第4章　アクティブネットワーク技術を利用したキャッシュ方式

本章では，アクティブネットワーク技術を用いて途中経路上の機器にキャッシュ機構を導入するintermediate cachingの手法を提案する．また，この手法を第3章で述べた多重名前空間システムに適用した．本章は，以下に述べる構成になっている．まず，第4.1節では，intermediate cachingの方式説明を行う．次に，第4.2節では，キャッシュ機構などを一切持たず名前解決のセマンティクスをそのままプログラムした多重名前空間システムの設計と実現について述べる．次に，第4.3節では，第4.2節で述べた多重名前空間システムに対するintermediate cachingの実現の詳細，そして，第4.4節では実装，第4.5節で性能評価のための実験とその結果について述べ，第4.6節でまとめる．

4.1　Intermediate Caching

提案方式は，通信経路上の機器を用いて名前探索のキャッシュ機構を導入するものである．

広域ネットワーク上に散在するサーバとクライアントの間の通信においては，直接的に通信が出来る場合は稀で，一般にルータなどを介してパケットが受け渡されることで通信が行われる．普通はルータにおいては，ルータ自身に送られてきたパケットを次に受け渡すべきルータに対して送信する処理のみが行われる．しかし，これらの機器に何らかの処理を行わせることで，以下に述べるような利点が生まれる．

第一に，これらの処理は端点のサーバ・クライアントに透明にすることが可能であり，従来から存在する通信プロトコルを経路上の機器に改良を加えることで容易に拡張することができる．第二に，経路上の機器で処理を完了することが出来た場合，それ以上先にパケットが伝播することが無くなるため，パケット数を減少させ，反応時間を短縮させることが可能となる．

提案方式の構成を 図4.1に示す．クライアントが送信した探索要求のパケットは，まずそのクライアントと同じマシンに存在するアクティブノード(active node)に送信

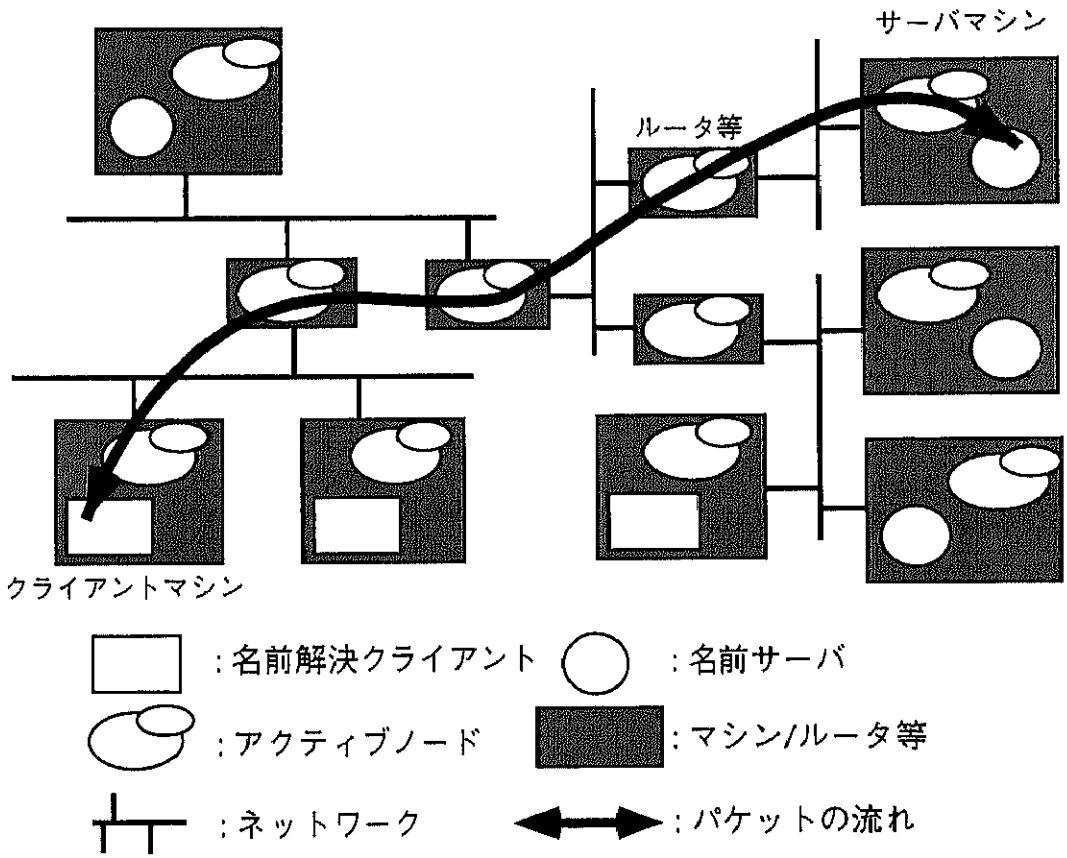

サーバマシン
ルータ等
クライアントマシン
：名前解決クライアント
：名前サーバ
：アクティブノード
：マシン/ルータ等
：ネットワーク
：パケットの流れ

図 4.1: 提案方式の構成

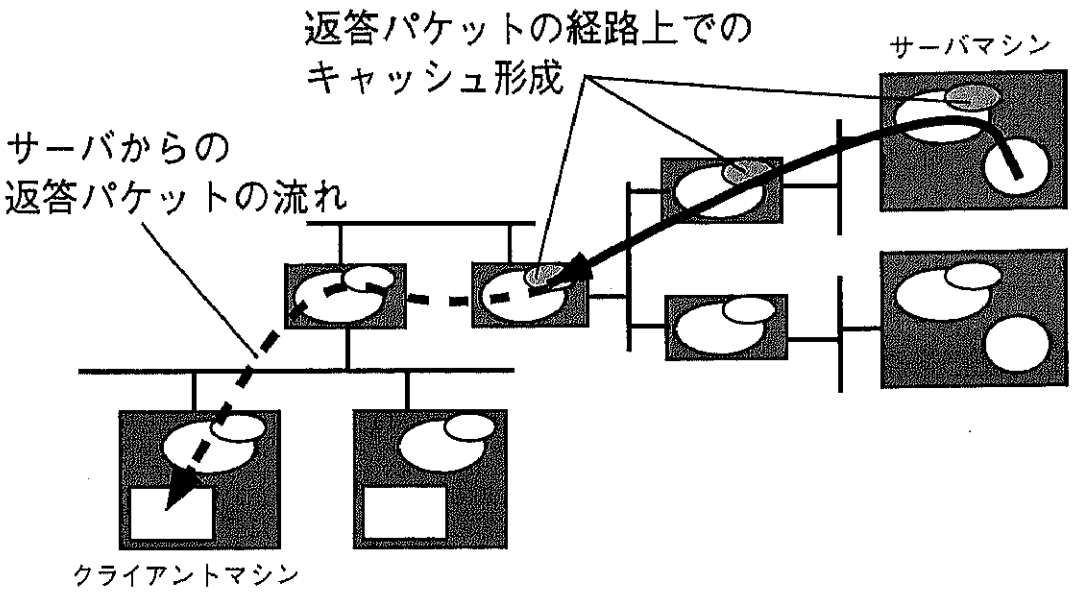


図 4.2: キャッシュの形成

される．アクティブノードとは，各マシンに１つずつ存在する仮想的な通信デバイスであり，通過するパケットに対して以下の処理を行う．アクティブノードは，必要ならばそのパケットの内部を解析したのち，目的のサイトへの経路を決定し，次のアクティブノードへ送信する．目的のマシンに到達すると，そのマシンのアクティブノードはマシン内のローカルな通信でサーバにパケットを配送する．サーバからクライアントへパケットが送られる際も同じ手順で送信される．これらの配送処理は，サーバとクライアントには透明なものとなっており，名前空間探索のキャッシュの処理は，このアクティブノードにおいて行われる．

アクティブノードにおける単純なキャッシュ構成方式は以下のようになる．クライアントとサーバの間で交換されるパケットがアクティブノードを通過する際にアクティブノードはその内容を解析し，サーバからの返答パケットであればキャッシュに情報を追加することでキャッシュを形成する (図 4.2)．例えば，名前の探索要求に対する返答パケットから「ある名前がある名前空間に存在する」という情報を得てキャッシュに記録する．そして，クライアントからの探索要求パケットに対し，各アクティブノードは自分が持つキャッシュと探索要求を照合する．もし適切な返答が可能な場合は，そのアクティブノードがあたかもサーバであるような振る舞いをしてクライアントに返答する (図 4.3)．このとき，探索要求パケットはサーバに転送されないため，パケットのトラフィックの減少と，クライアントの要求に対する反応時間の短縮が可能である．

