仮想マシンの実行に特化したセンサネットワークOS

田辺 聡[1,a)]　福田 浩章[2]

概要：無線センサネットワーク (WSN) では，プログラムの再配備や保護機能のため，仮想マシンが広く利用されている．仮想マシンは WSN 用 OS で実行されるが，これらの OS は多様なアプリケーションの実行を前提としないため，プログラムの並列実行やプロセス間通信といった機能も仮想マシンが行っている．本来，これら機能は OS が提供すべきであるが，WSN では OS と仮想マシンが個々に研究されてきたため両者の機能分割が明確ではない．そこで本研究では，アプリケーションの実行に仮想マシンを前提とした OS を提案する．この OS では，汎用 OS が提供する保護機能，プロセス間通信，仮想マシンのスケジューリングを実現し，これらをシステムコールとして提供する．

キーワード：センサネットワーク OS，仮想マシン，オーバーレイ・ネットワーク

Sensor Network OS Specialized for Virtual Machine Execution

Tanabe Satoshi[1,a)]　Fukuda Hiroaki[2]

Abstract: In the wireless sensor network (WSN), virtual machines (VMs) are widely used because of reallocation of programs and protection functions. VM is executed by the OS for WSN, but since these OSs do not assume execution of various applications, the VM also performs functions such as parallel execution of programs and communication between processes. Originally, these functions should be provided by the OS, but since OS and VM have been separately studied in WSN, the functional division of both is not clear. Therefore, in this paper, we propose the OS for execution of applications on VM. In this OS, the protection function, interprocess communication, and VM scheduling provided by the general purpose OS are implemented, and these are provided as system calls.

Keywords: Wireless Sensor Network OS, Virtual Machine, Overlay Network

1. はじめに

無線センサネットワーク (WSN) は，センサを搭載した複数のセンサノードによって構成される無線ネットワークである．WSN の特徴として，センサノードは電力が有限であり，計算資源も極小であることが挙げられる．このような理由から WSN 用の OS は，省資源性や低消費電力を目指して開発されてきた．そのため，これらの OS では多様なアプリケーションの実行を前提としておらず，プログラムの並列実行やプロセス間通信を行う場合は仮想マシンを利用しなければならない．本来これらの機能は OS が提供すべきであるが，前述の理由や OS と仮想マシンが個々に研究されているといった理由から両者の機能分割が明確ではない．そのため，PAVENET[1] や VAWS[2] のように OS と仮想マシンが同様の目的で研究されていることもある．

また近年，IoT(Internet of Things) の普及に伴い，増加したデータをクラウドに送信する前に処理を行うフォグコンピューティングという考え方がある．フォグコンピューティングの計算資源として既存のセンサノードを利用する場合，センサノードは再利用可能である必要がある．現在の WSN では，OS と仮想マシンの機能分割が明確ではないため，再利用可能であるとは言えない．

そこで本研究では，OS と仮想マシンの機能分割を明確にするために，仮想マシンの利用を前提とした OS を提案する．この OS では，汎用 OS が提供する保護機能，プロセス間通信，仮想マシンのスケジューリングなどの機能をシステムコールとして提供することで，OS と仮想マシンの機能分割を明確にする．このことは，仮想マシン開発者の負担軽減にも繋がる．また，ムーアの法則によりセンサ

[1] 芝浦工業大学 大学院 理工学研究科 電気電子情報工学専攻
Shibaura Institute of Technology
[2] 芝浦工業大学
Shibaura Institute of Technology
[a)] ma16070@shibaura-it.ac.jp

ⓒ 2017 Information Processing Society of Japan

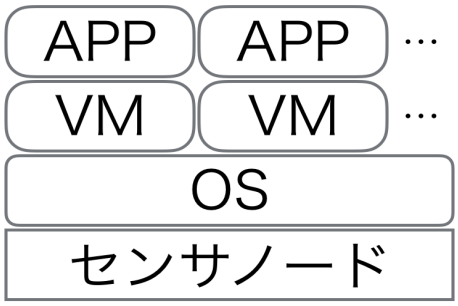

図 1 センサノード内構成図
Fig. 1 Structure of a sensor node.

