファイルの転送

ロードを指示すると、ファイルの転送が始まります。検索結果のキーには、ファイル本体を持っているノードの情報（IP アドレスとポート番号）が含まれているので、Winny はそのノードと接続し（新たなコネクションを張る）、ファイルの転送に使用します（図 3-7）。このコネクションを転送リンクと呼び、ファイルを 1 つ転送するごとに開き、転送が終わると閉じます。

実際には、ファイルは暗号化したキャッシュファイル形式で転送され、ダウンロード側ノードのキャッシュフォルダに格納されます。転送が完了すると、ファイルはキャッシュファイル形式から元どおりのオリジナルファイルに復元され、ダウンロードフォルダに置かれます。

ダウンロードしたファイルは公開される

実は、ファイルをダウンロードしたあと、キャッシュフォルダに残ったファイルも公開され、他のノードから検索したりダウンロードしたりできるようになります。このことは、これまで説明してきた Winny の匿名性の実現方法とも関連してくるのですが、これについてはあとで詳しく説明します。

図 3-8 のノード B には、キャッシュしたファイルに対応するキーが作成され、隣接ノードとの定期的なキー交換で拡散していきます。

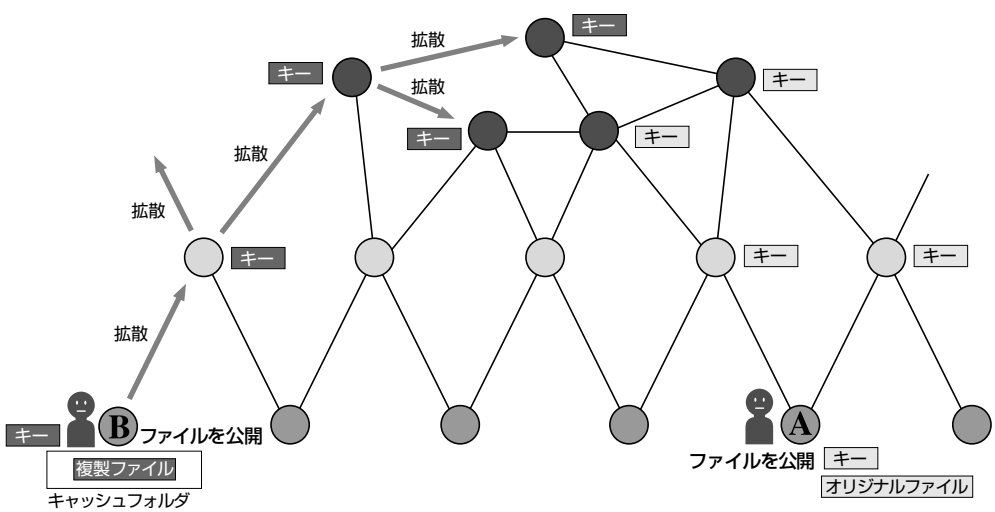


図 3-8　キャッシュの公開

さらに別のノードがダウンロードする

さらにもうひとつのノードＣが、さきほどのファイルに関連するキーワードを指定して検索をしたとしましょう。図 3-9 の例では、検索の結果、ノードＣは無事にファイルのキーを取得しましたが、このキーはノードＢから拡散されたものであり、ノードＢが持っているファイルを指し示しています。

キーを手に入れたノードＣがファイルのダウンロードを指示すると、キーに含まれているファイルの位置情報を使用して、ノードＢと転送リンクを張り、ファイルをダウンロードします（図 3-10）。

ノードＢは、ダウンロードしたファイルを公開することによって、ファイルを中継したことになります。一方ノードＣは、ダウンロードしているファイルが一次情報発信者のオリジナルなのか、キャッシュされている複製なのかを区別することはできないので、誰が一次情報を持っているのかを判断できません。Winny の匿名性は、プロクシーサーバと同様のキャッシングによって実現していることを述べましたが、その実際はこのようなものです。

キャッシュにはもうひとつ効用があります。Winny ネットワーク全体ではノードＡとノードＢが同じファイルを配布することになるわけですから、ノードＡだけで

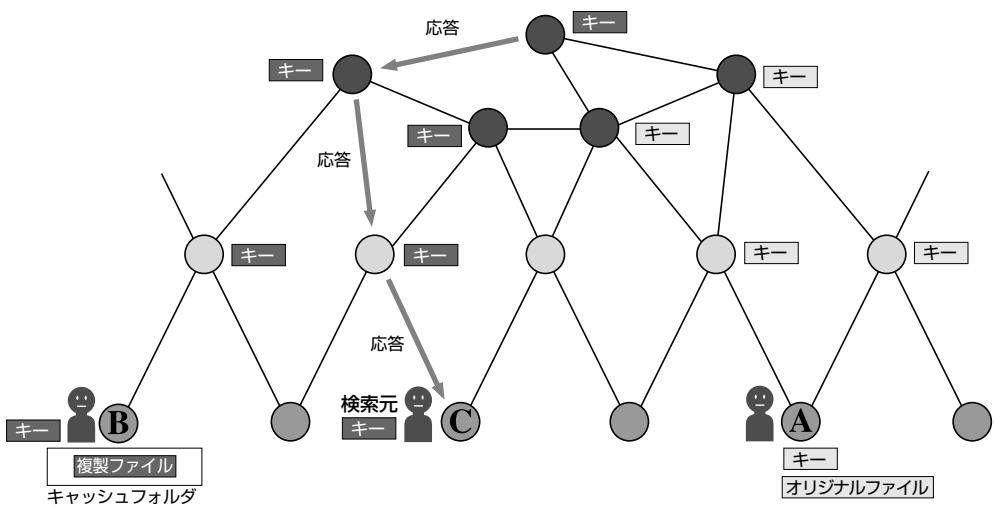


図 3-9　ファイルの検索――その２

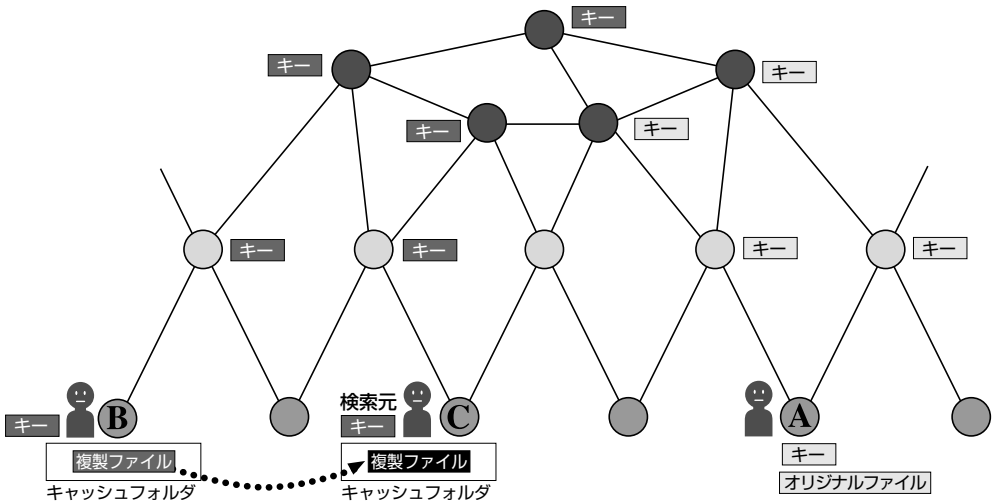


図 3-10　ファイルの転送――その 2

ファイルを配布するよりずっと効率がよくなります。

さらに、Winny はここに挙げた方法よりももう少し積極的な方法で中継を発生させています。次に進みましょう。

3.3 中継

さきほど、ファイルのキーは検索リンクを通して上流に拡散していくと説明しました。実は、キーに含まれているファイル本体の位置情報（IP アドレスとポート番号）は、拡散の途中に一定の確率で書き換えられます。

図 3-11 はそのようすを示したものです。ノード A がファイルを公開すると、ファイルの位置情報としてノード A の IP アドレスとポート番号を含むキーが検索リンクを通じて隣接ノード X に送られます。この例では、キーはノード Y にも送られ、さらにノード Z に送られるとき、位置情報の IP アドレスがノード Y のものに置き換えられています。

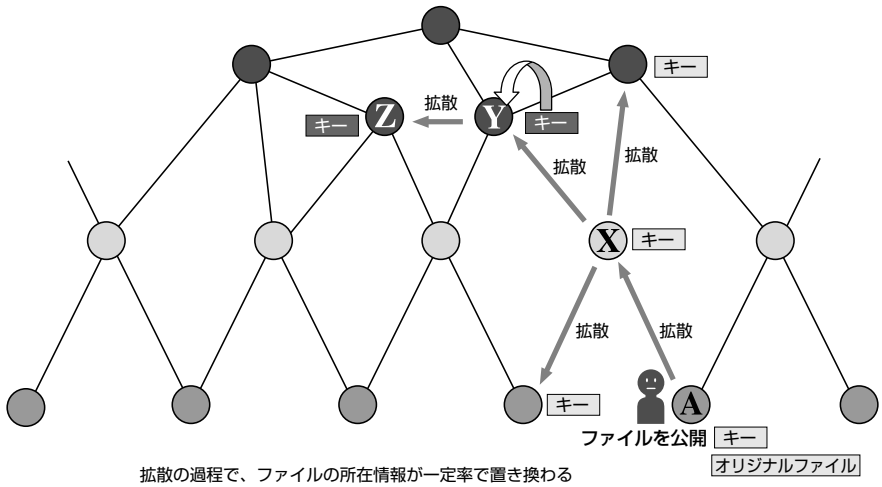


図 3-11　キーの拡散と書き換え

では、図 3-12 のように、ノード Z が持つキーをノード B が入手し、このキーを使ってファイルをダウンロードすると何が起きるでしょうか？

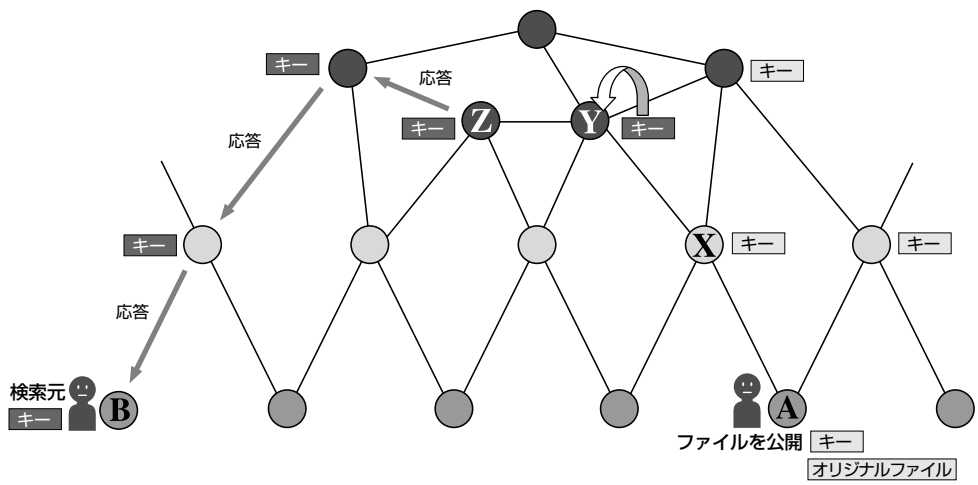


図 3-12　位置情報が書き換えられたキーを入手

ノード Z のキーはファイルの位置情報がノード Y になっています。このため、ノード Y はファイルを持っていないのに、ノード B から転送リンクが接続されて

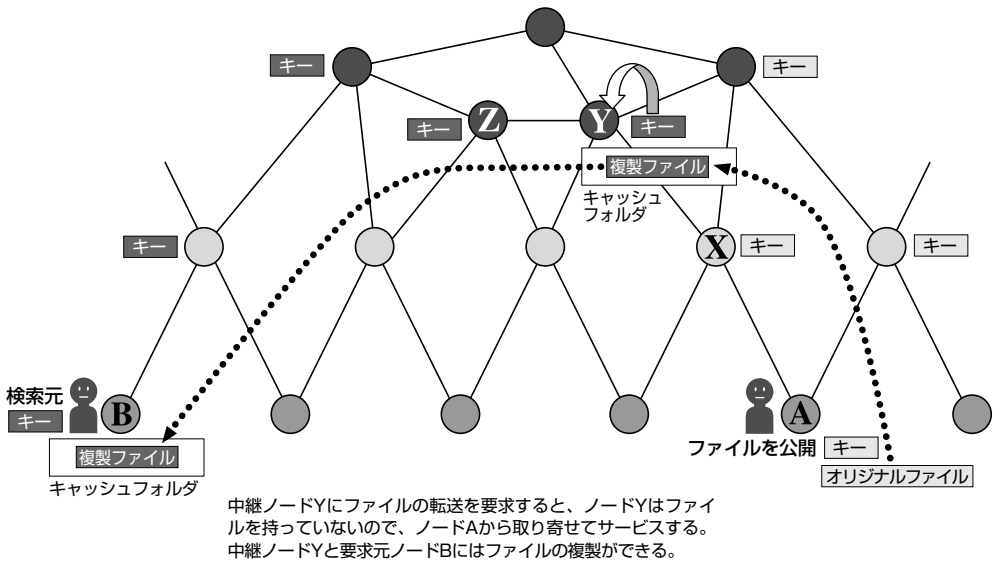


図 3-13　ファイルの中継

アップロードを要求されます。ノード Y は、ファイルが自ノードになければ、自分が持っているキーを使ってノード A からファイルをダウンロードします。Winny には、ダウンロード中のファイルをそのまま同時にアップロードする仕組みがあるので、それを利用してファイルをノード B に中継します（図 3-13）。

キーの IP アドレス書き換えは、定期的なキーの拡散時だけでなく、検索時のクエリパケットの転送でも行われるので、頻繁にダウンロードを要求されるファイルにはどんどん中継が発生します。結果的に、人気のあるファイルほどあちこちのノードにファイルがキャッシュされることになります。プロクシーサーバのキャッシングと同様に、キャッシュしたファイルは別の要求に応じてアップロードできますから、Winny ネットワークのファイルの配布効率は全体的によくなります。この仕組みによって、ダウンロード要求者には一次情報発信者のノードと中継ノードの区別はできないので、匿名性も実現できます。

補足的なことですが、中継をしている途中で要求元からのダウンロードがキャンセルされても、中継ノードが開始したダウンロードは続けられます。このため、中継ノードには高い確率でファイル本体全体がキャッシングされます。

3.4 大規模な P2P ネットワークに耐える

以上、ごく簡単なファイル共有の例を通して、ファイルをどのような仕組みで配送し、どのような仕組みで匿名性と効率のよさを実現しているかを見てきました。しかしこれだけでは、Winny のユーザーが増えてネットワークが巨大になったとき、ユーザーはほしいファイルがなかなか見つからないまま検索を繰り返し、Winny ネットワークは検索クエリのパケットで飽和して、崩壊してしまったでしょう。

特別なサーバを設けることなくサービスを実現している P2P ネットワークでは、大規模になった P2P ネットワーク内に散在している情報をいかにして検索・転送するかが大きなテーマになってきました。それに対する各種のアプローチは、1 章「P2P の基礎知識」で説明したとおりです。

Winny はこの問題に対して、

- 上流と下流
- クラスタリング

という独自の概念を導入して対処しています。次に、その 2 つの概念を解説していくことにしましょう。

上流と下流

すでに、Winny には回線性能の高いノードを上流、回線性能の低いノードを下流とする概念があることを述べました。しかし、そこにどのような効果があるかは明確に説明していませんでした。

この考え方は、「より能力の高いノードに、なるべく多くのキーとキャッシュファイルを集め、効率よくファイルを共有する」ことを意図したものです。そういう視点で、もう一度これまでのファイル転送の図を眺めてみてください。ノード間の定期的なキー交換と検索によって、キーは上流にたくさん集まります。キーが拡散していった先では一定の確率で中継が生じますから、キャッシュファイルも上流のノードに蓄えられていきます（図 3-14）。

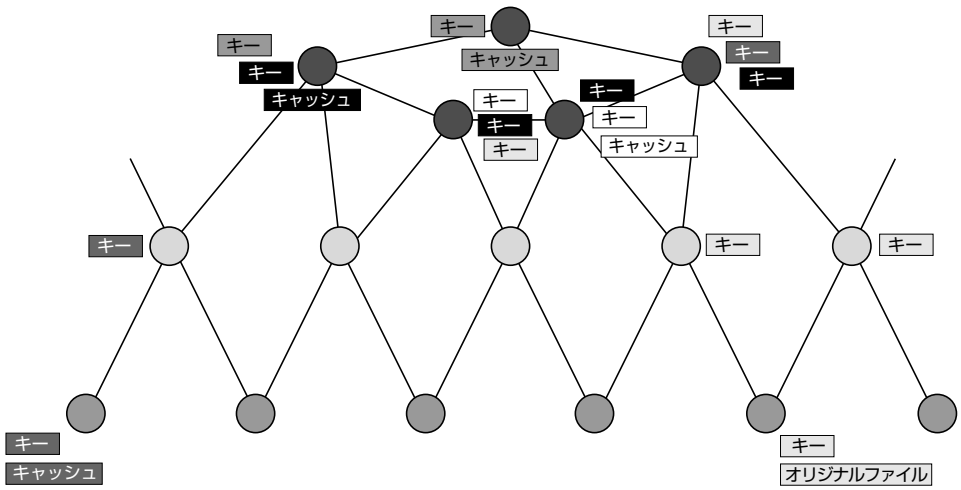


図 3-14　Winny ネットワークの上流にはキーとキャッシュファイルが集まる

また、能力の高い上流のノードにキャッシュファイルがたくさん集まっているので、Winny ネットワークは全体として多数のノードのダウンロード要求に応えることができます。

コラム　ユーザーが指定した速度情報がノードの上下関係を決定する

ここで説明した仕組みを実現するために、ユーザーは Winny の初回起動時に、使用回線のアップロード速度を指定するようになっています（標準では 120K バイト/秒）。ノードとノードが接続するときには、ユーザーが指定したこの速度情報を基にして、互いにアップロード性能を比較し、上流と下流という上下関係を形成します。

川の流れと Winny ネットワーク

さて、キーは上流のノードにどんどん集まってきますが、上流ノードの CPU 性能やメモリ保有量が大きいとはかぎらず、結局保持できるキーの数には限界があります。実際のところ、1 つのノードが持つキーの数は数万個程度のものです。このため上流ノードでは、大量のキーが流入する一方で、受け取ったキーがすぐに溢れて消えていきます。これは、ネットワークの規模が大きくなるほど顕著です。結果的に、上流のノードには絶えず新鮮なキーが流れていることになり、ファイルを検索

したときのヒット率が高くなります。

一方、下流のノードが持つキーは、自分自身でキャッシュを持っているものか、検索の結果受け取ったキーが多くを占めるでしょう。

このようなわけで、上流のノードは他のノードを検索しなくても、自ノードを流れる大量のキーを見ていれば、目当てのキーを見つけてファイルをダウンロードできることが多くなります。一方、下流のノードは、キーが大量に流通している上流ノードに向かって頻繁に検索クエリを投げかけることにより、目当てのキーを見つけ、ファイルをダウンロードするという図式になります。

このイメージを表したのが図 3-15 で、イメージとしては川の流れに近いものになります。

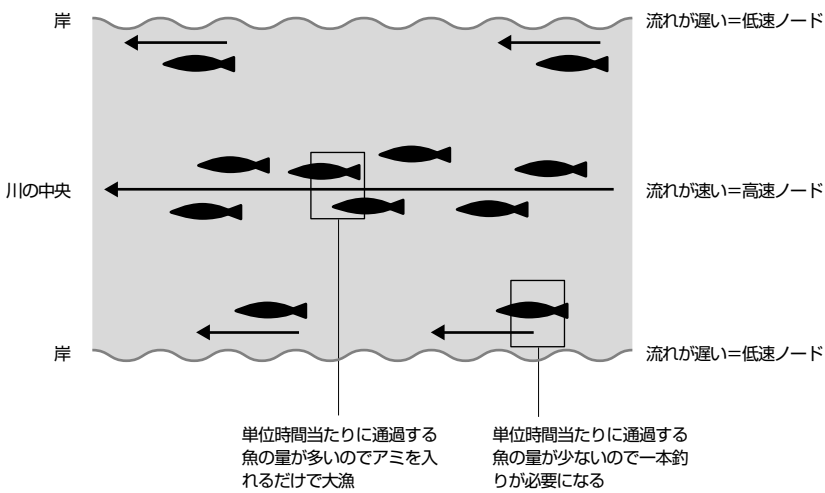


図 3-15　川の流れ的理論

川に、水の流れに乗って泳ぐ魚（キー）がいると思ってください。川の中央（高速回線の上流ノード）は水の流れが速いので、同じポイント（ノード）を見ていると、大量の水が流れ込んでは出て行きます。この流れに乗って魚が泳いでいるので、網をさし入れる（条件を指定して自動ダウンロードを働かせる）だけで、魚がたくさん獲れることになります。

岸（低速回線の下流ノード）は流れが遅く、水が淀んでいて魚はほとんどいないのですが、代わりに竿で中瀬に向かって針を投げれば（検索をすれば）、一本釣りで

あっても比較的割りよく魚を釣り上げられます。

では、上流と下流という概念を採り入れなかったらどうでしょうか。図 3-16 は、上流と下流の概念がある場合とない場合の、それぞれの検索方法の違いを示しています。両者のあいだで、検索クエリの転送の数（図の矢印）に明らかな違いがあることに気がつくはずです。

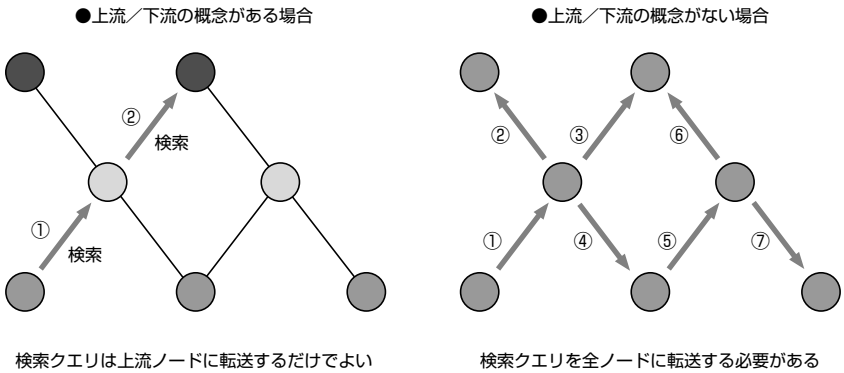


図 3-16　上流／下流の概念がある場合とない場合

クラスタリング

Winny ネットワーク内には、嗜好の異なるさまざまなジャンルのファイルが存在し、それにひも付けされたキーがあちこちに散在しています。上流にキーやキャッシュファイルが集まっているとはいえ、Winny ネットワークが大規模になれば、ユーザーがファイルを公開しても、キーはダウンロード希望者が検索できる範囲にはなかなかたどり着かなくなります。しかし、集めているファイルの傾向が似ているユーザー同士がなるべく近くに集まるように Winny ネットワークを形成すれば、ダウンロード希望者は少ない検索ステップでファイルを入手可能になり、情報の共有という面では効率がよくなります。

そこで Winny では、クラスタリングという概念が導入されています。クラスタリングとは、ひとことで言ってしまえば「好みの似ている」ノードをひとかたまり（クラスタ）にすることです。たとえば、クラシックファンのノードの周囲にはクラシックファンのノードが、サッカー愛好家のノードの周辺にはやはり同じような嗜好の

ノードがあったほうがよいでしょう。クラスタリングは、まさにこうした嗜好性の近いノードを緊密に連携させるための仕組みです。

図 3-17 は、Winny におけるクラスタリングを非常に単純化した概念図です。

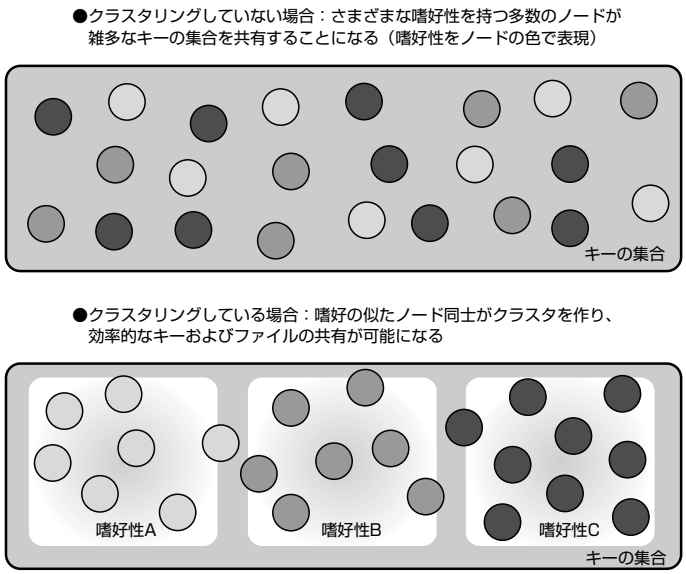


図 3-17　クラスタリングの有無によるネットワークの比較

では、何を基準にクラスタリングを行うかというと、それはユーザーが自動ダウンロード機能を利用するときに条件として指定するキーワードです。

Winny は他のノードと検索リンクを接続するとき、持っているノード情報のなかから、自動ダウンロード時に指定したキーワードのうち最初の 3 語が最も類似しているノードを選んで接続します。このときの詳細な規則などは、4 章「実装」で説明しています。再接続の機会がくるたびに、検索キーワードの似ているノードと選択的に接続していけば、やがて好みの似たノード同士が集まるようになります。

このようにして、上流／下流とクラスタリングの概念を導入した結果、Winny のネットワークは図 3-18 のようなイメージになります。ピュア P2P 型ファイル共有ソフトにクラスタリングと上流／下流の概念を採り入れたのは、重要なポイントだと考えます。そのどちらかだけで、大規模なネットワークを実用的な効率を保って維持することはできなかったでしょう。

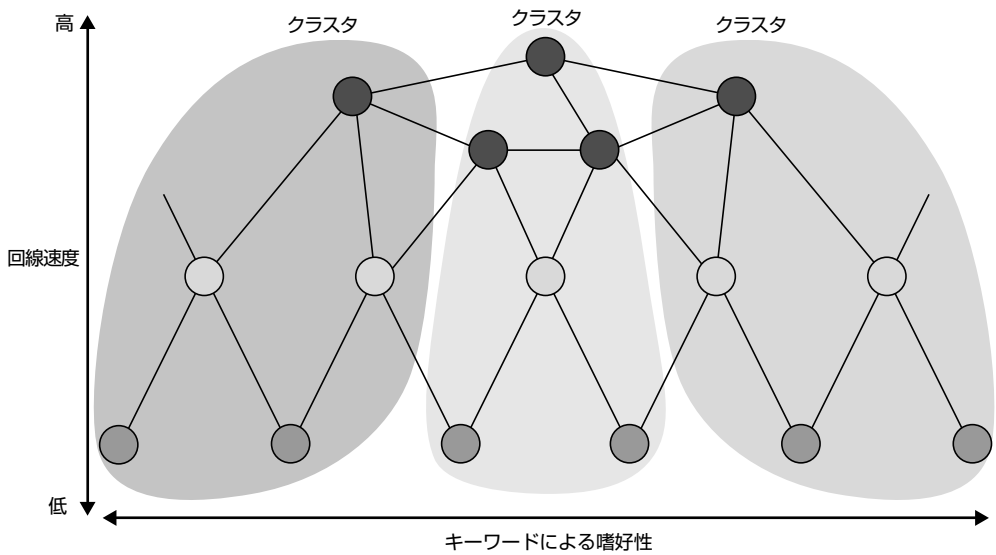


図 3-18　クラスタリングと上流／下流によるノードの組織化

3.5 Winny のその他の要素

以上で、Winny がファイルをどのような仕組みで配送し、匿名性と効率の向上を実現しているのか、という話はおしまいです。しかしそのほかにも、ユーザーが使いやすいように多重ダウンロードと自動ダウンロードという仕組みがあり、Winny の特徴のひとつになっています。そこでこの 2 つを含め、Winny の構成要素をもうすこし詳しく説明します。

アップロードフォルダ～キャッシュフォルダ～ダウンロードフォルダ

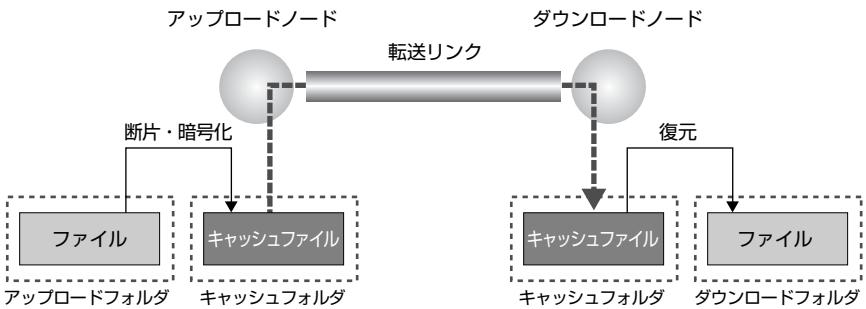
ユーザーの明示的な操作でファイルをアップロードする場合でも、ユーザーの操作によらず、キャッシングや中継機能によってキャッシュファイルを公開する場合でも、Winny はアップロード可能なファイルはすべて、キャッシュフォルダ内にキャッシュファイル形式で保管します[*3]。

ノードにキャッシュファイルができると、それに対応するキーが作成され、Winny

*3　実は、ファイル本体をアップロードフォルダとキャッシュフォルダとで二重に持つのは無駄なので、キャッシュフォルダにはキャッシュファイルのヘッダだけが置かれ、要求があるとアップロードフォルダの本体がキャッシュファイル形式に変換して転送されます。

ネットワーク内に拡散していきます。キーを入手した他のノードからファイルの本体を要求されると、Winny はキャッシュフォルダ内のキャッシュファイルをアップロードするわけです。

ファイルの転送を要求したノードは、ダウンロードしたキャッシュファイルをキャッシュフォルダに格納します。ファイルの転送後、キャッシュファイルから元のファイルが復元され、ダウンロードフォルダに置かれます（図 3-19）。



キャッシュフォルダの内容は他のノードに公開され、アップロードの対象となる

図 3-19　アップロードフォルダ～キャッシュフォルダ～ダウンロードフォルダ

ダウンロードしたあとキャッシュフォルダに残っているキャッシュファイルは、Winny ネットワークの他のノードに公開され、アップロード可能となります。これは、これまで説明してきたとおりです。

キャッシュファイル

キャッシュファイルは、ファイル本体を 64K バイトごとのブロックに分割して暗号化したもので、先頭にはファイルの内容を要約したヘッダが付いています。Winny はノード内に保有しているキャッシュファイルのキーをメモリ上で管理しており、この中には細分化したブロックのどの部分を持っているかを示すビットマップ情報が含まれています。Winny はキャッシュファイルをダウンロードするとき、ビットマップ情報の整合性を保つので、たとえダウンロードが途中で失敗したとしても、キーのビットマップを見れば、ノード内にファイルのどの部分のキャッシュブロックがあり、どの部分を再送すればよいかがわかるようになっています。

