REDの安全弁機構の設計と実装

長 健二朗

フローバルブは，ネットワークをフロー制御されない通信から防護し，かつ，エンド・エンドのフロー制御を促進する目的で，REDに追加される安全弁機構である．フローバルブは，従来ペナルティボックスと呼ばれてきた概念の実装と捉えることもできるが，その目的はあくまでネットワークの防護にある．フローバルブは，REDの制御可能範囲を越えるトラフィックの増加を検出し，過送出フローを強制的にバックオフさせることによって，ルータ上の資源を防護する．フローバルブによって，軽度の輻輳時にはパケット損失率を低く押え，また，重度の輻輳時には慎重にバックオフを行なうというエンド・エンドのフロー制御が奨励される．シミュレーションの結果，フローバルブはネットワークを効果的に防護すると同時に，正規のTCPが起こしうる輻輳を遮閉する効果が確認された．また，実際にFreeBSDカーネルに実装し検証を行なった．

1 はじめに

フローバルブは，ネットワークをフロー制御されない通信から防護し，かつ，エンド・エンドのフロー制御を促進する目的で，RED[3]に追加される安全弁機構である．フローバルブのもととなるアイデアはFloydらによって提案され，REDペナルティボックスという概念で知られてきた[3][4]．REDは，TCPが主なトラフィックであるネットワークの輻輳緩和や利用率向上に効果がある反面，フロー制御されないトラフィックによって簡単に到着パケット廃棄動作(Drop-tail)に戻ってしまう問題がある．フロー制御されないトラフィックを検出し制裁することは，REDを防護するために必要であると同時に，エンド・エンドによるフロー制御を促進することになるため，インターネット全体の安定や拡張性につながる．

Floydらは[4]において，ベストエフォートが前提のインターネットにおけるエンド・エンドによるフロー制御の重要性について論じている．さらに，輻輳時に不当に帯域を消費しているフローを検出して，その帯域使用に制限をくわえる手段の必要性を示した上で，それによって，エンド・エンドのフロー制御が促進されることを説いている．そのなかで，いくつか概念的な実現方法を提案して理論的な検討をしているが，具体的に実装可能なレベルには至っていない．

本研究の目的は，その理論モデルを近似する効率よい実装を行ない，運用ネットワークでの実証を可能にすることである．Floydらも指摘しているように，重要なのは，正確な理論モデルの追求ではなく，システムを構築して実証実験を行ない経験を得ることである．

2 背　景

本提案は非適応型フロー(unresponsive flow)が適応型フロー(responsive flow)を圧迫する可能性のあるトラフィックを制御対象とする．一部のリアルタイム通信の優先制御を行なうネットワークでは，通常ネットワークの入口でポリシングが行なわれるため，非適応型フ

A Safety-valve for RED.

Kenjiro Cho, (株)ソニーコンピュータサイエンス研究所, Sony Computer Science Laboratories, Inc.

コンピュータソフトウェア, Vol.16, No.4(1999), pp.10–22.
[論文]1999年5月30日受付.

ローが適応型フローを圧迫することはない．しかし，本提案は，そのような場合でも，設定に問題が生じて想定外のトラフィック流入が発生する場合等に対する防護手段を提供することができるため，幅広い適用範囲を持つ．

2.1 RED (Random Early Detection)

RED (Random Early Detection) は，平均キュー長に対応した確率でパケットを廃棄することによって，能動的にキューを管理する仕組みである[3]．パケット廃棄確率の計算には，平均キュー長 avg と，min_{th} と max_{th} という 2 つの閾値が使われる．平均キュー長 avg が min_{th} から max_{th} へと増加するに従い，廃棄確率は，0 から最大廃棄確率 max_p へと遷移する．

RED は，キューが破綻する前に輻輳を通知することによって，TCP の同期現象の発生を防ぎ，その結果，帯域利用率も向上する．また，キュー長を短く保てるので，遅延が改善されるなどの効果がある．さらに，ある程度の連続パケット到着を許容するので TCP との親和性がよく，各フローは帯域占有率に比例した確率でパケットを損失するため公平性も確保される．RED は，フロー毎の情報 (per-flow state) を持つ必要がないので，スケーラビリティの点でも有利である．しかしながら，RED には非適応型のフロー (unresponsive flow) に対する防護機能がないため，非適応型のフローによってキュー長が max_{th} を越えると，単なる到着パケット廃棄の動作に戻ってしまう問題がある．

一方，FRED (Fair Random Early Detection) [10]は，RED を改良して，キュー内にパケットがバッファされている各フローの情報を保持し，ある程度の帯域割当の公平性を実現する提案である．FRED によっても，特定のフローが不当に帯域を使用することを防ぐことが可能である．フローバルブは，限られた数のフロー情報を保持するように RED を改良するという点では FRED と共通するが，改良の方向性が異なっている．FRED が帯域利用の少ないフローを優先するのに対して，フローバルブは輻輳の原因となっているフローを遮断する．また，FRED では，悪質なフローに罰則を与える仕組みがないため，エンド・エンドのフロー制御を奨励することにならない．

2.2 RED ペナルティボックス

Floyd らは，RED の考案当初から，悪質なフローを検出することを示唆しており，[4]の中でこの考えを発展させている．以下にその概要をまとめる．

今日のインターネットにおいて，フロー制御を行なわない通信が増加してきていることは，インターネット全体にとって大きな脅威である．輻輳時において，フロー制御される通信が使用帯域を絞る一方で，フロー制御を行なわない通信がその分の帯域を占有するという，著しい不公平を生じる．さらに，フロー制御の欠如は，さまざまな形でネットワークを圧迫し，その連鎖反応で輻輳崩壊 (congestion collapse) に至ることになる．

フロー制御の欠如への対策としては，エンド・エンドのフロー制御を促進する以外にも，WFQ(Weighted Fair Queueing)[14]などのパケット・スケジューリングを普及させることや，料金体系の工夫という手段も考えられる．しかし，この中で唯一エンド・エンドのフロー制御の促進だけが，共有や協力というインターネットの運用の基本原理を培う方向性を持っている．

エンド・エンドのフロー制御を促進するには，フロー制御を行なわないフローを検出してなんらかの制裁を加える仕組みをルータに備えることが必要である．

その手段として 3 つの方法を提案する．第一の方法は，「TCP 親和性テスト」(TCP-friendly test) と呼ばれ，パケット到着量が正規の TCP と同等以下であることを検証する．これは，正規の TCP のスループットの上限が，パケット損失率，ラウンドトリップタイム，パケットサイズから求められることを根拠としている．第二の方法は，「適応性テスト」(unresponsiveness test) と呼ばれ，パケット損失率の上昇に応じて適正にパケット到着量が減少することを検証する．パケット損失によってウィンドウサイズを半減することを前提にすると，パケット損失率が x 上昇すると，パケット到着量は少なくとも $\sqrt{x}$ のオーダーで減少しなければならない．第三の方法は，「不当帯域使用テスト」(disproportionate-bandwidth test) と呼ばれ，特定のフローが他のフローに比べて不当に多くの帯域を使用していることを検出する．

このような概念は，RED に特定なものではないが，[4]の中では RED を使ったモデルが説明に使われてい

る．さらに，[5]では，RED が廃棄したパケットの履歴をもとにフローのパケット到着量を推測する方法も検討されている．

Floyd らは「ペナルティボックス」という表現は使っていないが，不正なフローを検出して制裁を加えるという概念は，「ペナルティボックス」として研究者の間で広く知られるようになる．ペナルティボックスは，概念としては分かり易い簡単なものであるが，その実現は容易ではないことが分かってきた．統計手法をもとにした理論を，実際に実時間で応用するのは容易でない．複雑なアルゴリズムの集合として構成される TCP の挙動を正確に理論解析することは不可能であり，「不当帯域使用」や「TCP 親和性」を理論的に定義することも難しい．また，ネットワーク・トラフィックは激しく変動するため，フローの統計情報を効率良く収集することも困難である．さらに，ルータは各フローに関して限られた情報しか得られないことや，不正なフローをいかに制裁すればよいかという問題もある．

