修 士 論 文

IP ネットワークに発信規制の概念を適用した輻輳制御の研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

本田 英之

2008 年 3 月

修 士 論 文

IP ネットワークに発信規制の概念を適用した輻輳制御の研究

指導教官　日比野靖 教授

審査委員主査　日比野靖 教授
審査委員　田中清史 准教授
審査委員　丹康雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

0610077 本田 英之

提出年月: 2008 年 2 月

概 要

本稿では, IP ネットワークの新たな輻輳制御の手法として, キャリアのネットワークで用いられている発信規制の概念を取り入れた輻輳制御手法を提案する.

社会的な要因によって 1 箇所にパケット送信が集中する様な場合, 従来の輻輳制御手法では集中している場所でのみ制御を行うため, 制御が追い付かなくなってしまう. また, 集中している場所を通るが, 集中とは関係ないパケットが影響を受けてしまうと言う問題点が有る. この問題を解決するために, 集中している宛先に絞って制御を行う制御手法を提案した.

提案する制御手法を適用したシミュレータの実装を行い, シミュレーションにより効果的な輻輳制御が可能である事を示す. また, 従来手法との比較を通じて, 提案手法の有用性を示す.

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

近年, IP 電話の普及などにより回線の使用率が増大したため, IP ネットワークにおける通信を円滑にし, ネットワークの性能を落とさない様にする輻輳制御がより重要となってきている.

IP ネットワークにおける現在の輻輳制御の手法として, AIMD, ECN, RED など, 様々な方法が提案されている [1, 2]. これらの手法では, 輻輳の起こりかけたルータにおいて, 送られてきたパケットに対して一定の基準に基づき棄却処理を行い, ルータ内のパケットを減少させて輻輳を解消しようとする [1]. その結果, 輻輳の起こりかけたルータ, またはそのルータから直に繋がっているノードに対して輻輳が解消される. しかし, 予め予測された量を超える莫大なパケットが送られた場合には, その処理が間に合わず輻輳が起こってしまう. また, 個々のルータでネットワーク全体の輻輳の状況を把握することは出来ず, 発信元では輻輳が起こった事が分からないので, 応答が無ければパケットを再送してしまう. よって, それまで以上のパケットが輻輳の起こったルータに対して流れ込んでしまい, 結果的に更に大規模な輻輳が起こってしまう. これでは根本的な輻輳の解決とは言えない. 特に, 社会現象により 1 箇所にパケットが集中した場合に, その場所に近いノードの通信までも大きく影響を受けてしまう. 集中とは全く関係の無い通信が阻害されてしまうのは大きな問題点であると言える.

以上の事から, より効果的な輻輳制御の手法が必要である.

1.2　研究の目的

本研究では, IP ネットワークにおける輻輳を解消する事を目指す. そのための有効な方法として, あらゆる通信が IP 上で行われる事を想定した上で, 従来手法に加えて, キャリアで使われている発信規制の概念を IP ネットワークに適用した新しい輻輳回避の手法を提案する. これは, 個々のルータで特定の宛先へ向かう通信に対する回線使用率が増加し, 輻輳が起こりかけた際に, その宛先に向かうパケットに対して規制を行う. そして規制されている宛先に向かうパケットに関しては, 送信元に近い場所でパケット棄却を行い, 特定の宛先への送信を規制する. こうする事で, 1 つのルータが処理するタスクを減少させる事が可能となる.

そして, 社会現象が原因となる様な 1 箇所へのパケット集中が起きた場合に, そのすぐ

側へと向かう様なパケットに関して, 集中する宛先へのパケットが規制されるため, 集中とは関係の無いパケットは影響を受けずに通信を行う事が可能となる.

本研究では, この提案手法実現のためのモデルの構築及び検証を行う.

1.3 論文の構成

本論文の構成は以下の通りである.

第 2 章では, 輻輳の概念, 輻輳制御のための一般原則, その方針について述べる.

第 3 章では, 従来の輻輳制御手法として, トランスポート層とネットワーク層における輻輳制御に関して述べる.

第 4 章では, 第 3 章で挙げた従来の輻輳制御では処理しきれない場合に関して述べ, その問題点を提示する.

第 5 章では, 第 4 章において提示した問題点を解決する新たな輻輳制御の手法を提案する.

第 6 章では, 第 4 章で挙げた問題が出る様なネットワークモデルを構築し, 第 5 章で挙げた提案手法を用いてシミュレーションを行う. その時, 同じ条件で従来手法でもシミュレーションを行う.

第 7 章では, 第 6 章で行ったシミュレーションを元に, 提案手法と従来手法を比べ, 考察する.

第 8 章では, 第 7 章の考察から, 本研究全体のまとめを行い, 今後の課題を提示する.

第2章　輻輳制御

2.1　輻輳とは

[1] によると, ホストからサブネットに投げられたパケットの数が処理能力の範囲内であれば, 送信誤りで配送されない場合を除いて, 全てのパケットは送信される. しかし, あまりのパケットがサブネットに流れ込んでしまうと性能が落ちてしまう. この様な状況を輻輳と言う. そして更に, 通信量が大きくなりすぎてルータが処理しきれなくなると, パケットは紛失しはじめてしまう. こうなってしまうと事態は更に悪化し, 性能は完全に落ちてしまい, ほぼ全てのパケットは送信されなくなってしまう. 輻輳が起きてしまうと, ネットワークにおける通信が全く働かなくなってしまうので, 輻輳制御が必要となる.

2.2　輻輳制御とフロー制御

輻輳制御とフロー制御は通信量を制御する点では同じであるため, 混同されやすく, その違いが分かりにくい. ここで, [1] による輻輳制御とフロー制御の違いを示しておく.

輻輳制御は, サブネットが与えられた通信量を確実に運べる様にするための制御である. 全てのホストやルータの振る舞い, ルータでの蓄積転送処理, サブネットの転送容量を低下させる要因に依存していて, これらの状態が変化すると輻輳制御が行われる. 送信元と宛先だけではなく, ネットワーク全体の通信を滞りなく行うための制御である

フロー制御は, 送信元と宛先の 2 点間での通信に関する制御である. 宛先が送信元に対して, 受信出来ない様な速さでデータを送信させないように制御を行う. そして, 頻繁に宛先から送信元にフィードバックを行っていて, 宛先がどう言った状態なのかを送信元に通知している. ネットワーク全体の通信は関しては関知せず, 送信元と宛先の通信能力を見ながら制御を行う.

2.3　輻輳制御の分類

輻輳を制御 · 回避するために様々な手法が提案されており, [3] では輻輳制御に関して細かく分類がされている. その中で最も大きく分類されているのが開ループと閉ループである.

開ループは, 問題が起こらない様に最初に確実な設計を行い, 問題を解決しようとす

る [1]. これは, 一度システムが立ち上がって実行されると, 途中で訂正や変更が行われる事はない. これは, 輻輳が起こってからどの様に対処するかと言うより, 輻輳を最小限抑える設計を考慮している. そして, 様々な層で適切な方法を用いてこれを達成しようと試みる. [4] では層ごとに輻輳制御の方針を分類してあり, 表 2.1 にそれを示す.

層	方針
トランスポート	ラウンドトリップ遅延の判断 タイムアウト 再送 順序の違うパケットのキャッシング 確認通知 フロー制御 バッファの管理
ネットワーク	コネクションメカニズム パケットのキュー・サービス パケット棄却 生存時間制御

表 2.1: 輻輳作用の方針

閉ループは, フィードバックで制御を行おうとしており,

1. いつどこで輻輳が起こったかを検知するためにシステムを監視する
2. 動作が行われる場所に監視した時に得た情報を伝える
3. 輻輳を解決するためにシステムを調整する

と言う 3 つの部分からなる.

1. の監視を行い輻輳を発見するための主な尺度として, 以下が挙げられる.

- バッファ容量不足のため破棄されたパケットの割合
- 平均のキュー長
- タイムアウトして再送されたパケットの数
- 平均のパケット遅延
- パケット遅延の標準偏差

これらの値は, 輻輳が酷くなるにつれて大きくなる.

第3章　従来の輻輳制御手法

3.1　トランスポート層における輻輳制御

3.1.1　概要

ここでは, トランスポート層における輻輳制御として, TCP の輻輳制御に関して述べる. [5] によると, TCP は,

- スロースタート
- 輻輳回避
- 早期再送
- 早期回復

の 4 つの輻輳制御プロトコルを用いて輻輳制御を行っている. まずはこれらに用いられている用語を定義する. 次に, それぞれのプロトコルに関して説明を行う. 次に, [5] による輻輳制御の手法に関して述べる. 次に, 代表的なトランスポート層における輻輳制御として, [6] に挙げられている,

- TCP Tahoe
- TCP Reno
- TCP Vegas

に関して述べる. 最後に, 提案手法と近い形で用いられている ECN について述べる.

3.1.2　用語の定義

セグメント

セグメント (SEGMENT) とは, 任意の TCP/IP のデータ, または ACK パケット, もしくはその両方を表す.

送信側最大セグメントサイズ

送信側最大セグメントサイズ (SENDER MAXIMUM SEGMENT SIZE : SMSS) は, 送信側が送信出来る最大のセグメントサイズを表す.

受信側最大セグメントサイズ

受信側最大セグメントサイズ (RECEIVER MAXIMUM SEGMENT SIZE : RMSS) は, 受信側が受け入れ可能な最大のセグメントサイズを表す.

フルサイズセグメント

フルサイズセグメント (FULL-SIZED SEGMENT) は, 許容出来る最大バイト数のデータを持つセグメントを表す.

受信ウインドウ

受信ウインドウ (rwnd) は, 直近で通知された受信ウインドウを表す.

輻輳ウインドウ

輻輳ウインドウ (cwnd) は, TCP の送信可能サイズを制限する状態変数を表す. TCP が送信する際には, 受信ウインドウサイズと比べて小さい方のウインドウサイズで送信される.

初期ウインドウ

初期ウインドウ (INITIAL WINDOW : IW) は, 3-way ハンドシェイクが完了した後の送信側の輻輳ウインドウサイズを表す.

ロスウインドウ

ロスウインドウ (LOSS WINDOW : LW) は, TCP の送信側が自身の再送タイマを使用する事でロスを検知した後の輻輳ウインドウサイズを表す.

リスタートウインドウ

リスタートウインドウ (RESTART WINDOW : RW) は, アイドル時間後に TCP が送信を再開した後の輻輳ウインドウサイズを表す.

フライトサイズ

フライトサイズ (FLIGHT SIZE) は, 送信したものの, まだ確認応答 (ACK) されていないデータの量を表す.

スロースタート閾値

スロースタート閾値 (ssthresh) は, スロースタート状態から輻輳回避に移るかどうかを判断するための閾値を表す. cwnd がこの値を超えると, 状態は輻輳回避に移行する.

再送タイマ

再送タイマ (retransmission timer) は, セグメントが送信されるとスタートする. タイマが切れる前に ACK が来れば, タイマは停止する. タイマが切れた場合は, セグメントの再送が行われ, またこの送信でのタイマがスタートする.

ラウンドトリップ時間

ラウンドトリップ時間 (Round Trip Time : RTT) は, ACK が返ってきたときの再送タイマの値を最初の再送タイマの値, つまり送信から ACK までの時間を表す.

3.1.3 輻輳制御プロトコル

スロースタート

スロースタートアルゴリズムは, まず cwnd = IW として通信を始める. TCP は新たにデータセグメントの ACK を受信する度に cwnd を最大で SMSS バイトずつ加算していく. つまり, cwnd は指数関数的に増加していく.

cwnd が ssthresh を超えるか, 輻輳が見られた時に, スロースタートの状態は終わる.

輻輳回避

輻輳回避に移行すると, 今度はセグメント毎ではなく, バーストに対して, ACK を受信する度に cwnd を増加させる. 今度は線形的に増加する様になる. これは, 送信がタイムアウトするまで続く. タイムアウトが起きると, ssthresh を

$$ssthresh = \max(\frac{FLIGHTSIZE}{2}, SMSS \times 2)$$

に設定し, また状態をスロースタートに戻す.

早期再送

早期再送アルゴリズムは, 3 つの重複した ACK を受け取った時, つまり, 間に他のパケットを挟まずに 4 つ同じ ACK を受信した時に起こる. 受信した後, TCP は再送タイマが切れるのを待たずに, 失ったセグメントの再送を行う.

早期回復

早期回復アルゴリズムは, 早期再送アルゴリズムが失ったセグメントを再送した後, 重複 ACK ではないものを受信するまで, この新しいデータ送信を管理する. 早期再送アルゴリズムと早期回復アルゴリズムは通常 2 つセットで実装される.

