TCP の公平性を考慮したパケット処理方式のネットワークプロセッサへの実装に関する検討

倉田 謙二† 長谷川 剛‡ 村田 正幸‡

† 大阪大学大学院基礎工学研究科情報数理系
〒 560-8531 大阪府豊中市待兼山 1-3
Phone: 06-6850-6616, Fax: 06-6850-6589
E-mail: k-kurata@ics.es.osaka-u.ac.jp

‡ 大阪大学サイバーメディアセンター
〒 560-0043 大阪府豊中市待兼山町 1-30
E-mail: {hasegawa, murata}@cmc.osaka-u.ac.jp

あらまし 本稿では、ルータにおいてフロー間の公平なサービスを実現するための方式として、ZL-RED (Zombie Listed RED) を提案する。ZL-RED は、SRED において提案されている Zombie List を用いて、他のフローに比べてパケット到着率の高いフロー (mis-behaving フロー) を検出し、そのフローのパケット廃棄率を高く設定することで、フロー間の公平性を向上させる。さらに、シミュレーションによる ZL-RED の性能評価を行い、TCP コネクション間の不公平性を大きく改善できることを示す。また、ZL-RED をネットワークプロセッサに実装する際に考慮すべき点についての検討を行う。

キーワード TCP (Transmission Control Protocol)、TCP Reno、TCP Vegas、公平性、SRED (Stabilized RED)、Network Processor

A Study on the implementation of ZL-RED mechanism on the Network Processor

Kenji Kurata† Go Hasegawa ‡ Masayuki Murata‡

†Department of Infomatics and Mathematical Science
Graduate School of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: +81-6-6850-6616, Fax: +81-6-6850-6589
E-mail: k-kurata@ics.es.osaka-u.ac.jp

‡Cybermedia Center, Osaka University
Toyonaka, Osaka 560-0043, Japan
E-mail: {hasegawa, murata}@cmc.osaka-u.ac.jp

Abstract In this paper, we propose a ZL-RED (Zombie Listed RED) algorithm, which ehhances a SRED algorithm to provide better fairness among many flows at the router buffer. ZL-RED uses the *Zombie List*, which is originally proposed by SRED, to detect mis-behaving flows which send packets at higher rate than others. Then, ZL-RED sets higher packet discarding probability to those mis-behaving flows. We evaluate an effectiveness of ZL-RED by simulation experiments, and show that ZL-RED can actually improve fairness among TCP connections. Furthermore, we discuss the implementation design of ZL-RED mechanism on the network processor.

Keyword TCP (Transmission Control Protocol), TCP Reno, TCP Vegas, Fairness, SRED (Stabilized RED) , Network Processor

1 はじめに

近年、インターネットのユーザーは急激に増加しており、ユーザーが利用するアプリケーションも電子メールやWWWアクセス、ファイル転送といったデータ系のアプリケーションのみではなく、音声や映像のストリーミング、オンラインゲームといったリアルタイムアプリケーションまで多様化しているが、これまでのインターネットはデータ系アプリケーションのみを対象として発展してきたため、様々な問題が発生しつつある。例えば、リアルタイムアプリケーションが主に用いるトランスポート層プロトコルである UDP (User Datagram Protocol) のコネクションと、データ系アプリケーションが用いる TCP (Transmission Control Protocol) のコネクションが同一パス上に存在すると、TCP コネクションのスループットが低下し、UDP コネクションとの間に不公平が生じることが明らかになっている [1]。

このような不公平性は、TCP のコネクション間にも存在する。その一例として、TCP の異なるバージョン間で発生する不公平性があげられる。現在のインターネットにおいては、TCP Reno バージョンが最も一般的に用いられている。TCP Reno は、Fast Retransmit や Fast Recovery といった輻輳制御機能 [2] を持つが、さらに高い性能を得るための高機能化を図った TCP に関する研究も行われている。その中で、1995 年に提案された TCP Vegas バージョン [3] が、高スループットを得られるとして注目されている。TCP Vegas は、パケットロスによってのみネットワークの輻輳を検出する従来の TCP Tahoe/Reno バージョンとは異なり、パケットの RTT (Round Trip Time) を観測してネットワークの輻輳を検出し、ウィンドウサイズを増減させる。

これまでの研究で、TCP Vegas は、ネットワーク内に TCP Vegas コネクションのみが存在する環境においては非常に高い性能が得られることが明らかになっている [4, 5, 6]。しかし、TCP Vegas コネクションと従来の TCP Tahoe/Reno コネクションがネットワーク内に混在する環境においては、TCP Vegas コネクションのスループットが低下することが指摘されている [7, 8]。これは、双方の輻輳制御方式が持つ、ネットワーク内に存在するルータのバッファの利用方針の本質的な違いに原因がある。

このような不公平性を解決する方法の一つとして、ルータのバッファにおいて RED (Random Early Detection) [9] を用いることが考えられる。RED は、輻輳の初期段階から積極的にパケットを確率的に廃棄することにより、バッファ溢れを抑制する。[10] において、RED を用いることによって TCP Reno コネクションのウィンドウサイズの不当な増大を抑え、TCP Vegas コネクションとの間の不公平性をある程度解消できることが示されている。しかし RED には、コネクション数の増加に応じてキュー長が増大するという問題や、上述の UDP コネクションと TCP コネクションの間の不公平性を解決することができないことも指摘されている [11]。

そこで、[11] では、RED による確率的なパケット処理を利用し、ネットワーク内のアクティブフロー数を推定し、そのフロー数に応じてパケット廃棄率を変化させることでキュー長を安定させる SRED (Stablized RED) が提案されている。しかし、SRED では、ルータのバッファ内のキュー長を安定させることを主な目的としているため、コネクション間の公平性については十分に考慮されていない。