ノードのリソースは今後ますます増大し，電力伝送や太陽光発電の進歩により消費電力の問題も改善されると仮定する．そのため本研究では，センサノードのリソースと電力は十分確保されているという前提で議論する．

2. OS に必要な要件

WSN では複数のアプリケーションを 1 つのセンサノードで動作させる需要がある．例えば，商品の在庫管理をしているセンサノードがあり，そこに商品の状態管理をする必要が生じた場合，既存のセンサノードに新たなアプリケーションを追加する必要がある．1 つのセンサノードで複数のアプリケーションを実行するためには，それらを管理するための OS が必要であり，各アプリケーションを並列実行させるための仕組みも必要である．また，すべてのセンサノードを回収するのは困難なため，アプリケーションの追加や変更を行うときはネットワーク経由でセンサノードにプログラムを送信して実行する．そのため，センサノードにはプログラムを解釈し実行するための仮想マシンを搭載しなければならない．従って，本研究で想定するセンサノード内の構成は図 1 のようになる．

この様な構成の場合，取得した値を即座に処理するためのリアルタイム性の保証に加え，複数の仮想マシンを管理するための機能，仮想マシンが利用するための統一された仕組み，別のセンサノードのプログラム同士が通信するための仕組みなどが OS の機能として必要である．

2.1 リアルタイム性の保証

WSN においてリアルタイム性は重要な要素の 1 つである．例えば地震の揺れを素早く検知するには，センサが読み取る値を即座に処理する必要がある．

WSN における OS は，イベント駆動モデルとスレッドモデルに大別できる．イベント駆動モデルではすべてのタスクがイベントにより起動され，実行中のイベントが終了するまで次のイベントは実行されない．そのため，イベント駆動モデルではリアルタイム性が損なわれる．一方スレッドモデルでは，優先度の低い処理の実行中に優先度の高い割り込みが発生すると，優先度の低い処理は中断され優先度の高い処理が割り込まれる．そのため，スレッドモデルではリアルタイム性をサポートすることができる．

本研究でもスレッドモデルを用い，優先度を持った多重割り込みを提供することでリアルタイム性を実現する．

表 1 システムコールの一部
Table 1 A pert of system-call.

システムコール名	処理
send_net()	ネットワークによる送信
receive_net()	ネットワークによる受信
start_app()	アプリケーションの起動
stop_app()	アプリケーションの停止
copy_app()	アプリケーションの複製
mem_alloc()	メモリの確保
mem_clean()	メモリの破棄

2.2 複数仮想マシンの管理

1 つのセンサノードで複数の仮想マシンを実行するには，OS が個々の仮想マシンを管理しなければならない．具体的には仮想マシンのスケジューリングと仮想マシンのコンテキストの保存である．本研究では，タイマーによる時分割スケジューリングを基本とし，仮想マシンに優先度を設けて多重割り込みを実現する．

また，OS は仮想マシンのコンテキストとしてスタックポインタとプログラムカウンタの値を保持する．タイマー割り込みが発生すると，汎用レジスタの値は仮想マシンのスタックへ退避するため，OS はレジスタの値を保存しておく必要はない．

2.3 システムコールの提供

WSN で仮想マシンを実行するための統一した仕組みとしてシステムコールを提供する．本研究で提供するシステムコールの一部を表 1 に示す．

プログラムはネットワーク経由で仮想マシンに送信されるため，ネットワークによる送受信を行うためのシステムコールが必要である．送信されたプログラムは仮想マシンによって起動され，必要がなくなれば停止される．また，センサノードへの負荷が高くなった場合はプログラムの状態を複製して別のセンサノードへ転送し，転送先で実行するための仕組みが必要である．さらに保護の観点から，プログラムに割り当てられたメモリ以外へのアクセスを禁止する必要がある．これをメモリにアクセスするためのシステムコールとしてラップすることで，仮想マシンで保護を意識する必要がなくなる．

2.4 アプリケーション同士の通信

1 つのセンサノードに複数のアプリケーションが存在し，各センサノード間でアプリケーション同士が通信を行う場合，お互いのアプリケーションは相手のセンサノード ID

ⓒ 2017 Information Processing Society of Japan

とアプリケーション ID を知る必要がある．また，これらの ID の組は，前節で述べたようなアプリケーションの転送によって変わることがある．そこで本研究では，変更をオンデマンドに取得するための仕組として分散ハッシュテーブル (DHT)[3] を用いる．DHT は，ファイルの位置情報を 1 つのセンサノードに偏ることなく管理する管理方式である．本研究ではセンサノード ID とアプリケーション ID を DHT によって管理し，後述する独自のプロトコルスタックを用いて通信を行う．

3. 関連研究

WSN では，OS やミドルウェアに関する様々な研究が行われてきた．本節では，これらの既存研究について前節で述べた OS の要件の観点から考察する．

PAVENET[1] はスレッドモデルを用いることでリアルタイム性を実現した OS である．PAVENET は Microchip 社の PIC18 マイクロコントローラに特化した実装を行うことで，リアルタイム処理を省資源かつ低オーバーヘッドで実現している．しかしながら，CPU に特化した設計を行っているため，移植性が低いという問題点がある．また，PAVENET ではスレッド間での共有データの排他制御を隠蔽するためのプロトコルスタックを提供しているが，アプリケーション間の通信については述べられていない．

t-kernel[4] は，不正なメモリアクセスに対する保護を実現している OS である．t-kernel はアプリケーションモジュールのロード時にモジュール内のすべての命令を解析し，メモリアクセスなどの処理を書き換えることで仮想メモリやメモリ保護の機能をハードウェアの機能を利用することなく実現している．そのため，t-kernel にはすべての命令を精査するための複雑な仕組みと多大なメモリが必要であり，変更後のモジュールサイズも 6 倍から 9 倍に増加してしまう問題もある．本研究では，システムコールとしてメモリにアクセスするための命令を提供することで，t-kernel の手法と比べ簡単に保護が実現できる．また，t-kernel では通信については述べられていない．

