卒業論文 2014 年度 (平成 26 年度)

D-Switch: A Dynamically Switchable Scheduling System for Operating Systems in Wireless Sensor Networks

指導教員

慶應義塾大学環境情報学部

徳田 英幸

村井 純

楠本 博之

中村 修

高汐 一紀

Rodney D. Van Meter III

植原 啓介

三次 仁

中澤 仁

武田 圭史

慶應義塾大学 環境情報学部

小町 芳樹

ysk@ht.sfc.keio.ac.jp

卒業論文要旨 2014 年度 (平成 26 年度)

D-Switch: 無線センサネットワークにおけるオペレーティングシステムに対する動的なスケジューリングスイッチング機構

近年技術の発展によりセンサノードの低価格化，高性能化が進み，それに伴いネットワークに繋がる物理センサが自動的に多様なデータをやり取りし，それらを様々な形で活用する無線センサネットワークが普及しつつある．主にその結果は，環境モニタリングやターゲットトラッキングにおいて顕著に表れており，センサネットワークが我々を支える生活基盤となり得る将来は遠くはない．

無線センサネットワークにおけるオペレーティングシステムはイベントモデルとスレッドモデルに分類できるが，リアルタイム性のある処理が長期間にわたって行われるような環境では，どちらのモデルも要件をすべて満たすには至っていない．Dunkels らの開発した Contiki[1] では，Protothreads[2] というシステムを採用することで，イベントモデルでありながらもスレッドモデルと同様にタスクの切り替えを行うことができる．しかしながら，Protothreads を採用した場合でも，タスク実行中にさらに優先度の高いタスクの実行を要請された際に，スレッドモデルのように割り込みをし，タスクを切り替えることはできないのが現状である．

本研究では Protothreads を利用し，時間的制約を伴ったイベントが発生する環境における，オペレーティングシステムに対する動的なスケジューリングスイッチング機構を提案する．本システムは通常イベントモデルとしてタスクの実行を行うことで省電力を実現し，リアルタイム性の必要なタスク発生時には割り込みを発生させ，優先的にそのタスクの処理を行うことを可能にしている．最後に本システムの性能を評価し，有用性を証明する．

キーワード：

無線センサネットワーク，環境モニタリング，ターゲットトラッキング，オペレーティングシステム，リアルタイム処理

慶應義塾大学環境情報学部

小町 芳樹

Abstract of Bachelor's Thesis Academic Year 2014

D-Switch: A Dynamically Switchable Scheduling System for Operating Systems in Wireless Sensor Networks

In recent years, developed technology made sensor nodes cheaper and high performanced. Along with the advencement of technology, physical sensors connected to the network transmit and receive various datas automatically, and Wireless Sensor Networks (WSN) which use that datas in various ways are popularized. In particular, the results are remarkable in environmental monitoring and target tracking. In the future, WSN will become the foundation of our life supporting.

Operating Systems (OS) in WSN are divided into events model and threads model. However, neither one of these models has still fulfilled all of those requirements in environments where real time tasks have been processed for a long period. Event models like OS can switch tasks as same as threads model by using Protothreads[2] in Contiki[1] proposed by Dunkels et al. However, if a task which has more priority was posted while another task has been processed, the processed task cannot be interrupted, and cannot be switched to the arrived task with Protothreads in the existing circumstances.

In this research, we propose a dynamically switchable scheduling system for operating systems using Protothreads in environments where events with time constraint occurred. This system enables to save energy by executing tasks as a general events model, to trigger interruption, and preferentially to process tasks when real time task has been occurred. Finally, we evaluate this system and prove feasibility.

Keywords :

Wireless Sensor Networks, Environmental Monitoring, Target Tracking, Operating Systems, Real Time Processing

Yoshiki Komachi

Faculty of Environment and Information Studies
Keio University

目次

図目次

表目次

第 1 章

序論

本章では，まず無線センサネットワークオペレーティングにシステム関する本研究の背景を述べ，次いで，本研究の目的について言及し，最後に本論文の構成を示す.

