特別研究報告書

ピアツーピア環境におけるファイル発見を実現する
ネットワーク負荷を抑える転送先学習アルゴリズム

指導教官　　美濃 導彦 教授

京都大学工学部情報学科

川西 智也

平成14年2月8日

ピアツーピア環境におけるファイル発見を実現するネットワーク負荷を抑える転送先学習アルゴリズム

川西 智也

内容梗概

本報告書では、ネットワークにおいて互いに対等な関係で通信を行うピアツーピア型環境において、負荷を抑えつつ、ファイル発見を実現するための転送先学習アルゴリズムを提案し、シミュレーションによって確認した効果について述べる。

ネットワークにつながるコンピュータの数が多くなるとともに、ネットワーク上にあるファイルを活用したいという要求が高まってきた。そのような要求に応えるには一般的にクライアントサーバシステムが用いられてきた。ネットワークに接続されたコンピュータは、サーバがファイルの蓄積・検索・提供を行い、それを利用するクライアントから利用するという形態である。このシステムではサーバの負荷が高い一方、技術の進歩でクライアントの性能は飛躍的に高性能化しており、余剰能力が生じていた。

そこで、クライアントの持つ能力を活用するために 2 つのファイル発見プロトコルが提案された。1 つは蓄積・提供の機能をクライアントに持たせ、サーバがクライアントが持つファイルのインデックスを持ち、サーバが検索を行うプロトコル、そしてもう 1 つは、蓄積・検索・提供の機能をすべてのそのネットワークに接続するコンピュータに持たせ、それらが協働することでファイル発見を実現するプロトコルである。

本稿では、後者のシステムがネットワーク全体を監視し履歴情報を収集することが不可能である点、またどこに障害があったとしてもシステム全体が利用できなくなることがない点に着目し、後者のシステムを対象とする。以下、このシステムをピアツーピア型のファイル発見プロトコルと呼ぶ。

ピアツーピア型のファイル発見プロトコルとしては、Gnutella が有名である。Gnutella ではファイルの発見要求を伝える Query パケットをノード間で順に受け渡すことによってファイル発見を行う。ここで、Gnutella の実装では、転送を行うときにすべての隣接ノードに Query パケットをブロードキャストするため、Query パケットの転送が行われるごとに指数関数的に Query パケットの数が増えてしまう問題がある。Query パケットの数が増えると、転送処理のため

計算資源を消費し、同時実行しているアプリケーションに悪影響を与え、発見をしてからその結果が返ってくるまでの時間が長くなる。

そこで、本稿では「すべての隣接ノードに Query パケットを転送する」という処理を、「より多くより早く発見できる隣接ノードに限定して Query パケットを転送する」という処理に変更した手法を提案する。転送先を少することで、Query パケットの増加を抑え、ネットワークの負荷を抑えようとするものである。

Query パケットの転送先を限定するために確率的な転送機構を導入する。それによって、ファイルを多くかつ早く発見できるような隣接ノードに高い確率で転送し、そうでない隣接ノードに転送する確率を低くする。

Query パケットの転送確率を更新するために、転送先学習アルゴリズムを提案する。転送先学習アルゴリズムでは、Query パケットをどの隣接ノードに転送してファイルを早く多く発見できたかの情報を収集するために、従来の Gnutella において応答に使用される QueryHit パケットを活用する。Gnutella では、Query パケットの転送が順に転送され、要求されるファイルを持つノードに到達するとそこで QueryHit パケットが生成される。QueryHit パケットは対応する Query パケットがそのノードに到達するまでにたどった経路を逆向きに転送され、Query パケットを生成したノードまで送り返される。

本稿で提案する手法では、QueryHit パケットが送り返されていくときに Query パケットの転送確率を更新する。ノードにおいて Query パケットを転送する際には、この転送確率を使用して Query パケットの転送先となる隣接ノードを決定する。転送確率を更新する処理は IP ルーティングの分野に適用されて非常に優秀な効果が得られた AntNet という学習アルゴリズムを参考にした。

この方式で実際に発見数が向上したかどうかをシミュレーションにより調べた。その結果、同じネットワーク負荷で発見できる数を従来手法の 6 倍多く発見できることを確認した。また、ユーザが短い時間に数多く Query パケットを生成するような状況において、従来のブロードキャストによる場合では各ノードのキュー (処理待ちのパケットを置く場所) で待つパケットの数が爆発的に増加するのに対して、本方式を採用した場合ではキューで待つパケット数は安定していたことが確認された。また、応答にかかる時間について調べたところ、本方式の方が従来のブロードキャストの場合よりも短い時間で応答が得られることが分かった。このように、応答時間や、キューでの待ち時間、ネットワーク負荷の観点で優れていることが確認された。

An Algorithm of Learning Next Hop to Reduce Network Load for File Discovery

Tomoya KAWANISHI

Abstract

This paper presents an algorithm of learning next hop for file discovery to reduce network load in Peer-To-Peer network in which each computer communicate with each other on an equivalent basis, and verifies effectiveness of the algorithm by simulation experiments.

With increase of computers connected to the network, users want to access files in the network more. Client-server system has been commonly used to discover and utilize files via the network. Computers in a system on the network are classified into two kinds; a server to store, search and provide files, and clients to utilize them. In this system the server is usually under heavy load, while with the advancement of technology the clients have surplus performance of computing, storage, network bandwidth and so on.

Two types of file discovery protocol are suggested to deterilize its surplus resources. One is that clients store and provide files and a server only searches files to find the location with its index of files in clients. the other is that without any server, all computers connected to a network have the same functionality and collaborate one another to store, discover and provide files.

This paper puts focuses on the latter type for the following three reasons. First, it is easy to manage the system. Second, it has so high anonymity that it is impossible to keep watch and collect logs on the whole network. Third, no single node fauilure stops the whole functionality of the system.

Gnutella is one of the well-known file discovery protocols among the latter network protocols. In the Gnutella network, each node collaborates one another for the file discovery with a *Query* – a packet for the file discovery.

The file discovery mechanism of Gnutella is as follows. When users request a file a *Query* is generated on the node. The node forwards the *Query* to all of its *neighboring nodes* – the nodes it is connected to. The nodes that receive the *Query* forward copies of all of its *neighboring nodes*. This is broadcasting.

The nodes that have the requested file respond the Query by a *QueryHit*

– a packet to indicate where and what file is found. The nodes forward the *QueryHit* back by way of a reverse path where the corresponding *Query* was forwarded. The node that generates the *Query* gets *QueryHit*s and gets the requested file.

The broadcasting method increases the number of *Query* packets exponentially per one hop, which causes many problems. First, computing resources for forwarding the *Query* and *QueryHit*s may disturb applications users run at the same time. Second, it takes longer time to get *QueryHit*s for the node that generates a corresponding *Query*. Third, network load increases.

This paper suggests a mechanism of select *neighboring nodes* not "a node forwards copies of a *Query* to all the *neighboring nodes*", but "a node forwards copies of a *Query* to the limited *neighboring nodes* to get *QueryHit* more and faster". It is expected that limiting the number of nodes to forward *Queries* avoids the exponential increase of the number of packets.

To select a better *neighboring node* for the next hop, each node assignes probability is assigned to every *neighboring node*. The probabilities of *neighboring nodes* gets lower where a few *QueryHit*s get back slowly, and the probabilities of nodes gets higher where many *QueryHit*s get back early.

To refine the probabilities of forwarding on a node, the author proposes a reinforcement learning algorithm based on AntNet which shows excellent performance in IP routing.

This algorithm uses turn around time of *QueryHit* to determine how good the *neighboring node* is. The nodes refine the probability of forwarding according to information of how fast the *QueryHit* gets back on all nodes where a *QueryHit* is forwarded back on the path that the corresponding *Query* is forwarded. A node selects *neighboring nodes* to forward a *Query* according to the probability.

Simulation experiments show that our method can discover the six times more number of nodes that provide a target file with same network load. Under a situation that nodes generates *Queries* very frequently, the number of packets in queue increased rapidly in broadcasting but the number is more stable in the proposed method. Also,the proposed method can discover files more quickly than the broadcasting method.