2.3 高パケット損失率での TCP の挙動

Floyd らは，TCP の挙動解析をもとにペナルティボックスを提案したが，それ以外でも TCP の解析研究が進んできている[2][4][6][11][13][15]．なかでも本研究に関連するのは，RED が能動的にパケット廃棄を行なっている状態での TCP の挙動である．

TCP は，パケット損失率が 1% 以下程度であれば，高速再送 (Fast Retransmit) アルゴリズムによってスループットを維持できる．ところが，パケット損失率が増加するにしたがい，高速再送が失敗するようになり，再送タイムアウトによる再送が増えてくる．

高速再送を前提とした TCP の定常状態モデルは，平均パケット損失率 p に対する輻輳ウィンドウ W を与える[4][13][15]．

$$W = \sqrt{\frac{8}{3bp}} \qquad (1)$$

ここで，b は 1 個の確認応答 (ACK) により確認されるパケット数で，ほとんどの TCP の実装では 1 パケットおきに確認応答を送出するため，通常 $b = 2$ である．

式 (1) は，p が増加するにしたがい，W が $1/\sqrt{p}$ の割合で減少することを示す．図 1はこれを $b = 1$ と $b = 2$ の場合について図示したものである．ところが，TCP の高速再送アルゴリズムは少なくとも 3 個の重複確認応答を必要とするため，輻輳ウィンドウが 3 パケット以下になると，TCP はスループットを維持できなくなる．実際には，パケット廃棄は等間隔では発生しないことと，輻輳ウィンドウが $W/2$ と W の間で振動することを考慮すると，TCP がスループットを維持するためには，W が 8 以上であることが必要となる．図 1より，$b = 2$ の場合，p が 2% を越えると $W < 8$ となる．したがって，p が 2% を越えると一般の TCP のスループットは著しく低下することが分かる．

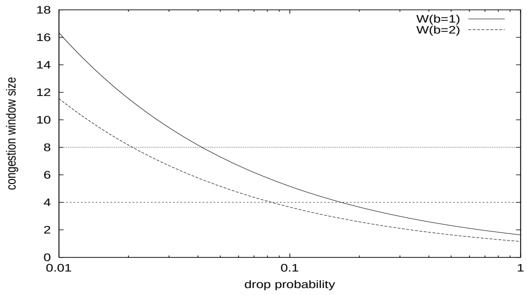


図 1 TCP の定常状態モデルにおけるパケット損失率と輻輳ウィンドウサイズ

Mathis らは，p の増加に伴って再送タイムアウトによる再送が支配的になってくる状態遷移について解析している[13]．ここでは，複数の TCP の実装において，スループットの低下が $p < 1\%$ で始まることが確認されている．

一方，Kumar はマルコフ再生過程をもとにした確率モデルを使い，パケット損失の多い無線リンクにおける TCP の挙動解析を行なっている[9]．ここでも，TCP のスループットの低下が $p < 1\%$ で始まることが示されている．

これらの解析はいずれも，p が 1% を越えると，一般に使われている TCP のスループットが急激に低下するという点で一致する．これは，再送タイムアウトが発生し，輻輳ウィンドウが 1 パケットに設定され，スロースタートを実行するためである．また，再送タイムアウトが繰り返されると，指数バックオフを実現するため，再送タイムアウト値は倍々に増やされる．

表 1　異なるパケット損失率下での **Reno TCP** の挙動

drop rate	pkts/sec	rexmits/sec	timeouts/sec
0.0025	180.90	0.40 (0.2%)	0.01 (0.0%)
0.0050	173.06	0.69 (0.4%)	0.03 (0.0%)
0.0075	153.46	1.15 (0.7%)	0.09 (0.1%)
0.010	140.92	1.55 (1.1%)	0.07 (0.1%)
0.025	89.83	2.37 (2.6%)	0.34 (0.4%)
0.050	55.72	2.75 (4.9%)	0.68 (1.2%)
0.075	38.28	2.87 (7.4%)	1.04 (2.7%)
0.10	26.43	2.86 (10.8%)	1.27 (4.8%)
0.25	4.83	1.26 (26.1%)	0.82 (17.0%)
0.50	0.43	0.18 (41.9%)	0.15 (34.9%)

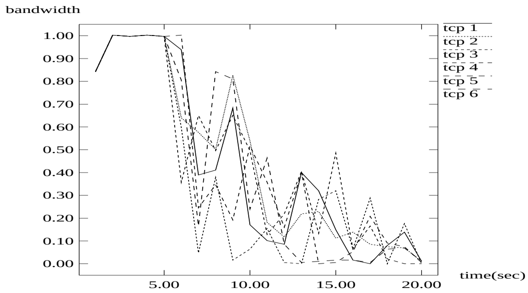


図 2　単一 **TCP** セッションの使用帯域の変化

再送タイムアウトにより，TCP はかなりの時間を損失することになる．再送タイムアウト値は，ラウンドトリップタイムとその平均偏差から計算されるが，実装に用いられている粗粒度タイマーがタイムアウトの期間に大きく影響している．例えば，BSD の実装では，500 ミリ秒のタイマー粒度で，かつ，最小再送タムアウト値は 2 タイマー粒度分に設定されるため，最小再送タイムアウト期間は平均 750 ミリ秒になる．つまり，再送タイムアウトに要する時間は高速再送が成功した場合と比べて桁違いに大きい．

再送タイムアウトは，TCP の挙動に多大な変動を加えるため，RED ペナルティボックスでの挙動テストをする際に大きな問題となる．RED ペナルティボックスでは統計的なモデルを使ったテストを提案しているため，十分なサンプル数が揃わないとテストを適用できない．しかしながら，再送タイムアウトによって，変動量が大きくなるとさらに多くのサンプル数が必要になる上に，パケット到着数が大幅に低下するため時間当たりのサンプル数が激減する．

表 1と図 2は，再送タイムアウトによる変動を示すシミュレーション結果で，*ns* シミュレータ[12]を用いてパケット損失率に対する TCP の挙動解析を行なったものである．ここでは，Reno TCP を用いているが，NewReno でもほぼ同様の結果が得られる．設定の詳細は 5章で示すが，有効帯域は 1.5Mbps，ラウンドトリップタイムは 56 ミリ秒となっている．*ns* シミュレータのタイマー粒度は 100 ミリ秒であるため，再送タイムアウトの平均最小値は 150 ミリ秒であり，一般の TCP より再送タイムアウト時間が短いことに注意されたい．

まず，パケット損失率に対するパケット到着数の変化を示す．表 1は，異なるパケット廃棄確率のもとで 100 秒間，単一の TCP セッションを観測した結果である．図中の値は，総パケット数，総再送パケット数，再送タイムアウト数を単位時間あたりの発生数に換算した値である．パケット廃棄率の増加に伴って，総パケット数が急激に減少するのがわかる．このことは，統計的手法を適用可能なサンプル数を得るのに非常に長い時間を必要とすることを示している．

次に，異なる TCP セッション間の挙動のばらつきを示す．図 2は，単一 TCP セッションの単位時間のスループットを，パケット廃棄率を増加させながら 6 回観測したものである．パケット廃棄確率は 5 秒ごとに，0%，2.5%，5%，10% と 4 区間に分けて変化させている．図からわかるように，スループットの変動は，全体としてはルールに従っているが，個々のセッションの挙動にはおおきなばらつきが観測される．この結果は，個々の TCP セッションに統計的手法を適用するのは，この時間粒度，つまり数十秒のオーダーでは不適当であることを示している．

以上のことから，再送タイムアウトが発生するほどパケット廃棄率が高くなると，単一 TCP セッションに統計的手法を実時間で適用することは現実的でないことがわかる．これが，ペナルティボックスの実装が困難である理由である．

ところで，本稿は TCP に限定して解析を行なっているが，他のプロトコルにも同じルールが適用できる．現状，インターネットのトラフィックは圧倒的に TCP が占めており，当面この状態が続くと予想される．そし

て，今後導入される新しいプロトコルも，既存のTCPと共存できることが条件となる[16]．これは，あらゆる条件下で，TCPと同等以下の帯域使用とならないと，TCPを枯渇させせることになるからである．