1.1 本研究の背景

近年センサの低価格化，高性能化により，センサネットワークが普及し，それに伴いセンサ用のオペレーティングシステムの研究も発展を遂げている．センサネットワークのオペレーティングシステムには主に 2 種類あり，省資源で低消費電力を実現したイベントモデルと，リアルタイム処理を可能としたスレッドモデルが存在している．センサネットワーク用のオペレーティングシステムではイベントモデルが主流となっている．

イベントモデルの TinyOS[3][4] は，省資源で低消費電力の実現に成功しているが，リアルタイム処理を行うことができておらず，それに加えてユーザが一連の処理を細かい処理に分割しなければならないため，プログラムが書き辛い．それに対して，Nano-RK[5] はリアルタイム処理をサポートしているが，イベントモデルのオペレーティングシステムと比較すると資源の消費が大きいため，電力の消費が激しい．

Contiki[1] はイベントモデルを採用したオペレーティングシステムで Protothreads[2] と呼ばれるメカニズムを使用しており，イベントモデルでありながらスレッドモデルと同様の挙動を行うことができる．そのため，イベントモデルの短所であるプログラムの書きにくさを解消しつつ，実行中のタスクの切り替えを可能にしている．しかし，タスクを処理する優先順位を明確に決める手法が定められていないため，タスクの切り替えはできるが，未だにリアルタイム処理は不可能なのが現状である．

1.2 本研究の目的

Contiki はイベントモデルでありながら，マルチスレッドでタスクを変更することができるため，リアルタイム処理と親和性があると推測できる．本研究では，Contiki における動的なスケジューリングスイッチング機構を提案し，省資源性で低消費電力を保ちながら，リアルタイム処理の可能なオペレーティングシステムの実現を目的とする．

1.3 本論文の構成

本論文では，2 章において，無線センサネットワークにおけるアプリケーションを分類し，それぞれにおける特徴について議論する．3 章では，無線センサネットワークにおけるオペレーティングシステムについて紹介し，各システムについて比較を行う．4 章で，無線センサネットワークにおけるモデル選択に関する問題点について言及し，5 章では，その問題を解決する手法について，設計と実装の観点から考察する．6 章にて，提案手法に対する評価を行い，7 章で本研究のまとめを行う．

第 2 章

無線センサネットワーク

本章では，最初に無線センサネットワークについて導入する．次に本研究の想定環境である，環境モニタリングとターゲットトラッキングについてそれぞれ例を提示し，エネルギーの節約とリアルタイム処理の必要性について述べる．

2.1 はじめに

近年，小型センサの発展を促すセンサ微小電気機械システム技術の進歩により，無線センサネットワークが急速に普及してきている．無線センサネットワークで使用されるノードは非常に小さく，処理機能や計算リソースが制限されているものの，センシング，計測，情報収集能力，そしてユーザに対してセンシングしたデータを転送する機能でさえも備えている．代表的なセンサノードとして，Iris Mote[6] や MicaZ[7]，SunSPOT[8] などがある．

Iris Mote は図 2.1 のようなものであり，アメリカの Crossbow Technology 社が開発したセンサネットワーク用の無線端末である．それに対して，MicaZ はカリフォルニア大学バークレー校におけるスマートダストプロジェクト [9] によって開発された．ハードウェア，オペレーティングシステム，開発言語，シミュレータ，ライブラリといったアプリケーション開発環境を提供しており，アプリケーション開発が容易であるため，現在センサネットワークの研究で多く使用されている．図 2.2 に MicaZ の写真を示す．MicaZ は小型なため，様々な場所に応用することができる．SunSPOT は Sun Small Programmable Object Technology の略称で知られる，図 2.3 のような無線センサネットワークデバイスであり，Java で実装することができるため，初心者でも扱いやすいセンサデバイスとなっている．

図 2.1　Iris Mote

図 2.2　MicaZ

図 2.3　SunSPOT

無線センサネットワークアプリケーションを分類した場合，図 2.4 のように環境モニタリングとターゲットトラッキングの大別される [10]．環境モニタリングとターゲットトラッキングについてはそれぞれ，2.2 と 2.3 にて詳細に解説する．

