L_044

ネットワークシステムとプロセススケジューラとの協調による UDPパケット破棄の削減

Reducing UDP packet drops by Cooperating with Network System and Process Scheduler

山本 紫乃† 楢崎 修二†
Shino Yamamoto Shuji Narazaki

1 はじめに

近年では，Gigabit Ethernetの導入やQoS，プロトコル等の研究開発が進んでいる．その結果，ネットワーク帯域が飛躍的に増加し，画像や音声等による大容量の通信が可能になったため，エンドホストへの負荷の集中が通信全体のボトルネックとなっている．こうした現状に対し，エンドホストのパケット受信処理の高速化や，その安定性を改善することが求められている．

そこで本研究では，エンドホストのパケット受信処理を行うオペレーティングシステム（以下OS）に注目した．OS全体ではプロセスに対して資源割当てを行う方針に従っているが，パケット受信処理に関してはそうではないため，エンドホスト全体の効率が悪化することがある．

本論文では，UDPのパケット受信処理での効率の悪化を解決するため，OSのネットワークシステムとプロセススケジューラとが協調した解決策の提案を行う．これにより，エンドホストでのパケット受信処理の高速化を実現するとともに，安定性も改善した．

2 パケット受信処理

本章では，パケット受信処理について述べる．なお本論文ではパケットについて，ネットワークシステムでのプロトコル処理の終了後はデータと呼ぶことにする．

パケット受信処理の特徴は，パケットがNIC（Network Interface Card）に到着する時をOSが予期できないことである．そのため，ネットワークシステムでは，割込みハンドラ経由でパケット受信処理を行うコードを実行する（図1参照）．ネットワークシステムで各パケットに対応するプロトコル処理を実行した後，データを受信プロセスに対応するソケットのキューに格納する．そして，ソケットでデータを待っている先頭のプロセス（図1ではプロセスAとC）が起床する．

これ以後の処理は，プロセススケジューラが行う．プロセススケジューラでは，各プロセスの生成時に決定する静的優先度と，実行可能状態になってからの待ち時間を考慮した動的優先度とに従ってプロセスの実行順序を決定し，実行待ちキューにプロセスを登録している．そして，優先度が高いプロセスから実行し，実行している時に，対応するソケットからデータを取り出す．

†長崎大学 Nagasaki University

表1: 実験の仕様

計測クライアント	
プロセッサ	AMD Athlon(tm) XP 2200+ 1.80[GHz]
メモリ	768 [MB]
OS	Linux 2.6.15
サーバ	
プロセッサ	AMD Sempron(tm) Processer 3000+ 1.80[GHz]
メモリ	1024 [MB]
OS	Linux 2.6.16.1
共通仕様	
distribution	Debian GNU/Linux 3.1
NIC	Realtek Semiconductor Co., Ltd. RTl-8169

このような流れでの現在のパケット受信処理では，以下に挙げる問題がある．

まずネットワークシステムでは，到着順に全てのパケットを公平に処理しているため，OSがパケット処理に使用したCPUやメモリ等の資源をパケットの受信プロセスが使用したとみなすと，プロセス優先度に従った資源割当てを行っているプロセススケジューラの方針と異なり，プロセス優先度を考慮しない問題がある．その結果，優先度の高いプロセスの待ち時間が増加し，システム全体の性能が低下する可能性がある．

さらに，UDPでは通信を保証するためのフロー制御や輻輳制御等を持たないため，OS内でパケット受信処理に要するバッファやキュー等のメモリが枯渇した後も，パケットが到着する可能性がある．しかしOSでは，バッファやキューの閾値を越えて到着したパケットを，その後の処理を行わずに公平に破棄するため，パケット破棄の問題が発生する．このパケット破棄の問題は，プロトコルスタックの入口（図1では破棄1）とソケットのキュー（図1では破棄2）とで起こる．特に，ネットワークシステムでのパケット受信処理が完了しているデータを破棄する破棄2は，そのデータに対するそれまでの資源も無駄になるため，破棄1と比較して無駄が大きい．また，破棄するパケットの受信プロセスの優先度をかえりみないため，受信プロセスの優先度をパケットの優先度（以後，受信プロセスの優先度をパケットの優先度とする）とみなすと，優先度の低いパケットを破棄した場合よりも，優先度の高いパケットを破棄した場合には，システム全体の性能に与える影響が増大する．