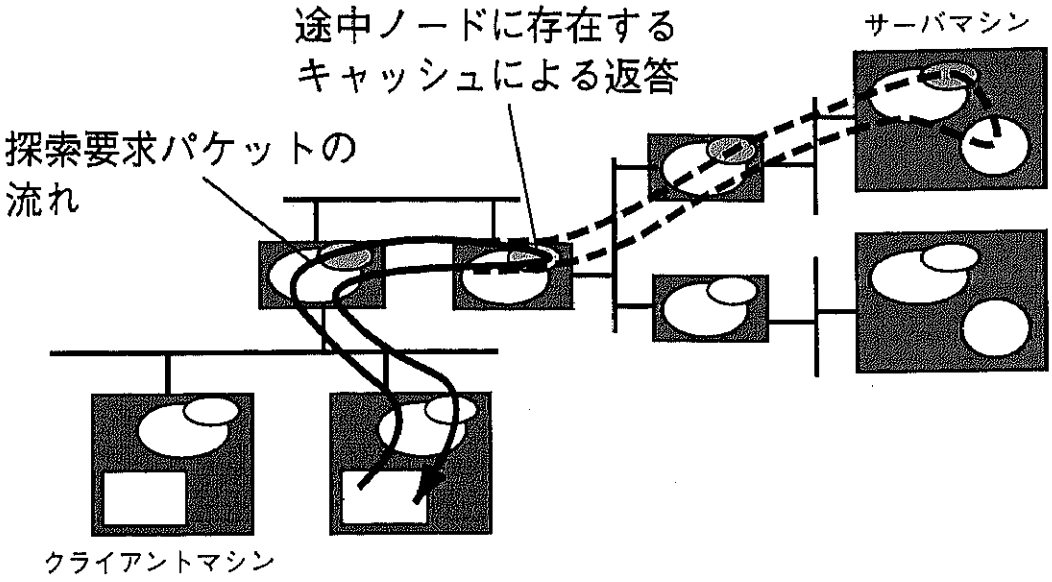


図 4.3: キャッシュの利用

しかし，この単純なキャッシュ構成方式には2つの欠点が存在する．第一は，一貫性制御に多大なコストがかかるということである．第二は，キャッシュ機構によりサーバに近いノードへ要求パケットが到達しなくなることで，そのノードのキャッシュエントリ置換アルゴリズムは，名前のアクセス頻度を検出することができなくなる点である．

まず，第一の問題について述べる．この単純なキャッシュ構成方式では，クライアントとサーバの間にある全てのアクティブノードがキャッシュを保持することになる．経路上の全てのアクティブノードがキャッシングを行うと，名前空間が更新された際にその全てのアクティブノードに対してキャッシュの無効化を行わなければならず，その際に交換されるパケットの数や，キャッシュの存在位置を記憶するためにかかるコストが増大する．これを解消するために，名前の更新頻度をパラメータとして，キャッシュするアクティブノードを制限する機能を導入する．これにより，更新頻度が高い名前に関してはキャッシュするアクティブノードを最小限にし名前の更新コストを下げ，更新頻度が低い名前に関しては最大限にキャッシュし，レスポンスの向上を図ることが可能となる．

提案方式では，サーバおよびクライアントには透明にキャッシュ機構を実現することを目指している．そのため，サーバおよびクライアントから更新頻度の情報を得ることは，透明性を損なうことになるので採用することはできない．提案方式では，情報をキャッシュに記録するノードを，キャッシュによって返答をした次のノードに限る．

この方式により，情報の伝播を最低限に押さえることができ，かつアクティブノードでキャッシュがヒットするごとに隣のノードに伝播して行くようにできる．すなわち，更新の頻度が高く，すぐに無効化される名前の伝播は最小限になり，無効化されない名前は次第に隣のノードに伝播してゆくためレスポンスを向上させることが可能となる．このキャッシュの伝播を我々はインクリメンタルなキャッシュの伝播と呼ぶ．

次に，第二の問題について述べる．提案方式では，キャッシュエントリにキャッシュがどこに伝播していったかを記録しておき，無効化メッセージがその名前のキャッシュを持つ全てのノードに到達するようにしている．このことにより，あるノードでキャッシュが無効化される際，その下流ノード全てのキャッシュも同様に無効化される．これは，名前に関する情報のアップデートに伴うもののときには，この無効化は無害である．なぜなら，この無効化は必須のものであるからである．しかし，無効化はキャッシュのオーバーフローによっても引き起こされる．キャッシュがクライアントの近くまで伝播するほど，サーバに近いノードのキャッシュへのアクセスは減少する．そして，そのノードにおいてキャッシュのオーバーフローが起きたとき，何らかのキャッシュ置換アルゴリズムによってアクセスの少ないキャッシュエントリは消去される候補になる可能性が高い．そして，そのエントリが消去されると，その下流にあるノードの対応するキャッシュも同様に無効化される．これは，提案方式の性能を低下させる恐れがある．

提案方式では，ノード間でキャッシュの使用状況について情報を交換させている．アクティブノードは，ある一定回数キャッシュエントリにアクセスが生じた場合，その情報を上流のノードに伝播させる．このメッセージはクライアントのメッセージと同様の形態をしているが，アクティブノードはそれを受け取ると，キャッシュのアクセスカウンタをインクリメントするのみで，さらにそれを上流へと転送することはしない．これにより，上流ノードでのキャッシュ置換アルゴリズムに対し，適切なキャッシュ使用状況を知らせることが可能となっている．

さらに，このメッセージはノードや経路の障害を検出するためにも使用する．上流のノードが故障していた場合，無効化のメッセージが届かなくなるため，下流ノードにある情報は，たとえ名前サーバで名前の更新が起こったとしても無効化されることなくノードに残ってしまう．これを避けるため，提案方式ではこのキャッシュ使用状況を調べるパケットをもちいて上流サイトとの接続性を検査する．もし，キャッシュ使用状況を調べるパケットに対する返答が上流ノードから返されなかった場合，上流ノードは故障していると考え，その上流ノードから伝播してきた情報を無効化する．

表 4.1: クライアントからの操作要求

メッセージの種類	用途/返却値
ADDNAME	名前の追加
	OK, LINK, ERROR
ADDLINK	リモートリンクの追加
	OK, LINK, ERROR
LOOKUP	名前の探索
	OK, LINK, NOTFOUND
DELNAME	名前・リモートリンクの削除
	OK, LINK, ERROR

4.2 多重名前空間システムの設計と実現

多重名前空間を，その名前解決のセマンティクスから単純に実現した名前解決システムについて以下で述べる．

クライアントからの要求

クライアントの要求するサービスに対応して，サーバに対して送るメッセージは表 4.1 に示すものになる．ここで，返却値 OK は要求に対する動作が行われたことを示し，返却値 LINK は，その名前がリモートリンク先にあることを示す．また返却値 NOTFOUND は探索した名前が存在しないことを示し，返却値 ERROR は何らかのエラーが発生したこと示す (既に存在する名前を使って名前の追加要求を行った場合など)．以下，返却値を含んだパケットを返答パケットと呼ぶ．

クライアントは，表 4.1 に示されるメッセージを使用し，サーバへ要求を行う．この際，サーバの位置情報に関してクライアントは名前空間のルートサーバの位置情報のみを知っており，他のサーバに関する情報は，問い合わせの際にリモートリンク先としてサーバの位置情報を得る．サーバから返却値 LINK が返された場合はそのリモートリンク先に要求メッセージを送ることを繰り返す．そして，最終的に返却値 OK, NOTFOUND, ERROR のいずれかを得て，要求を完了する．

サーバの動作

クライアントからの要求に対するサーバの動作を 図4.4 に示す．この図から判る通り，純粋に名前解決のセマンティクスに従った処理のみが実行される．

動作例

図3.4 に示される単一名前空間内で，"/b/g" を探索した場合の通信内容について述べる．まずクライアントは，名前サーバ1へ"/b/g" の探索要求メッセージ LOOKUP を送信する．これに対し，名前サーバ1は"/b" 以下のファイルに関しては名前サーバ3を参照せよとの情報を持った返却値 LINK をクライアントに返す．そしてクライアントは改めて名前サーバ3に"/g" の探索要求メッセージ LOOKUP を送信する．最後に名前サーバ3は，この名前が存在することを示す返却値 OK とその物理資源名を返す．

多重名前空間に対する操作は，ここまでに述べた単一名前空間に対する操作をビューパスに指定された空間に対し順次実行することで行う．名前の追加は，ビューパスの最上位の空間に対して行うので，ビューパスの最上位の空間に対してのみ上記の名前追加の操作を行う．名前の探索は，ビューパスの最上位の名前空間から順に各名前空間内で探索を行い，最初に名前が見つかった空間で探索を中断し，その空間での論理資源名に対する物理資源名をユーザが求めるものとして返答する．また，名前の削除は，名前の探索と同様にビューパスの最上位の名前空間から順に各名前空間内で削除を行い，返却値 OK(あるいは ERROR) が返されるまで名前の削除を繰り返すことによって行う．

4.3 多重名前空間システムへの Intermediate Caching の適用法の詳細

以下では，第4.1節で述べたインクリメンタルなキャッシュ伝播を実現する intermediate caching 方式について述べる．

キャッシュの内容

アクティブノード内のキャッシュは，以下の組をキーとして検索する．

- 名前空間名

```
switch (メッセージの種類) {
case ADDNAME:
case ADDLINK:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    ERROR();
  } else {/* 名前が存在しない */
    名前/リンクを登録;
    OK();
  }
  break;
case LOOKUP:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    OK(物理資源名);
  } else {/* 名前が存在しない */
    NOTFOUND();
  }
  break;
case DELNAME:
  if (リモートリンク先で管理) {
    LINK(リンク先サーバ, リンク先で問い合わせる名前);
  } else if (名前が存在) {
    名前を削除;
    OK();
  } else {/* 名前が存在しない */
    ERROR();
  }
}
```

