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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4 (14)
2007／3／16

ARM7TDMIボードへのLinux Kernel 2.6.14の移植と性能評価

金子 雄† 石山 政浩†

† 株式会社 東芝 研究開発センター 通信プラットホームラボラトリー
212-8582, 川崎市幸区小向東芝町1
E-mail: †{yu1.kaneko,masahiro.ishiyama}@toshiba.co.jp

あらまし 近年，ビル内の制御ネットワークにおいてIPノードの低コスト化が求められている．また，ノード数の増加や外部ネットワークとの接続性，通信の安全性を考慮すると，IPノードはIPv6やIPsecの機能を備えるべきである．以上の要件をふまえ，我々はARM7TDMIをCPUコアとする計算機へLinuxカーネル2.6を移植し，性能を評価した．評価の結果，ARM7TDMIは制御ネットワークにおけるIPノードに要求される性能を満たすことがわかった．
キーワード Linux, ARM7TDMI, 性能評価

Porting and Performance Evaluation of Linux Kernel 2.6.14 to ARM7TDMI

Yu KANEKO† and Masahiro ISHIYAMA†

† Communication Platform Laboratory, Corporate Research & Development Center, TOSHIBA Corp.
1, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki, 212-8582
E-mail: †{yu1.kaneko,masahiro.ishiyama}@toshiba.co.jp

Abstract The high cost of IP nodes is a problem for popularizing IP-based building automation. IPv6 and IPsec, meanwhile, are necessary for IP nodes to guarantee network security, network band and system scalability. In this papar, we port Linux kernel 2.6 to ARM7TDMI and evaluate the performance. As a result, the performance of ARM7TDMI meets the requirements of an IP node in building automation.
Key words Linux, ARM7TDMI, performance evaluation

1. はじめに

近年，ビル内の照明や空調などの設備機器をネットワークを介して制御することにより，ビルを高機能化したいという要求が増加している．例えば，空調と温度センサをネットワークを介して相互接続することで，空調とセンサの連係動作が容易になり，センサの温度に応じて適切に空調を制御することで，ビルの低消費電力化が実現できる．以後，このような設備機器を制御するネットワークのことを制御ネットワークと呼ぶ．

従来の制御ネットワークでは空調設備や防災設備などのサブシステムごとに専用のネットワークを構築し，プロトコル変換を行うゲートウェイを介して各専用ネットワークを接続している．このような構成では，ネットワークごとに特定のファシリティベンダの製品に依存した通信プロトコルを扱う必要があるため，運用管理が複雑となる．また，ゲートウェイが必要となるため，制御ネットワーク構築のためのコストが増加する．

これらの問題を解決するために，制御ネットワークにおける通信プロトコルとして，インターネットでの運用実績のあるオープンなプロトコルであるIP (Internet Protocol) の利用が求められている．制御ネットワークにおける全ての通信をIPを用いて行うことができれば，ゲートウェイは不要となり，コストを削減できる．また，単一のベンダへの依存性が弱まるため，運用管理が容易になり，ベンダ間での競争によるコストダウンが期待できる．

しかし現状では，コストの問題により制御ネットワークの全てをIP化することは困難である．さいたま新都心ビルの事例[10]では，監視制御端末とゲートウェイまでをIPノード化しており，フィールド側の機器はIPノード化していない．ここでIPノードとはIP機能を備えた端末のことである．ゲートウェイと監視制御端末はIPによる通信を行い，ゲートウェイとフィールド側の機器はLonWorks(注1)やBACnet(注2)などの制御用プロトコルによる通信を行う．制御ネットワークをIP化することを考えた場合，現状ではこのようなネットワーク構成

(注1)：LonWorks is a registered trademark of Echelon Corporation.
(注2)：BACnet is a registered trademark of ASHRAE.

表 1 Armadillo-J Version 10A の仕様

プロセッサ	NS7520
CPU コア	ARM7TDMI
クロック	55MHz
SDRAM	8MByte
FLASH	2MByte
ネットワーク機能	100BASE-TX 対応
基盤サイズ	50 x 37.5[mm]

が一般的である.

