2005－OS－100（11）
2005／8／3

# リアルタイムパケットの通信制御

野村哲弘† 石川裕†

時間制約のあるタスク間の通信の制御について，各メッセージパケット毎に時間制約情報を付加してパケット配送時間をカーネルに明示して保障する方法を提案する．カーネル内で各パケットの送信時間を算出してそれらの時間制約を満たすように送信順序を調整して送出する．これらにより通信の中で起こる優先度逆転現象を解決する．上記機構を Linux 上の Qdisc(キューイング規則) を用いて実装する．様々な時間制約を持つパケットを混在させた環境でもそれぞれの時間制約を満たす通信が行なわれることを示す．

# Communication Control of Real-time Packets

Akihiro Nomura † and Yutaka Ishikawa†

In the control of communication between tasks whose packet's delivery time has deadline, we propose a new method, telling the delivery time information of each packet to kernel to make these deadline constraints be satisfied. The transmission order is determined based on the delivery time. In this mechanism, the priority inversion problem is resolved in the transmission queue. This mechanism is implemented in Linux using the Qdisc (queuing discipline) framework. We show that the communication is done with various deadline constraints satisfied.

## 1. はじめに

今日のネットワーク技術の進歩に伴い，リアルタイム通信への需要は日増しに高まってきている．このような中で，殆どのリアルタイム送信制御アルゴリズムでは “通信の優先度は送信タスクの実行優先度に比例している” という前提の元に成り立っている．この仮定は常に正しいものではなく，各タスクにおけるパケットの依存関係によっては，優先度の逆転現象を発生させてしまうことがある．

本論文ではパケットの時間制約自体をそのパケットの送信優先度として扱う手法を提案する．本手法の適応範囲として想定している環境は，近距離のノード間においてリアルタイム通信と一般の通信が同一のネットワーク上を流れるシステムである．システム上のリアルタイム通信についてはソフトデッドライン通信だけではなく，通信の失敗がシステムの崩壊をももたらし得るハードデッドライン通信も考慮する．このような通信が行われる環境に於いては，時間制約を満たす通信が出来なかった場合に想定されるダメージの大きさに違いの有るような通信が同一ネットワーク上を流れることとなり，重要度の低いパケットの通信が重要度の高いパケットの通信を阻害する可能性が考えられる．

以下では，このような環境でリアルタイム通信が失敗し得る状況に対して通信の両端点のソフトウェアで対応できる手法について考察する．本提案手法は時間制約情報を扱うことに特化しているため，送信処理に於いては単純な待ち行列モデルでパケットを管理する．また，Linux 上で本提案手法を実装し，時間制約を守った通信が行われていることを示す．

† 東京大学
The University of Tokyo

## 2. 提案手法

### 2.1 パケット制御構造

パケットの送出時間制約の保障のためには，そのパケットが通過するネットワークインタフェース上の全てのパケットを制御しなければならない．このため，リアルタイム通信を行なうネットワークインタフェースの物理層の手前にそのインタフェースを通過する全パケットを監視するモジュールを設置する．このモジュールはプロトコル層からのパケットの送信要求を留めておき，デバイスドライバからの送信パケット要求に応じてパケットを配信するものである．本モジュールは2つの待ち行列を持ち，片方は時間制約のあるパケットを，もう片方は時間制約の無いパケットを保持する．パケットは図 1 のような流れで各待ち行列に格納，若しくは時間制約を満たせないとして廃棄される．

### 2.2 リアルタイムパケットの送信可能判定

時間制約のあるパケット各々を待ち行列に入れる時点で以下のようなパラメータを計算し，必要に応じて更新していくことで，新しいパケットの到着時における時間制約の充足可能性を判断する