ここで実験により，パケット破棄の問題について示す．スイッチングハブを挟んで1対1で通信できるよ

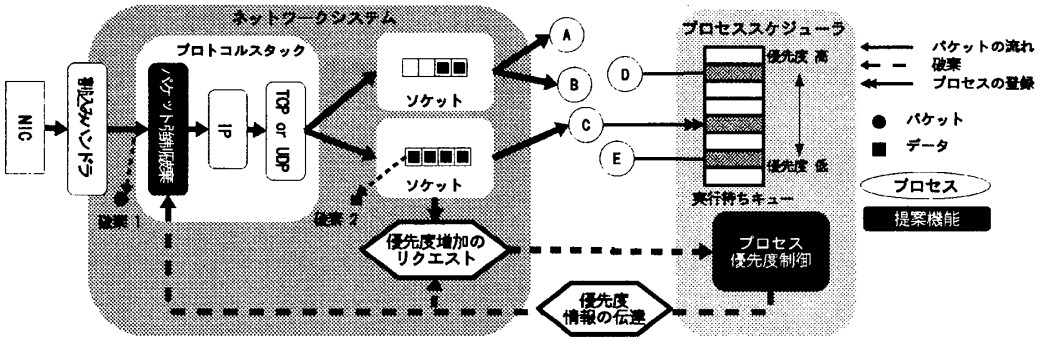

図 1: 提案するアーキテクチャ

表 2: UDP の破棄数 [M packets]

送信数	破棄数	受信数
50,000	11,600	9,502

うに構成し，サーバから計測クライアントに複数のプロセス間で UDP のパケットを送信させ，パケットの受信数と破棄数とを調べた（表 1）．

実験結果を表 2 に示す．破棄 1 と破棄 2 の合計を表す破棄数は，送信数の 32%と非常に多く発生しており，パケット受信処理の安定性のボトルネックとなっている．

2.1 関連研究

Linux に関する研究 PBQs[3] では，宛先となるプロセスの優先度に従った優先度別のキューを使用して破棄する対策をとっている．しかし，ソケットのデータをプロセスに渡すことについては対策を行っていない．そのため，特定のプロセスへのパケットが集中した場合には，PBQs では宛先が同じパケットは優先度別のキューで選別できないため，破棄 2 は発生すると考えられる．

次に，BSD に関する LRP[2] や SRP[1] では，NIC から受け取ったパケットをプロセス毎のキューに格納し，プロセスが実行中であり対応するソケットのキューに空きがある場合のみ，対応するパケットのプロトコル処理を許可する対策をとっている．さらに，各プロトコル毎の受信処理部分を 1 つの階層とし，柔軟な階層構造を実現している．しかし，特定のソケットにパケットが集中すると，多くのパケットが同一のソケットのキューの空きを待つことになるため，そこでも待つためのキューでパケットの破棄が発生すると考える．

また現在，リアルタイム性が高いアプリケーションは，実行時に高い静的優先度を指定して実行すると，優先的に実行できることが知られている．しかし，その他のプロセスの待ち時間が増加し，システム全体に影響を与える問題があるため，動的な制御を行うべきである．

そこで本研究では，未だ解決していないパケット破棄の問題に注目する．

3 提案手法

前述したように，現在の一般的な OS のネットワークシステムのパケット受信処理では，プロセス優先度を考慮していない．また，プロセススケジューラはパケット受信処理の状況を考慮したプロセスの実行順番の決定を行わない．従って，ネットワークシステムとプロセススケジューラとは十分に協調していないため，システム全体の効率が悪化し，パケット受信処理でのパケット破棄の問題を引き起こす原因の 1 つであると考える．