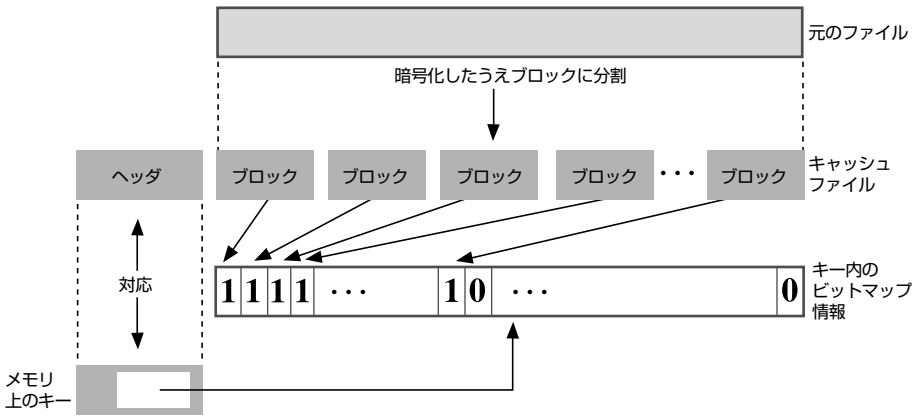


図 3-20　キャッシュはブロックに分割される

ファイルの分割と多重ダウンロード

ファイルをブロックに分けて転送することには、次のような利点があります。

まず、ファイルの転送が途中で失敗したとしても、ファイルの先頭から再送する必要がありません。それまでに転送されていないブロックだけをあとから転送すればよいので、Winny ネットワークは全体として、転送に失敗したときの再送信に必要な通信コストが小さくなります。

また、同じファイルを持つノードが複数ある場合は、それぞれのノードから別々のブロックを並行してダウンロードできます。これが多重ダウンロードです（図 3-21）。

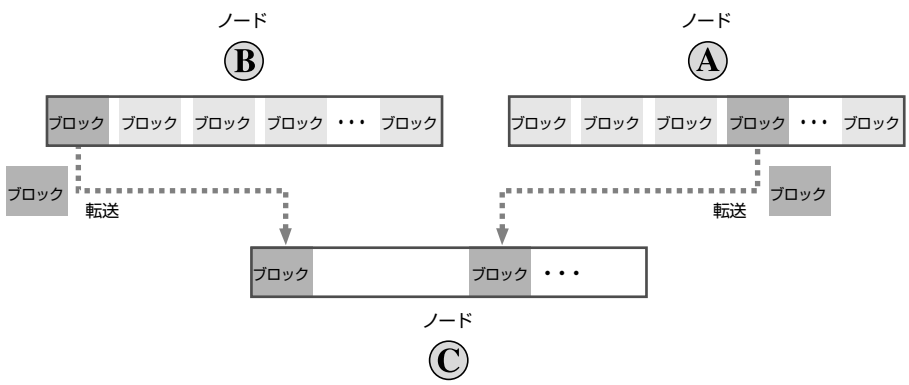


図 3-21　多重ダウンロード

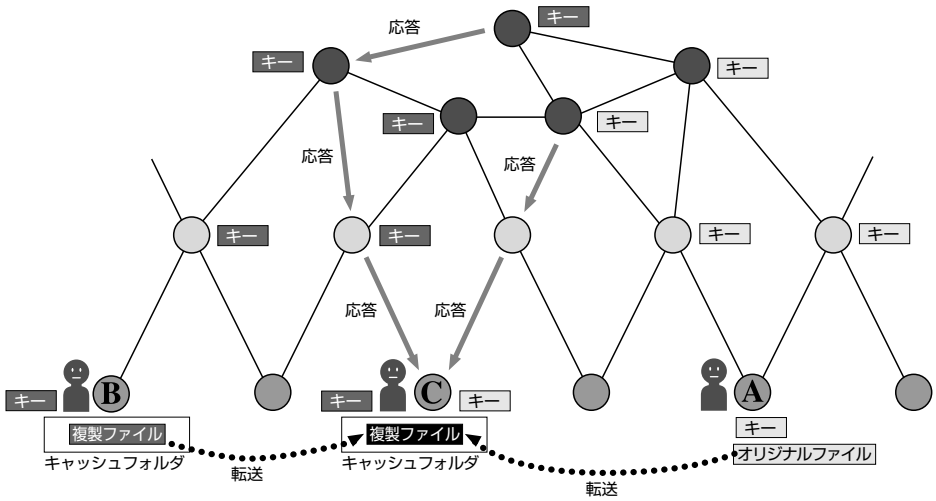


図 3-22　例で見る多重ダウンロード

アップロードノードの回線速度がダウンロードノードの回線速度よりも遅いような場合にも、この機能を使って高速にダウンロードができます。

65ページの図 3-10 の例をもういちど見てみましょう。図 3-22 では、ノード C が探しているファイルがノード A とノード B にあります。自動ダウンロード機能は定期的に検索を繰り返すので、ノード C が検索の結果ノード A のキーを見つけてファイルをダウンロードしているあいだ、さらにノード B のキーも見つかれば、ノード C は同じファイルをノード B からも同時にダウンロードできます。

ノードの発見――「最初の一歩」問題

中央集中型の固定的なサーバがなく、接続状況が流動的なノード群で運用されている P2P ネットワークでは、各ノードが最初にどのようにして P2P ネットワークの他のノードを見つけて接続するかがいつも問題になってきました。

ピュア P2P 型の Winny は、Winny ネットワークのノードどれか 1 つを知っていれば、そこに接続して芋づる式に他のノードの情報を得られるようになっています。ただし、はじめて Winny を起動したときだけはノード情報をまったく持っていないので、初期ノード情報として、そのとき起動している（と思われる）他の Winny ノードの IP アドレスとポート番号を設定してやります。

Winny の場合、たいへん原始的な方法ですが、掲示板などを通して他のユーザーに教えてもらうようになっています。また、インターネット上の情報サイトなどから暗号化されている初期ノード情報の一覧を入手して、設定ファイルにコピーしてから Winny を起動する方法も一般的です。

Winny の終了時には、そのとき接続しているノードの情報が設定ファイルに保存され、2 回目以降の起動時にはその情報を使って自動的に他のノードと接続するようになります。

3.6 Winny ネットワーク再考

この章の冒頭で、Winny ネットワークは、「IP ネットワーク上に Winny ノード相互が築いたアプリケーションによるネットワーク」だと説明しました。ここで、この意味をもういちど考えてみたいと思います。

オーバーレイネットワークという側面

Winny ネットワークは、インフラとなっている IP ネットワークの上に、Winny というアプリケーションが積み上げたネットワークです。Winny ネットワーク内のどのノードとどのノードが隣り合わせで接続されるかは、Winny が動作しているコンピュータが物理的にどのようなトポロジーで IP ネットワークに接続しているのかということとは関係がありません（図 3-23）。

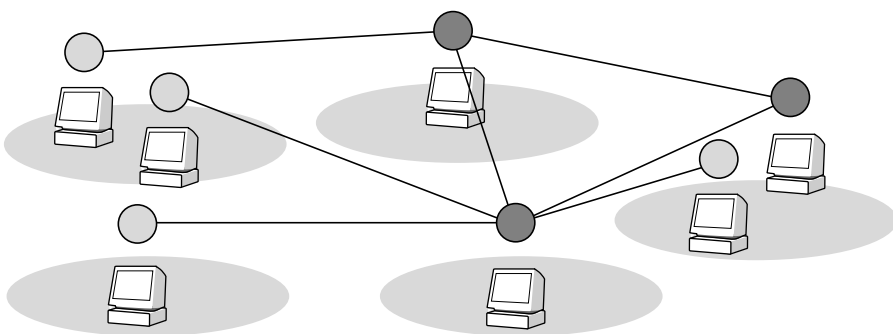


図 3-23　IP ネットワークと Winny ネットワーク

たとえば、同じローカルエリアネットワーク（LAN）内で2台のコンピュータがWinnyを動作させているとき、2つのIPネットワークにおける距離は非常に近いのですが、それぞれのユーザーの嗜好が大きく異なれば、Winnyネットワーク上での距離は遠くなってしまうでしょう。

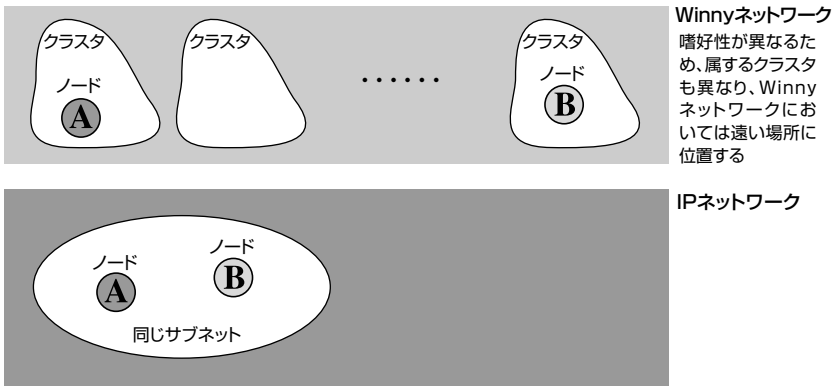


図3-24　Winnyによって形成されるネットワーク

Winnyネットワークという系

中心的な役割をはたすサーバがなく、対等な役割のノード群によってファイル共有のネットワークを作るとき、効率のよいネットワーク形態とはどのようなものでしょうか？　まず最初に考えることは、ファイルを保有するノードからファイルを必要としているノードまでの距離（ホップ数）を短くすることです。ファイルの転送時間が短くなり、経由するネットワークやノードへの負担を小さくできるからです。

すべてのノードが相互に接続しあう形態であれば、どのノードとも直接（1ホップ）通信できますが、ネットワークの規模が大きくなると、すべてのノードを相互に接続させるのは現実的ではありません。これまで説明してきたように、ネットワーク内のいくつかのノードと接続を保ち、離れているノードには隣接ノードを介して芋づる式に通信させることになります（図3-25）。

ネットワークのこのような問題を、面識のない人に伝言式でメッセージを送る場合に例えることができます。

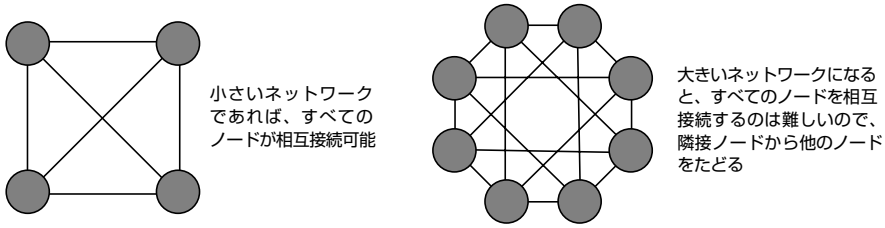


図 3-25　ネットワークの規模とノード間のホップ数

差出人は、なるべく受信人にツテがありそうな知り合いを選んでメッセージを伝言し、その知り合いはさらに別の知り合いに伝言し、幸運に恵まれれば何人かを通して受信人に届けられるでしょう（もちろん届かない場合もあります）[*4]。

しかし、メッセージの内容や受信人は、差出人の生活環境や関心事と結びついていることが多いので、似たものに関心のある人が集まるコミュニティであれば、メッセージの伝言はもうすこし成功率が高くなり、仲介人も少なくてすみます。Winnyのクラスタリングは、まさにそのようなグループを形成する仕組みです。検索キーワードを使って、似ているファイルを集めている人を同じグループにまとめ、ファイルの検索や転送に必要なホップ数を少なくしているのです。

さらにメッセージの伝言では、なるべく「顔の広い人」にメッセージを託したほう

*4　1960 年代後半に心理学者のミルグラムが行った実験では、米国のカンザスとネブラスカの住人が面識のないボストンの住民に手紙を渡すのに、160 例のうち約 30%が成功し、その成功例では平均 6 人の仲介で手紙を転送することができたといいます。

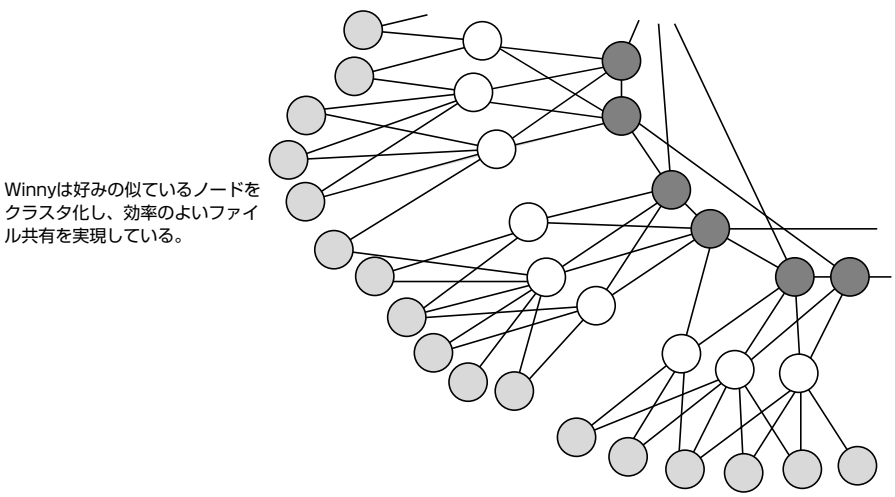


図 3-26　Winny ネットワーク

が、伝言の成功率が高くなるに違いありません。Winny ネットワークでは、回線速度の速いノード（上流ノード）を「顔の広い人」にしています。「顔の広い人」に情報を集める一方で、問い合わせを「顔の広い人」に投げかけることによって、ファイル共有の効率をよくしているのです。

Winny ネットワークは、現在の多くのネットワークのように、誰か特定の管理者が構成を決めたり管理をしているわけではありません。各ノードが協調し、それぞれの関心（検索キーワード）と接続回線速度に基づいて、自律的に形成されます。これは、ピュア P2P ネットワークの興味深い特徴だといえます。

3.7 まとめ

本章では、Winny というプログラムが、どのような方法で匿名性を備えつつも実用的な効率を保ってファイル共有を実現しているのかを見てきました。

- **ファイル共有の実際**

  Freenet は、ファイル自体をブロックに細分化してネットワーク内に拡散させていき、ファイルを入手しようとするノードは検索の届く範囲にたどり着いたブロックを組み立てていました。Winny では、ファイル自体ではなくファイルの要約情報となるキーをネットワーク内に拡散しておき、ファイルを入手しようとするノードはまずキーを検索し、キーに含まれている情報からファイルの所在ノードに接続してファイルを転送します。

- **匿名性の実現**

  匿名性は、キャッシュと中継という 2 つの仕組みで実現されています。ネットワーク内のあちこちに一次情報と同じ内容のキャッシュができると、一次情報を発信したノードとキャッシュを持つノードの区別はつきません。中継は、さらに別の方法でキャッシュの複製をネットワーク内に分散させ、匿名性を高めると同時に、ファイル配布の効率も向上させます。

- **ファイル共有の効率を高める**

  Winny は、高速回線につながっている上流のノードにより多くのキーとキャッシュが集まるように工夫し、検索とファイル共有の効率を高めています。これは、キーや検索パケットを上流のノードへと転送させることによって可能になっています。また、クラスタリングという概念を導入し、嗜好の似ているノードを同じクラスタにまとめて検索とファイル共有の効率を高めています。

- **Winny によって形成される P2P ネットワーク**

  Winny ネットワークは、IP ネットワークの上に Winny が形成するアプリケーション層のネットワークです。ノードのネットワーク内での位置は、Winny が動作しているコンピュータの物理的な接続形態とは関係なく、クラスタリングと回線速度に応じた上流・下流という尺度によって、動的に変化します。

実　装

4

章

3章「Winnyの仕組み」では、あるユーザーが公開したファイルがどのような仕組みで他のユーザーに届けられるか、また大規模な利用でも効率よくファイルを共有可能とする仕組みを説明しました。この章では、Winnyの実装を詳しく解説します。

4.1 プログラムの概観

Winnyは多くのプログラム要素から構成されていますが、主要なものは、タスク管理、ノード管理、クエリ管理、キー管理を担当する4つの部分です（図4-1）。

ノード管理は、他ノードの情報を収集し、他ノードとのコネクションを管理します。

クエリ管理は、Winnyネットワーク内のキーの検索を行います。クエリ管理は、ノード間のキーの受け渡しを担当し、問い合わせる条件をクエリパケットに詰めて隣接ノードに送り出したり、他のノードから受け取ったクエリパケットからキーを取り出したりします。

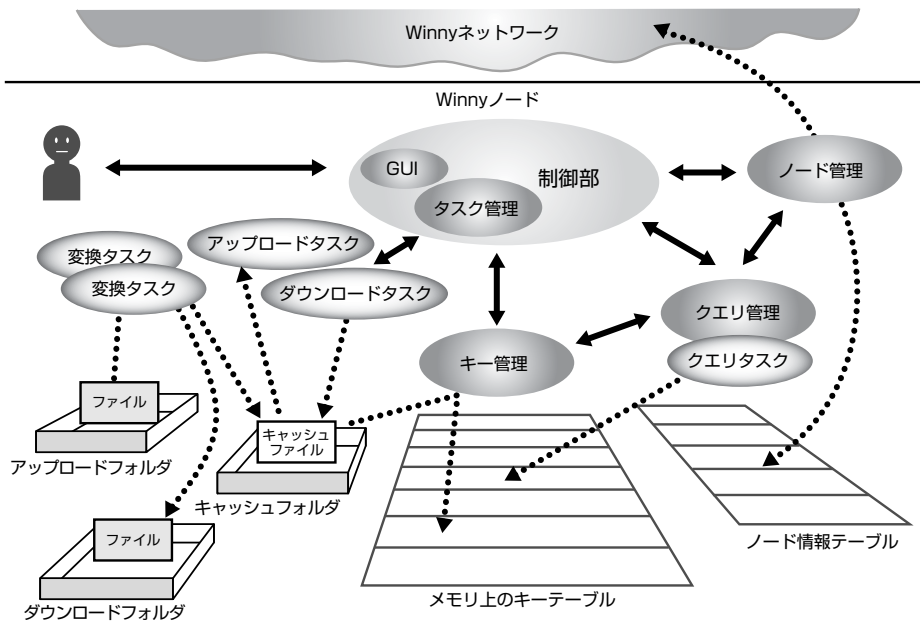


図4-1　Winnyの構成要素

キー管理は、自ノード内に持っているキャッシュファイルのキーと、拡散や検索によって他のノードから受け取ったキーを、メモリ上で一括して管理します。

またWinnyは、アップロードやダウンロード、オリジナルファイルとキャッシュファイルの変換、ハッシュ値のチェックなどを独立したタスクとして動作させ、並行して処理します。タスク管理部は、これらのタスクの起動や実行状態を管理します。

Winnyは、以上のようなプログラム要素が連携しながら、検索やファイル転送を実現しています。プログラム要素それぞれについては次節以降で詳しく説明することにして、その前にWinnyの大まかな動作を眺めてみます。

検索リンクを接続する

Winnyは起動するとすぐに、上流のノード2つと検索リンクを張ります。このときの接続相手は、ノード情報テーブルのなかからクラスタリングを考慮して選びます。相手ノードが接続ノード数の制約で新しい検索リンクを張れない場合、相手ノードから代わりのノード情報が候補として送られてきます。Winnyはこの情報を使って再度検索リンクの接続を試みるとともに、ノード情報テーブルにも格納します。ノード情報テーブルは、以降に検索リンクが切断したりWinnyを再起動したとき、

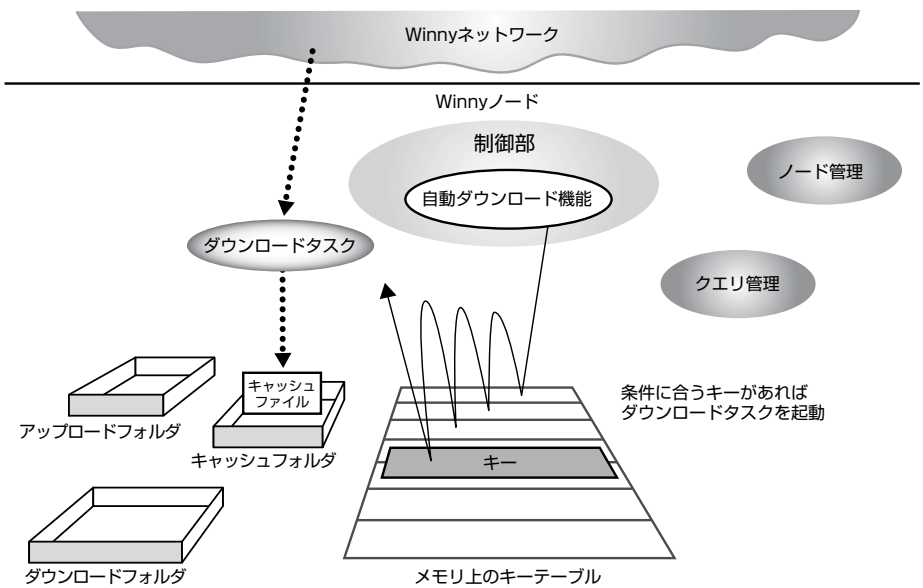


図4-2　常にノード内のキーを走査する

新しい接続先を選ぶために使用します。

一方 Winny は、他のノードからの要求があれば最大５つまで検索リンクの接続を受け付けます。リンクの最大数を超えて接続をそれ以上受け付けられない場合、代わりのノードをノード情報テーブル内から選んで紹介します。検索リンクは以降、キーの拡散や検索に使用します。

ファイルの検索とダウンロード

キーの拡散や検索で受け取ったキーは、メモリ上のキーテーブルに蓄積されます。キーはファイルのダウンロード時に参照するほか、他ノードからの検索や他のノードへのキーの拡散に使用します。

Winny は、ノード内のキーテーブルに蓄積されているキーを定期的に走査し、ユーザーが指定したダウンロード条件に合うものが見つかると、キーに含まれているファイル位置情報を使って転送リンクを張り、ファイルをダウンロードします（図 4-2）。

ノード内にキーが見つからなければ、検索クエリを送信します。検索クエリで見つかったキーはノード内のキーテーブルに格納されるので、キーテーブルの定期的な走査で検出されてダウンロードが始まります（図 4-3）。

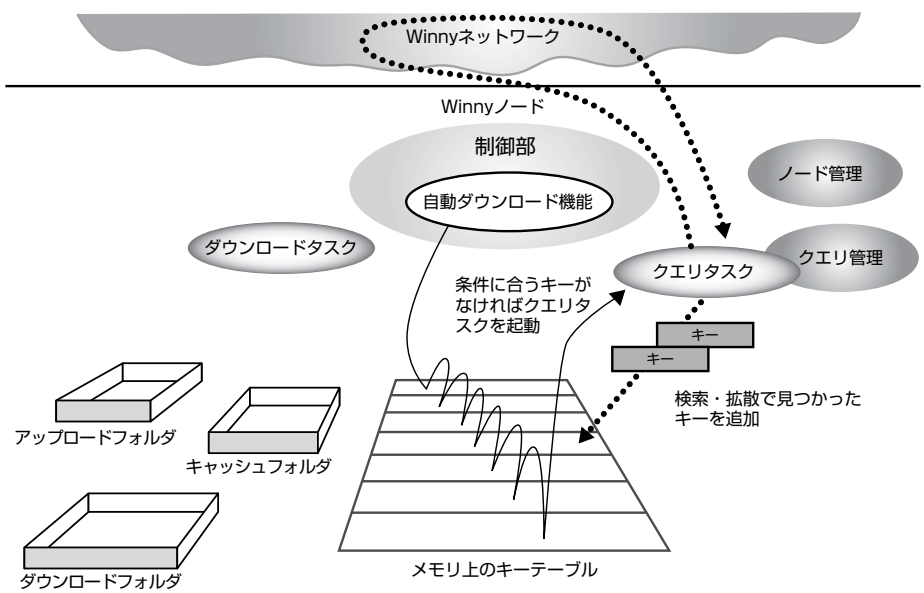


図 4-3　ノード内で見つからないキーに対して検索クエリを送り出す

キーの検索からファイルのダウンロードまでの一連の動作は、53ページの［ユーザーの目から見たWinny］で説明した自動ダウンロード機能によるものです。自動ダウンロード機能は汎用的なもので、ユーザーの手操作によるファイルのダウンロードも、特定のファイルIDを検索条件として実現されています。

ファイルのアップロード

ユーザーがファイルをアップロードフォルダに置くと、Winnyはそのファイルから生成したキーをメモリ上のキーテーブルに格納するとともに、キャッシュフォルダにファイル本体を置きます。キャッシュフォルダ内にはダウンロードや中継によって他のノードから取り寄せたキャッシュファイルも格納されており、他のノードからの要求に応じて送信します。アップロードに必要なキャッシュファイルに不足がある場合、前項の手順でキャッシュファイルのダウンロードが並行して行われます。

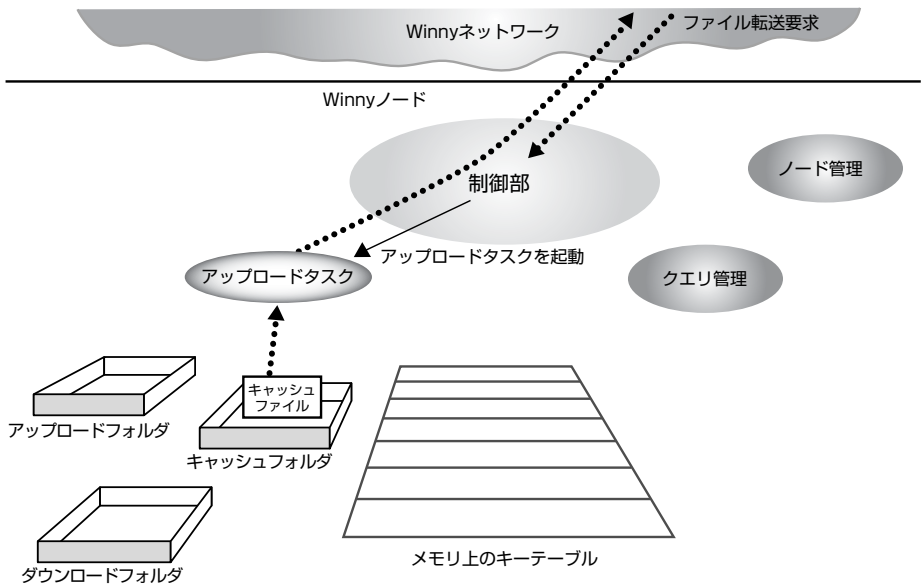


図4-4　他のノードからの要求に応えてキャッシュファイルを送信する

以降では、この動作を実現しているプログラムの要素それぞれについて、詳細に説明します。

4.2 ノード管理

検索リンクの接続時に相手ノードにリンクの余裕がないとき、相手ノードは接続する代わりに別のノード情報を送信してきます。ノード管理部は、このノード情報を収集して保存するとともに、実際の通信に用いる TCP コネクション（BSD ソケット）を管理し、非同期通信のための通信バッファリングを行います。

他のノード情報の提供

Winny は、他のノードと検索リンクを接続するとき、そのノードの IP アドレス、ポート番号、ノードの回線速度とクラスタリングに使用するキーワードなどを収集して保存します。回線速度とクラスタリングキーワードは、どのノードと接続すれば効率がよいかを判断するために使います。

Winny は、他のノードから検索リンクの接続を要求されたとき、最大接続数を超えているなどの理由で接続を断る場合がありますが、このときに代替の他ノード情報を送信します。接続を断られたノードは、これによって接続候補となる他ノードの位置情報を知ることができます。代替のノードには、接続してきた相手のクラスタリング情報を元に、相手に近いと思われるノードが選ばれます。

なお、1 つのノードが保持できる他ノード情報は 600 個までです。これを超えると、Winny は自ノードから遠いと判断されるノード情報を選んで破棄します。この結果、ノードはクラスタ的に近い他ノード情報のみを保持するようになり、検索リンクの切断やシステムの再起動で検索リンクを接続しなおすときに、より好みに近いクラスタに接続できます。ユーザーが検索キーワードを変更すると、ノード情報を紹介する側のノードも新しいキーワードと相関度の高いノード情報を選んで送ってくるようになります。つまりそのノードは、望むクラスタのノードが直接見つからなかったとしても、比較的近そうなクラスタのノードに接続することができます。

このように、隣接ノードからより近いクラスタのノードを順に教えてもらってゆっくりクラスタ間を移動し、目的のクラスタにたどり着きます（図 4-5）。クラスタリングのために実施している具体的な接続規則については「クラスタリングとノード間の相関度」の項を参照してください。

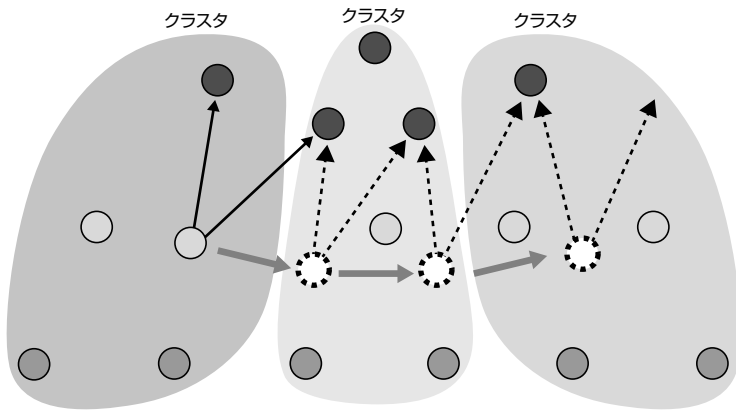


図 4-5　クラスタの移動

検索リンクの接続

Winny は、なるべく近いクラスタの上流ノード 2 つと常に検索リンクを張る一方で、他のノードから検索リンクの接続要求があれば最大 5 つまで受け付け、下流ノードとします (図 4-6)。

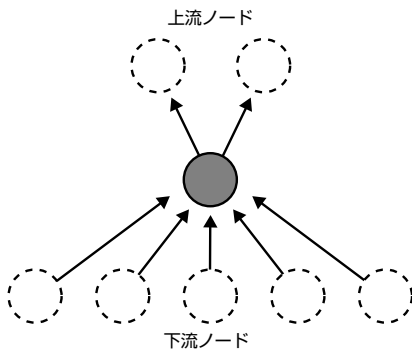


図 4-6　検索リンクと上流・下流ノード

上流リンク数を 2 つとした理由は、検索リンクが 1 つでは上流ノードに問題が生じたときに検索ができなくなるからです。かといって、上流ノードを 3 つ以上にすれば、1 つの上流ノードがサポートしなくてはならない下流ノードが増えすぎて、上

流ノードの負担が高くなります。また、下流にいくつまでノードを接続できるようにするかは、ネットワーク内にあるノードの回線速度分布に依存します。これは、高速ノード側と低速ノード側の回線速度比が小さいほど、下流側の検索リンク数を増やさないといけません。開発当時に行ったシミュレーションの結果では[*1]、下流ノードの数を 4 + α にしないと、接続できないノードが生じることがわかったため、余裕をみて下流の最大接続数を“5”としています。

*1 詳しくは「設計時のシミュレーション」参照。

以上のような背景から、プログラムが適用しているルールは、次のとおりです。

- 接続している上流ノードが 2 つに達していなければ、4 秒に 1 回の割合で他の上流ノードに接続を試みる。
- 接続先ノードには、持っているノード情報のリストのなかからクラスタリング情報が近似しているものを順に選ぶ。
- 上流ノード数が 3 つを超えたら、接続時間の短い検索リンクを選んで切断する。
- 下流ノード数が 5 つを超えたら、接続時間の短い検索リンクを選んで切断する。
- 接続相手とのアップロード速度差が大きい場合（比が 20 以上）には接続しない。

この方法で、Winny は検索リンクが切断しても、上流のノード 2 つと接続を保ち続けます。なお、上流か下流かは次のように判断します。

- アップロード速度の早いほうが上流。
- アップロード速度がほぼ同じ場合（差が 2 割以内）は、待ち受け側（accept 側）が上流、接続側（connect 側）が下流。

ノードのアップロード速度は、ユーザーが設定によって申告した値です。なお、Winny で用いる回線速度の単位は K バイト/秒です。通常の回線速度ではビット/秒がよく用いられますが、Winny ではファイル転送量をユーザーにわかりやすいようにバイト単位（K バイト/秒）で表示しており、それに合わせたためです。

