社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－124 （7）
2006／3／16

Linuxカーネル上での イーサネット優先度制御・フロー制御機構の実装と評価

堀 武司† 戸田 賢二‡

†北海道立工業試験場 〒060-0819 北海道札幌市北区北19条西11丁目
‡産業技術総合研究所 情報技術研究部門 〒305-8568 つくば市梅園1-1-1 つくば中央第二
E-mail: †horit@hokkaido-iri.go.jp, ‡k-toda@aist.go.jp

あらまし Ethernetの実時間性やパケットロスなどを改善するために，Ethernetに対する優先度制御，フロー制御機構の拡張を検討した．また，提案手法をLinux 2.4カーネル上に実装し，実機を用いて遅延，パケットロス率，帯域制御などの評価試験を行った．

キーワード 実時間Ethernet, 優先度制御, フロー制御, Linuxカーネル

Implementation and Evaluation of Ethernet Priority and Flow-control mechanism on Linux Kernel

Takeshi HORI† and Kenji TODA‡

†Hokkaido Industrial Research Institute, N19 W11, Kita-Ku, Sapporo 060-0819, Japan
‡National Institute of Advanced Industrial Science and Technology Information Technology Research Institute, Tsukuba Central 2, Umezono 1-1-1, Tsukuba, Ibaraki 305-8568, Japan
E-mail: †horit@hokkaido-iri.go.jp, ‡k-toda@aist.go.jp

Abstract To reduce packet latency and packet loss probability of Ethernet, we tried to introduce priority and flow control mechanism to Ethernet. Then we tested packet latency, loss and bandwidth control of the proposed method by using the network consists of four Linux 2.4 PCs.

Keyword Real-time Ethernet, Priority, Flow control, Linux kernel

1. はじめに

Ethernetは今日最も普及したネットワーク規格であり，100Mbps, 1Gbpsの高速通信に対応した機器が安価で入手可能である．利用分野も，PC，ワークステーション間を結ぶLAN構築だけに止まらず，広域ネットワークやSAN(Storage Area Network)などから，ネット家電などの組込み機器まで，幅広い分野への普及が進んでいる．

一方，EthernetにはCSMA/CD(Carrier Sense Multiple Access with Collision Detection)方式に起因する送信の衝突・再送があり，通信の実時間性や到達性を厳密には保証出来ない問題がある．そのため，時間制約のある応用分野，例えば産業用機器やロボットなどのリアルタイム制御などの分野への適用は困難であった．これらの分野では，各用途に特化した特殊なネットワーク規格（例えば，自動車分野におけるCAN, LIN, FlexRayなど）が用いられているが，リアルタイム性以外のコストや通信速度の面ではEthernetが優位である．そのため，リアルタイム性が要求される分野においても安価で高性能なEthernetを利用するために，Ethernetのリアルタイム化に関する様々な研究[1]～[4]が行われている．

本稿では，Ethernetのリアルタイム性やパケットロス率の改善を実現するために，戸田らによって行われたシミュレーション評価[1][2]に引き続き，Ethernetへの優先度制御機構，フロー制御機構の導入を検討する．また，提案したプロトコルをLinux 2.4カーネル上で実装し，実機によるネットワーク上で性能評価を行う．

2. Linux 2.4カーネルの動作

最初に，既存のEthernetやOSのプロトコルスタックの振る舞いを確認するために，次のような予備試験を行った．

2台のLinuxホストのNIC間をクロスケーブルで直結し，図1に示す通信試験プログラムを実行した．試験は，ペイロード長1200オクテットのUDPパケット

を連続的に送受信するだけの単純なものである．しかし，10万パケットの送信に対して約7万5000ものパケットロスが発生した．

```
int socket;
char buf[1200];
/* 送信側 */
for (i=0; i<N; i++) {
        send(socket, buf, sizeof(buf), …);
}

/* 受信側 */
for(;;) {
        recv(socket, buf, sizeof(buf), …);
}
```

図 1 UDP パケットを送受信するテストコード（抜粋）

Ethernet, IP, UDP のいずれもパケットの到達性保証は無いため，パケットロス発生は当然とも考えられる．しかし，カーネルの挙動を詳しく調べると，その大部分は Ethernet 自体ではなく，