3.1.4 輻輳制御手法

TCP Tahoe

TCP Tahoe は, スロースタート, 輻輳回避, 早期再送を用いた最も初期の手法である. スロースタート時には, cwnd = 1 で始まり, 輻輳回避の際には ssthresh = FLIGHT SIZE / 2 となる. この手法では, 輻輳の状態に関係なくパケットロスが起こる度にウインドウサイズは 1 となるため, 伝送効率は悪い.

TCP Reno

TCP Reno は, パケットロス時の処理が TCP Tahoe と異なり, 早期回復を加えた手法となっている. 早期再送後に重複 ACK と違う ACK を受信したら, ウインドウサイズを ssthresh として, 輻輳回避からスタートする. しかし, タイムアウトが原因でパケットロスした場合は, TCP Tahoe と同じ様に, ウインドウサイズは 1 となる.

TCP Vegas

TCP Vegas は, これまでの手法とは違い, 送信したセグメントの RTT を観測し, これをウインドウサイズの調整に用いる. RTT が大きければ輻輳が起きていると判断して cwnd を減少させ, 逆に RTT が小さければ cwnd を増やす. この手法はウインドウサイズを一定値に固定しているので, セグメントロスが起こらず安定したスループットが得られる. そして, ウインドウサイズの増加は, TCP Tahoe, TCP Reno の半分となっている.

3.1.5 ECN

Explicit Congestion Notification(ECN) は, ルータがエンド・ノードにネットワークの輻輳状態を明示的に通知する機能を持つ. ルータにおいて,

- パケットを中継する際に, 転送待ちパケット数を検査
- 転送待ちのパケット数が一定数を超える場合は, ネットワークの混雑度が大きいと判断
- 中継するパケットに輻輳の情報を付けて転送

を行う.

3.2 ネットワーク層における輻輳制御

3.2.1 概要

ここでは, ネットワーク層における輻輳制御として, 主にルーティングアルゴリズムとキューに関して述べる. ルーティングアルゴリズムでは, 宛先までの経路で最もコストの少ないものを選び, 効率良くパケットを送信する. キューは, パケットをためておく入れ物だと言える. 複数の入力フローと 1 本の出力リンクでパケットを処理するルータがあるとして, そのルータは一度に 1 つのパケットしか処理が出来ない. この時, キューにパケットを入れ, 保持しておく事で一定の順番を保って送信する事を可能にする [7].

3.2.2 ルーティングアルゴリズム

[1] によると, ルーティングアルゴリズムはネットワーク層のソフトウェアの一部であり, それぞれの入力パケットをどの出力ラインに送れば良いのかを決定する. ルータ内部においては, ルーティングアルゴリズムの役割である, 経路表に書き込みを行い, その内容を更新するプロセスだけでなく, フォワーディングと呼ばれる, パケットが到着した際に経路表を見て利用出来る出力回線を探すプロセスとがある.

ルーティングアルゴリズムには,

- 正確性
- 単純性
- 耐故障性
- 安定性

- 公平性
- 最適性

といった特性が望まれる. 大規模なネットワークでは, そのネットワークトポロジが何度も変化する可能性があり, その度に, 経路表を更新しなければならない. しかしそれは, ホスト上の全ての作業を中断させたり, ネットワークをリブートさせたりせずに, それを行わなければならない. そのために, 様々なルーティングが提案されており, ネットワークがブートされる時に情報が更新される静的ルーティングや, トポロジの変化や通信量の変化を反映して経路の決定を変える動的ルーティングがある.

3.2.3 キュー

キューを用いたアルゴリズムとして, 主に

- Droptail
- RED

を挙げる. ここでは, この 2 つに関して述べる.

Droptail

[7] によると, Droptail は, 単純な FIFO(First In First Out) のキューで, パケットの到着順にキューに格納され, その順番で出力される.

Droptail はその名の示すように, キューの中のパケット数をかぞえ, キューの最大長を超えるとパケット棄却する. キューの長さはバッファサイズでは無く, パケット数で見ている所に特徴が有る.

RED

[1] によると, RED は, ルータに輻輳が解消出来ない様な状況になってしまう前にパケットを棄却させ, 輻輳になる前に回線の混雑を解消してしまうアルゴリズムである. この時, 一番重要になってくるのが棄却を行うタイミングで. これを決定するために RED はキューの平均長を動的にカウントしていく. これがある閾値を超えた時, 輻輳が起きていると判断し, パケット棄却が行われる.

第4章　従来の輻輳制御の問題点

4.1　概要

ここでは従来の輻輳制御の問題点について述べる. まず, ネットワーク全体から見た輻輳制御の問題点について述べる. 次に, トランスポート層, ネットワーク層における輻輳制御に分けてより細かく従来手法の問題点を見ていく.

4.2　ネットワーク全体における問題点

従来の輻輳制御では, 社会現象の様な人間の行動が関わる様な場面で, 図 4.1 の様なモデルを考えてみる. 図の大きな□は複数のリンクを持つノードを示し, ○はリンクを一つだけ持つノードを表す. ノードを結ぶ線はリンクを表し, 太さは回線の太さを表す. リンクのそばにある小さな□はそれぞれパケットを表す. 中央の雲はその中がどの様なネットワークトポロジであるかが分からないと言う前提を表す (但し, 雲の中に入ったリンクは全て繋がっているものとする). 最後に dst と書かれたノードは, 今回の例で全てのパケットの宛先を表す. この様な前提で考えた場合, 1 箇所にパケット送信が集中した場合に, 処理が間に合わなくなる. 集中した 1 箇所でのみで制御していたのでは, 絶対に処理が出来ずに輻輳が起こり, 全くパケットを通す事が出来なくなってしまう (図 4.2). 図で, 輻輳が起こってしまったノードは■で表す.

更に, 通信が出来ない事が分かると, 繋がるまでパケットを再送しようとするため, 余計に輻輳が酷くなってしまう. この時に, 混雑が分かっていて, ルーティングアルゴリズムによって別のルートを通って行ったとしても, 図の様なネットワークの場合, 結局パケットは届かず, 輻輳は伝搬していってしまう (図 4.3).

ここで更なる例として, 新たに図 4.4 の様なネットワークモデルを考えてみる. ノード a, b, c, d, i は一斉にノード l にパケットを送信し, ノード j はノード k にパケット送信を行うものとする. 図 4.5 の様にノード l に向かうパケットがノード H に集中してしまい, 処理が間に合わなくなり輻輳が起こる (図では, それぞれの○のノードに書かれているのは宛先のノードを表し, パケットの中のアルファベットは, そのノードで発生した事を意味する). そしてこの場合, パケット集中の主たる原因ではないノード j からのパケット送信も影響を受けてしまっている.