図 4-7 は、上記のようなルールでノードが接続していき、Winny ネットワークを形成するようすを簡単な例で示したものです。

上部の黒いノードは、高速回線を使用している上流ノード、淡色のノードは中流ノード、中間色のノードは下流ノードです。

上流ノードになると、自ノードより上流（高速）のノードを見つけるのは難しくなります。そこで Winny では、相手の回線速度が自ノードの 8 割以内であれば、妥協

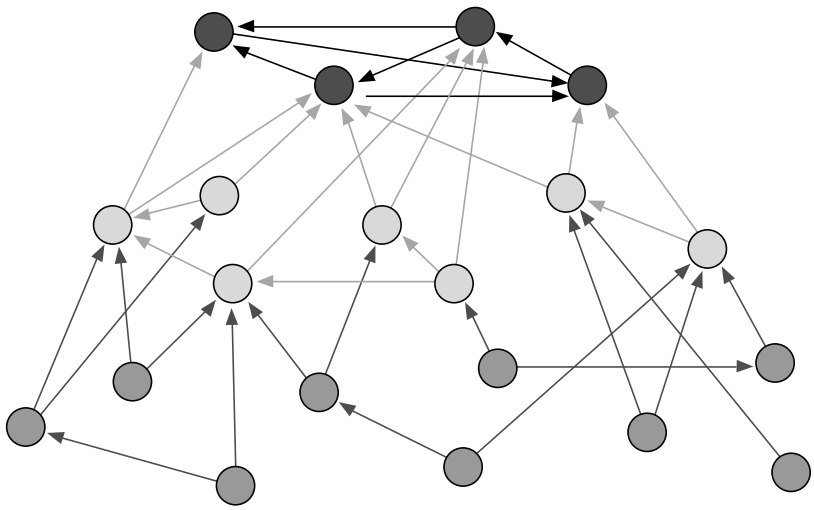


図 4-7　ノードの接続と Winny ネットワークの形成

して同程度の速度ノードとみなし、上流ノードとして接続します。

Winny は、検索時にクエリを上流のノードへと転送していくため、同程度の回線速度のノードと接続しているときでも、そのどちらかを上流方向と決めておく必要があります。そこで、同程度の回線速度のノードは、他方に接続を要求したほうが下流、接続を受けたほうを上流としています。

Winny は、ノードを回線の速度によっていくつかの層に分けて扱います。厳密に区別しているわけではありませんが、おもに次の 3 つを想定しています。

- 上流――光ファイバーで接続しているノード
- 中流――ADSL などで接続しているノード
- 下流――ISDN などで接続しているノード

上流のノードと下流のノードが直接接続すると、本来であれば中流のノードを接続すべき上流の待ち受けポートが下流のノードにとられてしまいます。そこで Winny は、速度差の大きいノードから接続を要求されると、接続を拒否します。

このような仕組みによって、Winny が作る検索リンクのネットワークは、下流は回線速度に応じた階層構造をなし、上流は網構造となります。Winny はこのネットワークを用いてキーの拡散や検索を行います。

ノード情報

Winny は、検索リンクの接続のために他のノードの情報を収集して保存しています。他のノードの情報を取得するタイミングは 2 つあります。まず、初期ノード情報や他のノードからの紹介で、IP アドレス、ポート番号、接続回線のアップロード速度、クラスタリングに使うキーワードを取得します。またそれらを使って検索リンクを接続したあと、直接相手から接続環境とバージョン情報を教えてもらいます。

Winny が管理しているノード情報を**表 4-1** にまとめます。Winny が他のノードと検索リンクを張るために最低限必要なのは、IP アドレスとポート番号だけです。しかし、どのノードと接続すれば効率がよいかを判断できるように、他のノードから紹介された情報には接続回線のアップロード速度とクラスタリング用のキーワードも含まれています。

表 4-1　ノード情報

取得のタイミング	内　容	備　考
接続前に他のノードから渡される情報	IP アドレス（IPv4）	必　須
	ポート番号（TCP）	必　須
	アップロード速度（K バイト/秒）	任　意
	クラスタリングキーワード 3 つ	任　意
接続後に相手ノードから取得する情報	接続形態 （直接／ NAT 経由／ DDNS ／開放ポートなし）	
	サーバントのバージョンなど	

初期ノード情報や他のノードから紹介される情報のうち、アップロード速度とクラスタリング用のキーワードは、そのノードのユーザーが設定した任意の値です。

接続形態

Winny のノードが置かれているネットワーク環境はさまざまです。たとえば、ファイアウォールの内側にあるようなノードとの通信には工夫がいります。そこでノード情報に、接続形態を示す以下のような種別のどれかをセットします[*2]。

*2 表示上は DDNS となりますが、内部的には NAT+DDNS という組み合わせもありえます。

- インターネットに直接接続しているグローバル IP アドレスのノード（RAW）
- NAT ＋ポートフォワーディングを使用して接続している（NAT）
- ダイナミック DNS を利用している非固定グローバル IP アドレスのノード（DDNS）
- NAT や HTTP プロクシーなどのファイアウォールを介した接続で、開放されているポートがない（Port0）

RAW

Winny が特別な対処をしなくてもノード間の通信が可能です。

NAT

Winny は他のノードと接続するときに、自ノードの IP アドレスを相手ノードに申告します。しかし、ノードが NAT を介してインターネットに接続している場合、LAN 内にある Winny ノードから WAN に向けて送られたパケットは、NAT によって送信元 IP アドレスが LAN 内で使用しているものから WAN 側向けのものに変換

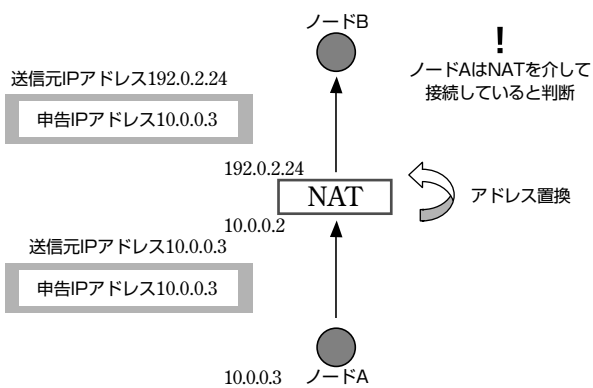


図 4-8　NAT を介した接続

されます（図 4-8）。

このため相手ノードは、申告した IP アドレスと実際の接続で得られた送信元 IP アドレスが異なると、NAT を介して接続しているものと判定し、NAT の内側のノードからキーを受け取ったときには、キー内の IP アドレスをグローバル IP アドレスに置き換えて他のノードに送ります（図 4-9）。

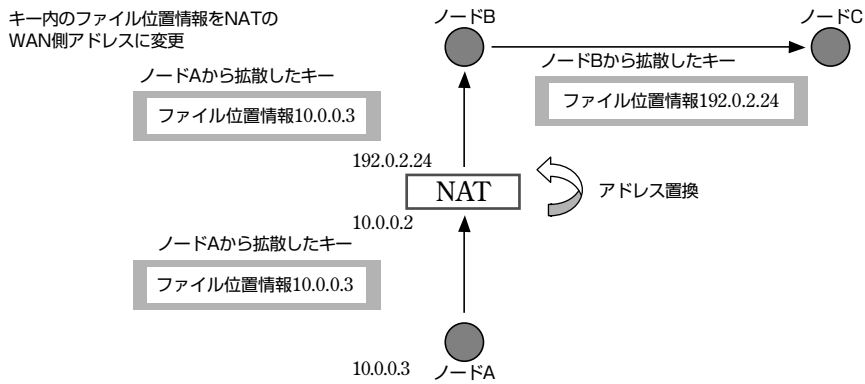


図 4-9　NAT の内側にあるノード

DDNS

DDNS を利用しているノードは、接続相手とのネゴシエーションで自ノードの位置を IP アドレスの代わりに DDNS 名（ドメイン名）で申告できます。DDNS 名を受け取ったノードは、DDNS 名から IP アドレスを索き、以降に再接続が必要になったときに使用します。また受け取った DDNS 名は、ノード情報としてファイルに保存しておき、ノードが再起動したときにはあらためて DDNS 名から IP アドレスを索いて接続します。

Winny は、各ノードの IP アドレスが動的に変化しても問題がないように設計してありますが、クラスタリングを効果的に機能させるためには、Winny の再起動時に相関度が高いことがわかっているノードに再接続できたほうがよいでしょう。DDNS の設定はそれを助けるものです。ユーザーが DDNS 情報を設定しておけば、たとえそのノードの IP アドレスが変わっても、隣接ノードは再接続が可能になります。

なお、DDNS 名で受け取ったノード情報を別のノードに紹介するときには、DDNS 名ではなくアドレス解決後の IP アドレスを使用します。しかしその IP アドレスを

受け取ったノードがDDNSノードに接続したときには、最初のネゴシエーションで受け取ったDDNS名のノード情報を自ノード内のファイルに保存するので、このノードも再起動時にはDDNS名からIPアドレスを索いて検索できます。

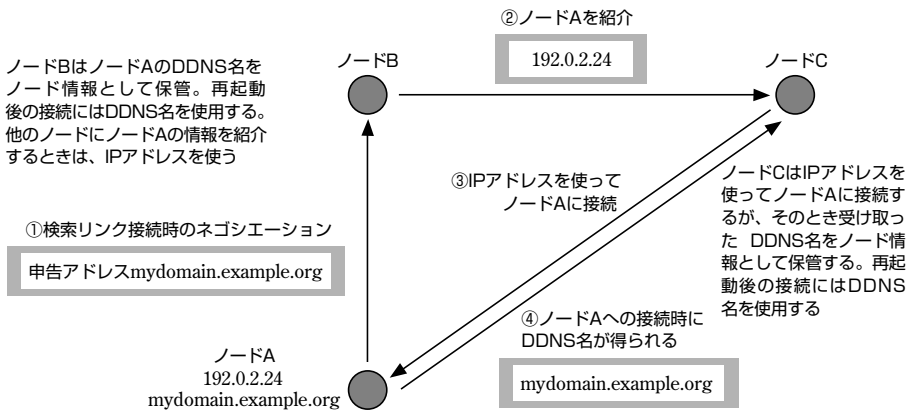


図4-10　DDNSノード

Port0

Port0は、ファイアウォールを通過できないといった理由でWinnyがコネクションの待ち受けに使うポートを設けられない場合に、ユーザーが申告するものです。Port0ノードに対しては他のノードがコネクション接続要求を出すことができない

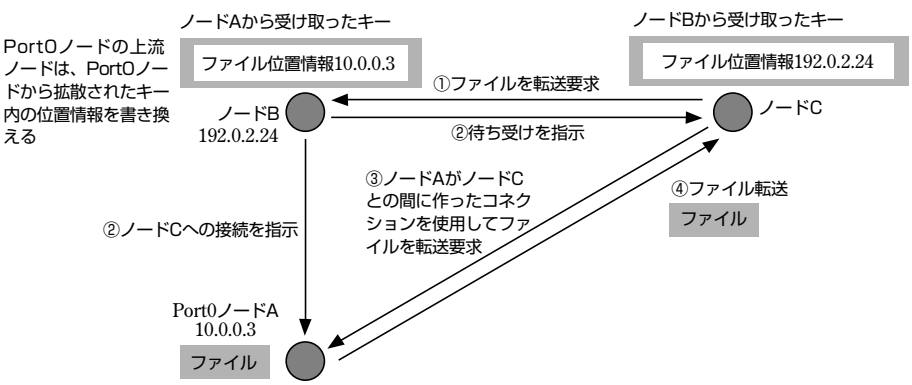


図4-11　Port0ノードからのファイルの転送

ので、Port0 ノードが持つファイルがほしいときには、まず Port0 ノードの上流ノードにファイル転送の要求を出します。そのために、Port0 ノードのファイルのキーは、位置情報が中継を行う上流ノードの IP アドレスに置き換わってネットワーク上に広まります。

Port0 ノードのファイルを転送する方法は、ファイルの要求側が待ち受けポートを開放しているかどうかによって違います。ポートが開放されている場合、中継ノードはファイル所有者（Port0 ノード）のノード情報を要求側に通知して待ち受けを指示する一方で、所有者ノードには要求側ノードに転送リンクを張るように指示します。この方法でできた転送リンクを使って、要求側は所有者ノードにファイルを要求し、取得します（図 4-11）。

要求側も Port0 ノードである場合、中継ノードがファイル所有者との検索リンクを通してファイルを取り寄せ、要求側ノードに転送します（図 4-12）。

なお、ノードが Port0 状態であるのに申告していないと、他のノードが接続を試みて失敗することになります。このようなノードが多いと Winny ネットワーク全体の効率が悪くなるので、Winny では接続を受け付けたノードが逆方向の接続を試し、申告されたポートが本当に接続可能かどうかを確認します。もし逆方向の接続ができなければ、そのノードは BadPort0 ノードとして扱い、ノード情報を破棄します。また、BadPort0 と判定したノードには警告を通知します。この警告数が多いと、Winny は自動的に動作を停止し、待ち受けポートの設定ミスによる悪影響を防ぎます。

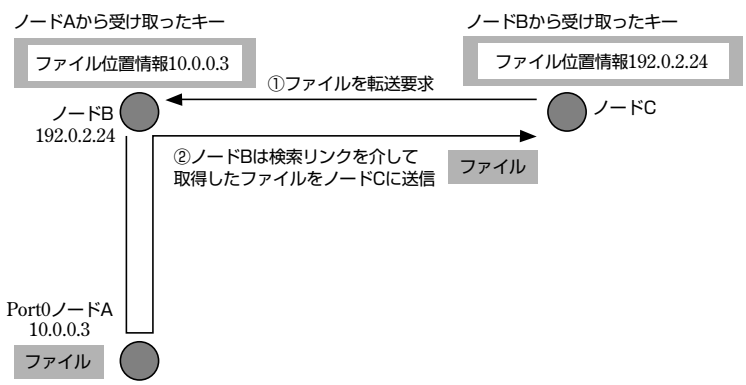


図 4-12　Port0 ノードからのファイルの転送――ダウンロード側ノードも Port0 の場合

コラム ダイナミック DNS

ダイナミック DNS（Dynamic DNS : DDNS）は、非固定 IP アドレスのホストをドメイン名でアクセスできるようにするサービスです。

利用方法はサービスサイトによって若干違いますが、たとえば example.org というドメイン名のサービスサイトに、ユーザー用の mydomain.example.org というサブドメイン名と、いまホストに割り当てられているグローバル IP アドレスを登録しておくと、そのホストはインターネット上から mydomain.example.org という名前でアクセスできるようになります。

接続事業者と固定 IP アドレスの契約をしていない場合、ホストに割り当てられる IP アドレスは頻繁に変わっていくので、IP アドレスの変化を検出してダイナミック DNS に登録されている IP アドレスを更新するプログラムを使い、いつも決まった名前で最新の IP アドレスを索けるように保ちます。最近では、ブロードバンドルータにこの機能が用意されていたり、ISP が IP アドレスの割り当てにあわせて自動的に DNS を更新するようなサービスを提供するケースもあります。

ノードのバージョン情報

Winny はたびたびバージョンアップをしてきたので、接続するノード双方のバージョンが異なり、それぞれが別々のプロトコルを使用するという状況も考えられます。そこで、接続開始直後のネゴシエーションで相手のバージョン情報を調べ、それにあわせて使用するプロトコルを動的に変えるようになっています。プロトコルに互換性がなければ Winny のネットワークは分断されてしまうので、このようにしてできるだけ異なるバージョン間でも機能するようになっています。

また Winny は、自分よりも新しいバージョンのノードが複数見つかったら、ユーザーに警告メッセージを出すようになっています。これは、新しいバージョンを公開したときに、ユーザーにできるだけ早くバージョンアップをしてもらえるように導入したメカニズムです。

クラスタリングとノード間の相関度

3章「Winnyの仕組み」で説明したように、Winnyネットワークにはクラスタリングという概念があり、好みの似ているノードをグループ化することによって、Winnyネットワーク内のファイル共有効率を高めています。

問題は、「好みの似ているノード」という主観に基づいた相関性をどうやって求めるのかということです。Winnyでは、ユーザーに3つのキーワードを指定してもらい、そのキーワードの近似度からノード間の論理的な距離を求めます（図4-13）。

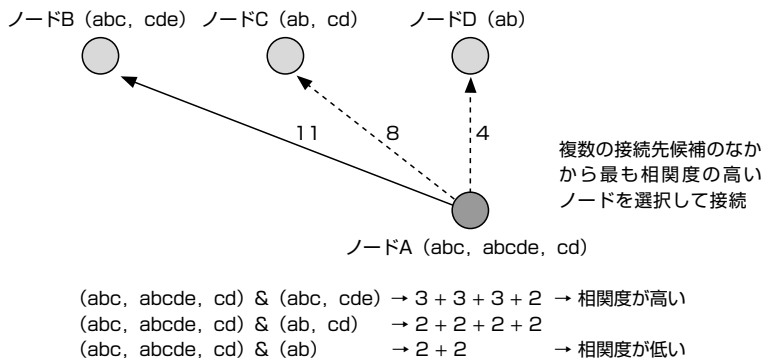


図4-13　ノードの相関度を評価する

具体的には、各ノードが指定した3つのキーワードそれぞれを組み合わせた9通り（3 × 3）の文字列ペアを作り、その部分文字列一致数の合計が大きな組を近似度が高いと判断します（図4-14）。なお、現在のWinnyの方式では、“abc”と“ab”のように、どちらかが他方に含まれる場合のみを部分一致として処理しています。“abcd”と“bcx”のように双方の文字列の一部分が同じでも、部分一致とは扱いません。相関係数の算出方法は次の基準で検討し、試行錯誤の結果、現在の方法に落ち着きました。

- 結び付きは比較的緩やかなほうがよい。
- 各ノード間の差がはっきり出るようにしたい（みんな同じ値になるとクラスタに分かれない）。
- ユーザーから容易に参加クラスタの選択が可能な方式。

（abc，abcde，cd）&（abc，cde）→ 3 + 3 + 3 + 2 = 相関度11

ノードA側のキーワード

ノードB側のキーワード	“abc”	“abcde”	“cd”
“abc”	3	3	0
“cde”	0	3	2
“ ”	0	0	0

ノードの3つのキーワードを組み合わせ、それぞれの一致文字数の合計で相関度を評価する。短いキーワードが長いキーワード側に完全に含まれる場合、短いキーワードの文字列長だけ相関度が大きくなる。

図 4-14　ノードの相関度を評価する

キーワードを 3 つにしたのは、大分類、中分類、小分類を想定し、この程度で問題ないだろうととりあえず決めただけで、特にこの数でなくてはいけないというものではありません。

ユーザーにとって、クラスタリングキーワードという概念はわかりにくいだろうと思います。そこで、自動ダウンロードで指定されているキーワードから抜粋し、そのままクラスタリングキーワードとして用いています[*3]。

なお、ダウンロード条件にハッシュ値も指定されている場合、検索キーワードは該当ファイルのファイル名であり、自動ダウンロードのためのキーワードではないため、比較対象から除外されます。

この仕組みによって、メカニズムを理解していないユーザーでもノードは自然とクラスタリングされますし、メカニズムを理解しているユーザーは自分で積極的に参加クラスタを選んで渡り歩くことができます。

*3　自動ダウンロードに指定しているキーワードが似ていれば、ノード間の相関度が高いだろうと仮定したのです。

コラム ノード間の距離

ノード間の距離が遠いか近いかは、あるノードから目的のノードまでのホップ数――ノードをいくつ経由するかで表します。Winny は常に他のノードとの間に検索リンクを張っていますが、この検索リンクでつながっている隣のノードがホップ数 1、そのまた隣がホップ数 2 になります。Winny のクラスタリングは、好みの似たノード同士が少ないホップ数となるように、検索リンクの接続時に相手を選ぶことによって実現されます。

クラスタリングと検索リンクの接続相手の選択

クラスタリングは、検索リンクを接続・切断するときに、各ノード情報から接続優先度を評価することによって実現しています。接続優先度は、これまでに説明したノード間の相関係数と、それまでの接続実績を反映した動的接続優先度から算出します。

接続優先度の算出ルールは次のとおりです。

- 接続優先度は、動的接続優先度とノード間相関係数の和である。
- 動的接続優先度とノード間相関係数はともに 100 以下に抑制される。
- あるノードに検索リンクの接続を試行すると、動的接続優先度は 8 減らされる。
- あるノードからファイルがダウンロードできた場合、動的接続優先度は+20 される。

Winny は検索リンクの接続先を選択するとき、保有しているノード情報をノード接続優先度でソートし、最も優先度が高いノードに対して検索リンクの接続を試行します。また、保持しているノード情報があふれて破棄する必要が生じた場合、ノード接続優先度の小さなものから破棄します。この 2 つの動作によって、Winny ではクラスタリングが実現されています。

検索リンクの接続実績があるノードは、接続に成功しても失敗しても、動的接続優先度が低くなります。これは特定のノードとばかり接続することを防ぐためです。ファイルのダウンロード成功時には、動的優先度を上げてそのノードとの結びつきを強めます。この場合も、特定のノードとだけ結びついて、いつまでもつながらな

い状態が続くのを防ぐために、それぞれの優先度は 100 以下にクリップされます。

> **コラム 接続優先度のクリッピング**
>
> ノード接続優先度が高くなりすぎると、クリッピングによってクラスタリング効果が悪くなります。このため、クラスタリングキーワードをむやみに長くしてノード間相関係数を高くしすぎると、むしろクラスタリングの効率を下げることになり、逆効果といえます。ただしその方法は、特定のキーワードを知らないと参加できないという閉鎖的なクラスタを作る目的に利用できるかもしれません。

なお、Winny 1 β の開発中に、試みとしてノードが保有しているキャッシュファイルの情報をクラスタリングに用いてみました。これは、キャッシュファイルと各ノードのクラスタリングキーワード間の相関係数を求めることにより実現できます。しかし、ユーザーにとっては簡単なルールのほうがクラスタの選択が容易であり、結果的にはそのほうがクラスタリング効率が良かったので、最終的には現在の方法に落ち着きました。

4.3 クエリ管理

Winny は、前述の方法でできた検索リンクのネットワークを用い、キーの拡散と検索を行います。拡散と検索のいずれの場合でも、キーの送信側はクエリパケットにキーをパックして送り出し、受信側はクエリパケットからキーを取り出します。このため、Winny ではキーの拡散も一種のクエリとして扱う実装となっています。すなわち、ノード間のキーの受け渡しにはすべてクエリを使い、Winny のクエリ管理部はノード管理とキー管理の受け渡しを行います。

拡散クエリ

Winny は 30 秒に 1 回、検索リンクを張っている隣接ノードに対して拡散クエリ要求を送信します。隣接ノードはこれに対し、応答としてクエリパケットにキーをパックして送り返します。Winny は、このような方法でキーを Winny ネットワークに拡散させていきます。実際には、キーの拡散は転送回数を限定した検索クエリ

であり、拡散と検索は同じ仕組みで実現されています。

拡散クエリにパックするキーの個数は、接続する2つのノードのうち回線速度の遅いほうにあわせて変わります。この詳細については、キー管理の項で述べます。

キーの拡散というと、キーの送信側が受信側に対して強制的に送り込むプッシュ型の方法がまず思い浮かびます。しかしこの方法では、相手が忙しくて受け取れないときにも無理にキーを送り付けることになり、相手ノードの通信バッファが溢れてダウンしてしまうかもしれません。Winnyでは、受信側の拡散クエリ要求に対して送信側が応答のクエリを返信するプル型の方法を採用しており、送受信双方のノードが余裕のあるときにキーを送受信できます。

検索クエリ

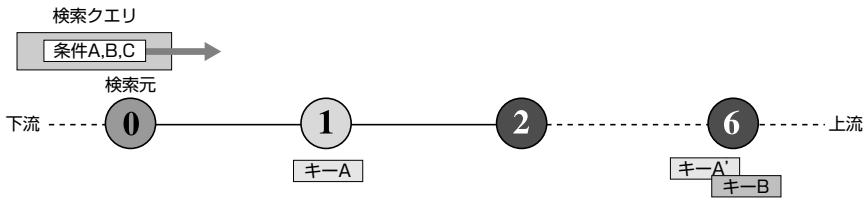
拡散クエリ要求は隣接ノードに対して送信され、キーをパックした拡散クエリが自ノードに戻ってきます。これに対して、検索クエリは一定の条件を満足するまで遠方のノードへ転送され、その過程で検索条件にマッチするキーをクエリにパックして戻ってきます。実際にどのように検索が行われるか、例を挙げて説明したものが図4-15です。

まず検索元ノードでは、ユーザーが指定したキーワードを検索条件としてクエリに詰め、隣接ノードに送り出します。クエリを受け取ったノードは、その条件にマッチするキーがあれば、そのキーをクエリにパックし、さらに隣のノードに転送します。クエリ内のキーの個数が限界に達すると、ルートを逆に遡って検索結果のキーを詰めたクエリが検索元ノードに戻ります。

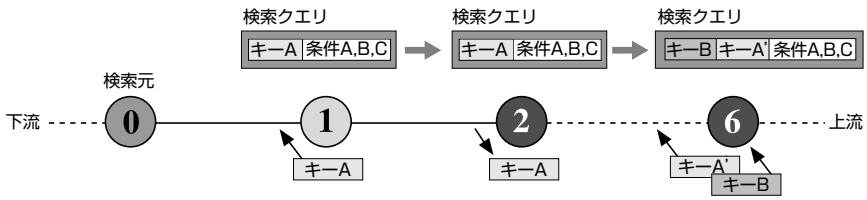
検索クエリは、上流方向のノードに向けて送信されます。上流ノードが下流ノードにあるキーを検索するためには、基本的に下流ノードからの拡散クエリによって送られてくるキーを蓄積し、溜め込んだキーに対して検索を行うことになります。

ただし、最上流近辺のノードでは拡散クエリでしかキーが受け取れないため、この方法では特定のキーワードにマッチするキーを取得したい場合に効率が悪くなります。そのためWinnyは、検索時に1つ下流のノードに対しても、特別な検索条件を付けて拡散クエリ要求を送信します。このクエリは拡散クエリの一種であるため、それ以上下流には転送されず、すぐに検索結果が返ってきます。

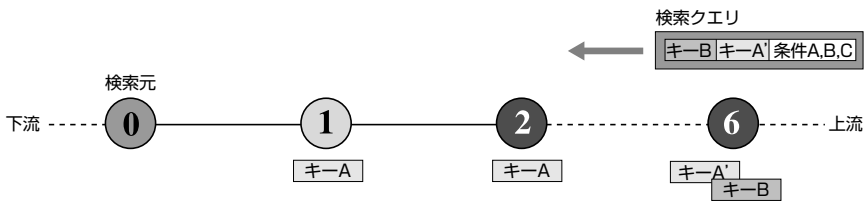
●検索クエリは上流に向かって送られる



●条件にマッチするキーがあればクエリに詰める



●転送を6ホップ、または30個のキーをパックすると、クエリは経路を逆にたどって検索元に送り返される



●クエリが通過するノードには途中で収集したキーが取り込まれる

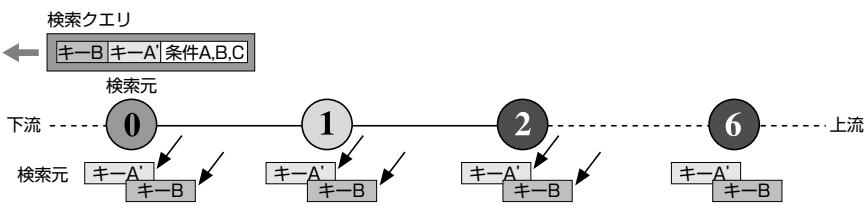


図 4-15　検索クエリ

検索クエリの転送ルールを以下にまとめます。

- クエリは基本的に検索リンクの上流へ伝わる。
- 検索が終了するのは、
  - ◇ それ以上は上流ノードがない（ただし 1 つ下流には送れる）。
  - ◇ クエリを発行したノードを基点として 6 ホップ先のノードまで。
  - ◇ 経路がループした（次ホップがすでに通過したノード以外にない）。
  - ◇ クエリにパックしたキーの個数が最大数（30 個）に達した。
  - ◇ ルートが複数ある場合、クエリを 1 回処理するごとに検索ルートを順に変える（2 度目でルートが変わる）。
- Winny ではクエリは増殖しない。
- 直接の子ノードに対しては、検索条件付きの拡散クエリで別に問い合わせる。

なお検索の過程で、同じハッシュ値のキーが複数のノードで見つかる可能性があります。このときは、先に見つかってクエリにパックされたキー内のファイルの位置情報が、あとから見つかったキーで上書きされます。このため、検索元ノードがあるファイルについて 1 回の検索クエリで見つけることができるのは、1 つのキーだけ――得られるファイルの位置情報は 1 ノードだけです。

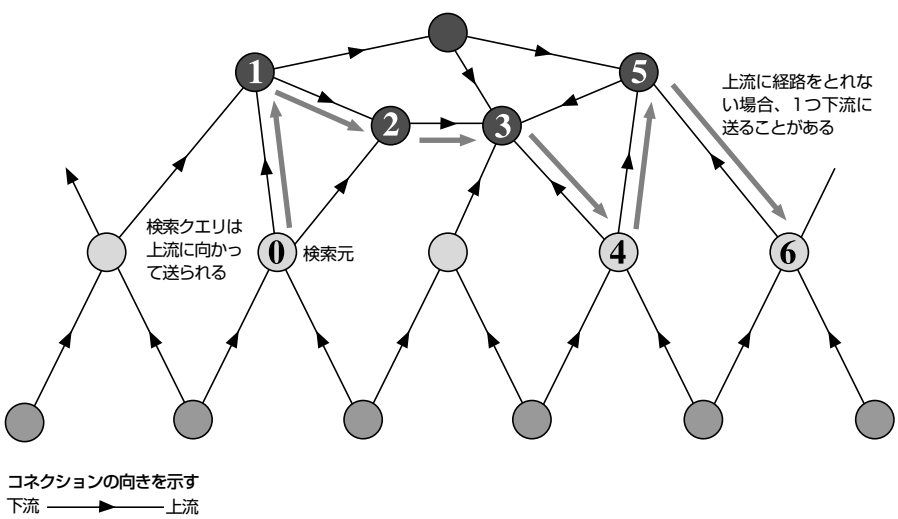


図 4-16　検索ルート（最初の検索）

図 4-16 を見ながらもうすこし説明します。検索クエリは、検索リンクの上流に向かって転送されていきます。図 4-16 では、ノード 0 から送り出された検索クエリが、ノード 1、ノード 2、ノード 3 へと転送されています。また、上流に適当な転送先ノードがない場合には、1 つ下流に条件を満たすノードがあれば、そちらにクエリを転送します。たとえば、図 4-16 のノード 3 では上流ノードが存在しないので、例外的に下流のノード 4 へクエリを転送しています。

また、クエリが同じノードを循環しないように、クエリの転送先は一度転送されたノードを避けて選びます。図 4-16 の検索クエリのあとで、ノード 0 が再度検索クエリを送信すると、検索ルートは図 4-17 のようになります。

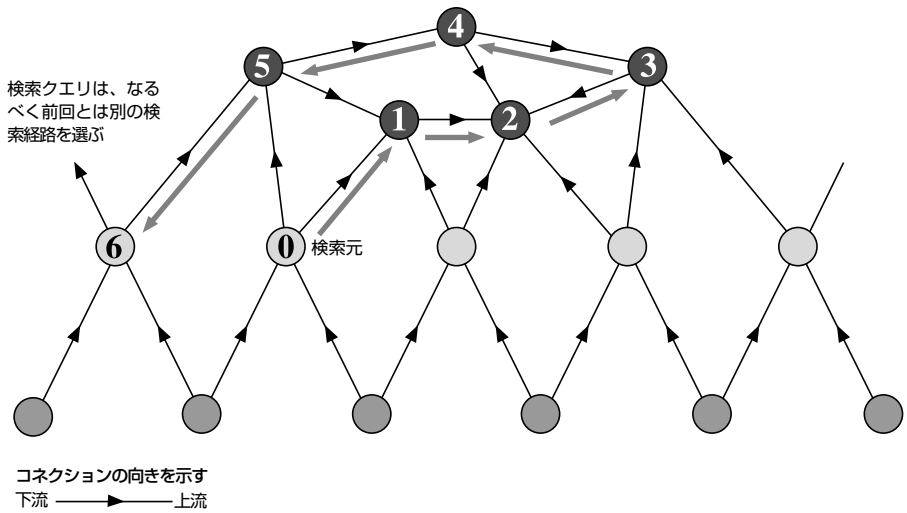


図 4-17　検索ルート（2 回目の検索）

これらのルールにより、検索は上流ノードを中心に検索が行われることになります。Winny では、拡散によってキーがおもに上流に広まっているため、このように上流を指向した検索だけで好成果が得られるのです。