ピアツーピア環境におけるファイル発見を実現する
ネットワーク負荷を抑える転送先学習アルゴリズム

目次

第1章　緒論

近年のネットワーク技術の進歩にともない、ネットワークに接続されたコンピュータに蓄積されたデータやプログラムを他のコンピュータから利用することが一般的になった。これはインターネットを介したファイル共有のためのサービスが登場したことによる部分が大きい。

このようなサービスはクライアントサーバシステムとして始まった。クライアントサーバシステムでは、すべてのファイルが保管されているサーバがあり、それを利用するクライアントからの要求によって、そのサーバにあるファイルが検索され、クライアントに提供される。

クライアントサーバシステムでは処理はすべてサーバで一括して行っていた。そのため、サーバには負荷が集中した。サーバは頻繁な能力増強に迫られた。一方クライアント側のコンピュータの高性能化は著しかった。多くのクライアントでは、その処理能力は十分活用されていなかった。

ここにピアツーピア型のシステムが登場した。ピアツーピア型のシステムではファイル提供のためのサーバを廃し、従来クライアントとしていたコンピュータに情報を蓄積、提供することを可能にした。

ピアツーピア型ではファイルのダウンロードのときに、ホストやサーバを介さずに提供するノードと必要とするノードが直接通信する。サーバの機能をクライアントに分散して持たせることで、従来のクライアントサーバシステムと同様の機能が実現される。

このようにすることによって、それまでサーバに集中していた処理の負荷がクライアントに分散化され、共有のための余分な手間・タイムラグを省くことができるようになった。クライアントサーバシステムでは、ユーザはファイルを共有したいとき一度サーバにそのファイルをアップロードすることが必要であった。しかし、ピアツーピア型のファイル共有ではファイルが生成されてから、アップロードするまでにあったタイムラグがない。ピアツーピア型のシステムでは情報の発生源に情報を蓄積し、そこへの検索を可能にしたからである。

このようなピアツーピア型のファイル共有を実現するプロトコルはいくつか登場してきているが、これらは大きく分けて、ハイブリッド型と純粋型に分かれる。

ハイブリッド型プロトコルでは、情報の提供は各ノードで行うが、どのノード

が何を持っているかを示すファイルのインデックスはサーバの持ちデータベースによって中央管理されており、必要なファイルを手に入れるときは、サーバに問い合わせる必要がある。この特徴から、以下、本稿ではハイブリッド型を中央管理型と呼ぶ。

一方、純粋型プロトコルでは情報の蓄積も検索もすべて各ノードにおいて分散的に行われる。集中的なデータベースの構築は行われず、どのノードが何を持っているかを他のノードが事前に知ることはない。以下、本稿では単にピアツーピア型と言った場合、純粋型ブロトコルを指すものとする。

ピアツーピア型プロトコルは、運用面、耐障害性、匿名性で中央管理型に比べて優れている。サーバが存在しないため、サーバの管理を行う人員が必要でなく、またサーバ構築・設計のための手間も必要ではない。また、そこに障害が発生するとシステムがすべて停止するような一点が存在しないため、耐障害性か強い。また、ネットワーク全体に対して履歴情報を保存することができないため、非常に匿名性が高い。

このような運用面・耐障害性・匿名性の観点から、本研究ではピアツーピア型に着目する。

しかし、ピアツーピア型の現在の実装には問題がある。ネットワーク負荷が高いことは特に大きな問題である。ピアツーピア型プロトコルとして代表的な Gnutella では、ファイル発見するときに使用する Query パケットの転送の仕組みに問題があるので、ネットワーク負荷が高くなってしまう。

Gnutella では、それぞれのノードは少数 (典型的な実装では 4) の Query パケットの転送先となるノードを持つ。これを隣接ノードと呼ぶ。Gnutella では、この隣接ノードすべてに対して Query パケットを転送することで、ファイル発見を行う。このため、Query パケットが順に転送されていくごとに Query パケットの数が指数関数的に増えてしまい、ネットワーク負荷が非常に高くなる。実際に Gnutella を使用すると 165kbps の帯域が平均して占有され、ネットワーク負荷が非常に高くなる [1]。これは広く利用されている ISDN 回線の 64kbps よりも大きく、より広帯域な ADSL 回線にとっても常時この帯域が使用されてしまうことは大きな負担である。

ネットワーク負荷が高いと、それを利用するユーザは欲しいファイルを発見することができなかったり、発見するまでに長い時間がかかったり、またユーザが使用している他のアプリケーションがネットワークを利用するときに悪影

響が生じたりする。接続回線の帯域が乏しいノードがGnutellaネットワークに参加することが実質的に不可能になるのはもちろんのことそのような参加者がネットワーク全体のパフォーマンスを下げてしまうことになる。参加することができても、ファイルをダウンロードする際に使うことができる帯域が小さくなってしまうのである。さらにネットワーク負荷を抑えるよう、Gnutellaのネットワークの各種パラメータを調整するという選択肢もあるが、それではファイルを多く発見することができない。

そのため、ネットワーク負荷の低減と多くのノードからファイルを発見することを両立し、さらに応答時間が短いような手法が必要である。本稿では、これを実現するような転送先を限定する学習アルゴリズムを提案する。

本報告書では、2章でファイル発見について説明し、3章で提案する転送先学習アルゴリズムについて説明する。4章で、様々な状況下でそれを用いた場合と、Gnutella方式との間で比較を行い、その実験結果について考察する。5章では結論および今後の課題について述べる。

第2章　ファイル発見

この章では広義のピアツーピア環境において、ファイルを発見・共有するために使用されている既存プロトコルについて説明し、その得失を比較する。

広義のピアツーピア環境とは、任意の複数のコンピュータ間で直接にコンピュータ資源(ファイル、計算能力等)を共有するような環境である。クライアントサーバシステムではサーバにあるコンピュータ資源を活用していたが、広義のピアツーピア環境ではサーバの持つ資源だけでなく、そのサービスに参加するすべてのコンピュータ資源が活用できるようになる。

2.1　既存のファイル発見プロトコル

広義のピアツーピア環境でのファイル共有とは、そのサービスに参加するコンピュータが持つファイルから直接的に検索・ダウンロードすることでファイルを発見・共有するものである。

発見とは、ある検索基準からそれに該当するファイルをどのノードが持っているかを特定することであり、共有とはそのファイルを直接にダウンロードすることで、複製を行うことである。

このようなファイル発見プロトコルには、大きく分けて中央管理型と狭義のピアツーピア型がある。中央管理型とは、どのファイルがどこにあるかをデータベースとしてまとめ、そのデータベースを持つサーバに対して、問い合わせることでファイルを発見する方式のことである。それに対してピアツーピア型は、そのようなデータベースを事前に構築することなく、ノード同士が協働し情報を交換しあうことによって、ファイル発見を行う方式である。

2.1.1 中央管理型

中央管理型では、他のコンピュータに蓄積されている情報を取得するために他のコンピュータが何を持っているかを中央管理する。

この型のプロトコルとして Napster が著名である。この Napster を参考に、この方式でのファイル共有について述べる。Napster はこの種のプロトコルとして最初のもので、MP3 ファイルを互いに共有するために開発された。MP3 とは、音声データを圧縮した形式の１つである。他の同様なプロトコルでは、細かな違いはあるが、ファイル共有を実現する手法はほとんど同様である。

Napster では次のような形でファイルを発見する。

- クライアントがサーバに接続する。
- サーバはクライアントの共有ディレクトリにあるファイルの情報を取得してデータベースに登録する。
- クライアントは欲しいファイルをサーバに問い合わせる。
- サーバはデータベースを検索して、どのクライアントがそのファイルを持っているかを調べ、問い合わせたその結果をクライアントに通知する。
- クライアントでは、どのクライアントからダウンロードするかを選び、サーバに入手先として希望するクライアントを通知する。
- サーバは指定された入手先クライアントと通知してきたクライアントとの間でサーバを経由しない直接のコネクションを確立できるようにする。

サーバではどのクライアントが何を持っているかという情報のみを管理する。クライアントではサーバに問い合わせることでファイルを発見し、ダウンロードするときはそれを持つクライアントと直接コネクションを張る。

2.1.2 ピアツーピア型

ピアツーピア型ではすべてのノードが完全に対等な立場で通信を行う。ノードは互いに協力しあうことによって求めるファイルを発見する。以下ではこの型のプロトコルとして著名な Gnutella を参考にしてどのようにファイル発見を

行うかについて説明する。

ファイル発見を行う方法

Gnutella では、IP 層でのネットワーク構造の上に独自の論理ネットワークを形成する。Gnutella では、ファイルのダウンロードのとき以外はその論理ネットワークの中で直接の接続関係にあるノードとのみ通信する。このような、あるノードから見て論理的に接続関係にあるノードのことを隣接ノードという。

通信相手をわずかな隣接ノードに限定することによって、非常に高い匿名性が実現される。限定された隣接ノードとのみ情報を交換するため、ネットワーク全体で何が行われているかを監視し、履歴情報を保存することは不可能である。