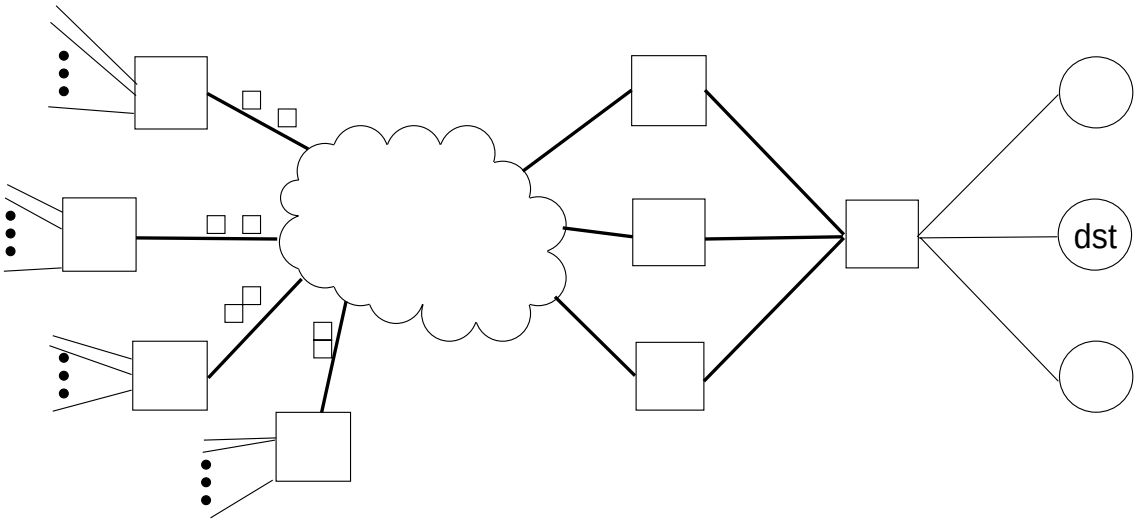

図 4.1: 1 箇所にパケット集中するネットワークモデル

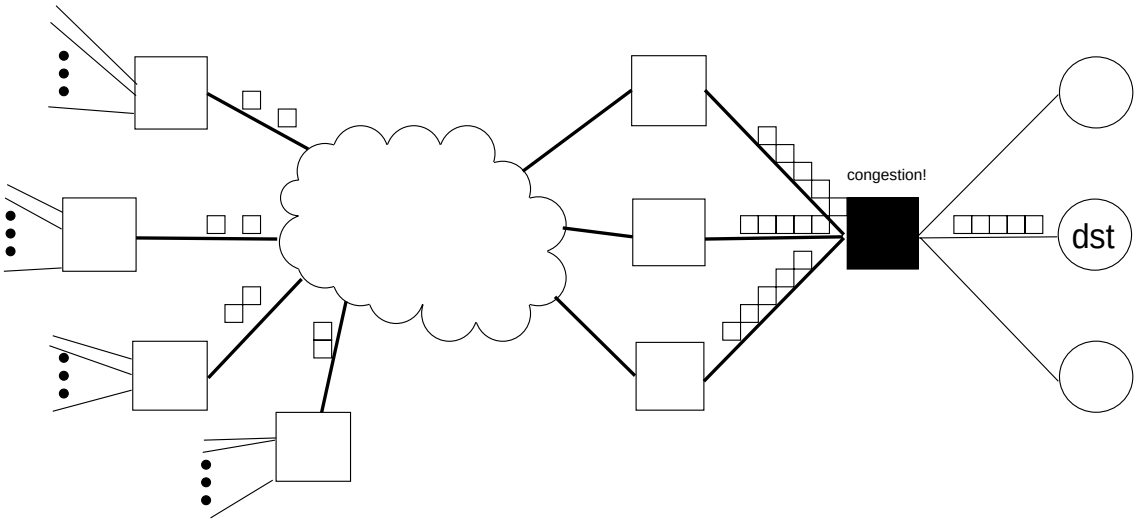

図 4.2: 輻輳の発生

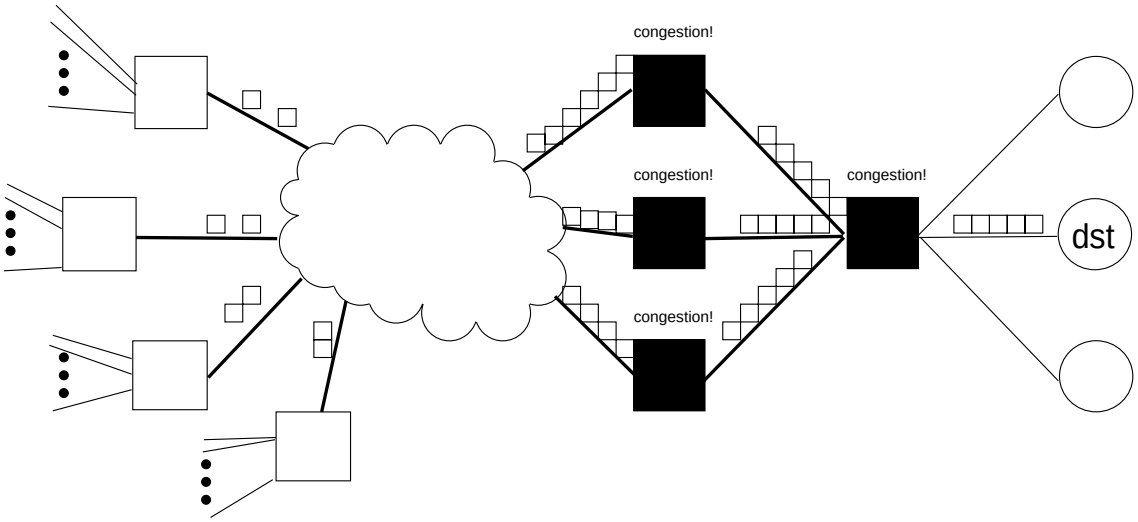

図 4.3: 輻輳の伝搬

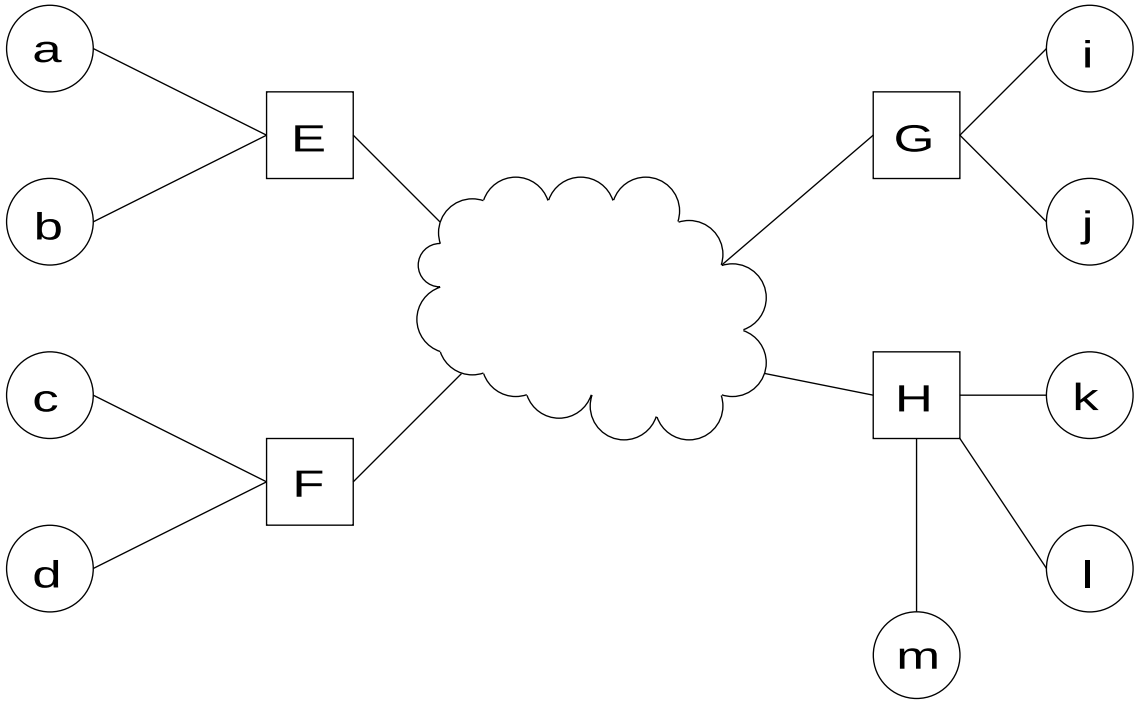

図 4.4: 集中とは関係ない送信を含んだネットワークモデル

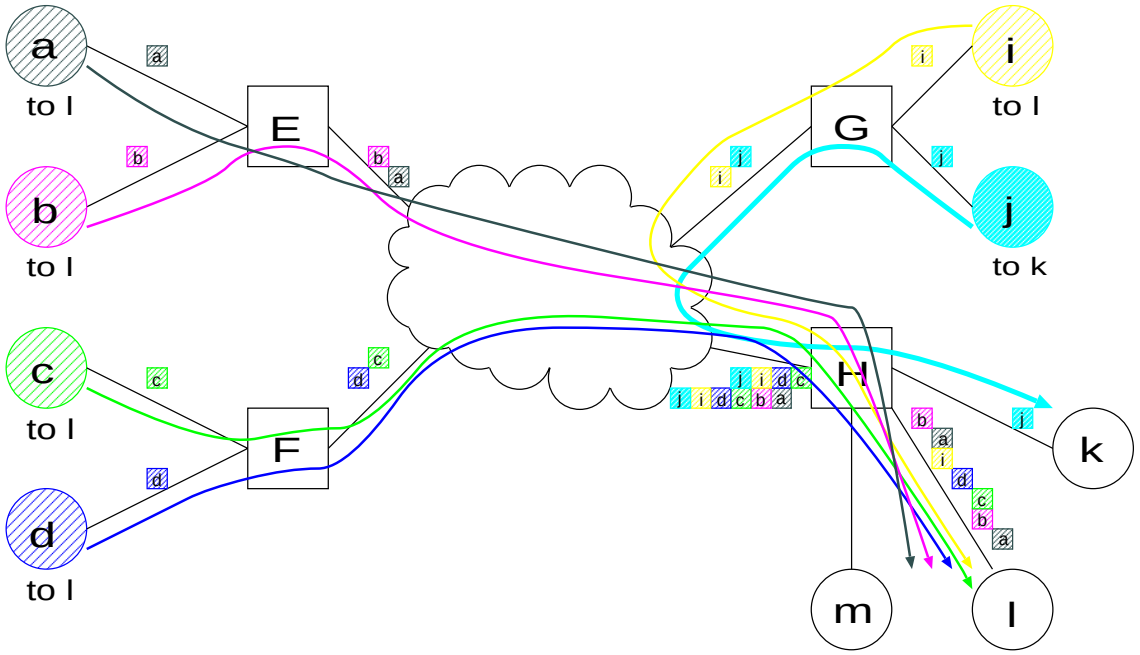

図 4.5: 輻輳に関係の無いパケット通信の阻害

4.3 トランスポート層における輻輳制御の問題点

トランスポート層での輻輳制御の問題点として, End to End での通信であるため, 途中の経路でのネットワークの状態を知る事が出来ない点が挙げられる. そのため, 送信するセグメントのウインドウサイズや, 送信のタイミングでしか制御が出来ず, 途中の経路で起こる輻輳に対して効果的な対処をする事が出来ない. そもそもこの層は輻輳制御には向いていないと言える.

更に, 基本的に送信元自らが通信を控える事で輻輳を回避しようとしているので, ECN を用いて輻輳状態が分かったとしても, 輻輳の原因が TCP 以外の通信である様な場合には, 通信を控えた分だけ結局他の通信に取られてしまい, 輻輳は全く回避出来ない.

4.4 ネットワーク層における輻輳制御の問題点

ネットワーク層での輻輳制御の問題点としては, 現在用いられている AQM などの手法では, 自ノードの輻輳の情報と, 隣接するノードの輻輳の情報までしか分からない点にある. 図 4.5 の場合, いくらノード H で輻輳が起きていてパケット処理が間に合わない様な状態でも, ノード E, F, G では特に輻輳状態では無く, ノード H で輻輳が起きている事が分からないので, ノード l に向かってただパケットを流してしまう. これでは輻輳は解決する事が出来ず, いずれ図 4.3 の様な状態に陥ってしまう.

更に, ネットワーク層では輻輳検知をキュー長を監視して行っており, 回線使用率につ

いては監視していない. キュー長と回線使用率との間には相関関係は有るものの, キューは一旦長くなり始めると指数的に増加してしまうため, キューが長くなってから輻輳制御を開始したのでは間に合わない可能性がある. キュー長を見ていては輻輳制御のタイミングを上手く取る事が出来ず, 適切な制御を行う事が難しい.

第5章　発信規制の概念を用いた輻輳制御

5.1　概要

従来の輻輳制御の問題点を解決し, 輻輳の根本的な原因を絶つ為に, キャリアのネットワークで用いられている発信規制の概念を IP ネットワークに適用した輻輳制御手法を提案する.

提案手法では, あるノードにおいて回線使用率を宛先毎に監視し, 輻輳を検知した場合, 輻輳を検知した宛先への通信に対して規制を行い, ある確率でパケットを棄却する. 回線使用率を監視し, 規制を指示するノードは宛先に近いノードで, 規制の通知をうけてパケットを棄却するノードは発信元に近いノードで行う. こうする事で発信元に近い所で規制が可能となり, 輻輳を根本的に解決する事が可能となる. 更に, 輻輳の原因ではない通信には規制がかからず, パケットが棄却されないので, 輻輳制御の影響を受ける事が無い.

5.2　提案手法の適用例

図 4.5 に示したネットワークに対して提案する輻輳制御を適用する.

5.2.1　前提条件

前提条件として, 個々のノードに対して, ネットワーク全体の IP アドレスは地理情報と対応させ, 組織的に割り当てられているものと仮定する. もしくは, IP アドレスと地理的な情報とがきちんと対応付けられているものとする. こうすることで, 実際に輻輳の原因となっている呼の発信源を容易に知る事が出来, 発信規制を柔軟に行う事が可能となる.

5.2.2　ピボットノード

実際にパケット監視, パケット棄却を行うノードをピボットノード (*pivot node*) と定義する. ピボットノードは予め通信が集中しそうな場所に設置する. 前提条件から, IP アドレスが組織的に割り当てられるとすると, ピボットノードはある程度末端から上位のアドレスにあるノードに置く.

例として, 図 5.1 の場合, ノード E, F, G, H, N, O は全てピボットノードにする事が可能

である. まず, ノード G, H, N, O をピボットノードとして設置し, もし, N, O だけでは制御が不十分になる様な場合には, 更に E, F をピボットノードとする事も可能となる.

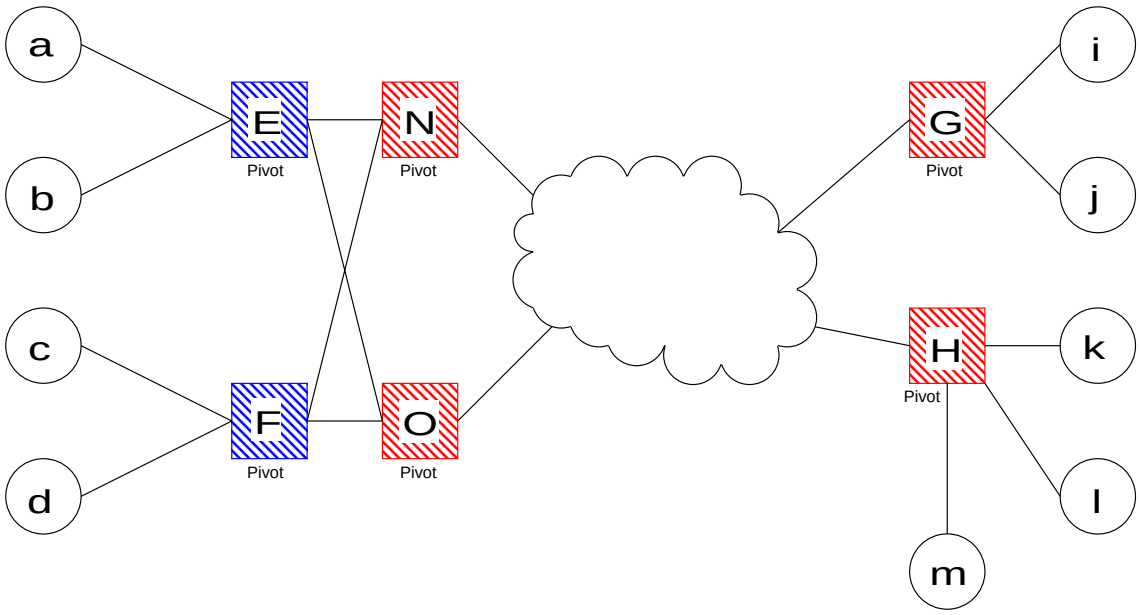

図 5.1: ピボットノード

5.2.3 提案手法の輻輳制御の流れ

ノード H で輻輳が検知されると, 最も回線使用率を占めている l 宛てのパケットに対して規制がかかる. すると, その通知を受けたノード E, F, G は l に向かうパケットにのみある確率で棄却を行い, 別の宛先 k に向かうノード j からの通信は全く影響を受けない. この結果, 図 5.2 の様になり, 輻輳の解決がなされる.

5.3 追加する機能

ここでは, 発信規制を実装するに当たって追加する機能に関して説明する. この機能はいずれもピボットノードに対して追加する. 機能が働く一連の流れを図 5.3 に示す.

