アクティブネットワーク技術を利用した サーバ及びクライアントに透明なキャッシュ機構について

東村 邦彦 [1]　　加藤 和彦 [2,3]

1 筑波大学大学院 博士課程 工学研究科
2 筑波大学 電子・情報工学系
3 科学技術振興事業団

1　はじめに

広域ネットワーク上での DNS 等を用いた名前解決や，HTTP，FTP 等を用いたデータの取得などは，地理的に分散したサーバへの問い合わせによって行われる．これらの問い合わせにおいて，サーバや途中経路上のトラフィックの軽減，解決にかかる時間の短縮の観点から，キャッシュ技術などをもちいて無駄な問い合わせを避けることが重要な要件となる．

従来，キャッシュはクライアント側や，クライアントが明示的に指定した proxy などで行うことが普通であった．しかし，近年，PLAN[1]，ANTS[2]，NetScript[3] に代表されるアクティブネットワーク研究の出現により，ルータなどのネットワーク経路上のマシンに何らかの処理を行わせることが考えられるようになってきている．また，現在でも通常のワークステーションや PC をルータとして使用しているサイトや，ルータ専用機上に Web のキャッシング機能を導入した製品などもあり，ルータ上で何らかの処理を行うことは現実的なものと言える．

我々は，名前解決やデータの取得を行うために必要なパケットが通る経路上に存在するルータなどの上に配置する簡単なキャッシュ機構を提案する．この機構により，名前解決やデータの取得の効率化を図ることができる．クライアントや，クライアントが明示的に指定した proxy にキャッシュ機構を埋め込むのではなく，途中経路でキャッシュすることには，以下の利点がある．第一に，ネットワーク上で透過的にキャッシングを行うため，端点にあるサーバとクライアントは名前解決やデータの取得のアルゴリズムだけを純粋に記述するだけで良くなり，プログラミング上の利点が得られる．第二に，キャッシュする場所を，サーバ・クライアント間の経路上で任意に設定できることにより，サーバに近い場所にキャッシュを置くことで一貫性を細かく制御したり，クライアントに近い場所に置くことで，レスポンスを向上させたりできるなど，キャッシュする内容によって柔軟な対処を行うことが可能となる．

本論文では，この提案方式を簡単な名前解決システムに適用し，その性能の評価を行う．以下，第 2 章で提案方式の詳細，第 3 章で提案方式の名前解決システムへの適用，第 4 章で実装状況と実験について述べ，第 5 章でまとめる．

2　途中経路上でのキャッシュ

提案方式は，通信経路上の機器を用いて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，ルータに届い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及び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

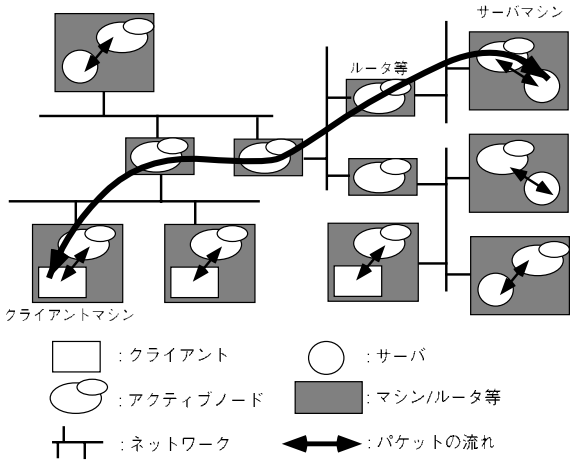

図 1: 提案方式の構成

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 1 に示す．クライアントが送信した要求パケットは，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) に送信される．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，それぞれにとってローカルのアクティブノードは，仮想的な通信デバイスとして振る舞うこととなる．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 2)．そして，クライアントからの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