Gnutella では以下のようにしてファイル発見を行う。

1. ファイルを発見しようとするノードで、Query パケットが生成される。
2. Query パケットはすべての隣接ノードに対して転送される。
3. Query パケットを受信したノードは、そのすべての隣接ノードにさらに転送するという処理を繰り返す。
4. 転送されたノードに求められているファイルがあれば、QueryHit というパケットを生成する。
5. QueryHit パケットは、Query パケットが通った経路を逆向きに転送され、その Query パケットを生成したノードに QueryHit パケットが返される。

QueryHit パケットが Query パケットの通った経路を逆向きにたどるための仕組みは非常に簡単である。ノードは Query パケットを転送するときに、それがどの隣接ノードから来たのかを覚えておく。そのために Query パケットには識別子があり、ファイルを発見したときに生成される QueryHit パケットとその Query パケットの識別子を同じにする。ノードはある Query パケットに対応する QueryHit パケットを受信すると、その識別子を見て、対応する Query パケットが送られてきた隣接ノードに転送する。

以上が転送を行う仕組みである。しかしこのような転送を行う仕組みだけでは、際限なく転送が繰り返されることが防げない。それを防ぐために TTL(Time To Live) と複数回受信パケットの破棄の 2 つの仕組みが用意されている。

TTL とは、Query パケットに設定されている寿命のことである。TTL の値は生成されたときに決定され、Query パケットが転送されるたびに 1 ずつ減ぜられる。これによって、生成されたノードから、最初に Query パケットに設定された回数だけ、転送されると Query パケットは自動的に捨てられることにな

る[1)]。

複数回受信パケットの破棄とは、同じノードが同じ識別子の Query パケットを 2 回以上受信した場合、2 回目以降に受信されたパケットがそれ以上転送処理されず、捨てられることである。

2.2 両プロトコルの比較

これまでに示した 2 種類のプロトコルにはそれぞれ得失がある。

1. **ファイルの発見にかかる時間**　中央管理型の方が短い。中央管理型では、サーバで検索を行い、ファイル発見が一箇所で完結するため、比較的早く応答が返ることになる。ピアツーピア型では、順に受け渡される Query パケットが到達したノードからの応答を待つ必要があるので、時間が多くかかる。
2. **計算負荷**　中央管理型ではサーバの負荷は高いが、クライアント側の負荷は低い。ピアツーピア型の場合は、すべてのノードが同等の役割を持つため、比較的高い負荷になる。
3. **消費するネットワーク帯域**　これは計算負荷と同様であり、中央管理型ではサーバに負荷が集中する。しかし、ピアツーピア型では、負荷の集中は起こらないが、ノードでのネットワーク負荷が高くなる。
4. **人員コスト**　中央管理型では共用されるサーバを管理するために、技術の高い人を配置する必要があるので高い。ピアツーピア型では共用されるサーバが必要とならないため、そのような人材を確保する必要がない。
5. **導入コスト**　中央管理型では高性能なサーバを確保する必要があるので高い。ピアツーピア型ではそれに参加するノードの性能は高なければならないが、システムを構築するときには高性能なコンピュータ必要としないので、導入コストは比較的低い。
6. **耐障害性**　中央管理型ではファイル発見用のサーバがメンテナンスや、障害のため、一時的に停止してしまうとそのシステム全体が機能しなくなる。ピアツーピア型の場合はどのノードがそのシステムから離脱しても、ファイル発見の機能を他のノードが代替することができ、システム全体が停止することはない。

[1)] この値は典型的な実装では 7

7. **管理できる範囲**　中央管理型ではログイン、ログアウトの記録をとることができ、どのような検索が多いといった履歴の情報がサーバで取得することができる。ピアツーピア型ではそのようなログをとることはできない。
8. **匿名性**　中央管理型はクライアントはサーバに共有するすべてのファイルの情報を通知する必要がある。必要以上にクライアントの情報が公開されることになる。また、全体の情報がサーバで取得されるため、そこで履歴情報を保存できる。ピアツーピア型ではネットワーク全体を監視し履歴情報を保存することが原理的に不可能であるため、匿名性が高い。

この中でピアツーピア型の優位点として、どの一点が壊れても全体の機能は維持されるという点および匿名性の高さが重要である。どの一点が壊れても全体の機能は維持されるために、メンテナンスや突然の障害によって中断し、システム全体が停止することがない。また、ユーザは、自分がどんなファイルを検索し，共有しているかについて誰かに収集・管理されることを嫌うものである。ピアツーピア型では原理的に非常に匿名性が高く、この点でも優れている。

これらの利点は中央管理型で得られるものではなく、ピアツーピア型に固有のものである。

本研究ではこのような長所が実現されるピアツーピアの技術に着目し、以降ではピアツーピア型を対象とする。

第3章　ネットワーク負荷の抑制

この章では、ピアツーピア型ファイル発見プロトコルに分類され、現在広く利用されている実装の1つである Gnutella ではネットワークの負荷が非常に高くなってしまうことを指摘し、それを改善する手法を提案する。

3.1　ブロードキャストの問題点

ピアツーピア型ファイル発見では Query パケットを転送するときにブロードキャストする、つまりすべての隣接ノードに転送を行う結果、Query パケットの数が指数関数的に増加してしまう。この Query パケットの増加が数々の問題を引き起こす。

まず、Query パケットの数が増えると、ノードにかかる転送処理の負荷も高くなる。すると、ノードでの計算資源が消費され、また多くのネットワーク帯

域が消費されることになる。すると、ノードで実行されている他のアプリケーションにも悪影響が出ることになる。

また、Query パケットの数が増加すると、ファイルを発見してその結果が返ってくるまでに待つ時間も長くなる。それは Query パケットの数が増加すると、キューで待つ時間が長くなるからである。キューとはノードがパケットを受信したとき、すぐに処理できないため、一時的にそのパケットをためるバッファのことをいう。転送されたのち、パケットはキューで次のノードに転送されるまで順番を待つ必要がある。この順番待ちをする時間はノードで待つパケットの数に比例して長くなる。Query パケットの数が増加すると、ノードで待つパケットの数も必然的に増加し、その結果キューで待つ時間も長くなり、ファイルを発見するのにかかる時間も長くなることになる。

ネットワーク負荷が高くならないようにするためには、TTL や隣接ノード数の値を調整する方法がある。こうすると、ネットワーク負荷を抑えることはできるが、返ってくる応答の数はその分少なくなってしまうことになる。選択肢は多いほうが良いので応答の数はできるだけ多い方が望ましい。

そのため、応答の数を増やすことを可能にしながら、ネットワーク負荷を低く抑える手法が必要となってきている。

3.2 ネットワーク負荷の軽減手法

ピアツーピア型のファイル発見において、現在実装されているブロードキャストによって Query パケットを転送するという手法では Query パケットの数が指数関数的に増加する。Query パケットの転送をブロードキャストによって行う部分を改良することで Query パケットの増加を抑え、ネットワーク負荷を少なくすることができれば、より短い時間でファイルを発見できるようになる。しかし、単純にネットワーク負荷を抑えようとすると、発見できる数が少なくなってしまう。したがって発見できる数少なくしないことと、Query パケットの数の低減を両立することが目標となる。

基本的なアイデアは「すべての隣接ノードに Query パケットを転送する」代わりに「より多くより早く発見できる隣接ノードに Query パケットを限定的に転送する」ことである。転送先が少なくなればその分 Query パケットの数の増加は抑えられることになる。

このような方法が適用できる理由は、ノードが保有するファイルの量・質に

は偏りが大きく、応答を返す確率がノードによって大きく違うからである。

例えば、何もファイルを提供しないがそのサービスに参加している Free Rider と呼ばれるノードが Gnutella ネットワークには大量に存在していることが知られている。Gnutella ネットワークとは、Gnutella プロトコルで通信し合うノード群によって構成されるネットワークのことである。Gnutella ネットワークでは 70%のノードが Free Rider であると言われている [3]。

一方、一部のノードが大量のファイルを提供している。下位 1%のノードが 37% ものファイルを提供しており、上位 5%が全体の 70%、上位 20%が全体の 98% ものファイルを提供している [3]。つまり、ノードによって応答する確率に大きな偏りがある。

このように応答する確率に大きな違いがあることを利用して、応答が返る確率が高い隣接ノードに対して Query パケットを転送する確率を高くして、応答が返る確率が低い隣接ノードに対して Query パケットを転送する確率を低くする。

これによって、Query パケットの増加を抑えつつ、応答するのにかかる時間をより短くすることができると考えられる。Query パケットの増加の仕方が同じであれば、より多く発見できるようになることが期待できる。

転送先を限定するには、どのノードに転送しどのノードには転送しないのかを決定する必要がある。本提案手法では隣接ノードごとに転送する確率を設定し、その転送確率を 3.4 節で示す学習アルゴリズムによって逐次更新、改善していくという手法を用いる。このアルゴリズムでは早く応答が返る隣接ノードに転送する確率をより高くすることによって応答するのにかかる時間を短縮する。