図 4.4: サーバ側の処理．LINK() 等の表記は，その返却値を括弧内の値と共にクライアントに送信するという処理を示す

- 論理資源名

そして各キャッシュは，以下のデータを持つ．

- キャッシュの伝播先
- 参照回数カウンタ
- 物理資源名 (論理資源名が存在する場合)
- リンク先のサーバ名 (論理資源名がリモートリンクの場合)
- 存在しないという情報自体 (論理資源名が存在しない場合)

名前が存在しないという情報をキャッシュする利点は，「ある論理資源名が存在しない」という情報を使うことで，無駄な名前空間の探索をスキップさせることが可能となる点である．これは，DNS におけるネガティブキャッシュ(DNS NCACHE)[3] にヒントを得た手法である．

4.3.1 キャッシュの形成と利用

キャッシュがクライアントの要求に対してインクリメンタルにキャッシュが伝播してゆくために，アクティブノードが無制限にキャッシュ処理を行わないよう以下の処理を行う．

サーバから直接返答パケットを受け取ったアクティブノードは，その内容を解析しキャッシュに情報を加える．ここで，この情報が relay 先のアクティブノードより先でキャッシングされることを抑止するため，パケットに「Delegation Counter(DC)」を付加する．DC は整数 $N_{prop}-1$ を初期値として保持している．DC が付加されたパケットを受け取ったアクティブノードは，DC を調べ，その値が 1 以上の時は通常のキャッシュ処理を行い，DC を 1 減少させた後次のノードに relay する．DC が 1 未満の場合はそのパケットに対するキャッシュ処理を行わず，単に次のアクティブノードに relay する処理のみを行う．この処理により，サーバ・クライアント間の通信経路上のアクティブノードのうち，最初はサーバに近い N_{prop} 個のアクティブノードにのみキャッシュが存在することになる．

クライアントからの探索要求がアクティブノードに到着した際，その要求がキャッシュを用いて解決できる場合，キャッシュの参照回数が N_{acc} 以上であるとき，アクティブノードは返答パケットに N_{prop} を値として持つ DC を付加して返答する．そして，relay 先のアクティブノードで情報がキャッシュされたことを記憶するため，各キャッ

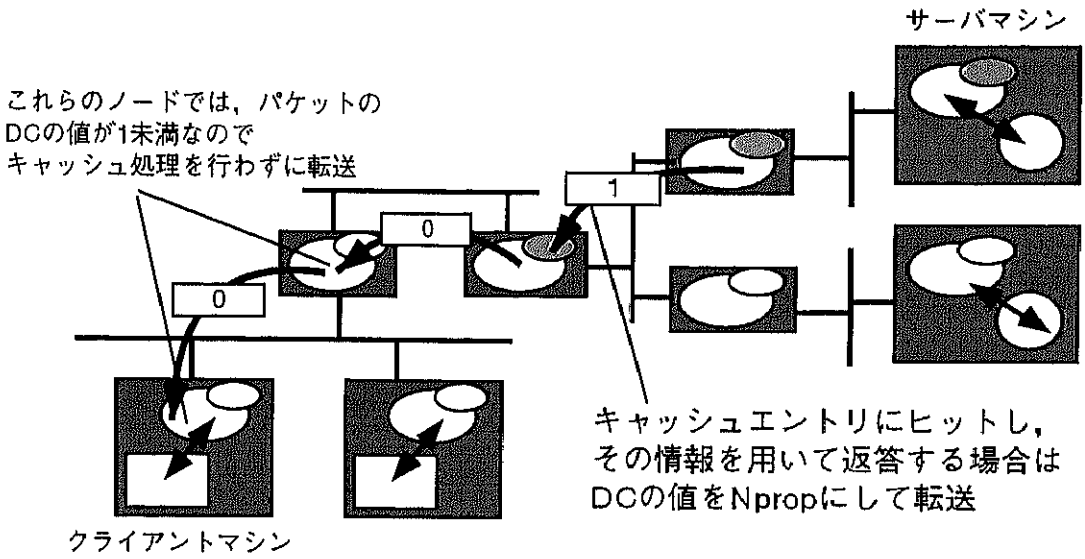


図 4.5: Delegation Counter の付加によるキャッシュ伝播の制限

シュエントリに relay 先のアクティブノードを記録する．参照回数が N_{acc} 未満の時は，DC を 0 として返答する．

図 4.5 に，DC の付加によってキャッシュの伝播が制限される例を示す．

4.3.2 キャッシュの無効化

サーバに名前が登録されスキップしていた名前空間に名前が出現した場合などは，名前解決の整合性を保つためにこのキャッシュを無効化する必要がある．アクティブノードでは，メッセージ ADDNAME，ADDLINK，DELNAME に対するサーバからの返答値 OK を監視することで名前空間の更新を検出する．

名前空間の更新を検出したアクティブノードは，更新された情報をキャッシュしているアクティブノードに，該当するキャッシュの無効化を指示するメッセージ INVALIDATE を送信する．この時，送信すべきアクティブノードを決定するために，前述したキャッシュに記録したキャッシュの伝播先を使用する．キャッシュの伝播先は，パケットを relay した先のアクティブノードのみ記録すれば充分であるため，この情報の保持によってキャッシュのデータ量が爆発的に増えるということはない．

例えば，メッセージ ADDNAME によって名前が追加された場合，アクティブノードはその返却値 OK を検知し，クライアントへの返答パケットのほかに，キャッシュエントリに記録された情報の伝播先に対して，無効化すべき名前空間名と論理資源名

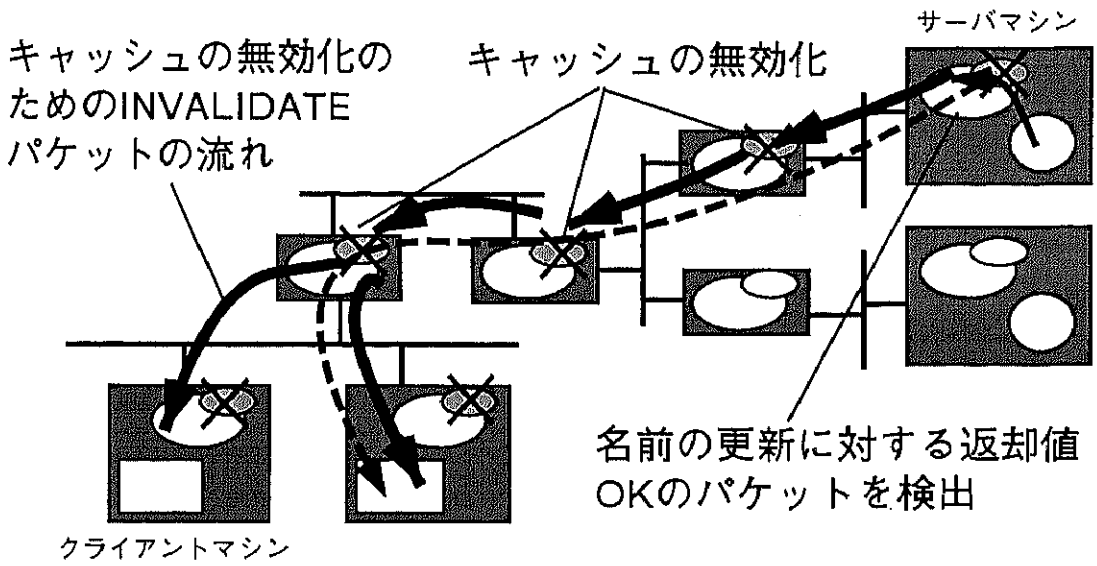


図 4.6: キャッシュの無効化

の組を含んだキャッシュの無効化メッセージ INVALIDATE を送信する (図 4.6). 無効化メッセージ INVALIDATE を受け取ったアクティブノードは，該当するキャッシュを無効化するとともに，そのノードからキャッシュが伝播した先に同様に無効化メッセージ INVALIDATE を送信する．このようにアクティブノード間で無効化メッセージが伝播することでキャッシュの無効化が実現される．

ここまでに述べたアクティブノード内でのキャッシュ処理のアルゴリズムをまとめたものを図 4.7 に示す．

4.3.3 キャッシュのオーバーフロー

アクティブノード内のキャッシュが一定量を超えオーバーフローを起こすなどの理由によって，キャッシュエントリの一つを消去する必要が生じる場合がある．キャッシュエントリ内には，キャッシュの無効化パケットを伝播させるために必要な情報が含まれているため，単純にエントリを消去するだけでは不十分である．そこで，消去すべきエントリが無効化されたものとして，上で述べたキャッシュの無効化の伝播処理を行った後，消去すべきエントリを消去する．

```
if (Delegation Counter が付加されていない)
  DC = Nprop;
if (DC < 1) {
  次のアクティブノードに relay;
} else {
  switch (パケットの種類) {
  case LOOKUP:
    キャッシュを検索
    if (マッチするキャッシュエントリが存在) {
      返答パケットの作成;
      if (参照数 >= Nacc)
        DC = Nprop;
      else
        DC = 0;
      クライアントに返答;
      送り先のアクティブノードを
        キャッシュエントリに記録;
      参照数をインクリメント;
    } else {
      次のアクティブノードに relay;
    }
    break;
  case ADDNAME:
  case ADDLINK:
  case DELNAME:
    次のアクティブノードに relay;
    break;
  case 返答パケット:
    キャッシュを検索
    if (マッチするキャッシュエントリが存在)
      if (受信したデータと矛盾) {
        キャッシュの伝播先に
          INVALIDATE パケットを送信;
        キャッシュの削除;
      }
    } else {
      受信した情報をキャッシュに記録;
    }
    DC = DC - 1;
    次のアクティブノードに relay;
    break;
  case INVALIDATE:
    if (マッチするキャッシュエントリが存在)
      キャッシュの伝播先に
        INVALIDATE パケットを送信;
      キャッシュの削除;
    break;

  }
}
```