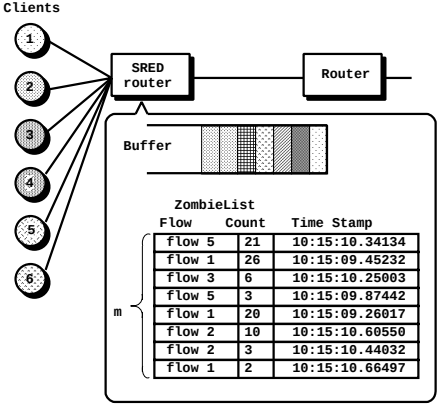


図 1: SRED

そこで、本稿では、SRED が持つ機能を利用することにより、コネクション間の不公平性を解消するためのパケット処理方式として ZL-RED (Zombie Listed RED) [12] を提案する。ZL-RED では、[11] で提案された Zombie List のメカニズムを利用することにより mis-behaving フロー、すなわち他のフローより多くのパケットを送信しているフローをルータで検出し、そのフローのパケットをより高い確率で廃棄する。その結果、TCP コネクション間の公平性を向上できることをシミュレーションによって明らかにする。

また、近年のインターネットの利用の拡大により、ルータにおいて柔軟性やスケーラビリティが要求されている。その中で、基本的な枠組のみをハードウェアで提供し、上位の処理はソフトウェアで実行するネットワークプロセッサが現在注目を浴びている。そこで本稿では、ZL-RED をインテル社製ネットワークプロセッサ IXP1200 上に効率的に実装するために考慮すべき点についての検討を行う。

以下、2、3 章ではそれぞれ SRED、ZL-RED についての説明を行う。次に、シミュレーションによる SRED と ZL-RED の性能評価を 4 章で行い、ZL-RED の有効性を示す。5 章では、ZL-RED のネットワークプロセッサへの実装について考察する。最後に 6 章でまとめと今後の課題を述べる。

2 SRED: Stabilized RED

本章では、提案方式の基となる SRED のアルゴリズム及び特徴について簡単に説明し、改善すべき点が存在することもあわせて指摘する。SRED の詳しいアルゴリズムについては [11] を参照されたい。

SRED は、ルータで Zombie List と呼ばれる固定サイズのテーブルを保持する (図 1 参照)。図中の m は Zombie List のサイズ (エントリ数) を示す。このテーブルの各エントリ (Zombie と呼ぶ) は、(フロー ID、カウンタ、タイムスタンプ) の組で構成されており、ルータにパケットが到着する毎に、到着パケットに関する情報をエントリに書き込むことによって順次更新される。フロー ID は、特に厳密に定義されるものではなく、到着パケットの送信側の IP アドレスと受信側の IP アドレスの組を用いたり、送信側の IP アドレスのみを用いたりできる。パケットがルータに到着したときに、Zombie List のエントリに空きがあれば、到着パケットの情報から新たな

Zombie を構成し、Zombie List のエントリとして追加する。Zombie List に空きがなければ、Zombie List からランダムに Zombie を 1 つ選択し、その Zombie のフロー ID と到着パケットのフロー ID とを比較する。フロー ID が一致した場合 (これを「ヒット」と呼ぶ)、その Zombie のカウンタを増加させる。ヒットしなかった場合は、確率 p_{swap} (定数) でその Zombie を到着パケットのフロー ID で上書きし、カウンタを 0 に初期化する。ここで、$P(t)$ を到着パケットがヒットする確率の平均値 (平均ヒット率) とすると、$P(t)$ の逆数 $1/P(t)$ は、ルータ内に存在するアクティブフロー数にほぼ等しくなるため [11]、Zombie List を使うことによってルータバッファ内に存在するフロー数を推測することができる。

次に、以下の式を用いることにより、バッファ内パケット数に応じて基本となるパケット廃棄率を決定する。

$$p_1 = \begin{cases} 0 & \text{if} \quad 0 \le q_{len} < \frac{1}{6}B \\ \frac{1}{4} \times p_{max} & \text{if} \quad \frac{1}{6}B \le q_{len} < \frac{1}{3}B \\ p_{max} & \text{if} \quad \frac{1}{3}B \le q_{len} \end{cases} \tag{1}$$

ここで、B、q_{len} はそれぞれルータのバッファサイズ、パケット到着時のルータのバッファ内のパケット数を表しており、p_{max}、th_{min} [packets] は RED において定義されているものと同様の定数である。

次に、フロー数の推測値 $1/P(t)$ を用い、フロー数が少ない時はパケット廃棄率を小さくし、フロー数が多い時はパケット廃棄率を大きくする。

$$p_2 = p_1 \times min\left(1, \frac{1}{(256 \times P(t))^2}\right) \tag{2}$$

最後に、以下の式を用いて、到着パケットがヒットしたか否かにより、パケット廃棄率を変化させる。

$$p_3 = p_2 \times \left(1 + \frac{Hit(t)}{P(t)}\right) \tag{3}$$

ここで、$Hit(t)$ は、到着パケットがヒットした場合は 1、ヒットしなかった場合は 0 とする。

SRED は、平均ヒット率 $P(t)$ の逆数 $1/P(t)$ によって、フロー数を推測することができ、それを式 (2) のように用いることによって、フロー数が少ない時にはパケット廃棄率を小さくし、フロー数が多い時にはパケット廃棄率を大きくしている。これにより、フロー数に関係なくバッファ内パケット数を安定させることができる [11]。しかし、特にフロー数が多くなると、フロー数の推測が不正確になる。図 2 は、図 4 のネットワークモデルを用いて、10〜1280 本の TCP Reno コネクションが同時にデータ転送を行ったときの実コネクション数と推測コネクション数 ($1/P(t)$) の関係を表している。この図から、Zombie List のサイズが小さい場合や、フロー数が大きい場合には、推測フロー数が実フロー数よりも小さくなることがわかる。これは、フロー数が大きくなることにより、各フローのパケット到着レートがばらつき、フロー数の推測に大きな影響を与えるためである [11]。