転送確率の使用法とフィードバックの方法の部分が、本提案手法において最も重要な部分である。

3.3 転送先の決定と転送確率の更新

ここでは、本手法で、どのようにしてファイル発見を行うのかについて詳説する。

3.3.1 転送先の決定

ファイル発見を要求するノードにおいて Query パケットが生成されると、確率的な機構によりすべての隣接ノードの中から転送先隣接ノードが選ばれて、転送される。

ノードが Query パケットを受信すると、すぐに処理が可能な場合はただちに処

理するが、現在処理中のパケットが存在する場合は、そのパケットを一旦キューに入れて待たせる。

パケットを処理するときは、まず初めにTTLがゼロ、またはすでに受信したことがあるパケットかどうかを調べる。その場合は破棄し、そうでない場合は転送処理を行う。

転送処理を行うときは、その転送処理を行った時間を記憶しておく。これは、早く応答が返ってくるような隣接ノードに対してより転送しやすくするように学習するために使用する。

転送するときは次の作業をN回だけ繰り返す。Nは隣接ノードを選択する回数である。その結果、同じ隣接ノードが複数回選択されたとしても、実際には一回しか転送しない。パラメータNは受信した1つのQueryパケットをどれだけ多くの隣接ノードに転送を行うかを決めるパラメータで、Nを多くすると、比較的転送確率が低いノードに対しても、転送を行う可能性が高くなる。

転送するときの隣接ノードの選び方には、2とおりあり、そのどちらになるのかは探求確率P_eという確率で決定される。

- 探求確率P_eの確率で転送先をランダムに決定する。
- $(1 - P_e)$の確率で、隣接ノードごとの転送確率にしたがって転送する。

開始直後の状態では、転送確率はすべての隣接ノードに対して等確率となるため、ほぼランダムに転送されることになる。

学習が進むと、ファイルの発見頻度が高く、発見するのにかかる時間が短い隣接ノードに転送される確率が次第に高くなり、そうでないノードへ転送する確率は次第に低くなる。学習が進むと上の操作をN回行ったとしても、少数の隣接ノードにのみ転送する状態になることが期待される。そして、この結果、ネットワーク負荷を小さく抑えつつ高速に大量の目的ファイルを発見することができる。

探求確率P_eはネットワークの変化に対応しやすくするために用意されている。この仕組みがないと、硬直的な転送となり、今までにはなかった場所にファイルを提供するノードが出現したとしても、そのノードのファイルを発見することができない。P_eを高くすると、ネットワークの変化に対応する能力が高くなる。しかし、P_eを高くしすぎると、転送先の数が減りにくく、ネットワーク負荷を抑える効果が小さくなる。したがって、この2つの値のバランスをとることが重要である。

3.3.2 発見した時の処理

ファイルを発見したときは、 QueryHit パケットを生成する。QueryHit パケットは対応する Query パケットと同じ識別子を持つ。

3.3.3 応答と転送確率の更新

QueryHit パケットは対応する Query パケットがたどってきた経路を逆向きに転送される。QueryHit パケットが転送されるときには、それを転送してきた隣接ノードに対する Query パケットの転送確率を大きくし、それ以外の隣接ノードに対する転送確率を小さくする。このようにすることで、次回の Query パケットに対して発見確率をさらに高くすることができる。転送確率の更新はその Query パケットを生成したノードだけではなく、QueryHit パケットを中継する途中のノードに対しても行う。

転送確率の更新は早く QueryHit パケットが返ってくればくるほど強く学習を行うようにする。そうすると、早く応答する確率が高まる。短時間で応答が返せるようになる。

3.3.4 キューにおける待ち

キューとは、パケットをすぐに処理できない場合に、一時的にそれを保存しておき、パケットを処理できるような状態になってから処理を行うために溜めておくバッファのことである。

Query パケットの転送処理を先に述べたが、ノードに到達するとすぐに転送処理が行われるわけではない。転送処理が行われるためには、ノードに到達したのちにノードのキューで処理される順番を待つ必要がある。待つ順番は次のように決まっている。

1. QueryHit パケットはすでにキューにある Query パケットより優先され、Query パケットより前に挿入される。先に待っている QueryHit パケットがある場合はその最後にまわされる。
2. Query パケットが到達するとキューの一番最後に待たされる。

QueryHit パケットは Query パケットより重要であり、優先的に転送する。QueryHit パケットを優先的に転送することによって、応答が早くなるだけでなく、学習がより速く進む。

3.4 転送先学習アルゴリズム

3.4.1 AntNet

IP ルーティングにおける非常にロバストでかつ効率的な手法として、AntNet という手法が提案されている [6]。AntNet は強化学習の１つであり、アリの巣でのアリの挙動に触発されて着想された分散環境下における IP ルーティングアルゴリズムである。AntNet は様々な環境で実験された結果、既存のどんなアルゴリズムよりも、遅延や負荷の少ない転送が実現できることが示されている。

本方式ではこの AntNet を参考にして、転送アルゴリズムを設計した。その理由は、AntNet に、以下のような利点があったからである。

- IP ネットワークに対して実験した結果、良い結果が得られた実績があったこと。
- 自律分散型の学習アルゴリズムであること。
- アルゴリズムそのものが単純であること。
- Gnutella と対応をつけやすいこと。
- 保持する必要があるデータ量が少ないこと。

AntNet では、前進アリと後退アリの 2 つが IP ネットワーク上を動き回り、経路情報を集めることで、最適なルーティングを決定する。前進アリとは、新しいルートを探索するために動き回るエージェントとして働くパケットのことで、後退アリとは、前進アリにより探索・発見された結果をフィードバックするために生成されるパケットのことである。

AntNet では、ある宛先ノードに向かっているパケットを転送するとき、あるノードから次にどの隣接ノードへと転送するかを確率的に決定する。そのため、ノードには隣接ノードに対してその確率に従ってパケットは転送される。宛先ノードに到達すると、前進アリは、後退アリを生成する。後退アリは対応する前進アリがたどった経路を逆向きに進んで、パケットを生成したノードまで戻る。戻る途中のノードでは転送確率を更新していく。

この前進アリと後退アリの対応は、Gnutella のプロトコルにおける Query パケットと、QueryHit パケットの対応に非常に似ている。また、AntNet での隣接ノードは物理的な接続関係で、Gnutella での隣接ノードは論理的な接続関係であるという違いはあるが、十分対応づけが可能である。そこで、AntNet[6] における学習アルゴリズムを参考に、本方式ではファイル発見のための学習アル

ゴリズムを提案する。

3.4.2 転送先学習アルゴリズムの詳細

転送先学習アルゴリズムは本方式において性能を最も左右する部分である。

転送先学習アルゴリズムによって、Query パケットをある隣接ノードに転送する確率が決定される。Query パケットの転送の結果、QueryHit パケットが返されると、その QueryHit パケットがたどったノードすべてにおいて、この転送先学習アルゴリズムに従い、転送確率が更新される。

転送先学習アルゴリズムは次の 3 つの値から Query パケットの転送確率を更新する強さを決定する。

- 応答時間 T。Query パケットがそのノードを通り、対応する QueryHit パケット が、そのノードに戻ってくるまでの時間。
- そのノードでの応答時間の平均 μ。
- そのノードでの応答時間の標準偏差 d。

この転送先学習アルゴリズムでは、取得する必要がある情報はすべてそのノードのみから取得可能である自律的な手法である。応答時間 T はそのノードが転送時に時間を記憶することで分かり、そのノードでの応答時間の平均 μ や標準偏差 d はそのノードの過去の履歴から分かる。これらを取得するために外部のノードと情報を交換する必要はない。

また、ノードが保持する必要がある情報は応答時間 T を算出するために、Query パケットを転送した時刻を記憶しておくことと、応答時間の平均と標準偏差だけであり、非常に少ない。従来手法と比べて増えたのは、Query パケットを転送した時刻を記憶することである。Query パケットの識別子の数だけ記憶する必要があるが、従来手法でもその Query パケットが一度そのノードを通ったことがあるかどうかを覚えておく必要がある。

この転送先学習アルゴリズムでは、まず次のように r_0 を算出する。

$$r_0 = \begin{cases} \frac{T}{c\mu} & \text{if} \quad \frac{T}{c\mu} < 1 \\ 1 & \text{otherwise} \end{cases}$$