Agilla[5] は後述するエージェントを利用したミドルウェアである．Agilla では，コードやスタックなどの状態をひとまとまりにしたエージェントと呼ばれるソフトウェアが複数動作する．エージェントは自身を複製，消滅させながらセンサノード間を移動し処理を行う．Agilla では，エージェント間通信にタプルスペース [6] を持ちた黒板モデルを採用している．そのため，データのやり取りに即時性がく，山火事の検知や住宅の侵入検知など即時性が求められる場合には対応が遅れてしまう．

Nano-RK[7] はスレッドモデルの OS であり，ソケットに似た軽量ネットワークプロトコルスタックを提供している．データを送信したいアプリケーションはソケットを作成し，そのソケットを介して通信を開始することができる．Nano-RK はデータを受信すると，そのパケットヘッダに存在するポート番号を使用してアプリケーションのバッファを識別し，受信データを対象となるアプリケーションバッファにコピーした後，アプリケーションに通知する．しかし，Nano-RK はアプリケーションが移動することを想定していない．そのため，アプリケーションが移動すると，センサノードとアプリケーションとの対応が取れず通信することが出来ない．

Contiki[8] も Nano-RK と同様にマルチスレッドとネットワークプロトコルをサポートする OS である．Contiki にはシングルホップ通信とマルチホップ通信ができる軽量のプロトコルである Rime と，TCP/IP プロトコルを利用した uIP がある．Rime ではパケットにアプリケーション個別の ID を設定することができないため，アプリケーション同士の直接通信ができない．一方，uIP はアプリケーション ID を指定することができるが，TCP/IP プロトコルは，TCP のヘッダサイズが 20 バイト，IP のヘッダサイズが 20 バイトとサイズが大きい．これは後述する ZigBee のパケット規格に対して大きな割合を占める．

この様に，これら既存 OS ではアプリケーションの移動を考慮した直接通信の手段がない．次節では，本研究においてアプリケーション間で直接通信をするための設計を述べる．

4. プロトコルスタックの提供

WSN の通信方式においては，Wi-Fi や Bluetooth と比べ消費電力や接続可能台数の観点で優れている ZigBee が広く利用されている．ZigBee には，AT モードと API モードの 2 種類の通信モードが存在する．AT モードは，あらかじめ指定しておいた相手に対して，シリアルに書き込まれたデータを加工せずに送信する簡易的な通信方式である．そのため AT モードでは，通信先のアドレスを事前に通信元の XBee モジュールに書き込んでおく必要があり，一対一通信しかできない．一方 API モードは，送信先アドレスをプログラムから設定することができ，データの送信時にパケットも作成する．そのため API モードでは一対多通信ができ，パケットを読むことで送信元アドレスやデータ落ちを検出することができる．また，API モードでは送信のたびに送信や配送に問題があったかどうかを確認するための ACK が返され，3 回まで再送を試みる．これらのことから，WSN において ZigBee 通信を行う場合，API モードを利用することが望ましい．

前述したように，各センサノード間でアプリケーション同士が通信を行う場合，センサノード ID の他にアプリケーション ID が必要になる．ZigBee にはこのような概念がないため，OS が通信プロトコルを提供する必要がある．

ZigBee の無線規格である IEEE 802.15.4 はパケットの最大サイズが 127 バイトと定義されている．そのため，

ⓒ 2017 Information Processing Society of Japan

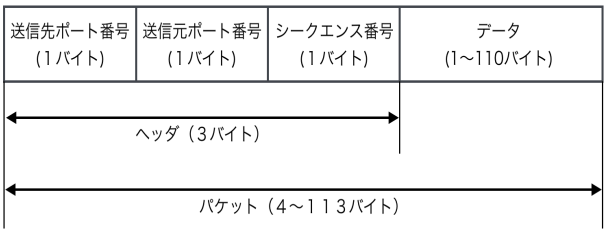

図 2 パケット構成図
Fig. 2 Structure of packet.