さらに SRED は、式 (3) を用いて、ヒットしたパケットの廃棄率を大きくする。このことにより、フロー間の公平性が向上することが考えられる。これは、パケット到着レートが他のフローに比べて高いフローは、Zombie List 内に多くのエントリを含むため、そのフローの到着パケットがヒットする確率が高くなるためである。し

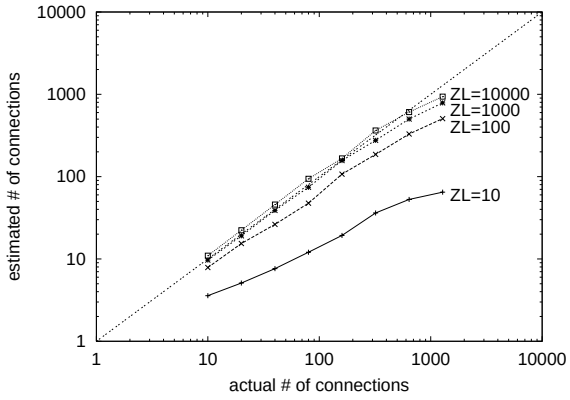


図 2: 実コネクション数と推定コネクション数の関係

かし、[11] で示されているように、SRED では公平性を大幅に向上することはできない。そこで、次章で説明する提案方式 (ZL-RED) では、Zombie List のエントリ内に含まれる情報をより有効に利用することにより、mis-behaving フローを検出し、パケット廃棄率を高く設定し、フロー間の公平性を向上させる。

フロー間の公平性を向上させるための方式としては、DRR (Deficit Round Robin) [13] のように、フローごとにキューを用意して到着パケットを処理する方式や、FRED (Flow Random Early Drop) [14] のように、各フローの情報をルータで保持して、その情報を用いてパケットの処理を行う方式などがある。これらの方式を用いると高い公平性を得ることができるが、フロー数に比例した処理を必要とする。そのため、フロー数が多くなると、十分な実行速度が得られない可能性がある。一方、SRED、及び次章で提案する ZL-RED では、フローごとの情報を持たず、固定サイズの Zombie List を用いるため、フロー数の増加に対してその処理が大きくならないという利点を持つ。

3 ZL-RED: Zombie Listed RED

本章では、2 章で述べた SRED の持つ特長を維持しながら、SRED では改善することのできないフロー間の公平性を向上させる方式として、ZL-RED (Zombie Listed RED) を提案する。ZL-RED は、他のフローよりも高いレートでパケットを送出している mis-behaving フローを検出し、そのフローのパケットの廃棄率をより高く設定することより、公平性を向上させる。

ZL-RED では、SRED が持つ Zombie List はそのまま利用し、まず基本となるパケット廃棄率を式 (1)-(3) を用いて求める。次に、Zombie List 内の全ての Zombie の (カウント+1) の和 T(これを Total Occurrence と呼ぶ) と、到着パケットのフローに相当する Zombie の (カウント+1) の和 F(これを Occurrence と呼ぶ) を求める。図 3 では、ルータにフロー ID が 1 のパケットが到着した場合の例を示している。この場合、Zombie List 内で ID が 1 のフローの Occurrence は、$F = (26+1) + (20+1) + (2+1) = 51$ となり、Total Occurrence は $T = (21+1) + (26+1) + (6+1) + (3+1) + (20+1) + (10+1) + (3+1) + (2+1) = 99$ となる。

そして、以下の式が成立する場合に、到着パケットのフローは mis-behaving フローであると判断する。

$$F > T \times P(t) \tag{4}$$

$P(t)$ の逆数が推定コネクション数となるため、式 (4) の

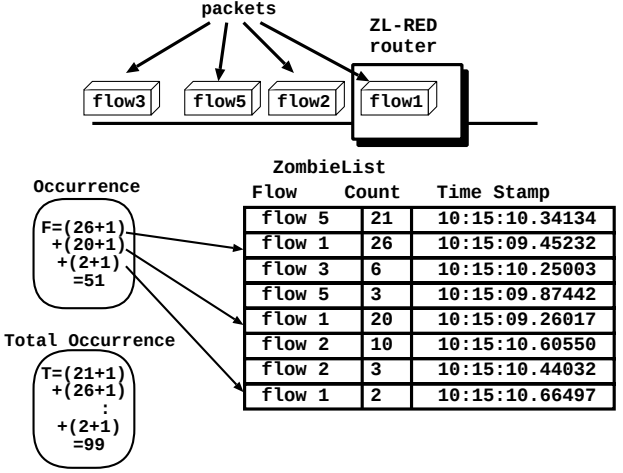


図 3: Occurrence と Total Occurrence の計算

右辺は、Zombie List 内の各フローの Occurrence の平均値を表す。そのため、式 (4) が成立すると、そのフローのパケット到着レートが、平均パケット到着レートより高いことを表す。従って、式 (4) を用いることによって、mis-behaving フローを検出できると考えられる。そこで、F と $T \times P(t)$ の比を用いて、以下のようにパケット廃棄率を変化させる。

$$p_4 = \begin{cases} p_3 \times \frac{F}{T \times P(t)} & \text{if} \quad F \leq T \times P(t) \\ p_3 \times \frac{F}{T \times P(t)} \times a & \text{if} \quad F > T \times P(t) \end{cases} \tag{5}$$

a は、mis-behaving フローのパケットをより高い確率で廃棄するためのパラメータで、$a \geq 1$ である。

最後に、全体のパケット廃棄率が、SRED のそれ (式 (3)) に等しくなるように、パケット廃棄率を以下のように調整する。これは、式 (5) によって mis-behaving フローのパケット廃棄率が変化すると、全体のパケット廃棄率も変動し、SRED の特長の一つであるキュー長の安定性が失われるためである。

$$p_5 = p_4 \times \frac{p_3}{P_{avg}(t)} \tag{6}$$

ここで、$P_{avg}(t)$ は、これまでの平均パケット廃棄率である。以上のアルゴリズムにより、平均パケット廃棄率を SRED と同等に維持しながら、mis-behaving フローのパケットをより高い廃棄率で廃棄することができるため、フロー間の公平性の向上が期待できる。