増殖しないクエリ

Winny では、このように 1 つのクエリが次の探索ノードを選びながら転送されていきますが、Gnutella のようなピュア P2P 型のファイル共有ソフトの多くは、送信先ノードに複数のノードが接続されていると、そのすべてにクエリを転送し、ノードを経由していくにつれて流通するクエリパケットの数が倍々に増えていきます。Winny でも開発途中で何度かその方式をテストしました。しかし、通信量の増大や自動ダウンロード機能などとの兼ねあいもあり、最終的にはクエリの探索数を増やさずに順に検索していく方式を採用しています。

Winny が採った方法では、1 回の検索で少ないノードしか検査できません。しかし、あらかじめ検索経路にキーが拡散されているという前提があり、クエリは検索を実行するたびになるべく別のルートを経由するようになっているため、ユーザーは検索結果に満足できなくても何度か検索を繰り返せば比較的よい結果が得られます。

1 回の検索で全ノードを高速に探し、検索率を落とさず、また通信量が増大してネットワークが溢れることがないような方法があればよいのですが、ネットワーク規模が大きくなれば、すべてを満足させることはできません。そのため、Winny は全体的な検索率を犠牲にし、その代わり Winny ネットワークのノード数がどんなに増えても検索速度や通信負荷はそのまま維持できるようにしたのです。

このあたりは、実際に実装してユーザーにテストしてもらい、そのフィードバックに対応していった結果によるものです。Winny には、このような方法で導入された機構が数多くあります[*4]。

*4 こうした開発・テスト手法に関しては、5 章「P2P ソフトの開発手法」で取り上げることにします。

コラム [検索] ボタンを連打すると……

Winny は、[検索] ボタンを押したときにクエリを複数個発行しますが、実は [検索] ボタンを連打するとこの数が減っていきます。これは、検索結果に満足できなかったユーザーは [検索] ボタンを連打するだろうと想像して実装したものです。

もともと Winny では、自動ダウンロード機構があるために、クエリの発行が比較的頻繁に生じます。また下流のノードでは、上流ノードに比べてキーを見つけにくくなるので、ユーザーが検索ボタンを連打しがちです。そこで、[検索] ボタンの連打によって、限られている帯域が多量のクエリパケットで溢れるのを防ごうとしたのです。

4.4 キー管理

ユーザーがファイルをアップロードフォルダに置くと、Winny はファイルからキーと暗号化したキャッシュファイル形式の本体（ボディ）を生成します。キーに含まれている情報を表 4-2 にまとめます。

表 4-2　キー情報

インデックス情報	備　考
ファイル名	
ファイルサイズ	
ファイル ID（ファイルの内容の MD5 ハッシュ値）	
公開者情報（トリップ：片方向関数で生成された公開者認証文字列）	
ファイル流通量（参照された回数）、最終アクセス日時	
キャッシュのビットマップ情報	
キーのバージョン情報	
ファイル本体の位置情報	IP アドレスとポート番号

キーはメモリ上のキーテーブルに置かれ、他のノードからの拡散要求や検索クエリによって Winny ネットワーク内のノードに広がっていきます。ファイル本体は、他のノードからの転送要求によって広まっていきます。

Winny は、ファイルの公開時に生成したキーだけでなく、拡散・検索要求で他のノードから取得したキーもメモリ上のキーテーブルで一元的に管理しています。アップロードしたファイルやダウンロードしたファイル、もしくは中継によって暗黙のうちにダウンロードしたファイルについては、キーに対応する本体がキャッシュフォルダ内にキャッシュファイル形式で保管されています。逆にいえば、ノード内にキャッシュファイルがあるとき、そのキーは必ずメモリ上にあることになります。

キャッシュファイルは、ユーザーが削除しないかぎり消えることはありません。キャッシュファイルには、メモリ上のキーに対応しているヘッダ部が付きます（図 4-18）。

以降では、キーの流通に関してもうすこし詳しく見ていきます。

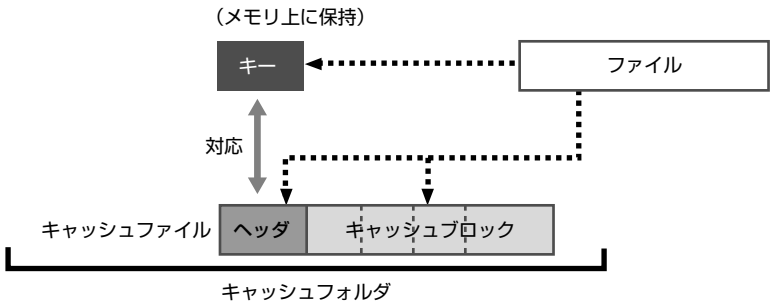


図 4-18　キー情報とキャッシュファイル

キーの拡散

キーは基本的に、上流ノードからの定期的な拡散要求によって上流に拡散していきますが、わずかながら下流にも拡散します。下流への拡散は、試行錯誤のテストの結果採り入れたものです。

Winny が 1 分間にキーをどのくらい拡散しているかを示したものが図 4-19 です。隣接ノードにどのくらいキーを転送するかは、ノードの回線速度によって変わります。具体的な 1 分間の転送個数は、低速側ノードの回線速度を 40K バイト/秒で割り（端数は切り捨て）、その値を 120 倍した数です。ただし、上流に向けた拡散では、1 分間のキーの転送個数は最低 1200 個です。

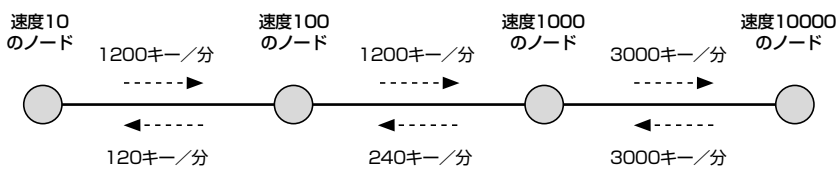


図 4-19　キーの拡散

回線速度の速い上流ノードは、上流と下流のいずれにも非常に高い頻度でキーを送ります。しかし、回線速度の遅い下流ノードになると、上流にキーを送るだけで、上流から拡散されるキーはあまり受け取らなくなります。

この仕様の根拠となっている理由は次のようなものです。

● **下流では、キーを大量に拡散させると帯域が不足する**

回線速度の速いノードであれば、キーを大量に転送しても通信路が溢れることはありません。しかし回線速度が遅いと、キーを大量に転送させただけで帯域を使い切ってしまい、ファイル本体を転送できなくなります。

● **下流のノードが持っているキーはそれほど多くない**

下流のノードは拡散によるキーの蓄積をあまり望めず、キーの発見はキーワードを指定した検索に頼ることになります。すなわち、そもそも下流ノードが持っているキーは数が少なく、その多くは自身がキャッシュファイルを持っているものか、もしくは検索の結果受け取ったキーです。このため、上流と同じ率で上流に向かってキーを拡散しても、通信負荷はそれほど高くなりません。

● **上流ノードに集まったキーを検索することでシステムが機能している**

どのノードも上流に向けて検索をするので、下流のノードが上流のノードへ大量にキーを送らないと、下流にしかないキーは誰も発見できません。

● **上流ノードもキーのリフレッシュが必要**

上流ノードは、回線速度はたしかに速いのですが、メモリや CPU 性能が高いとはかぎらず、結局保持できるキーの数には限界があります。保持できるキーの数はユーザーの設定で変えられますが、数万個程度です。つまり、上流ノードでは常に大量のキーが流入するとともに、受け取ったキーはすぐに溢れて消えていき、ノードが保持できる数よりも多くのキーが循環する形になります。このため、一度検索して見つからなくても、何度か検索を繰り返せば見つけられる可能性が高くなります。

以上をまとめれば、上流のノードは、定期的な拡散で流れ込むキーが大量に蓄積しているので、他のノードを検索する必要があまりありません。メモリ上のキーが見つかってファイル本体をダウンロードすることが多くなります。一方、下流のノードは、メモリ上にはそれほどキーが蓄積されないので、キーを大量に持っている上流ノードに頻繁に検索をかけ、それで得たキーを使ってファイル本体をダウンロードします。Winny ネットワークは、このような仕組みでバランスしているのです。

中継と拡散

Winny の匿名性を実現するカギはファイルの中継にあり、それはキーが拡散する過程で、キー内に含まれているファイル本体の位置情報を自ノードの位置情報に書き換えることによって発生します。この書き換えルールは、全体の効率や匿名性に影響をおよぼすので重要な部分ですが、ルールを複雑にすると思わぬバグを作りかねません。そこで、非常に簡単なルールを適用しています。

キーの書き換え

Winny は、持っているキーをクエリパケットに詰めて他のノードに送り出すとき、そのキーのキャッシュファイルを何パーセント保有しているかによって一定の確率で IP アドレスを書き換えます。これによって中継が発生します。中継は、キャッシュブロックをたくさん持っているファイルほど生じやすくなります。

まず、完全なキャッシュファイルを持っている場合は、かならずキー内の IP アドレスが自ノードのものに書き換わります。これは、アップロードファイルを持ってる場合も同様です。

キャッシュファイルの保有量が少なくなると、書き換えの確率が下がっていきます。まったくキャッシュファイルを持っていない場合は書き換え率が 4%程度になりますが、0%にはなりません。そのため、まったくキャッシュファイルを持っていなかったノードにも、わずかな確率ながら中継が生じ、アップロードをする可能性ができます。

実際に転送が生じる確率は、他の要素の状況に応じて高くなります。次にこのメカニズムについて説明します。

要求の多いファイルを上流ノードにキャッシュさせる

これまでに説明したように、キーの拡散はホップ数を限定した検索として実現されているので、検索と拡散のどちらの場合でも、ノードに届いたキーは同じ確率でキー内の IP アドレスが書き換わります。

このため、検索クエリに乗って頻繁に流通しているキーほど IP アドレスが書き換えられ、中継の発生率が上がっていきます[*5]。また、上流ノードになるほど検索クエリはたくさん流通しているので、上流ノードのほうが中継の発生が多くなります。

*5 問い合わせに対してキー情報を送り返す回数が増えると、キーの書き換えの確率が上がり、結果的に中継の発生率が高くなります。

必然的に、頻繁に検索されるような要求の多いファイルは中継によりキャッシュファイルが広まり、またアップロード能力の高い上流ノードに大量にキャッシュファイルが溜まるという効果もあるので、ネットワーク全体の共有効率はよくなります。

上流ノードで中継が多量に発生すると、中継のために上流ノードへのダウンロードが生じて帯域が圧迫され、上流ノードのユーザー自身が望むようなダウンロードができなくなってきます。しかし、各ノードにある程度キャッシュファイルが溜まったあとは、他のノードからの要求に応じてキャッシュファイルを送信するだけになるので、ダウンロード帯域があくようになります。

Winny ネットワークをうまく機能させるためには、あまりアップロードをしていない余裕のある上流ノードにも、積極的に要求の多いファイルのキャッシングに参加してもらいたいところです。キャッシュファイルを持っていないノードのキーの書き換え率は低いのですが、上流ノードであれば頻繁にキーが流れてきて中継の発生率が高くなり、要求の多いファイルはかなり高い確率でキャッシングされることになります。

しかしこのようにしていくと、上流ノードはどんなにキャッシュファイルを溜めても、持っていないファイルの中継が生じることになります。そこで、アップロード帯域の限界をアップロード本数で判断し、限界に達している場合はキー内の IP アドレス書き換えを抑止します。この対処によって、ファイルをダウンロードしようとするノードが上流ノードに接続したものの、上流ノードにアップロード帯域がなくて無駄な接続・切断を繰り返すのを防げます。

キャッシュファイル

中継によって他のノードから取り寄せたファイルは、キャッシュフォルダに置かれ、他のノードからの要求に応じて転送されます（図 4-20）。

キャッシュフォルダは、他のノードへのアップロードが可能なキャッシュファイルを収容しているプールです。Winny の概念では、アップロードフォルダに入れて公開されたオリジナルファイルについても、同じようにファイル本体のキャッシュファイルが置かれます。しかし、アップロードフォルダにオリジナルファイルがある場合、キャッシュフォルダにも同じ内容のファイル本体を持つことは無駄なので、実装ではキャッシュフォルダにこのあと説明するヘッダ部分のみを置き、ファイル本体はオリジナルファイルを参照するようになっています（図 4-21）。

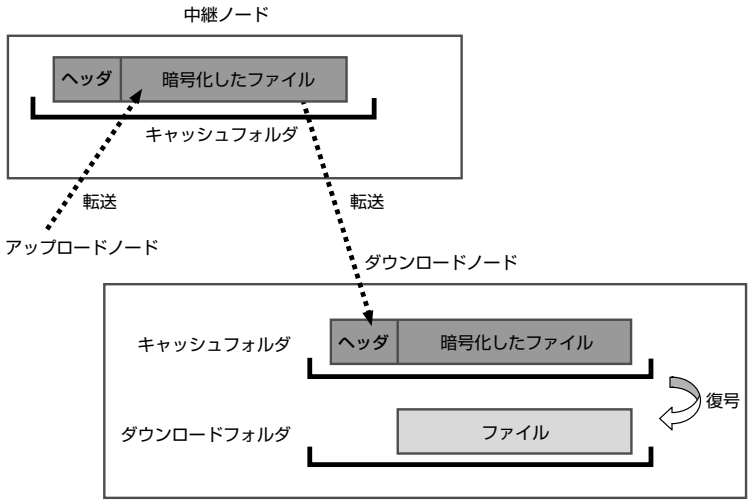


図 4-20　ダウンロードファイルとキャッシュファイル

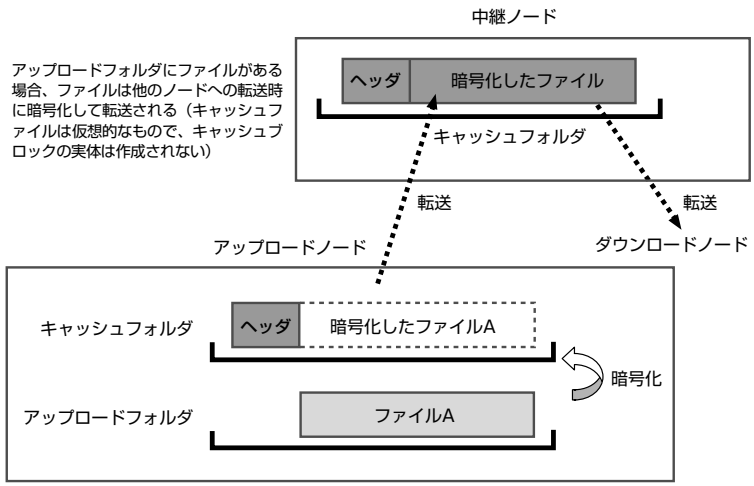


図 4-21　アップロードファイルとキャッシュファイル

このように、アップロードフォルダで公開しているファイルなのか、中継によって蓄えられたキャッシュファイルなのかによって、ファイル本体の所在に違いはありますが、他のノードから見れば受け取るキャッシュファイルは同じです。どちらなのかを意識することはありませんし、判断することもできません。

キャッシュブロック

キャッシュフォルダ内にあるキャッシュファイルは、ファイル本体を暗号化したもので、先頭にファイルの内容を要約したヘッダ部が付いています。ヘッダ部は、メモリ上にあるキーと対応しており、ほぼ同等の内容になっています。

Winny は、ファイルの任意の部分を任意の順番でダウンロードできるようにするため、キャッシュファイルを 64K バイトの固定サイズのブロックに区切って管理しています。つまり、大きなファイルであれば、先頭から 64K バイトずつ区切ったブロックを単位として、アップロードやダウンロードを行います。またこのため、1 つのファイルを複数のノードから同時に並行してダウンロードすることができます。

Winny は、細分化したファイルのどの部分をノード内に持っているか、またどの部分が欠けているのかを知っている必要があるので、メモリ上のキーのなかにキャッシュ保有状況を示すビットマップ情報を持っています。これは図 4-22 のような構造になっていて、該当部分のキャッシュがあれば“1”、なければ“0”になります。

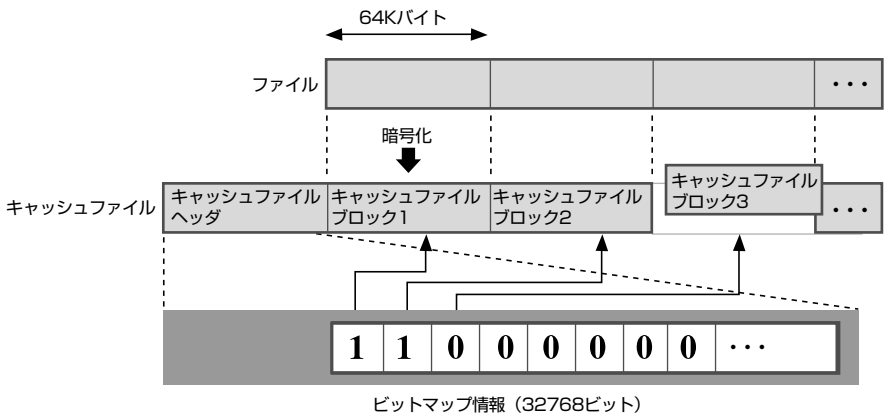


図 4-22　キャッシュファイルとキャッシュブロック

Winny で扱えるファイルの大きさは 2G バイトまでです。このため、1 つのファイルのキャッシュブロック数は最大で 32768 個となり、キャッシュヘッダにはこれだけの大きさのビットマップスペースが固定サイズで確保されています。しかし、多くのキーはキャッシュファイルがノード内になく、ビットマップが示すキャッシュ

保有状況は“0”になっているので、無駄なビットマップスペースを省くために、メモリ上のキーでは可変サイズの構造体となっています。

キャッシュファイルの構造

キャッシュファイルの内部構造をもうすこし詳しく見てみましょう。キャッシュファイルはヘッダ部とキャッシュ本体から構成されます（図 4-23）。ヘッダ部の内容はメモリ上のキーとほぼ対応していますが、キーに含まれているファイル本体の位置情報（IP アドレスとポート番号）などは含まれていません。

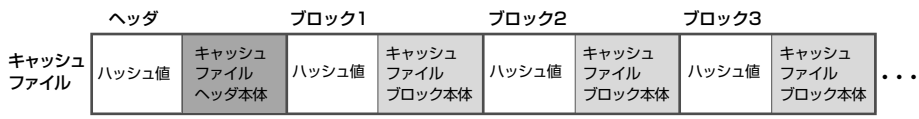


図 4-23　キャッシュファイルと誤り検出用のハッシュ値

キャッシュファイルの本体には、転送時の破損を検出できるように、各ブロックの先頭に誤り検出用のハッシュ値が挿入されます。この値は、ブロック全体から算出した 128 ビットの MD5 ハッシュ値です。もっと簡単な CRC でもよいのですが、この部分の計算処理負荷はそれほど高くないので、ファイル ID の生成などにも使用している MD5 を利用しています。

また、ヘッダの先頭にはヘッダ全体から算出したハッシュ値が付加されます。キャッシュファイルは頻繁にアクセスするものなので破損の可能性は無視できません。また利用者による改ざんを防ぐためにも、異常を検出して対処できるようにしたのです。

同様に改ざんを防ぐために、キャッシュヘッダやキャッシュブロックそれぞれを暗号化しています。Winny では、このときの暗号化に RC4 を使用しています。Winny がどのような部分に暗号技術を使用しているか、またその効果については、5 章「P2P ソフトの開発手法」の「Winny と暗号技術」をご覧ください。

また、キー内に格納されているファイル ID はオリジナルファイルの内容全体のハッシュ値なので、復号したファイルのハッシュ値と比較すれば、オリジナルファイルと同じかどうかを確認できます。また、キャッシュブロックのハッシュ値と内容を別のものに入れ替えれば、キャッシュファイルからオリジナルファイルへの復

元時に、ファイル内容全体から求められるハッシュ値とファイル ID が異なるので、エラーとして検出できます。

キャッシュブロックの保有状況とキーの状態

ノードがキーを持っているときでも、ノード内にはかならずしもキャッシュファイルがあるとは限りません。キャッシュファイルがあったとしても、キャッシュブロックは一部だけしか持っていないかもしれません。また、アップロードフォルダにファイルがある場合、そのキーは完全なキャッシュファイルを持つものとみなせますが、このとき実際にキャッシュフォルダに入っているのはファイル本体のないヘッダのみです。このように、ノードがキーを持っている場合であっても、保有しているキャッシュファイルの状況はさまざまなので、Winny は内部的に表 4-3 のような 4 種類の区分を設けています。

表 4-3　キーの状態

キーの区分	キャッシュファイル	キャッシュブロック保有率
仮想キー	×	0%
部分キャッシュキー	○	0%以上、100%未満
完全キャッシュキー	○	100%
アップロードファイルキー	○	0%、アップロードフォルダにファイル本体の実体がある

仮想キーとは、キーの拡散や検索クエリによって得たキーをノード内に持っているが、対応するキャッシュファイルはない状態です。部分キャッシュキーは、キャッシュファイルを持っているものの、キャッシュブロックが完全には揃っていない状態です。部分キャッシュキーの場合でもキャッシュブロックの保有率が 0%となることがあり、キャッシュフォルダにキャッシュファイルヘッダだけがあるという状況になります。しかし、ヘッダだけであってもキャッシュファイルは存在しているので、仮想キーとは区別されます。

ユーザーは、自ノード内のキャッシュファイルの保有量を検索でヒットしたキーから知ることができます。しかし、他のノードのキャッシング状態を知ることはできません。Winny は匿名性を保つため、クエリパケットにキーを詰めて他のノード

に送るときに、キーの区分やキャッシュファイル保有率に関する情報は除外し、キーの状態を仮想キーとしてセットしているからです。またユーザーは、自ノード内にあるキャッシュファイルの一覧を無条件に見ることはできません。

キーの上書きルール

拡散や検索によって他のノードから受け取るキーは、すべて仮想キーです。このため、自ノードに部分キャッシュキーや完全キャッシュキーがすでにあるとき――つまりノード内に当該キャッシュファイルが一部でもある場合には、自ノードのキーを上書きしないようにしなければいけません。しかし、部分キャッシュキーを持っている場合は、そのファイル位置情報からダウンロードができなくなっている可能性もあります。同じハッシュ値の仮想キーを受け取ったとしたら、そのキーのIPアドレスやポート番号だけは書き換えないと、キャッシュファイル本体をダウンロードするせっかくの機会を失ってしまうことになります。そのため、キーの各要素を一定の上書きルールにしたがって書き換えます。このルールのおもなものを挙げます。

- キーの識別に使用するハッシュ値――上書きしない。
- キャッシュファイル本体のIPアドレスとポート番号――上書きする。
- ファイル名――アップロードファイルキーや完全キャッシュキーには上書きしない。それ以外は上書きする。
- トリップ――自ノードのキーにトリップがないときだけ採用。すでにトリップがある場合は上書きしない。
- 更新日時――新しいほうを優先して上書きする。
- 被参照量――値が大きいほうを優先して上書きする。
- キーの状態――以下の優先順で上書きする。

  アップロードファイル ＞ 完全キー ＞ 部分キー ＞ 仮想キー

- キャッシュファイルのビットマップ情報――上書きしない。

キーに含まれているファイルの位置情報は、1つのノードのみです。ただし、メモリ上のキーに含まれているファイルの位置情報は、拡散と検索によって時間とともに頻繁に書き換わっていき、その一方で自動ダウンロード機構はメモリ上のキーを

走査し続け、条件にマッチするキーを見つけるとダウンロードタスクを生成していきます。この結果、同一キーのファイルを持つ複数のノードそれぞれにダウンロードタスクが生成され、並行してダウンロードが行われることになります。これが多重ダウンロードです。しかし、同じノードに対して複数のダウンロードタスクが生成されることはありません。

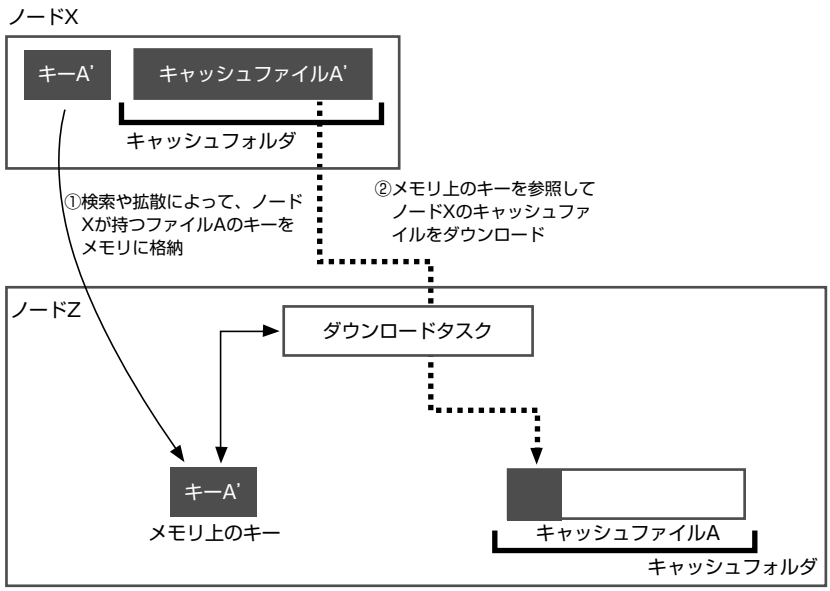


図 4-24　メモリ上のキーの書き換えとダウンロードタスクの起動

図 4-24 は、ノード Z にファイル A のキーが蓄積され、検索条件とマッチすることを検出した自動ダウンロード機構がダウンロードタスクを起動したところです。キー内の位置情報を元にして、ダウンロードタスクはノード X からファイル A をダウンロードしています。図 4-25 は、その後ファイル A のキーが書き換わり、再び検索条件とマッチすることを検出した自動ダウンロード機構がもうひとつダウンロードタスクを起動したところです。ノード X と並行してノード Y からもファイル A をダウンロードしています。

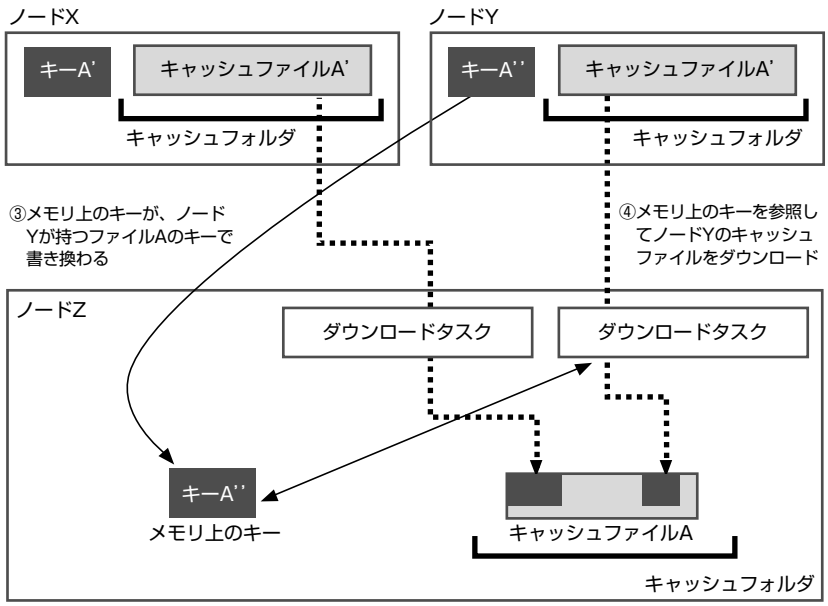


図 4-25　多重ダウンロード

キーの寿命と削除

オリジナルファイルやその完全なキャッシュファイルを持っているノードがすべてオフラインになると、そのファイルはダウンロードできなくなるので、対応するキーを Winny ネットワーク全体から削除する必要があります。

ファイルは断片化されたキャッシュファイルで流通していますから、ネットワーク上に分散しているキャッシュファイル断片を組み合わせて元どおりの完全なファイルを再構成できる可能性は残っています。しかし Winny では、ファイルを完全な状態で持っているノードがなくなった時点で、ダウンロードが不可能になったと判断して処理します。

タイマーを使ったキーの削除

では、ダウンロード不能となったキーはどのように削除するのでしょうか。Winny はこのために「キーの寿命」を設けています。すなわち、キーのそれぞれに一定値のタイマーを設け、定期的に減じて“0”になったらダウンロード不可能とみなすの

です。

拡散や検索の際に、クエリが完全なキャッシュファイル（オリジナルファイルかもしれない）を持っているノードを経由すると、クエリにはタイマーが一定値の新鮮なキーが詰められます。このクエリを受け取ったノードはキーのタイマーがリフレッシュされ、キーは延命します。しかし、完全なファイルを持っているノードが周囲になくなると、キーの寿命が尽きてしだいにネットワーク上のノードから消えていき、最終的にダウンロードはできなくなります。キーの寿命を長く設定すると、キーが広範なクラスタに分布することになり、各ノードが保持するキーが増えすぎて溢れることになります。キーの寿命の初期値は、現在約 1500 秒程度に設定していますが、これは実際にテストして調節した結果です（図 4-26）。

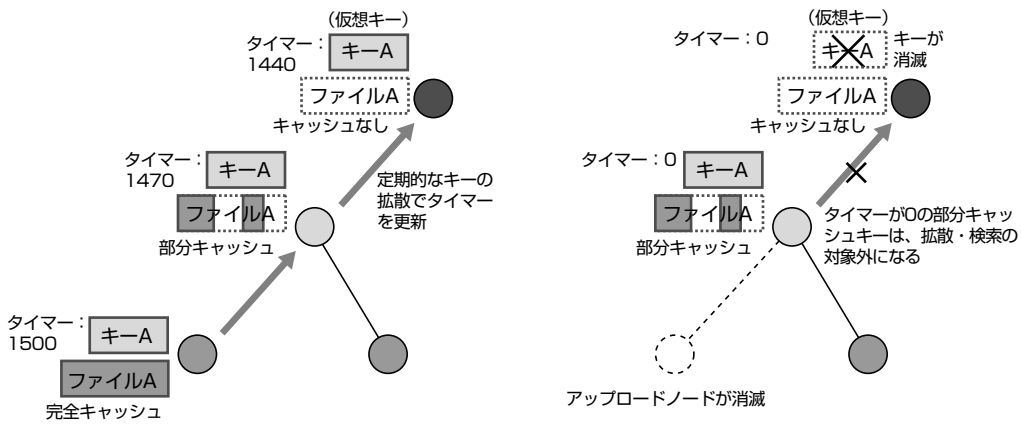


図 4-26　キーの寿命と削除

タイマーという手段の是非

キーを削除する方法としてはもうひとつ、どのキーがどのノードから送信されたキーかを記録しておき、そのノードがオフラインになったら削除するという方法も考えられます。これは、ファイルの存在を管理する正確な方法ではありますが、ノードの状態が頻繁に変化するのでノードの生死を正確に把握するのは難しいこと、また他ノードにはできるだけ情報を出さないようにしたかったことから、この方式を採りませんでした。

寿命が尽きたキーは、キーの状態によって取り扱い方が変わります。仮想キーで

あれば、タイマーが“0”になった時点で消滅します。部分キャッシュキーの場合はダウンロード不可となりますが、キャッシュファイルが存在するのでキーは消滅しません。また、他のノードに送信されることはなくなります。