ZigBee が使用する 14 バイトのヘッダを除くと送受信できるデータサイズは 113 バイトである．ここに，前述した Conticki の uIP を用いると，ヘッダとしてさらに 40 バイト使用されてしまうため，使用できるデータサイズが 73 バイトにまで減少してしまう．使用できるデータサイズのうち実に 36%をヘッダで消費してしまうのはデータ通信効率の低下につながるため，ヘッダサイズを抑えた新たなプロトコルが必要である．

4.1 プロトコルの設計

本研究で提案するパケットの構成を図 2 に示す．このプロトコルではパケットのヘッダとして，送信先のポート番号，送信元のポート番号，シークエンス番号を加える．1 つのセンサノードで 256 種類以上のプログラムが実行されることは考えにくいため，ポート番号はそれぞれ 1 バイトで十分である．また，110 バイトより大きなデータを送信する場合は 110 バイト以下にデータを分割しなければいけない．この処理も OS によって行われ，データの分割時にはシークエンス番号もヘッダとして加えられる．ただし，データサイズが 110 バイト以下であればシークエンス番号として 0 を加える．このシークエンス番号も 1 バイトで表現される．つまり，28,160 バイトより大きなデータは一度に送信することができない．しかしながら，リソースの消費量が多いとされているスレッドモデルの MANTIS[9] でさえ，OS サイズは 14,500 バイトである．アプリケーションのサイズは OS のサイズより小さいと考えられるため，シークエンス番号のサイズは 1 バイトで十分だと言える．これらの理由によりパケットのヘッダサイズを 3 バイトに抑えることができた．これは使用できるデータサイズの 2%であり，十分に小さい値だと言える．

DHT で管理するデータは，センサノード ID，ポート番号，アプリケーション ID である．アプリケーションに対してデータを送信するときは，OS が DHT からアプリケーション ID を探索し，それに対応するセンサノード ID とポート番号を取得する．そして，ポート番号を元にパケットを作成し，センサノードに対してパケットを送信する．また，アプリケーションを別のセンサノードに転送する場合は，まず送信元の OS が DHT への読み書きをブロックする．そして，アプリケーション ID を Key とする value を，送信先のセンサノード ID とポート番号に変更してブロックを解除する．この仕組みによって，1 つのセンサノードで複数のアプリケーションが動作し，且つアプリケーションが移動する場合であってもアプリケーション同士の直接通信が可能となる．

5. まとめ

本研究では OS と仮想マシンの機能分割を明確にするために，仮想マシンの実行を前提とした OS の提案を行った．また，この様な仕組みになったときに OS に必要とされる要件をあげ，アプリケーション同士の通信についての設計を述べた．

現在，Arduino Uno を用いてゼロから OS の実装を行っている．現在までに，ZigBee の API モードを使用した通信，タイマ割り込み，スレッドの作成，幾つかのシステムコールの実装が完了した．今後は，多重割り込み，DHT，プロトコルスタックの実装を行う予定である．

参考文献

[1] 猿渡，鈴木，水野，森川：無線センサノード向けハードリアルタイムオペレーティングシステムの設計，情報処学会研究報告，ユビキタスコンピューティングシステム研究会，UBI-13-29．2007.
[2] 鈴木，猿渡，南，森川：無線センサノードのためのハードリアルタイム保証が可能な仮想マシン，電子情報通信学会論文誌，J92-B，1．2009.
[3] 江崎浩: P2P（ピア・ツー・ピア）教科書, インプレス R&D, 2008-1.
[4] L. Gu and J. A. Stankovic, *t-kernel: Providing Reliable OS Support for Wireless Sensor Networks*, Proc. 4th International Conference on Embedded Networked Sensor Systems, pp.114, Boulder, CO, USA, Nov. 2006.
[5] Chien-Liang Fok, Gruia-Catalin Roman, Chenyang Lu.: *Agilla: A Mobile Agent Midleware for Self-Adaptive Wireless Sensor Networks*, In Proceedings of the ACM Transactions on Autonomous and Adaptive System, Vol.4, No.3, 2009, Article 16.
[6] Gelernter, David. *Generative communication in Linda.*, ACM Transactions on Programming Languages and Systems TOPLAS 7.1 1985, 80-112.
[7] Eswaran A, Rowe A, Rajkumar R.: *Nano-RK: An Energy-Aware Resource-Centric RTOS for Sensor Networks*, Proceedings of the 26th IEEE Real-Time Systems Symposium, Miami, FL, USA. 58 December 2005.
[8] A. Dunkels, B. Gronvall, and T. Voigt, *Contik - a Lightweight and Flexible Operating System for Tiny Networked Sensors*, Proc. 29th Annual IEEE International Conference on Local Computer Networks, pp.455462, Tampa, FL, USA, Nov. 2004.
[9] Bhatti S, Carlson J, Dai H, Deng J, Rose J, Sheth A, Shucker B, Gruenwald C, Torgerson HR. *Mantis OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms*, Mob. Netw. Appl. 10:563579, 2005.

ⓒ 2017 Information Processing Society of Japan