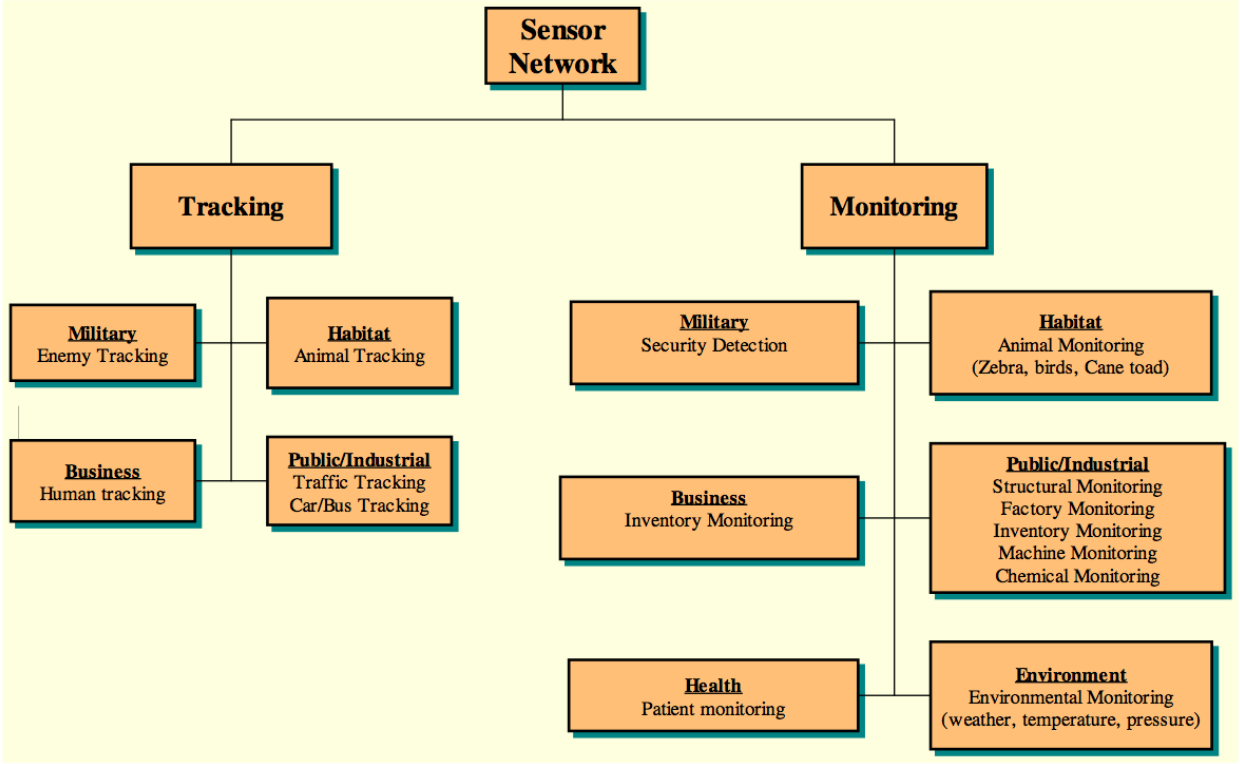

図 2.4　センサアプリケーションの概要

2.2　環境モニタリング

環境モニタリングの歴史は長く，かつては手動によりデータの収集を行っていたが，デジタルデータロガーの普及により，より安く，容易にデータの授受を行うことができるようになった．しかしながら，環境モニタリングでは大抵の場合，対象となる範囲全域での監視が求められていることに対して，デジタルデータロガーを利用した場合，ある一点においてのみのモニタリングしか行うことができなかったため，デジタルデータロガーに代わる環境モニタリングの次なる手段として，無線センサネットワークが採用され始めている．センサノードには様々なセンサが搭載されており，センサノードを対象とする地域に配置し，ネットワークを構築することで，デジタルデータロガーでは成し得なかった，広範囲におけるモニタリングを可能としている．

無線センサネットワークを用いた環境モニタリングでは，モニタリング対象の行動に変化があった際には，それに応じたタスクを優先的に行わなければならない．このような時間的制約を伴ったタスクの処理が必要なアプリケーションを想定した場合，オペレーティングシステムとしてリアル

タイム処理を行うことが可能なものを選択することが多い．無線センサネットワークにおけるオペレーティングシステムのリアルタイム性については，3.3 において詳細に述べる．本節では代表的な環境モニタリングとして，森林火災検知と野生動物の生態調査を例に挙げる．