今後，制御ネットワークにはセンサをはじめとする多種多様なフィールド側の機器が接続されると考えられる．現状の制御ネットワークの構成では，新たな機器を追加する度に，その機器に対応するゲートウェイを用意することになり，コスト面で不利，運用面で不便である．このような事態を回避するためには，フィールド側の機器を IP ノード化し，全ての通信を IP という共通基盤上で行えるようにする必要がある。一方でフィールド側の機器の数は多く，事例 [10] ではゲートウェイの数が約 210 個なのに対し，監視制御下にあるフィールド側の機器数は約 3540 点である．したがって，フィールド側の機器の IP ノード化のためには，IP ノードの低コスト化が必要となる.

本報告では，低コストかつ低性能な計算機上に Linux カーネル 2.6.x を移植し，その性能を評価する．低性能な計算機として，ARM7TDMI を CPU コアとするボードである Armadillo-J [3] を使用する．Linux は既存の多くのネットワークアプリケーションを利用でき，さらにカーネル 2.6.x は IPv6 (Internet Protocol version 6) や IPsec (IP Security Protocol) などの通信プロトコルを備えているため，IP ノードに求められる機能を満たしていると言える．移植したカーネルの性能評価を行うことで，IP ノードに求められる計算機性能の指標を得ることができる.

2. Armadillo-J と uClinux

本節では，Armadillo-J と uClinux [8] について説明する．Armadillo-J は Atmark Techno 社 [4] の製品であり，ARM7TDMI をプロセッサコアとする小型ボードである．本報告で使用した Armadillo-J Version 10A の仕様は表 1 のとおりである．Armadillo-J には標準で Linux カーネル 2.4.22 をベースとする uClinux が搭載されている．uClinux とは，Linux のメインラインカーネルに，MMU (Memory Management Unit) が無い低性能な計算機上で動作するための修正を加えたカーネルのことである．そのため，MMU を持たない ARM7TDMI 上でも動作する．uClinux の開発成果は徐々にメインラインカーネルに取り込まれており，バージョン 2.6.10 からは MMU を持たない ARM 用のコードが取り込まれている.

3. Linux カーネル 2.6.14 の移植

移植する Linux カーネルのバージョンは 2.6.14 とした．Linux カーネルをバージョン 2.4.x からバージョン 2.6.x に移行することで，IPsec や IPv6，カーネルレベル・プリエンプションなどの，制御ネットワークにおいても有用な機能を利用できるようになる．上記機能は，Linux カーネル 2.4.x でも，一部のバージョンにパッチを適用することで実現できる．しかし，Linux カーネル 2.6.x では上記機能は標準実装されており，現在も開発が続けられている．したがって，Linux カーネル 2.6.x に移行することは有意義であると言える．以降，単にカーネルと記述する場合は Linux カーネルのことを指すものとする.

カーネル 2.6.14 の移植においては，Armadillo-J に標準搭載されているシリアルコンソールやイーサネットのデバイスドライバのカーネル 2.6 対応化，割り込みハンドラの設定方法の修正，MMU に依存した処理の修正などを行った.

4. 性能評価

本節では，移植したカーネル 2.6.14 の性能評価について述べる．評価した項目は，リアルタイム性能，ネットワーク性能，プロセス数に対する拡張性，システムコールの処理時間の 4 つである．リアルタイム性能，ネットワーク性能，プロセス数に対する拡張性の 3 つの性能は，制御ネットワークにおける IP ノードにとって重要な性能である．システムコールの処理時間の評価は，ネットワーク性能とプロセス数に対する拡張性の評価結果を考察するために行った.

以下，まずシステムコールの処理時間の評価結果について述べ，その後，それに基づいてネットワーク性能とプロセス数に対する拡張性の評価結果を考察し，最後にリアルタイム性能について述べる.