5.3.1 パケット監視

ノードに流れてきたパケットのサイズ, 時間を宛先で分けて取得し, そこから宛先別の回線使用率を割り出す. 割り出した回線使用率を監視し, 閾値を超えた時, 超えた宛先は発信規制リストに加える. その後, 回線使用率が閾値の範囲内に収まったら, その宛先は

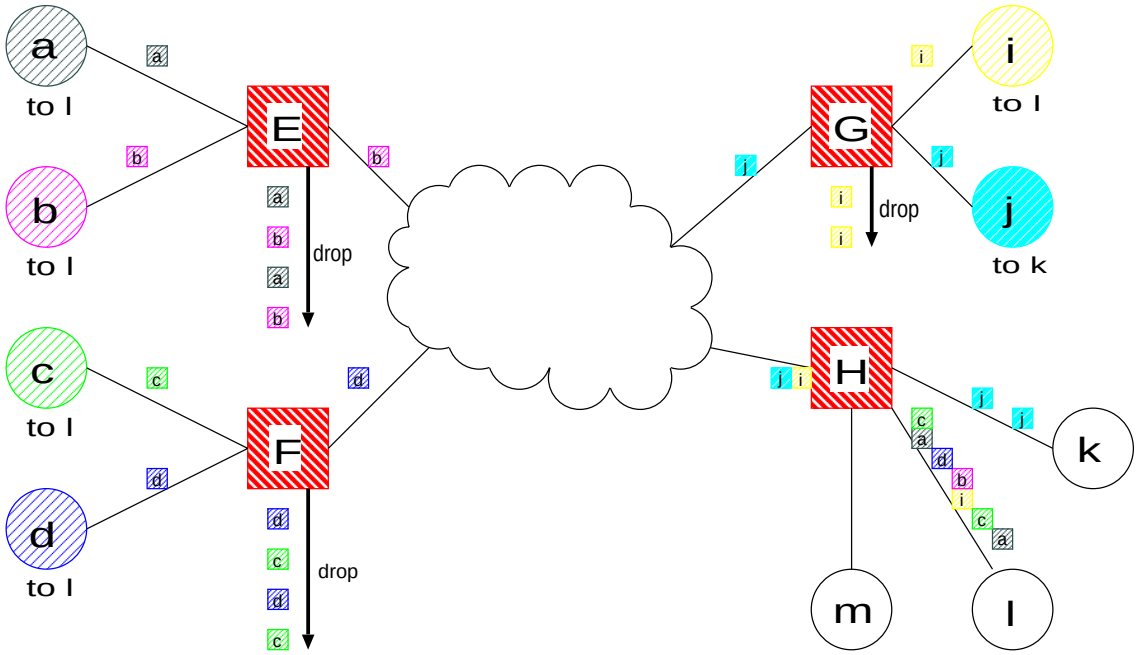

図 5.2: 発信規制による輻輳制御

発信規制リストから削除する. この発信規制リストの内容はピボットノード間で通信を行い, 動的に情報を更新する. キューではなく, 回線使用率を監視する事で, 輻輳に対して素早く対処する事が可能となる.

5.3.2 発信規制リスト

発信規制を行う対象となる宛先を全てこのリストに登録する. 基本的に, リストに登録するのはどのピボットノードでも可能だが, 削除するのは登録したピボットノードにのみが可能とする.

5.3.3 発信規制

ノードに流れてきたパケットの宛先を見て, それが発信規制リストへ向かうパケットであった時, そのパケットをある確率で棄却する.

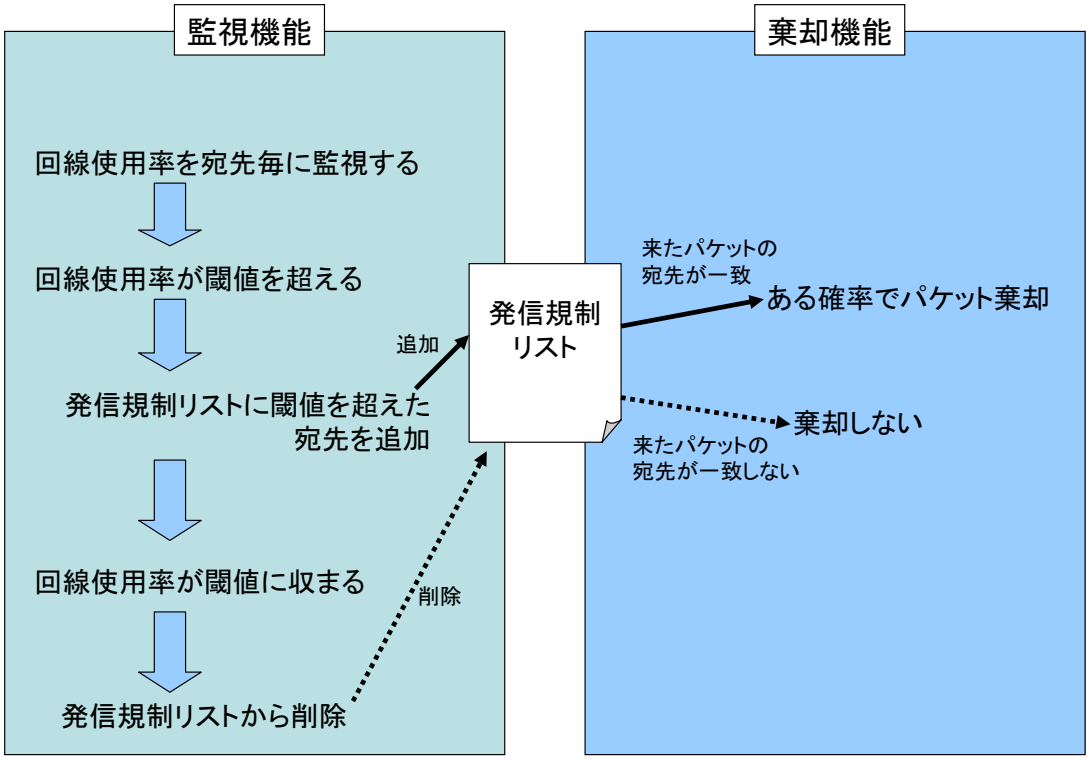
監視機能
回線使用率を宛先毎に監視する
回線使用率が閾値を超える
発信規制リストに閾値を超えた
宛先を追加
回線使用率が閾値に収まる
発信規制リストから削除
追加
削除
発信規制
リスト
棄却機能
来たパケットの
宛先が一致
ある確率でパケット棄却
棄却しない
来たパケットの
宛先が一致しない

図 5.3: 発信規制の流れ

第6章　シミュレーションによる実験

6.1　概要

ここでは, 提案した新しい輻輳制御の効果を検証する. 提案手法のシミュレーション及び, 従来手法のシミュレーションを行い, それらの結果を比較する.
まず,

- 提案手法の実装法
- シミュレーションに用いるネットワークモデル
- 実験条件など

を示す. そして, 提案手法及び比較対象となる従来手法のシミュレーション結果を示す.

6.2　提案手法の実装

提案手法の効果を検証するために, カリフォルニア大学バークレイ校で開発されたネットワークシミュレータである NS2 に提案手法を組み込んだ. 組み込んだ提案手法の概要は以下の通りである.

- 提案手法はキューとする.
- 発信規制リストを作成する. このリストは他のノードからも見える様にする.
- 提案手法は到着したパケットのサイズと到着時間を宛先別で配列に格納する.
- 回線使用率の測定間隔は予め設定しておき, 測定時に回線使用率を計算する.
- 回線使用率の計算は, 配列に最後に格納された情報から順に最初に格納された情報に向かって順にパケットサイズを加算していく. その際に到着時間も見ておき, 回線使用率の測定間隔分加算されたら, 加算されたパケットサイズをリンクの帯域幅と回線使用率を見る時間幅で割り, 回線使用率を算出する. まだ開始後間も無く, 時間幅分経っていない場合は, 時間幅の代わりにそれまでの時間で割る.
- 回線使用率が閾値を超えた時, その宛先のアドレスを発信規制リストに加える.

- 到着したパケットの宛先が発信規制リストにある場合は, ある棄却率でそのパケットを棄却する.
- 規制がかかっている状態で回線使用率が閾値を下回った時, 発信規制リストから規制をかけていた宛先アドレスを削除する.
- もし, 規制をかけているにも関わらず回線使用率が上昇し, 閾値 + (100 − 閾値)/2 を超えた時, 第二次規制として, 棄却率を (100 − 棄却率)/2 だけ引き上げる.

6.3 ネットワークモデルの定義

図 6.1 に示すネットワークモデルを用いてシミュレーションを行う. 図において, 丸のノードはパケットを送信, 受信を行うノードを, 四角のノードはピボットノードを表す. ノード 1, 2 は UDP でノード 7 にパケットを送信し, ノード 4 は RTP でノード 7 に送信する. これらはパケット集中の原因となる通信である. ノード 0 は TCP で, ノード 6 に送信する. これは, パケット集中とは関係の無い通信である. 各々のノード, 及びリンクの性能については表 6.1 にまとめる.

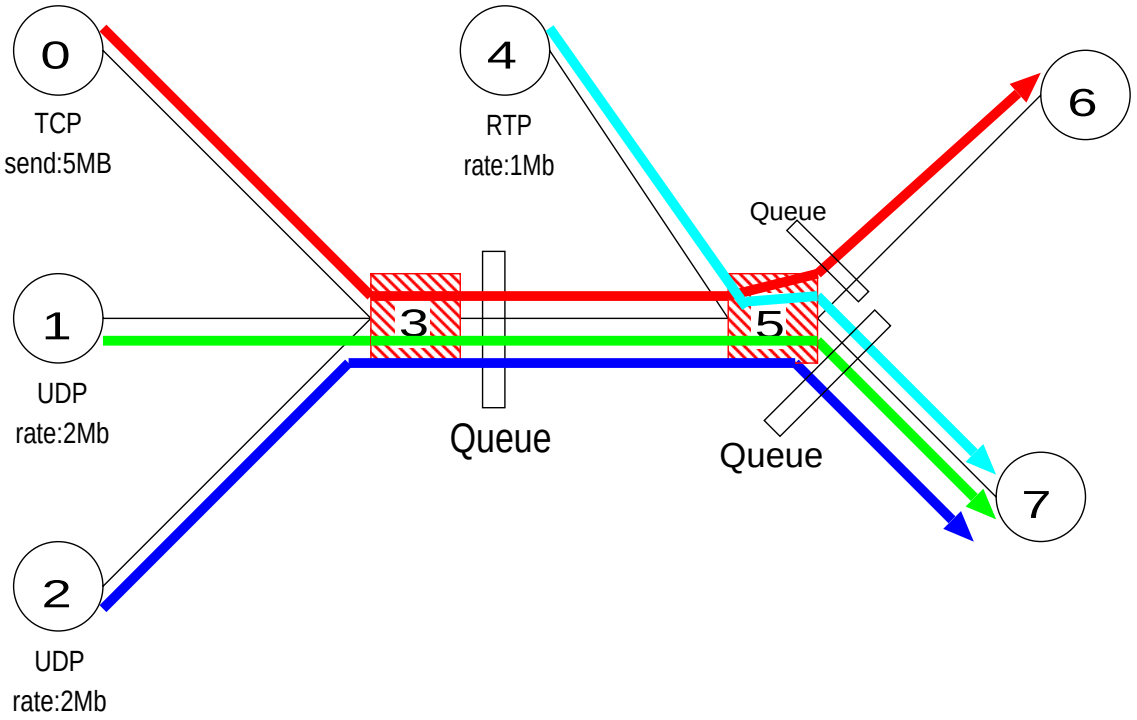

図 6.1: シミュレーションに用いるネットワークモデル

ノード	リンク	帯域	遅延	キュー
0–3	全二重	3Mbps	3ms	Droptail
1–3	全二重	3Mbps	3ms	Droptail
2–3	全二重	3Mbps	3ms	Droptail
3–5	全二重	5Mbps	3ms	提案手法
4–5	全二重	3Mbps	3ms	Droptail
5–6	全二重	3Mbps	3ms	提案手法
5–7	全二重	3Mbps	3ms	提案手法

表 6.1: ネットワークモデルの性能

6.4 実験方法と実験条件

提案方式の効果を検証するため, コンピュータによるシミュレーションを行う. ネットワークモデルとして, 6.3 節で示したモデルを用いる. 今回のシミュレーションでは, 比較対象として, 提案手法と同じネットワーク層での輻輳制御である Droptail, RED を採用する.

以下に, 実験方法と条件を示す.

- NS2 に提案手法を組み込み, シミュレーションを行う.
- 送信シナリオは表 6.2 に示す.
- シミュレーション時間は全部で 80 秒とする.
- 発信規制のための回線使用率の閾値は 90%とする.
- 発信規制時のパケット棄却率は 5%とする.

実験に用いたプロトコルは, ノード 0 からノード 6 への送信のみが TCP で, パケット集中には関係の無い通信である. これは, 混雑に対して過敏に反応し, 自らの送信ウインドウをすぐに減らしてしまう TCP の性質から輻輳制御の影響をみるのに丁度良いと考えたためである. 更に, TCP Tahoe を用いており, タイムアウトが起こってしまうと一気に伝送効率が落ちてしまうため, なるべくタイムアウトさせない様な制御を行う必要が有る. パケット集中に関係するその他のプロトコルは, ネットワークの状態に関係無くパケットを送信する性質から, パケット集中による輻輳を起こすのに最適であり, 集中している通信をどう制御するかを見る事が提案手法の効果を見るのに適切だと考えられる.