2.2.1 森林火災検知

森林火災は原生林において生じる制御不可能な火災であり，自然資源と人的資源の両者に多大な被害を与える．具体的に，木々を根絶やし，基盤となる施設を燃やし，さらには都市部付近における死亡者数を増加させてしまう場合も少なくない．森林火災の共通の原因として，落雷や人間の不注意，そして燃料の野ざらしによる極度の乾燥などが挙げられる．火災は森林の生態系の一部としてみなされることもあるが，ほとんどの場合において，火災により発生する公衆安全や自然資源に対するダメージは耐え難いものであり，被害を最小限に抑えるためにも火災の早期発見と鎮圧が極めて重要である．しかしながら低解像度かつ長期走査により，中規模もしくは大規模火災監視システムは適時検知を行うことは困難であり，高精度のリアルタイム火災検知を提供できる拡張性のある手法が求められていた．

近年無線センサネットワークにおける発展が著しく，森林火災を検知する際に役立つ，温度，相対湿度，さらに煙などの様々な現象をセンシングすることが可能であることから，無線センサネットワークがこれらの問題を解決できる手法として関心を集めている．火災が発生している間，定期的な監視を実現するために，センサノードは数週間にわたってオペレーションをすることが可能であり，森林火災によって生じる被害と比較しても，低コストで大規模センサネットワークは容易に展開することができる．

森林火災モニタリングに対する無線センサネットワークの可能性について考察した Doolin らは，サンフランシスコとカリフォルニアにおける制御下にある火災から実験的な結果を導き出した [11]．システムは GPS を搭載した数個の Mica Mote から成り，収集された温度，湿度，そして気圧データはベースステーションへと転送され，データベースへと格納後，異なるアプリケーションによって提供される（図 2.5）．実験から，火災発生地域多数のセンサを設置することで，火災が発生するより前に火災発生の前触れを予測することができると述べている．

また，森林火災の消火においては類焼を予測するために温湿度の変化を観測することが重要であり，これまでは消防士が測定器を持って位置時間ごとに測定を行っていたが，この手法では消防司令所への報告に数分を要することに加え，類焼する可能性のある危険な場所に消防士を向かわせなければならない．森林火災を消火する消防士を支援するシステムである，FireWxNet[12] ではこのような温湿度の測定を，無線センサネットワークとそのネットワーク間を接続する長距離無線リンクで構成されるシステムを利用して行うことを提案している（図 2.6）．実際にこのシステムを展開し，温湿度の測定結果や，データの通信成功率，バッテリ寿命について考察している．局所的な測定をセンサネットワークで行い，そのデータを長距離無線リンクで伝送することで，より現実的な無線センサネットワークの展開ができるという結果を示している．