アップロードファイルキーや完全キャッシュキーの場合は、そもそも他のノードからキャッシュファイルをダウンロードする必要がなく、キーが消滅することもないので、このタイマーを使うことはありません。

4.5 ファイルの転送

Winny の自動ダウンロード機構は、ダウンロード条件に合うキーがメモリ上のキーテーブル内に見つかると、その都度ダウンロードタスクを生成してファイルを取り寄せます。Winny は、ファイルをキャッシュファイル形式で転送します。転送の間は最低限の暗号化が施され、受信側は MD5 による誤り検出情報を使って、転送時のファイルの劣化や改ざんを検出できます。

2 章「Winny 紹介」でも説明したように、Winny はファイルをダウンロードするたびに、アップロードノードとの間で転送リンクのコネクションを張ります。転送リンクは、ノード間を常に接続している検索リンクとは別のものです。

キャッシュファイルは 64K バイトのキャッシュブロックの並びであり、ダウンロードやアップロードの際にはキャッシュブロックを 1 つずつ送受信していきます。キャッシュブロックは暗号化され、誤り検出情報としてハッシュ値が付けられています。送信側は、キャッシュフォルダ内のキャッシュブロックをいったん復号してハッシュ値を計算し、キャッシュブロックの先頭にある誤り検出情報と照合してから、暗号化されているキャッシュブロックを送信します。一方受信側は、受け取ったキャッシュブロックを復号し、そのブロックから算出したハッシュ値と送られてきた誤り検出情報を照合します。この方法で違いが見つかると、Winny はブロックが破損しているものと判断してブロックを破棄し、ブロックの再送を行います。

キャッシュブロックは非同期に転送される

一般的なデータの送受信においては、送信側が受信側から受信成功応答を受け取ってから次の送信を開始します。しかし通信の遅延が大きいインターネット上では、この方法で大量のデータを送受信したのでは非常に時間がかかります。そこで Winny

では、ブロックの受信に成功してから次のブロックの送信を要求するのではなく、受信側が大きな範囲でブロックの送信を要求し、送信側はどの部分かを示す位置情報とともに各ブロックを順に送信する方式をとっています。ブロックにはファイル内のどの部分かを示す情報が付加されてくるので、「多重ダウンロード」で説明する方法を使って、ファイルを非同期に転送することが可能になります。

また中継ノードは、ノード内にキャッシュファイルを持っていなくても、ファイルの送信要求を受け取ることがあります。

要求されたファイルの完全キャッシュファイルがない場合、Winny はただちにダウンロードタスクを起動して、不足しているキャッシュブロックのダウンロードを開始します。送信の過程で不足している部分が判明するのを待たずに、送信要求を受けた時点でキーの状態からキャッシュファイルが完全でないことを検出し、すぐにダウンロードを実施するわけです。このため、ファイルの先頭部分の送信要求があった場合でも、その後ろに不足している部分があれば、先頭ブロックの送信と並行して後半の部分のダウンロードが開始されます。

このような方法によって、中継時の転送速度が低下するのを防げます。

転送リンクの制御

Winny は、仕組みのうえでは、複数のファイルの任意のブロックを好きな数だけ並行してダウンロード／アップロードできます。しかし、無制限にファイルをダウンロードできるようにすると、特定のノードが Winny ネットワーク内の回線帯域を占有してしまう可能性があり、システム全体の転送効率が悪くなります。そこで、各ノードで同時に張れる転送リンクの数を制御しています。

多くのファイル共有ソフトでは、ファイルのダウンロードやアップロードを許可するかどうかを、ユーザーが手動で操作するようになっています。しかし Winny の場合、ダウンロードの開始や制御はすべて自動ダウンロード機構が行い、ユーザーが制御できるのは切断の指示のみです[*6]。どれだけ並行してダウンロードするか、あるいはどの順番でダウンロードするかはプログラムが制御するので、帯域の使用量をプログラム側で制御可能となり、ネットワーク全体のファイル転送効率を最適化できます。

*6 Winny では、ユーザー操作によるダウンロードも条件を特定した自動ダウンロードによって実装されています。

ダウンロードとアップロード速度のバランス

一般的に、ファイル共有ソフトにおける流量制御の基本方針は、ダウンロード量とアップロード量のバランスをとるというものです。この方法は、各ノードの転送効率やユーザー心理を考慮したものですが、ネットワーク全体の効率という観点では最良とはかぎりません。

現在のインターネット利用者の多くは、下りが速く上りが遅い非対称な ADSL を利用しています。ADSL 回線を利用しているノードにダウンロードを無制限に許せば、光回線のように高速で上りと下りが対称な回線利用者への依存度が高くなります。しかしだからといって、アップロード量に見合ったダウンロード量を許すというルールを厳密に適用すれば、多数を占める ADSL ユーザーのダウンロード量を制限することになり、結果的にネットワーク全体のダウンロード総量は少なくなります。このため Winny では、低速回線ノードへのルールの適用を緩和しています。

高速なノードはアップロード量の割にはダウンロード量が少ないので、多少割が合いませんが、ネットワーク全体を通した効率はよくなります。ADSL 回線ノードの数が圧倒的に多いという事情と、光回線ノードのアップロード能力が ADSL 回線ノードの数十倍もあるという状況を考慮すれば、現実的な方法だと考えます。

開発当時の日本では、高速回線のアップロード速度が他の回線より非常に速かったため、Winny では高速回線ノードへの依存度が高くなっています。結果として、現時点では上り・下り非対称の比較的低速なノードに有利なチューニングとなっています。

同時ダウンロード数と同時アップロード数の制御

以上のことから、Winny は次のようなルールで、同時に起動できるダウンロードタスクの数（転送リンク数）を制限しています。

- 何もアップロードしていない状態であっても、同時に 2 つまでファイルをダウンロード可能とする。
- ダウンロード速度は制限せず、回線の下り帯域いっぱいまで利用する。
- 十分な速度でアップロードをしているノードは、高速回線につながっているものとみなし、アップロードに使用する転送リンクの最大数を増やす。具体的に

は、回線速度が 40K バイト/秒速くなるごとに、アップロードに使用できる転送リンク数を増加させる。

- アップロードの転送リンク最大数が増えれば、ダウンロードの転送リンク最大数も増加させる。ダウンロードの転送リンク最大数は、アップロードの転送リンク最大数＋ 2 である。

このようにした根拠を説明しておきましょう。

ISDN 回線などを使用しているノードはアップロードもダウンロードも遅いので、多数の転送リンクを使って同時にダウンロードしようとすれば、細い回線をさらに分割して使うことになり、ダウンロードの効率が悪化するだけです。この場合は、最低限の同時ダウンロード数で十分です。これは、アップロードに関しても同様で、転送リンクは最低限の数で十分です。しかし、転送リンクの最大数を“1”とすると、唯一の転送で問題が生じた場合にまったく転送が進行しなくなる可能性があるので、最低限の転送リンク数は“2”としています。

ADSL 回線などを使用しているノードは、アップロードは低速なもののダウンロードは高速なので、ユーザーにとっては、アップロード速度に関係なくダウンロード帯域をフルに使ってダウンロードできるほうが快適です。ただし、これらのノードが多数の転送リンクを使ってむやみにダウンロードをすれば、高速回線ノード全体のアップロード帯域を占有することになり、高速ノードはダウンロード用の帯域を確保できなくなります。そこで、同時ダウンロード数を最低限の 2 つ程度にとどめ、あとはダウンロード帯域に応じて高速にダウンロードできるようにします。

光回線などを使用しているアップロードもダウンロードも速いノードは、回線速度の遅い多数のノード群から多数のダウンロード要求を受けられるようにしないと、Winny ネットワーク全体の効率を上げられません。そのため、アップロード速度に応じて転送リンク最大数を増やしています。転送リンクの基準転送速度は、低速側ノードのアップロード能力によって決まりますから、Winny では低速 ADSL のアップロード速度を基準とし、40K バイト/秒を 1 回線の速度とみなして、アップロード用の転送リンク最大数を増やします。

ノードの回線速度はユーザーの申告を元にしていますが、申告がかならずしも正しいとは限りません。そこで Winny は、実際にノードのアップロード速度が速いことを確認してから、アップロード側の転送リンク枠を増やします。

また高速ノードがファイルをダウンロードする場合は、複数の低速ノードを束ね

てダウンロードするほうが有利なので、ダウンロード用の転送リンク最大数も同時に増やす必要があります。

アップロード要求が集中したときのリンク切断

アップロード要求が集中する人気ファイルには、中継が頻繁に発生し、あちこちのノードからアップロードができるようになります。しかしそれでも、初期には特定のノードに要求が集中し、転送リンクが多数接続されます。そこで、前項で挙げた制御ポリシーにしたがって同時転送リンク数を制限しますが、限界を超えたリンクの切断に関しても考慮すべき点があります。

アップロード中の転送リンクを切断する場合は、基本的に転送リンクの接続時間が短いほうを選び、接続時間が長いほうは残します。いったんダウンロードを開始したリンクはできるだけ持続したほうが、キャッシュファイルが細分化されず、全体として Winny ネットワーク内に完全なキャッシュファイルを増やせるからです。

しかし、アップロード能力に優れているもっと高速なノードがあるときは、キャッシュファイルをそちらのノードに転送したほうが効率はよくなるでしょう。そこで、接続時間の条件に加えて回線速度も考慮し、自分のノードより相手の回線のほうが速ければ切断しにくくなっています。ただし、ノードの回線速度はユーザーが申告するものなので、過剰な申告があるかもしれません。そこで、自ノードより申告速度が速いかどうかのテストを行っています。自ノードより回線速度が速いノードとの転送リンクは、遅いノードよりもリンク切断率が下がります。

多重ダウンロード

これまで述べてきた制御により、アップロードが速い回線では同時ダウンロード数の枠が増えます。これは、上り帯域が細いノードからの低速なアップロードを複数束ねて速い回線として使おうと考えたからですが、転送リンクの最大数を増やしただけでは多数のファイルを同時にダウンロードできるだけで、1つのファイルを速くダウンロードできるわけではありません。1つのファイルをより速くダウンロードできるように組み込んだのが、多重ダウンロードというメカニズムです。

手分けをすれば速くダウンロードできる

多重ダウンロードは、1つのファイルを複数のノードから同時にダウンロードする機能です。ファイルは細分化されたキャッシュブロックごとに転送されるので、1つのファイルの別々の部分を複数のノードから並行して転送すれば、高速にダウンロードできます。

そのためWinnyは、同じファイルに対して複数のダウンロードタスクを実行できるような設計になっています。すなわち、ファイル送受信の基本機能として多重ダウンロードが自動的に働き、ダウンロードでもアップロードでもファイルの任意のブロックを並行して送受信可能です。

しかし、やみくもに細分化したキャッシュを多重ダウンロードするだけでは、同じ部分のダウンロードが生じて無駄になります。そこで、ノード内に同じファイルをダウンロードしている別のタスクがあれば、そのタスクとはできるだけ離れているブロックを選んでダウンロードするようになっています。

速いタスクがファイル前方からローラー式にダウンロード

ダウンロードタスクの基本動作は、ダウンロード済みのブロックを飛ばしてアップロード要求を出すというものです。また、1つのファイルをいくつかのタスクでダウンロードするときは、ダウンロード速度が速いタスクがファイルの前方から後方に向かって集中的にダウンロードしていく一方で、ダウンロード速度が遅いタスクは、先回りをして後ろの部分を少しずつダウンロードします(図4-27)。高速のダウンロードタスクは、ところどころダウンロードが済んで虫喰い穴があいているようなキャッシュファイルの残りを、高速にダウンロードしていくという図式になります。

他のキャッシュファイル分散方式を考える

Winnyは基本的に、不足しているキャッシュブロックのうち最も先頭にあるものからダウンロードします。ファイル後半だけを先にダウンロードするのは中継転送でファイル後半のアップロードを要求された場合のみです。

そこで、はじめにファイルをアップロードしたノードが別々のノードにファイルの別々の部分をアップロードし、利用者がそれを組み合わせてダウンロードできれば、Winnyネットワーク全体のファイル共有効率がよくなるのではないかとも考え

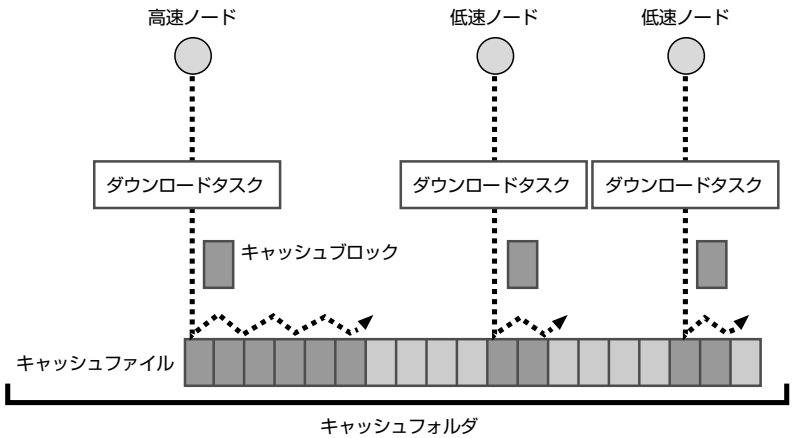


図 4-27　多重ダウンロード

ました。しかし、実際に試してみた結果、どのブロックをどのノードが持っているかがわからない Winny ネットワークでは、細かいキャッシュファイルがあちこちのノードにできるだけで、転送リンクの切断が増えることがわかりました。

Winny の場合、人気のあるファイルはすぐに高速な上流ノードにキャッシングされ、ダウンロードが特定のファイルに集中しても、まもなくファイルが拡散していくので、問題にはなりません。しかし、アップロード能力のないノードがはじめにファイルを公開した場合、そのノードのアップロード速度以上にファイルが拡散することはありません。そのノードにダウンロード要求が集中すれば、結局どの要求ノードも途中までしかダウンロードできないという状況になりがちです。

これを防ぐために、はじめに公開したノードを特別扱いし、周辺ノードからの拡散クエリを待たずに、自ら積極的に周辺ノードへキャッシュファイルを送り付ける方式も検討しました。Freenet ではこのような拡散方式を採用していますが、中継方式との両立が難しいということも考え、Winny では採用を見送っています。

ファイル ID とハッシュ値

多重ダウンロードの仕組みは、複数のアップロード側ノードにまったく同じ内容のファイルがあることが前提となっています。そうでないと、ダウンロードしたキャッシュブロックを連結してできるファイルの整合性がとれなくなります。

このため Winny は、ファイルを識別するファイル ID として、ファイルの内容から算出したハッシュ値を用いています。これは、「ファイルの公開」でも説明したとおりで、128 ビットの MD5 を用いた 16 バイト長の整数です。

ファイルを識別する手段としては、通常のファイルシステムで使われているような名前を使うことも考えられます。しかし、ファイル共有ソフトのネットワーク内では、同じ内容のファイルに別の名前が付けられることも多く、たとえ名前が異なっていても、内容が同じであれば同一ファイルとして扱えるほうが好都合です。ハッシュ値はいわばファイルの内容の要約であり、Winny はハッシュ値を使ったファイル ID によって、複数ノードに存在するファイルの同一性をほぼ確実に識別可能になります。

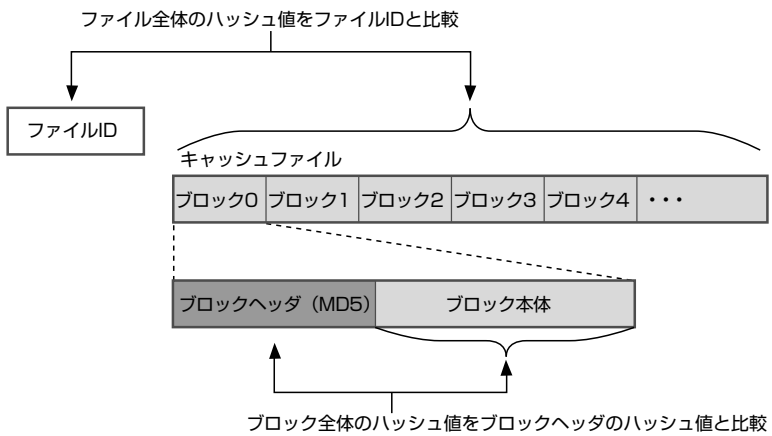


図 4-28　ファイル全体とキャッシュブロックごとのチェック

アップロードファイルとハッシュ値

ファイル ID が作成されるのは、アップロードフォルダに新しいファイルが追加されたときです。Winny はアップロードフォルダに新しいファイルが追加されたことを検出すると、キャッシュフォルダにそのファイルのキャッシュファイルヘッダだけを生成し、そこにファイル名とファイル全体から計算したハッシュ値を格納します。

なお、キャッシュファイル自体のファイル名は、ファイルのハッシュ値を暗号化したものです。Winny は次回に起動したとき、ファイル名から判断して対応する

キャッシュファイルが見つかるので、ハッシュ値の算出を省略します。

大きいファイルのハッシュ値の算出には時間がかかる

ファイルが大きくなると、ハッシュ値の算出に時間がかかるようになります。何が問題なのかというと、CPU の処理時間ではなく、ファイルの読み込みに時間がかかることです。ファイルの内容全部からハッシュ値を求めるとなると、最低でも一度はファイルをディスクからメモリに読み込まなくてはなりません。Winny が扱えるファイルサイズは最大 2G バイトですが、これだけの大きさのファイルを読み出せば数分はかかります。ハッシュ値の算出には最低限これだけかかることになります。

起動のたびにこれだけ時間がかかるのは問題なので、Winny はアップロードフォルダに新しいファイルが追加されたことを検出すると、そのときに一度だけハッシュ値を算出します。ハッシュ値は、バックグラウンドの別タスクで算出し、ほかの作業と並行して進めることができます。このため、アップロードフォルダに巨大なファイルを大量に置くと、ハッシュ値のチェックに何時間もかかる可能性がありますが、それを待つことなくダウンロードなどの他の操作が可能です。

4.6 タスク管理と Windows スレッド

他のノードとの間のアップロードやダウンロード、ハッシュ値のチェックには、時間がかかる可能性があり、それらを並行して処理できるようにプログラムを作成しなければいけません。

このようなときには、OS 側のマルチスレッド機能を使用するのが一般的ですが、マルチスレッドに頼るとあとで面倒なバグが生じることが多く、特にデッドロックの対処やデバッグがやっかいです。このため Winny では、自前で内部のタスク管理をしています。

Winny が使っている方法は、古典的な Window システムなどで用いられているノンプリエンプティブマルチタスクと同様のものです。一定時間で処理を自主的に終了するように各作業タスクを作成し、暇を見て順に実行していくことによって、各タスクは並行に動作します。このような部分をすべて自分でプログラミングすることに、いろいろな意見があるかもしれませんが、タスク間で込み入った連携をする場合でも、タスクの同期などのコントロールが容易になり、結果的にシンプルなシ

ステムにできると判断しました。

しかし、通信を担当する部分とGUIを担当する部分は、別個のWindowsのスレッドとして独立させないと、メニューを開いている間は通信が止まってしまうという問題が生じます。このためWinny 1は、2つのWindowsスレッドで構成し、通信側スレッドの内部で内部タスクを自前に処理しています。

その後Winny 2では、開発環境をC++Builderに替えて、プログラムを一から作り直していますが、このときには通信処理をGUIと分けないでも大丈夫だろうと判断し、マルチスレッドは用いていません。実はその結果、条件によっては通信が止まったり遅くなったりすることがあるのですが、Winny 2ではファイル共有ソフトとしての機能はそれほど重視していないため、そのまま現在にいたっています。

Winnyのタスク管理部は、以下に挙げる各種の作業タスクを起動します。

- アップロードタスク
- ダウンロードタスク
- キャッシュファイル変換タスク
  - ◇ キャッシュファイルからダウンロードフォルダ内のファイルへ変換
  - ◇ アップロードフォルダ内のファイルからキャッシュファイルへ変換
- フォルダチェックタスク
- ハッシュチェックタスク
- クエリ処理タスク

ほかにBBS関連のタスクなどもありますが、これらの詳細は省略します。タスク管理部そのものは、各種の作業タスクを順に実行させているだけのごく簡単なものです。以降では、作業タスクの動作をもうすこし見ていくことにします。

ダウンロードタスク

自動ダウンロード機構は、ノード内に蓄積されているキーを定期的に走査し、条件にマッチするキーが見つかると、それに対応したダウンロードタスクを生成します。ダウンロードタスクはそのとき、ファイルをダウンロードするためのコネクションを1つ張ります。具体的には、キーに含まれているファイル本体のIPアドレスとポー

ト番号を元にして、キャッシュファイルを持つノードと転送リンクを接続します。

転送リンクを接続したあとは、キーのなかのビットマップ情報を参照し、自ノードへのダウンロードが済んでいないキャッシュブロックの送信要求を順に送ります。相手ノードからは、ブロック位置情報とともにキャッシュブロックが送信されてくるので、ノード内のキャッシュファイルの該当部分に書き込みます。

なお、多重ダウンロード機能によって、ノード内には同じファイルをダウンロードしているタスクが複数存在することもあります。

しかし、1つのノードに対して起動できるダウンロードタスクは1つだけです。すでにダウンロードを実行しているノードに対して、新たにダウンロードタスクが起動されることはありません。

アップロードタスク

ダウンロードタスクの役割は、1つのファイルをダウンロードすることであり、ファイル転送が終わると終了します。これに対してアップロードタスクの役割は、他のノードから要求されたキャッシュブロックを送信することであり、他のノードから送信要求を受け取るたびに生成され、要求されたブロックすべての送信が終了すると消滅します。

このタスクは、要求されたキャッシュブロックがキャッシュフォルダ内にあれば読み出して送信し、なければタイムアウトするまでキャッシュブロックが存在するかどうかをポーリングします。これを、要求されたブロックの数だけ順に繰り返します。要求されたキャッシュブロックがないのにタイムアウトまで待つのは、待っている間に中継によって他のノードからキャッシュブロックが読み込まれる可能性があるからです。送信すべきキャッシュが手元にないからといって、すぐにアップロードを中断するわけではありません。

なお、どのキャッシュブロックを送信すべきかは、ダウンロード側ノードが要求するキーのハッシュ値とブロック位置によって示されます。アップロード側ノードは、メモリ上のキーのなかからダウンロード側ノードに要求されたハッシュ値に該当するものを探します。見つかったキーが仮想キーもしくは部分キャッシュキーであれば、ノード内にはキャッシュファイルが完全には揃っていないことになります。そこで Winny は、アップロードタスクと同時に同じキーでダウンロードタスクも起動し、中継が始まります。

キャッシュファイル変換タスク

Winny は、ノード間のファイルのやり取りを暗号化されたキャッシュファイル形式で行います。アップロードフォルダに置かれた公開ファイルは、キャッシュファイルの形に変換して送信されます。またダウンロード側は、受信したキャッシュファイルを、他のノードからのアップロード要求に備えてそのままキャッシュフォルダ内に蓄えます。

ここで、アップロードフォルダ内のオリジナルファイルをキャッシュファイル形式に変換したり、ダウンロードしたキャッシュフォルダ内のキャッシュファイルをダウンロードフォルダ内のオリジナルファイルに復元するのが、キャッシュファイル変換タスクです。両者は別個のタスクですが、どちらもほぼ同じような働きをします。

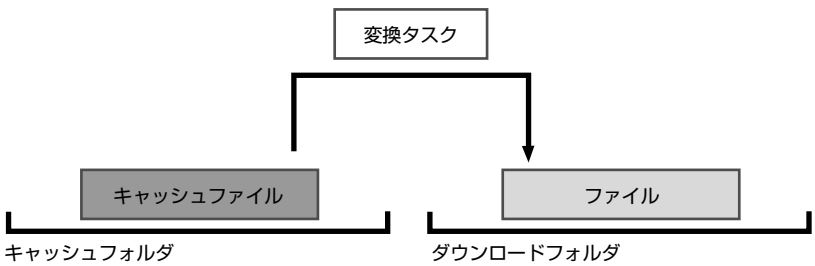


図 4-29　キャッシュファイル変換タスク

オリジナルファイルからキャッシュファイルへ

アップロードフォルダのオリジナルファイルを 1 ブロック分（64K バイト）ずつ読み込み、キャッシュブロックに変換したあと、キャッシュフォルダ内のキャッシュファイルの一部として格納します。この過程では、キャッシュブロックごとのハッシュ値を求め、暗号化したキャッシュブロックの前に挿入します。

Winny 初期の設計では、このタスクはアップロードファイルに対してはじめてアップロード要求があったときに、自動的に起動される予定でした。しかし、これではアップロードフォルダとキャッシュフォルダとでファイル本体を二重に持つことになり、倍のディスクスペースが必要になります。そこで β 1 の公開直前に設計を変更し、他のノードへアップロードする時点でキャッシュファイル形式に変換す

るようにしました。この変更によって、無駄なディスク消費は減らせたのですが、アップロード時の処理が複雑になっています。

なお、ユーザーの操作によって明示的にアップロードファイルをキャッシュファイルに変換し、他ノードからのアップロード要求に対するレスポンスを改善することができます。この用途のために、アップロードファイルのキャッシュファイル変換機能があります。ただし Winny は、キャッシュフォルダとアップロードフォルダに同じファイルが見つかると自動的にキャッシュファイル本体を削除するので、手操作でキャッシュファイル変換をしたら、すぐにアップロードフォルダ内からファイルを取り除く必要があります。

キャッシュファイルからオリジナルファイルへ

通常の設定では、ファイルのキャッシュブロックすべてのダウンロードが完了し、完全なキャッシュファイルができると、キャッシュファイルから元のオリジナルファイルに変換するタスクが自動的に起動されます。

このタスクは、暗号化されているキャッシュブロックを復号し、キャッシュブロックごとのハッシュ値を再計算して破損をチェックしたうえ、他のキャッシュブロックとあわせてダウンロードフォルダ内にオリジナルファイルを組み立てます。

変換のあとも、キャッシュフォルダにはキャッシュファイルが残るので、形式は異なるものの、キャッシュフォルダとダウンロードフォルダには同じ内容のファイルが存在することになります。アップロードファイルからキャッシュファイルへの変換とは異なり、ディスクスペースを無駄にしている仕様となっていますが、Winny はダウンロードフォルダの内容をチェックしておらず、このフォルダの管理はユーザーに任せているため、このようになっています。

変換タスクがキャッシュブロックの破損を検出した場合は、ブロックの状態を示すキー内のビットマップ該当部分を未ダウンロード状態にセットし、キャッシュファイル全体の状態を完全キャッシュから部分キャッシュに変えます。この操作によって、破損部分だけがダウンロードし直され、最終的にダウンロードフォルダにはオリジナルファイルが復元されます。

なお、ファイルをダウンロードしたあとのキャッシュファイル変換タスクの起動は、設定でオフにすることが可能であり、ユーザーは手操作でキャッシュファイル変換タスクを起動し、キャッシュファイルを元のオリジナルファイルに変換することもできます。変換はすべてのキャッシュブロックが揃っていない状態でも可能な

ので、ユーザーはファイルのダウンロード途中でも、一部のキャッシュブロックだけをダウンロードフォルダに復元し、ファイルの内容を確認できます。

復元したファイルの妥当性をチェックする

Winny は、復元したファイル全体のハッシュ値も求め、ファイル ID そのものとなっているハッシュ値と比較することによって、オリジナルファイルと同じかどうかを確認します。そのために、ファイルの内容の信頼性はハッシュ関数として用いている MD5 の信頼性にかかっています。最近、MD5 そのものの脆弱性が見つかっており、これを悪用したファイル内容の改ざんを心配される方がいらっしゃるかもしれません。しかし、元のハッシュ値を変えないように任意の内容でファイルを捏造することは困難なので、これはそれほど問題にならないはずです。将来的には、SHA-1 などのより安全なハッシュ関数に置き換えるほうがよいでしょう。

Winny は、キャッシュファイルブロックの送受信や中継でも、その都度ブロックのハッシュ値を照合してエラーを検出しているので、復元したファイル全体のハッシュ値が合わないことはめったにないはずです。しかし、転送過程のなんらかの問題で、個々のブロックのハッシュ値は合っているのに、復元したファイル全体のハッシュ値が合わないというケースがまれにあります。あるいは、わざとファイルの破損を生じさせるようなキャッシュファイルを送信している悪戯ノードがあるのかもしれません。このような場合は、どのブロックに問題があるのかわからないので、Winny は持っているキャッシュファイルの全ブロックをクリアして再度ダウンロードします。

フォルダチェックタスク

Winny が起動されたときやユーザーがアップロードフォルダを追加したとき、このタスクが起動され、アップロードフォルダやキャッシュフォルダの内容を再スキャンします。アップロードフォルダ内のファイルについては、対応するキャッシュファイルがキャッシュフォルダになければ、後述のハッシュチェックタスクを起動し、ファイル全体のハッシュ値を求めてキャッシュファイルのヘッダ部を作成します。なお、複数のフォルダを同時に調べることは、ディスクアクセスの効率を低下させ、チェックに要する時間を増すだけなので、同時に動作するフォルダチェックタスクは 1 つのみに制限しています。

ハッシュチェックタスク

大きなファイルのハッシュ値の算出には時間がかかるので、それを別タスクとして動作させ、他の作業と並行して処理します。ハッシュチェックタスクはこの処理を行います。

ただし、ディスクにアクセスするような処理を多数並行させても、結局ディスクアクセスが競合し、時間がかかるだけです。このため、ファイルはシーケンシャルに1つずつチェックします。これは、キャッシュファイルをチェックするタスクやキャッシュファイル変換タスクと同じで、複数のタスクが起動されてもそのうちの1つだけが動作するようになっています。

4.7 自動ダウンロード機構

ユーザーがファイルを個別に選んでダウンロードを指示すると、Winnyは特定のファイルのハッシュ値が指定されている自動ダウンロードとして処理します。すなわち、ファイルのダウンロードはすべて自動ダウンロード機構によって起動・制御されます。

自動ダウンロードの仕組み

自動ダウンロード機能は、ダウンロード条件に合うキーがメモリ上のキーテーブルのなかにあるかどうかを定期的に検索し、マッチするものがあればそのキーに入っているファイルの位置情報を使用して、ファイルをダウンロードします。また、キーテーブルにマッチするものがなければ検索クエリを発行します。

メモリ上のキーは、拡散クエリによる隣接ノードからの定期的なキーの取得や、検索クエリによるキーの取得でどんどん入れ替わっていきます。また、検索クエリで探索するルートは検索のたびに変わっていきます。すなわち、ある時にメモリ上にも検索クエリでもキーが見つからなかったとしても、時間がたつと条件に合うキーが見つかる可能性があります。ユーザーは、ダウンロードの条件を指定し、あとは放って気長に待っていれば、自動ダウンロード機能が検索とダウンロードを繰り返し、すこしずつ条件に合うファイルが溜まっていきます。

自動ダウンロード機構は、他のノードに対して頻繁に検索クエリを出す必要があり、ダウンロード条件を複数指定できるようにすると、CPU 負荷やネットワーク全体の通信負荷などが問題になるため、これまでファイル共有ソフトではあまり実現されてきませんでした。

Winny の場合、あらかじめ拡散によってある程度キーが広まっており、検索クエリの頻度を減らせるので、この機構はそれほど負荷とならずに実現できます。また、1 回のダウンロード要求でファイルの先頭から最後までダウンロードしきれないことも多いので、Winny ではダウンロードの再開を自動化できる自動ダウンロード機能が不可欠です。

ダウンロード条件のチェック頻度を制限する

自動ダウンロード機能が多量のダウンロード条件を常にチェックすれば、ノードやネットワークにかかる負荷は高くなります。そこで Winny は、時間あたりにチェックするダウンロード条件の数を 1 秒につき 2 回までに限定しています。ダウンロード条件は一定間隔で順に処理するので、ユーザーが指定した条件の数が増えれば、検索やダウンロードを試行する頻度が下がっていくことになります。