送信元	宛先	送信プロトコル	送信パラメータ	開始時間 (秒)	終了時間 (秒)
0	6	FTP	5MByte	0	——
1	7	UDP	2Mbps	3	80–
2	7	UDP	2Mbps	6	80–
4	7	RTP	1Mbps	9	70

表 6.2: 実験条件

6.5 評価方法

シミュレーションの結果生成されるトレースファイル, ログファイルからそれぞれの回線使用率などのデータを抽出し, 従来手法との比較を行う. 評価項目は,

- TCP の通信終了時間
- TCP の通信におけるパケット棄却数

とする. TCP の通信終了時間の比較から, パケット集中の原因ではない通信が受ける影響を評価する. また, パケット棄却数の比較から, 提案手法による輻輳制御の効果を示す.

6.6 結果

6.6.1 ノード 5–7 での回線使用率

ノード 5–7 での回線使用率とパケット棄却数を図 6.2–6.7 に示す.
図 6.2, 6.3 は提案手法, 図 6.4, 6.5 は Droptail, 図 6.6, 6.7 は RED である.
グラフはそれぞれ, 縦軸は回線使用率もしくはパケット棄却数, 横軸は時間 (秒) を表す. 凡例については, $node1, 2, 4$ はノード 1, 2, 4 から送られてきたパケットが占める回線使用率またはパケット棄却数を, $total$ はノード 5–7 全体の回線使用率を表している. なお, 提案手法のグラフに関してのみ追加のパラメータとして, $proposal7$ は提案手法で監視している回線使用率を, $threshold$ は発信規制のための回線使用率の閾値を, $regulation7$ は 110 の所では第一次の発信規制を, 120 では第二次の発信規制が行われている事をそれぞれ表している. これは, 回線使用率とは関係は無く, 単純に発信規制が行われている事を表す.

図 6.2 より, まず提案手法に関して見ると, ノード 1, 2 からは 2Mbps の通信が, ノード 4 から 1Mbps の通信が行われているため, ノード 4 の通信が始まる 9 秒から, ノード 5–7 のリンクでの回線使用率は 100%になってしまう. それに対し, 閾値は 90%に設定しているので, 発信規制がノード 4 の通信が終わる 70 秒までかけられている. よって, 図 6.3 よりパケット棄却も 9 秒から 70 秒までの間中ずっと行われており, 特に第二次規制がかかっている 30 秒手前では多くのパケットが棄却されている.

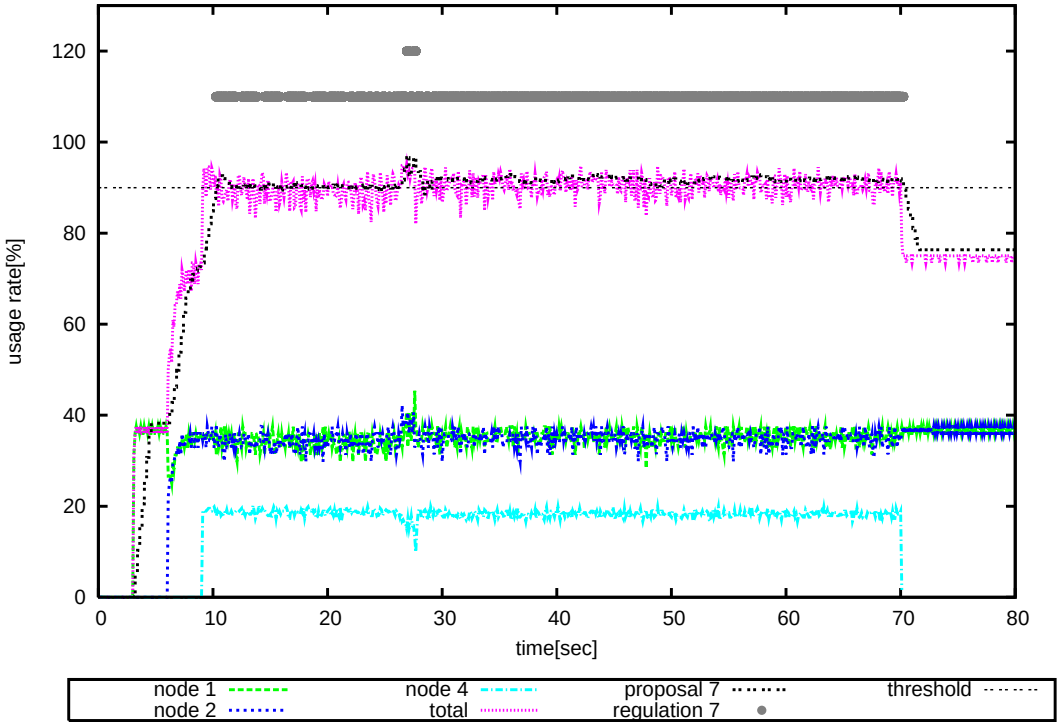

図 6.2: ノード 5–7 での回線使用率 (提案手法)

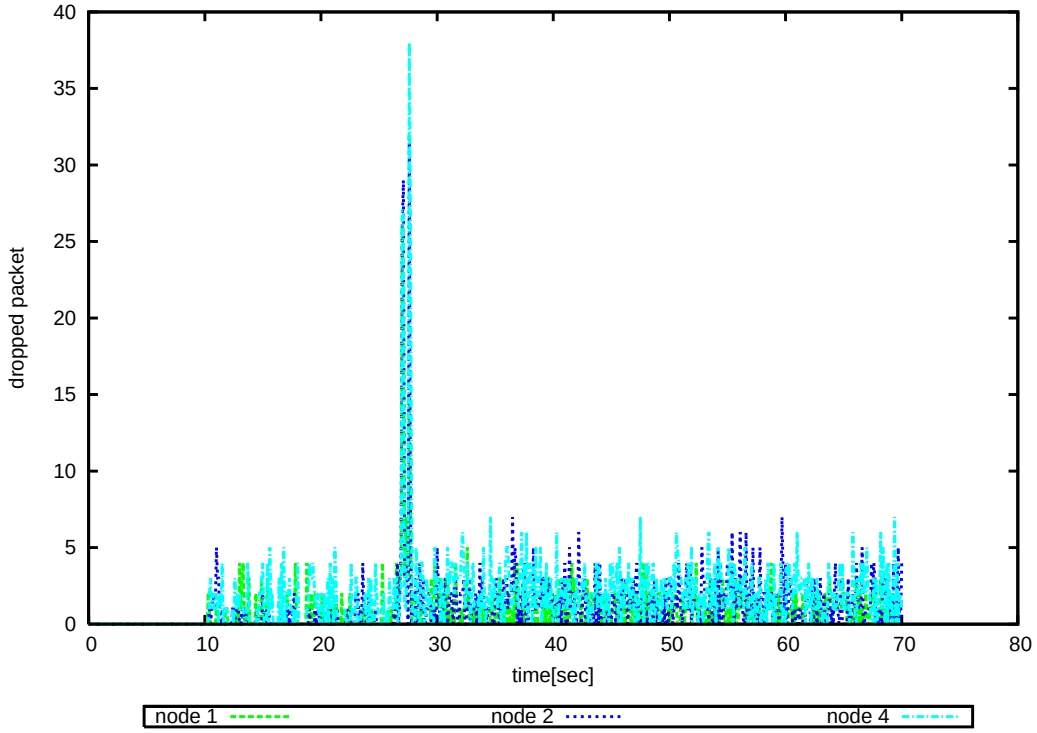

図 6.3: ノード 5–7 でのパケット棄却数 (提案手法)

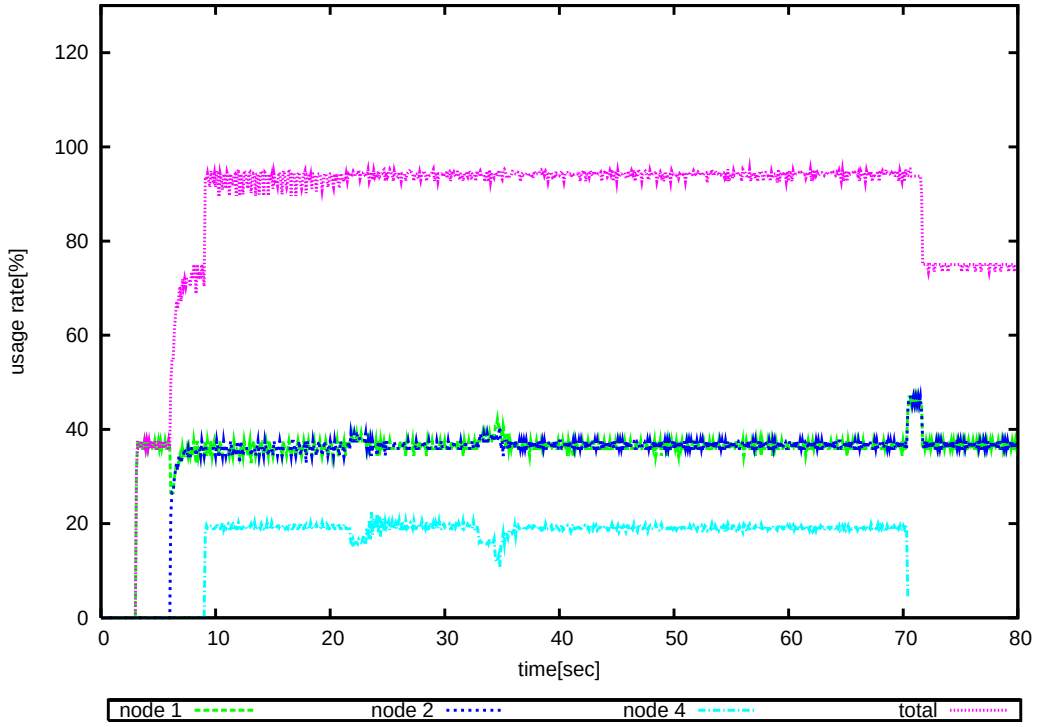

図 6.4: ノード 5–7 での回線使用率 (Droptail)

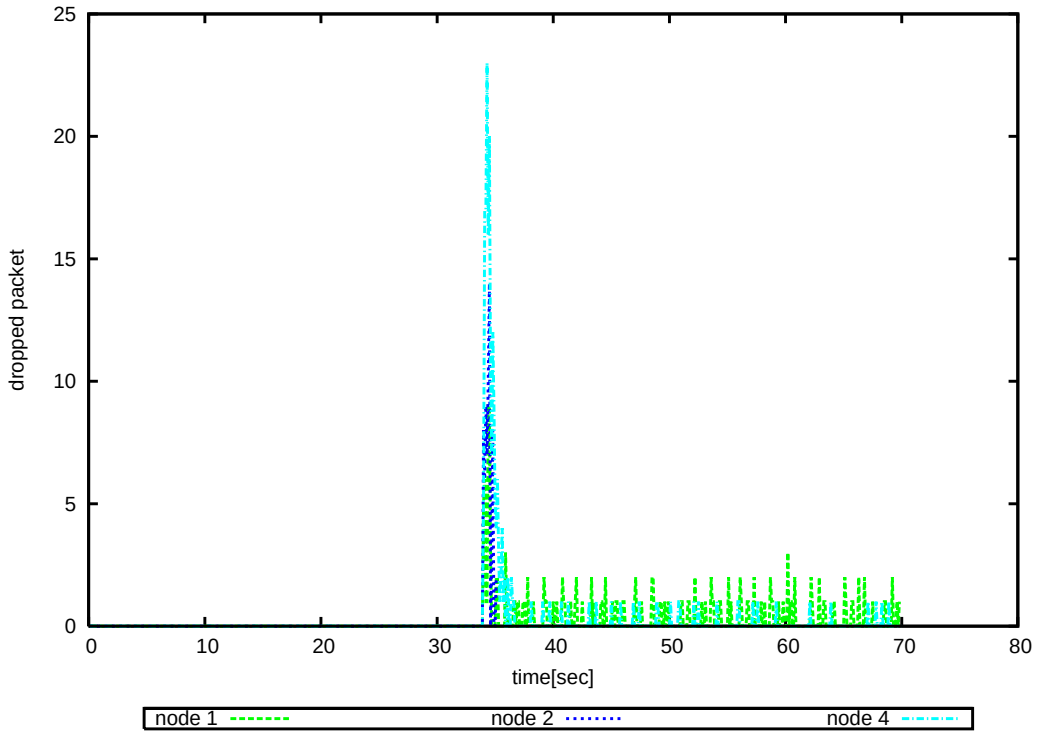

図 6.5: ノード 5–7 でのパケット棄却数 (Droptail)