森林火災検知システムにおけるネットワークライフタイムは，約 6 ヶ月から成る火災シーズンを

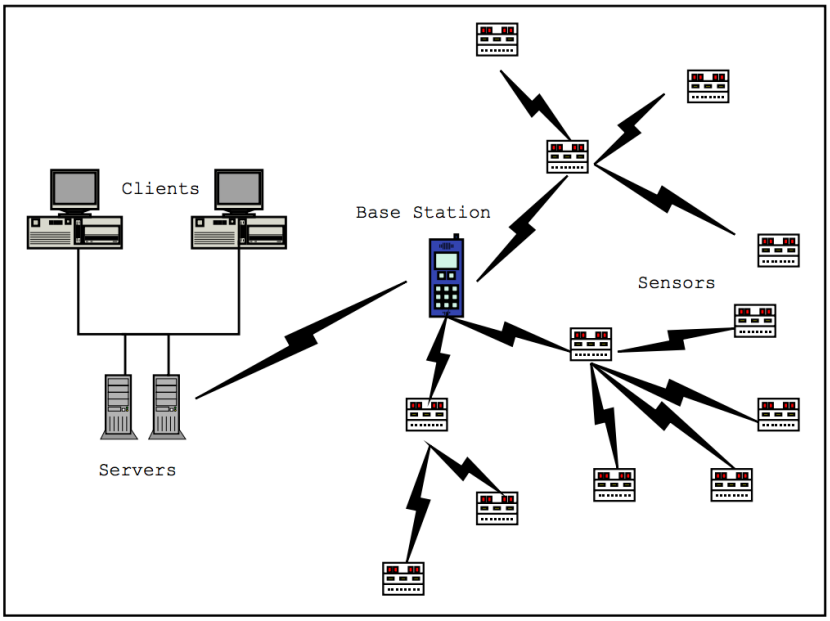

図 2.5　無線センサを用いた森林火災モニタリングシステム構成図

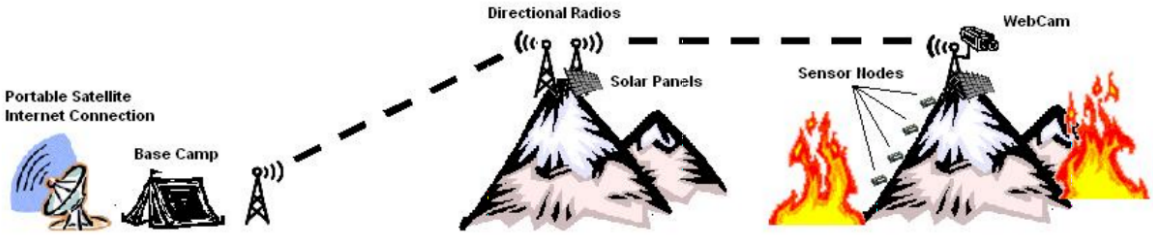

図 2.6　FireWxNet 概要

少なくとも上回ることが望ましいが，一般的にそれぞれのノードが数週間にわたって稼働することは難しいのが現状である．したがってこの要件を満たすためにも，エネルギーの消費を抑えるようなシステムを提案する必要がある．

2.2.2　野生動物の生態調査

生命科学における研究者たちは動植物の観察時の人間が与える影響について懸念を抱いていた．近年，無線センサネットワークが既存の動植物の観察手法に代わる手法のひとつとして注目されている．動物の場合一般的にセンサは繁殖期のはじまりに前もって設置され，植物の場合にはセンサの設置は地面が凍る，もしくは休眠状態にあるときに設置されるため，動植物に与える影響が少なく，繰り返し現地調査を行うことが困難であるような危険な島におけるモニタリングへの負担も軽減できることから，無線センサネットワークを基盤としたモニタリングは，センサネットワークが

普及する以前の弊害を無視した手法と比較して，目覚ましい業績をあげている．

カリフォルニア大学バークレー校とインテルの研究員によりグレート・ダック島に展開された，Mica Mote を基盤としたウミツバメの生態を観測するための階層型センサネットワーク [13] を生態調査の例として紹介する．システム構成図は図 2.7 の通りである．動物の掘った穴など，調査対象となる 32 箇所に Mica Mote が設置され，区画ごとにグループ化されたそれらの Mote は，センサデータをゲートウェイまで転送し，さらにローカルネットワークを通して，ベースステーションまでデータを転送する．その後，ベースステーションでデータを記録し，定期的に複製されたデータはデータベースへと格納される．ユーザはデータベースサーバ内の複製されたデータにアクセスすることや，サンプリングレートや電力を操作する変数を調整するなどの，インタラクションをするためのデバイスを利用することが可能である．