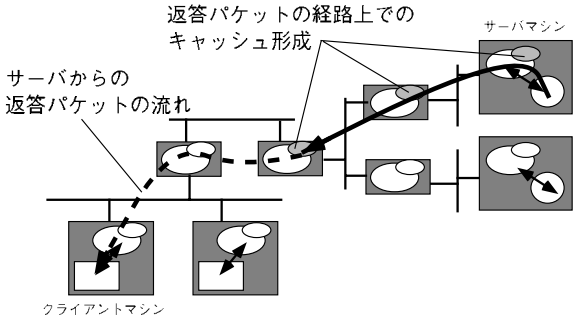

図 2: キャッシュの形成

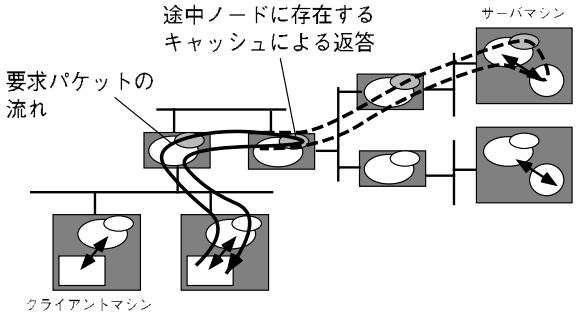

図 3: キャッシュの利用

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 3)．このとき，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．

しかし，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，サーバ上の情報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，情報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，更新頻度が高い情報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情報の更新コストを下げ，更新頻度が低い情報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，パケット経路にあるノードのうちサーバ寄りの N_{prop} 個に限定する．また，キャッシュが N_{acc} 回参照されるまでは，そのノードより先にキャッシュが伝播することを制限する．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情報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．

3 名前解決システムへの適用

本章では，提案方式を簡単な名前解決システムに適用した場合のキャッシュアルゴリズムについて述べる．まず，第 3.1 節において提案システムの実験のために使用した名前解決システムについて述べ，その後，第 3.2 節でキャッシュアルゴリズムの詳細を述べる．

3.1 名前解決システム

名前空間は，ドメインネームや Unix のファイルシステムのような木構造をしており，グローバルで単一な名前付けが行われるものとする．

名前空間は，システム内の複数のマシン上に存在する**名前サーバ**と，その間を結ぶ**リモートリンク**から構成される (図 4 参照)．各名前サーバには，名前空間の一部が保存される．リモートリンクは，各名前サーバに保存されている名前空間の続きに当たる名前空間へのリンクを保持している．リモートリンクの情報はシステム管理者が行うもので，ユーザはリモートリンクの存在を意識する必要はない．