- 送信側ホストの送信キュー（キューがフルの場合，ユーザプロセスに通知/ブロックせず，単にパケットを破棄する）
- 受信側ホストのソケット受信キュー（recv システムコールによる読み出しが間に合わない場合，パケット破棄が発生する）

などカーネル内部の処理で発生しており，改善の余地があると考えられる．

3. 優先度付きフロー制御機構

今日のネットワークでは，輻輳発生時にはパケット破棄で対応し，到達性保証等は上位層プロトコル(TCPなど)に任せる，という考え方が主流である．一方，LANなどの小規模，高速ネットワークでは，パケットを破棄せず hop-by-hop でのフロー制御を行った方が効率的であるとの報告[3]もある．

そこで本研究では，通信遅延の低減とパケットロスの防止を両立を目指し，図2のような優先度付きフロー制御機構を検討した．

- 前提条件として，CSMA/CD によるパケット衝突を避けるため，各ノード間の結線は Ethernet スイッチと全二重接続を用いて行うものとする．
- パケットに優先度情報を付与し，パケット送信時やスイッチによる中継では，優先度の高いパケットから先に処理を行う．
- 送信側，受信側の NIC にそれぞれ処理可能なパケット優先度のしきい値（以下，送信レベル，受信レベルと呼ぶ)を設ける．
- 受信側でバッファが溢れるなどの理由で受信処理が困難になった場合，NIC の受信レベルを上げて低優先度パケットの受信を拒否する．それと同時に，対向する送信側 NIC に受信レベル変化を通知するためにフロー制御パケットを送信する．送信側 NIC では，受信された制御パケットに従い送信レベルを変化させる．

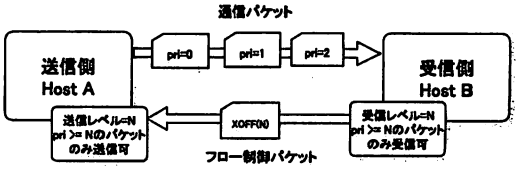

図 2 優先度付きフロー制御プロトコル

Ethernet における既存のフロー制御規格としては，IEEE802.3x で規定された PAUSE フレームがある．これは全二重通信においてNICやスイッチの受信バッファ溢れを防止するために導入されたもので，受信側から送信側に対して指定した期間だけパケット送信の停止を要求する．現行 Ethernet 製品の大半が IEEE802.3x フロー制御に対応しているが，データリンク層でのフロー制御は逆に TCP 等の輻輳制御を阻害するという理由で，あまり積極的には利用されていない[3].

今回実装したフロー制御プロトコルでは，フロー制御パケットを IEEE802.3x 制御フレームの拡張として設計している。

Ethernet パケットに付与する優先度の表現方法は，送受信側で整合が取れていればどの様な方法を用いてもよい．戸田らは Ethernet フレームタイプの未使用部分に優先度情報を埋め込む方法[1][2]を提案した．また，IEEE802.1Q 規格では VLAN タグ内に 3 ビットの優先度フィールドを用意しており，一部のスイッチ製品ではこれを用いた優先度制御に対応しているものも存在する．

本研究では，既存 Ethernet との互換性を考慮し，上位層プロトコルとして IP, UDP を用いる前提とした上で IP ヘッダの TOS(Type Of Service)フィールドの値を使用した．TOS には遅延最小，コスト最小などの特性を表すビットがあり，これらの組み合わせを 0〜15 の 16 レベルの優先度値に写像して解釈する．また，フロー制御パケット自身と ARP パケットに関しては最優先での処理が必要であるため，最高優先度(15)を固定的に割り当てた．

4. Linux カーネル上での実装

3 章で述べた内容に基づき，Linux カーネルのネットワークスタックに改造を加えて優先度制御，フロー制御機構の実装を行った．実装に使用した Linux カーネルのバージョンは，kernel-2.4.27 である．以下，送信処理部，受信処理部，Ethernet ブリッジの順に，実装の概要を説明する．

4.1. パケット送信部

パケット送信部に関しては，主に，

- デバイスドライバ送信キューの優先度化
- 送信レベル未満の優先度のパケットの送信停止
- 送信キュー溢れ時における送信プロセスのブロック

などの機能拡張を行った．

Linux のネットワークデバイスでは，送信キューは Qdisc(Queuing discipline)としてモジュール化されており，ポリシの異なるキューを動的に差し替える事が可能である．標準的な Qdisc モジュールである pfifo_fast でも 3 レベルの優先度キューが用いられているが，今回は専用 Qdisc モジュールを新規に作成した．開発したモジュールは 16 レベルの優先度キューを持っており，キューが溢れた時のパケット破棄を低優先度から行うなどの特徴を持つ．