4.1 システムコールの処理時間

システムコールの処理時間の評価では，東芝製の PC である PORTEGE 2010 (CPU:Pentium III, クロック数 866MHz, メモリ 512MByte) でも評価を行った．これは，ARM7TDMI における評価結果が，ARM7TDMI に依存した評価結果ではないことを調べるためである．以降，Pentium III のことを i386 と表記する.

本評価では lmbench [6] を使用した．lmbench とは，様々なカーネルの内部処理に要する時間を計測するテストプログラム群である．評価に使用したプログラムを以下にまとめる.

- lat_ctx : プロセスの切り替えの処理時間を計測する．主にスケジューリングとコンテキストスイッチの処理が含まれる.
- lat_pipe : pipe を用いたプロセス間通信の処理時間を計測する.
- lat_fifo : FIFO (First In First Out) 特殊ファイルを用いたプロセス間通信の処理時間を計測する.
- lat_fcntl : 2 つのプロセスが 1 つのファイルのロックとアンロックを交互に行う際の処理時間を計測する.
- lat_syscall : getppid, write, read, open などのシステムコールの処理時間を計測する.
- lat_sig : sigaction の処理時間を計測する.

評価結果を表 2 に記す．ARM7TDMI でも i386 でも，多くのテストにおいてカーネル 2.6.14 のほうが処理時間が長くなっている．lat_ctx の結果から，プロセス数が 20 程度の場合には，カーネル 2.6.14 はカーネル 2.4.22 よりもプロセス切り替えの

表 2　ARM7TDMI と i386 における lmbench の評価結果．各欄の値は，10 回の試行結果の平均値であり，単位はマイクロ秒である．A は ARM7TDMI，i は i386 を表す．

プログラム	2.4.22/A	2.6.14/A	2.4.22/i	2.6.14/i
lat_ctx (プロセス数 20)	95.9	175.78	1.74	2.41
lat_pipe	447.444	778.873	5.207	9.836
lat_fifo	484.856	791.0488	5.296	9.886
lat_fcntl	1527.204	1282.238	8.707	12.718
lat_syscall getppid	14.173	15.745	0.402	0.214
lat_syscall write	35.728	59.235	0.553	0.733
lat_syscall read	37.978	65.411	0.611	0.943
lat_syscall open	306.036	494.108	3.647	8.128
lat_sig sigaction	97.547	103.173	1.075	1.172

処理時間が長いことがわかる．これは，lat_pipe や lat_fifo などのプロセス切り替えを行うテストにおいて，カーネル 2.6.14 の処理時間が長くなっている理由の 1 つである．read や open などのテストはプロセス切り替えを行わないが，カーネル 2.6.14 のほうが処理時間が長くなっている．このことから，カーネル 2.6.14 ではファイル操作やディスクアクセスの処理時間が長くなっていることがわかる．getppid と sigaction はプロセス切り替えやファイル操作を行わないが，カーネル 2.6.14 のほうが処理時間がわずかに長い．これは，SMP (Symmetric Multi Processing) 対応のために，排他制御が細かく行われるようになったからだと考えられる．ただし，i386 ではカーネル 2.6.14 のほうが getppid の処理時間が短くなっており，一部の処理に要する時間はプロセッサに依存している可能性がある．

4.2　プロセス数に対する拡張性

制御ネットワークにおいてフィールド側の機器が IP ノード化され，柔軟な連携が可能になると，IP ノードは同時に複数の制御アプリケーションを実行し，複数の相手と通信を行うようになると考えられる．したがって，プロセス数に対する拡張性は重要となる．本評価では，hackbench [5] というプログラムを使用した．hackbench ではグループという単位ごとに送信プロセスと受信プロセスを 20 ずつ生成する．そして，全送信プロセスが同一グループの全受信プロセスに，pipe を使用してメッセージを送信する処理に要する時間を計測する．このため，hackbench では pipe 処理とプロセス切り替え処理が多く発生する．Armadillo-J はメモリサイズが 8MByte と小さいため，グループごとの送受信プロセスの生成数をそれぞれ 10 ずつとして評価した．以下，hackbench の評価結果について述べる．