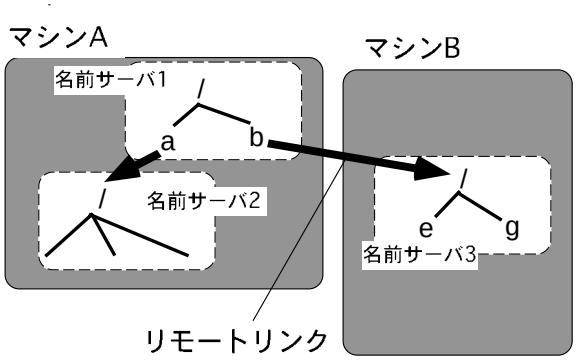

図 4: 名前空間のサーバ構成

表 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

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 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 1 に示す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．

```
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 5: 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

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 5 に示す.

動作例

図 4 に示される名前空間内で，/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.

3.2 インクリメンタルなキャッシュ構成方式

以下では，提案方式による名前解決の効率化手法について述べる.

キャッシュの内容

アクティブノード内のキャッシュは，論理資源名をキーとして検索する．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
- キャッシュの参照回数
- 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，サーバへの無駄な問い合わせを省略することが可能となる点である.

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

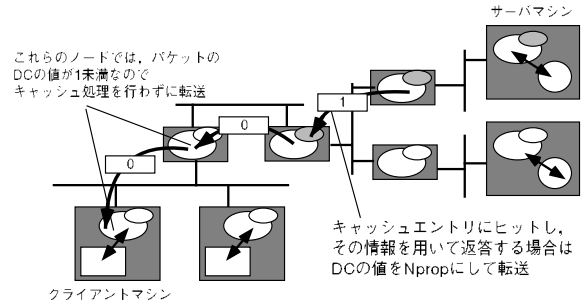

図 6: DC の付加によるキャッシュ伝播の制限

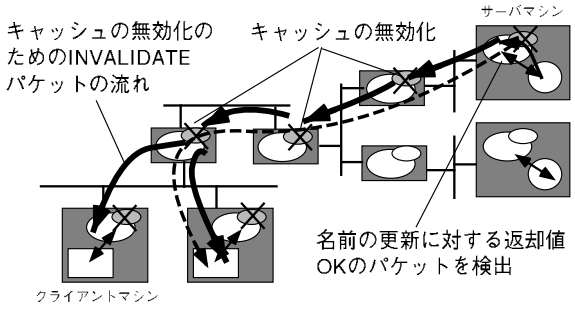

図 7: キャッシュの無効化

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 6 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.

キャッシュの無効化

サーバに名前が登録され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，無効化すべき論理資源名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 7)．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 8 に示す.