標準の Linux2.4 カーネルでは，パケット送信は以下の手順で処理される．

(1) ユーザプロセスが send 等のシステムコールを発行する．
(2) プロセスコンテキストで UDP, IP 層の処理を行い，Qdisc に Ethernet パケットを投入する(dev_queue_xmit()関数)．Qdisc の空きがない場合は，送信パケットはそのまま破棄される。
(3) NIC が送信可能状態ならば，プロセスコンテキストのまま Qdisc 内のパケットの送信処理(qdisc_run()関数)を試みる．
(4) NIC が送信不可状態ならば，後でソフト割り込みハンドラから qdisc_run()を実行するようにスケジュールし，一旦終了する．

拡張したカーネルでは，送信フロー制御を実装するために以下の点について変更を行った．

- パケット優先度がデバイスの送信レベルより低い場合，qdisc_run()は Qdisc から NIC へのパケット転送を行わない．
- Qdisc に空きがない場合，dev_queue_xmit()はプロセスをウェイトキューに繋ぎスリープさせる．
- Qdisc に空きが出来た時，ウェイトキューにスリープ中のプロセスがあれば起床させる．

これにより，送信キューに空きがない場合はパケットを破棄せず，send がブロックして待機するようになる．Linux カーネルではデバイスドライバまでの処理を原則プロセスコンテキストのままで実行するため，このような実装が比較的容易である．

送信時のブロック動作は選択式であり，ソケットに対して setsockopt システムコールで明示的に指定した場合のみ有効となる．

4.2. パケット受信部

標準 Linux カーネルでは，パケット受信処理は次の手順で行われる．

(1) NIC の受信割り込みハンドラでは，NIC からのパケット読み出しとカーネル受信キューへの投入のみを行い，そのまま終了する．
(2) その後直ちにソフトウェア割り込みハンドラが起動される．ハンドラでは，カーネル受信キューからのパケットの取り出し， IP, UDP 層の処理を行い，UDP パケットをソケット受信キューに投入して終了する．
(3) ユーザプロセスが recv システムコールを発行し，ソケット受信キューからパケットを読み出す．

受信部に関しては，送信部と同様に受信キューの優先度化を行った後，受信レベルの制御手法を検討した．

当初は受信キュー長に応じて受信レベルを変化させる方法を検討したが，この方法ではうまく動作しなかった．これは，受信キューのパケットはソフトウェア割り込みハンドラによって直ちに取り出され上位層へ送られるため，受信キューにはパケットがほとんど滞留しないためである。

そこで，NIC ドライバ側の受信キューではなく，受信パケットが蓄積されるソケット受信キューの長さをトリガとして受信レベル制御を行う方針で実装を行った．

フロー制御を利用する場合，ユーザが setsockopt システムコールにより受信ソケット優先度を指定する．優先度は任意の値を指定できるが，通常は受信するパケットと同一優先度を指定する．キュー長がしきい値を超えると，ネットワークデバイスに対して受信レベルを(自分自身のソケット優先度+1)にセットする様に要求し，パケット流入をブロックする．複数のソケットから受信レベルセットの要求を行った場合は，最も高い優先度が用いられる。

4.3. Ethernet ブリッジ

3 台以上のノードでネットワークを構成するには Ethernet スイッチが必要であるが，提案手法を実装したスイッチはまだ存在しないため，NIC を複数搭載し

た Linux PC を用いて Ethernet スイッチ(ブリッジ)の機能を実現した.

Ethernet ブリッジの基本機能は, Linux カーネルに含まれるブリッジ実装のコード(net/bridge/*)を使用し, ブリッジ特有のフロー制御処理等を追加した.

ブリッジの処理が送受信端のノードと大きく異なる点は, パケットの送信処理がプロセスコンテキストからではなく, 受信パケットを処理するソフト割り込みハンドラから行われる事である. そのため, 送信時に Qdisc に空きがない場合でも待ち状態に入る事が出来ない. そこで, 待ち状態に入る代わりにパケットを入力元デバイスの受信キューに書き戻す事とした. この場合, パケットは受信キュー内に蓄積される事になるため, 当初の方針通り受信キュー長をトリガとして受信レベルの制御を行う事が可能となった.