そこで，パケット破棄の問題を解決するため，ネットワークシステムとプロセススケジューラとが協調する手法を提案する．本手法は，ネットワークシステムでパケットの受信プロセスの優先度に従ったパケットの強制的な破棄を行う機能（以下，パケット強制破棄）と，プロセススケジューラでソケットの状態に応じたプロセス優先度の動的な変更を行う機能（以下，プロセス優先度の制御）との 2 つの機能からなる．

パケット強制破棄は，破棄 1 が発生する前に，優先度が低いパケットを確率に基づいて強制的に破棄するというものである．これにより，優先度が高いパケットの破棄を防ぐことができ，強制的に破棄したパケットに使用するはずであった資源を，優先度が高いパケットや他の処理が使用することができる．その結果，プロセス優先度に従った資源割当を行うプロセススケジューラでの方針に近づく．

次にプロセス優先度制御は，優先度の高いプロセスにデータを渡すソケットのキュー長が閾値を越えた時に，そのプロセスの優先度を高くする．このようにプロセスの優先度を変更することにより，ソケットからデータを取り出すまでの待ち時間が短縮し，破棄 2 を防ぐことができる．

この機能では，同じ優先度を持つプロセスの中で，パケット破棄というデータの損失の可能性のあるプロセスのみを優先的に実行することになる．しかし，後述する条件が成立する間のみの一時的な優先であるので，システム全体には大きな影響は出ないと考える．

なお，パケット受信処理においてプロセス優先度を考慮しない問題は，パケットはいつ到着するか分からないため，全てのパケットについて宛先となるプロセスの優先度を考慮する対策を取ることがオーバヘッド

になる可能性がある．このオーバヘッドを考え，提案手法では，パケット破棄の問題の解決策をとる時にプロセス優先度を考慮することとし，以上の2機能を提案する．

3.1 優先ソケットの選択と活性化条件

先に述べたように，それぞれの機能ではプロセス優先度を考慮することにした．従って，プロセス優先度を考慮して選択したソケットを優先するソケットとし，それに関するパケットやプロセスを優先的に処理することで問題解決を行うため，実行に先立ちプロセススケジューラ (図1に示すプロセス優先度制御の部分) において，ソケットを選択しておく必要がある．

ただし，sshやtelnet等のプロトコルを利用する時には，ソケットを開いているプロセスへ渡したデータを他のプロセスに転送するような状況が考えられる．そこで，ソケットを開いているプロセスと，そのプロセスとローカルで通信している全プロセスとを対象として，最も高い優先度を持つものをその優先度をソケットの優先度とする．この優先度が高い上位 n 個のソケットを，優先するソケット $S = \{S_i | i = 1, 2, \cdots, n\}$ とし，n はユーザが設定する事とする．

また，後述の処理では，S_i のポート番号と S_i の優先度の決定に用いたプロセスを何度も参照する必要がある．そこで，計算コストの削除のため，それぞれを変数 $P = \{P_i | i = 1, 2, \cdots, n\}$ と $Q = \{Q_i | i = 1, 2, \cdots, n\}$ に記憶しておく．

S_i の更新は，以下の条件が成立した時に行う．

条件1 S_i が閉じた時

条件2 他のソケットの優先度が，S_i の優先度より高くなった時

この条件1はネットワークシステムで，条件2はプロセススケジューラで判断する．

次に，それぞれの機能の活性化条件を以下に示す．

パケット強制破棄 破棄1が発生した時

プロセス優先度制御 S_i のキュー長が閾値を越えた時

それぞれの条件が成立する時に，対応する機能を実行する．パケット強制破棄の活性化条件は，ネットワークシステムが図1に示すパケット強制破棄の部分をパケットを通過する時に確認する．また，プロセス優先度制御に対する活性化条件は，ネットワークシステム内の各ソケットのキューにパケットを挿入する際に確認する．しかし，パケットの通過や到着のたびに活性化条件を確認するとオーバヘッドになると思われるため，必ず一定時間が経ってから再度確認する．