3 フローバルブ

フローバルブは，REDの負荷が限度に達すると作動する安全弁機能を提供する．具体的には，REDの制御可能範囲を越えるトラフィックの増加を検出し，輻輳の原因となっているフローをブロックすることによって，ルータ上の資源を防護する．REDは平均キュー長がmax_{th}を越えると，単なる到着パケット廃棄の挙動になってしまう．しかし，適応型フローのみが集約されている場合，通常このような事態には至らない．したがって，輻輳に対して適応しないフローが原因となってそのような事態に至る可能性が高く，その場合には，原因となる非適応型フローをブロックすることによって，輻輳を解消し，キュー長を適正範囲に戻すことができる．

フローバルブは[4]に示された考え方をもとにしているが，その最大の目的は，たとえ非適応型のフローが存在しても，常にREDが適正範囲で動作できるようにシステムをエンジニアリングすることである．つまり，少ないサンプル数や過渡的な条件でも適切に安全側に反応し，かつ，サンプル数の増加や状態の安定に伴って理論値に近付くような工夫がなされている．

フローバルブの機能はペナルティ・ボックスのそれと似ているが，異なる発想に基づいている．すなわち，違反者にペナルティを与えるのではなく，ルータ上の資源を容易に防護するための安全弁という考え方であり，以下の2つの新しい考え方が基本となっている．

第一には，「違反者」（非適応なフロー）を検出するのではなく，「過送出」なフローを検出することである．「過送出」とは，ルータから見て，輻輳の原因と思える量のパケットを送出していることと定義する．違反者を検出しようとすると，違反の証拠が必要であり，この証拠検出がペナルティボックス実現の最も難しい部分である．それに対して，過送出の判断は輻輳状態にあるルータが独自の基準で行なうため，送信者側の過失を証明する必要はない．各ルータは独自に「過送出テスト」と呼ぶ簡単なテストを実行することによって，過送出フローを検出する．

第二には，過送出と判定されたフローを，送信が指数的にバックオフするまで，すべての到着パケットを廃棄してブロックすることである．「バックオフ・テスト」と呼ぶ簡単なテストを用いて，指数バックオフを観測する．フローをブロックしてしまうことに関しては，あまりにも単純で罰則が重過ぎるという懸念があるかもしれない．ところが，ほとんどのTCP実装に対して，単期間のブロックによる損失は，なにも行なわない場合と比べて意外なほど差ないことがわかった．これは，過送出と判断されるほどパケット損失率が高い状態であれば，何もしなくても再送タイムアウトする確率が非常に高いためである．つまり，強制的に再送タイムアウトさせなくても，2.3節で示したように，TCPが同程度の損失をする可能性はすでに十分高い．また，フローバルブによる強制的なバックオフの影響に関しては，5.2節で検証する．

加えて，単純にブロックする方法は，確率的にパケットを廃棄する方法に比べていくつかの利点がある．第一の利点は，トラフィックの急増に対しても，迅速な対応が可能なことである．目的があくまでネットワークの防護である以上，外部から攻撃された場合にも，速やかに動作する仕組みでなければならない．

第二の利点は，輻輳の解消に要する時間が少なくてよいことである．重度の輻輳によって，REDが機能しない状態に陥っているなら，早急にREDの制御可能なレベルにまで戻す必要がある．

第三の利点は，最長罰則期間が限られることである．確率的に罰則を与える手法では，たまたま不運が重なって繰り返し罰則を受ける可能性がある．これに対し，指数バックオフは決定論的アルゴリズムであるため，限られた時間内で観測できることが保証される．

第四の利点はバックオフ動作の重要性の強調である．再送タイムアウト動作とバックオフ動作は，ともに，輻輳崩壊を避けるために欠くことのできない仕組みである．しかしながら，近年のTCP関連研究ではこの部分の重要性があまり注目されてこなかった．どちらかというと，再送タイムアウトを避けることによって性能改善する提案が多く見られるが，これらの方法は個々のフローのスループット向上が目的であり，全体の輻輳緩和

には効果がない．現状の TCP 実装の多くは，輻輳に対して慎重に再送タイムアウトするよう実装されていて，そのおかげで輻輳崩壊の危険性が低くなっている．性能重視の TCP というのは，ある意味で，パケット損失低減に対する意識の欠如を示すことになっている．

第五の利点は，慎重な実装に対する公平性である．フローバルブは，パケットをブロックしてバックオフを観測することによって，性能重視の TCP 実装が保守的な実装より有利になることを防ぐ働きがある．過送出と判定されないための唯一の手段はパケット損失を少なくすることであり，一度過送出と判定されると再送ポリシーに関わらず，同一の罰則を受けることになる．

エンド・エンドの通信にとっては，フローバルブを備えたネットワークを利用するには，中程度以下の輻輳時にはパケット損失率を低く保つこと，重度の輻輳時には慎重にバックオフすることが必須となる．フロー制御を行なわない通信の性能は，輻輳が起こると著しく低下する．また，フローバルブの動作は，RED の確率的なパケット廃棄とトラフィックのダイナミックスで変わるため，フローバルブをごまかすことはフロー制御を実装するよりはるかに困難である．

フローバルブの考え方と従来のペナルティボックスの考え方には大きな違いがあるが，結果として実現される機能を見ると大きな違いはない．その意味で，フローバルブをペナルティボックスの一実現手法としてとらえることもできる．

3.1 フローリスト

フローバルブの設計では，フローは「単一の TCP セッション」または「単一のトランスポート・セッション」として定義する．フローバルブは各フローの状態を保持する必要があるが，保持すべきフロー数は少ないものですむ．

過送出フローによって輻輳が起こる場合，ルータの RED 機構で廃棄されるパケット中，かなりの割合がこの過送出フローのものであると仮定できる．したがって，廃棄パケット数の多いフロー順にリスト管理を行なえば，過送出フローの検出にはそれほど多くのフローの状態を保持する必要はない．また，フローバルブの目的は，RED を防護することなので，たとえ狭帯域の非適応型フローが検出できない場合があってもかまわない．

フローバルブは，各フローについて 3 個の変数を保持する．p_{avg} はそのフローの平均パケット廃棄率，f_{avg} はそのフローが全到着パケットに占める割合，t_{last} はそのフローが最後にパケット廃棄された時間である．p_{avg} と f_{avg} は過送出テストに，t_{last} はバックオフ・テストに使用される．

3.2 過送出テスト

過送出テストの目的は，輻輳原因となっているフローの検出である．RED がトラフィックを制御できている間は，トラフィックの負荷に応じて確率的にパケット廃棄を行ない，適応型のフローはパケット損失に反応して送出レートを制御する．この場合，平均キュー長は max_{th} 以下に維持され，パケット廃棄率も max_p 以下に維持される．

通常，トラフィック環境は激しく変動するので，パケット廃棄率も時間とともに大きく変動することになる．このような環境では，輻輳に適応するフローは，パケット廃棄率が低い間に多くのパケットを送出し，パケット廃棄率が高い間は送出量を少なく保つ．したがって，よりうまく適応するフローは適応が下手なフローに比べてパケット損失率を低く保つことができる．RED は，通常状態で，パケット廃棄率を max_p 以下に保つように設計されている．したがって，もし，特定のフローのパケット廃棄率が max_p を越えていれば，このフローがうまく適応できていない，または，全く適応していない可能性が高い．そこで，パケット廃棄率の閾値 p_{th} を max_p に設定して，$(p_{avg} > p_{th})$ であるフローを検出すればよい．

しかし，たとえパケット廃棄率が高くても，帯域使用率の低いフローは，他のフローが引き起こした輻輳の犠牲となってパケット廃棄率が上昇している可能性があるため，除外する必要がある．帯域使用率の判定においても，単純に f_{avg} を固定の閾値と比較してもよいのだが，フローバルブでは p_{avg} からそれに対応する閾値を計算する関数を用いる．この閾値関数 $f_{th}(p)$ はあとで「TCP 親和性テスト」を近似する形で導出する．ここでは，とりあえず閾値関数 $f_{th}(p)$ が存在すると仮定すると，過送出なフローを検出するには，次の条件を評価