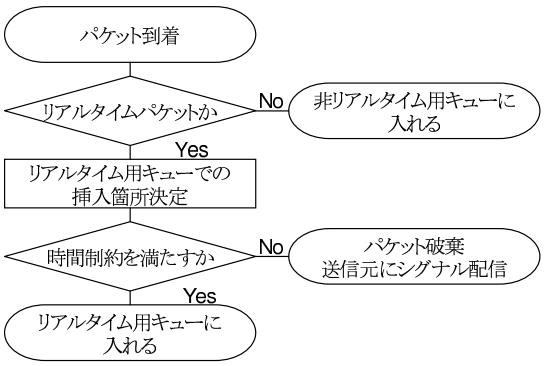


**図 1** パケット到着時の処理イメージ
Fig. 1 Brief image of handling incoming packets

- パケット送信にかかる時間 $T$
- 送信デッドライン時刻 $D$
- 遅延余裕時間 $M$

パケット送信時間 $T$ はパケットのサイズとネットワークインタフェースのバンド幅から計算され，送信デッドライン時刻 $D$ は送信アプリケーションから与えられる値である．ただし，送信デッドライン時刻 $D$ は，送信完了に対するデッドライン時刻である．ユーザが指定するデッドライン時刻は受信側プロセスが実行権を得てパケットを読みこむまでの時間を指す．したがって，ネットワーク上での遅延とタスク切り替え等のコストの上限値を予め引いた値を $D$ とする必要がある．

遅延余裕時間 $M$ は以下に示す条件の元に計算される．

時間制約のあるパケット用の待ち行列を $R$ と表わす．$R$ 上のパケットをデッドラインが近いものから順に $p_1 \cdots p_n$ と表わし，新たに挿入されるパケットを $p_{ins}$ と表わす．また，各パケットのパラメータを $D_j, T_{ins}$ 等と表わすこととする．先頭のパケットがキューから出る時刻を $T_{now}$ と表わす．

$R$ 上の各パケットのパラメータ $M$ は常に以下の式で算出される値を取る．

$$M'_j = (D_j - T_j) - (T_{now} + \sum_{k=1}^{j-1} T_k) \quad (1)$$

$$M_j = \min_{k \geq j} M'_k \quad (2)$$

式 (1) における $M'_j$ はパケット $p_j$ の送信時刻が $T_{now}$ から休むことなく送信した場合に比べてどれだけ遅延してもよいかを示している．実際には先に出るパケットの遅延は後に出るパケットにも伝播するため，この $M'_j$ ではなく，後続の全てのパケット $p_k$ の $M'_k$ の値との最小値である $M_j$ がそのパケットに許される遅延となる．即ち $M$ は式 (3) の条件を満たす必要がある．

$$M_j \leq M_{j+1} \quad (3)$$

式 (2) で定める $M$ についてはこの制約が満たされることが保障されている．この $M$ は**図 2** 上段にあるように，各リアルタイムパケットについての最早送信可能時刻とデッドライン制約下における最遅送信時刻との差として表わすことも出来る．

新たにパケット $p_{ins}$ を $R$ に挿入する場所は，$D_{i-1} \leq D_{ins} < D_i$ を満たす $p_{i-1}$ と $p_i$ の間となる．

上記の方法で求められた値 $M$ を使うと，$p_{ins}$ を $R$ に入れた際の各パケットの時間制約が保証可能かは以下の条件で示される．

$$T_{ins} \leq M_i \quad (4)$$

### 2.3 リアルタイムパケットを受理する場合

式 (4) の条件を満たす場合，パケットを $R$ に挿入する処理は，挿入されるパケット $p_{ins}$ が上記の条件を満たすかを確認し，待ち行列に追加した後に，キュー上の各パケットについて，以下の手順で各パケットのパラメータ $M$ の値を更新することによって行われる．

- $j \geq i$ なる $p_j$，即ち $p_{ins}$ の挿入によって送信時間が遅れる全てのパケットについて $M_j$ の値を $T_{ins}$ だけ減らす．
- $M_{ins}$ に仮の値として式 (1) で与えられる $M'_{ins}$ を代入する．ここで $i > 0$ の場合は式 (1) の代わりに，式 (5) における $M''_{ins}$ で代用することができ，その方が計算のコストが低い．

$$M''_{ins} = D_{ins} - (D_i - M_i) - T_{ins} \quad (5)$$

- 式 (3) の条件を満たすように，$p_{ins}$ からキューを逆に辿りながら，$M_j > M_{j+1}$ の場合には $M_j := M_{j+1}$ としてパラメータ $M$ を更新する処理を $M_j \leq M_{j+1}$ なるパケット $p_j$ に到達するまでくり返す．上の操作の時点で $M'_{ins} \neq M''_{ins}$ となっていた場合は，ここで $M_{i+1}$ から伝播する値が必ず選択されるので，前述の $M'_{ins}$ を $M''_{ins}$ で代用することは問題にならない．