3.2 パケット強制破棄

この機能のアルゴリズムを図2に示す．図1に示すようにIPの処理の直前で各パケットごとに，パケットのヘッダにある宛先ポート番号が P に含まれない時には，確率に従って強制的に破棄する．

さらに，パケット強制破棄ではパケットのヘッダにあるポート番号を参照しているが，通常の処理でポート番号を参照できるのはUDPの層である．UDPの層で破棄を行うと，各パケットに必要な受信処理の半分以上が終了した後になるため，強制的に破棄するパケットに対してそれまでに使用した資源が無駄になる．そこで，資源の無駄を最小限に抑えるため，UDPの下位層にパケット強制破棄を実装し，この処理の中でもパケットのヘッダ部分から宛先ポート番号を求めることにした．

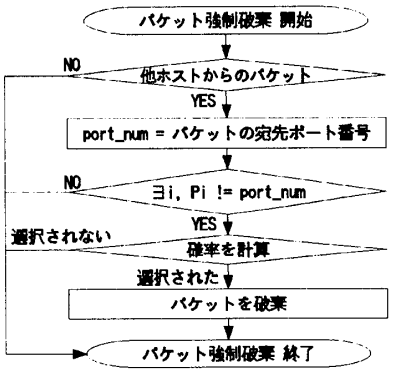

図 2: パケットの強制破棄のアルゴリズム

3.3 プロセス優先度制御

活性化条件が成立した S_i から，図1に示す優先度増加のリクエストを出す．その後，プロセススケジューラがプロセスの優先度を決定する際には，リクエストを出した S_i に対応する Q_i に対しては増分を与え，それ以外のプロセスの優先度は従来通りとする．

また，プロセス優先度の具体的な変更に関しては，システム全体を左右することであり，OSに依存すると考えられるため，一般的に定義するのは難しい．そのため，増分は実装OSごとに調節が必要である．

4 評価実験

本手法の効果を確かめるため，Linuxに実装して評価実験を行った．実験環境や実験方法は，2章での設定と同じとし，オリジナルとそれぞれの機能を実行した時と両方の機能を実装した時とを比較した．

ここで，Linuxへの実装について説明する．まず今回の実装では，優先するソケット数（$= n$）は2とし，それぞれの機能の活性化条件は1秒に1回以下の頻度で確認することとした．

次に，プロセス優先度制御の活性化条件は，S_i のキュー長が上限の半分以上になった場合とした．また，Linuxでのプロセス優先度は，静的優先度と待ち時間で決まる動的優先度（-5〜+5）との合計で決まり，値が小さい方が優先度が高い．そこで今回の実装では，プロセスの優先度制御の活性化条件が成立した時には，そのプロセスの優先度の増分値を常に一番高い5，つまり動的優先度を-5とすることにした．このように，優先度の増分値を本来の動的優先度の範囲内とすることで，システム全体への影響を抑えることを考慮した．

表 3: UDP の実験結果 [M packets]

		実装前	強制破棄	優先度制御	両方を実装
送信数	Q_i				1,000
	全プロセス				50,000
プロトコルスタックでの受信数	全プロセス	9,500	9,490(0.999)	9,520(1.002)	9,440(0.933)
破棄 1 での破棄数	全プロセス (1)	2,130	2,130(1.004)	2,140(1.011)	2,120(1.002)
破棄 2 での破棄数	Q_i	403	215(0.543)	337(0.837)	160(0.397)
	全プロセス (2)	9,440	6,740(0.714)	9,440(0.999)	5,120(0.542)
パケット強制破棄での破棄数	全プロセス (3)	-	4,000	-	4,220
プロセスでの受信数	Q_i	56.9	70.3(1.24)	63.5(1.12)	71.8(1.26)
破棄数の総和 (1)+(2)+(3)		11,600	12,900(1.114)	11,600(1.002)	11,500(0.99)