図 4.7: インクリメンタルなキャッシュ構成法におけるアクティブノードでの処理

4.3.4 キャッシュアクセス情報の伝播

キャッシュエントリが N_{info} 回アクセスされると，そのアクティブノードはそのメッセージを CACHEINFO メッセージとして上流のアクティブノードに転送する．CACHEINFO メッセージを受信したノードは，その名前のアクセスカウンタをインクリメントし，送信元のノードに返答する．CACHEINFO を送ったノードがこの返答を受信できなかった場合，その上流ノードへの経路もしくは上流ノード自体が故障していると判断し，そのキャッシュエントリの無効化を行う．

4.4 実装

本節では，前節までに述べた intermediate caching 方式のプロトタイプシステムの実装の詳細についてのべる．

本プロトタイプシステムは，Active Network Encapusulation Protocol(ANEP)[2] をネットワーク層のプロトコルとして使用している．ANEP は，PLANet や ANTS などのアクティブネットワーク・システムにおいて共通に使用されるプロトコルであり，IP[25] や IPv6[8] やリンク層 (Ethernet 等) の上にアクティブネットワークのパケットを送信可能とするためのプロトコルである．ANEP では，パケットのフォーマットのみを規定しており，そのパケットを受信した機器がどのような操作を行うかの規定はされていない．たとえば，パケットの送信先を示すフォーマットは示されているが，そのパケットを送信先に転送すべきかは一切規定されていない．

ANEP のパケットのフォーマットを図 4.8 に示す．パケット内の各フィールドの意味は以下の通りである．

Version プロトコルのバージョンを示す．

Flags パケットのタイプ ID が不明な場合の処理を指定する．

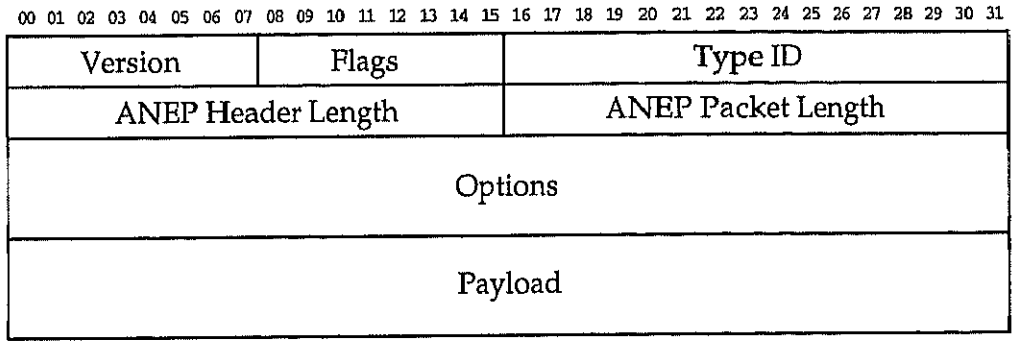
Type ID パケットのタイプ．この値でパケットを処理すべきルーチンを指定する．

Options ヘッダ部に記述すべき追加情報．

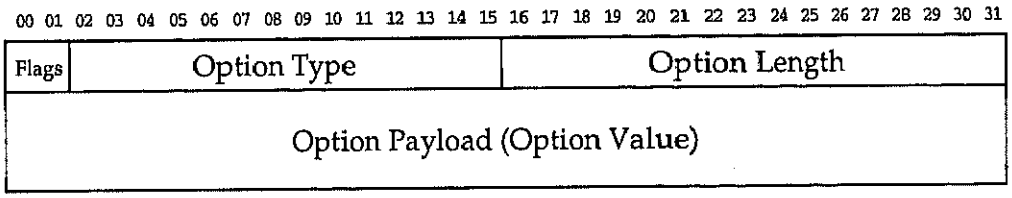
Option 部は，図 4.8 の下半分に記述したフォーマットをとる．Option のタイプは 4 種類が文献 [2] で規定されている．

本実現では，アクティブネットワークを利用したキャッシュ機構の導入のために，プロトタイプとして ANEP に基づいた簡単なアクティブネットワークシステムを作成した．このシステムを用いて仮想的なネットワークを構成し，その上に名前解決システムを実装する．

Header



Options



Option Type
1 = Source Identifier
2 = Destination Identifier
3 = Integrity Checksum
4 = N/N Authentication

図 4.8: ANEP パケット

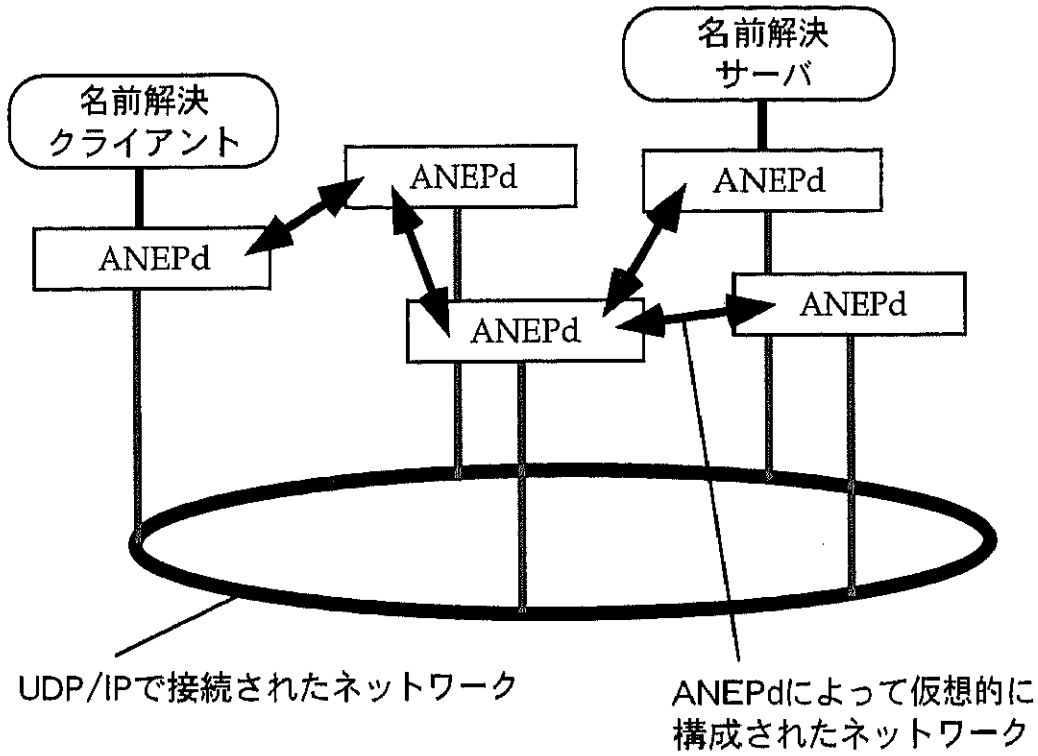


図 4.9: プロトタイプシステムの概念図

表 4.2: 実験環境

計算機	Sun Netra t1 7 台
CPU	UltraSPARC-IIi, 440MHz
メモリ	512MBytes
ネットワーク	100BASE-TX FastEthernet (スイッチを経由し接続)
OS	Solaris 2.6
実装言語	Java (JDK 1.2.2)

プロトタイプの概念図を図 4.9 に示す．図中の ANEPd(Active Network Encapsulation Protocol Daemon) が，作成したプログラムである．ANEPd は Java 言語 [4] を用いて記述した．ANEPd は仮想的なネットワークを UDP/IP[24] の上に構築する．つまり，ANEPd によって配送されるパケットは，その仮想的なネットワークによって送信され，ある 2 つのノードが UDP/IP 上では直接通信できる場合であっても仮想的なネットワークに沿ってパケットは送信されることになる．

図 4.10 に ANEPd の構成図を示す．ANEPd は，仮想的な通信デバイスとして振る舞う．すなわち，ANEPd に ANEP パケットを送ることで，ANEPd はそれを UDP/IP のパケットにラッピングし，パケットに指定されたアドレスへパケットを送信する．このとき，パケットの送信先は ANEPd がもつルーティングテーブルに基づいて決定される．つまり，単純に UDP/IP のパケットとしてネットワークに送出するのではなく，仮想的な ANEPd のネットワーク上のルーティングに基づいてメッセージの送受信が行われる．また，ネットワークから UDP/IP のパケットが到着すると，ANEPd は受信したパケットを ANEP のパケットとして取りだし，type ID にしたがって指定されたハンドラを呼びだす．各ハンドラは，そのパケットに対して適切な処理を行う．本提案方式のキャッシュ機構は，この ANEPd の内部のハンドラとして実装されている．