以上の操作によって $M$ が式 (2) の条件を満たすように更新され，パケットの挿入処理が完了する．図 2 はこの一連の動作の前後におけるパラメータ $M$ の変化の一例である．

### 2.4 リアルタイムパケットを破棄する場合

式 (4) の条件を満たさなかった場合はそのパケットを送信しようとすることで待ち行列上の $p_{ins}$ を含むいずれかのパケットが時間制約を満たせなくなることを示している．この場合は挿入しようとするパケットを破棄して，送信元のプロセスにはパケットが時間制約を満たせずに破棄された旨のシグナルを送信する．

### 2.5 非リアルタイムパケットの処理

時間制約の無い通常のパケットはリアルタイムパケットとは別のキューに保存される．このキューではリアルタイムパケットと違い時間制約による整列やパケットの排除の必要性が無いため，単純な FIFO 構造をとっている．

### 2.6 パケットの送出

ネットワークデバイスドライバから送信パケットを出す要求が到着したときに，実時間通信の待ち行列にパケットが有ればその先頭のパケットを，時間制約が守られていることを確認して送出する．なお，理論的には，この時点で時間制約が守られなくなることは無い．リアルタイムパケットが待ち行列上に無いときに

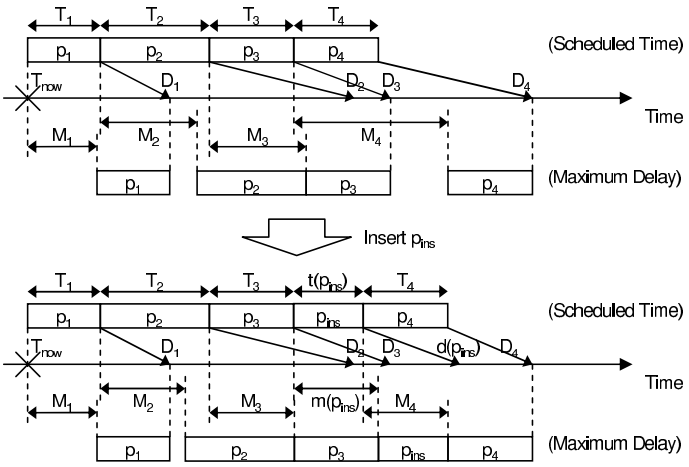


図 2　時間制約を満たすかの判定
Fig. 2　Verification of deadlines in each packets

限り，非実時間のパケットを返す．

このようにして常に時間制約を満たしながらパケットを送出することが出来る．

## 3. 実　　装

上記機構を Linux kernel 2.6 上の IP プロトコルに対して実装する．リアルタイム通信に於けるトランスポート層のプロトコルとしては UDP のようなデータグラムプロトコルが使用されることを想定している．

### 3.1 API

送信側に於いては，リアルタイムパケットと通常のパケットとを区別し，カーネルに時間制約情報を与えるための API が必要となる．新たなシステムコールをシステムに追加すると，本機構を用いていない環境ではプログラムが全く動作しなくなるため，可搬性に欠ける．そこで，新たなシステムコールを導入せずに既存のシステムコール `setsockopt` で特定の IP オプションヘッダを付けるという形でこの情報を与える．ここで IP オプションヘッダに時間制約情報を埋めこむことによって受信側でもこの情報を利用できるという利点が生まれる．

### 3.2 Linux における Qdisc

Linux カーネルには Qdisc(キューイング規則)[1] と呼ばれるパケット制御機構がある．Qdisc はネットワークインタフェースの送信口を通過する全パケットについて優先度付きキューイングや帯域制御などを行うものである．開発者は Linux カーネルモジュールの追加によって新たな Qdisc を定義，導入出来る．Qdisc は Linux カーネルのネットワーク処理部の中で最もデバイスドライバに近い位置にあり，ネットワークデバイスと 1 対 1 の対応を持っているため，デバイスの全送信パケットを制御できる．このことから，本提案機構の実装に Qdisc を利用する．