こうして算出される r_0 は、その応答時間 T がどれだけ良かったかを示す無次元の値である。r_0 は小さければ小さいほどその応答時間が良い値であったことを示す。ここで、μ は応答時間の平均、c はそれを補正するパラメータである。妥当な値として c は 2 とした。1 を超えた場合は 1 にする。この個所はオリジナル

の AntNet と同じ処理である。

次に、その応答時間 r_0 がどれだけ信頼できるかという観点で、次の式で算出される値 U を用いて、r を調整する。e は自然対数の底である。

$$
\begin{aligned}
U &= \begin{cases} \mathrm{e}^{-a\frac{d}{\mu}} & \text{if} \quad r_0 < t \\ \mathrm{e}^{-a'\frac{d}{\mu}} - 1 & \text{if} \quad r_0 > t \end{cases} \\
r_1 &= r_0 - U
\end{aligned}
$$

U は標準偏差 d に関して単調減少の関数であり、標準偏差 d が小さくなると、大きな値になる。U を引くことで r_0 の値を小さくし、良い値にする。この式で使用しているパラメータ a、a' はそれぞれ 10、9 に設定している。ここはオリジナルの AntNet では標準偏差が大きすぎる場合に対し、より複雑な処理を行っているが、本方式では簡略化した [6]。簡略化を行った理由は IP ルーティングとファイル発見の差に起因する。IP ルーティングでは同じノードに向かうことを目的あるため、往復にかかる時間はある程度一定である一方、ファイル発見では様々なノードから応答が返る可能性があるため応答が返る時間の変化が激しいからである。

ここで、閾値 t よりも r_1 が小さいとき (良い値のとき)、U は正であり、閾値 t よりも r_1 が大きいとき (悪い値のとき)、U は負の値となる。

これは、短い時間で応答が返ってきた場合に特に強く学習し、応答に時間がかかった場合はあまり学習しないようにするために行っている。この境界が閾値 t によって決定される。

次に、$r_1 \in [s, 1]$ となるように調整する。s は非常に小さな値で、過剰に学習することを防ぐために用意されたパラメータである。$s = 1.0 \times 10^{-20}$ に設定している。これはオリジナルの AntNet では $s = 0$ となっている。

$$
r_2 = \begin{cases} 1 & \text{if} \quad 1 < r_1 \\ r_1 & \text{if} \quad s < r_1 < 1 \\ s & \text{if} \quad r_1 < s \end{cases}
$$

最後にこの r_2 をべき乗して、圧縮した値にする。

$$
r_3 = r_2^h
$$

これによって、r_3 は大きく 1 に近づき、学習するのに適した値になる。h の値

をいかに設定するかということは、学習する速さを決定する上で非常に重要である。今回 $h = 0.05$ に設定した。

こうして得られた r_3 を用いて転送確率を次のように更新する。ここで、P_f は QueryHit パケットが転送されてきた隣接ノードへの転送確率で P_n はそれ以外の隣接ノードへの転送確率である。

$$\begin{aligned} P_f &= r_3 P_f + 1 - r_3 \\ P_n &= r_3 P_n \end{aligned}$$

QueryHit パケットを転送した隣接ノードの先には応答を返したノードがあり、その隣接ノードに転送する確率を上げ、それ以外の隣接ノードへの転送確率を下げることによって、転送した結果応答が返る確率を高めている。

3.4.3 パラメータの値が持つ効果とその選定理由

本方式で採用するほとんどのパラメータはオリジナルの Antnet とほとんど同じ値に設定した。この理由はそれらのパラメータがロバストであり、その値を変更することによって大きく学習結果が変化するという性質のあるものではないからである。さらに AntNet で十分良い結果が得られていたことも理由に挙げられる。c、a、a' がオリジナルの AntNet と全く同じ値を採用している。パラメータ s は非常に重要度が低いパラメータである。学習の初期段階において、応答時間が非常に長かったあとに、応答時間の短い QueryHit が返ると、あまりに強く学習してしまった。これは、そのようなことがおこらないように、学習の強さに制限を与えるために入れたパラメータである。

AntNet のパラメータはロバストで、値を変化させてもあまり違いは現れないのだが、比較的重要で値の変化の影響が大きいパラメータは、べき乗する係数 h と、閾値 t の 2 つである。

h の値によって学習する速度が決定される。この転送先決定アルゴリズムにおいては、転送先が少数に限定されることが重要である。転送先が少数に限定されるためには、いくつかの隣接ノードに対して転送確率が少なくなる必要がある。しかし h の値が大きいと、あまり良くない転送に対する転送確率も高くなりやすく、なかなか転送先が限定されない。これについていろいろな値を試してみた結果、$h = 0.05$ のあたりがバランスが取れることが分かった。

閾値 t の値は、発見する数と応答にかかる時間とのバランスを考えて、決定されるべき値である。閾値 t の値によって、応答にかかる時間がどれくらい短い

と強く学習し、長い場合はあまり学習しないようにするのかが決定される。閾値 t の値が大きいと、発見できる数が多くなるが、応答にかかる時間が長くなる。さらに閾値 t を大きくすると転送しても応答が返って来ない場合が増える。逆に、閾値 t が小さすぎると、応答数が非常に減ってしまう。応答数が減るとユーザの選択肢は少なくなり、あまり望ましくない。オリジナルの AntNet ではスケールファクタ c が 2 であり、閾値 $t = 0.5$ なので、平均応答時間以下に更新することになる。

本稿では応答にかかる時間を短くするためにやや小さい値を採用し、閾値 $t = 0.45$ を採用した。

第 4 章　実験と評価

3.4 節で提案した手法が Gnutella のように、すべての隣接ノードに Query パケットを転送するような方法 (ブロードキャスト方式) に比べて、ネットワーク負荷、発見できた数、発見にかかった時間といった観点で優れていることを示すために実験を行った。

4.1　実験での目標と評価基準

評価基準にはネットワーク負荷の評価基準として 2 つ、応答結果の良さの評価基準としてさらに 2 つを採用した。

ネットワーク負荷の評価基準

- **滞留数**　ネットワーク内にあるノードのキューで待っているパケットの数の平均。
- **転送回数**　生成された 1 つの Query パケットが転送されることで TTL がゼロになったり同じノードに複数回到達したりして、その Query パケットの子孫がすべて消滅するまでに転送される回数の平均。

応答結果の良さの評価基準

- **応答数**　生成された 1 つの Query パケットに対して返される QueryHit パケットの数。
- **応答時間**　生成された 1 つの Query パケットに対して最初の QueryHit パケットが返されるまでの時間。

これらの指標には、ある程度関連がある。例えば、滞留数が減ると、その結

果、キューでの待ち時間が短くなり、応答時間も短くなることが期待できる。また、転送回数が多くなると、より多くのノードを訪れることになり、その結果応答数も増える。学習により転送先となる隣接ノードを適切に選ぶことによって、転送回数を増やさずに、応答数を増やすことは提案するアルゴリズムの目標の 1 つである。

4.2 実験で想定する環境

シミュレーション実験では、次のような仮定を置いた。

1. 離散時間でシミュレーションを行い、1 つのパケットを 1 つの隣接ノードに転送するのにかかる時間を、1 単位時間とした。これは、あるノードが単位時間あたりに転送できるパケットの数が 1 個であり、転送にかかる時間は転送先となる隣接ノードの数に比例してかかるということである。したがって、あるノードが 1 つのパケットを 4 つの隣接ノードに転送するには 4 単位時間が必要となる。こうした理由は IP ネットワークでユニキャストを繰り返すことによって、隣接ノードに転送するということを仮定したからである。
2. 応答を返す可能性は 100%か 0%のどちらかであるとした。これはファイルを一種類に限定した場合について実験したことになる。
3. パケットがキューで待つ個数に限界があるとした (本方式では 35、ブロードキャスト方式では 100)。キューからあふれたパケットは破棄され、それ以上の転送処理を行わない。これはトラフィックが爆発してしまった場合でも、正常な処理を続けるためである。
4. ファイルを提供するノードの配置はランダムにした。
5. 特に断らない場合、ノード数は 10000 とした。この理由は文献 [7] において、同時に Gnutella のネットワークに接続しているユーザの数は 10000 程度であるという記述に基づく。
6. ノードは 0.01 の確率でファイルを提供するノードであるものとした。ノード数 10000 の場合は約 100 個のノードがファイルを提供していることになる。
7. 接続関係となる隣接ノードはランダムに選ばれる。Gnutella のネットワークは IP 層の上に構築され、それは物理的な接続関係に依存せず、すべてのノードが互いに直接通信できる。この上に構成することを考えると、どのノードを隣接ノードとしてもおかしくなく、これは妥当な仮定である。

実験においては、パケット生成率、TTL、隣接ノード数を変化させながら実験した。パケット生成率とは、各ノードが 1 単位時間に、どれくらいの確率で 1 個の Query パケットを生成するかを示した確率である。