また、SRED では、パケットの確率的廃棄を開始するバッファ内パケット数の閾値は、バッファサイズに比例した値となっている (式 (1))。バッファ内パケット数がこの閾値 ($\frac{1}{6}B$) に達するまでは到着パケットは FIFO 規律によって処理されるため、バッファサイズ B が大きく、閾値も大きい場合には、FIFO 規律が原因で公平性が著しく低下する [8]。従って、ZL-RED においては、式 (7) に示すようにパケットの確率的廃棄を始める閾値をバッファサイズに依存しない定数 th_{min} とする。

$$p_1' = \begin{cases} 0 & \text{if} \quad 0 \leq q_{len} < th_{min} \\ \frac{1}{4} \times p_{max} & \text{if} \quad th_{min} \leq q_{len} < \frac{1}{3}B \\ p_{max} & \text{if} \quad \frac{1}{3}B \leq q_{len} \end{cases} \tag{7}$$

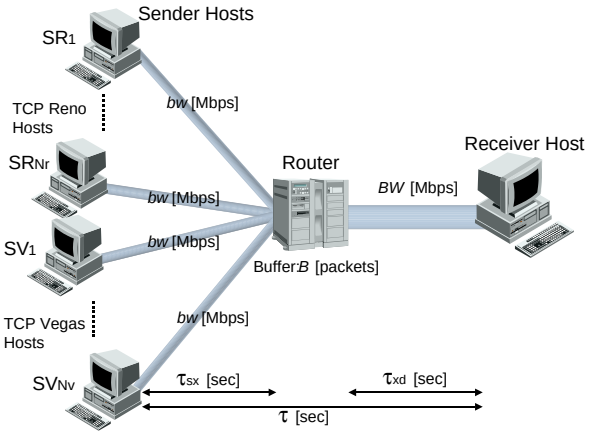


図 4: ネットワークモデル

4 数値例

本章では、3 章で提案した ZL-RED を用いたシミュレーション結果を示し、ZL-RED の有効性の評価を行う。

4.1 ネットワークモデル

本章で用いるネットワークモデルを図 4 に示す。モデルは TCP Reno または TCP Vegas を用いる送信ホスト (SR_1, ... SR_{Nr}、SV_1, ... SV_{Nv})、受信ホスト、ボトルネックルータ、及びそれらを接続するリンクからなる。送信ホストとルータとの間のリンクの帯域はすべて等しく $bw = 45$ [Mbps]、ルータと受信ホストとのボトルネックリンクの帯域を $BW = 45$ [Mbps]、ボトルネックルータのバッファサイズを B [packets]、送信ホスト SR_1, ... SR_{Nr} とボトルネックルータとの間の伝搬遅延時間を $\tau_{rx} = 20$ [msec]、送信ホスト SV_1, ... SV_{Nv} とボトルネックルータ間との伝搬遅延時間を τ_{vx} [msec]、ボトルネックルータと受信ホストとの間の伝搬遅延時間を $\tau_{xd} = 20$ [msec] とする。また、ルータのバッファにおけるパケット処理規律として、2 章で説明した SRED と、本稿で提案する ZL-RED を用いる。また、シミュレーション時間を 500 [sec] 、SRED、ZL-RED 共通のパラメータとして $p_{max} = 0.15$、ZL-RED のパラメータとして $th_{min} = 5$ [packets]、$p_{swap} = 0.25$ とし、送信ホストが TCP を用いてデータを送信したときの平均スループット、及びパケットロス率の比較を行い、フロー間の公平性に関する評価を行う。

4.2 TCP のバージョン間の公平性

本節では、送信ホスト SR_1, ... SR_{Nr} は TCP Reno を用い、送信ホスト SV_1, ... SV_{Nv} は TCP Vegas を用いた場合の評価結果を示す。ここでは、送信ホストとボトルネックルータ間との伝搬遅延時間は $\tau_{vx} = \tau_{sx} = 20$ [msec] とし、その場合の TCP Reno コネクション、TCP Vegas コネクションのそれぞれのコネクション数と、TCP Reno/Vegas コネクションの平均スループットの関係を図 5(a)、5(b) 及び 5(c) に示す。また、図 5(d)、5(e) 及び 5(f) はパケットロス率を示している。ここでは、SRED 及び ZL-RED の Zombie List のサイズ (エントリ数)m は 1000 とし、ルータのバッファサイズをそれぞれ 1000 [packets] (図 5(a)、5(d))、5000 [packets] (図 5(b)、5(e))、10000 [packets] (図 5(c)、5(f)) とした場合の結果を示している。

これらの図から、SRED 方式ではルータのバッファサ

イズに関係なく TCP バージョン間の公平性を維持することができず、特にバッファサイズが大きくなると、公平性が悪化することがわかる。これは、3 章で述べたように、SRED においては確率的なパケット廃棄を始めるバッファ内パケット数の閾値 th_{min} が、バッファサイズに比例しているため、バッファサイズが大きくなると、同時に閾値も大きくなり、th_{min} までのバッファ領域は完全な FIFO 規律で処理されるためである。FIFO 規律における TCP のバージョン間の不公平性は、TCP Reno と TCP Vegas の輻輳制御方式の本質的な違いが原因で発生することが [8] で示されている。