すればよい．

$$(p_{avg} > p_{th})\ AND\ (f_{avg} > f_{th}(p_{avg})) \quad (2)$$

過送出テストは輻輳原因となっているフローの検出が主眼であるめ，たとえ非適応型のフローでも p_{avg} が p_{th} を越えなければ検出されない．いっぽう，適応型フローでも，p_{avg} が p_{th} を越えることが起こり得る．例えば，ウィンドウサイズとラウンドトリップタイムがともに大きい場合は，まとめてパケットを損失しやすいため，過送出と判断される場合がでてくる．また，TCP の規格上は，作為的な損失間隔を与えれば，パケット損失率が 10% を越えても送出レートを維持することができる．

フローバルブは，ある意味で，TCP の規格が持つ不公平性を是正する働きをする．TCP 同士が競合した場合に，ラウンドトリップタイムが小さい方が明らかに有利であることは良く知られている．つまり，正規の TCP でも，他のフローを押し退けて，帯域を占有することがあり得る．また，セッションの最初のスロースタートでは，$ssthresh$ と呼ばれる輻輳回避フェーズに移行するべき点がまだ設定されていないため，まとまったパケット損失をすることがある．フローバルブは，このような正規の TCP が起こし得る輻輳に対する防護としても機能する．

3.3 バックオフ・テスト

バックオフ・テストは，過送出と判定されブロックされているフローを解放するために使う．バックオフ・テストは，再送間隔がバックオフ閾値 d_{th} を越えているかを判定する．これは，パケット到着時に，前回パケットが廃棄された時刻 t_{last} と現在時刻を比較することで容易に実現できる．送信者は，すべてのパケットが廃棄されるので，再送間隔が d_{th} に達するまで，再送間隔を倍増し続けることになる．

再送間隔が指数的に増えることを観測するには，d_{th} を指数分布する乱数に設定すればよい．実際には，数秒程度までバックオフすることが確認できれば十分である．より簡単な実装として，固定の閾値と秒粒度で丸めた時刻を比較すれば，丸め誤差のばらつきが乱数として機能する．

過送出と判定されたフローは，バックオフ・テストに合格しない限り，解放されない．したがって，バックオフしない通信は，以降完全に遮断される．安全弁という機能からすると，輻輳が解消すれば過送出フローを解放してよいのだが，このような厳しい罰則の導入がフロー制御に関する意識の浸透に効果的だと考えている．

3.4 パケット到着閾値関数

過送出テストのために使われる閾値関数は，パケット損失率から，それにに応じた帯域使用率を求めるものである．この閾値は，TCP 親和性テストを応用し，RED のキュー状態に関する知識を使って，フローの適正帯域使用率を近似する．この関数の目的は，あくまで過送出の判断のための簡単な近似値を得ることであり，TCP の正確なモデル化が目的ではない．それよりも，少ないサンプル数や，過渡状態でも使用可能なモデルが必要となる．

TCP のスループットに関しては，いくつかの解析モデルが存在する[9][13][15]．なかでも，Padhye らのモデル[15]は，再送タイムアウト，指数バックオフ，大きいラウンドトリップタイムを考慮しているので，パケット損失率 p が大きい場合に適したモデルである．最大ウィンドウサイズに制限されていない場合の，TCP のスループットをパケット数で表した関数 $B(p)$ は以下の式で近似される．

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 min(1, 3\sqrt{\frac{3bp}{8}})p(1+32p^2)} \quad (3)$$

ここで，RTT はラウンドトリップタイム，T_0 は再送タイムアウトの初期値である．近似には p が十分小さいことを仮定して，この式は導出されているが，実測による検証の結果，広い範囲の p において実測値によく適合することが知られている．

一方，TCP は再送タイムアウトの初期値 T_0（RTO とも呼ばれる）を以下のように計算する[7][8]．

$$RTO = srtt + 4 \cdot rttvar \quad (4)$$

$srtt$ はラウンドトリップタイムの加重平均値，$rttvar$

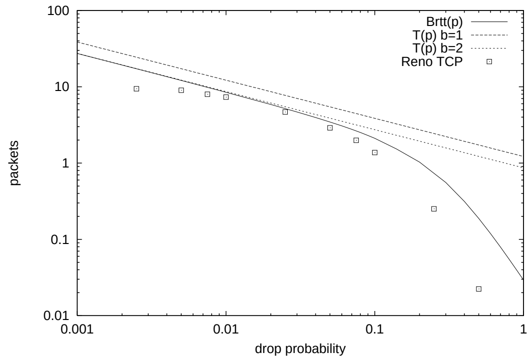


図 3　**RTT** あたりの **TCP** スループット

はその平均偏差である．式 4から，$(T_0 > RTT)$ が仮定できる．さらに，一般の TCP で採用されている 1 パケットおきの確認応答ポリシーを仮定して，$b = 2$ とすると，ラウンドトリップタイムあたりのスループットをパケット数で表した $B_{rtt}(p)$ は，次式で与えられる．

$$
\begin{aligned}
B_{rtt}(p) &= B(p) \cdot RTT \\
&< \frac{1}{\sqrt{\frac{4p}{3}} + min(1, 3\sqrt{\frac{6p}{8}})p(1+32p^2)}
\end{aligned} \quad (5)
$$

$B_{rtt}(p)$ は，TCP セッションがラウンドトリップタイムあたりに送出できるパケット数を表す．式 3が単位時間あたりのパケット数を示すのに対し，式 5が RTT 時間あたりのパケット数であることに注意されたい．正確には，式 4の $rttvar$ は無視しており，また，もとのモデルは粗粒度タイマーを考慮していないので，$B_{rtt}(p)$ は過大に評価される．図 3に示した $B_{rtt}(p)$ によると，p が 10% の場合，TCP はラウンドトリップタイム内に 2 パケットしか送出できないことが分かる．図中には，もとの TCP 親和性テストの見積り $T(p)$ も，b が 1 と 2 の場合についてプロットしている．$T(p)$ は再送タイムアウトを考慮していないため，p が 1% を越えると $B_{rtt}(p)$ から大きくずれてしまう．さらに，図中の四角マークは表 1 に示したシミュレーション結果をプロットしたもので，式 5が広い範囲の p に対し TCP のスループットをよく近似していることが確認できる．

次に，RED の状態に関する知識を使って，$B_{rtt}(p)$ をフローの正当な帯域使用率に変換する．ルータで観測

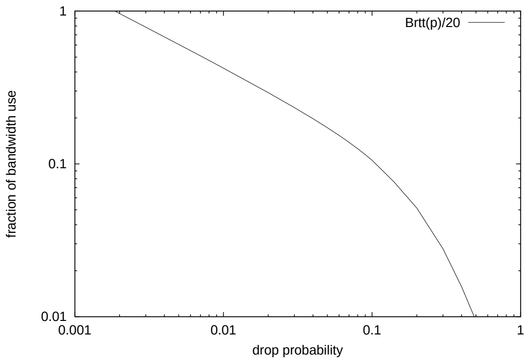


図 4　パケット損失率から計算した
単一 **TCP** セッションの適正帯域占有率

されるフローのパケット廃棄率 p が大きいとすると，このフローにとって，この先のリンクがボトルネックになっていることがわかる．フローのラウンドトリップタイムは，このルータにおけるキューの待ち時間を含む．一般に，ボトルネックでのキュー待ち遅延は，エンド・エンドの遅延のかなりの割合を占める．このルータでの平均キュー長は，RED の avg 変数で表されているので，キュー待ち遅延を，avg 個のパケットが送出される時間と近似することができる．1 つの TCP セッションは，ラウンドトリップ時間内に $B_{rtt}(p)$ 個以下のパケットしか送れないので，キューの中にはそれ以下のパケットしか存在しないはずである．したがって，フローの正当な帯域使用率 f は，以下の式で近似できる．

$$
f < \frac{B_{rtt}(p)}{(avg + \alpha)} \quad (6)
$$

ここで，α は他の遅延要因で，例えば，ネットワーク・カード内のバッファを考慮すると，数パケット分の遅延を加算することができる．もし，遅延の大きいリンクを使っているのなら，その分を加算することもできる．実際には，それ以外にも，他のルータにでのキュー待ち遅延，フォワーティング遅延や，復路の輻輳なども ラウンドトリップタイムに加えられることになる．