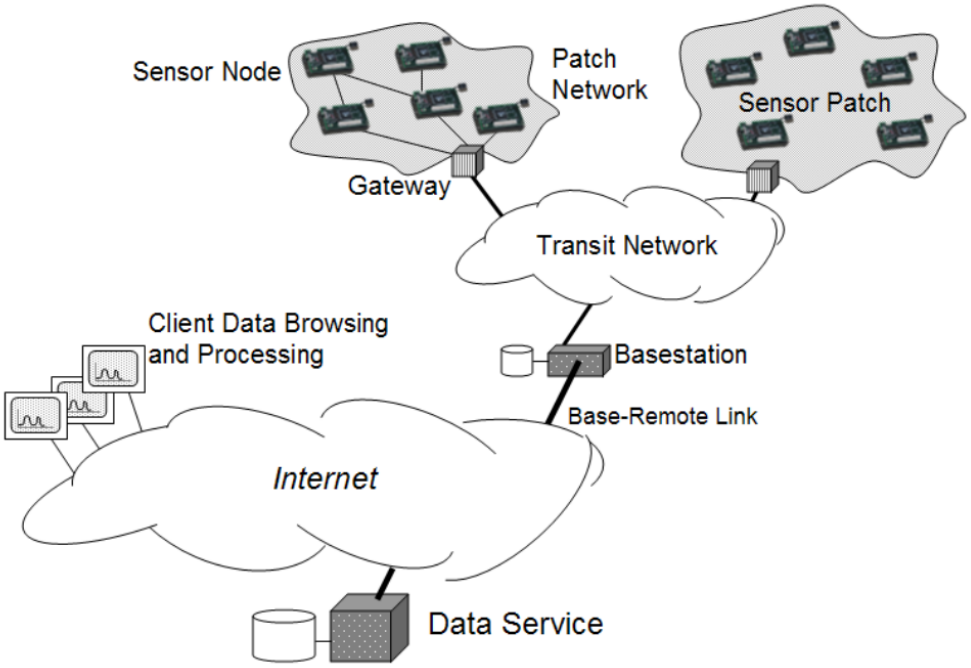

図 2.7　GDI のシステム構成図

調査対象の特徴によっては，現地調査を複数回行うことが困難となる場合があることは既に述べたが，無線センサネットワークに用いられる小型デバイスにおけるバッテリの駆動時間は，一般的なラップトップなどと比較してかなり短いのが現状である．バッテリを交換する回数が増えるにつれ，無線センサネットワークから得られる利益は減少してしまうため，火災検知システムと同様に，省エネルギーが実現できるようなシステム構成が求められている．

2.3　ターゲットトラッキング

特定のイベントの観測のための，資源の限られた幾千ものセンサノードから成る無線センサネットワーク技術に普及により，様々なシナリオにおける無線センサネットワークの利活用が現実的になりつつあることは既に述べたが，交通量管理や侵入者検知などのターゲットトラッキングの分野も，その影響が顕著に表れている分野のひとつである．

イベントを検知したセンサの周辺ノードはそのイベントを監視し，ラップトップやベースステーションのような外界と通信する機能を持ったシンクノードに対してイベントの発生を知らせるのだが，ターゲットトラッキングにおいて，対象を検知し，その旨をシステムのゲートウェイノードに警告するタスクにより中継されたデータは，適時にゲートウェイまで届けられるべきである．したがって，環境モニタリングと同様にターゲットトラッキングでもリアルタイムオペレーティングシステムを採用すべきである．本節ではターゲットトラッキングの例として，軍用監視システムについて言及する．

2.3.1　軍用監視システム

監視任務において，ターゲットとする敵の能力や位置に関する情報を正確に入手することが何よりも重要であるが，そのような任務には隊員の危険を伴うものが多い．無線センサネットワークを利用することで，リスクを最小限に抑えつつ，敵の侵入に関する情報を的確に把握する研究が盛んに行われている．図 2.8 は軍用監視システムにおける概念図である．

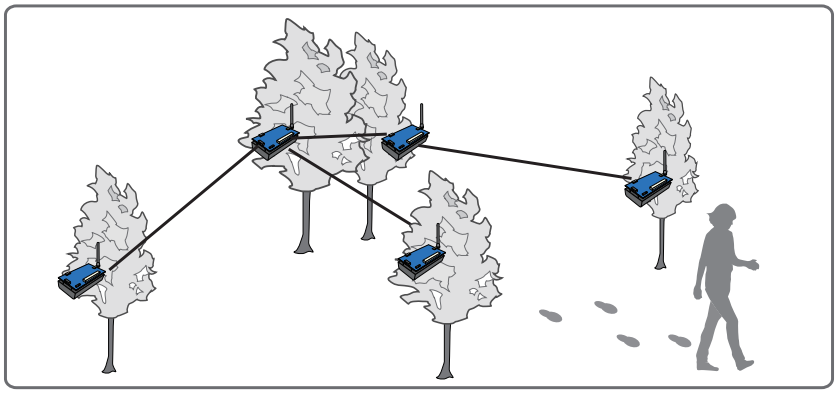

図 2.8　軍用監視アプリケーションにおける概念図

近年提案されている監視システムのほとんどがシミュレーションを通して期待できる成果を挙げているが，シミュレータを利用して単純化された仮説は実環境において意図した通りの結果を得られないこともしばしばである．Tian He らの研究 [14] では，省エネルギーかつステルス性を保ちながら移動する車両の位置を検知し，追跡するセンサデバイスを用いた監視システムの設計