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 8: 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

表 2: 実装環境

CPU	Sun Ultra-60
OS	Solaris 2.6
実装言語	Java JDK 1.1.7
プロトコル	UDP/IP

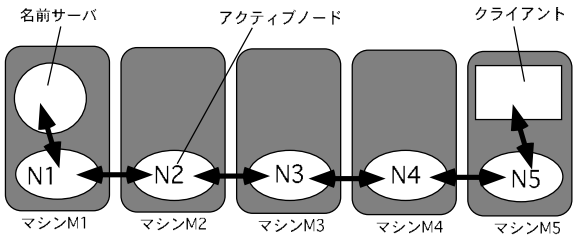

図 9: 実験環境

4 実装および実験

提案方式に基づいたキャッシュ機構の実装を表 2 に示す環境で行った.

本論文では，アクティブノードにおけるキャッシュアルゴリズムのパラメータ N_{prop}, N_{acc} を変化させた際,

- 要求が解決されるまでの hop 数
- アクティブノードを通過するパケットの数
- 無効化パケットの割合

がどのように変化するかを調べた.

実験環境を図 9 に示す.

実験では，まず，100 種類の名前から 50 種類を無作為に抽出し，その名前の登録を行った．その後，作成した名前空間に対し，名前探索，追加，削除を行った．この際に行う操作は，名前の更新 (追加と削除) の全体の操作に占める割合を 10%，50%，90%と変化させた 3 種類とした．実験の単純化のため，今回の実験では名前サーバを 1 つとしている.

アルゴリズムに与えるパラメータを，組 (N_{acc}, N_{prop}) の形で表すとする．実験で使用したパラメータは，伝播制限の弱い (1,4) から，(1,2)，(1,1)，(2,1)，(4,1)，(6,1) と次第に伝播制限を強くした 6 種類である．また比較実験としてキャッシュを全く行わない場合のパケット数なども測定した.

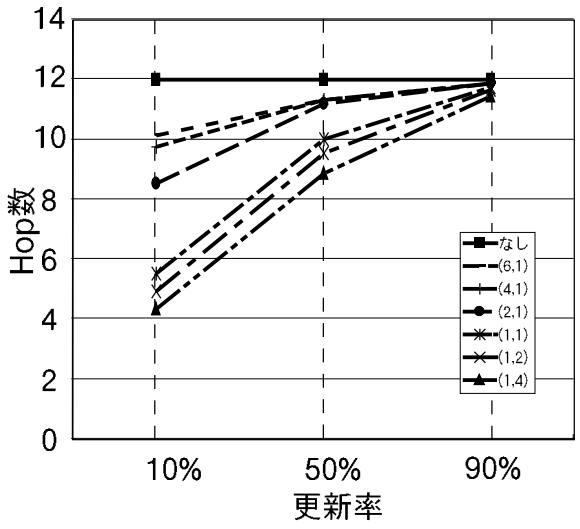

図 10: 更新率と hop カウント数．(n,m) は，キャッシュアルゴリズムに与えるパラメータを $N_{acc} = n$, $N_{prop} = m$ としたことを示す．

図 10 に，更新率を変化させた場合の各パラメータでの hop 数の変化を示す．ここで，hop 数とは，クライアントのリクエストが解決されるまでに何回メッセージの転送が発生したかを示す値である．例えば，図 9 において，クライアントからサーバまでを往復した場合，その hop 数は 12 となる．このグラフに示されるとおり，伝播制限を弱めるほど，更新率が低いときの hop 数は大きく削減できることが判る．