自動ダウンロード機能で指定できる条件は以下のとおりです。

- **ファイル名の一部分**
  空白で区切ると AND 条件、“-”を付けると否定条件。
- **ファイルのハッシュ値**
  ハッシュ値の指定を省略すると、ファイル名条件にマッチするファイルすべてが対象となります。
- **ファイルのトリップ**
  省略すると任意のトリップが想定されます。
- **ファイルの大きさ**
  上限と下限を M バイト単位で指定します。

補助的な仕様ですが、Winny はユーザーからの指定があれば、条件にマッチするファイルをダウンロードしてキャッシュファイルに変換したあと、ダウンロード条件を削除します。これは、条件にマッチするファイルを 1 つだけダウンロードし、

同じファイルが何度もダウンロードされるのを防ぐ目的で使用します。

自動ダウンロード条件は、このあと説明する無視条件と連動しており、無視条件にマッチするキーは、自動ダウンロード条件とマッチしていてもダウンロードは開始されません。

4.8 無視フィルタ機構

無視フィルタは、ファイル名に特定のキーワードを含むものを、自動ダウンロードの対象から除外するという機能です。ダウンロード条件と同じように、無視条件としてファイル名の一部やハッシュ値、トリップ、ファイルサイズを指定すると、ファイルのダウンロードを抑止できます。無視条件に指定したファイルは、中継も行われなくなるので、キャッシュファイルとして持ちたくないファイルの指定にも利用できます。また指定があれば、該当するキーをキャッシュファイルとともに丸ごと削除したり、そのキーを送ってくるノードに無視警告を出します。

無視フィルタとほぼ同等のことは、自動ダウンロードや手操作での検索時に、“-”で始まる検索条件を指定することによって可能です。しかし、複雑なダウンロード条件を指定するのは面倒なので、ダウンロードしたくないファイルのハッシュ値やファイル名文字列を特定できる場合は、無視フィルタのほうが使いやすいでしょう。

この機能を利用すると、たとえばファイル名に“.exe”を含むもののダウンロードを禁じて、正体不明の実行ファイルを実行しないようにしたり、キーワードを指定して自動ダウンロードするファイルのなかから特定のハッシュ値のファイル群を除外する（捏造ファイルを無視する）ことができます。

また無視フィルタを使えば、問題のあるファイルが広がることを防ぐことになります。しかし、Winny のようなピュア P2P 型のシステムでは、この機能を十分に活用するためには、多くのユーザーの協力が必要です。

無視フィルタは、ファイルのダウンロード開始時に無視条件との照合を行います。このため、多量の無視条件を指定すると、Winny 全体の処理速度が遅くなります。また、削除オプションを指定している条件については、定期的に自ノードに保持しているキーすべてと照合します。これは負担の大きい仕事なので、1 秒間にチェックする削除条件の数を 1 つに限定しています。削除指定のない、ダウンロードを無視するだけの条件は、キーと照合する必要がないので省略されます。

4.9 まとめ

Winny のプログラムがどのように実装されているかを、詳しく説明してきました。

- Winny の主要なプログラム要素は、タスク管理、ノード管理、クエリ管理、キー管理を担当する 4 つの部分です。
- Winny はいくつかの作業をマルチタスクで同時に並行して処理しています。タスク管理部は、このタスクの起動や実行状態を管理します。
- Winny の検索からダウンロードまでの過程は、自動ダウンロード機能によって制御されています。自動ダウンロード機能は、ノード内に蓄積されているキーを定期的に走査し、ダウンロード条件に合うキーが見つかると、ダウンロードタスクを起動します。
- ノード管理は、他のノード情報とそのノードへのコネクションを管理します。
- クエリ管理は、ノード間のキーの受け渡しを担当し、問い合わせるキーをクエリパケットに詰めて別のノードに送り出したり、他のノードから受け取ったクエリパケットからキーを取り出したりします。
- キー管理では、キャッシュフォルダ内のファイルに対応するキーや、拡散や検索によって他のノードから受け取ったキーを管理します。

P2Pソフトの開発手法

5章

これまでに Winny の振る舞いと実装の詳細を見てきましたが、実際に P2P アプリケーションを作成する方にとって興味があるのは、個々の設計ではなく、実際にアプリケーションを作成・運用した経験に基づくノウハウでしょう。そこで本章では、Winny の開発過程ではどのような点を重視したのか、またどのような工夫をしたのか、検討したさまざまな点をお話していくことにします。

5.1 P2P アプリケーションのテスト

通常のアプリケーションプログラムの多くは、開発やテストのすべてを開発者の環境のなかだけで行うことができます。しかし、P2P アプリケーションは、多数のノードが連動して動作する特殊なネットワークアプリケーションであることから、1 台の PC という閉じた環境や、数台の PC による実験では、最終的な動作を確認できません。実際には、ノードが動作するネットワーク環境は多様ですし、ユーザーは予想しにくい操作をします。設計した仕組みでちゃんと動作するかどうかは、多数のノードで実際に動作させてみないとわからないのです。したがって、P2P アプリケーションの開発過程においては、テストをどのように行うかが大きな問題となります。

シミュレーションを利用する

Winny の場合、当初からテスト版を用意し、1 つの PC で複数のノードを起動して連動状況を確かめたり、Winny ネットワークが形成されるようすをシミュレーションしたりしています。この方法で、Winny を実行するとどういう状況になるかを推測しながら、設計と実装を進めました。

とはいえ、こうした環境でできることは、ローカルな環境でのたかだか数十ノードのテストや、条件を極めて限定した数万ノードのシミュレーション程度です。まちまちの環境にある数十万ノードを超える P2P ネットワークの動作を予測することはできません。最終的には、非常に多くのテスト協力者にアプリケーションを動かしてもらい、どんな問題が生じるかを検証することが必要になります。

Winny で実際にどのようなシミュレーションをしたかは、5.2「シミュレーション

の活用と限界」で紹介します。

P2P ネットワークの形成と維持

P2P アプリケーションのネットワークは、参加するユーザーによって形成される非常に動的な系です。私にはこれがまるで生き物のように思えます。P2P アプリケーションの開発とテストは、P2P ネットワークがあってこそ可能であり、ネットワークが大きくならなければ大規模なネットワークで運用可能かどうかを確かめることはできません。このため、いかに多くのユーザーに協力してもらえるかが開発初期の重要な課題となります。またもうひとつ、いったん形成された P2P ネットワークを維持し続けることも、課題になります。

Winny は、2 ちゃんねるという非常に多くの利用者が集まる掲示板を通じて、多くのユーザーの協力を得ることができました。このことが、Winny 成功の鍵となっています。

最初の設計が肝心

長期にわたって P2P ネットワークを維持していくには、やはり開発初期の設計が肝心だということになります。将来起こりうることを想像し、それを見越したシステムの仕様や設計にするのは、難しいことです。そこで事例として、Winny の長期設計で成功した点と失敗した点を 5.6「長期的な設計における成功と失敗」にまとめることにします。

問題点はすぐに改修しなければいけない

ソフトウェアにバグや設計ミスがあることは避けられません。テスト協力者が離れていくのを防ぐためにも、開発の初期段階で見つかった不具合には迅速に対応する必要があります。特に、P2P アプリケーションには生物のような性質があり、時間の経過とともに問題が悪化します。可能であれば問題が発見されたその日のうちに対処することが望まれます。また、すばやい対応だけでなく、先を見越した適切な対策が求められます。

Winny の場合、Winny 1 を開発していた 1 年間のあいだに、問題に対処して数百回のバージョンアップを行いました。平均すると、1 日に 1 回程度の頻度になります。特に初期は問題が多発し、修正によって新たな問題が生じたり、期待どおりの

結果にならずに頻繁に更新を繰り返すこともありました。1日に3回もバージョンアップすることすらしばしばだったのです。

プログラムを気軽に作り変えることはできない

形成と継続が難しいからこそ、できあがったP2Pネットワークは貴重なものです。設計が悪かったからといって、気軽にアプリケーションを作り直したり、それまでと互換性のないバージョンを作ることはできません。すべてのユーザーが新しいプログラムを使ってくれるとは限らず、形成されていたP2Pネットワークが分断されて、結局ユーザーが離れていってしまうかもしれません。何か問題が発生したとしても一からやり直すことは難しいのです。

開発が進まないP2Pアプリケーション

日本の国内でも、常にいくつかのP2Pアプリケーションが開発されています。けれども、そのネットワークが維持されて規模が大きくなることは珍しく、多くは盛り上がることなくひっそりとプロジェクトが消えていきます。

プロジェクトがうまく育たない典型的なシナリオは、使用者が少なくてコミュニティが育たず、バージョンアップも行われない、というものです。使用者が少なく、テストができないので開発が進まず、ソフトウェアの品質が向上しないので、使用者が増えない……という負のループに陥ってしまうわけです。

開発が活性なP2Pアプリケーション

P2Pアプリケーションは、開発初期の状態を乗り越えて安定すれば、自然とコミュニティが育っていき、P2Pネットワークの維持は容易になります。ユーザーが増加して、新たな問題が見つかり、すばやく問題に対応し、さらにユーザーが増加し……、というよい方向へのループに入っていけば、大きなP2Pネットワークに育っていきます。

攻撃に耐える

開発の後期になると、システムへの攻撃にどれだけ耐えられるかが重要になってきます。やはり、設計の段階でどこまで攻撃を予測しているかが大きなポイントとなります。Winnyは多くの利用者の関心を集めた半面で、システムにありとあらゆる攻撃が加えられることになりました。しかし、それも重要なテストのひとつと考

えることができるので、長い目で見ればよかったことなのかもしれません。

バージョンアップと旧ネットワークの廃棄

実装の経験で得た教訓のひとつは、古い P2P ネットワークの破棄は想像以上に難しいということです。少し話は飛びますが、ここではこのことについて触れておきます。

P2P ネットワークを維持していくうえで、ソフトウェアのバージョンアップは大きな課題です。自動アップデート機能を組み込めばよいと考えるかもしれませんが、そのような機能は P2P ネットワークへの攻撃の標的となりやすく、弱点が発見されればせっかく形成された P2P ネットワークが崩壊してしまうでしょう。ワームやウィルスなど、別の問題点を引き起こす可能性も高くなります。

バージョンアップ警告を出す

ここで関連してくるのが、アプリケーションそのもののバージョンアップをどうするのかという問題です。ピュア P2P 方式では、ユーザーにどのバージョンのプログラムを利用するかを強制することができず、開発側はバージョンアップを促すことしかできません。Winny の場合も、バージョンアップ警告のメカニズムを開発途中で導入しています。

Winny は一連の通信に先立って、接続相手のノードと互いに Winny プログラムのバージョン情報を交換します。これにより、自分が用いているプログラムよりも新しいバージョンを利用しているノードがいるかどうかを知ることができます。古いバージョンを用いているユーザーには、バージョン警告を出します。この仕組みは、Winny を頻繁にバージョンアップしているとき、テスト協力者にプログラムの更新を促すのに効果がありました。

通信プロトコルをまったく互換性のないものに取り替える際には、すくなくともいま述べたようなバージョン警告だけは出さないといけません。さもないと、ユーザーがバージョンアップに気づかず、P2P ネットワークが旧バージョンと新バージョンのものとに分断してしまう可能性があります。

プログラムを新旧のバージョンに対応させる

プロトコルを互換性のないものに変える場合、それまでのユーザーを引き継ぐた

めにはシステムの互換性を保つことが重要です。プログラムを２つのプロトコルの両方に対応させなければならないので面倒ですが、できれば２つ前のバージョンぐらいまで対応すべきでしょう。

Winny の場合、バージョン警告の機能を悪用して、相手に偽りのバージョン警告を出すような悪戯ノードが出現しました。設計では、こうしたことにも配慮が必要でしょう。Winny の場合、自分より新しいバージョンが１つ見つかったときすぐにバージョン警告を出すのではなく、周辺に自分よりも新しいバージョンを使用しているノードがある程度増えてきた時点で警告を出しています。

5.2 シミュレーションの活用と限界

P2P アプリケーションの開発においては、テスト協力者の存在が重要だと述べました。しかし、プログラムを公開する前に、閉じた環境での最低限のテストは欠かせません。

多くのノードで構成されるネットワークアプリケーションの場合、手持ちの環境だけでテストできる状況は限られます。だからといって、実際に多数のノードで動作させてみないとわからないのでは、テスト協力者に見放されます。やってみないとわからない部分でも、なんらかの方法で検証しておく必要があります。

また、前節では長期的設計が重要であり、将来問題になりそうな箇所は事前に検討しておくべきだということを述べました。しかし、どのような設計がよいかは最初からわかるわけではありません。そこでここでは、事前検証の手段として、シミュレーションを紹介します。

これは、検討事項の部分だけを抜き出し、そのモデルを簡略化した仮想的なシミュレータプログラムを作成するものです。多数のノードで構成される P2P ネットワーク上の動作も、仮想的に再現することはある程度可能です。この方法により、いろいろな検討事項をモデル化して検証できます。

設計時のシミュレーション

ノード間を検索リンクでつないで形成する Winny ネットワークは、回線速度による階層化とクラスタリングという独特の概念によって構造化されています。しか

し、設計段階ではこれできちんと動作するかどうかはわかりません。また、検索リンクをどのように接続すれば最適な Winny ネットワークを形成できるか、というパラメータも、机上の推測だけでは決めかねます。

そのため、Winny 1 のβ 1 を公開する前に、ノード間のリンク機構とキー検索機構に関連するシミュレーションプログラムを作成し、基本モデルの設計とパラメータの決定に使用しました。実は、Winny を作ると決めて一番最初に書き始めたコードは、Winny 本体のプログラムではなく、このシミュレータでした。

一方、Winny 本体のプログラムはというと、まず基礎設計とは関係のないところから書き始めています。あとで必ず必要となるソケットを用いた通信部だとか、ハッシュ計算や暗号部などから始め、これらの基礎的なライブラリと平行する形でシミュレータプログラムを作成しました。このシミュレータで動作を検討しながら Winny の基本設計を決定し、BBS 上でみんなと検討しつつ、Winny 本体のプログラムを作るという形で Winny 1 の開発は進んでいったのです。

βテストの開始から 2 年以上たったいまに至るまで、Winny ネットワークが停止することなく稼動しているのは、この初期段階の検討の成果ともいえます。

図 5-1 は、ノード間の検索リンク接続とキーの拡散・検索に関するシミュレーションの結果です。上方にあるのが高速回線のノードで、下方にあるのが低速回線のノード、線は検索リンクを示しています。付録 B「シミュレーション」に同じ画面をカラーで掲載してありますので、そちらもご覧ください。

各ノードは上流と下流に何本の検索リンクを張ったらよいのか、検索ヒット率はどの程度見込めるか、何ホップ先まで検索すればよいか、といったことは、このシミュレーションでテストしながら設計しています。

なお、このシミュレーションでは、開発当時の通信インフラの状況に合わせてノードの分布を設定しています。当時はまだアナログダイアルアップ回線や ISDN 回線の利用も多く、ADSL がやっと普及してきたころです。Winny のユーザーは、光回線の利用者が 1 割以下で、多くは高速タイプもしくは低速タイプの ADSL やケーブルテレビ回線が占め、残りは ISDN 以下の回線という構成です。図 5-1 はそれを反映し、ノードは大きく分けて 4 層に分布しています。

シミュレーションの限界

このシミュレータでは、各ノードがファイルを平均 100 個ずつ公開していると仮

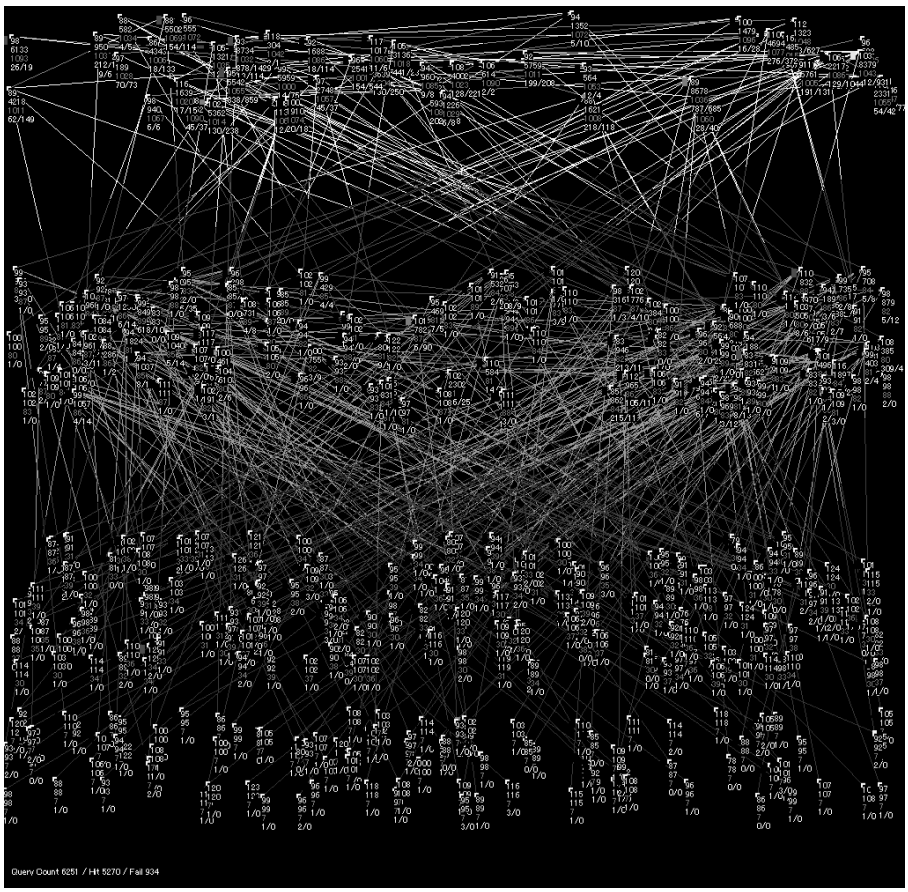


図 5-1　検索リンクの接続とキーの拡散・検索のシミュレーション

定し、検索キーがどのように分布し、どの程度クエリで検索ができるのかを検証できます。その部分に限ったシミュレータなので、1万ノード程度の動作は確認できます。その結果わかったことは、ノード数が数千程度までならクエリの検索ヒット率は90%程度見込め、それを超えると別のメカニズムが必要になるということでした。設計の初期にクラスタリングの必要性を認識していたのは、このシミュレーションの結果によります。

しかし、シミュレータでの予測にも限界があります。実際にβテストを開始してすぐに直面したのは、検索キーの数が多すぎてノードが処理しきれないこと、またそもそもファイルがダウンロードできないということでした。ファイルがダウンロー

ドできないのは、ポートを開けていないユーザーが多いためだとあとで判明するのですが、検索キーの流量はこちらが考えていたよりもはるかに多く、上流の高速回線のノードが負荷に耐えられないことが初期の問題のひとつとなりました。

ユーザーの振る舞いは予測しにくい

各ノードにかかる通信量や処理コストに関しては、ある程度シミュレータで予測していましたが、最終的にどの程度の負荷がかかるかは、ファイルの公開数などのユーザーの振る舞いに依存するのでわかりません。

このような部分は、実際に運用して確かめるほかありません。シミュレータによる方法も、慣れれば傾向を掴めるようになってくるのですが、ネットワーク規模が大きくなるにつれて予測不能な要素も増えてきますので、実際にテストしてみたほうが早いということになりがちです。実際、Winny の動作パラメータの多くは、試行錯誤の結果決まったものです。

なぜ Winny ネットワークの状況を把握できないのか

Winny は、ネットワーク全体の状況を把握するためのモニター機構を備えていません。これは、そもそもモニター機構の実現が難しいということと、モニター機構がクラックの対象になりうると考えたためです。P2P システムなのに特定のノードに集中接続するようなメカニズムを組み込めば、特定ノードに DDoS 攻撃[*1]をすることになってしまいます。また、全体に影響をおよぼすようなブロードキャスト系の機構も、同様にクラックの可能性があるため備えていません。

このようなことから、Winny ネットワーク全体がどういう状況なのかは観察したことがなく、開発した私にもわからないのが実状です。開発者なんだから、何か特別なツールで系の状況を解析できるんじゃないかと期待する方がいらっしゃるかもしれませんが、このような事情でネットワーク全体の状況に関して考察することはできませんでした。

なお、Winny のシミュレータとしては、上記で挙げたもののほか、クラスタの形成や、キャッシュの分布状況、それに基づいたダウンロード成功率のシミュレーションなども考えられます。しかし、これらの題材はユーザーの振る舞いに依存する要素が多く、基本パラメータの予測が困難だったので、作成しませんでした。シミュレーションが役に立つのは、シミュレータに与える条件が特定できる部分だけということになります。

*1 DDoS 攻撃（Distributed Denial of Service attacks）：分散している複数のノードが特定のノードに集中的に接続して負荷をかける攻撃。

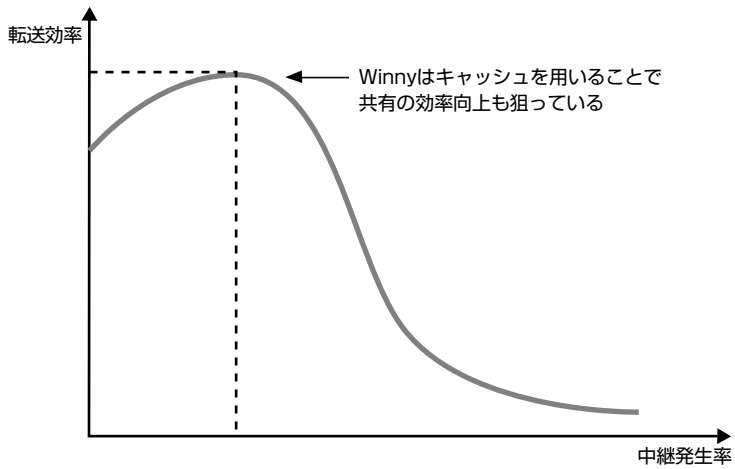
5.3 匿名性と転送効率の両立

匿名性と効率には、トレードオフの関係があります。これは、匿名性を重視しているFreenetの共有効率の悪さから推測したことです。そのようななかでWinnyは、各ノードにプロクシーサーバの機能を組み入れることによって必要最低限の匿名性を実現しつつも、ファイルのキャッシングにより転送効率の向上を狙ったファイル共有システムだといえます。

Winnyは、第一情報発信者のノードから最終的な利用者のノードまでの中間に中継ノードを生じさせることによって、匿名性と効率の向上を図っています。匿名性は、経由する中継ノードの数が多いほど高くなるので、匿名性を重視するのであれば中継ノードの数を増やすほうがよいでしょう。

しかし、あるファイルをひとりの利用者に転送する場合だけに注目すると、ファイルが最終的な利用者のノードに届くまでの間に中継の数だけファイルがリレー式に転送されることになり、ネットワーク内に余分なトラフィックが生じることになります。むやみに中継ノードを発生させては転送効率が悪化してしまいます。しかし、中継ノードにキャッシュされたファイルは別の利用者からの要求にも利用できるので、適度な中継ノードの発生であれば、むしろ全体的に見た転送効率はよくなります。これは、HTTPプロクシーサーバのキャッシングのおかげでWebページ参照の効率がよくなるのと同じです。

では、適度な中継発生率とはどれくらいなのでしょうか？　実際には、中継の発生率と効率性の関係は図5-2のようになるはずで、これは最適な中継発生率を探って頻繁にバージョンアップ繰り返した結果、経験的に得られたものです。この結果を基にして、Winnyは効率が最大となるように中継の発生率を調整しています。



横軸は中継発生率、縦軸が転送効率。中継をまったくしない場合よりも少し中継した場合のほうが転送効率はよい。しかし、あまり中継が発生すると転送効率は指数関数的に低下する。

図 5-2　匿名性（中継発生率）と転送効率

5.4 Winny と暗号技術

Winny は、クラック対策とプライバシー保護のために、次に挙げるような部分を暗号化しています。

- **初期ノード情報**
- **ノード間通信**
- **キャッシュファイル**
- **プログラム本体**

これらの暗号化には、共通鍵方式を使用しています*2。

また Winny 2 では、BBS に書き込む人を認証するために RSA 暗号を使用しており、他人を騙った書き込みを検出できるようにしていますが、RSA 暗号で投稿内容を暗号化しているわけではありません。

*2　プログラム本体以外は RC4 を使用しています。

ノード間通信の暗号化

Winny は、他のノードに宛てた通信内容を暗号化して送り出します。一方相手ノードは、Winny が内蔵している機能を使って受け取った通信を復号します。つまり、ノード間でやり取りされる通信内容は暗号化されているので、通信経路上の第三者は覗き見ることができません。

しかし Winny の場合、通信相手は不特定であり、Winny プロトコルを使う誰もが通信相手になれます。たとえば、Winny プロトコルを真似て Winny ノードに接続するようなプログラムの作成ができれば、接続相手のキャッシュファイルを観察することも可能です。このため、たとえどんなに強い暗号方式で通信内容を暗号化したとしても、通信相手は解読可能であり、通信内容は隠しようがありません。

また Winny のキャッシュファイルは、そもそも公開されているデータと考えるべきです。

なぜ通信を暗号化するのか

このようなわけで、本来は通信内容を暗号化する意味はあまりないのですが、どのようなファイルをやり取りしているかは一種のプライバシー情報なので、他のユーザーが通信パケットをダンプしたくらいでは内容がわからないようにしておいたほうがよいでしょう。また、Winny ネットワークのクラックを防ぐために、Winny のプロトコルやプログラムソースは非公開にしていたので、通信を暗号化することにはプロトコルの秘匿を保つという意味もあります。

Winny は、このようなプライバシーの保護とシステムへのクラック対策を目的として、通信内容を簡単に見ることができない程度の簡単な暗号化を施しています。具体的には、暗号方式に RC4 を使用し、暗号キーはランダムに生成します。DH 鍵交換などは行っていません。Winny はたいへん頻繁にコネクションを開くので、処理速度を重視し、コネクションのオープンに時間のかからない実装となっているのです[*3]。

しかしこのような対策をとっても、Winny プログラム本体にはプロトコルを解釈する部分が含まれているので、プログラム本体を解析すればプロトコルの解析は不可能ではありません。

*3 RC4 は、ストリーム暗号としては比較的高速な部類に入ります。

暗号を破られても致命的な問題が生じないことが重要

そうなると、暗号を工夫するより、暗号を破られても致命的な問題を起こさないような設計が重要になってきます。実際、Freenet はシステムがオープンであっても匿名性を実現しています。つまり、プロトコルが明らかになっても、匿名性を保つことは可能だということです。これは Winny でも同様です。

Winny の開発が停止したあと、Winny 自体の解析も進んでプロトコルが判明しつつあります。最近は、Winny の通信をブロックするファイアウォールも開発されているようです。

公開鍵方式の暗号を使用すれば通信内容の解析は難しくなったでしょうが、それでも Winny プロトコルを模倣してネゴシエーションすれば、たとえ通信内容が暗号化されていたとしても接続相手が Winny かどうかを判断可能です。

初期ノード情報の暗号化

これまでにも述べたように、初期ノード情報として登録する文字列は暗号化されています。初期ノード情報は、Winny を終了させたときに自動的に Noderef.txt ファイルに保存され、再起動時にはこのファイルからノード情報が再現されます。そこで、Noderef.txt ファイルの内容も同様の形式で暗号化されています。


```
暗号化されている初期ノード情報の例
@c6cce7fa71d00bcb06de2e2709d5391a34
@a6c4095094ddb57a3c5941e26caa5eaa746d15
@fe36be263afa59ded5
```

この暗号化に使用するキーは固定的なもので、ユーザーが変えることはできません。そのため、この暗号を解読することはそれほど難しくはありません。しかし、そもそもノード情報というのは IP アドレスとポート番号なので、Winny を起動して他のノードに接続したときに調べることは可能で、どんなにがんばって暗号化したとしても隠しようがありません。

それでもこの部分を暗号化しているのは、BBS 上や Web 上に公開されても目で確認したりクロールプログラム[*4]が簡単に収集できないようにしたためで、心理的な効果とプライバシー保護がおもな理由です。

*4 公開されている Web などの情報から自動的に IP アドレスなどを収集するプログラム。

なお、ノード情報は暗号化されると先頭に“@”マークが付き、後ろに16進数表記の文字列が続きます。この16進表記文字列の先頭1バイトはチェックサムです。

キャッシュファイルの暗号化

情報の供給者と利用者のプライバシーを考えたとき、キャッシュファイルは任意に公開された情報なので、本来はキャッシュファイルを暗号化する必要はありません。しかし、キャッシュファイルの内容はシステム側だけが把握していればよい情報ですし、キャッシュファイルの改ざんによるシステム妨害が容易になるのは避けたいと考えました。

そこで、キャッシュファイルの各所に誤り検出用の情報を設け、可能なかぎり改ざんを検出できるようにしたうえ、キャッシュファイルを暗号化し、改ざんに対する耐性を強化しています。

なおキャッシュファイルの設計として、ファイル全体のキャッシュファイルが揃わないと暗号が解けないような方法や、1つのファイルを1つのキャッシュファイルに対応させるのではなく、ノード内のキャッシュ全体を1つのファイルにすることでキャッシュ管理をより効率化する方式など、いろいろな方法が考えられます。特にこのキャッシュシステムの設計が、第三世代P2Pファイル共有ソフトの肝になるはずです。

プログラム本体の暗号化

Winnyは、クラック対策のためにプログラム本体のWinny.exeファイルも暗号化しています。これは、.exeファイルをWindowsの通常のEXE形式にせず、.exeファイル内に暗号化したプログラム本体とそれを展開するプログラムを格納しておき、プログラム起動時にプログラム本体をメモリ上に展開してから実行する仕組みになっています。

Winnyは、このような.exeファイルを作成するexeパッカーというツールを使用して、オリジナルプログラムを暗号化しています。

プログラムの暗号化は、プログラムの解析を困難にすることが目的なので、.exeファイルのサイズを圧縮するだけのプログラムでも十分用は足ります。exeパッカーも、もともとは.exeファイルを小さくすることが目的のプログラムです。

Winny 1 と Winny 2 のβ版初期では、ソースコードが公開されている UPX という exe パッカーを多少変更して使用しています。Winny 2 のβ版後半では何度か変えていますが、ほとんどは tELock を用いています。

5.5 システム妨害に対抗する

開発の前半では、そもそもシステム本来の機能が不完全な動作だったので、バグ修正や調整に時間をかけています。後半になってテスト協力者が増加してくると、システムへの攻撃も増え、プログラムの変更はその対策が中心になりました。

Winny ネットワークは多くの人が集まる掲示板を背景に開発を行っていたため、ありとあらゆる攻撃に晒されることになりました。しかしそれに対処することによって、検証とともに耐障害性を向上させることができました。これは、独特な開発スタイルだと思います。

Winny ネットワークへの攻撃

P2P アプリケーションで想定すべき攻撃として、Winny ネットワークに対して行われた攻撃の代表的なものをいくつか挙げます。

ごみファイルを大量にばら撒く

特に目立ったのは、ごみファイルに人気のありそうな名前[*5]を付けて大量に流すものです。こうしたファイルは、自動ダウンロード機能との相乗効果でネットワーク全体に自動的に広まり、共有ファイルシステムを混乱させます。この対策として、現在では 4 章「実装」で述べた無視フィルタ機能や各種の警告機能を導入し、ある程度対処できるようになっていますが、完璧な対策とはいえません。

また、この悪戯は、ネットワークとノードに負荷をかけるという側面もあります。しかし Winny は、設計上ファイルの検索率が悪いという面があり、その問題点をクラスタリングで対処しています。したがって、このような悪戯をするノードがクラスタから外れるようにすれば、ユーザーへの悪影響を低減できます。実際には、無視ノード警告という仕組みを作り、無視条件にヒットするような検索キーを大量に送ってくるノードとの接続を切って、クラスタの外に追いやるようにしました。