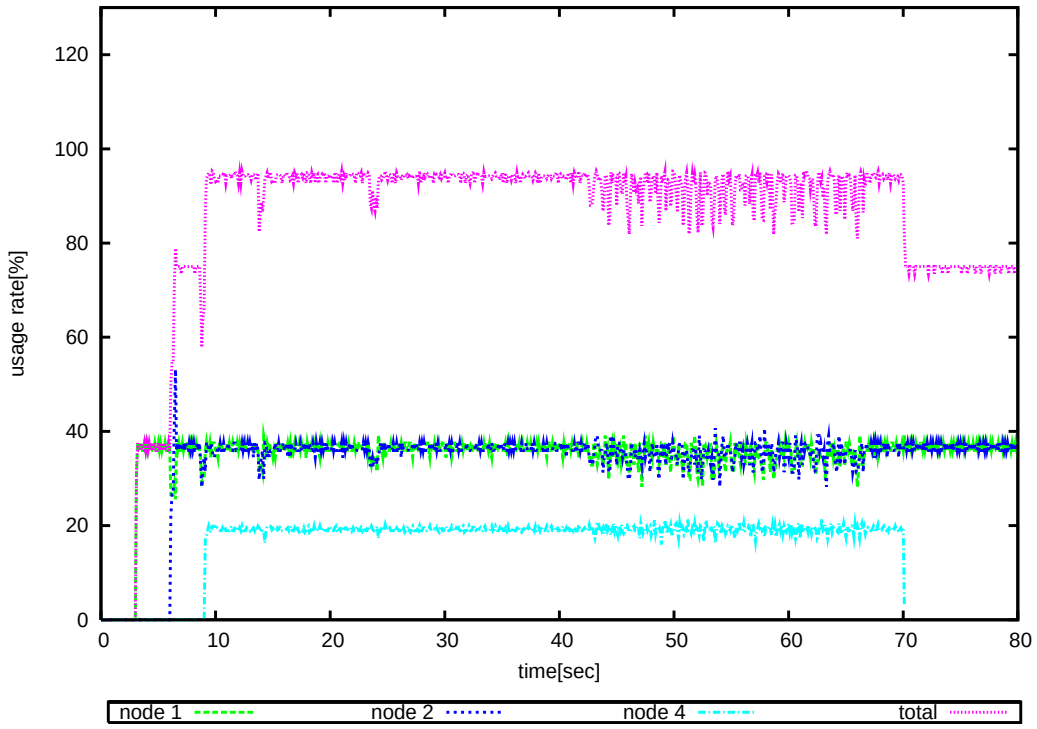

図 6.6: ノード 5–7 での回線使用率 (RED)

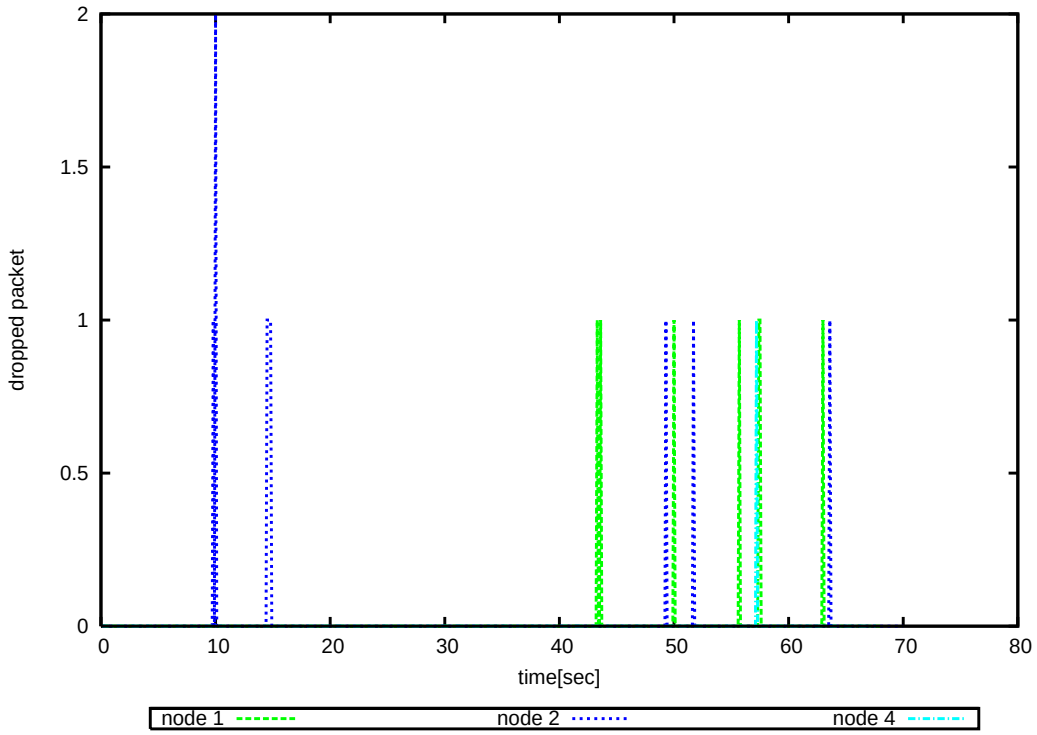

図 6.7: ノード 5–7 でのパケット棄却数 (RED)

Droptail, RED を見ると, 図 6.4, 6.6 より, 回線使用率はどちらも 90%を超える値で推移しており, 図 6.5, 6.7 より, パケット棄却数も提案手法と比べ大幅に少なく, ノード 5–7 に関して言えば提案手法よりも効率の良い通信が行われていると言える.

6.6.2 ノード 3–5 での回線使用率

ノード 3–5 での回線使用率を図 6.8–6.10 に示す.

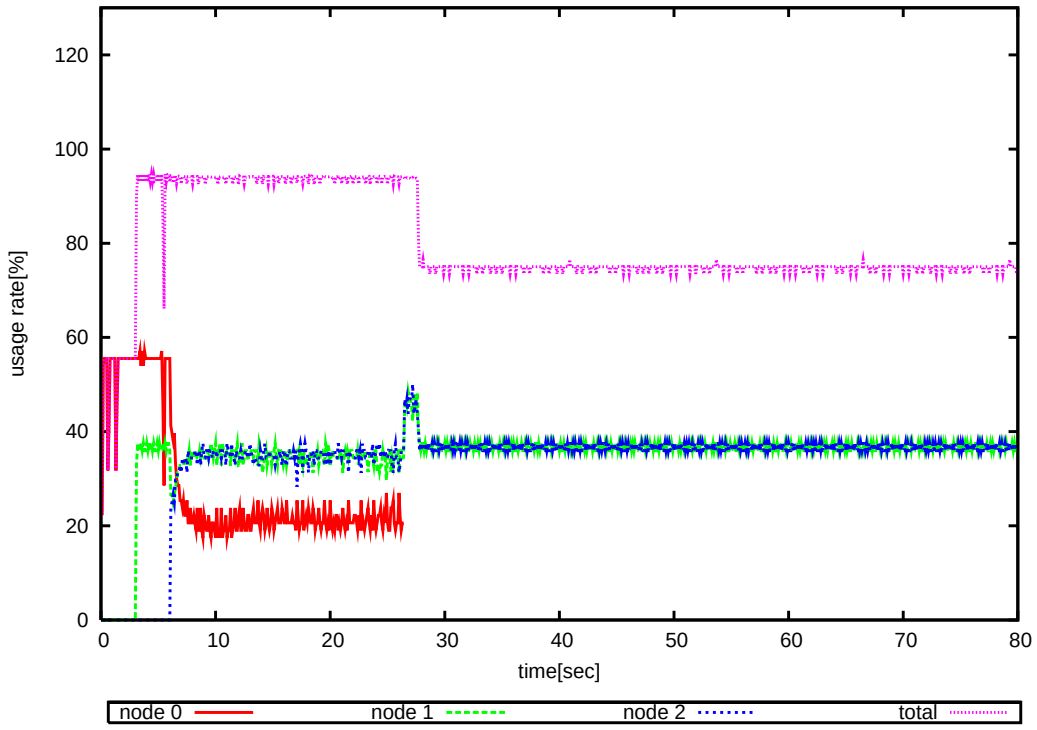

図 6.8: ノード 3–5 での回線使用率 (提案手法)

図 6.8 は提案手法, 図 6.9 は Droptail, 図 6.10 は RED である.

グラフはそれぞれ, 縦軸は回線使用率 (%), 横軸は時間 (秒) を表す. 凡例については, *node*0–2 はノード 0–2 から送られてきたパケットが占める回線使用率を, *total* はノード 3–5 全体の回線使用率を表している.

図 6.8 より, 提案手法では, ノード 5 においてノード 7 に向かう通信の回線使用率を 90%までに抑えている. 更に, ノード 3 においては輻輳は起きていないが, ノード 5 において輻輳を検知しているため, ノード 3 でもパケット棄却が行われている. このため, パケット集中とは関係の無い TCP の通信は影響を受ける事なく通信が出来ており, 他の手法と比べて最も早く通信を終える事が出来ている.

図 6.9 より, Droptail は, その性質上全ての通信に対して平等にパケット棄却が行われる. パケット集中に対して関係の無い通信に対してもパケット棄却を行うので, どうして

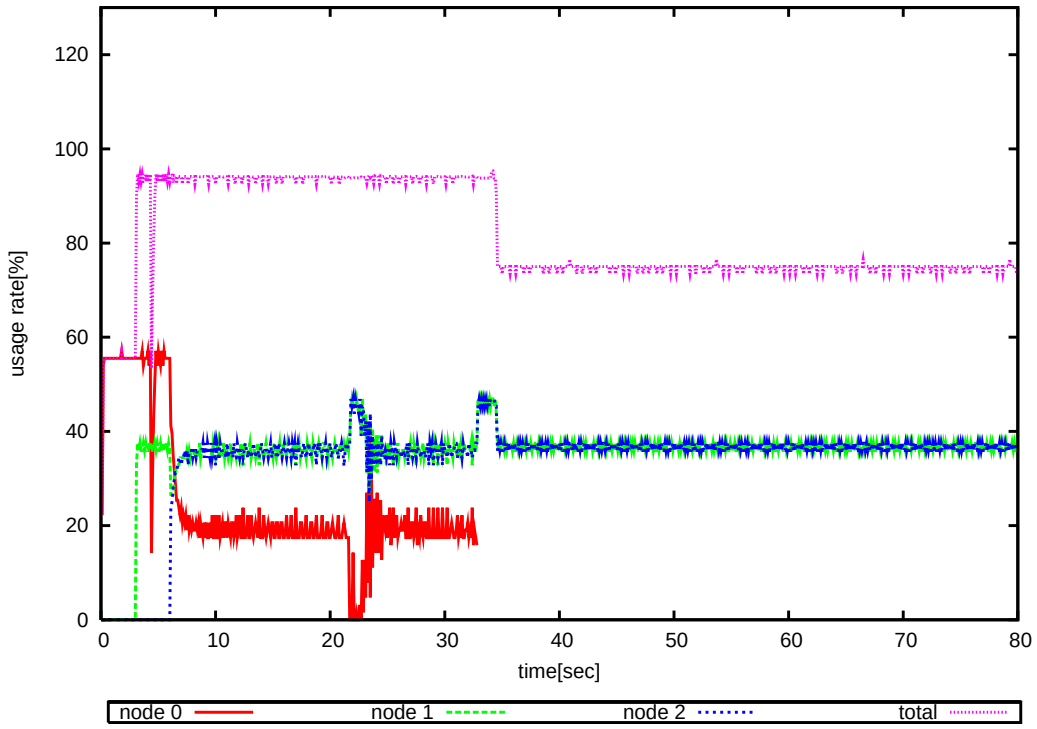

図 6.9: ノード 3–5 での回線使用率 (Droptail)

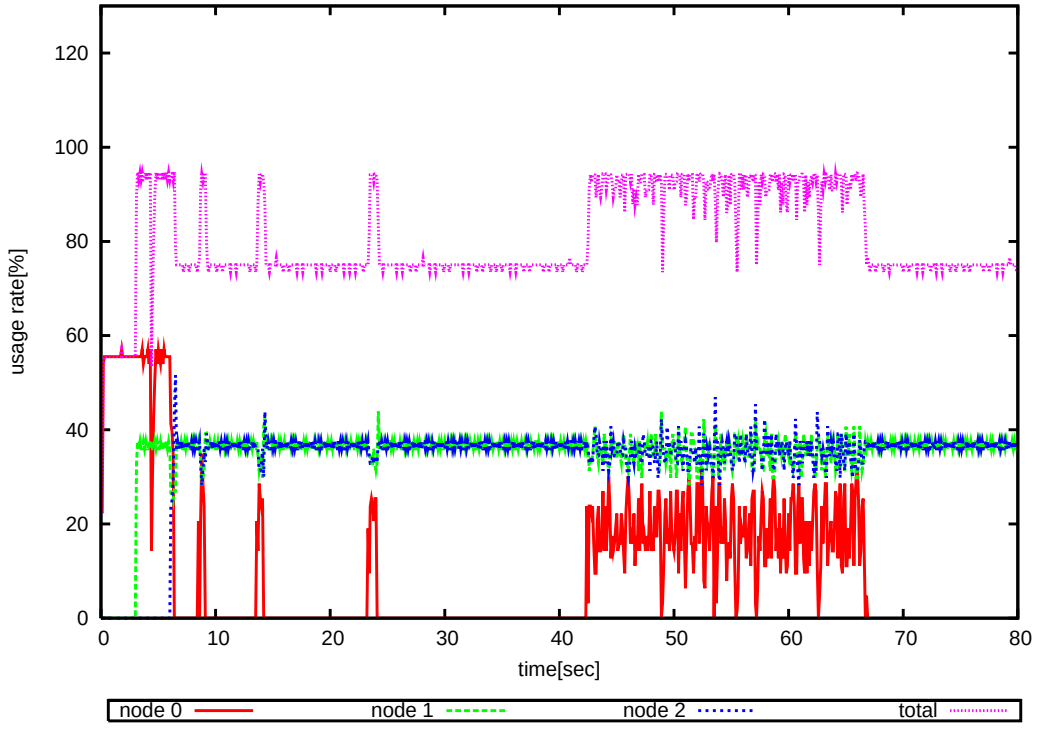

図 6.10: ノード 3–5 での回線使用率 (RED)

もパケットロスが起きてしまう. 20 秒の辺りでパケット棄却が起こっており, その数はごく少数であるにも関わらず, TCP のパケットを棄却してしまっているため, 提案手法よりも TCP の通信効率は悪くなってしまう.