また、ZL-RED においては、バッファサイズに関係なく公平性が改善されており、特にコネクション数が多いときに効果が大きいことが分かる。これは、式 (4)、(5) によって、送信パケット数が平均より多いフロー、つまりここでは TCP Reno コネクションのパケットがより高い確率で廃棄されるため、TCP Reno コネクションのウィンドウサイズが小さくなり、TCP Vegas との差が小さくなるためである。また、パラメータ a の値を変化させることで、mis-behaving フローのパケット廃棄率を大きくすることにより、さらに公平性を改善できる (図 7) が、a を大きくすると全体のパケットロス率が大きくなり、スループットの低下の原因となる可能性があるため、慎重に設定する必要がある。

次に、SRED 及び ZL-RED の Zombie List のサイズを 10、100、1000、10000 エントリとしたときの結果を図 6 に示す。図 6(a) 及び 6(b) は、TCP Reno コネクション及び TCP Vegas コネクションのコネクション数と、各バージョンの平均スループット及びパケットロス率との関係をそれぞれ示している。これらの図から、Zombie List サイズが 100 エントリ以上であれば、実際のコネクション数が Zombie List のサイズよりも大きい場合においても、公平性には大きな影響を与えないことがわかる。これは、図 2 に示したように、Zombie List のサイズがある程度大きければ、推定フロー数は Zombie List のサイズに依存しないためである。しかし、フロー数に対して Zombie List のサイズが極端に小さい場合は、推定フロー数が実際のフロー数よりも小さくなり (図 2)、mis-behaving フローの検出が効率的に行えないため、バージョン間の不公平性が大きくなっている。従って、Zombie List のサイズはフロー数に対してある程度の大きさに設定する必要があるといえる。

4.3 リンクの伝搬遅延時間が異なる環境での公平性

本節では、全ての送信ホストで TCP Reno を用い、送信ホスト SR_1, ... SR_{Nr} とボトルネックルータとの間の伝搬遅延時間 τ_{rx} と、送信ホスト SV_1, ... SV_{Nv} とボトルネックルータとの間の伝搬遅延時間 τ_{vx} が異なる場合の評価結果を示す。

図 8(a)、8(b) は、送信ホスト SV_1, ... SV_{Nv} とボトルネックルータとの間の伝搬遅延時間 τ_{vx} と、TCP コネクションの平均スループットの関係を示している。また、図 8(c)、8(d) はパケットロス率を示している。ここで、SRED 及び ZL-RED の Zombie List のサイズ (エントリ数) は 1000 エントリとし、TCP のコネクション数をそれぞれ 10 本 (図 8(a)、8(c))、100 本 (図 8(b)、8(d))、とした場合の結果を示している。

これらの図から、SRED 方式ではコネクション数に関係なく、公平性を維持することができず、特に伝搬遅延時間の差が大きくなると、公平性が悪化することがわかる。ZL-RED の場合においても、伝搬遅延時間の差が大きくなると公平性が若干悪化するが、SRED を用いた場合に比べて公平性は改善されており、また、a の値が大きくなるほど、高い公平性を示している。これは、4.2 節で述べたように、式 (4)、(5) によって、送信パケット数が平均より多いフロー、つまりここでは送信ホスト SR_1, ... SR_{Nr} の TCP コネクションのパケットがより高い確率で廃棄されるため、送信ホスト SR_1, ... SR_{Nr} の TCP コネクションのウィンドウサイズが小さくなり、スループットが送信ホスト SV_1, ... SV_{Nv} のスループットとの差が小さくなるためである。

5 ネットワークプロセッサへの実装

5.1 ネットワークプロセッサについて

インターネットの利用の拡大により、QoS (Quality of Service) などさまざまな要求が生じており、現在の ASIC (Application Specific Integrated Circuit) ベースのプロセッサでは、そのような要求の変化に迅速に対応できない。そこで、これらの要求に対する柔軟性やスケーラビリティを持たせるために、基本的な枠組のみをハードウェアで提供し、上位の処理はソフトウェアで行うネットワークプロセッサが現在注目を浴びている。本章では、インテル社製ネットワークプロセッサ IXP1200 [15] の仕組みを簡単に紹介し、IXP1200 上へ ZL-RED を実装する際に考慮すべき点についての検討を行う。

5.2 ZL-RED の IXP1200 上への実装

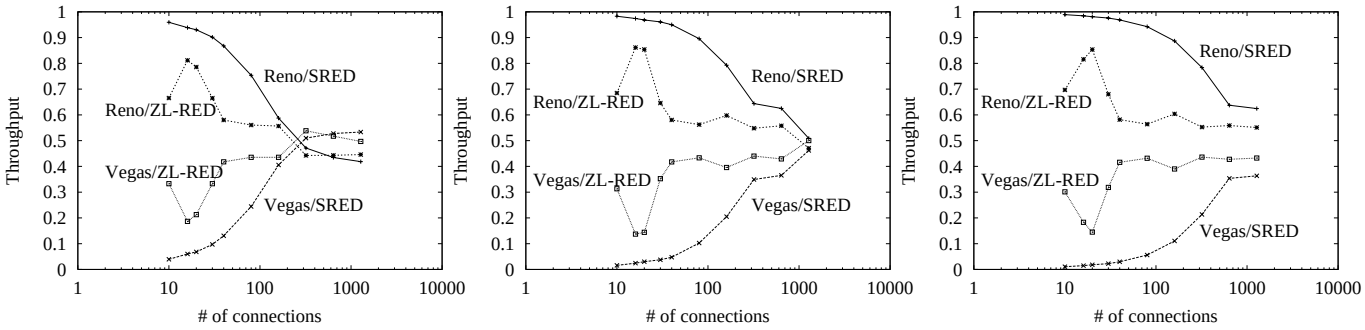
5.2.1 スレッドの割り当て

IXP1200 はパケット転送処理のために、6 つマイクロエンジンで 24 個のスレッドを実行できる。パケット受信処理では、パケットの受信、エラーチェックやルーティングテーブルの検索、出力ポートの決定といった様々な処理を行うのに対して、送信処理については、ポートにパケットを出力するという処理のみであるため、受信処理に多くのスレッドを割り当てる。また、一つのマイクロエンジン上のスレッドは、プログラムメモリを共有するが、プログラムメモリの容量は限られているため、一つのマイクロエンジン上では同一の処理を行うものとする。このようなことから、ZL-RED の実装においては、16 個のスレッドをパケット受信用に割り当て、8 個をパケット送信用に割り当てる。