*5 クラスタリングに用いられそうなキーワードを複数並べたものがよく使われました。

ファイルを捏造する

配布するファイルの中身とファイル名の内容は一致している必要がないので、他の人が広めたファイルに別の名前を付けてファイル共有システムを混乱させるという悪戯もありえます。

これに対しては、検索キーの名前の上書きルールを工夫して、名前が上書きされにくいように工夫しています。またそれとは別に、Winny には捏造警告という専用のメカニズムがあり、無視条件の指定によって特定のハッシュ値を持つ検索キーに捏造警告を付けることが可能です。捏造警告を指定された検索キーは、他のノードでは赤く表示されるので、注意を促すことができます。

ただし、誰かが問題ファイルと判断しても、別のユーザーには有用かもしれず、また問題のないファイルに捏造警告を付けるという新たな悪戯が生じるので、捏造警告の信憑性は絶対的なものではありません。

そこで、検索キーが拡散する途中で捏造警告のマークを入れるだけの機構とし、結果的にネットワーク参加者の多数決を採るようなメカニズムにしています[*6]。検索キーの流通時に捏造警告をチェックして、マークする、マークしない、どちらでもない、のいずれかを指定し、大多数がこのファイルを正しいものと扱ってダウンロードすれば、検索キーには捏造警告が出ませんし、多くが捏造警告を指定していればその検索キーには捏造警告が付きます。

*6 キーの拡散時に、無視条件で捏造警告フィルタにかかれば警告を付け、ハッシュ値でダウンロード条件が指定されているときには捏造警告をはずします。

ウィルス問題

最近では、Winny ネットワークを介してウィルスがばら撒かれるという問題が生じています。これも想定外のことですし、開発が停止してから出てきた問題なので対策がなされないままです。

P2P アプリケーションを利用したウィルスは、他の P2P アプリケーションでもよく見かけます。そこでウィルスの混入を避けるために、開発途中で要望の多かった自動アップデート機構も採り入れず、このような問題に対処していたつもりでした。しかし、.exe ファイルをわざと実行させる工夫がしてあったり、アーカイブを展開するだけでスタートアップフォルダに実行ファイルを展開させるなど、予期していなかった手法を使ったウィルスも存在します。バージョンアップすれば対処できますが、現在は私がバージョンアップすることはできません。

転送リンク接続数コントロール部の改変

ファイルをダウンロードできるのは、誰かが同じだけアップロードしているからであり、Winny ネットワーク全体のダウンロード総量は常にアップロード総量と等しくなります。つまり、ファイル共有の効率をよくするためには、Winny ネットワーク内のダウンロード量とアップロード量のバランスを保つ必要があります。

そこで Winny は、各ノードのダウンロード量を、アップロード量――すなわち回線速度に応じて、接続リンク数を自動的に調整し、ネットワーク全体のバランスをとっています。しかし、より多くのダウンロードを可能にしようとして、Winny プログラム内の転送リンク接続数のコントロール部分を改変するユーザーが現れました。このような改造は、Winny ネットワークのダウンロード/アップロードのバランスをくずし、Winny ネットワーク全体の効率を悪くします。このため、このあと説明するような方法でプログラムの改変に対処しています。

Winny プログラムへの攻撃

これまでにも述べたように、Winny は Winny ネットワーク内のダウンロード量とアップロード量のバランスを保つために、プログラムのソースコードやプロトコルを非公開にしています。このため、使用している暗号、プロトコル、プログラムをリバースエンジニアリングなどの解析行為から保護するために、Winny プログラム自体へのクラック対策をいくつかとっています。

プログラムの書き換えを検出する

まず Winny は、自身が書き換えられていないかどうか Winny.exe ファイル本体のハッシュ値を使って確認し、プログラムが書き換わっていたらそれに応じて動作を変えます。またメモリ上に展開されたプログラムが別のプロセスによって書き換えられる可能性がありますので、外部からのプログラムの書き換えも検出しています。

このようなクラック対策は、シェアウェアや市販のパッケージなどではよく見かけるものです。より厳密を期して、exe パッカーを自作することも行われるようですが、それでもプログラムの改変を完全に防ぐことはできないでしょう。

結局、Winny のクラック対策はあくまでシステムの解析を防ぐための時間稼ぎだといえます。

なぜ Winny はオープンシステムでなかったのか

これまでの経験から、Winny は転送リンク数のコントロールコード部がよく狙われます。すでに説明したように、ここは Winny ネットワーク全体のファイル共有効率をコントロールする部分です。Winny をクラックして、各ノードがアップロードを抑えてダウンロードを増やそうとすれば、囚人のジレンマ状態に陥り、Winny ネットワークの系全体の効率は低下します[*7]。系全体の共有効率を維持するためには、この部分を保護しなくてはなりません。

Winny がオープンシステムではないのは、この点がおもな理由です。

*7 みんなが「ただ乗り」(フリーライド) をすると、P2P ネットワークの効率は落ちます。しかし Winny では、キャッシュを持っていないノードや性能の低い ADSL ノードのフリーライドも受け入れる設計になっています。

5.6 長期的な設計における成功と失敗

P2P アプリケーションが動き出し、ある程度のテスト協力者が集まって、さてできあがったネットワーク上で発生する問題はといえば、実は設計上どうしようもないことが多いものです。P2P アプリケーション特有の事情として、プログラムに大規模な変更を加えると、それまで形成してきたネットワークとの互換性がなくなり、結局一からやり直しになるということがあります。難しい問題ですが、P2P ネットワークを大きく成長させられるかどうかは、先を見越した設計かどうかによって決まるといえます。

初期段階で問題が生じたときも、迅速な対応をしなければならないので時間は限られています。そのなかで、将来を予想する時間を多く確保し、重要な箇所に時間を割き、どうでもいい部分は手を抜くといったバランス感覚が必要です。重要な部分で手を抜くと、ユーザーが増えて系の規模が大きくなったとき、あとで必ずその点が問題になります。

長期的な設計という点では、Winny でもうまくいった部分とそうでない部分があります。その具体的な点を示すことは、あとに続く人の参考になるでしょうし、Winny の設計がなぜそうなっているのかを理解する助けとなりますから、以下で個別に紹介します。

長期的設計――クラスタリングの場合

クラスタリングは、P2P ネットワークの規模が大きくなったときに必要だということで、設計当初から導入を計画していましたが、どのような方法をとるかは未定でした。このため初期のβ版公開時には、クラスタリングの概念を導入しやすいように基本部分だけを組み込んでありました。おそらくは、検索リンクの接続をコントロールすることになるだろうと予測し、はじめから検索リンクを動的に接続・切断できるようにしておいたのです。この仕組みは、長期的な設計という点では比較的うまくいった部分です。

当初、系へ参加するノード数が 1 万くらいまでなら、クラスタリングの必要はないだろうと想定していました。これは、ピュア P2P 方式の Gnutella で何が問題になったかを知っていたからです。参加者が増えていけば、検索がトラフィックやその他のさまざまな部分で問題になることはシミュレーションの結果から予想できたため、ノード数が 1 万を超えるころにクラスタリングを導入することにしたのです。

クラスタリングの変遷

現在の Winny は、ユーザーが申告したキーワードの相関度を基にしてクラスタリングを行っています。しかしこれは、β版の公開後、比較的あとになって編み出された方式で、それまでに考えていたさまざまなアイデアのひとつにすぎません。長期的設計の例として、この方式に至った経緯を説明しておきましょう。

一次元トーラスを用いる方法

私が最初に提案したクラスタリング方法は、クラスタを一次元トーラス状に形成し、各ノードに設定してもらった実数パラメータを使ってクラスタと対応づける、というものです（図 5-3）。

一次元トーラスは、たとえば値を 0 から 1 の間というように設定すると、1 の隣が 0 に連続している切れ目のない環状構造となります。また、24 時が 0 時に連続している時刻も、一次元トーラスと考えることができます。

図 5-3 はこの方法を使ったクラスタの概念図です。各ノードのユーザーは、パラメータとして一次元トーラスのどのあたりに属するかを、たとえば“0.3”“0.34”の

ように申告します。クラスタそれぞれのノード数は数千を上限とし、回線速度を反映した階層構造をなします。隣のクラスタとのやり取りはおもに上層のノードが行い、検索リンクとは別のリンクを使ってクラスタ間のキーをやり取りするというような設計でした。

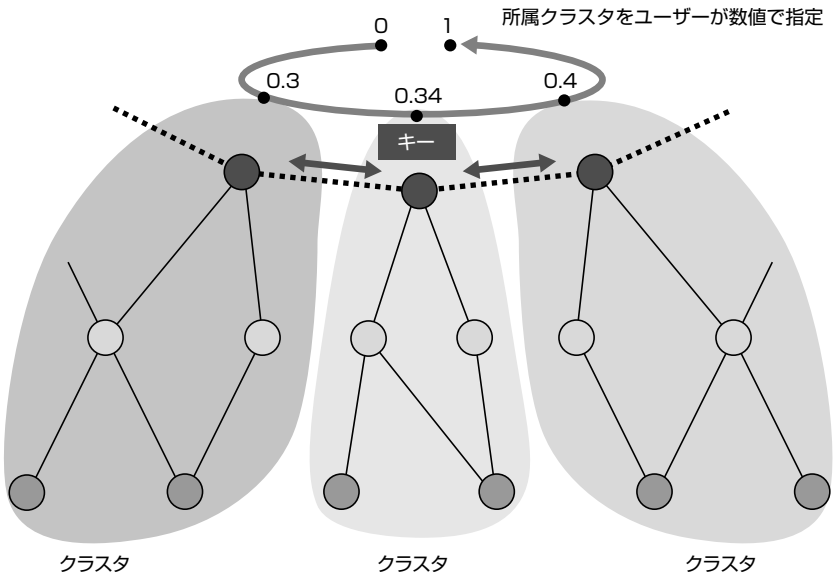


図 5-3　一次元トーラスを用いたクラスタリング

ピュア P2P 方式では、検索依頼をネットワーク内の全ノードに届けたいところです。しかし、全ノードに対してブロードキャストすることは、ネットワーク帯域にかかる負荷を考えると現実的ではありません。そこで一般的には、検索依頼の配送範囲を限定しています。前述のようなクラスタリングは、遠いノードへの検索依頼のルーティングを解決するために考えたものです。

いまなら分散ハッシュテーブル（DHT：Distributed Hash Table）を使うことも考えられますし、現在の Winny は P2P ネットワーク上で遠方にあるノードへの検索は諦めてしまっているので、もうすこし効率のよい方法もあるでしょう。Winny に関していえば、検索に関連する部分は非常に弱い設計となっています。一次元トーラスを用いた上記の方法は、とりあえずどちらの方向に検索をかければよいかを判断するために、各ノードに横方向の座標の概念を入れてやろうと考えたのです。

パラメータは一次元でなくてもよい

*8 多次元のハイパーキューブ構造など、私も多少は分散計算にかかわった経験があるので、聞いたことがあったのです。

さて、一次元トーラスを使う方法に対して、Winny 開発スレッドではいろいろな意見が出ています。この一次元の数字を、色の色相に当てはめたらいいんじゃないかという、面白い意見もありました。しかし、特に興味を惹かれたのは、そもそも一次元である必要もないし、多次元でもいいのではないかというものでした。

この辺のことは、分散計算ではお約束の話でしょう[*8]。たしかに、一次元空間にこだわる必要はなく、多次元空間を用いてクラスタリングを行えば、より効率はよくなるように思えました。となると、問題はどのように多次元空間の位置を各ユーザーに設定してもらうかです。そしてそのころ、Winny にはキーワードを用いた自動ダウンロード機能が導入されていたので、このキーワードを各ノードの位置決定に使用するというアイデアを思いつきました。

検索キーワードをクラスタリングに使用する

そのために導入されたのが、3 つのキーワードを使ったクラスタリングです。これは、キーワードそれぞれを組み合わせ、一致する部分文字列を調べて、一致した文字数の合計が大きいノード間をより近いクラスタと判断する方式です。

この方式がよいかどうかは、今後の状況を見ていかないとわかりません。クラスタリングの導入は、Winny の検索方式があまりよくないことから導入されたごまかしの手法ともいえます。もっとよい方式があることは間違いありません。しかし、ネットワーク規模の拡大への対処という意味では、このクラスタリング方式は比較的うまくいき、結果的に問題はほぼ解決されています。

Winny がピュア P2P 方式を採っている以上、ノードが多くなれば、ファイルの検索率や検索時間、検索に必要な通信負荷のそれぞれが、互いにトレードオフの関係となります。Winny の場合、ノード数がどんなに増えてもクラスタ内のノードの密度はそのままで、クラスタだけが細分化して増えていきます。したがって、ネットワーク全体の検索率は、規模の拡大とともに悪化していきますが、検索時間や検索効率、検索によって生じる通信量は、各クラスタ内では一定であり、どんなに規模が大きくなっても全体が破綻することはありません。

長期的設計――キャッシュシステムの場合

クラスタリングは、開発当初から導入が考慮され、きれいに問題を解決できた部分です。次に、キャッシュシステムの設計――うまくいったとも、失敗したともいえる部分について話をしましょう。

Winny のキャッシュファイル形式は、V1 から V6 まで、都合 5 回の変更を重ねています。キャッシュファイルは、匿名性と効率を実現する要となる部分です。キャッシュファイルの互換性がなくなることは、それまで Winny ネットワーク内に蓄えられてきたキャッシュファイルが利用できなくなり、Winny で共有している情報を破棄しなければならないことを意味します。Winny ネットワークのノード数を維持するためには、なんとしても互換性は保たなければなりません。

このことは開発当初から認識し、バージョンアップを考慮したキャッシュファイル形式にしてあったので、ネットワーク上にできているキャッシュファイルそのものの破棄は、ある程度は抑えることができました。

しかし実際には、キャッシュファイル形式のバージョンアップは想定していたものの、キャッシュファイルに関連する他の部分も、開発の途中で発生したさまざまな問題のために改修することになります。キャッシュシステムまわりは、想像した以上に問題が生じた部分でした。

これまでにキャッシュすべてを破棄したのは、キャッシュファイル形式が V1 から V2 へ移行した開発初期の一度だけです。その後、ネットワーク上のキャッシュファイルの量がかなり増えた段階で V4 から V5 へ移行し、このときにもキャッシュファイルの互換性を失いました。しかし、キャッシュファイルを完全に破棄することはできなかったので、プログラムを 2 つの形式の両方に対応させてしのいでいます。その結果、ネットワーク内には同じファイルの 2 つの形式が混在することになりました。

キャッシュシステムの変遷

ここでは、Winny ネットワークの拡大で何が起き、キャッシュシステムがどう変遷したのかを簡単に記しておきます。キャッシュファイルのヘッダの変遷を**表 5-1** にまとめます。

キャッシュシステムの初期設計でまずしておいたことは、キャッシュファイルのバージョン情報をキャッシュファイルヘッダに含めることと、キャッシュファイルヘッダに十分な未使用（予約）領域を確保することでした。

表 5-1　キャッシュファイルのヘッダの変遷

キャッシュファイル形式のバージョン	変更の内容
V1	最初のキャッシュファイル形式。ヘッダサイズは 64 バイト
V2	V1 キャッシュの問題点となっていたファイル名情報を、ヘッダに含めるように変更した。そのために必要なサイズ 256 バイトが増えた。ヘッダサイズは 320 バイトになった
V3	検索キーに新たに加えられたトリップ情報をキャッシュファイルヘッダに収容した。このための領域として、それまでキャッシュファイルの暗号キーに使っていた部分を割り当てている
V4	BBS 機構を Winny 1 に導入したため、これに関連する情報をヘッダに追加した。フラグを追加しただけなので、ヘッダの未使用領域を用いて互換性を保つことができた
V5	V4 キャッシュのヘッダと内容に違いはないが、解釈に変更が生じた。Winny はキャッシュファイルのバージョン情報を見て挙動が変わる
V6	Winny 2 の BBS 用に作られたキャッシュファイル形式。Winny 2 は、ファイル共有には V4 と V5 のキャッシュファイル形式を使用している

内部データにバージョン番号を組み込む

Winny のように変更が頻繁なプログラムにとって、バージョン情報はとても重要な情報です。問題が発生したときの手がかりとして、プログラム本体のバージョン情報をはっきりさせておくことは当たり前ですが、プログラムが扱うデータにもバージョン情報を持たせれば、システム全体の寿命を長持ちさせることができます。

Winny ではプログラム本体のほか、通信プロトコル、キャッシュファイル、メモリ上の検索キー、BBS ファイルのそれぞれがバージョン情報を持っています。バージョン番号があれば、扱っているデータがどんな形式なのかを簡単に判定できるので、プログラムを複数の形式に対応させることができます。

バージョン情報がなくても、プログラムがデータ形式を解析して判断できる部分

もあります。たとえば、ダウンロードフィルタファイルや無視フィルタファイルなどには、ユーザーの利便性を考慮してバージョン情報を付けていません。しかし、バージョン情報がないとあとで問題になることがわかっている部分には、はじめから含めるようにしてあります。

ファイル名情報を追加する

当初はかなり余裕（未使用領域）をもって作ったつもりだったキャッシュファイル形式でしたが、V6 キャッシュのヘッダを設計する時点でほとんど空きがなくなり、ヘッダのほかの領域を流用する事態となりました。

キャッシュファイルのヘッダは、V1 の 64 バイトから V2 以降の 320 バイトへと、途中で大きくなっています。ヘッダが増大した原因は、ファイル名情報としてキャッシュファイルのファイル名を使うのをやめ、ヘッダ内にファイル名情報を持つようにしたためです。これは想定外のことで、V2 キャッシュに移行するときには、それまでできていたキャッシュをすべて破棄しなければなりませんでした。

そもそも、ヘッダ長を固定にしていたことも問題で、ヘッダにヘッダ長の情報を用意しておけばよかったのでしょう。どちらにせよ、ヘッダが大きくなったからといって、それまでのキャッシュファイルすべてを新しいバージョンに移行させることはできないので、最終的にはヘッダだけ別ファイルにするなどの工夫がいるものと思われます。

ハッシュ値に問題が見つかる

Winny のネットワーク部分が完成に近づいたころになって、V4 キャッシュには大きな問題が見つかりました。ハッシュ値の計算方法に問題点が見つかり、結局ハッシュ値の計算方法そのものを変えなければならなくなったのです。本来であれば、ネットワーク上にできていたキャッシュをすべて破棄しなければならないところです。しかし、V4 キャッシュはすでにかなり広まっていたため破棄することはできず、結局新しい V5 キャッシュと混在できるようにして、Winny 2 に至るまで引き継がれることになりました。

Winny 2 の BBS に対応する

さて、V5 キャッシュまでは Winny 1 用に開発されたものですが、Winny 2 では BBS 機能のために V6 キャッシュを新たに定義しています。しかし、Winny 2 のファ

イル共有には、Winny 1 で使用してきた V4 と V5 の両方のキャッシュファイルを使っています。当時の計画では、あとで V5 キャッシュを破棄してファイル共有にも V6 キャッシュを使用する予定でした。

これは、認証メカニズムを BBS 側に導入し、自ノード以外に拡散した BBS スレッドファイルを特定のノードで一元管理できるようにする予定があり、この仕組みをファイル共有でも利用できる可能性があったからです。これが実現できれば、一度公開したファイルをあとから消す機構なども可能になったはずですが、Winny 2 の開発はβ段階で中断しているため、そのままになっています。

ここにきて、ヘッダにはほとんど未使用領域がなくなり、V6 キャッシュではファイル名のエントリを一部分流用[*9]しています。キャッシュファイル形式にはそろそろ限界が見えますが、ヘッダの基本的な形式は最後まで保たれ、最低限の互換性ながらも長期の使用に耐えたといえます。

*9 BBS 専用のトリップ認証情報。

このように、キャッシュファイルにバージョン情報を持たせたことはあとで役立ちましたが、設計のミスをカバーしたというのが本当のところです。

長期的設計——ハッシュ値の場合

次に、Winny の長期的設計のなかでも失敗したと思われる例を挙げ、先をうまく見越せないとどんなことが起きるかをお見せしようと思います。失敗というのは、Winny がファイル ID として使用するハッシュ値の設計です。

ファイル ID

Winny では当初から、ファイルを識別するために名前ではなくなんらかの ID を使用し、それをキーの中に格納することにしていました。基本的には、ファイルの内容から計算できるハッシュ値を利用しており、これは現在も変わっていません。問題は、このファイル ID の具体的な内容です。

ファイル ID の内容は、ファイル本体の暗号化の方法にともなってこれまでにも二転三転しています。ファイル本体の暗号化をどうするかは、ファイルのアクセス制御ともかかわる部分であり、その方針を探っている状態であったからです。

キャッシュファイルの暗号化

設計当初の Winny 1 βでは、キャッシュファイルの暗号化に使う暗号キーは、ファイルを配布するユーザーが任意に設定できるようになっていました。ファイルを利用するユーザーは、検索の際に検索キーワードとあわせて配布者が指定した暗号キーを指定しないかぎり、検索キーを発見することも、ダウンロードしてファイルを復号することもできません。つまり、あらかじめ暗号キーを渡したユーザーのみファイルにアクセスできるようになっていました。

しかし開発の途中で、暗号化によるアクセス制御はファイルのフォーマットの問題であり、ファイル共有ソフトというネットワークインフラ側の問題ではないということがはっきりしてきたため、Winny 1 の開発途中でこの機能は削除し、固定的な暗号キーを使うようになっています。そして、ユーザー設定の暗号キーの代わりに導入されたのが、現在のトリップ機能です。

内部的には、Winny 2 にも暗号キーを設定するメカニズムは残されていますが、現在は使用されていない機能となっています。

この暗号キーの扱いに関連して問題になったのが、検索キーの ID です。ファイル ID と暗号キーとは連動しているからです。

キャッシュファイル形式の変遷とファイル ID

ファイル ID は、キャッシュファイルのバージョンと密接な関係があります。V1 キャッシュのころは、ファイル ID にはファイル全体のハッシュ値ではなく、ファイルの先頭ブロックから求めたハッシュ値とファイルの大きさとを組み合わせた値を使用していました。ファイル全体のハッシュ値を求めるためにはファイルの内容を全部読み出す必要があり、時間がかかりすぎると考えたためです。また当初は、ファイルの破損がそれほど問題になるとは思っていなかったのです。

しかし、この設計の手抜きがあとで問題になります。ファイルは多数のノードを経由して流通するため、キャッシュファイルの破損は非常にシビアな問題だということが判明しました。加えて他の問題も明らかになったため[*10]、ファイル ID をファイル全体から求める方式に変えることになったのです。これは、キャッシュファイルのバージョンアップ（V1 から V2 に変更）をともなう大がかりな変更になりました。

V2 キャッシュのころは、まだユーザーが暗号キーを設定可能だったので、せっかくファイル全体を走査して ID を算出しても、同じファイルの ID が暗号キーによっ

*10 「同名ファイル問題」（別のファイルに同じ名前をつけられた場合の対処問題）。キャッシュファイル名にキー名を含めているのが問題だったため、キャッシュファイルヘッダを変更する理由にもなりました。

て変わってしまいます。このため、ファイルのハッシュ値をそのままファイルのIDに採用するのではなく、任意の暗号キーで暗号化して求めた各ブロックハッシュ値を順にXOR演算で加算した値を用いていました。これは単に暗号化したファイル全体のハッシュ値を求めるのが面倒だったからなのですが、この方法も固定的なキーを使うようになったV4キャッシュの採用時に問題になります。

ハッシュ値をファイル全体から計算する

Winny 1のβ1公開から半年以上たち、Winny 1の開発が後半にさしかかったころ、Winny本体をWinnyネットワークで流すという実験が行われました。このときに、特定のファイルを別のものにすり替え可能かどうか集中的にテストされています。しかし、本来クローズドなはずのキャッシュファイル形式やメカニズムが解析され、新たな悪戯が編み出されました。

それは、暗号化されているキャッシュファイルを、内部のブロック単位で入れ替え、同じIDになるような贋のファイルを作って、Winnyネットワーク上に流すというものです。V4キャッシュではキャッシュブロック単位のハッシュ値をXOR演算しているだけなので、これでもID値が変化しないことに目をつけた悪戯です。

そのためV5キャッシュ以降は、検索キーのIDとしてファイル全体のMD5ハッシュ値をそのまま使うようになっています。すでに暗号キーは固定で、検索IDが暗号キー依存ではなくなったため、一般的によく使われるファイルのMD5ハッシュ値そのものを採用することが可能になりました。これはユーザーの利便性の点からも好都合です。これにより、ダウンロードしたファイルのハッシュ値とファイルのIDが一致するかどうかをチェックするだけで、改ざんを検出できるようになりました。

Winnyの設計当初はWinnyがここまで広まると想定していなかったため、このようなシステム妨害への想定が甘く、後付けの対処をすることになりました。設計上、失敗した部分だといえます。

5.7 Winny 1開発のあとで

Winny 1に当初予定していた実装はほぼ終了しています。作者の感想として、Winnyというプログラムそのものが予想以上に広まったということがあります。

Winny 1は、ファイル共有効率のよい第三世代ファイル共有ソフトはありうるの

か、また実際に広まるだろうかという、技術的な実験色の濃いプロジェクトでした。実験としては想像以上にうまくいったといえるでしょう。ただ、Winny が普及した結果、ファイル共有ソフトの是非など、新たな問題を提起することになりました。これに関しては、時間が経ってみないと結論が出ないでしょう。検証された技術が社会でうまく活用されれば開発者の冥利につきますが、どうしても問題だというのであれば規制されることになるでしょう。

さて、Winny の話はこれでほぼ終わりですが、BBS 機能に関する話がまだ残っています。Winny 2 はまさにこの BBS 機能に特化したプロジェクトでしたが、その開発は途中で停まっています。しかし、P2P 方式の同様の BBS プロジェクトは現在もいくつか進行中です。

次章では、P2P 方式の BBS など、Winny 2 以降につながるその先の話について、少し考えてみることにしましょう。

5.8 まとめ

以上、Winny の開発を通して工夫した点や問題になった点を挙げ、今後の P2P アプリケーション開発で参考となるようにまとめました。

- P2P アプリケーションのテストは、ローカルな開発環境内で完結させることは難しく、いかにテスト協力者の力を得、できあがった P2P を維持するかが重要です。
- そのためには、できるだけ品質の高い状態でプログラムをリリースし、不具合は速いスピードで改修していく必要があります。
- 設計では長期的展望がたいせつであり、シミュレーションはそれを助ける手段となります。
- Winny の暗号化は、クラック対策とユーザーのプライバシー保護が主眼であり、匿名性とは関係がありません。
- Winny は、クラックを防ぎ、ネットワーク全体のトラフィックのバランスを保つために非公開のシステムとしました。しかし、今後の開発では検討を要する部分です。

残された課題と可能性

6章

2章「Winny紹介」で説明したように、Winnyにはファイル共有ソフトとして開発されたWinny 1と、P2Pによる大規模掲示板（BBS）を目指したWinny 2の2つのバージョンがあります。Winny 2のBBSとしての機能は、本書では取り扱ってきませんでした。それは、Winny 2がまだβテストの途中で完成からは程遠いところにあったということと、このP2P方式のBBSの分野は試行錯誤の段階でまだ全容が見えていないためです。

ファイル共有ソフトとしてのWinnyにはやり残したことはないと考え、ファイル共有ソフトとしてのWinny 1は開発と公開を終えています。しかし、技術的には別の可能性がいろいろ残っています。

6.1 BBS機能とアクセスコントロール

Winny 1には、アクセスコントロールの機能がありません。これは、匿名性と効率の両立を考えると、実現は技術的に無理だろうと考えたからです。しかしWinny 2の開発中、分散させたBBSをなんとか管理可能にする必要があると考えていました。その機能をファイル共有にも応用すれば、実現の可能性があります。

Winny 2は、BBSに書き込んだ人には匿名性がありますが、スレッド（掲示板上の話題）を立てた人には匿名性はありません。これは、Winny 2がBBSをどのような仕組みで実現しているかということと関連しています。Winny 2のBBSは、1つのスレッドをサービスするBBSノードをP2Pネットワークで束ねることによって大きなBBSに見せています。つまり、1つのスレッドは、あるノードが公開しているBBSだという実装です。BBSスレッドを立てているノードのIPアドレスは容易に特定できるので、スレッドを立てた人は特定可能です。

このことは、BBSの運用面でも、書き込みの管理責任をはっきりできるという利点もあります。しかし、実際の利用シーンではスレッドを立てた人の匿名性を護りながら、なおかつBBSの内容を管理できるようにすることが望まれます。

6.2 デジタル認証とアクセスコントロール

以上のようなわけで、Winny 2 では BBS スレッドを分散モデルでサービスしつつ、管理ができるような仕組みを開発する予定でした。デジタル認証によって、はじめにスレッドを作成した人がスレッドの内容を管理できるようにしようとしたのです。そして、これをファイル共有にも応用すれば、一度公開したファイルを削除する道へとつながります。

1 章「P2P の基礎知識」で述べたように、ピュア P2P 型の第二世代以降のファイル共有ソフトでは、ネットワーク全体のシステム管理や、一度配布した情報のコントロールは難しくなっています。しかし、デジタル認証技術と組み合わせることによって、この問題は乗り越えられる可能性があります。これが実現できれば、はじめに情報を公開した人と削除専用の管理者は、特定のファイルをネットワーク全体から消去できるようになるでしょう。

現在、Winny の開発は継続できない状況なので、このアイデアの成否は未確認のままですが、もし開発を進めることができれば、開発が停止したあとのウィルス騒ぎや情報流出騒ぎの対策ともあわせて、検討することは可能だったでしょう。

6.3 Winny 2 とファイル共有機能

Winny 2 では、内部的に Winny 1 のファイル共有機能を残しています。大規模 BBS を実現するためには、その P2P ネットワークを形成するノードを確保する必要があり、Winny 1 ですでに形成されている Winny ネットワークを活用するのが早道だというのが Winny 2 開発当初の私の考えでした。BBS の仕組みは機能的に見てファイル共有の仕組みと共通する部分があるため、ファイル共有の機能とは独立に BBS 機能を持たせるより、ファイル共有機構上に BBS を実現するのが現実的と判断したわけです。

しかし実際に動作するようになってみると、P2P 方式の BBS を維持するだけであれば、数万ものノードは必要ないことがわかってきました。レスポンスを考えれば、ノードが多すぎるのはむしろ逆効果で、安定した数百程度のノードでネットワーク

を維持するほうがよいというのが私の現在の考えです。

いまにしてみると、Winny 2 にはファイル共有機能はなくてよかったのだと思います。BBS 専用に設計していれば、また別の展開が考えられたでしょう。

6.4 ファイル共有ソフトの応用を考える

前述のように、ダウンロードや消去に関するアクセスコントロールが可能になれば、自分だけに見える形にしてファイルを放出し、P2P ネットワークを巨大なバックアップシステムとして利用することも考えられます。この場合は、プライバシーを保護するために匿名性が重要な技術となります。

また現在、著作物の不正な利用が問題になっています。しかしこの問題を解決するには、ファイル共有ソフトの技術だけではなく、課金システムや認証システム、また社会のシステムが伴なわなければいけません。実際に、DRM（著作権管理技術）が施されたファイルを流通させたり、商用に向いたシステムと組み合わせた P2P ネットワークも考えられ、その場合はファイル共有ソフトが著作物の流通に適したインフラになるはずです。Winny ネットワークは、そのようなファイル共有ソフトが大規模な情報共有インフラとして利用できることを示したのではないでしょうか。このようなインフラをどう応用していくかが今後の課題です。

あとがきに代えて

Winny 1にしても、Winny 2にしても、もともとは「できるかな？」という実験的な気持ちで始めたプロジェクトです。私にしてみれば他のちょっとしたフリーソフトを作るときと変わりはありません。私にとって、プログラムは表現手段であり、いつも、言葉で語るよりは実際に動作することを見せたほうが早いと考えます。これまでも、何かアイデアを思いつくと、ちょっとした実証プログラムという形で公開してきました。Winnyも、プロクシーのメカニズムの応用で効率と匿名性を両立できるのではないかと思いついて作った、そのような検証ソフトウェアのひとつだったのです。しかし、これほど広まることは完全に予想外でした。