しかし，図 11 に示す通り，通過するパケット数で比べると，キャッシュの伝播制限が弱いほど更新率に対する変動が多く，特に更新率が非常に高い場合では，全くキャッシュを行わない場合と比べても通過パケット数が非常に多くなることが判る．これは，キャッシュが広く伝播することによって，その無効化パケットの送信がオーバーヘッドとなっていることによる．図 12 に無効化パケットの数の変化を示す．

このように，名前の更新率によって大きく性能が左右されるため，更新率や，経路上のアクティブノードの数によって適切なパラメータを選定することが重要な要件となる．

5 おわりに

アクティブネットワーク技術を応用に通信経路上でキャッシュを行うことで名前解決や情報の取得が効率化できることを示した．

今後の課題には以下のものが考えられる．第一に，アクティブノードで行うキャッシュ処理の高度化によるさらなる性能の向上がある．これには，無効化パケットの増大などを感知し，キャッシュアルゴリズムのパラメータを adaptive に変更する事，また，キャッシュのリプレースメントアルゴリズムの検討などが考えられる．第二に，UDP / IP レベルのパケットを直接 snoop するような低レベルな実装を行うことがある．現在，提案方式の実装には Java を用いているが，トラフィックが非常に大きくなると考えられる基幹のルータなどに実装する場合は，低レベルな部分で実装を行うことで最大限の高速化を行い，パケットのスループットを向上させることが必須といえる．第三に，FTP などの他のプロトコルへの提案方式の適用することが考えられる．

参考文献

[1] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pp. 86–93. ACM, 1998.

[2] David J. Wetherall, John V. Guttag, and David L. Tennenhouse. ANTS: A toolkit for building and dynamically deploying network protocols. In *IEEE OPENARCH'98*, April 1998. San Francisco, CA.

[3] Yechiam Yemini and Sushil da Silva. Towards programmable networks. In *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*, October 1996. L'Aquila, Italy.

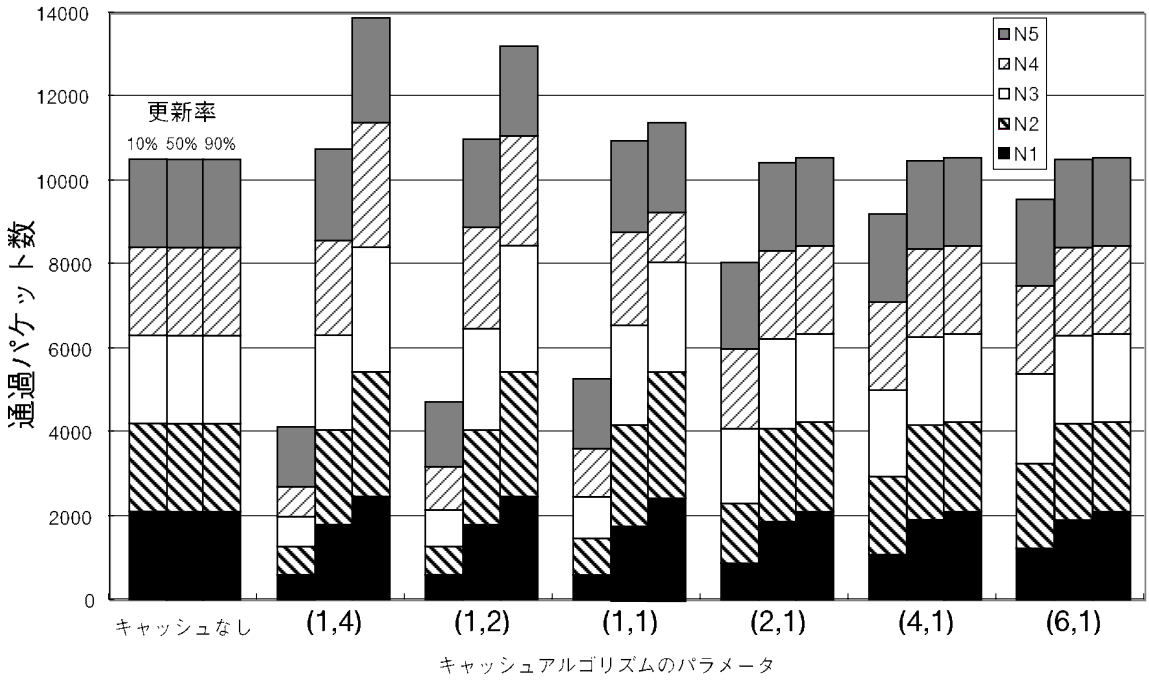

図 11: 通過パケット数

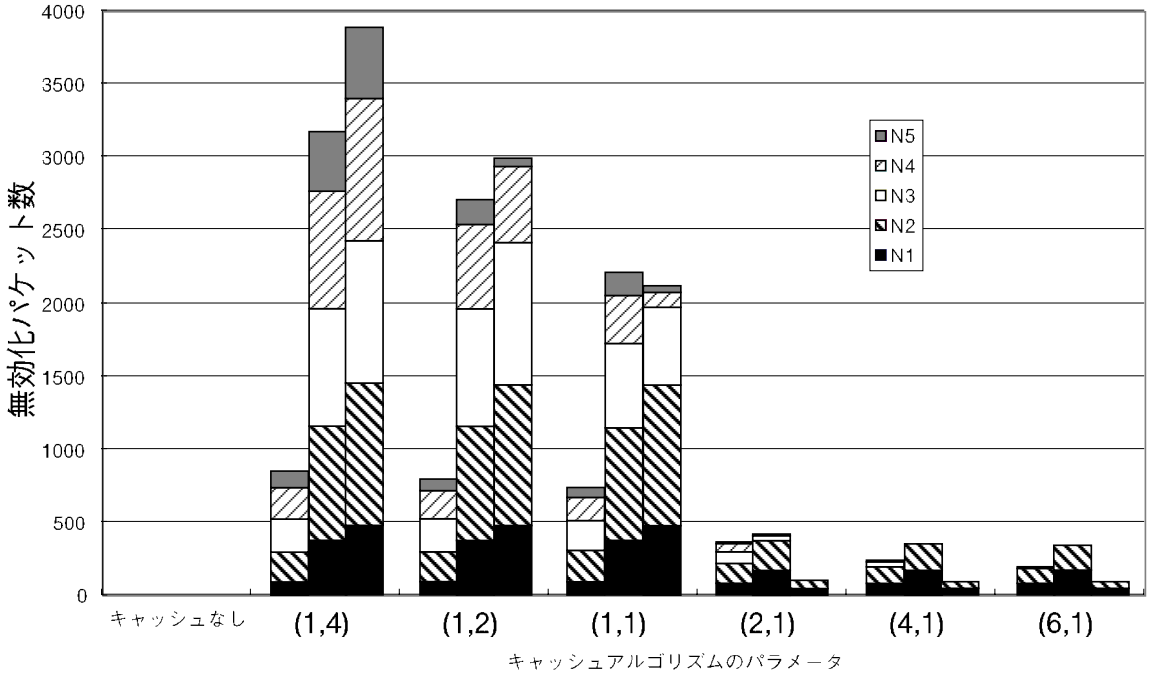

図 12: 無効化パケット数