4.5 実験

本章では，アクティブネットワーク技術を利用したキャッシュ機構について，提案方式による性能の測定のための実験とその結果について述べる．実験に使用した計算機環境を表 4.2 に示す．また，計算機の接続の構成を図 4.11 に示す．多重名前空間システムでは，リモートリンクや名前空間の重ね合わせが可能であるが，本実験では実験結果を明確にするため単一のサーバのみを使用して実験を行った．

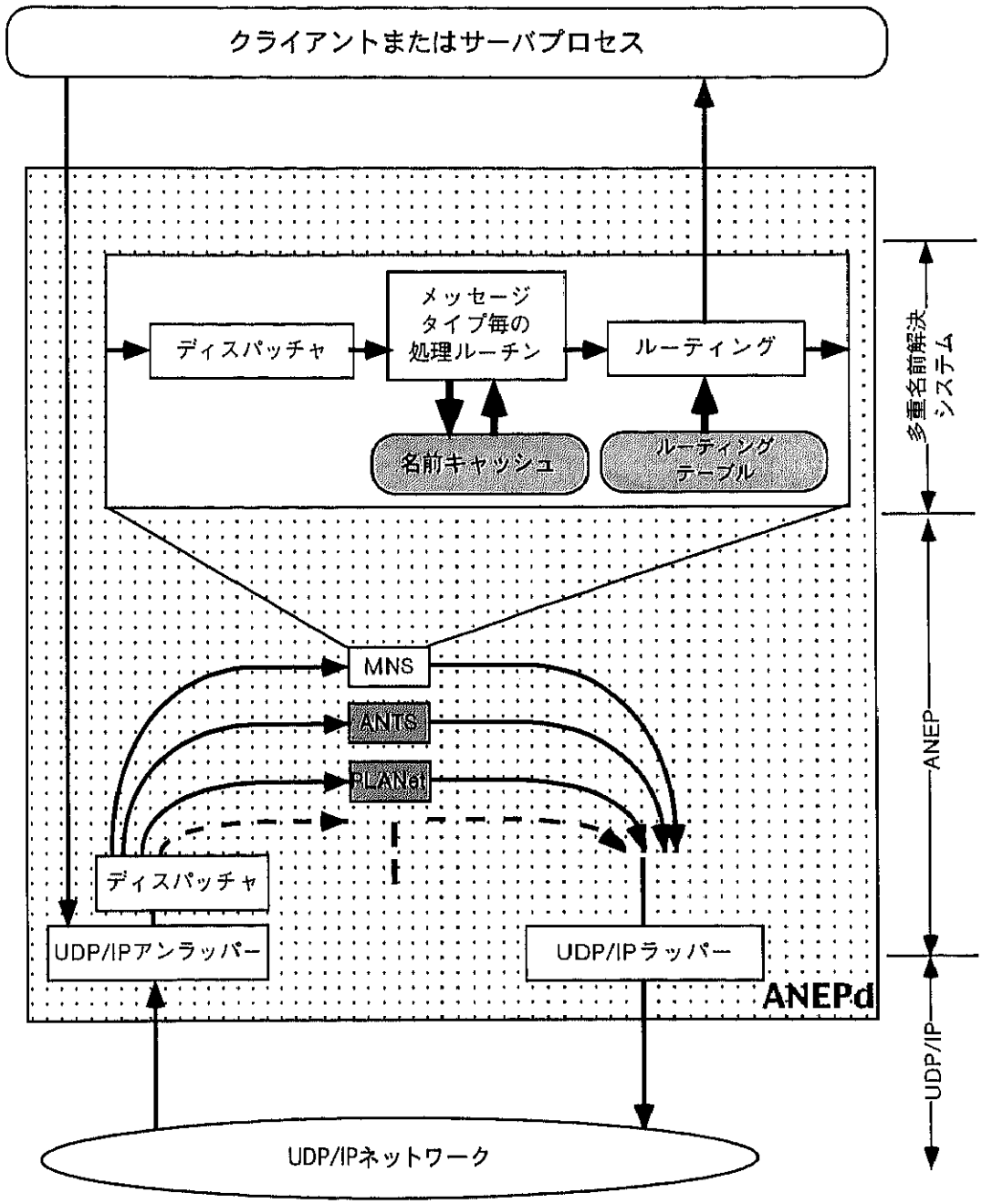


図 4.10: ANEPd の構成

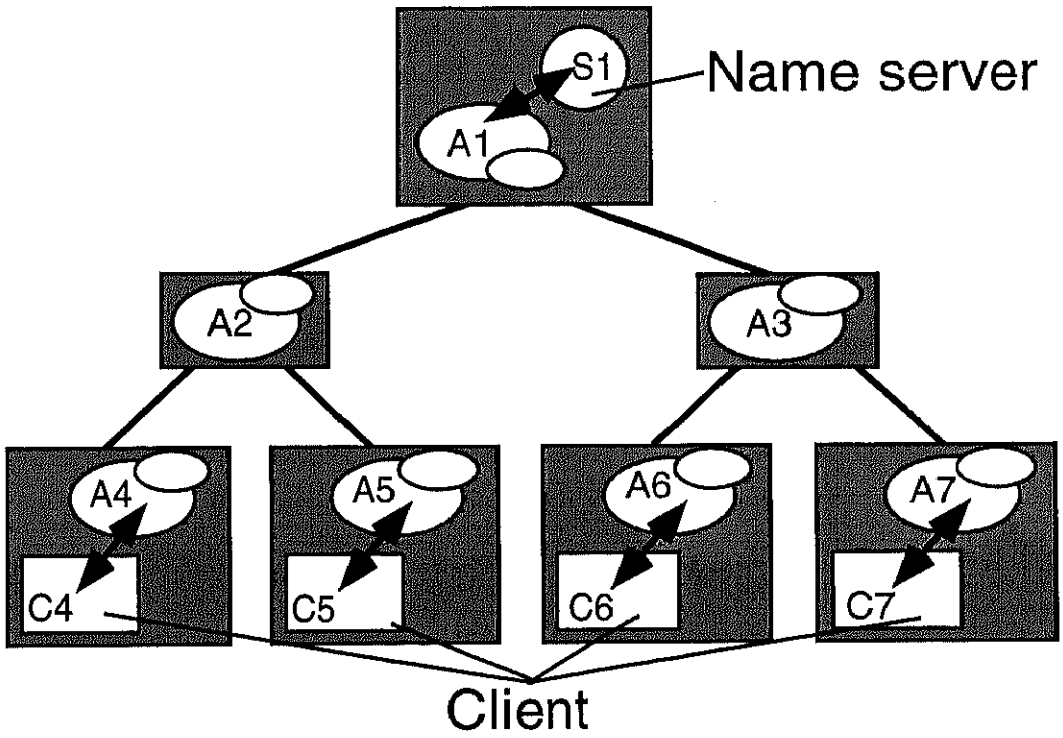


図 4.11: 計算機の構成

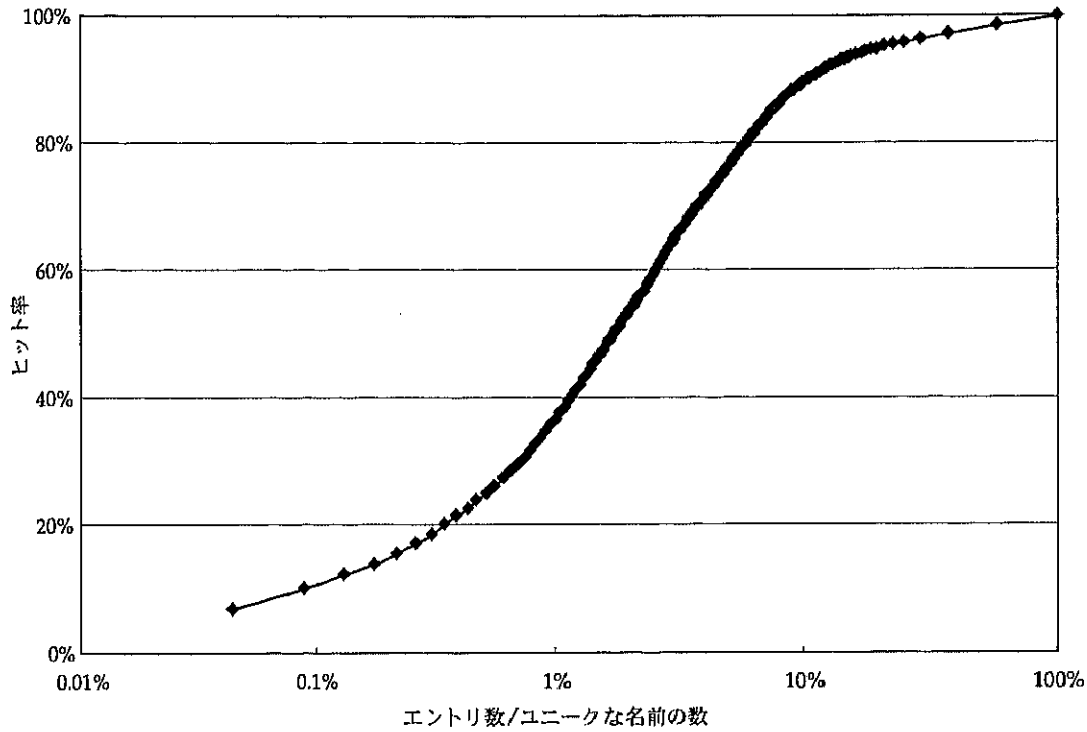


図 4.12: リクエスト列の局所性

実験において現実的なリクエスト列を使用するため，本実験では当研究室の DNS サーバへの問い合わせ記録から抽出した問い合わせ列を使用した．実験では，それぞれのクライアントは 1000 件のリクエストを発行する．そして，全リクエストに占める名前の登録および削除の割合を変化させた．名前の登録および削除の割合を更新率とする．実験では，更新率の値を 1%，10%，50%と変化させた．これらの 1000 件のリクエスト中には 500 種類の名前が含まれており，10%の名前が 90%のリクエストを占めるという局所性を持っている．名前の分布を図 4.12 に示す．この図は，名前の頻度が多いものから順に 10%(約 50 種類) の名前をキャッシュすると，キャッシュのヒット率は 90%となることを示す．