5. 実験

5.1. 実験環境

実験環境として, 4 台の PC-AT 機から構成されるネットワーク (図 3) を使用した.

Host S は Ethernet ブリッジであり, 他の 3 台のホストとスター型トポロジで接続している. 接続はクロスケーブルを用いた直接接続であり, 中間にハブ・スイッチ等は使用していない. Host A,B は送信ホスト, Host S は受信ホストであり, Host A,B で生成したテストパケットをブリッジ経由で Host S へ送信する.

各ホストの NIC は, Host S が 3COM 社製 3C59x, 他は Realtek 社製 RTL8139D である. 各 NIC の設定は 100Base-TX Full-Duplex, IEEE802.3x フロー制御有効とし, Qdisc の送信キュー長は 150 とした.

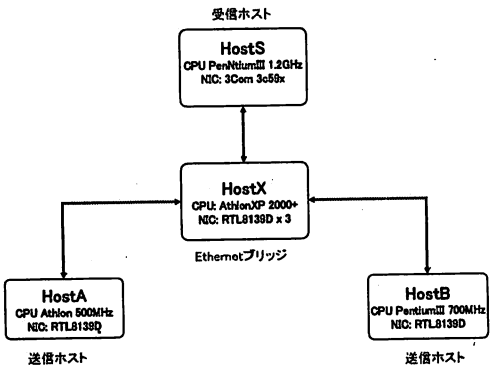

図 3 実験用ネットワークの構成

5.2. フロー制御の応答性

最初に, 対向する 2 ホスト間において受信側 NIC が XOFF 要求を出してから実際にパケット送信が停止するまでの遅れ時間の評価を行った.

Host X から Host S に対して UDP テストパケット(ペイロード長 1200 オクテット)をフルスピードで送信し, Host S 側で tcpdump コマンドを用いてパケットキャプチャを行い, 通信パケットおよびフロー制御パケットのタイミングを観察した. 実際に得られたキャプチャ結果の一例を図 4 に示す. この例では, 受信側から XOFF 要求パケットが出てからパケット流入が停止するまでに, 5 パケット分のオーバランが発生しており, 送信パケット停止までの時間は約 446 μ s である. 他の場合でも XOFF パケット送信後概ね 4〜5 パケットでパケット流入が停止しており, ソフトウェア実装によるフロー制御でも十分実用となる事が確認された.

RTL8139 のハードウェア送信キュー長は 4 パケット分であるため, オーバランしたパケットは主に XOFF 受信時点に既にキューに投入されていたパケットであると考えられる. より長い送信キューを持つ NIC の場合は送信オーバランが大きくなる可能性も考えられるため, キュー長を短く設定する, もしくは XOFF 時に新規送信パケットの投入をやめるだけではなくデバイスの送信動作自体を強制的に停止させるなどの対策が必要となる.

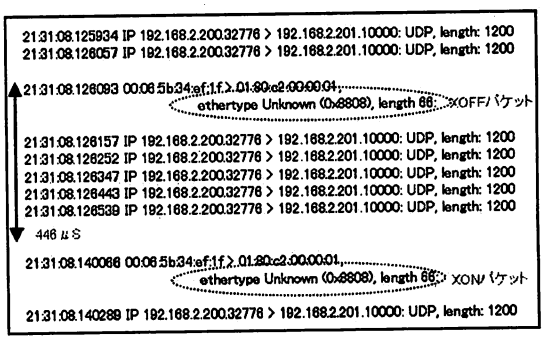

図 4 フロー制御パケットのキャプチャ結果

5.3. パケットロス率の評価

シーケンス番号を埋め込んだ UDP テストパケット (ペイロード長 1200 オクテット) を Host A から Host X を経由して Host S へ向けてフルスピード送信し, 経路中でのパケットロスの発生頻度を求めた. その結果, 100 万パケットの送信に対して 100〜1000 パケット程度のロスが観測された. ただし, カーネル内で明示的にパケット破棄を行っている箇所でのロスは 0 であり, 全てのパケットロスは Host X もしくは Host S の受信側 NIC におけるデバイスレベルでのパケット取りこぼしであった. また, 一回のパケットロスは 30〜100 パケット程度(NIC の種類によって異なる)の長さでバースト状のパターンで発生していた.