式 6は，過送出テストに使う場合，さらに簡単化できる．過送出テストにおいて，f_{avg} を評価する必要があるのは，パケット廃棄率 p_{avg} がすでに p_{th} を越えている場合で，f_{avg} が正当な帯域使用率以下のフローを除

外するために式 6を用いるわけである．ここで，あるフローが正当な帯域使用率以下しか使っていないにもかかわらず，その p_{avg} が p_{th} を越えているとする．p_{th} は RED の max_p に等しく設定されているので，RED のパケット廃棄率がすでに最大廃棄率 max_p を越えていることになる．その場合，平均キュー長 avg は max_{th} を越えていることになる．したがって，式 6を以下の式 7に置き換えることができる．

$$f < \frac{B_{rtt}(p)}{(max_{th} + \alpha)} \qquad (7)$$

式 7は，正当な使用帯域率を，p の関数として近似したものになる．式 7は，ルックアップ・テーブルを使って効率的に実装できる．また，p に p_{th} を代入して得られる固定値を使うことも可能である．

図 4は，式 7に $max_{th} = 15$ と $\alpha = 5$ をパラメータとして与えた場合を示している．パケット損失率が，5%，10%，20%，50% と増えていくと，対応する適正帯域使用率が，17%，10%，5%，1% と減少する．図 4は，直観的にも理解しやすいパケット損失率と帯域使用率の関係を与え，また，過送出テストに用いるに適した範囲に収まっているのがわかる．

式 7を導出する際に用いた仮定は，現状の一般的な環境を想定しているため，これらの条件が常に成立するわけではない．それでも，経験則等による根拠のない固定パラメータは含まれておらず，各条件は明確になっているので，環境が変わっても容易に条件を検証することができる．また，過送出テストにおいて，適正帯域の使用は補助的な役割なので，計算に多少の誤差が含まれていても，あまり問題とはならない．

また，ここでは，すべてのパケットが同等のサービス時間を持つ，すなわち，パケットサイズが等しいものとして扱っている．これは，RED のパケット・モードに対応するものである．RED にはパケット・モード以外に，バイト・モードがあって，平均キュー長をバイト数で計算することができる．その場合，ここで示したモデルをバイト・モードに対応するのは簡単で，そうする事によって，パケットサイズを考慮に入れることができる．

3.5 スケーラビリティ

過送出フローが検出される条件は，フローのパケット廃棄率が閾値を越えていて，かつ，フローの帯域使用率が適正帯域使用率を越えている場合である．この基本部分の考え方は，フローの数に影響されない．ただし，多くのフローが集約されるバックボーン・ネットワークでは，単一フローあたりの占める割合が小さくなるため，フローバルブが帯域使用率の低い非適応型フローを検出できない可能性が増えてくる．したがって，バックボーンでは，狭帯域の非適応型フローを検出することよりも，広帯域の非適応型フローに対する防衛手段という意味合いが強くなる．

また，別の問題として，集約されたフローに対して，フローバルブのアルゴリズムが適用できるかという問題もある．フローバルブが技術的に適用可能である事とは別に，問題のあるフローが含まれるからという理由で，集約フロー全体を罰してよいかどうかに関しては議論の余地があるが，ここでは技術的側面のみに触れておく．過送出テストで利用する特性は，フローが集約されても成り立つものなので，適切な重み付けをすれば，過送出テストは適用できる．一方，バックオフテストは，単一フローのバックオフ・アルゴリズムを前提に考えられているため，変更が必要になる．これには，様々な方法が考えられるが，例えば，帯域使用率 f_{avg} が閾値以下になればフローを解放するという方法が取れる．

4 実　装

本章では，フローバルブの概念の実装例として，我々のプロトタイプ実装の詳細を示す．

4.1 フローリスト管理

これまで，フローは単一の TCP セッションとして定義してきたが，この定義をそのまま実装すると，複数の TCP セッションを並列に使ったほうが有利になってしまう．また，パケットが断片化されたり，暗号化されている場合は，単一 TCP セッションを識別できない場合もあり得る．そこで，実装ではソースとデスティネーションのアドレスの組でフローを識別している．

フローリストは，最近パケット損失を経験したフローのリストである．過送出フローのパケット損失のパター

ンは強い相関を持つので，たかだかフロー１０個分のリストサイズでも，10% 以上の帯域を占めている過送出フローを検出できると予想できる．本実装では，帯域使用が $f_{th}(max_p)$ を越える過送出フローを検出しなければならないので，$1/f_{th}(max_p)$ 個程度のエントリがあればよい．実際には，ゆらぎを見込んでその数倍のエントリを割り付けているが，それでも数十個のオーダーである．

フローリストは簡単な LRU(Least-Recently-Used) 置換ポリシーを使って管理される．RED がパケットを廃棄すると，フローリストの中から対応するフローのエントリを探す．対応するエントリが見つからない場合は，リストの最後尾のエントリと交換する．フローのエントリは，更新を受けた後，リストの先頭に置かれる．各フローエントリは，一定期間（プロトタイプでは３秒間）パケット廃棄がなければ解放される．

4.2 フロー変数

フロー変数 p_{avg} と f_{avg} は加重平均法 (Exponentially-weighted moving average) で効率良く実装できる．パケット廃棄率 p_{avg} は，パケットが到着する度に，次式を使って更新する．

$$p_{avg} = w \cdot x + (1 - w) \cdot p_{avg} \qquad (8)$$

ここで，w は加重平均の重みであり，x はパケットが廃棄された場合に 1 を，それ以外で 0 をとる．

帯域使用率 f_{avg} は n パケット到着毎に，次式を使って更新される．

$$f_{avg} = w' \cdot \frac{n}{seq - last} + (1 - w') \cdot f_{avg} \qquad (9)$$

seq はパケットがインターフェイスに到着する度にカウントアップされるシーケンス番号で，$last$ は前回 f_{avg} が更新された時の seq の値である．

ここでの加重平均法のように実装効率のよい手段には，偏りがでるので注意が必要である．この場合，トラフィックの増加を検出しやすい方向に偏っている．つまり，平均値はパケット到着毎に更新されるので，到着間隔が短いほど平均値が速く移動することになる．w が 1/128 の場合，p_{avg} が 0% から 10% になるのに，すべてのパケットが廃棄された場合１４パケット，20% のパケットが廃棄された場合８８パケットを要する計算になる．

```
void fv_enqueue(pkt)
{
  if (flowlist != NULL &&
      check_flow(pkt) == DROP)
    return;

  if (red_enqueue(pkt) == DROP)
    drop_by_red(pkt);
}
```

図 5　*enqueue* 関数

```
/* flow-valve entry structure */
struct fve {
  int     state;    /* GREEN or RED */
  double  p;        /* drop rate */
  double  f;        /* fraction of bw */
  int     count;    /* used to update f */
  int     seq;      /* if_seq of last drop */
  double  timestamp;  /* time of last drop */
};

int   if_seq; /* seq no of arrival packets */
const double P_THRESH = red_max_p;
const int    BACKOFF_THRESH = 1;
const int    N = 10;
const double Wp = 1.0 / 128.0;
const double Wf = 1.0 / 32.0;
```

図 6　構造体と定数

4.3 アルゴリズム

図 5から図 8にフローバルブのアルゴリズムを示す．簡単のために倍精度浮動小数点演算を用いているが，これらは簡単に固定小数点演算に変換でき効率良く実装できる．

フローバルブは，図 5に示すように，RED のキュー格納関数の包含関数として実装できるので，既存の RED 実装に簡単に追加できる．パケット到着時に，もしフローリストが空でなければ，*check_flow()* 関数が呼ばれパケットが検査される．ここで，そのパケットが属するフローが過送出と判断されれば，パケットはこの時点で廃棄される．RED がパケットを廃棄すると，*drop_by_red()* 関数が呼ばれ，フローバルブ・エントリ

```
void drop_by_red(pkt)
{
  struct fve *fve;

  if ((fve = flowlist_lookup(pkt)) == NULL)
    fve = flowlist_reclaim(pkt);
  flowlist_move_to_head(fve);

  /* update p: the following line cancels the
   * update in check_flow() and calculate
   *   p = Wp + (1-Wp) * p
   */
  fve->p = Wp + fve->p;
  fve->timestamp = now;
}
```