4.2.1　メッセージ送信の処理時間

図 1 は Armadillo-J における hackbench の評価結果である．各試行を 10 回ずつ行い，結果をすべてプロットした．処理時間はカーネル 2.6.14 のほうが長かった．表 2 から，カーネル 2.6.14 のほうが pipe やプロセス切り替えの処理時間が長いことがわかっており，それが hackbench の処理時間にも影響している．またグループ数を 3 にした場合，カーネル 2.4.22 ではメモリが不足し，メモリ確保に失敗して評価が行えなくなったが，カーネル 2.6.14 では評価を行うことができた．このことから

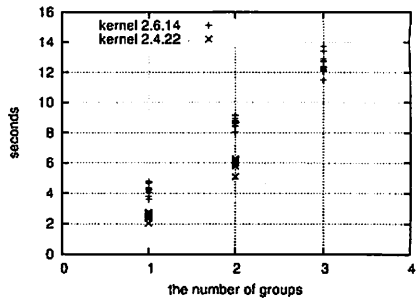

図 1　Armadillo-J における hackbench の処理時間．1 グループあたりのプロセス数は 20.

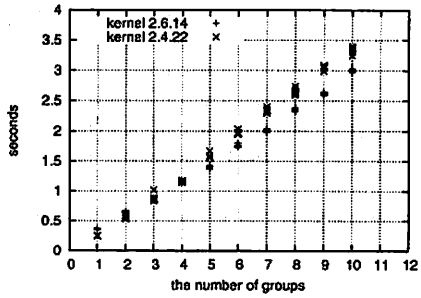

図 2　PORTEGE における hackbench の処理時間．1 グループあたりのプロセス数は 40.

らカーネル 2.6.14 はカーネル 2.4.22 よりも，プロセス数に対するメモリ使用効率が優れていることがわかる．メモリ使用効率に関しては 4.2.2 節で解説する．

次に，PORTEGE 2010 でも同様の評価を行った．PORTEGE は 512MByte のメモリを持つため，Armadillo-J よりもグループ数を増やして評価することができた．図 2 は，PORTEGE における評価結果である．グループ数が 4 以下の場合には，Armadillo-J での評価結果と同様にカーネル 2.6.14 のほうが処理時間が長い．しかしグループ数が 5 以上になると，カーネル 2.6.14 のほうが処理時間が短くなる．これは，カーネル 2.6.x ではスケジューラのアルゴリズムが改良されたためである．カーネル 2.4.x では 1 つのランキューに全ての実行可能プロセスを登録しているため，スケジューリングの際にはランキュー全体を走査する必要があるが，カーネル 2.6.x では優先度毎にランキューを用意しているため，スケジューリングの際にはランキューの先頭のプロセスを選択するだけでよい．図 2 は，プロセス数が増加した場合には，pipe やコンテキストスイッチの処理時間よりも，スケジューリングの処理時間の影響が大きくなることを表している．

4.2.2　メモリの使用状況

4.2 の評価において，カーネル 2.6.14 とカーネル 2.4.22 では，生成できるプロセス数が異なった．そこで，Armadillo-J において hackbench の評価中に/proc/meminfo と/proc/slabinfo を読み出し，メモリの使用状況を調べた．調査の手順は図 3 の通

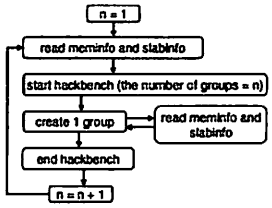

図 3　メモリの使用状況の調査手順．メモリ不足により評価が行えなくなった時点で調査終了とした．

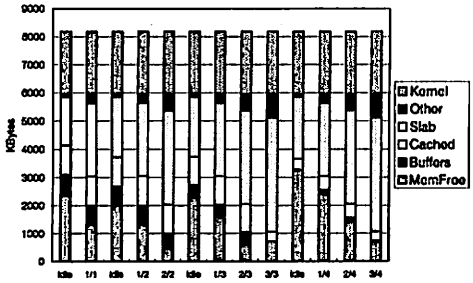

図 4　hackbench による評価中のメモリ使用状況 (2.6.14)