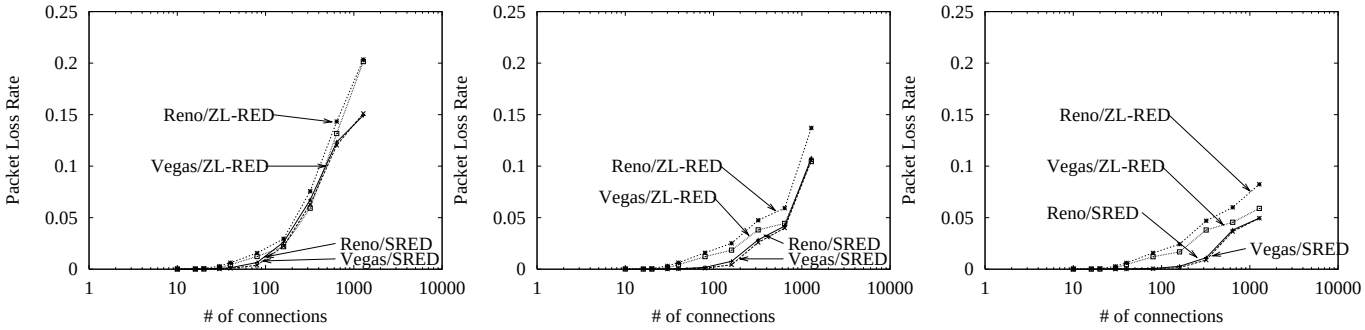
5.2.2 メモリの利用方針

SRAM/SDRAM ユニットがマイクロエンジンからメモリ処理のコマンドを受け取り、そのコマンドを実行し、マイクロエンジンがその結果を受け取るまでには、30 から 50 クロックサイクルを要する。IXP1200 では、マイクロエンジンは SRAM/SDRAM に対してコマンドを発行すると、その結果を待たずに以降の処理を続行することができる。SRAM/SDRAM ユニットはコマンドの処理が完了すると、マイクロエンジンにシグナルを送信し、そのシグナルによってマイクロエンジンはコマンドが完了したと判断する。そのため、あるスレッドがメモリアクセスを行っている間、別のスレッドに処理を切替えることで、パケット処理能力の向上が期待できる。

また、SRAM ユニットと SRAM 間は 32 ビットバスで結ばれており、400 [Mbyte/sec] の速度でデータ転送を行うことができる。また、SDRAM では 64 ビットバスを用いて 800 [Mbyte/sec] の転送速度を実現している。一方、単一コマンドの処理時間は、SRAM では 30 サイクル、SDRAM では 40〜50 サイクル必要となる。その

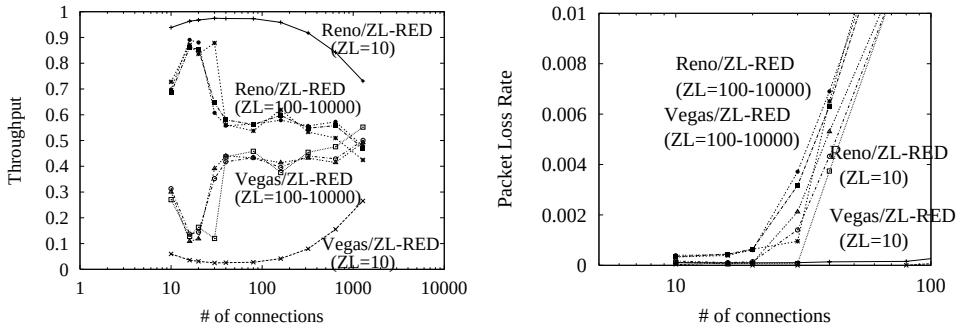


(a) スループット:B=1000 [packets], m =1000 エントリ　(b) スループット:B=5000 [packets], m=1000 エントリ　(c) スループット:B=10000[packets], m=1000 エントリ



(d) パケット廃棄率：B=1000 [packets], m=1000 エントリ　(e) パケット廃棄率：B=5000 [packets], m=1000 エントリ　(f) パケット廃棄率：B=10000 [packets], m=1000 エントリ

図 5: SRED と ZL-RED を用いたときの TCP の各バージョンのスループット及びパケット廃棄率



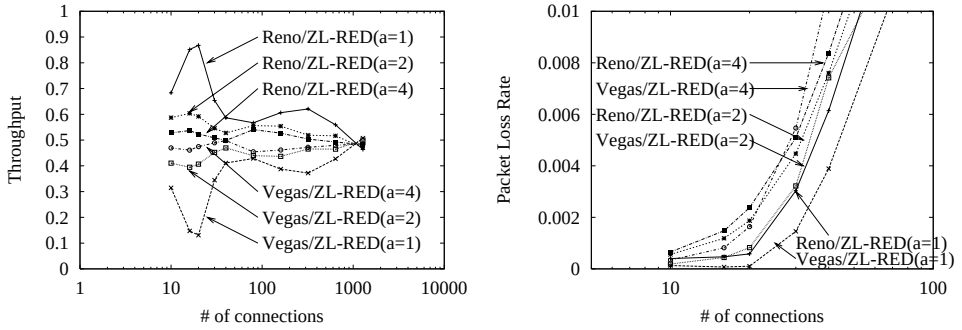
(a) スループット：B=5000　(b) パケット廃棄率：B=5000

図 6: 異なる Zombie List サイズを用いたときの TCP の各バージョンのスループット及びパケット廃棄率

ため、SDRAM にはアクセス頻度は少ないがサイズが大きなパケットのペイロード部分などのデータを格納し、SRAM にはルーティングテーブルや、パケットのヘッダ部分など、頻繁にアクセスされる情報を格納するというように、SRAM と SDRAM を効率的に使い分けることで高速なメモリアクセスが実現できると考えられる。