これらの状況から, パケットロスの原因は, NIC 受

信割り込みの長期間ブロックによりNIC受信キューからのパケット読み出し処理が遅れ，キューがオーバフローしたためと推測される．

受信割り込みハンドラが起動される時点で，既にNIC側でパケットロスが発生しているため，この問題はソフトフロー制御だけでは解決出来ない．抜本的な解決のためには，カーネル内部の長い割り込み禁止区間を除去し，受信割り込みが長期間（少なくとも，NIC受信キューがフルにならない時間）ブロックされない事を保証する必要がある．

もしくは，IEEE802.3xによるハードフロー制御が完全に機能していれば，デバイスレベルでのキューのオーバフローを防止出来るとも考えられる．ただし，その場合でもカーネル内の受信キューオーバフローによるパケット破棄を防止する手段として，ソフトフロー制御は必要である．

5.4. 優先度と通信遅延

パケットの優先度制御による通信遅延の変化の評価を行った．

負荷トラフィックとして，Host AからHost Sに対してUDPテストパケット（優先度1）をフルスピード送信した．その状態で，Host A，BからHost Sに対してICMP ECHO要求パケットを送信し，そのラウンドトリップタイムを計測した．図5は，ICMPパケットの優先度を負荷トラフィックよりも高い優先度4に設定した場合の結果である．無負荷時（グラフ前半）のRTT0.4～0.6msに対し，負荷トラフィックがある状態（グラフ後半）のRTTは平均0.8～1.0ms程度であり最悪値でも1.2ms以下に抑えられている．

一方，図6は，ICMPパケットの優先度が負荷トラフィックと等しい場合，すなわち優先度制御を行わない場合の結果である．負荷を加えた場合のRTTは20ms程度まで悪化しており，かつ突発的に100msから最大500ms以上の遅延が生じている。

以上の結果から，パケット優先度を適切に設定する事により，帯域飽和状態でも高優先度パケットの通信遅延を一定時間以下に抑えられる事がわかった．

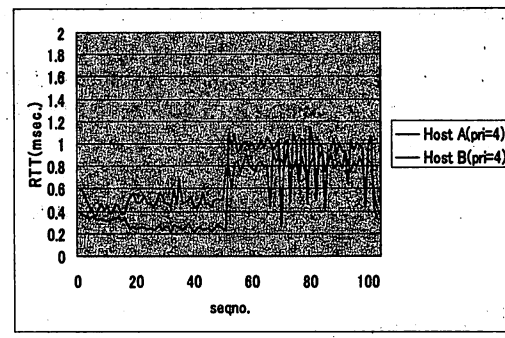

図 5 優先度制御を行った場合のRTT

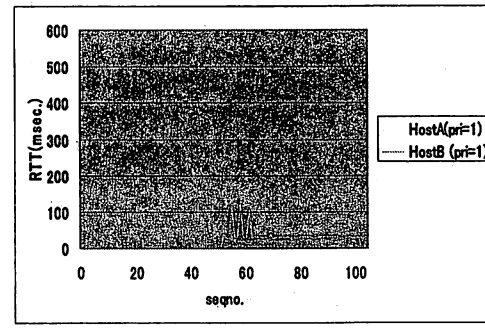

図 6 優先度制御を行わない場合のRTT

5.5. 優先度による帯域制御

最後に，優先度指定による帯域割り当ての制御性に関する評価を行った．

Host AとHost Bからそれぞれ異なる優先度のUDPテストパケット(1200オクテット)をHost Sに対して送信し，Host Sの受信側における通信レートを計測した．

図7は，Host Aからのみ送信を行った場合のグラフである．送信レートが階段状に低下しているのは，送信プロセス側でウェイトを挿入し送信レートを低下させているためである．

図8は，Host A，Host Bの両方から送信を行った場合の結果である．Host Aのパケットは優先度4，Host Bは優先度1である．また，Host Aの送信パターンは図7の場合と同様に送信レートを制御し，Host Bは常にフルスピード送信とした．

Host Aがフルスピード送信している区間（グラフ左端）では，Host B側の送信はほぼ停止状態となっているが，Host A側の送信レートが低下し帯域幅に空きが生じるにつれて，Host B側の帯域が増加している．Host A側の送信レートは優先度の低いHost Bのトラフィッ