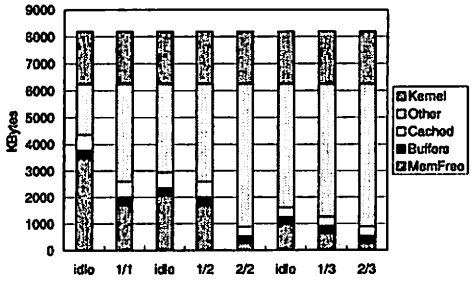

図 5　hackbench による評価中のメモリ使用状況 (2.4.22)

りである．

図 4，図 5 は meminfo を読み出して得たメモリ使用状況を表している．x 軸は図 3 で示した評価手順を時系列で表しており，idle はアイドル時に読んだ値を，分数は hackbench 実行中に読んだ値を表している．分数の分母は hackbench が作成する総グループ数であり，分子はすでに作成したグループ数を意味する．MemFree，Buffers，Cached，Slab の 4 系列は meminfo で定義されている項目であり，それぞれ，使用可能なメモリサイズ，ディスクブロック用のバッファサイズ，ページキャッシュのサイズ，スラブキャッシュのサイズを表す．Kernel と Other は独自に設定した値であり，Kernel はカーネルイメージのサイズ，Other は残りのメモリサイズを表す．カーネル 2.4.22 の meminfo には Slab の情報が存在しないため，meminfo からメモリの使用状況を読み取りにくい．

起動直後と 1 グループ作成時の MemFree を比較すると，hackbench と，hackbench が作成する 1 グループ分のプロセスが使用しているメモリサイズがわかる．カーネル 2.6.14 では約

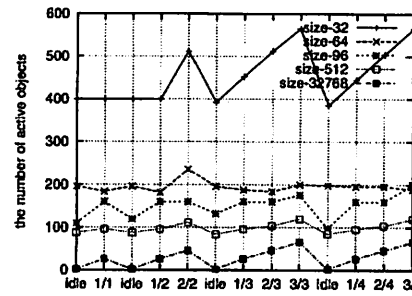

図 6　hackbench による評価中のスラブキャッシュ使用状況 (2.6.14)

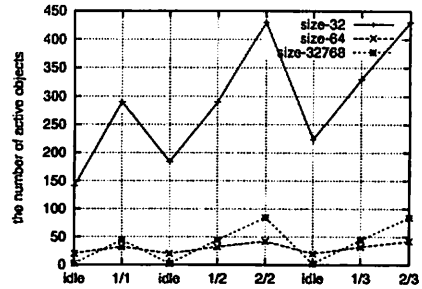

図 7　hackbench による評価中のスラブキャッシュ使用状況 (2.4.22)

1.1MByte，カーネル 2.4.22 では約 1.7MByte のメモリを使用しており，1 プロセスあたり約 32KByte の差が生じている．

図 4，図 5 では，hackbench の実行前よりも実行後のほうが，MemFree の値が小さくなっている場合がある．これはプログラム終了後に，カーネルが，使用したメモリの一部をスラブキャッシュ用のメモリとして保持するためである．カーネル 2.4.22 では，hackbench で作成するグループ数を増やすたびにアイドル時の MemFree が減少しており，meminfo の値からメモリの使用状況を把握することが困難である．これに対してカーネル 2.6.14 では，hackbench で使用したメモリの大部分が，実行後には MemFree へ返されており，使用可能なメモリサイズを把握しやすい．また起動直後において，カーネル 2.6.14 はカーネル 2.4.22 の約 2 倍の領域を Cached と Buffers に割り当てているが，MemFree が少なくなった場合には，それらの領域を使用してプロセスを生成していることがわかる．