TTL や隣接ノード数を増加させると、転送回数が増加し、Query パケットを生成したときに到達可能となるノード数が増加することになる。そうして、到達するノード数が増加すると、応答が返る数が増えることが予測される。

また、転送処理の際のパラメータである探求確率 P_e は 0.05、隣接ノードの選択を行う回数 N は隣接ノード数の 1.5 倍に設定した。

さらに転送先学習アルゴリズムでの各種パラメータは $c = 2$、$t = 0.45$、$a = 10$、$a' = 9$、$s = 1.0 \times 10^{-20}$、$h = 0.05$ とした。

4.3 シミュレーション結果

この章では、シミュレーションを行った結果について述べる。

まず、4.3.1 節で滞留数の変化について示す。4.3.1 節ではまず時間とともに滞留数がどのように変化するのかについてグラフを示し、次に隣接ノード数、TTL のパラメータを変えたときにパケット生成率の増加とともにどう変化するのかについて示す。

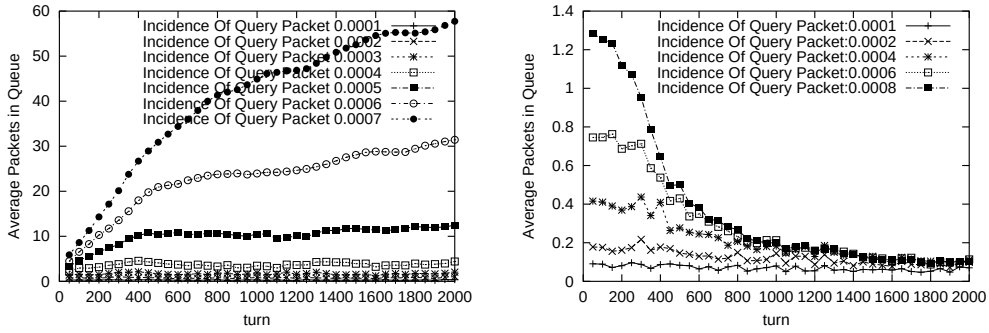
次に 4.3.2 節では、転送回数と応答数の関係について示す。4.3.3 節では転送回数と応答時間の関係について示す。4.3.4 節ではノード数を変化させたときの滞留数の変化、転送回数と応答数の関係、転送回数と応答時間の関係について示す。

4.3.1 滞留数の変化

図 1 はノード数は 10000、隣接ノード数は 4、TTL は 7 として、時間によって、滞留数がどのように変化するかについてシミュレーションした結果を示している。図 1(a) は、ブロードキャスト方式の場合に、図 1(b) では、本方式の場合にどのように変化したのかを示している。両者ともパケット生成率は 0.0001 から 0.001 の間で変化させたものをプロットしている。

この図の横軸 (turn) は、シミュレーション開始からの単位時間を、縦軸は滞留数 (Sum Of Packets In Queue) を示している。滞留数は少ないほどネットワークに対する負荷の観点から望ましい。プロットするときには、全体の傾向を見るために、100 単位時間の間の平均をとっている。

両者は一見して、傾向がまったく異なるグラフである。ブロードキャスト方



(a) ブロードキャスト方式　　(b) 本方式

図 1: 滞留数の変化

式では、パケット生成率が高くなるにつれ、滞留数も次第に増加する。これはパケットの処理能力よりもパケットが増える速度が速く、処理されずにキュー内に残ってしまうからである。

一方、本方式では、パケット生成率が高い場合でも滞留数は次第に減っていっている。シミュレーションを始めたばかりの初期の状態では、学習が進んでおらず、ほとんとランダムに転送してしまうため、かなり多くのパケットがキューで滞留する。しかし、時間が経つとともに学習が進み、パケットの転送先が制限されていくため、滞留数は時間とともに減少していく。

また、本方式では、パケット生成率が高い場合の方が単位時間あたりの応答数も多いため、学習の進む速度が速く、より急速に滞留数が少なくなるという現象も起きていることが分かる。

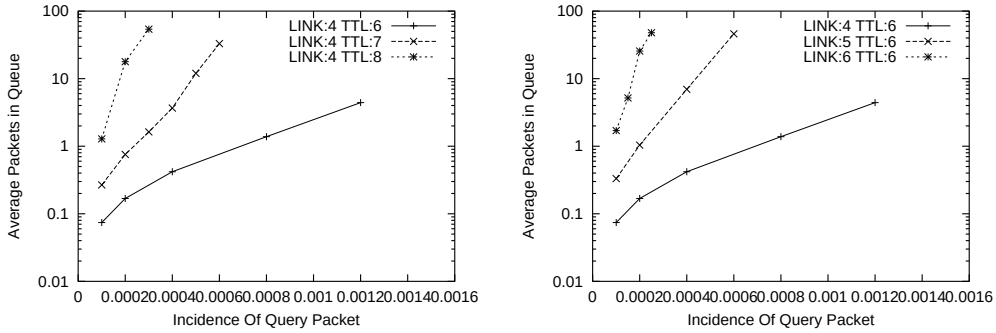
4.3.1.1 TTL、隣接ノード数の影響

上記の実験では、TTL、隣接ノード数は一定であった。次に、これらのパラメータを変化させたときに、どのように滞留数が変化するのかについて示す。

まず、図 2 はブロードキャスト方式で行った場合の滞留数の変化について示している。図 2(a) は TTL を変化させた場合、図 2(b) のグラフは隣接ノード数を変化させた場合である。

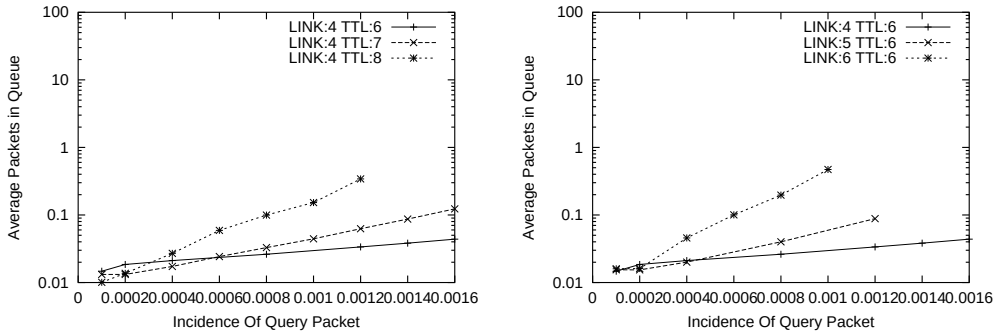
図 3 は、本方式で行った場合の滞留数の変化について示したものである。図 2 と同様に図 3(a) は TTL を変化させた場合、図 3(b) は隣接ノード数を変化させた場合である。横軸は Query パケットの生成率 (Incidence Of Query Packet)、縦軸は滞留数 (Average of Packets In Queue) とした。

これらを比べると、本方式の方が、滞留数が大きく減っていることが分かる。



(a) 滞留数の変化 (TTL)　　(b) 滞留数の変化 (隣接ノード数)

図 2: ブロードキャスト方式での滞留数の変化



(a) 滞留数の変化 (TTL)　　(b) 滞留数の変化 (隣接ノード数)

図 3: 本方式での滞留数の変化

さらにパケット生成率を大きくしたときの滞留数の増加率がブロードキャスト方式の場合はかなり急激であるのに対し、本方式ではゆるやかであることが分かる。

4.3.2　転送回数と応答数

ここでは、転送回数と応答数との間にどのような相関関係があるのかについて述べる。

多くのノードに対して応答してもらうようにするには、より多くのノードに Query パケットを転送することが必要になる。転送回数を多くすれば、多くのノードに Query パケットを到達させることができる。しかし、転送回数を多くすることは、ネットワーク負荷を重くことにつながるため望ましくない。