### 3.3 Real-time Qdisc

本提案機構を実現する Qdisc として Real-time Qdisc を提案する．Real-time Qdisc(**図 3**, **図 4**) は

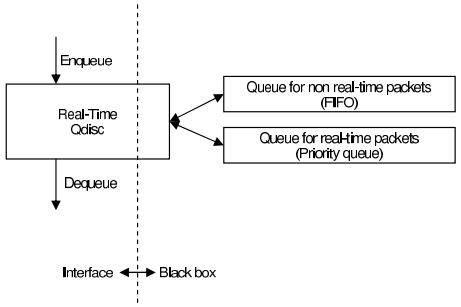


図 3　Real-time Qdisc のデータ構造とインタフェース
Fig. 3　Data structure and interface of real-time Qdisc

上位層から到着したパケットを図 1 に示すアルゴリズムでリアルタイムパケット用のキューと非リアルタイムパケット用のキューに振り分け，下位層からの送信要求に応じてパケットを取りだす．第 3.1 節で述べた時間制約を示す IP オプションヘッダを付加するよう指定されたパケットをリアルタイムパケットとして扱い，それ以外のパケットと IP 以外のパケットは非リアルタイムパケットとして扱う．第 2 章のアルゴリズムでデッドラインを保障したパケットの送信が出来ないと判断された場合には，送信元プロセスに対して `SIGARG` シグナルを送信する．このシグナルはソケットにおける緊急事態の発生を報告する POSIX シグナルであり，受信したプロセスのデフォルト動作はシグナルの無視である．

さらに，Qdisc の後に通ることとなるネットワークインタフェースのドライバ上のバッファを考慮して，送信量を元にバッファの残り容量を推定して送信にかかる時間の計算に反映させることにより，より正確な送信時間の推定を行えるようにする．また，このバッファ内に大量のパケットが溜まっている場合，急に到着したデッドラインが近いリアルタイムパケットが送信要求の時点ですでに時間制約を満たせない状態

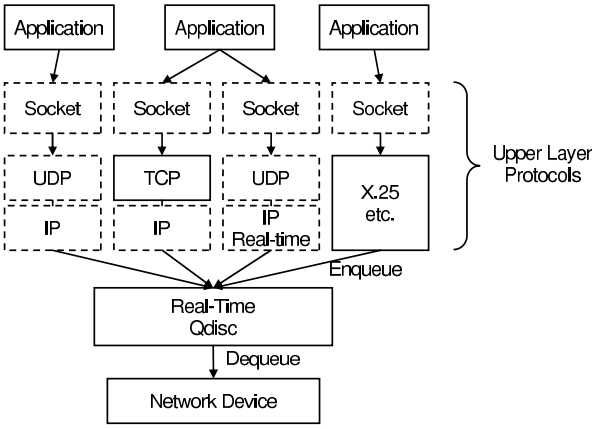


図 4　Real-time Qdisc と前後層の位置関係
Fig. 4　Relations between the Real-time Qdisc and the other layer

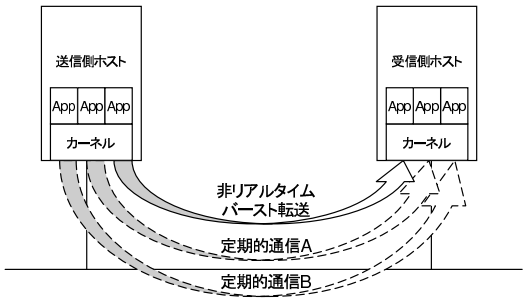


図 5　実験における接続の概要
Fig. 5　Connections in this experiment

表 1　実験に用いた実時間パケット
Table 1　Real-time packets in this experiment

| | デッドライン時間 | 送出頻度 |
|---|---|---|
| 通信 A | 10 ms | 30 packets / 3ms |
| 通信 B | 1 ms | 10 packets / 3ms |

になることが考えられる．この問題を回避するため，Real-Time Qdisc から出ていくパケットについては一定のバンド幅でペーシングを行い，ネットワークインタフェースのドライバ上のバッファに存在するパケットの量を常に一定量以下に保っている．なお，ネットワークインタフェースの送信割り込みが一定以上遅延しない環境では，このバッファに溜めておくパケットの量をある程度制限してもバッファーが空になって実際の通信路が空いてしまう問題は発生しない．