図 6 と図 7 は，カーネル 2.6.14 とカーネル 2.4.22 におけるスラブキャッシュの使用状況を表している．x 軸が示す意味は図 4，5 と同様である．各系列は slabinfo で定義されている項目であり，それぞれ特定サイズのメモリを割り当てるために使用されるスラブキャッシュである．例えば size-32768 という名前のスラブキャッシュは，32KByte のメモリを割り当てるためのスラブキャッシュである．カーネル 2.4.22 におけるプロセス生成時の size-32768 の使用量は，カーネル 2.6.14 における使用量の約 2 倍であり，また，その他のスラブキャッシュの使用量に大きな差はなかった．したがって，カーネル 2.6.14 とカーネル 2.4.22 において，1 プロセスに消費するメモリの差が約

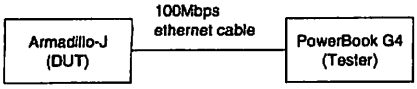

図 8　nuttcp による評価時の環境

表 3　TCP のスループット (単位は Mbit/s)

	2.4.20	2.6.14	2.4.20+IPsec	2.6.14+IPsec
送信	9.091	8.370	0.211	0.210
受信	14.411	13.474	0.255	0.247

32KByte であるのは，スラブキャッシュsize-32768 の使用量による影響が大きいと言える．

図 6, 7 では，hackbench の評価による影響が大きかったスラブキャッシュの値だけをプロットした．プロセスの生成によって影響を受けているスラブキャッシュの数は，カーネル 2.6.14 のほうが多いことがわかる．したがってカーネル 2.6.14 では，より細かくスラブキャッシュの使用を制御していると考えられる．

4.3　ネットワーク性能

IP 通信を行うことが前提である IP ノードにとって，ネットワーク性能は重要である．本評価では nuttcp バージョン 5.3.1 を使用した．評価環境は図 8 の通りである．Armadillo-J を DUT (Device Under Test)，PowerBook G4 をテスターとして，100Mbit Ethernet のケーブルで接続した．評価した項目は，DUT が送信側である場合と受信側である場合の，TCP と UDP のスループットである．いずれの場合においても，IPsec を使用する場合と使用しない場合を計測した．評価の際は，UDP データグラムの送信レート以外は nuttcp のパラメータを変更しなかった．IP はバージョン 4 を使用した．

IPsec では ESP (Encapsulating Security Payload) プロトコルを使用し，AH (Authentication Header) プロトコルは使用しなかった．ESP プロコトルでは 128bit 鍵を使用した aes-cbc による暗号化と，160bit 鍵を使用した HMAC-SHA1 による認証を行った．DUT が送信側である場合は，DUT からテスターへのパケットに ESP を適用し，また DUT が受信側である場合は，テスターから DUT へのパケットに ESP を適用するようなセキュリティポリシーとした．暗号化に用いる鍵は，あらかじめ手動で DUT とテスターに設定した．

ネットワーク性能の評価では，比較するカーネルのバージョンを 2.4.20 とした．これは，カーネル 2.4.x で IPsec 機能を実現するためのパッチが，バージョン 2.4.20 用であったためである．このパッチは，USAGI プロジェクト [9] によって提供されているものである．

4.3.1　TCP の評価

TCP のスループットの計測結果は表 3 のようになった．IPsec を使用しない場合は，送信，受信にかかわらず，カーネル 2.6.14 のほうがスループットが 1Mbps 弱小さかった．表 2 より，カーネル 2.6.14 のほうが write やプロセス切り替えの処理時間が長いことがわかっており，それが評価結果に影響していると考えられる．

文献 [2] では，Pentium III で IPsec を使用した場合のスループットの計測結果について述べているが，その値は IPsec を使用しない場合の約 60%である．それに対して，ARM7TDMI では，IPsec を使用するとスループットが約 2.5%から約 1.7%になっている．したがって，ARM7TDMI にとって IPsec の暗号化・複合化の処理は高負荷であると言える．