図 6.10 より, RED は, 本研究で重要視する通信効率で言うと, 完全にワーストケースだと言える. 常に一定の回線使用率を得ようとする UDP に, TCP の通信が入ろうとすると, RED の監視するキューの閾値を超えてしまい, パケット棄却が行われる. すると, TCP では輻輳が起きていると判断され, パケット送信を待つ. これを輻輳と判断しない程度にウインドウサイズが落ち着くまで繰り返し, 適当な通信を行おうとする. このため, TCP の通信は大幅に遅れてしまい, パケット集中に対して全く関係が無いのにも関わらず, 輻輳制御されてしまっている.

6.6.3 ノード 5–6 での通信

輻輳に対して敏感に反応する TCP の通信に関してより詳しく調べる. ノード 5–6 の回線使用率と輻輳ウインドウサイズを図 6.11–6.13 に示す.

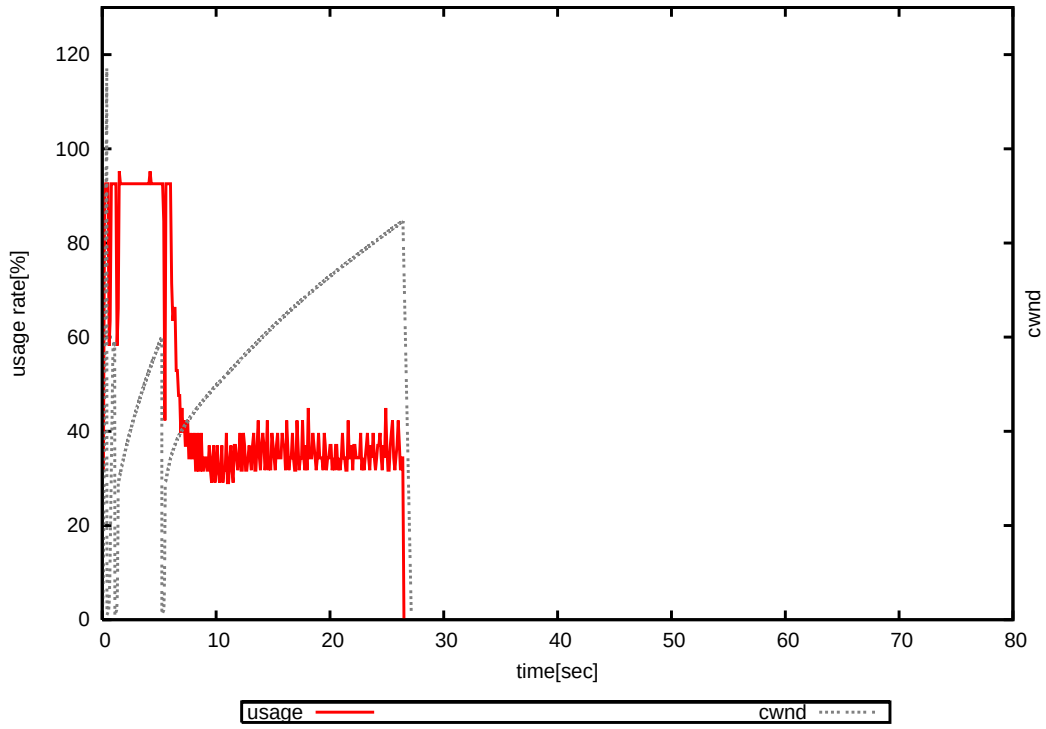

図 6.11: ノード 5–6 での通信 (提案手法)

図 6.11 は提案手法, 図 6.12 は Droptail, 図 6.13 は RED である.

グラフはそれぞれ, 縦軸は回線使用率 (%), 輻輳ウインドウサイズ (セグメント) を示し, 横軸は時間 (秒) を示している. 凡例は順に, 回線使用率, 輻輳ウインドウを示している.

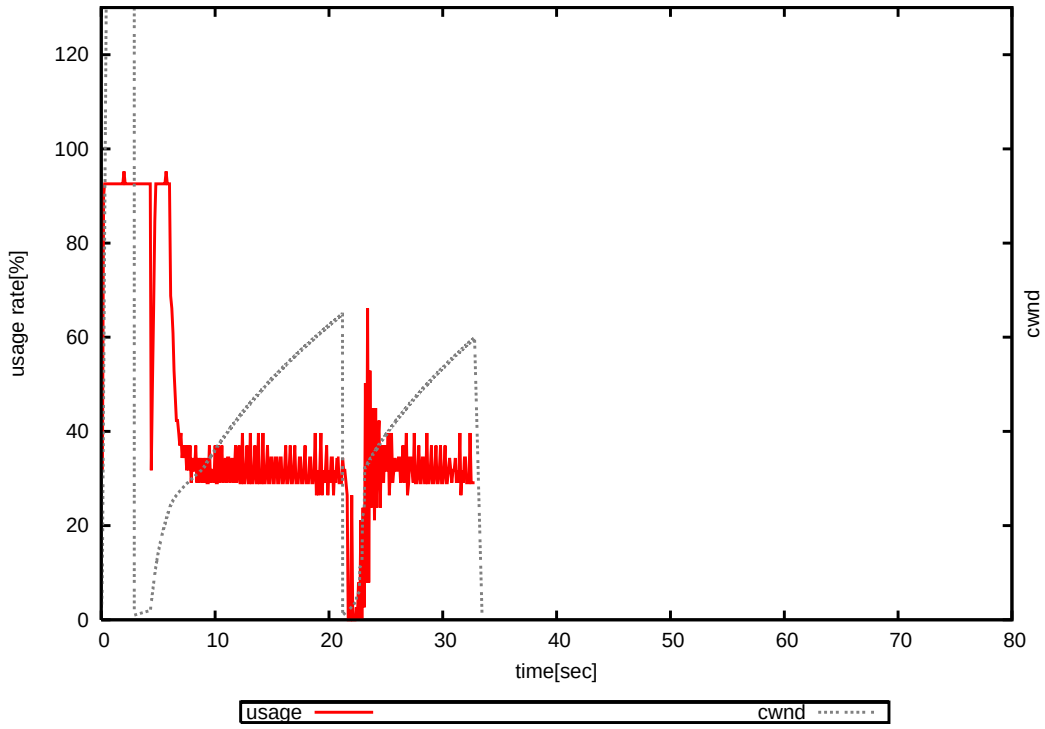

図 6.12: ノード 5–6 での通信 (Droptail)

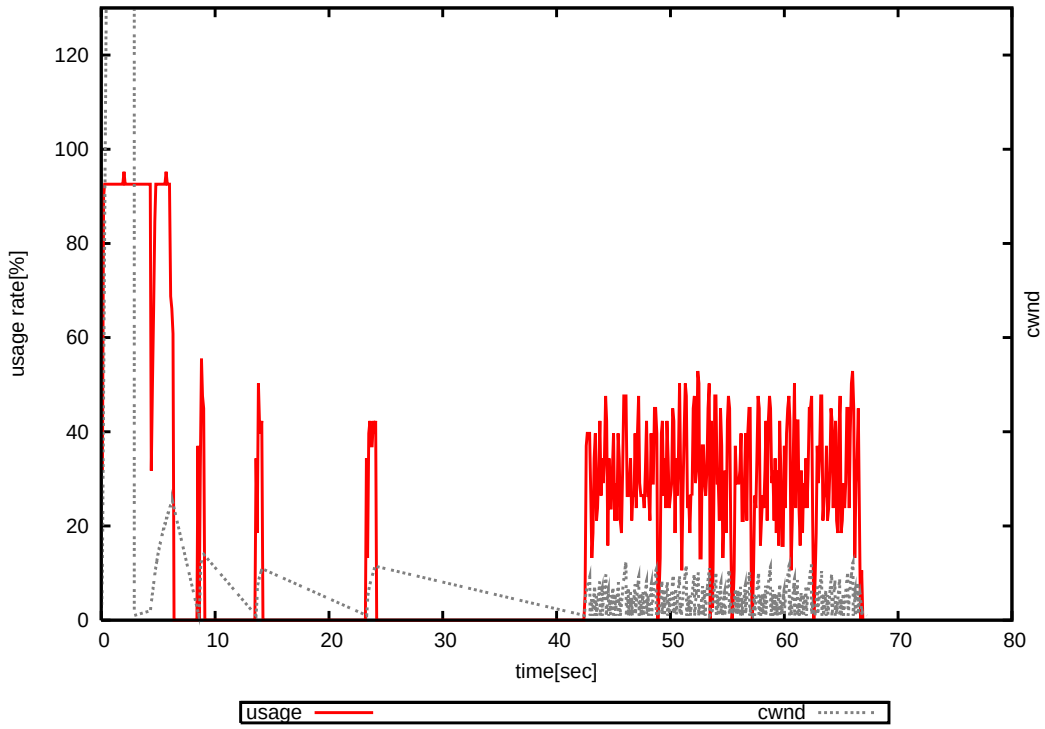

図 6.13: ノード 5–6 での通信 (RED)

図 6.11 より, 提案手法では, 5 秒以降は輻輳ウインドウが 1 に初期化されていない (3.1.4 節参照). これは, パケットロスが起きていない事を意味しており, ノード 7 へのパケット集中の影響を受けずに効率の良い通信が行われている事が分かる.

図 6.12, 6.13 より, Droptail, RED では, 両方共輻輳ウインドウサイズの初期化が提案手法よりも多く, 何度もパケットロスが起きていると分かる. こちらは逆にノード 7 へのパケット集中の影響を受けてしまっている事を意味する. RED に関してはその度合いの大きさが顕著であると言える.

6.6.4 それぞれの手法での通信終了時間

最後に, 提案手法, Droptail, RED, における TCP の通信終了時間とパケット棄却数を図 6.14, 6.15 に示す.