5.2.3 Zombie List の検索

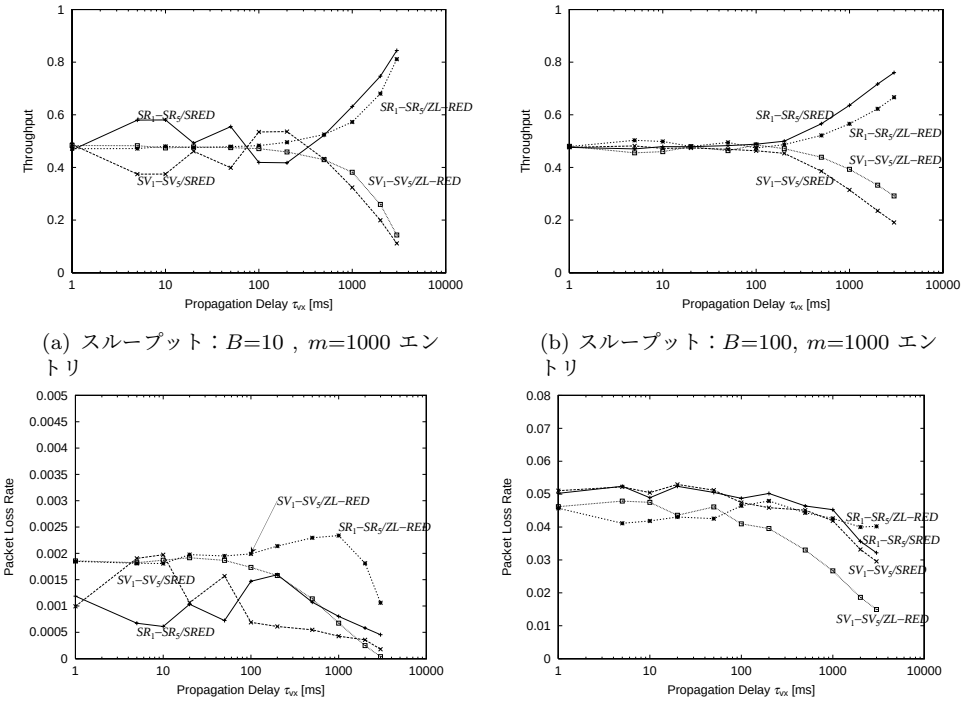
次に、Zombie List の実現についての検討を行う。パケット送信用のバッファは各出力ポートごとに設けられるが、Zombie List は SRAM 内に 1 つ用意し、ルータに到着する全てのパケットに対して、その Zombie List を使用する。ZL-RED では、パケットが到着するたびに、Total Occurrence と、そのパケットのフローの Occurrence を計算する必要がある。パケットの到着ごとに、Zombie List 全体を調べて Occurrence と Total Occurrence の計算を行うと、Zombie List のサイズに比例した時間が必要となり、パケット処理能力の低下を招くため、Zombie List の Total Occurrence については、パケットの到着のたびに計算するのではなくメモリ上にその値を保持することで処理能力の向上を図る。しかし、到着パケットのフロー ID から Occurrence を求める場合、フロー ID として取り得る値の範囲に対応する領域を配列として確保することは現実的ではない。そこで、フローの Occurrence を高速に計算するために、フロー ID からそのフローの Occurrence を検索するルックアップテーブルを SRAM 内に用意し、テーブルのエントリには、Zombie List に存在するフローの Occurrence を格納する。フローの種類が Zombie List のサイズよりも多くなることは無いため、ルックアップテーブルは Zombie List のサイズに対応した固定サイズになる。到着パケットがヒットした場合、到着パケットのフロー ID と、ランダムに選択された Zombie のフロー ID は一致するため、ルックアップテーブル内でそのフローに該当するエントリを一箇所書き換える。ヒットしなかった場合、選択した Zombie の



(a) スループット：B=5000 [packets], m=1000 エントリ

(b) パケット廃棄率：B=5000 [packets], m =1000 エントリ

図 7: 異なる a を用いた場合の TCP の各バージョンのスループット及びパケット廃棄率



(a) スループット：B=10 , m=1000 エントリ

(b) スループット：B=100, m=1000 エントリ

(c) パケット廃棄率：B=10, m=1000 エントリ

(d) パケット廃棄率：B=10, m=1000 エントリ

図 8: 伝搬遅延時間を変化させたときの TCP のスループット及びパケット廃棄率

情報を破棄し、到着パケットの情報で上書きするために、ルックアップテーブル内の 2 つのエントリを書き換える必要がある。

フロー ID とルックアップテーブルのエントリとのマッピングは、IXP1200 の FBI ユニット内にあるハッシュモジュールを用いることで高速に処理できる。ハッシュモジュールでは、48 ビットまたは 64 ビットのハッシュキーをハードウェアで生成できるため、本実装では、送信側の IP アドレスと、受信側の IP アドレスの対をフロー ID として使用する。つまり、フロー ID は 64 ビットで表現される。ハッシュを用いる場合に発生する、コリジョンに対する処理としては、オープンハッシュ法とクローズドハッシュ法の 2 種類の方式が一般に知られているが、オープンハッシュ法は、ハッシュ値が等しいエントリを線形リストでつなぐため、ポインタ操作が必要となり、メモリへのアクセス回数が増えるという欠点がある。クローズドハッシュ法は、コリジョンが発生すると、キーを異なるハッシュ関数を用いて再びハッシュを行う。本実装では、オープンハッシュ法ではなく、このクローズドハッシュ法を用いる。また、ルックアップテーブルのサイズは Zombie List のサイズの 2 倍とすることで (図 10)、コリジョンの発生を少なく抑える。現在実装中で、評価実験については今後報告する。