本実験では，CACHEINFO メッセージが伝播する頻度のパラメータ N_{info} は 10 に固定した．このため，10 回のキャッシュエントリへのアクセスに対し 1 回の CACHEINFO メッセージが発行されることになる．

4.5.1 実験 1: キャッシュサイズ

この実験で使用したキャッシュのパラメータを表 4.3 に示す．

表 4.3: 実験 1 で使用したパラメータ

N_{acc}	N_{prop}	更新率	キャッシュサイズ
1	1	1%, 10%, 50%	0–500 エントリ

実験では 3 つの観点から性能向上の度合いを評価した.

第一に, キャッシュ機構によってサーバに届くリクエストを削減できるため, サーバへのアクセス数を用いてサーバから見た性能の向上の度合いを測定した.

第二に, クライアントのリクエストは途中のノード上のキャッシュを用いて解決されるため, これによりクライアントへの返答が高速に行える. このため, クライアントのリクエストの平均ホップカウントを用いてクライアントの観点の性能の向上の度合いを測定した. ホップカウントとは, リクエストパケットが送信されてクライアントに戻ってくるまでの間に何回パケットが転送されたかを表す数である.

例えば, 図 4.11 においてクライアント C4 がリクエストメッセージを送信し, 名前サーバまでその要求が届き, 名前サーバが返答パケットを送り返した場合, ホップカウントは 8 となる. つまり, C4→A4 + A4→A2 + A2→A1 + A1→S1 + S1→A1 + A1→A2 + A2→A4 + A4→C4 の 8 回の転送を表す.

第三に, キャッシュ機構はネットワークのトラフィックを削減できる. このため, 交換されたパケットの総数を, ネットワークから見た性能の向上の度合いとして使用する.

ここで, 性能の向上の度合いとして, 以下の式で表される値 P_n を用いる.

$$P_n = \frac{p_0 - p_n}{p_0 - p_{500}} \times 100 \text{ [単位: \%]}$$

ここで, P_n はキャッシュサイズが n エントリのときの性能の向上の度合いを表す. また, p_n はキャッシュサイズが n エントリの時の性能 (サーバに到達したリクエスト数, ホップカウント, または交換されたパケットの総数) を表す.

名前サーバに到達したリクエスト数を図 4.13 に示す. 横軸はキャッシュサイズ, 縦軸は名前サーバに到達したリクエスト数を表す. この結果から, キャッシュサイズが 50 エントリ (すなわち, 総名前数の約 10%) であるとき, 性能の向上を示す値 P_{50} は, 更新率が 1%, 10%, 50%のときそれぞれ 45%, 32%, 31%となっている. 同様に, キャッシュサイズが 100 エントリ (すなわち, 総名前数の約 20%) であるとき性能の向上を示す値 P_{50} は, 更新率が 1%, 10%, 50%のときそれぞれ 70%, 51%, 45%となっている.

平均ホップカウントを示したグラフを図 4.14 に示す. 前述の結果と同様, キャッシュサイズが 50 エントリであるとき, 性能の向上を示す値 P_{50} は, 更新率が 1%, 10%,

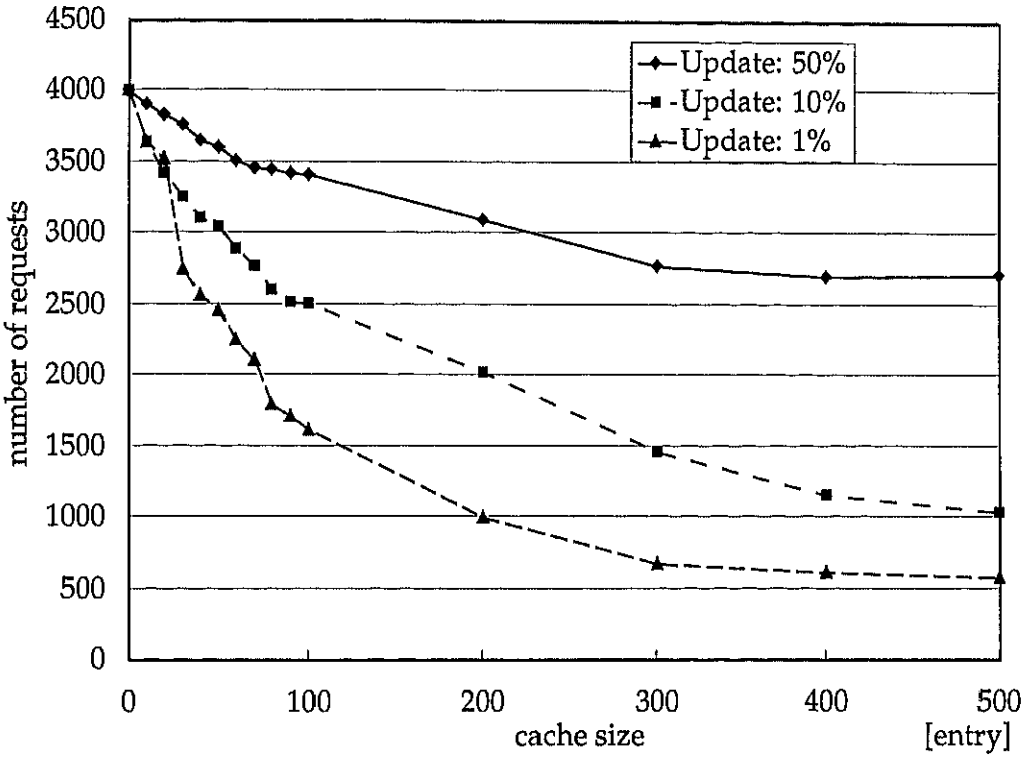


図 4.13: サーバに到達したリクエスト数

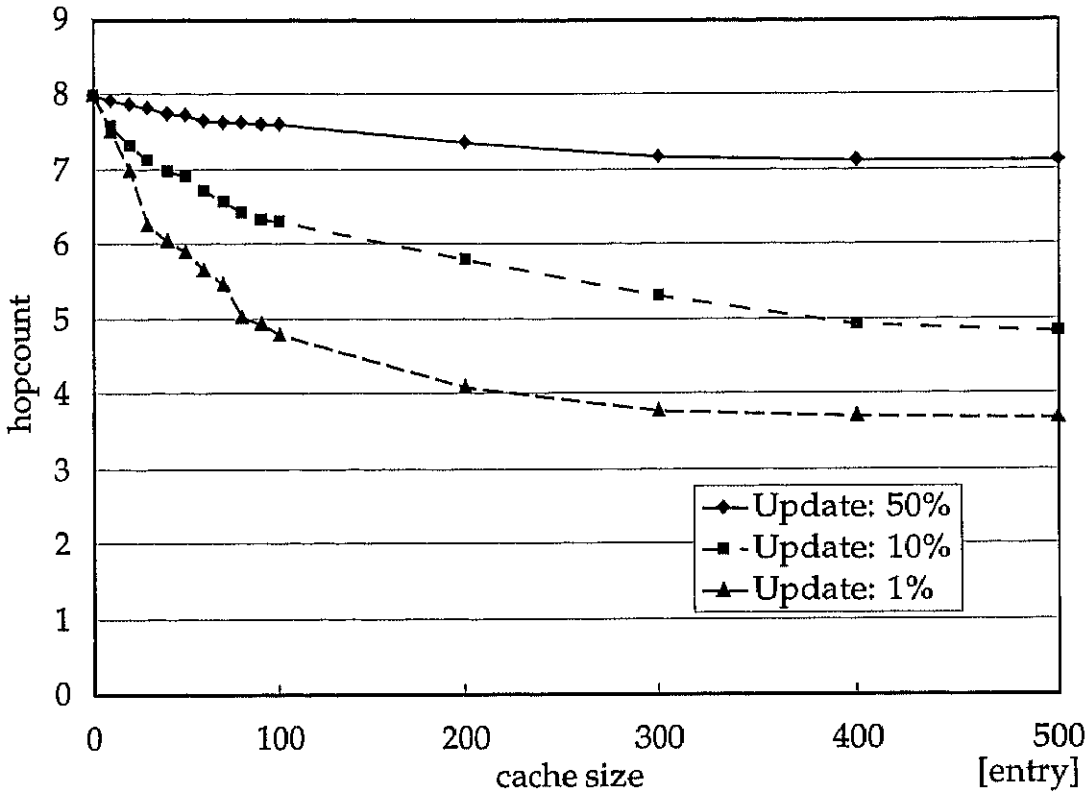


図 4.14: クライアントのリクエストの平均ホップカウント

50%のときそれぞれ 48%, 34%, 31%, 同様に, キャッシュサイズが 100 エントリであるとき P_{50} は, それぞれ 74%, 53%, 45%となっている.