Linux Qdisc には分類したパケットの一部のキューイングをさらに別の Qdisc に委譲することができるクラスフルな Qdisc も存在する．しかしながら，今回の目的は各パケットがネットワークインターフェースから送出されるタイミングを掌握，制御することにあり，上流の Qdisc が存在するとこの性質が崩れるため，本実装の Qdisc は他の Qdisc から呼び出されることを考慮しない．また，リアルタイムパケットの送出順とバックログを制御する必要から，リアルタイムパケットの分類後に他の Qdisc に委譲することも考慮せず，処理は Real-Time Qdisc で完結させる．

## 4. 評　　価

非リアルタイム通信及び時間制約の緩いリアルタイム通信が通信路を占有している状態でもそれよりも時間制約の厳しい通信が時間制約を損うこと無く実行できることを示す．

### 4.1 評 価 環 境

通信実験は Intel Pentium 4 3.06 GHz プロセッサ，主記憶 512 MB，Intel E1000(1Gbps) NIC を持つ 2 台のホストを 1 台の 1000BASE-T のスイッチに接続して行った．通信に用いた OS は Linux 2.6.11 に上記実装のカーネルモジュールを適用したものであり，送受信するパケットは UDP とした．

時刻測定はユーザ空間では `gettimeofday()` システムコールを用い，カーネル空間では `do_gettimeofday()` 関数を用いた．いずれも x86 アーキテクチャ上のタイムスタンプカウンタ (TSC) をクロックソースとしており，$\mu$s の精度での時刻を記録した．2 台のホスト間でこの測定精度のレベルでの時刻の同期を取ることは事実上不可能であるため，別途時間測定用のパケットを用いて後で補正をかけた．

### 4.2 評 価 内 容

**図 5** のように予め時間制約の無いパケットをバースト通信させた通信路に於いて，複数種の時間制約のあるパケットを別プロセスから定期的に送出し，後者の通信の各段階で各パケットごとのタイムスタンプを記録した．受信側ではバースト転送と各測定対象の通信の受信に別々のプロセスを用いた．実験に用いたリアルタイムパケットはそれぞれ**表 1** で示されるものである．これらを送出するプロセスはそれぞれ 3ms スリープする毎に一定の時間制約を持つパケットを一定数送出する．各パケットの UDP データグラムのサイズは 1400 バイトである．

測定対象の通信では以下に示す時点での時刻を測定した (**図 6**)．

(1) 送信側アプリケーションでの `send` システムコール直前
(2) 送信側カーネルの Real-Time Qdisc の `enqueue` 関数の先頭
(3) 送信側カーネルの Real-Time Qdisc の `dequeue` 関数の先頭
(4) 受信側カーネルに測定用に設置したパケットフィルタの `classify` 関数の先頭
(5) 受信側アプリケーションでの `recv` システムコール直後

なお，受信側パケットフィルタはタイムスタンプ取得のみを目的としており，パケットのフィルタリングは行っていない．

各時点の間の所要時間は以下のように解釈できる．

**パケット生成時間**　`send` システムコールを受けてカーネルの UDP 等のプロトコルスタックが `sk_buff`

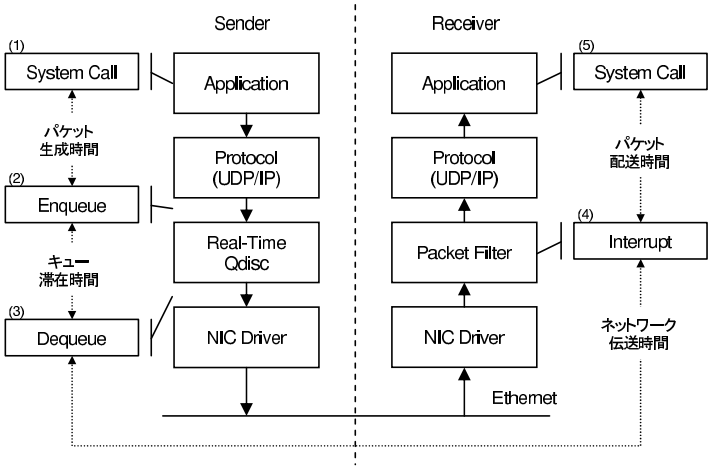


**図 6** 実験で測定したパケット通過タイミング
Fig. 6 Measured timing of transition measured in this experiment

**表 2** パケットの送信に使われた時間 (本機構)
Table 2 Result of Experiment (proposed mechanism)

[$\mu$s]

| | 通信 A | | 通信 B | |
|---|---|---|---|---|
| | 平均 | 最大 | 平均 | 最大 |
| パケット生成 | 6.6 | 31 | 6.8 | 31 |
| キュー滞在 | 2.8 | 23 | 2.8 | 23 |
| ネットワーク伝送 | 99.4 | 213 | 102.0 | 177 |
| パケット配送 | 37.7 | 928 | 40.3 | 158 |
| 合計時間 | 146.5 | 999 | 151.9 | 306 |