図 7 *drop_by_red* 関数

が割り当てられる．すでにエントリが存在する場合は単に更新される．

図 6はフローバルブ・エントリ構造体と関連する定数を示している．フローの状態はグリーンかレッドで表され，レッドは過送出状態を示す．

RED がパケットを廃棄すると，図 7の *drop_by_red()* 関数が呼ばれる．まず，フローリスト内にすでに対応するエントリが存在するか調べ，ない場合は LRU 方式でエントリを置換割り当てする．その後，エントリはリストの先頭に置かれる．続くブロックで平均パケット損失率とタイムスタンプを更新する．

図 8の *check_flow()* 関数はパケット到着時に呼び出される．まず，インターフェイスのパケットシーケンス番号が増やされ，次に，フローリスト上の対応エントリを探す．対応するエントリが存在しない場合は，OK が返り通常の RED の処理に進む．次のブロックで，フローの平均帯域使用率を，N パケットごとに更新する．さらに次のブロックで，式 2で示した過送出テストを実行する．フロー状態がグリーンで，かつ，p_{avg} が p_{th} を越えると，f_{avg} がチェックされる．帯域使用率閾値関数 *p2f()* は，式 7を実現する関数で，ルックアップ・テーブルにより効率的に実装できる．その次のブロックはフローのブロック動作の実行と，バックオフ・テストを行なう．フロー状態がレッドならば到着パケットは廃棄される．しかし，パケット到着間隔がバックオフ閾値 d_{th} を越えるとフローエントリは解放される．

```
int check_flow(pkt)
{
  struct fve *fve;

  if_seq++;
  if ((fve = flowlist_lookup(pkt)) == NULL)
    /* no matching entry in the flowlist */
    return OK;

  /* update f for every N packets */
  if (++fve->count == N) {
    fve->f = Wf * N/(if_seq - fve->seq)
             + (1.0 - Wf) * fve->f;
    fve->seq = if_seq;
    fve->count = 0;
  }

  if (fve->state == GREEN
      && fve->p > P_THRESH) {
    /* calculate threshold by lookup table */
    if (fve->f > p2f(fve->p))
      fve->state = RED;
  }

  if (fve->state == RED) {
    if ((int)now - (int)fve->timestamp
        > BACKOFF_THRESH) {
      /* no drop for BACKOFF_THRESH sec */
      fve->p = 0;
      fve->state = GREEN;
    } else {
      /* block this flow */
      fve->timestamp = now;
      flowlist_move_to_head(fve);
      drop(pkt);
      return DROP;
    }
  }
  p = (1 - Wp) * p;     /* update p */
  return OK;
}
```

図 8 *check_flow* 関数

5 シミュレーション結果

フローバルブの動作を，*ns* シミュレータ (ver. 2.1b3) を使って検証した．図 9に示すようなトポロジで，2 種類のテストを行なった．S1 と S2 は送信ホスト，S3 と S4 が受信ホスト，R1 と R2 がルータである．R1 がボトルネックリンクの入口となるため，R1 で RED とフローバルブを動作させている．RED の設

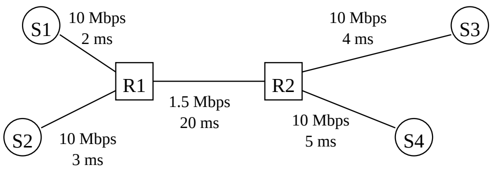


図 9 シミュレーション・ネットワークの構成

定値には，($min_{th} = 5$，$max_{th} = 10$，$max_p = 0.1$) を用いた．フローバルブの設定値には，($p_{th} = 0.1$，$w = 1/128$，$w' = 1/32$，$n = 10$，$d_{th} = 1$) を用いた．TCP は Reno バージョンを使っている．

図 10から図 12をとおして，グラフ (a) は R1 で観測されたパケットのシーケンス番号を示しており，パケットの送信パターンとパケット廃棄の関係を表している．フロー i のシーケンス番号 n は，$((n\ mod\ 90)/100 + i)$ の位置にプロットされ，９０パケットごとに回り込むため，各フローが行に対応するようになっている．パケットは，キューから抜き出された時点で灰色に，また，廃棄された場合は黒の'X' で示されている．グラフ (b) は，各フローの使用帯域を，図 10と図 11では 250 ミリ秒間隔で，図 12では 500 ミリ秒間隔で，有効帯域に正規化してプロットしている．グラフ (c) は，R1 における RED の平均キュー長と，瞬間的なキュー長をプロットしている．RED は，平均キュー長が 5 から 15 の間にある場合に，早期パケット廃棄を実行することになる．キュー長の限度は 25 である．グラフ (d) とグラフ (e) は，R1 が推測した各フローの平均パケット廃棄率 p_{avg} と，平均帯域使用率 f_{avg} をそれぞれ示している．パケット廃棄率の閾値 p_{th} は 0.1 なので，p_{avg} が 0.1 を越えると過送出の対象となる．グラフ (d) とグラフ (e) の各フローの値がリセットされている点では，３秒以上パケット廃棄がなかったために，フローバルブ・エントリが解放されている．

なお，2.3節の表 1と図 2にもちいたシミュレーションでは，S1 から S3 に向かって，20 セグメント分のウィンドウサイズで FTP タイプのデータ転送を行ない，R1 において指定確率でパケット廃棄を行なった．

5.1 テスト１

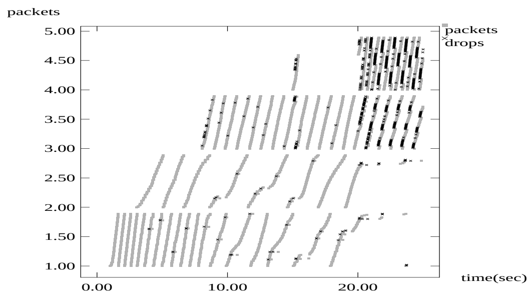
テスト１は，フローバルブが輻輳時に非適応型のフローを検出する挙動と，広帯域を占めるフローを迅速に制限する挙動を示す２５秒間のシーケンスである．２個の FTP と２個の固定レート (CBR) からなる４個のフローが使われている．

フロー１ ftp from S1 to S3 (winsize:20)
フロー２ ftp from S2 to S3 (winsize:5)
フロー３ CBR from S2 to S4 (800Kbps,1000B/pkt)
フロー４ CBR from S1 to S4 (1.6Mbps,1000B/pkt)

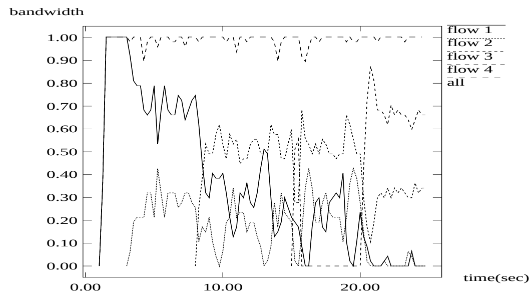
テスト１の開始時には，フロー１とフロー２が帯域を共有している．フロー２の使用帯域は，その小さいウインドウサイズに制限されている．時刻８に，CBR であるフロー３が開始し，他の２個の TCP は転送レートを減少させる．時刻１５に，もうひとつの CBR フロー４が 300 ミリ秒間だけ起動され，トラフィックの急増をエミュレートする．時刻２０に，フロー４は再度起動され，最後まで転送を続ける．

図 10は，従来の RED がテスト１でどのように振舞うかを示している．フロー４が２度目に起動された後では，平均キュー長が max_{th} を越えて，到着パケット廃棄の挙動に戻っていることがわかる．平均キュー長は max_{th} の前後で振動するため，瞬間的なキュー長が大きく振動する結果となる．TCP は共に転送レートをほとんど０にまで絞ってしまい，CBR フローが利用可能帯域を占有する結果となる．