また, 交換された総パケット数を図 4.15 に示す.

この結果によると, 更新率が 50%と高い場合, キャッシュが大きいことによる利点は失われてしまう. また, 交換されたパケットの総数は, キャッシュサイズより主に更新率に影響されることがわかる.

以上の結果, 全ての実験において, 総名前数の 10%という少ないキャッシュサイズで 50%前後の性能向上が得られた.

4.5.2 実験 2: パラメータ N_{acc} と N_{prop}

提案方式では, キャッシュの伝播は 2 つのパラメータ N_{acc} と N_{prop} でコントロールされる. 本実験では, 提案方式におけるパラメータの変化が性能にどのように影響するかを調べるため, パラメータ N_{prop} を 1 に固定して, パラメータ N_{acc} を 1 から 3 に

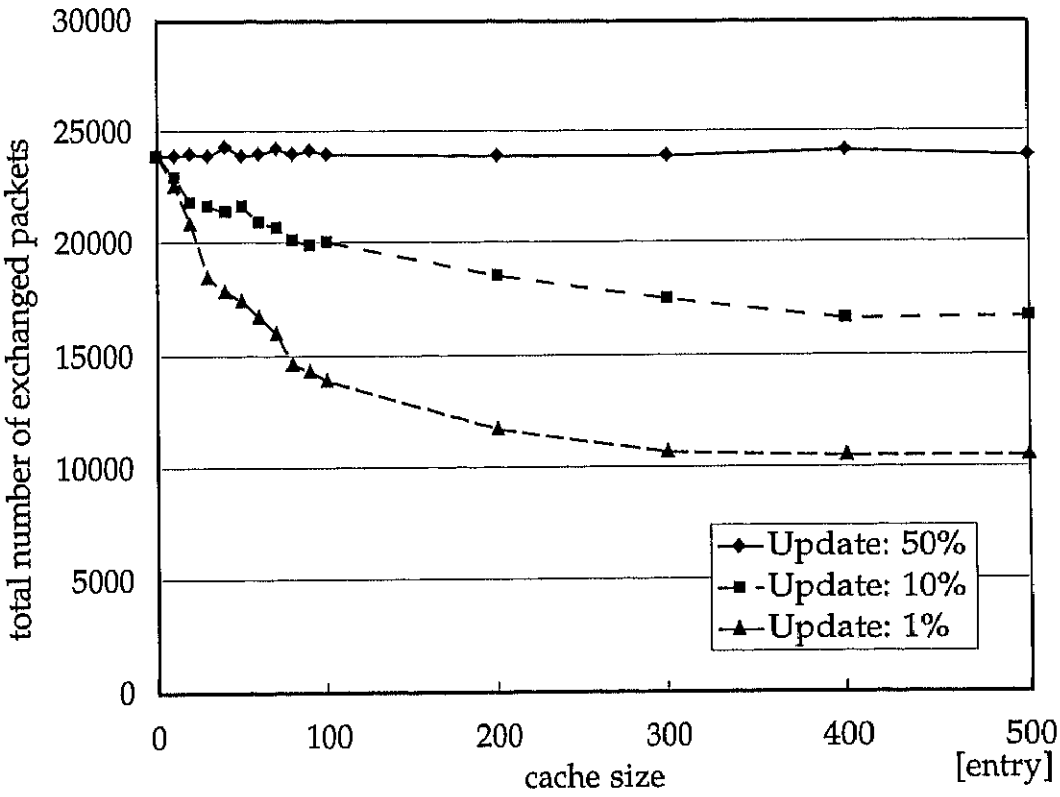


図 4.15: 交換されたパケットの総数

表 4.4: 実験 2 で使用したパラメータ

N_{acc}	N_{prop}	更新率	キャッシュサイズ
3	1	1%, 10%, 50%	100 エントリ
2	1	1%, 10%, 50%	100 エントリ
1	1	1%, 10%, 50%	100 エントリ
1	2	1%, 10%, 50%	100 エントリ
1	3	1%, 10%, 50%	100 エントリ

変化させた場合，およびパラメータ N_{acc} を 1 に固定してパラメータ N_{prop} を 1 から 3 に変化させて実験を行った．すなわち，以下に示す 5 種類のパラメータとなる:

$$(N_{acc}, N_{prop}) = (3,1), (2,1), (1,1), (1,2), (1,3).$$

パラメータ N_{acc} が大きくパラメータ N_{prop} が小さいとき，キャッシュエントリはゆっくりと伝播する．逆に，パラメータ N_{acc} が小さくパラメータ N_{prop} が大きいとき，キャッシュエントリは早く伝播する．

実験 2 で使用したパラメータを表 4.4 に示す．

第一に，ネットワークのトラフィックについて調べた．図 4.16 に交換されたパケットの総数を示す．伝播が早い場合 ($N_{prop} \geq 2$)，大幅なパケット数の増加が見られる．このようなパラメータの設定の場合，キャッシュエントリは，それがただ 1 回だけアクセスされただけでも広範囲に伝播し，キャッシュエントリの置換が起こった場合，それは高い確率で無効化される．今回の実装では，キャッシュ置換アルゴリズムに least-frequently-used(LFU) アルゴリズムを使用しており，1 回のみアクセスされた名前はキャッシュから置換されやすい．キャッシュ置換アルゴリズムの選択には考慮の余地があると考えられる．

第二に，名前サーバに到達するリクエスト数について調べた．図 4.17 に，名前サーバに到達したリクエストの総数を示す．サーバへの負荷軽減の度合いは，キャッシュアルゴリズムのパラメータに比べ名前の更新率に強く依存する．これは，名前の更新の際には，サーバへのアクセスが必須のものとなるためである．

第三に，クライアントのリクエストの平均ホップカウントについて調べた．図 4.18 に，クライアントのリクエストの平均ホップカウントを示す．キャッシュを早く伝播させると，リクエストの平均ホップカウントを減少させることが可能であるが，その反面，図 4.16 が示す通り一貫性を維持するための多大なオーバヘッドがかかる．