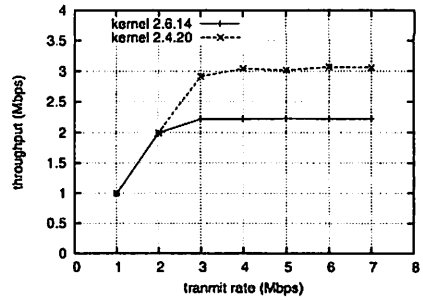

図 9　UDP の送信スループット

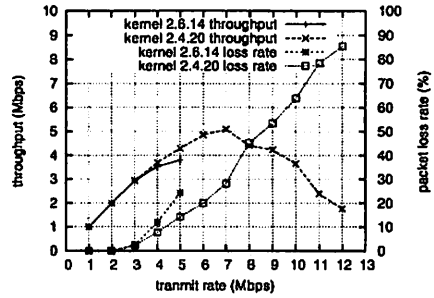

図 10　UDP の受信スループット．右側の y 軸はパケットロス率を表す．

4.3.2　UDP の評価

UDP のスループットは，nuttcp において UDP データグラムの送信レートを変化させて計測した．nuttcp は設定された送信レートで UDP データグラムを送信しようとするため，DUT が受信側である場合には，パケットロスが発生する．そこで，DUT が受信側の場合には式 1 に基づいてパケットロス率を計算した．また，UDP のデータサイズは 1024Byte とした．

$$\text{パケットロス率} = \frac{DUT \text{ が受信したパケット数}}{Tester \text{ が送信したパケット数}} \quad (1)$$

UDP の送信スループットを図 9 に示す．送信レートが 2Mbps 以下の場合は，どちらのカーネルも設定値どおりの送信レートで UDP データグラムを送信できた．しかし，送信レートを 3Mbps 以上に設定しても，カーネル 2.4.20 では約 3Mbps，カーネル 2.6.14 では約 2.2Mbps のスループットしか出なかった．IPsec を使用すると，最大のスループットは約 0.2Mbps となり，TCP と同様に大幅にスループットが低下した．

UDP の受信スループットは図 10 のようになった．図 10 において右側の y 軸はパケットロス率を表している．送信レートが 3Mbps 以下の場合はスループットとパケットロス率に違いは見られなかった．送信レートが 4Mbps 以上になると，カーネル 2.4.20 のほうがカーネル 2.6.14 よりもスループットが大き

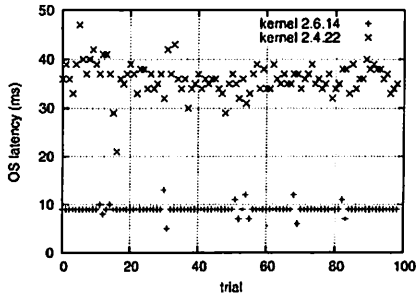

図 11 OS レイテンシの計測結果．x 軸は試行回数を表す．

くなり，また，パケットロス率が小さくなった．送信レートを 6Mbps にすると，カーネル 2.6.14 では計測途中で受信パケット用のソケットバッファの割り当てが行えなくなり，カーネルが停止した．カーネル 2.4.20 は送信レートが 7Mbps のときにスループットが最大となった．送信レートが 8Mbps 以上になると，さらにパケットロス率が増加し，スループットは減少した．

DUT が受信側で IPsec を有効にした場合は，送信レートを 0.3Mbps 以上に設定すると，評価途中で nuttcp が応答しなくなり，どちらのカーネルでも評価が行えなかった．

4.4 リアルタイム性能

制御ネットワークにおいて，制御命令に対する応答のリアルタイム性は重要である．カーネル 2.6.x ではカーネルレベル・プリエンプション機能が導入されており，リアルタイム性能の向上が期待できる．本評価では，OS レイテンシを計測するテストプログラムを作成した．OS レイテンシ [1] とは OS (Operating System) のリアルタイム性を表す指標であり，式 2 で求まる．

$$OSlatency = t' - (t + T) \qquad (2)$$

式 2 において，T はプロセスが休眠する時間，t はプロセスが休眠する直前の時刻，t' はプロセスが起床した直後の時刻である．OS レイテンシが 0 となることが理想的だが，起床時刻とタイマ割り込み時刻との間にずれが生じたり，カーネルが non-preemptible section を実行していたりするため，OS レイテンシは 0 にはならない．