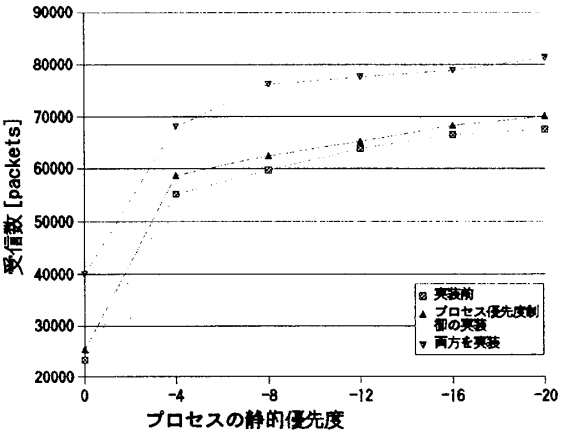

図 3: プロセス優先度制御の効果 [packets]

まず，各破棄数や受信数を表 3 に示す．プロトコルスタックでの全プロセスへの受信数は，両方とも実装した場合でも実装前と比較して，減少している．また，この実験では UDP を使用した一定量のパケット通信を行っているので，破棄 1 が多くなっていることと同様のことを表している．これは，提案手法を実装したためのオーバーヘッドによる影響だと考えられるが，受信数よりも破棄数の総和が減少しているため，無視できる程度の減少である．

これに対し，優先するプロセスの破棄 2 による破棄数と受信数は，実装前と比較して，いずれの場合にも改善が確認できた．

これらの結果は，パケット強制破棄を実装した場合には，優先度の低いパケットを強制破棄したことで，そのパケットに割り当てるはずであったメモリや CPU の資源等を，優先度の高いパケットが使用できた事によると考えられる．また，プロセス優先度制御を実装した場合には，通常の静的優先度よりも優先して CPU 資源を利用でき，ソケットのキューからデータを取り出す速度が速くなった事によると考えられる．さらに，効果が異なる 2 つの機能を同時に利用することで，両方とも実装した場合が一番効果があった．

さらに，プロセスに与えた優先度増分の影響を図 3 に示す．優先度を 5 高くしている今回のプロセス優先度制御の実装での受信数は，それぞれの静的優先度を 5 高くして実行した実装前の受信数に近い結果が出ている．両方を実装した場合には，さらに受信数は増加した．

5 まとめ

本論文では，現在の一般的な OS のネットワークシステムにおける，プロセス優先度を考慮していない問題やパケット破棄の問題を解決するため，ネットワークシステムとプロセススケジューラとが協調する手法を提案した．本手法は，パケット強制破棄とプロセス優先度制御の 2 つの機能からなる．

Linux に実装して評価実験を行った結果，破棄が発生する UDP のパケットに対してはソケットでの破棄を抑えることができ，優先するプロセスに対する受信数が増加した．

本手法の実装の課題として，より最大限の受信効率を発揮するため，現時点では固定している優先するソケット数を，パケットの受信数や破棄数を考慮して動的に変更するようにする必要があると考える．

そして，現時点ではパケットを強制的に破棄する確率は，全ての優先しないプロセスに対して一定であるが，各プロセスの優先度や対応するソケットのキューの状況にあったそれぞれの破棄確率にし，さらにプロセススケジューラの資源割当ての方針に近づけていきたい．

参考文献

[1] Jose Brustoloni, Eran Gabber, Abraham Silberschatz, and Amit Singh. Signaled receiver processing. pp. 211–223. USENIX 2000 Annual Technical Conference, 2000.

[2] Peter Druschel and Gaurav Banga. Lazy receivevr processing(lrp):a network subsystem architecture for server system. 2, pp. 261–276. USENIX Symposium on Operating System Design and Implementation, 1996.

[3] 尾崎亮太, 中山泰一. Linux におけるプロセス優先度に基づく受信処理の実現. 情報処理学会論文誌, Vol. 45, No. 3, pp. 785–793, Mar 2004.