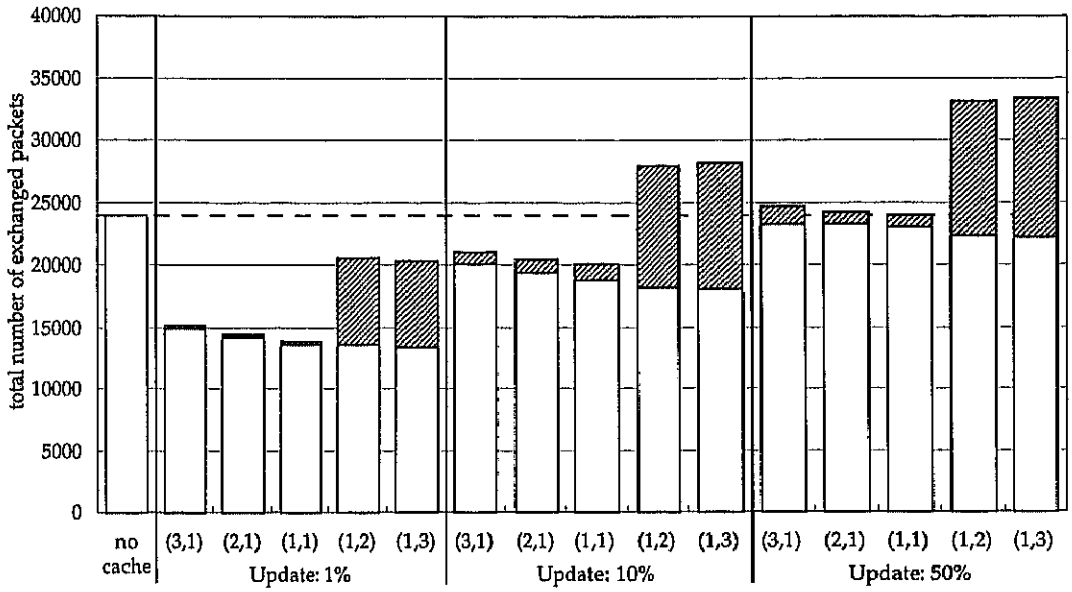


図 4.16: 交換されたパケットの総数．斜線部は一貫性維持によって生じたオーバヘッドを示す．

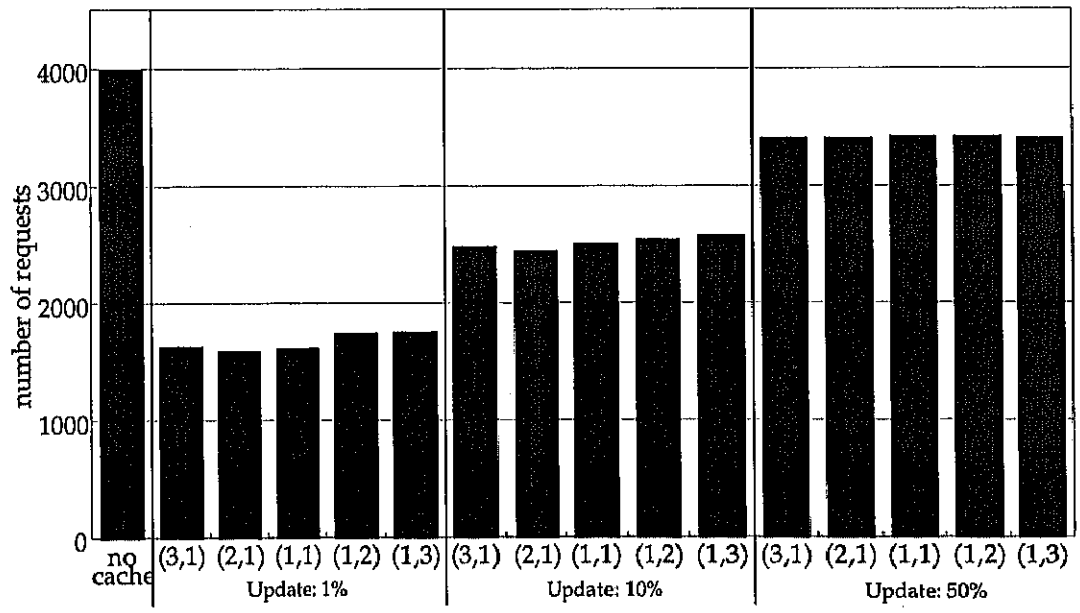


図 4.17: サーバへ到達したリクエストの総数

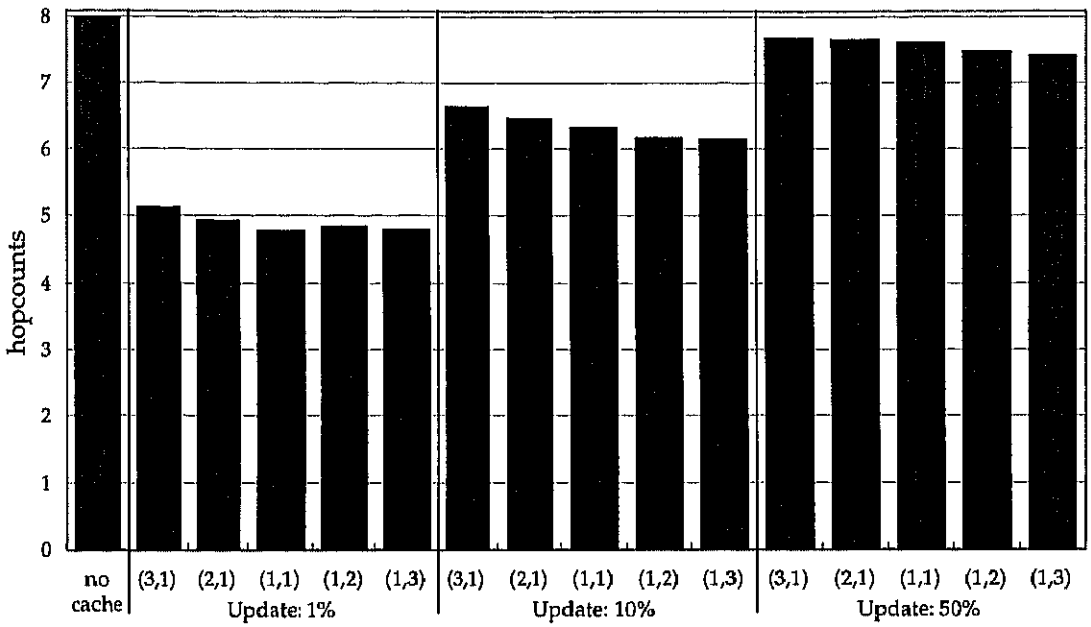


図 4.18: クライアントのリクエストの平均ホップカウント

表 4.5: 実験 3 で使用したパラメータ

N_{acc}	N_{prop}	更新率	キャッシュサイズ	伝播遅延
1	1	1%, 10%, 50%	100 エントリ	0–50ms

これは，クライアントの性能とネットワークの性能のトレードオフ関係になっている．一貫性管理のためのトラフィックを制限するためには，キャッシュの伝播を制限しなければならず，このときクライアントの性能は低下してしまう．また逆に，クライアントの性能を優先する場合は，キャッシュを積極的に伝播させることが必要となるが，その時は一貫性管理のためのトラフィックは非常に大きくなってしまう．

4.5.3 実験 3: 伝播遅延とレスポンスタイム

実験 3 では，クライアントにおけるレスポンスタイムについての実験を行った．この実験の際に使用したパラメータを表 4.5 に示す．

クライアントのレスポンスタイムとは，リクエストパケットを送信し，返答パケットを受け取るまでの時間である．図 4.19 にクライアントのレスポンスタイムのグラフを示す．横軸がノード間の伝搬遅延，縦軸がレスポンスタイムである．Ideal RTT(round-

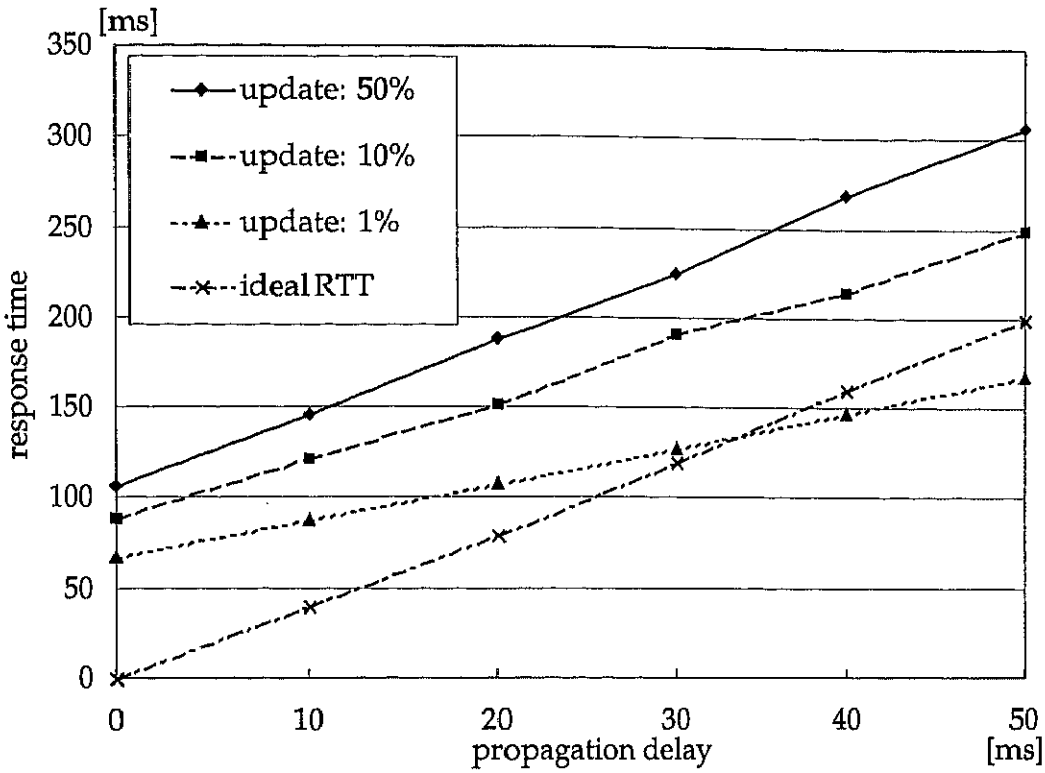


図 4.19: クライアントのレスポンスタイム

trip time) とは，キャッシュ無しのシステムで到達できる理論上の最低のレスポンスタイムである．すなわち，各ノードや名前サーバ上での処理時間を 0ms としたときのクライアントレスポンスタイムであり，キャッシュ機構がない場合に名前解決システムが到達できる理論的ピーク性能を示すものである．

この結果から，名前の更新率が比較的小さい場合 (1%)，ノード間の遅延が 35ms 以上の時に ideal RTT よりもレスポンスタイムが向上していることがわかる．今回の実装は，アルゴリズムの動作を確かめるプロトタイプ実装であるため，実装にはまだ性能向上の余地が存在する．また，ネットワークやサーバからの観点では，提案システムはトラフィックの削減を実現しており，この結果によって提案システムの有効性がスポイルされるということはない．

4.6 まとめ

本章では，名前解決に対するキャッシュシステムを，ネットワーク上に置く intermediate caching の手法を示した．提案方式では，クライアントおよびサーバに対し

透明な形でキャッシュを導入することで，プログラミング上の利点と既存のプロトコルに対するキャッシュの導入の容易さを得ることができることを特徴としている．また，本章では，提案システムを第３章で提案した多重名前空間システムに対して適用する例を示し，そのシステムにおける性能向上を実験によって確かめた．