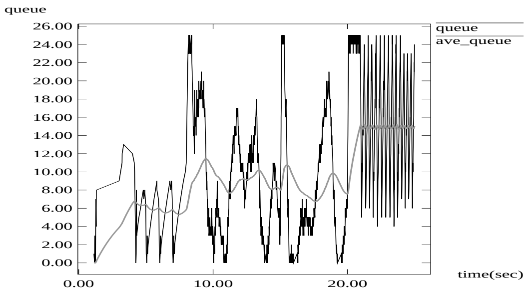
図 11はテスト１におけるフローバルブの効果を示す．フロー３が開始すると同時に，フロー３の平均パケット損失率が急増するが，他の TCP が転送量を絞るにしたがって，平均パケット損失率も下がってくる．この時点では，フロー３のパケット損失率が p_{th} に達しないため，フロー３は過送出とは判断されない．RED の平均キュー長は，min_{th} と max_{th} の間に維持され，TCP のパケット損失率も 2% から 3% 程度に保たれる．フロー４が起動されトラフィックが急増すると，フロー３のパケット損失率が p_{th} を越え，過送出と判断されたフロー３はこの時点でブロックされる．この動作は，フローバルブにおける非適応型フローの典型的な挙動である．すなわち，トラフィックが増加する際に，非適応型フローが検出される．フロー３は，ブロックされても転送レートを維持するので，二度と解放されない．



(a) 各フローのシーケンス番号



(b) 各フローの帯域使用



(c) キュー長

図 **10**　テスト1：従来の **RED** の場合

フロー３がブロックされてしまうと，TCP が再び全帯域を利用できるようになる．

TCP は，トラフィックが急増した間にも，ほとんど平均パケット廃棄率が増加していない．これは，RED が時折パケットを廃棄しているために，輻輳ウィンドウがすでに小さく保たれていることと，キュー長の急増によって，TCP のセルフクロッキング[7]の周期が延びることになるので，その結果，パケットを損失する前にすでに TCP はスローダウンできるからである．

時刻２０にフロー４が再起動されると，フロー４もすぐに過送出と判断されブロックされる．これは，フローバルブが広帯域非適応型フローに迅速に対応をする動作を示す．グラフ (c) が示すように，RED の平均キュー長は，たとえトラフィックの急増があっても，max_{th} 以下に維持されている．

5.2　テスト２

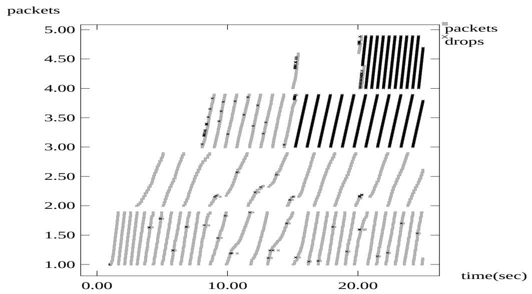
テスト２は，４個の TCP フローの干渉をしめす５０秒間のシーケンスである．フロー１とフロー２は，テスト１と同様で，フロー３とフロー４は，大きなウィンドウサイズを持つ間欠フローとなっている．

フロー１　ftp from S1 to S3 winsize:20
フロー２　ftp from S2 to S3 winsize:5
フロー３　ftp from S1 to S4 winsize:40
　off period:38–43
フロー４　ftp from S2 to S4 winsize:40
　off period:28–32,37–43

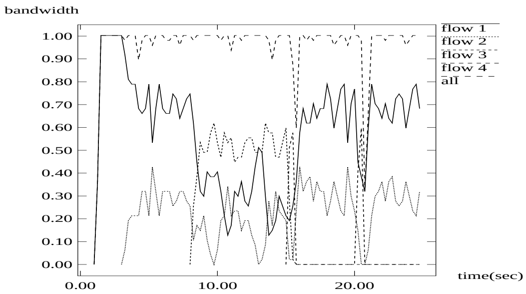
フロー３は，時刻７に起動された直後，最初のスロースタートでまとめてパケットを損失する． フロー３は，パケット損失率が p_{th} を越えて，過送出と判断されるが，バックオフの後解放される．その後は，$ssthresh$ が設定されるため，フロー３が再び過送出と判断されることはない．

図 13は，フロー３の起動時の動作を示している．RED の瞬間キュー長が限度に達して，フロー３が連続的にパケットを損失，過送出と判断されブロックされた後，指数バックオフをする様子がわかる．４度目の再送で，再送間隔が d_{th} に達して，フロー３が解放されている．

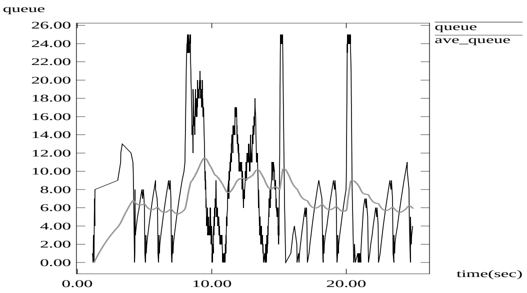
フロー４は，時刻１２に起動され，フロー３と同様にパケット損失を経験するが，この場合は，平均パケット