6 まとめと今後の課題

本稿では、ルータにおいてフロー間の公平なサービスを実現するための方式として、SRED で用いられている Zombie List を使い、mis-behaving フローを検出してそのパケット廃棄率を高く設定することで、フロー間の公平性を向上させる ZL-RED (Zombie Listed RED) を提案した。さらに、シミュレーションによって ZL-RED の性能評価を行い、TCP のコネクション間の公平性が大きく向上することを示した。しかし、特にコネクション数が少ない場合には、公平性の改善の程度は小さく、さらに公平性を向上させるためには、パラメータ調整を行う必要があるが、全体のパケット廃棄率が高くなること

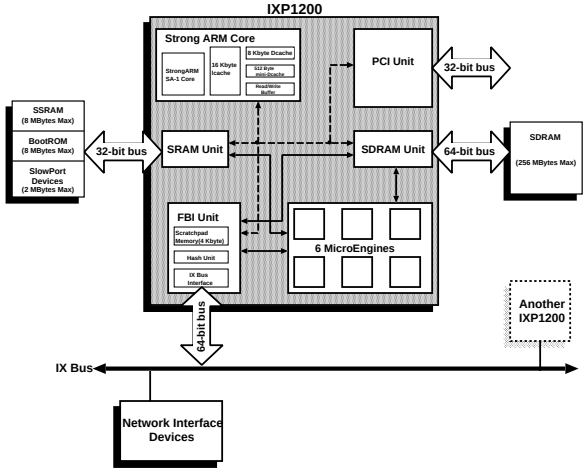


図 9: intel IXP1200 ネットワークプロセッサ

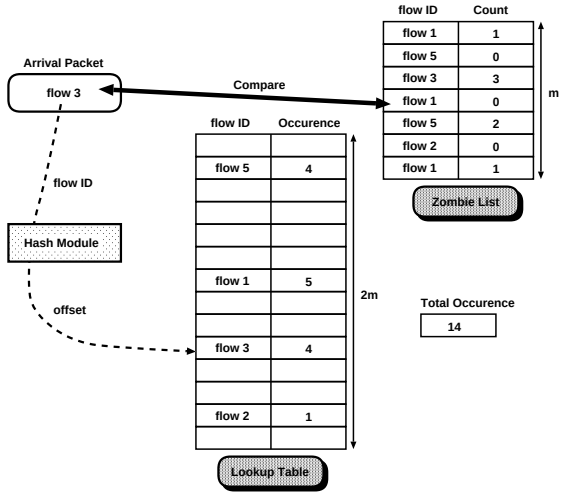


図 10: Lookup table for ZL-RED

が明らかになった。また、ZL-RED をネットワークプロセッサ IXP1200 上に実装する際に考慮すべき問題についての検討を行った。今後の課題としては、IXP1200 上に ZL-RED を実装し、ZL-RED が持つ高速性と公平性を実装実験によって示すことが挙げられる。

謝辞

本研究の一部は、日本学術振興会未来開拓学術研究推進事業における研究プロジェクト「高度マルチメディア応用システム構築のための先進的ネットワークアーキテクチャの研究」、科学技術庁の平成 10 年度科学技術振興調整費による「高度医療ネットワークに関する研究調査」、通信・放送機構「次世代広帯域ネットワーク利用技術の研究開発プロジェクト」、及び (財) 電気通信普及財団の研究助成「超高速ネットワークのためのトランスポート層プロトコルに関する研究」によっている。ここに記して謝意を表す。

参考文献

[1] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet," *IEEE/ACM Transactions on Networking*, vol. 6, August 1999.

[2] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Reading, Massachusetts: Addison-Wesley, 1994.

[3] L. S. Brakmo, S. W.O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proceedings of ACM SIGCOMM '94*, pp. 24–35, October 1994.

[4] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global Internet," *IEEE Jounal on Selected Areas in Communications*, vol. 13, pp. 1465–1480, October 1995.

[5] J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, "Evaluation with TCP Vegas: Emulation and experiment," *ACM SIGCOMM Computer Communications Review*, vol. 25, pp. 185–195, August 1995.

[6] U. Hengartner, J. Bolliger, and T. Gross, "TCP Vegas revisited," in *Proceedings of IEEE INFOCOM 2000*, March 2000.

[7] J. Mo, R. J. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proceedings of IEEE INFOCOM '99*, March 1999.

[8] K. Kurata, G. Hasegawa, M. Murata, "Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas," to be presented at *INET2000*, January 2000.

[9] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[10] 倉田 謙二, 長谷川 剛, 村田 正幸, "TCP バージョン間の公平性向上のための RED の改善方式に関する検討," **電子情報通信学会 技術研究報告** (SSE2000-52), pp. 13–18, June 2000.

[11] T. J. Ott,T. V. Lakshman,and L. Wong, "SRED: Stabilized RED," in *Proceedings of IEEE INFOCOM '99*, March 1999.

[12] 倉田 謙二, 長谷川 剛, 村田 正幸, "TCP バージョン間の不公平性を解消するパケット廃棄方式," **電子情報通信学会技術研究報告** (SSE2000-159), pp. 37–42, October 2000.

[13] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round robin," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 375–385, June 1996.

[14] D. Lin and R. Morris, "Dynamics of random early detection," in *Proceedings of ACM SIGCOMM'97*, pp. 127–137, October 1997.

[15] Intel IXP1200 Network Processor, available at *http://developer.intel.com/design/network/IXP1200.htm*.