クには影響されておらず，Host A 単独の場合と同一の挙動を示した．

Host A，B の送信レートの和は常に 95MBbps 程度に保たれている。これは 100Base-TX の帯域幅をほぼ使い切る値であり，フロー制御に伴うオーバヘッド等は特に生じていないといえる（後半で全体の送信レートが低下しているが，これは Host B のパケット送出能力が不足しているのが原因である）．

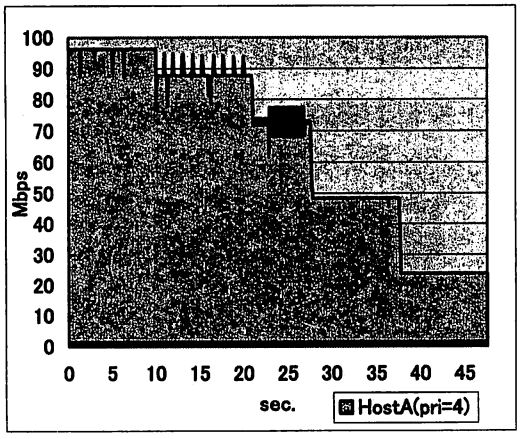

図 7 Host A のみの場合の送信レート

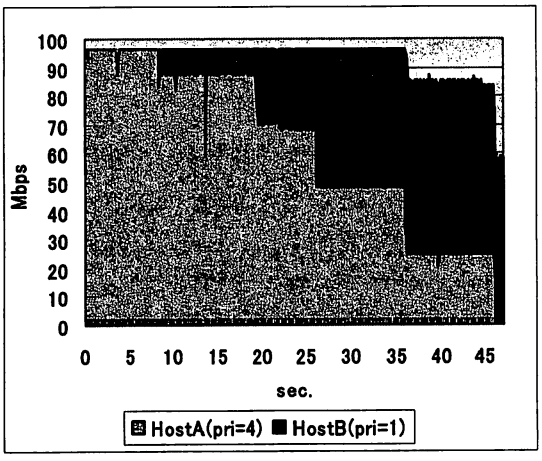

図 8 Host A, B 両方の場合の送信レート

6. まとめ

Linux 2.4 カーネルのネットワークプロトコルスタックをベースとして，Ethernet パケットの優先度制御，フロー制御機構を実装した．また，実機による評価試験を行った結果，パケットロス率や通信遅延の改善，および優先度による良好な帯域制御性を有する事が確認された．

今後の課題としては，受信パケット取りこぼしを防ぎ完全なロスレス通信の実現する事が第一に挙げられる．

また，今回の実装ではベースとして使用している Linux カーネルがリアルタイム OS ではないため，通信に関連するプロセスまで含めたリアルタイム化は難しい．今後は，ART-Linux や μITRON 上のネットワーク環境などでも同様の実装を試み，システム全体としてのリアルタイム化の検討を進めていきたい．

文　献

[1] 戸田賢二，関山守，Tsun-Ho Liu，"リアルタイムイーサネットとその予備的評価，"実時間処理に関するワークショップ RTP2003，信学技報 Vol.102 No.701 pp.25-28，Mar.2003.

[2] 戸田賢二，関山守，Tsun-Ho Liu，"リアルタイムイーサネットのシミュレーションによる評価，"SWoPP2003，pp.27-32，Mar.2003.

[3] Mark Karol, S.Jamaloddin Golestani, David Lee, "Prevention of Deadlocks and Livelocks in Lossless Backpressured Packet Networks," IEEE/ACM Transactions on Networking, Vol.11, No.6, pp.923-934, Dec. 2003.

[4] Jan Kiszka, Bernardo Wagner, Yuchen Zhang, Jan Broenink, "RTnet – A Flexible Hard Real-Time Networking Framework," 10th IEEE International Conference on Emerging Technologies and Factory Automation, Catania, Italy, Sep.2005.

[5] 石綿陽一,松井俊宏,"汎用 OS とデバイスドライバを共有できる実時間オペレーティングシステム，"信学技法 CPSY97-119, pp.41-48, 1998.

[6] 阿部 司，吉村斎,久保洋,"組込みシステム用 TCP/IP プロトコルスタックの実装と評価，" 情報処理学会論文誌，Vol.44, No.6, pp.1583-1592, Jun. 2003.

[7] 石田修，瀬戸康一郎，"スイッチングで使用されるプロトコル，"10 ギガビット Ethernet 教科書，pp.85-100，インプレス，April 2005.