本方式を使用した場合、応答を返すノードに転送する確率を増やすことによって、転送回数が同程度であっても、多く応答が得られていることを示す。

TTL や、隣接ノード数を変えることによって転送回数を変化させてシミュ

レーションを行った結果を図 4 に示す。横軸が転送回数 (Network Load)、縦軸が応答数 (The Number Of Discovery) である。

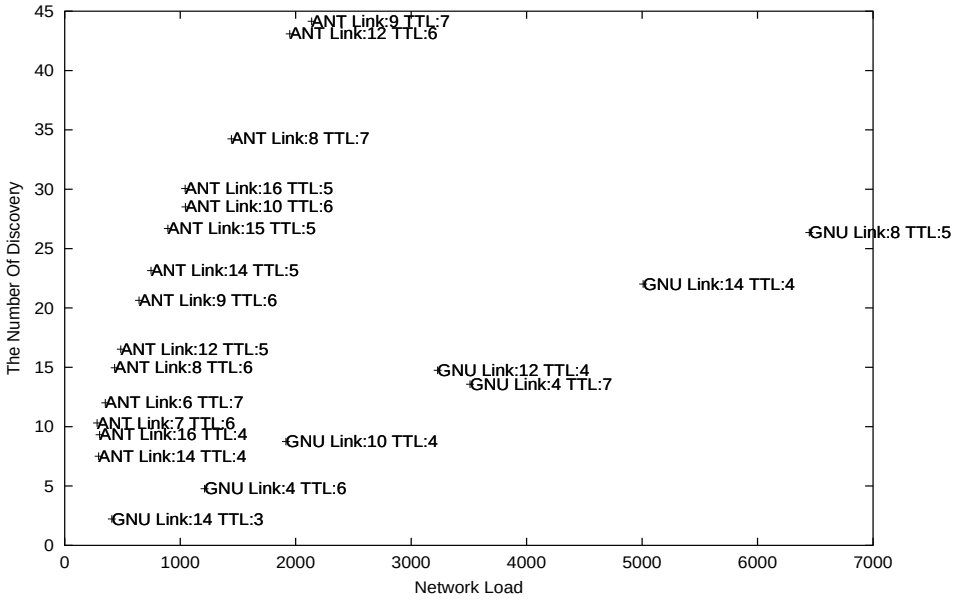


図 4: 転送回数と応答数の関係

本方式によるものには ANT と印字し、ブロードキャスト方式によるものには GNU と印字している。LINK に続く値は隣接ノード数、TTL に続く値は TTL である。

このグラフを見ると、まず大まかな傾向として、転送回数が増えるにつれて、その結果、返る応答数も多くなっていることが確認できる。

このとき、本方式の方が傾きが急峻であり、少ない転送回数で多くの応答を返すことに成功していることが分かる。

4.3.3 平均応答時間の比較

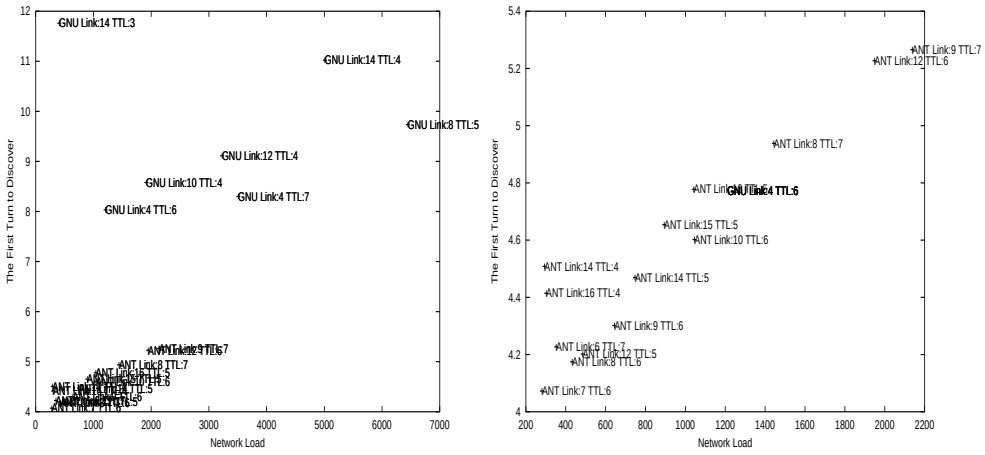
前節では、ネットワーク負荷と応答数との間で比較を行ったが、この節では、応答までにどれくらい時間がかかるかについて述べる。

ここでは前節で使用したのと同じ TTL、隣接ノード数を使用した。図 5 で横軸は転送回数 (Network Load)、縦軸は応答時間 (First Turn to Discover) である。図 5(b) は図 5(a) の中で、左下の部分を拡大したものである。

応答時間についても、圧倒的に本方式の方が優れている。左下に集まっているのが、本方式で実験した結果の応答時間であり、5, 6 単位時間で応答があるこ

とが分かる。

おおまかな傾向として転送回数が増えると、応答時間が遅くなるという傾向があるものの、その傾向はあまり強くないことが分かる。転送回数が同程度である場合、隣接ノード数を減らして、TTL を長くした方が応答時間は短くなるという結果が得られた。隣接ノード数を多くすると転送処理のために待たなければならない時間が長くなるため、応答時間は長くなるのだろうと考えられる。



(a) 平均応答時間の比較　　(b) 本方式での平均応答時間

図 5: 転送回数と平均応答時間

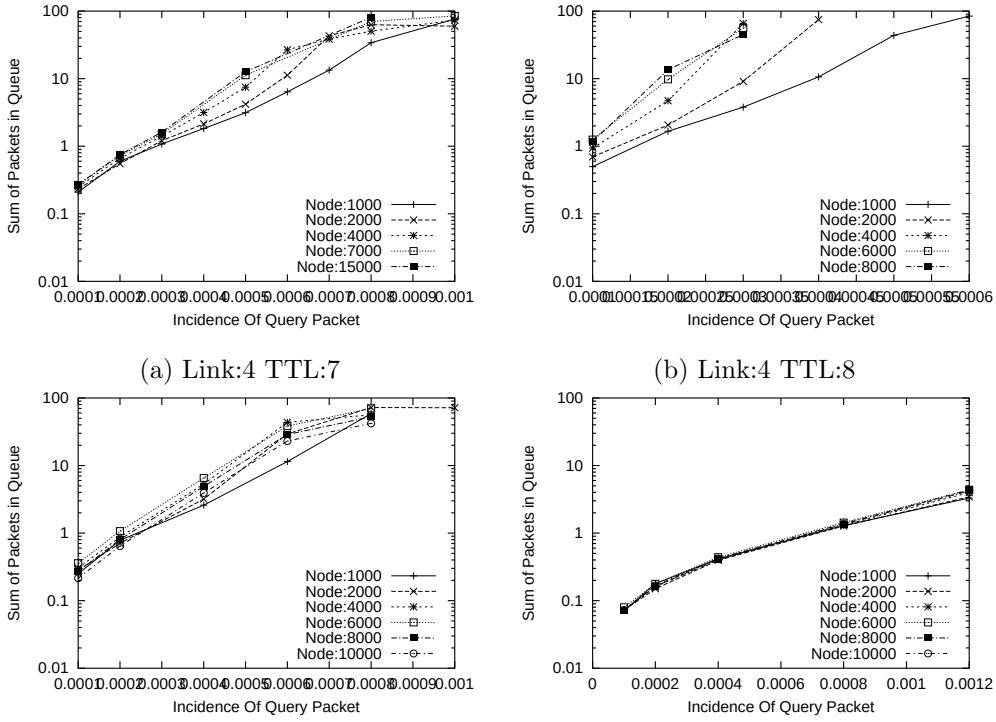
4.3.4　ノード数が与える影響

ここまでの実験ではノード数を 10000 に設定して、実験を行った。本節では、ノード数の変化によってどのようにネットワーク負荷に影響がでるのかについて調べる。ノード数が変化したときに滞留数、応答数、応答時間がどのように変化するかについてシミュレーションを行いその結果を示す。

4.3.4.1　パケット生成率と滞留数

まず滞留数の変化について述べる。同じパケット生成率でシミュレーションを行った。パケット生成率を横軸 (Incidence Of Query Packet)、滞留数を縦軸 (Average Packets in Queue) にし、縦軸は対数目盛りとした。1 つのグラフにさまざまなノード数に対して掲載している。TTL、隣接ノード数を変えて実験を試みた。図 6 はブロードキャスト方式で行った結果を示している。

この図はパケット生成率が上昇するとともに指数関数的にキューにパケットが溜まっていることが分かる。また、ノードが増えてもノード一個あたりのパ



(a) Link:4 TTL:7

(b) Link:4 TTL:8

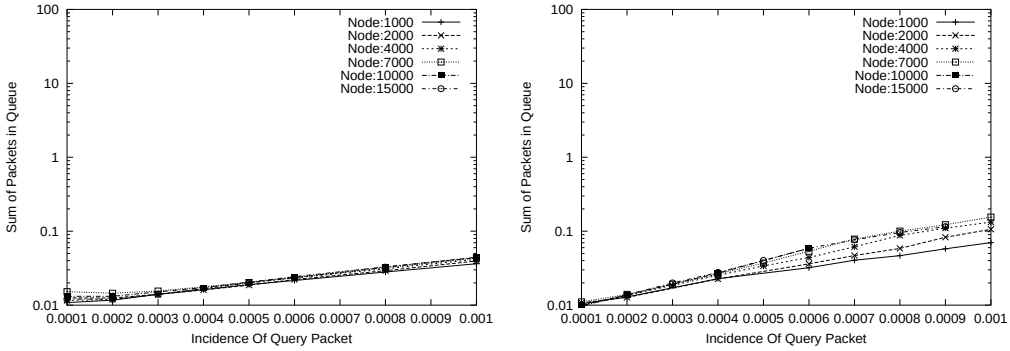
(c) Link:5 TTL:6

(d) Link:4 TTL:6

図 6: 滞留数の変化 (ブロードキャスト方式)