通信 A，通信 B の通信パターンを表 1 に示す

**表 3** パケットの送信に使われた時間 (Linux デフォルト)
Table 3 Result of Experiment (default in Linux)

[$\mu$s]

| | 通信 A | | 通信 B | |
|---|---|---|---|---|
| | 平均 | 最大 | 平均 | 最大 |
| パケット生成 | 10.3 | 86 | 10.5 | 57 |
| キュー滞在 | 11582 | 12317 | 11464.7 | 12311 |
| ネットワーク伝送 | 2660.7 | 2957 | 2574.1 | 2848 |
| パケット配送 | 51.8 | 564 | 42.4 | 464 |
| 合計時間 | 14305.6 | 15397 | 14091.6 | 15236 |

通信 A，通信 B の通信パターンを表 1 に示す

構造体上に送信可能なパケットを生成して Qdisc に渡すまでにかかる時間

**キュー滞在時間** Qdisc において FIFO や本手法等のスケジューリングに従うことにより，パケットの送信が他のパケットによって待たされる時間

**ネットワーク伝送時間** パケットが Qdisc を抜けてから送信側ネットワークデバイス上のキューに入り，実際に Ethernet 上をパケットが通過し，受信側ネットワークデバイス上のキューを経て，受信側カーネルで扱える `sk_buff` 構造体としてパケットが捕えられるまでの時間

**パケット配送時間** カーネル上に到着したパケットのプロトコル番号やポート等からそれを受信するプロセスを決定してその受信キューに配送し，当該プロセスがスケジューラの選択によって実行状態になって実際にユーザ空間にパケットが到着するまでの時間

上記の測定を本機構を用いた場合と用いなかった場合の両方に於いて行った．

### 4.3 結　　果

測定の結果，本機構を利用した場合は**表 2**，利用しなかった場合は**表 3** のような測定値が得られた．

送信側の時刻と受信側の時刻の間の比較を伴っているところにおいては実験の開始前と終了後に本機構を用いずに実験と同様な 3 ms 毎の定期送信をバースト通信が無い状況で双方向別々に行い，ネットワーク伝送時間の差から時計のずれを求めた．このとき，実験の開始時と終了時の 90 秒間の間に 2 台の間に約 25$\mu$s の時計のずれが新たに発生していることが観測された．表 2, 表 3 では，実験中の時計のずれは線形で進んでいったというモデルを仮定して，2 台のホスト間の時計のずれに起因すると思われる値を引いた後の結果である．

各表の合計時間の項を見ると，本機構を用いなかった場合には全く実現できなかったバースト通信下でのリアルタイムパケットの低遅延通信が実現できていることがわかる．本機構を用いた場合と用いなかった場合に於いて所要時間が顕著に変わっているのはキュー滞在時間とネットワーク伝送時間である．前者は Real-Time Qdisc による時間制約の有無及びその期限に基づいた優先度つきキューイング，後者は Real-Time Qdisc の後に入るネットワークデバイスのバッファーを考慮に入れた送信制御の結果であると考えられる

次に表 2 に見られる両通信の間での遅延の差異について論じる．時間制約が緩い通信 A のパケットの平均

消費時間の方が時間制約が厳しい通信 B のパケットの平均消費時間よりも全体的に短く，これは期待されている結果とは異なる．また，パケットのキュー滞在時間に差異は無く，表 1 の条件下では両通信間で元々優先度逆転現象が起こっていなかった可能性がある．Real-time Qdisc ではネットワーク伝送時間もネットワークデバイスのバッファのペーシングで間接的に制御しており，最大ネットワーク伝送時間には両通信間で差異が見られることから，両通信間の優先度逆転現象はこの部分で隠蔽されていると考えることも出来る．