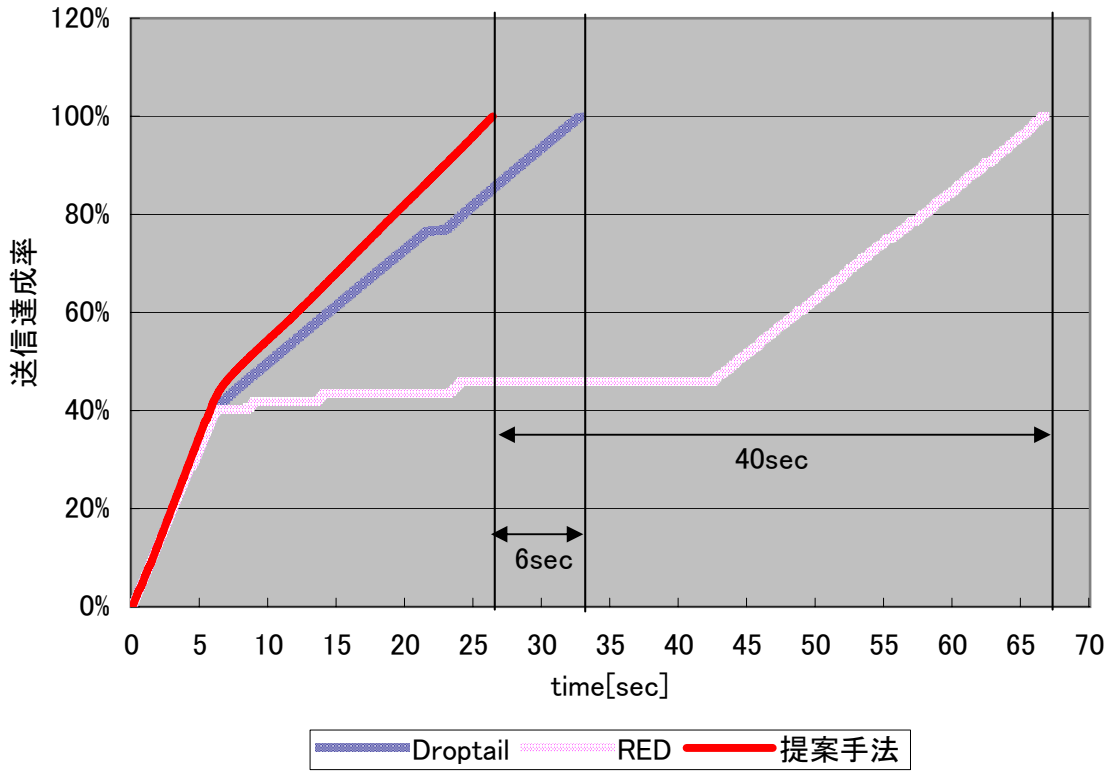

図 6.14: 手法別での TCP の通信終了時間

図 6.14 より, TCP の終了時間に関しては, 提案手法が最も早く終えている事が確認出来る. Droptail よりも 6 秒早いため, 提案手法の効果は高いと考えられる.

また, 図 6.15 より, パケット棄却数に関しても提案手法が最も少なく, 殆どパケット集中に対する輻輳制御の影響を受けていない. これは, 輻輳の原因となっている通信に対し

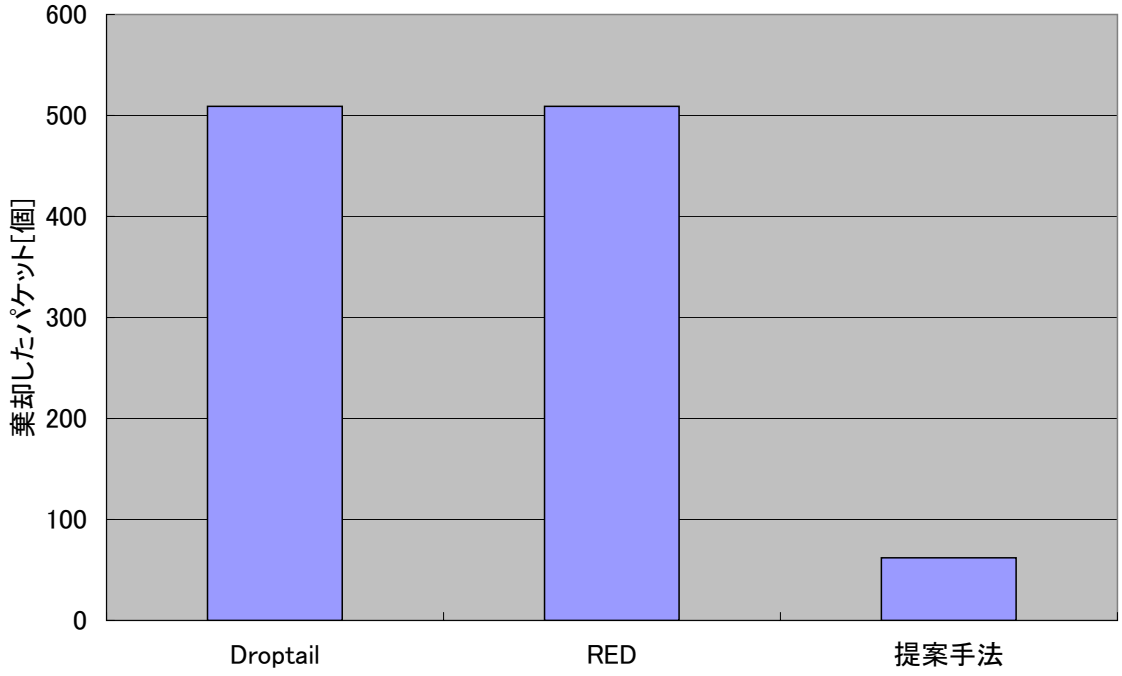

図 6.15: 手法別での TCP のパケット棄却数

てのみピンポイントでパケット棄却を行えており, 図 6.8–6.10 に示したノード 3 での回線使用率, 図 6.2–6.6 に示したノード 5 での回線使用率を見ても, 従来手法とは大して変化の無い事から, 効果の高い輻輳制御を行えていると言える.

第7章　考察

IP ネットワークに, キャリアで用いられている発信規制の概念を適用した新しい輻輳制御の手法を提案した. 提案手法の効果を検証する為にコンピュータによるシミュレーションを行い, 従来手法との違いを示した.

提案手法の効果を確認する為, 1 つのノードにパケットが集中する様なネットワークモデルを定義し, パケット集中の原因ではない通信が受ける影響と, 回線使用率を監視する輻輳制御の効果を調べるシミュレーションを行った. その結果, 提案手法は従来手法と比べ,

- パケット集中の原因ではない通信は影響を受けず, 早く通信を終える事が出来た.
- パケット集中の原因となる通信に関わるパケットに対して主に棄却を行うため, 原因ではない通信に対するパケット棄却数は非常に少ない.

提案手法は, 社会現象によって 1 箇所にパケットの集中が起こる様な場合に効果の有る手法であり, 回線使用率を監視する事で, より正確な輻輳制御) を実現する事が可能であると言える.

第8章　まとめ

8.1　本研究のまとめ

本研究では, IP ネットワークにおいて, 社会現象が要因となる 1 つの宛先へのパケット集中が原因となる輻輳の解決を目的とし, キャリアの発信規制の概念を適用した新たな輻輳制御の提案を行った. その効果を検証するため, NS2 によるネットワークシミュレーションを行い, 従来手法との比較を行った.

その結果, 提案した発信規制の概念を用いた輻輳制御は, 1 つの宛先にパケットが集中する事によって引き起こされる輻輳に対して, 他の宛先への通信には影響せずに輻輳制御する事が可能である事を示した. 社会現象によって引き起こされる様な 1 箇所への大量のパケット集中に対して従来手法よりも効果があり, 他への通信は阻害されないと考えられる. 更に, 回線使用率を監視する事で, キューを監視する従来手法と比べ, 輻輳制御をより効率化する事が出来た.

8.2　課題

今回, シミュレーションで示したのは比較的小規模なモデルだった. これが大規模なネットワークモデルになった場合に提案手法で同じ様に制御が可能なのかは明らかでは無い. 今後, より実世界に近いネットワークでシミュレーションを行い, その効果を検証する必要が有ると考えられる. 大規模なネットワークの場合, 1 つの宛先へのパケット集中による輻輳がいくつも予想されるので, 規制する宛先が複数の場合でも制御が可能なのかを確かめる必要がある.

謝辞

本研究を進めるにあたり, 日々熱心に指導をして下さった日比野靖教授に深く感謝致します. 研究に関連するネットワークの事だけでなく, 教育や経済, 雑学に至るまで, 貴重な事を幅広く学ばせて頂きました.

また, 数々の大変参考になる助言を頂いた田中清史准教授, 菅原英子助教に深く感謝致します.

その他, 貴重な意見を頂いた計算機アーキテクチャ講座の皆様をはじめとする, 多くの方々に深く感謝致します.

付 録A　新しい手法のNS2への組み込み方法

A.1　概要

ここでは, 提案手法を NS2 に組み込む方法について述べる.

まず, NS2 のディレクトリ構造に関して述べる. 次に, 組み込み方法に関して提案手法の組み込みを例に挙げ, 順を追って説明する.

A.2　NS2のディレクトリ構造

NS2 はネットワークにおける機能毎にディレクトリが分かれている. それを図 A.1 に示す.

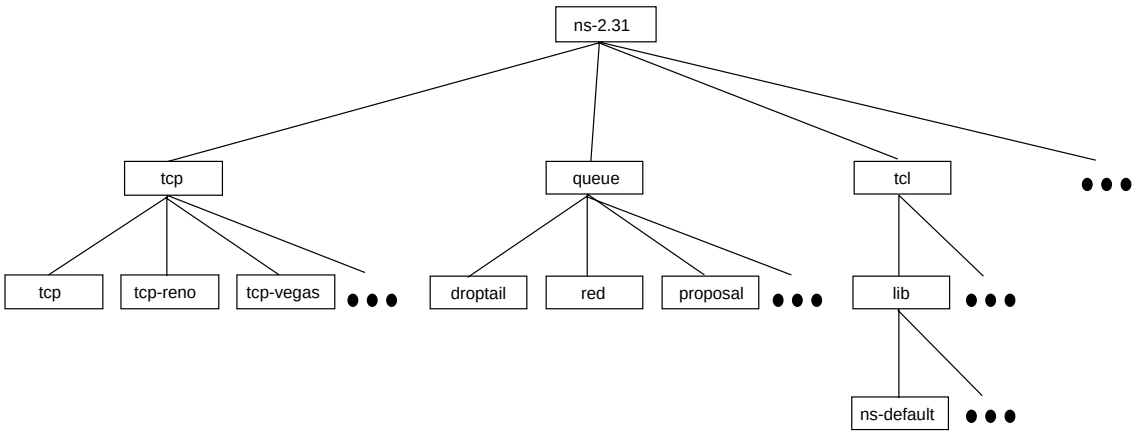

図 A.1: NS2 のディレクトリ構造

それぞれの□はディレクトリを表しており, 末端のものに関しては機能を表している. [7] によると, NS2 は C++のソースコードディレクトリと Tcl のソースコードディレクトリに分かれている. 図 A.1 では, Tcl よりも下のディレクトリは Tcl のソースコードが入っており, その他は C++のソースコードが入っている. tcp の中には tcp(TCP Tahoe), tcp-reno(TCP Reno) など, TCP の機能に関わるソースファイルが有り, queue の中には droptail(Droptail), red(RED) が有る.

A.3　提案手法の組み込み手順

提案手法はキューとして組み込んでいる. ここでは提案手法を組み込む一連の流れを説明する.

A.3.1　ソースコードの場所

[7] によると, 通常は自分で機能を新たに追加する場合には新たに local と言うディレクトリを作成し, その中にソースコードを入れる. だが, 今回の場合は特にファイル数も少なく, 追加する機能も少ないので, queue の中に直接入れた.

A.3.2　ソースコードの作成

今回, 提案手法のソースコードは, Droptail のソースコードを基に作成した. ソースコードに, パケット監視, パケット棄却の機能を入れ, メンバ名が重複しない様に変更した.

A.3.3　変数のバインド

NS2 のシナリオファイルから, パラメータを変化させてシミュレーションが行える様に, シナリオファイルの特定の変数とソースコード内の変数を対応付ける, 変数のバインドが可能である.

提案手法においては, 閾値, パケット棄却率などに関しては, 値を変えてシミュレーションを行う必要が有ったので, それぞれの値に対して変数のバインドを行った.

A.3.4　ns-default.tcl の編集

ns-default.tcl は, ns-2.31/tcl/lib にある Tcl ファイルで, このファイルにバインドされた変数の初期値を記述しておく.

A.3.5　コンパイル

コンパイルに関しては, 予め Makefile が用意されており, 組み込みたい機能のソースコードのファイル名を追加記述する. その後, make を行えば, 機能の追加が完了する.

参考文献

[1] Andrew S. Tanenbaum. “**コンピュータネットワーク第 4 版**”. 日経 BP 社, 2003.

[2] S. Jagannathan. “End to End Congestion Control in High-Speed Networks”. *LCN*, 2002.

[3] Alapati V. S. Reddy Cui-Qing Yang. “A Taxonomy for Congestion Control Algo- rithms in Packet Switching Networks”. *IEEE*, 1994.

[4] Raj Jain. “Congestion Control in Computer Networks: Issues and Trends”. *IEEE Network Magazine*, 1990.

[5] W.Stebens M.Allman, V. Paxson. “TCP Congestion Control”. *RFC2581*, 1999.

[6] Larry L. Peterson Lawrence S. Brakmo, Sean W. O’Malley. “TCP Vegas: New Techniques for Congestion Detection and Avoidance”. *ACM*, 1994.

[7] 銭飛. “NS2 **によるネットワークシミュレーション**”. 森北出版, 2006.