ケット数はほとんど変わらないことが分かった。

次に本方式でシミュレーションを行ったときの、滞留数の変化について図 7 に示す。本方式で行った場合の方が平均滞留数は少ない値になっていることが



(a) Link:4 TTL:7

(b) Link:4 TTL:8

図 7: ノード数の変化 (本方式))

分かる。ノード数による影響は本方式の場合もなく、ノード数の大小に関わらずほとんど同じ値となった。本方式にすることで、100 分の 1 程度に滞留数が

減っていることが分かる。

4.3.4.2　転送回数と応答数

転送回数を横軸、応答数を縦軸にして、シミュレーションを行った結果を示す。図 8(a) はノード数を 1000、4000、15000 に設定したときの転送回数と応答数の関係を示している。TTL および隣接ノード数は 4.3.2 節で行った実験と同じ値を使用した。図 8(a) において本方式の点群は ANT、ブロードキャスト方式の点群には GNU と印字している。

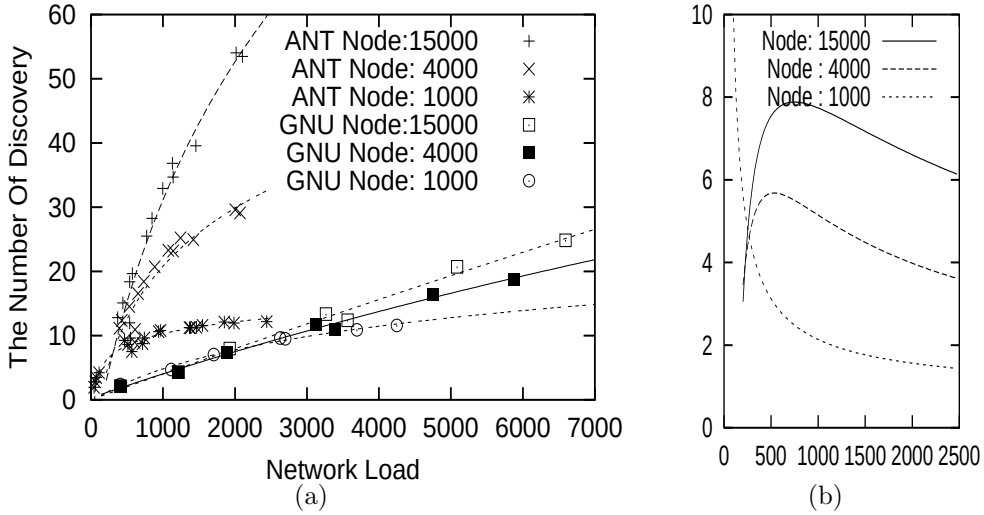


図 8: ノード数を変化させたときの転送回数と応答数

図 8(a) を見ると、本方式で行った場合は、ノード数が増えれば増えるほど、多くの応答が返ることが分かる。

それに対して、ブロードキャスト方式の場合は、ノード数が増えても、返る応答数は微増にとどまっている。ノード数が多くなるほど、本方式の優位が強まることが分かる。

さらに、ノード数が少ないときでも、常にブロードキャスト方式よりも少ない転送回数で同じ応答数が返るようにできていることが分かる。ノード数が少ないときでも本方式で行う方が有利である。

図**??**は、ネットワーク負荷を変えて得られた本方式で発見した数とブロードキャスト方式で発見した数の比を示している。これを見るとノード数が 15000 のとき、転送回数が 400 から 2500 のとき同一のネットワーク負荷で 6 倍多く発見できていることが分かる。

4.3.4.3 応答時間

ネットワーク負荷を横軸、応答時間を縦軸にして、ノード数を様々に設定したネットワーク構造について実験したのが図 9 である。

図 8(a) において本方式の点群は ANT、ブロードキャスト方式の点群には GNU と印字している。

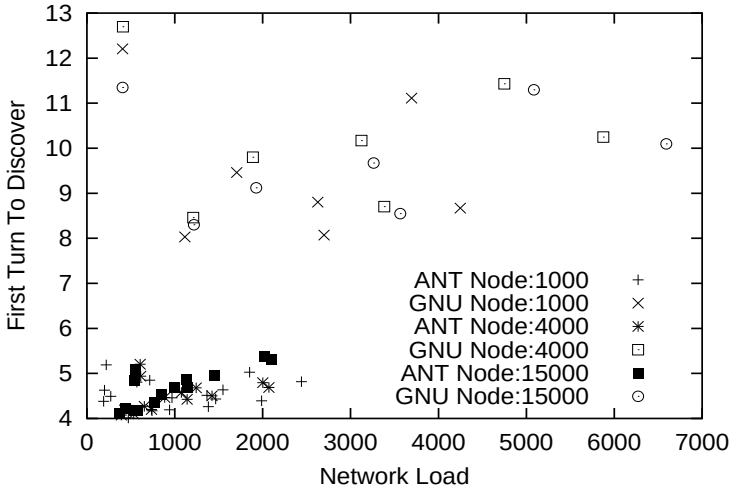


図 9: ノード数を変化させたときの転送回数と応答時間

図 9 を見ると、本方式で行った方が全体として、素早く応答が返ってきていることが分かる。両方式ともにノード数の影響については、ほとんど見られず、ノード数の大小が最初の応答が返るまでの時間に影響を与えるようなことはないことが分かる。

第 5 章　結論

本報告書では、Gnutella のような純粋型ピアツーピア環境において少ないネットワーク負荷で多くのファイルの発見を実現する転送先学習アルゴリズムを提案した。本方式は、自律的であり、ノードに必要とされるメモリも少なく、情報交換のために多くのパケットを必要とすることもない。シミュレーションを行い、すべての隣接ノードに転送していた従来手法と比較して、良好な結果が得られた。

本手法はパケット生成率の上昇に強く、従来手法ではキューが一杯になって

しまうほどパケット数が増えてしまうような状況であっても、本手法ではパケット数の増加が抑えられているとこが分かった。また、同等の転送回数で 6 倍の応答が得られるような転送を行うことが可能であり、応答が返るのにかかる時間も従来手法と比較して短い時間で可能であった。

今後の課題としては、ノードの応答確率の違いや、位置的または時間的に突発的に多く Query パケットを生成された場合について調べる必要がある。さらに、今回提案した手法で用いた学習アルゴリズムと、他の学習アルゴリズムとの比較も行う必要がある。

謝辞

本研究を行うにあたり、ご懇篤なご指導とご高配を賜わりました美濃導彦教授、中村素典助教授、岡部寿男助教授、藤川賢治助手に厚くお礼申し上げます。

参考文献

[1] *Gnutella Bandwidth Usage*
http://resnet.utexas.edu/trouble/p2p-gnutella.html

[2] *Gnutella Protocol Specification v0.4*
http://www.clip2.com/GnutellaProtocol04.pdf

[3] Eytan Adar and Bernardo A. Huberman
Free Riding on Gnutella (2000)
http://www.hpl.hp.com/shl/papers/gnutella/Gnutella.pdf

[4] Jordan Ritter
Why Gnutella Can't Scale. No, Really. (2000)
http://www-2.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html

[5] Anthony J. Howe University of Victoria
Napster and Gnutella: a Comparison of two Popular Peer-to-Peer Protocols (2000)
http://www.csc.uvic.ca/ ahowe/pdf/napstergnutella.pdf

[6] Gianni Di Caro, Marco Dorigo
AntNet: A Mobile Agents Approach to Adaptive Routing (1997)
http://aepia.dsic.upv.es/revista/numeros/12/baran.pdf

[7] Kunwadee Sripanidkulchai
The popularity of Gnutella queries and its implications on scalability
(2001)
http://www-2.cs.cmu.edu/ kunwadee/research/p2p/gnutella.html

[8] Andy Oram
Peer-to-Peer : Harnessing the Power of Disruptive Technologies
(O'REILLY, 2001) pp.94-122

[9] 伊藤直樹
『P2P コンピューティング 技術解説とアプリケーション』
(ソフトリサーチセンタ、2001)pp.18-42