また，パケット配送時間は大部分のパケットについては受信タスクの優先度をもとに安定しているが，稀に 900$\mu$s など大きく遅延するパケットが観測され，結果的にその遅延時間が通信全体の最大遅延時間の大部分を占めている．現時点でパケット配送時間を通信のリアルタイム性を元に優先制御する等のことは行っておらず，原因も究明できていない．

## 5. 関 連 研 究

Diffserv[2] は IP 上での QoS 制御のための一般的なフレームワークである．ただし，Diffserv はルータ間の QoS 制御のためのプロトコルであり，通信の両端点間の QoS を保障するものではない．

RCSP(Rate-Cintrolled Static-Priority)[3] や Leave-in-Time[4] は，コネクション指向ネットワークにおいて一定以下の遅延で通信を行う方法を示している．また，その他のリアルタイムパケットの送信機構についての比較は，文献 5) が詳しい．

これらの送信機構に共通して言えることは，各コネクションに対して静的優先度を設定してそこから得られる各優先度クラス間でバンド幅の上限等を規定することによってそれに応じた遅延の保証を得ていることである．ソケットの静的優先度を各パケットの静的送信優先度としている点で，通信の依存関係から発生する一時的な優先度逆転現象に対処する方法がないと言う問題が考えられる．また，通信のデッドラインをそれに附随する処理のデッドラインにまで拡大して適用することを考えたときに，パケット毎にしか優先度設定が出来ないこれらの送信機構を使うことは困難であると考えられる．

RFQ(Rainbow Fair Queueing)[6] は通信のフローに捕われないキューイングを提供する．しかし，中継ルータでパケットが溢れることとなった際に公平にパケットをドロップするため，ハードリアルタイムの通信には向かないと考えられる．

## 6. ま と め

リアルタイムパケットの時間制約をそのまま通信の優先度として扱う通信機構とその実装を示した．時間制約を通信の優先度として直接扱うことによってその保障をより単純な形で簡便かつ確実に行うことが出来る．

また，Linux カーネル上における本機構の実装を示した．Qdisc 機構を利用することによって比較的容易にパケットの送出順制御を行うことが出来る．さらに，Qdisc 以降のバッファに貯まるパケットを正確に見積もることによってリアルタイム通信の送信性能を大幅に向上させることが出来る．

提案方式において解決すべき問題点は以下の通りである．

第 4.3 節で述べたように，今回の実験では 2 つの通信間に優先度逆転現象が起こっていない可能性があるため，実験条件等を見直して本機構が優先度逆転現象を解決しているかを再検証する必要がある．さらに，受信側タスクのスケジューリングを含む配送の段階で大きなパケットの遅延が突発的に起こっており，その原因を究明し，解決する必要がある．

常に時間制約付きのパケットの送信を優先するため，時間制約の無い通常のパットに対する帯域保障が為されていないことが挙げられるが，通常のパケットへの帯域保障をすることによって逆にリアルタイムパケットの送信能力を低下させてしまう可能性が考えられるので，そのようなトレードオフを含めて考える必要がある．

今後の課題として，本実装をリアルタイムスケジューラへと組み込んで協調させることによって，実際にリアルタイム環境上で本機構を利用して，要求，処理，応答の一連の処理全体での時間制約を保障する形のリアルタイム通信が行えることを示すという課題が挙げられる．

**謝辞** 本研究の一部は，科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) の支援を受けた．

## 参 考 文 献

1) Hubert, B.: Linux Advanced Routing & Traffic Control HOWTO, `http://lartc.org/`.
2) Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and Weiss, W.: A framework for differentiated services, IETF RFC 2475 (1999).
3) Zhang, H. and Ferrari, D.: Rate-Controlled Static-Priority Queueing, *INFOCOM (1)*, pp. 227–236 (1993).
4) Figueira, N. R. and Pasquale, J.: Leave-in-Time: A New Service Discipline for Real-Time Communications in a Packet-Switching Network, *SIGCOMM*, pp. 207–218 (1995).
5) Monteiro, E., Quadros, G. and Boavida, F.: ATS - Advance-Time Scheduling: A Service Discipline for Quality of Service Provision.
6) Cao, Z., Wang, Z. and Zegura, E.W.: Rainbow Fair Queueing: Fair Bandwidth Sharing Without Per-Flow State, *INFOCOM (2)*, pp. 922–931 (2000).