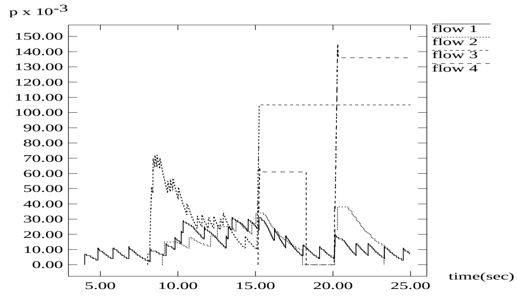
(a) 各フローのシーケンス番号



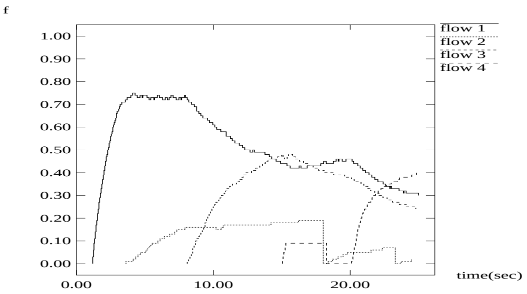
(b) 各フローの帯域使用



(c) キュー長



(d) 平均パケット廃棄率

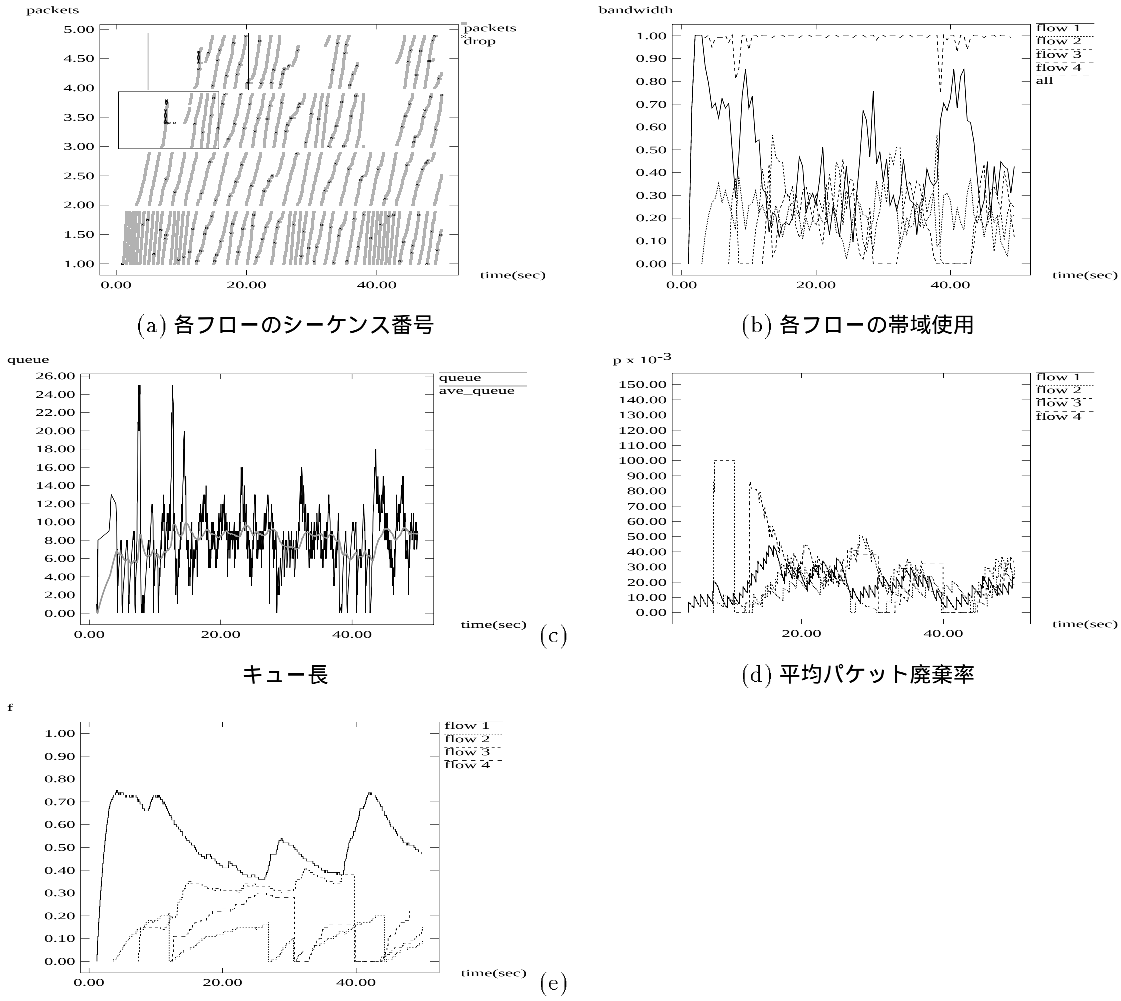


(e) 平均帯域使用率

図 11　テスト 1：フローバルブを適用した場合

損失率が p_{th} に達しないので，過送出と判断されるには至らない．図 14は，フロー 4 の起動時の挙動を示す．

図 13と図 14の違いは，バックオフ・テストの TCP への影響を示している．図 13は，バックオフ・テストによって強制的に TCP を再送させた場合の挙動，図 14は，通常の再送タイムアウトによる TCP の再送の挙動である．すでにパケット損失率がこのレベルに達していると，フローバルブが強制的にタイムアウトさせなくても，TCP が同程度の再送タイムアウトを経験する確率は十分高い．ラウンドトリップタイムとパケット損失のタイミングによっては，自然発生する再送タイムアウトによる TCP の被害が，バックオフ・テストによる被害より大きくなる可能性も十分ある．また，一般の TCP の実装では，最小再送タイムアウト値がもっと大きいので，挙動の違いはさらに小さくなる．ここで重要なのは，強制的にバックオフさせることで，そのルータで発生している輻輳を急速に解消できる点である．テスト 2 のトラフィック・パターンは，変動が大きいにも関わらず，平均キュー長も各フローの平均パケット損失率も，前述の最初のスロースタート時を除いて，適正範囲に維持されている．これは，RED が適正範囲で動作していれば，各 TCP フローの平均パケット損失率は適正範囲にあり，フローバルブが作動しないことを示している．



(a) 各フローのシーケンス番号

(b) 各フローの帯域使用

(c) キュー長

(d) 平均パケット廃棄率

(e) 平均帯域使用率

図 **12**　テスト 2：フローバルブ下での **TCP** の挙動

6 まとめ

インターネット上に，音声やビデオ，マルチキャストといった新しい形態の通信が増えてくる中，エンド・エンドのフロー制御はインターネットの健全な発展の鍵となる．しかしながら，現状のインターネットには，エンド・エンドのフロー制御を促進するような仕組みがない．

本稿では，フローバルブと呼ぶ RED の安全弁機構を提案し，小数のフロー情報を管理することによって輻輳時にルータ資源を防護できることを示した．フローバルブは，ルータの局所的判断で過送出フローを検出し，強制的に過送出フローを遮断する．フローバルブによって，RED ペナルティボックスの概念が，簡単な仕組みで実装できる．

フローバルブは，パケット損失率が高く，なおかつ，帯域使用率が高いフローを検出すると，強制的にフローをバックオフさせるという簡単な機構である．これによって，中程度以下の輻輳下でパケット廃棄率を低く維持し，重度の輻輳下で慎重にバックオフするというエンド・エンドのフロー制御を促進することができる．

シミュレーションの結果，フローバルブによって，たとえ不当なフローが存在しても，RED が制御範囲内にとどまり有効に機能することが確認された．さらに，正

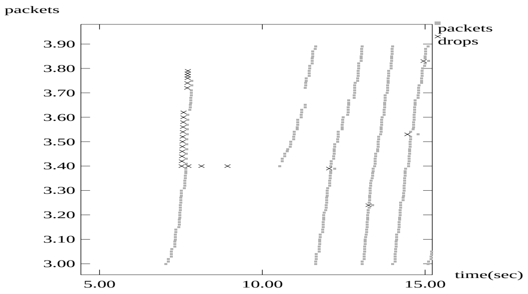


図 13 フローバルブによるバックオフ

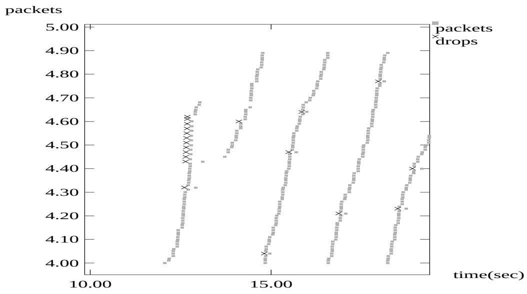


図 14 通常のタイムアウトによるバックオフ

規の TCP が起こし得る輻輳に対しても，有効な防護手段となることが確認された．

また，実際に，フローバルブを，ALTQ[1]による RED 実装の拡張として FreeBSD に実装し動作を検証した．フローバルブの実装自体は，C 言語で約 500 行である．本稿ではシミュレーションによる評価を用いたが，これは実装に比較して，キューの内部状態の詳細が得られ，また，再現性のあるデータが取得できるためである．現在，フローバルブの実装は，ALTQ のリリースに含まれ公開されているので，運用ネットワークでの実証実験がすぐに可能である．今後，実際のネットワークで運用され，エンド・エンドのフロー制御の促進に貢献できることを期待している．

参考文献

[1] Kenjiro Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. In Proceedings of *USENIX 1998 Annual Technical Conference*, New Orleans, LA, June 1998.

[2] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[3] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transaction on Networking*, 1(4):397–413, August 1993.

[4] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transaction on Networking*, 7(4):458–472, August 1999.

[5] S. Floyd, K. Fall and K. Tieu. Estimating Arrival Rates from the RED Packet Drop History. *Draft paper*, April 1998.

[6] S. Floyd. Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-way Traffic. *Computer Communication Review*, 21(5):30–47, October 1991.

[7] V. Jacobson. Congestion Avoidance and Control. *Computer Communication Review*, 18(4)314–329, August 1988.

[8] V. Jacobson. Berkeley TCP Evolution from 4.3-Tahoe to 4.3-Reno. In Proceeding of *the Eighteenth Internet Engineering Task Force*, p.365, September 1990.

[9] A. Kumar. Comparative Performance Analysis of Versions of TCP in Local Network with a Lossy Link. *IEEE/ACM Transactions on Networking*, 6(4), August 1998.

[10] D. Lin and R. Morris. Dynamics of Random Early Detection. In Proceedings of *SIGCOMM97*, 127–137, Cannes, France, September 1997.

[11] T. V. Lakshman and U. Madhow. The Performance of Networks with High Bandwidth-delay Products and Random Loss. *IEEE/ACM Transactions on Networking*, June 1997.

[12] S. McCanne and S. Floyd. NS (Network Simulator). http://www-nrg.ee.lbl.gov/ns/, 1995.

[13] M. Mathis, J. Semke, J. Mahdavi and T. Ott. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communication Review*, 27(3):67–82, July 1997.

[14] J. Nagle. On packet switches with infinite storage. *IEEE Trans. on Comm.*, 35(4), April 1987.

[15] J. Padhye, V. Firoiu, D. Towsley and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In Proceedings of *SIGCOMM98*, 303–314, Vancouver, Canada, September 1998.

[16] D. Sisalem, H. Schulzrinne, The Loss-Delay Adjustment Algorithm: A TCP-friendly Adaptation Scheme. In Proceedings of *NOSSDAV98*, Cambridge, UK, July 1998.