評価プログラムでは次の動作を 100 回繰り返し，OS レイテンシを計測した．

（1） gettimeofday を実行して時刻 t を取得する．

（2） sleep によって 1 秒間休眠する．

（3） 休眠後に gettimeofday を実行して時刻 t' を取得する．

また，non-preemptible な処理が発生しやすいように，評価プログラムの実行中には，バックグラウンドにて nuttcp による TCP 通信を実行した．

図 11 と表 4 が OS レイテンシの計測結果である．x 軸は試行回数を表しており，100 回の試行結果をすべてプロットした．図 11 から，カーネル 2.6.14 のほうがカーネル 2.4.22 よりも OS レイテンシが小さいことがわかる．また表 4 からは，カーネル 2.4.22 の OS レイテンシの最大値，最小値，平均値はカーネル 2.6.14 の約 4 倍であり，標準偏差値も大きいことがわかる．

表 4 OS レイテンシの最大値，最小値，平均値，標準偏差値

	kernel 2.4.22	kernel 2.6.14
minimum (ms)	21	5
maximum (ms)	47	13
average (ms)	36.010	9.020
standard deviation	3.234	0.915

この結果から，カーネルレベル・プリエンプション機能によって，カーネル 2.6.x のリアルタイム性能は向上していると言える．

5. ま と め

我々は ARM7TDMI を CPU コアとする計算機である Armadillo-J へ Linux カーネル 2.6.14 を移植し，ネットワーク性能やリアルタイム性能，プロセス数に対する拡張性，システムコールの処理時間を評価した．ARM7TDMI では，TCP で 10Mbps 以上のスループットを出すことができ，また，60 プロセス以上を生成できた．したがって，ARM7TDMI 程度の性能を備えていれば，制御ネットワークにおける IP ノードとして利用可能であると言える．ただし，ARM7TDMI にとって IPsec の暗号化・復号化の処理が高負荷であることもわかっており，IPsec を使用した通信を行うことを想定するならば，さらに詳細な検討が必要となる．

カーネル 2.4.x との比較によって，カーネル 2.6.14 はカーネル 2.4.22 よりもシステムコールやネットワーク通信の処理時間が長くなっていることがわかった．しかし，カーネル 2.6.14 はプロセス数に対する拡張性においてカーネル 2.4.22 よりも優れており，また，リアルタイム性能が向上していた．さらに IPsec や IPv6 などの機能を利用できるため，カーネル 2.6 への移行は有益だと言える．

今後の課題としては，システムコールやネットワーク通信の処理におけるオーバーヘッド部分を把握することと，そのオーバーヘッドの軽減方法を検討することが挙げられる．

文 献

[1] L. Abeni, A. Goel, C. Krasic, J. Snow and J.Walpole. A Measurement-Based Analysis of the Real-Time Performance of Linux. Proc. of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium, 2002.

[2] A. Ferrante and J. Owen. IPSec Hardware Resource Requirements Evaluation. Next Generation Internet, pp.240–246, 2005.

[3] Armadillo Official Site: Armadillo-J. http://armadillo.atmark-techno.com/armadillo-j/.

[4] Atmark Techno Inc. http://www.atmark-techno.com/.

[5] Linux Process Scheduler Improvements. http://developer.osdl.org/craiger/hackbench/.

[6] lmbench. http://sourceforge.net/projects/lmbench/.

[7] nuttcp. ftp://ftp.lcp.nrl.navy.mil/pub/nuttcp/.

[8] *uClinux*^{TM} – Embedded Linux Microcontroller Project – Home Page. http://www.uclinux.org.

[9] USAGI Project – Linux IPv6 Development Project. http:/www.linux-ipv6.org/.

[10] さいたま新都心ビルの事例. http://www.v6pc.jp/pdf/20051019-NTTFacilities.pdf.