歴史を振り返ると、情報を伝えるための新しい技術が何か確立されるたびに、暮らしに取り入れられ、新しい生活スタイルが形成されてきました。新しい生活スタイルはさらに新しい技術を促すというふうに相互に影響を与えながら、技術は進歩してきたのです。P2P型ファイル共有ソフトの原型といわれるNapsterが登場したのは、情報や家電製品のデジタル化が進み、多くの人がネットワークを利用するようになってきた1999年です。デジタルデータとネットワークが当たり前になった時代になり、その時代が生みだした技術だったのではないでしょうか。

Winnyは、数あるファイル共有ソフトのなかのひとつとして生まれました。おそらく今後も、いろいろなアイデアを採り入れたファイル共有ソフトが誕生していくでしょう。

Winnyネットワークは非常に多くの人に利用されました。このことは、ファイル共有ソフトの実用性や大規模な運用に耐えることを示していると思います。利用者に利便性があるかぎり、確立された技術がなくなることはありません。だとすれば、ファイル共有ソフトで生じている複雑な問題を順に解きほぐしていき、むしろ有効に活用する道を検討すべきではないでしょうか。

そのような明るい活用をお見せできるように、私もしばらく自分の力でできることを探っていくつもりです。

付録A

キャッシュファイルヘッダ

Winny の現在のキャッシュファイルヘッダの構造を表 A-1 に示します。当初はかなり余裕（予約領域）をもって作ったつもりでしたが、V6 キャッシュのヘッダを設計する時点でほとんど空きがなくなり、現在ではヘッダの他の領域を流用しています。

オフセット（バイト）	サイズ（バイト）	内　容	導入バージョン
000	10	ヘッダブロックハッシュ値（MD5）	V1
010	04	ヘッダバージョン（int）	V1
014	04	ファイルサイズ（DWORD）	V1
018	04	被参照量（ブロック数換算、DWORD）	V1
01C	04	アクセス日時（UNIX time）	V1
020	10	ファイル全体のハッシュ値（MD5）	V1
030	0B	ファイルトリップ（文字列）	V3
03B	02	予約領域	
03D	01	BBS ファイルかどうかのフラグ	V4
03E	01	ファイル名長（BYTE）	V2
03F	01	ハッシュ値の再計算フラグ	V2
040	F0	暗号化されたファイル名	V2
130	0C	BBS 認証用複号キー（文字列）	V6
13C	04	BBS の最終参照位置（int、発言番号）	V6
140	10000	キャッシュブロックビットマップ	V1

付録B

シミュレーション

次ページの画面「検索リンクのシミュレーション――100 ノードの場合」は、それぞれ 100 個のファイルを公開している 100 ノードに対して 4.2「ノード管理」で説明したような基本ルールを適用し、検索リンクをうまく張れるかどうか、また検索の成功率と検索リンクで生じる通信量がそれぞれどのくらいかをシミュレーションしています。ただしこの例では、各ノードのリンク接続数を、上流 1、下流 5 + α としています。

図中の黄色い節（点）はノードです。ノードは検索リンクを確保できなければ赤色で表示されますが、この画面では該当するものはありません。

赤色、青色、水色の四角は、検索リンクを飛び交っている検索パケットです。おそらく、青色の四角が上流検索クエリで、赤色の四角は検索にヒットして戻ってくるクエリであったと思います。

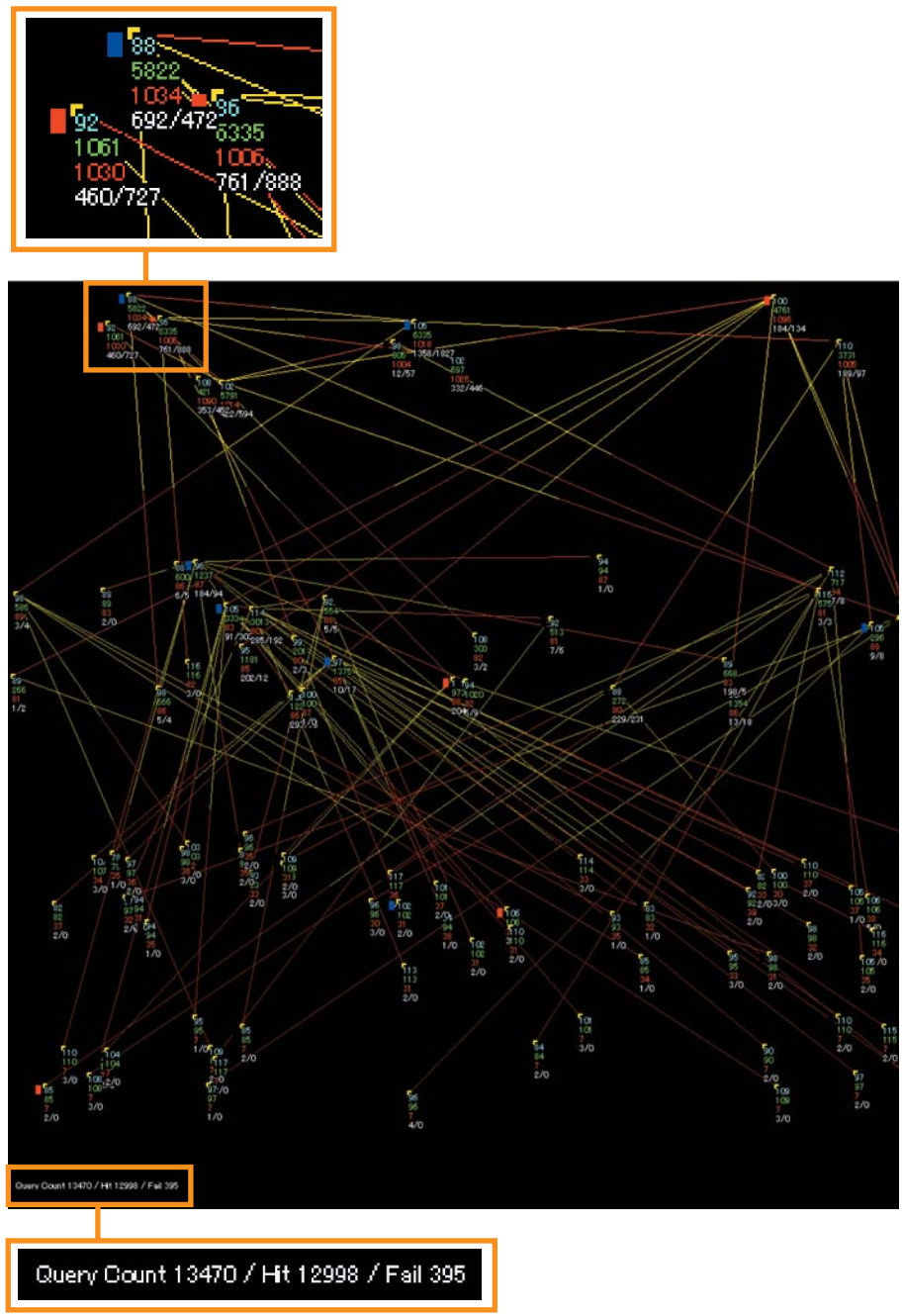
各ノード間を結ぶ検索リンクは、片側が上流を示す黄色、他方は下流を示す赤色で表示されます。

各ノードの脇に、公開しているファイル数（水色）、仮想キーの保有数（緑色）、回線速度（赤色、K バイト/秒）、通信で発生したダウンロード総量とアップロード総量（白色、K バイト/秒）が順に表示されています。

画面左下の表示は、Winny ネットワーク内で発行されたクエリ試行回数、その結果成功した回数と失敗した回数です。

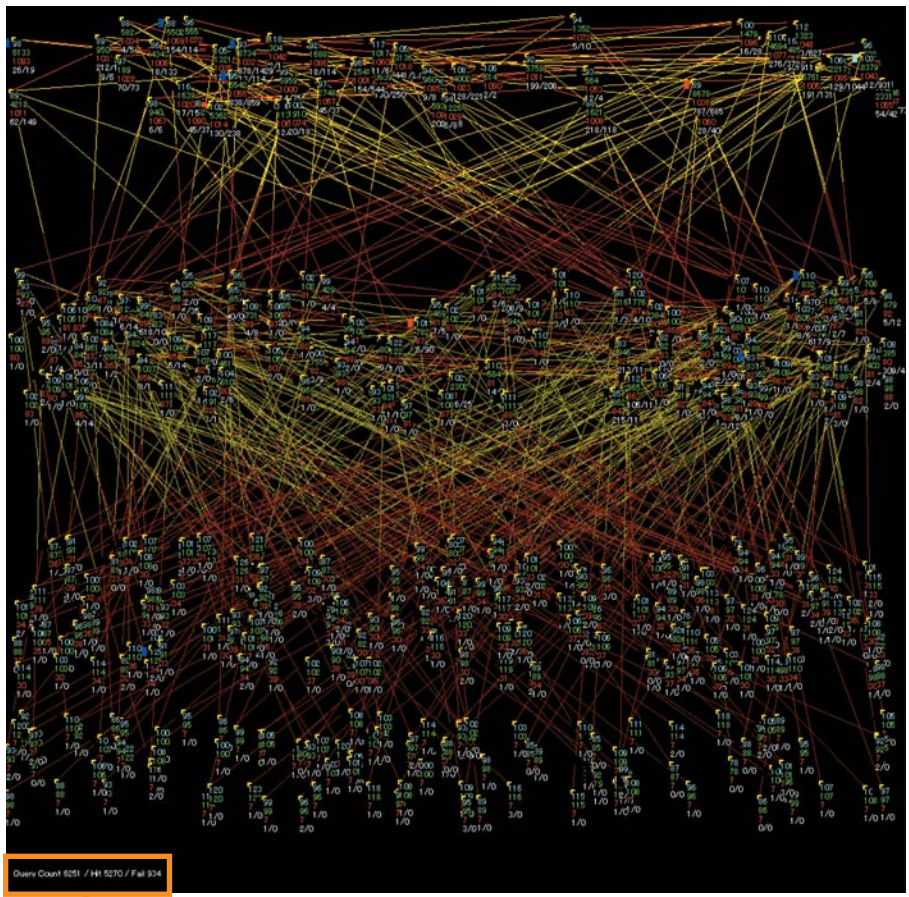
このシミュレーションでは、当時の通信事情から、各ノードが使用している ISDN 回線、低速 ADSL 回線、高速 ADSL 回線、光回線の比率を、2：4：3：1 と想定しています。画面の上方には高速回線のノード、下方には低速回線のノードを表示していますが、使用回線の種類によって 4 階層に分かれて表示されています。各ノードは適当な確率でオフラインになり、検索リンクが切れます。

ネットワーク上のファイル数は全部で 1 万個になりますが、よく見ると、上流ノードにはその半数のキーを持ってるノードがあります。



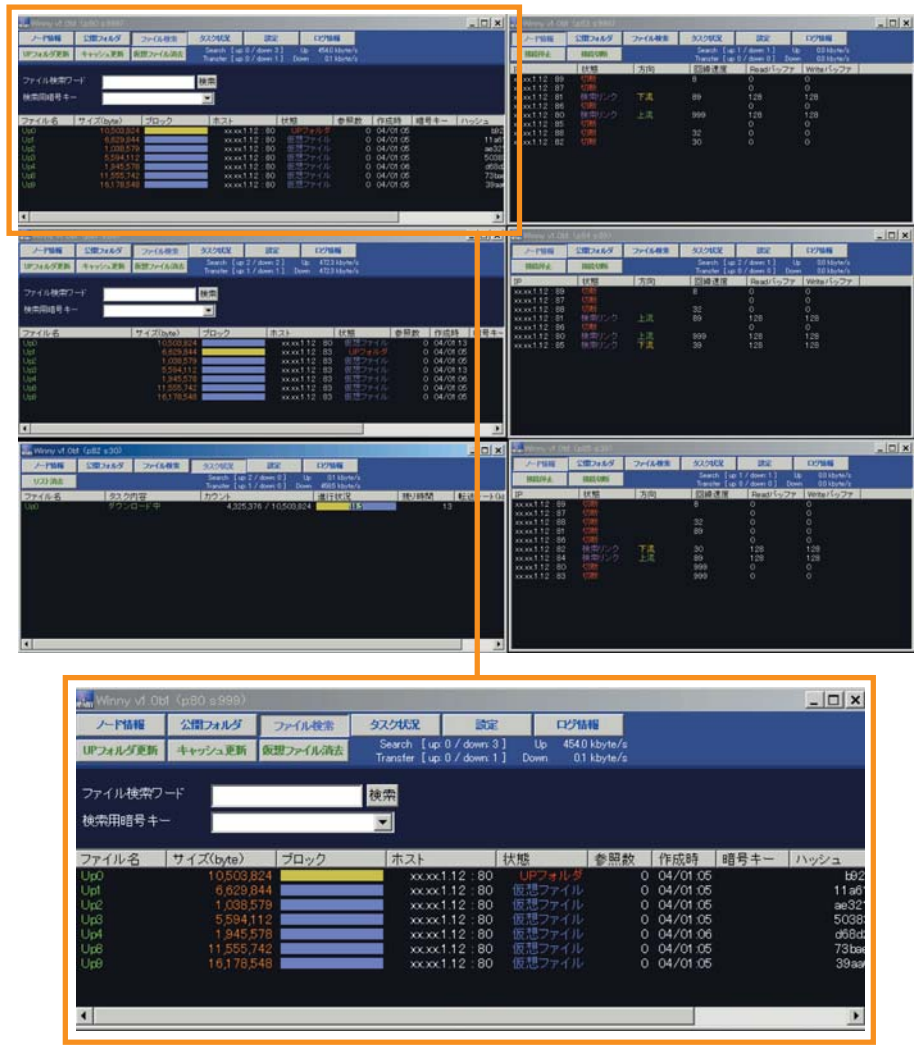
検索リンクのシミュレーション――100 ノードの場合

次の画面「検索リンクのシミュレーション――500 ノードの場合」は、同じシミュレーションプログラムを使ってノード数を 500 に増やしたところです。全体として検索クエリのヒット率がそれほど落ちないことを確認しました。



検索リンクのシミュレーション――500 ノードの場合

なお、このプログラムではキャッシュ機構や転送リンクのシミュレーションはしていません。ファイルの転送については、転送実験だけをすませたあと、Winny β 1 を公開しています。次の画面「ファイルの転送実験」は、ローカル環境で複数のノードを実際に接続させて転送のようすを観察しているところです（画面では 6 ノード）。各ノードのタイトルバーには、仮想ノードのポート番号 (p)、回線速度 (s) が表示されています。



ファイル転送の実験

付録C

Winny プロトコル詳細（2.0b7.1）

ここでは、技術資料として、Winny がノード間の通信に使用しているプロトコルをまとめます。プロトコル自体は非常に簡単なものですが、プロトコルを他の方が見ることは考えていなかったので、整理されているものではありません。

以下は、簡単なメモ程度のものですが、この資料が参考になるのは P2P アプリケーションの研究や開発に携わる方に限られるので、意味するところは理解していただけるでしょう。

C.1 基本的な規則

- 検索リンク、転送リンクのどちらでも、使用するプロトコルは共通です。転送系のコマンドは検索リンクでも使用されます。
- コマンドに対する応答は基本的にはありません。しかし、コマンドを送った結果、接続相手から新たなコマンドを応答として受け取るものがあります。
- プロトコル中のバイト並びはすべてリトルエンディアンです。
- コマンド 31 はコネクション切断要求ですが、存在しないコマンド番号を受け取ったときも切断要求として解釈します。
- 同一プロトコル内での接続バージョンを限定するために、コネクション切断コマンドには例外があります（Winny 2.0b7.1 ではコマンド 96〜99 は切断しません。かつコマンド 00 送信直前にコマンド 97 を送信します）。
- プロトコル解釈でなんらかの矛盾が生じたら、コネクションを強制切断します。
- クラック対策のため、通信内容全体が暗号化されています。また、一部分は多重に暗号化されていますが、詳細は省略します。

以降では、各コマンドとその引数となる項目を記述します。

基本プロトコル

- Winny は、リンクのコネクションを確立したあと、まず最初に以降の通信内容の暗号化に使用する 6 バイトの情報を、そのあとにプロトコルコマンドを含む一連の通信ブロックを送ります。
- 暗号化に使用する 6 バイトの情報のうち、先頭 2 バイトはダミー（乱数で生成）で、次の 4 バイトが RC4 の暗号キー（乱数で生成）です。

通信ブロック

- 先頭 4 バイトはブロック長です。
- 次の 1 バイトはコマンド番号です。
- 5 バイト目以降は任意長のコマンド引数です。

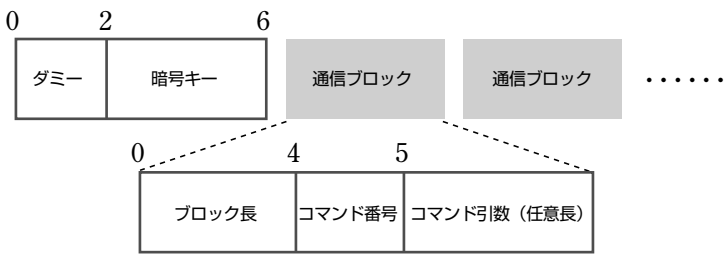


図 C-1　基本プロトコル形式

C.2 コマンド

コマンド 00——Winny プロトコルヘッダ

0 0
Winny 本体のバージョン情報
プロトコル認証文字列

- TCP コネクションの確立後、暗号キーに続けて最初に送るコマンドです（Connect 側、Accept 側双方）。
- 続けてコマンド 01 を送信します。
- BBS 検索リンクの場合、このコマンドの受信側ノードは応答としてコマンド 05 を返送します（待ち合わせは必要ない）。
- 接続相手のプロトコルバージョンに応じて警告を表示します。
- 認証文字列が違っていたら接続を強制切断します。

コマンド 01——回線速度の通知

0 1
自ノードの回線速度（K バイト/秒）

- コマンド 00 の送信後に続けて送信します。
- このコマンドに続けてコマンド 02 を送信します。

コマンド 02——コネクション種別の通知

0 2	
リンクの種別	…… 0：検索リンク／1：転送リンク／2：BBS 検索リンク
Port0 フラグ	…… 0：自ノードが非 Port0 ／ 1：自ノードが Port0 ノード
BadPort0 フラグ	…… 0：通常接続／ 1：BadPort0 判定のための逆接続
BBS リンクフラグ	…… 0：ファイル共有リンク／ 1：BBS リンク（BBS 転送リンクを含む）

- コマンド 01 の送信後に続けて送信します。
- このコマンドに続けてコマンド 03 を送信します。

コマンド 03――自ノード情報の通知

0 3
自ノードの IP アドレス
ファイル転送用待ち受けポート番号
DDNS 名文字列長
クラスタリング文字列長 1～3
DDNS 名文字列（任意長）
クラスタリング文字列 1（任意長）
クラスタリング文字列 2（任意長）
クラスタリング文字列 3（任意長）

- コマンド 02 の送信後に続けて送信します。
- このコマンドの受信でコネクション開始時の認証は終了します。
- 自ノードの IP アドレスを送信します。受信側ノードは、コマンド内の IP アドレスと実際の通信に使っている IP アドレスが異なる場合、送信側ノード（このコマンドの送信者）を NAT ノードとして扱います。
- DDNS 文字列とクラスタリング文字列に終端の NULL 文字（\0）はありません。

コマンド 04――他ノード情報の通知

0 4	
他ノードの IP アドレス	
ファイル転送用待ち受けポート番号	
BBS 待ち受けポート番号	
BBS ノードフラグ	…… 0：ファイル共有ノード／1：BBS ノード
回線速度	
クラスタリング文字列長 1～3	
クラスタリング文字列 1（任意長）	
クラスタリング文字列 2（任意長）	
クラスタリング文字列 3（任意長）	

- 検索リンクの接続要求を受け取ったものの、すでに検索リンクを最大数まで接続している場合、このコマンドによって他のノード情報を送信します。
- 送信するノード情報の数だけコマンド 04 を連続して送信します。
- このコマンドを必要数送ったあと、コマンド 31 を送信します。受信側ノードはそれを機に、接続を切断します。
- クラスタリング文字列に終端の NULL 文字 (\0) はありません。

コマンド 05——BBS ポート番号通知

0 5
自ノード BBS 待ち受けポート番号

- BBS リンクの場合、コマンド 00 への応答として BBS ポート番号を返します。

コマンド 10——拡散クエリ送信要求

1 0

- 30 秒に 1 回、隣接ノードにコマンド 10 を送信します。
- 受信側ノードは、任意数のキーをクエリ (コマンド 13) に詰めて返送します。

コマンド 11——キャッシュファイル送信要求

1 1	
ダウンロード側のタスク ID	…… コマンド 21 で返ってくる
送信開始ブロック番号	
送信ブロック数	
要求するファイルのハッシュ値	
要求するファイルのサイズ	

- ファイルのハッシュ値とブロック位置を指定して、キャッシュファイルの送信を要求します。
- 連続しているブロックのみ指定できます。取得したいブロック範囲が分断されている場合は、コマンド 11 を連続して発行します。
- 受信側はコマンド 21 で応答を返信します。

●受信側ノードがブロック本体を送信できない場合、タイムアウトで切断します。

コマンド 12――拡散クエリ転送要求（検索条件付き）

1 2	
検索するファイル名部分文字列	
検索するトリップ文字列	
検索クエリ ID	…… コマンド 13 で返ってくる

●下流の検索リンクに向けてキーを検索します。送信する検索クエリは孫ノード以下には伝わりません。

●受信側は、条件にマッチするキーをクエリ（コマンド 13）に詰めて返送します。

コマンド 13――クエリ送信

1 3	
返答クエリフラグ	…… 0：問い合わせクエリ／1：返答クエリ
拡散クエリフラグ	…… 0：通常クエリ／1：拡散クエリ
下流クエリフラグ	…… 0：上昇クエリ／1：一時下降クエリ
BBS クエリフラグ	…… 0：ファイルクエリ／1：BBS クエリ
クエリ ID	
検索キーワード文字列長	
検索キーワード（任意長）	
検索トリップ	
経由ノード情報リスト個数	
クエリ経由ノードの IP アドレス	
クエリ経由ノードのポート番号 :	
キー情報リスト個数	
キーの IP アドレス	
キーのポート番号	
BBS スレッド管理ノードの IP アドレス	

BBS スレッド管理ノードのポート番号	
ファイルサイズ	
ファイルのハッシュ値	
ファイル名長	
ファイル名（任意長＋暗号化情報）	
トリップ情報	
BBS トリップ認証情報長	
BBS トリップ認証情報（任意長）	
キー消滅判定タイマー	
被参照ブロック数	
キー更新日時（UNIX Time）	
キー無視フラグ	…… 0 以外ならキー無視警告
キーバージョン	…… 4、5、または 6
:	

- 隣接ノードへのキー送信すべてにこのコマンドを使用します。
- 検索時には、検索条件を付けて送信します（拡散クエリの場合は空）。
- ハッシュ値で検索する場合は、“%”に続けてハッシュ値文字列を指定します。
- クエリは適当なホップ数だけ上流ノードを検索したあと戻ってきます。
- 検索結果の返送や拡散クエリにもコマンド 13 を使用します。
- 受信側は、クエリの種類に応じて他のノードにクエリを中継します。この中継にもコマンド 13 を使用します。

コマンド 15——BBS 拡散クエリ転送要求（検索条件付き）

15
検索する BBS タイトル部分文字列
検索する BBS トリップ文字列
検索クエリ ID

…… コマンド 13 で返ってくる

- BBS キー検索時に下流 BBS 検索リンクに送信されます。クエリは孫ノード以下には伝わりません。
- 受信側は、条件にマッチするキーをクエリ（コマンド 13）に詰めて返送します。

コマンド 16——BBS 拡散クエリ転送要求

16

- 10 秒に 1 回、隣の BBS ノードにコマンド 16 を送信します。
- 受信側は、任意個の BBS キーをクエリ（コマンド 13）に詰めて返送します。

コマンド 17——BBS キー転送要求（検索条件付き）

17
検索条件（BBS 板名）

- BBS クラスタに参加していないノードが BBS キー一覧を取得するとき、送信します。
- 受信側は、対応する BBS キーをクエリ（コマンド 13）に詰めて返送します。

コマンド 21——キャッシュファイルブロック送信

21
コマンド 11 に対応するタスク ID
送信ブロック位置
キャッシュファイルブロック

- コマンド 11 への応答として該当キャッシュファイルブロックを送信します。
- 1 つの通信ブロックがコマンド 21 ブロック 1 つに対応するので、連続して受信

することがあります。

コマンド 22――Port0 ノードに対して転送リンクの接続を要請

2 2
ダウンロード要求ノード IP アドレス
ダウンロード要求ノード待ち受けポート番号

- Port0 ノードが持つファイルを要求する場合、ダウンロード側ノードは Port0 ノードの上流ノード（中継ノード）に対して要求を送り、中継ノードは Port0 ノードにコマンド 22 を、ダウンロード側ノードにはコマンド 23 を送信します。
- 中継ノードはコマンド 22 を使用し、Port0 ノードに対して転送リンクの接続先となるノード情報を通知します。
- コマンド 22 を受信した Port0 ノードは、指示されたノードへ転送リンクを接続し、コマンド 11 を待ちます。
- ただし、ダウンロード側ノードも Port0 ノードである場合、中継ノードはコマンド 22、23 を発行しないで、検索リンクを経由してアップロード側 Port0 ノードからファイルを取得し、ダウンロード側ノードに転送します。

コマンド 23――ダウンロード側ノードに Port0 ノード情報を通知

2 3
Port0 ノードの認証用 IP アドレス
Port0 ノードの認証用ポート番号

- Port0 ノードの中継ノードはコマンド 23 を使用して、ダウンロード側ノードに Port0 ノードのノード情報を送信します。
- このコマンドを受け取ったダウンロード側ノードは、指示されたノードからの転送リンクの接続を受け付けたあと、コマンド 11 を送信して Port0 ノードからダウンロードを開始します。
- コマンド 23 で送信されるノード情報は認証用で、転送リンク接続後のコマンド 03 で受け取る情報と比較することで、アップロード要求を受けた Port0 ノードからの転送リンクかどうかを確認します。

コマンド 24――BBS 投稿

2 4
投稿する BBS の ID
投稿メッセージ（任意長）

- BBS 書き込み時に、BBS を立てているノードとすでにリンクを接続ずみの場合、コマンド 24 で BBS に投稿します。
- 受信側は、自ノード内 BBS への投稿結果をコマンド 27 で返します。

コマンド 25――BBS 投稿中継依頼

2 5
BBS 投稿接続先の IP アドレス
BBS 投稿接続先のポート番号
投稿する BBS の ID
投稿メッセージ（任意長）

- BBS 書き込み時に、BBS スレッドノードにまだ接続していない場合、BBS 検索リンクを経由して隣接ノードに投稿を依頼します。
- 投稿を依頼されたノードは、BBS リンクを新たに接続し、コマンド 24 で投稿します。
- コマンド 27 で受け取った書き込み成否を、コマンド 26 で投稿側ノードに返します。

コマンド 26――BBS 書き込み中継結果通知

2 6
中継する BBS の ID
中継結果

…… 0：投稿中継失敗／-1：投稿中継成功

- コマンド 25 による BBS 投稿中継依頼の結果を返します。
- コマンド 27 で書き込み結果を受け取った中継ノードは、コマンド 26 で投稿元ノードへ結果を転送します。

コマンド 27——BBS 書き込み結果の通知

2 7	
投稿する BBS の ID	
投稿結果	…… 0：投稿失敗／-1：投稿成功

- コマンド 24 に対する投稿結果を返します。

コマンド 31——コネクション切断要求

3 1

- コネクション切断を依頼します。
- この手順で切断できない場合、タイムアウトで強制切断します（コマンド 32、33、34、35、37 ほかの特殊切断コマンドによる切断も同様）。

コマンド 32——コネクション切断要求（転送限界）

3 2

- アップロード側の転送リンクが限界数に達していることをダウンロード側に通知します。

コマンド 33——コネクション切断要求（BadPort0 警告）

3 3

- コネクション逆接続テストに失敗した（通信に必要なポートが開放されていない）ことを通知します。

コマンド 34——コネクション切断要求（無視ノード警告）

3 4

- 不正キー送信による無視ノード警告を受けたことを伝えます。

コマンド 35――コネクション切断要求（低速ノード警告）

3 5

●アップロード速度が遅いため、低速ノード警告を受けたことを通知します。

コマンド 37――コネクション切断要求（捏造警告）

3 7

●捏造キー送信による警告を受けたことを通知します。

C.3 プロトコル送信手順

リンク接続開始時の認証手続き

Connect 側ノード	Accept 側ノード
（ファイル共有ノードの場合）	
03 02 01 00 key →	
	← key 00 01 02 03（正常接続時）
	← 04 04 …… 31（リンク最大数を超えた場合）
（BBS ノードの場合）	
03 02 01 00 key →	
	← key 00 01 02 03 05

キーの拡散

拡散を要求するノード	キーを拡散するノード
（隣接ノードに対する拡散要求）	
10 →	
	← 13

キーの検索

検索元ノード	転送先ノード 1	転送先ノード n
（上流ノードに対する検索要求）		
13 →		
	13 →	
		← 13
	← 13	
（下流ノードに対する検索要求）		
12 →		
	← 13	

ファイル転送

ダウンロード側ノード	アップロード側ノード
03 02 01 00 key →	
	← key 00 01 02 03
11 →	
	← 21 21 21
31 →	

Port0 ノードのファイル転送

ダウンロード側ノード	中継ノード	アップロード側ノード
（アップロード側 ≠ Port0）		
11 →		
	22 →	
	← 23	
03 02 01 00 key →		
		← key 00 01 02 03
11 →		
		← 21 21 21
31 →		

ダウンロード側ノード	中継ノード	アップロード側ノード
（ダウンロード側 = Port0）		
11 →		
	11 →	
		← 21 21 21
	← 21 21 21	

BBS への書き込み

書き込み側ノード	BBS 保持ノード
（BBS 保持ノードが BBS 検索リンクの隣接ノードの場合）	
24 →	
← 27	

書き込み側ノード	中継ノード	BBS 保持ノード
（BBS 保持ノードが BBS 検索リンクの隣接ノードではない場合）		
25 →		
	24 →	
		← 27
	← 26	

索　引

フ

ヘ

ホ

マ

ム

リ

レ

著者紹介

金子　勇（かねこ いさむ）

昭和 45 年 7 月生まれ。茨城大学大学院にて博士（工学）の学位を取得。情報システム科学専攻。専門はシミュレーション環境、OS、可視化など。ネットワークは専門外であったが、原子力研究所勤務時代に複数のスーパーコンピュータをネットワークで接続し、その計算結果を可視化する研究にかかわった。その後、フリーソフトとして公開していた CG ソフトの商用化、IPA の未踏ソフト事業などに参加したのち、東京大学で特任助手として実践的プログラミングの指導に従事した。

趣味は暇プロ。何かアイデアを思いつくと、プログラムという形で表現し、検証してきた。小さいころからプログラミングを趣味とし、数々のプログラムを作成していまにいたる。

日常ではトラックボールを愛用し、キーボードを抱えたまま就寝、起きてまたキーボードに向かう。そのため電動式の起き上がりベッドを常用しているが、これは東急ハンズで買ったものであり、よく噂されているような介護用ベッドではない。

執筆協力

赤嶋映子、川崎晋二、中野克平

Winny の技術

2013 年 7 月 26 日　配信

著　者　金子 勇（かねこ いさむ）
発行者　塚田正晃
発行所　株式会社アスキー・メディアワークス
〒 102-8584　東京都千代田区富士見 1-8-19

カバーデザイン　辻　憲二
カバー図案　Kei Hiraki
図　　版　冨田 万紀子