と実装をし，実空間においてシステムの感度を適宜調整することによって，省エネルギー性と監視システムの性能におけるトレードオフについて考察している．またこれに加えて，同期型の通信プロトコルを設計し，常にビーコンを送信し，イベントが発生した場合はビーコンの送信を停止する Proactive 型と，イベントが発生した場合にビーコンを送信する Reactive 型の実装を行い，それぞれについて省電力性やステルス性などの評価も行っている．

監視システムアプリケーションを用いた任務は数日から数ヶ月にかけて行われるものが一般的であり，任務中の秘密保持の重大性や任務が敵の管轄地域行われる場合もあることから，任務期間中に資源の制限されたセンサデバイスの手動による充電はできないことがほとんどである．したがって任務期間中継続して使用するために，監視システムにおけるアプリケーションではセンサデバイスの寿命を向上させるような省エネルギーな構成が必要とされる．

また軍用の監視システムにおいて，デバイスが発見され，それに伴った迎撃を未然に防ぐことは極めて重要である．センサデバイスを小型化することにより，デバイスの発見を物理的に困難にすることができるが，もしセンサデバイスが監視をする際に活発に通信をする場合，無線周波数が傍受されてしまう可能性が高い．重要なイベントが発生したときを除いて，通信を控えることが求められている．

2.4 まとめ

本章では，まず，技術の発展によりセンサノードの低価格化，高性能化が進み，それに伴い，ネットワークに繋がる物理センサが自動的に多様なデータをやり取りし，それらを様々な形で活用する無線センサネットワークが普及してきたことを説明した．次いで，無線センサネットワークアプリケーションを環境モニタリングとターゲットトラッキングに分類し，それぞれについて例を紹介しながらエネルギー節約とリアルタイム処理の重要性について言及した．

第 3 章

無線センサネットワークにおけるオペレーティングシステム

本章では，まず，無線センサネットワークにおけるオペレーティングシステムについての導入を行う．次いで，無線センサネットワークにおけるオペレーティングシステムを分類し，それぞれについて具体的なオペレーティングシステムを例に挙げながら，その特徴についての考察を行う．最後に，それぞれのモデルを比較し，それぞれのモデルを採用するメリットとデメリットについて示す.

3.1 はじめに

無線センサノード上ではアプリケーションに応じてセンシング，データ処理，アドホックルーティング，無線の MAC 層，無線の物理層，センサノード同士の時刻同期，通信の暗号化など多様なタスクを並列に実行することが求められている [15]．無線センサネットワークにおけるオペレーティングシステムにはこれらのタスクを効率よく開発，実行するための仕組みが必要である．本節では，無線センサネットワークにおけるオペレーティングシステムを構築するにあたって，前提となる知識について説明する．

3.1.1 資源

省資源性はセンサノードの低消費電力性を実現するために重要な要素のうちのひとつである．計算資源と消費電力は密接に関係するため，できるだけ少ない計算資源で動作可能なオペレーティングシステムが求められている．オペレーティングシステムが要求する計算資源が大きいことによって，必要とされる CPU の性能が増加するのは好ましいことではない．

3.1.2 オーバヘッド

オペレーティングシステムにおけるオーバヘッドとは，処理に時間がかかるようになるなど，システムの負荷になるようなものを指すことが多く，省資源性と同様に，オーバヘッドを低く保つことも低消費電力性を実現するために必要となる．無線センサネットワークではタスクの多くは周期的な間欠動作であり，実行時間が短く，CPU のほとんどの時間はスリープ状態で実行状態の割合が小さいため，実行時におけるオペレーティングシステムのオーバヘッドを無視することはできない．

3.1.3 リアルタイムスケジューリング

ジョブの実行を設定された時間通りに作動させるリアルタイム処理は主にハードリアルタイム処理とソフトリアルタイム処理の 2 種類に分類される．ハードリアルタイム処理とソフトリアルタイム処理の特徴をそれぞれ以下に述べる．

ハードリアルタイム処理

課せられた処理が期限内に終了しなかったとき，システム全体に致命的なダメージが生じてしまうリアルタイム処理がハードリアルタイム処理である（図 3.1）．したがって，期限内での終了が保証されなければならないシステムに用いられる．

ソフトリアルタイム処理

ソフトリアルタイム処理を行うシステムでは，期限内に処理が終了しなくてもシステム全体に致命的なダメージを与えることはない．ただし，処理自体の価値は終了期限とともに減少していく（図 3.2）．

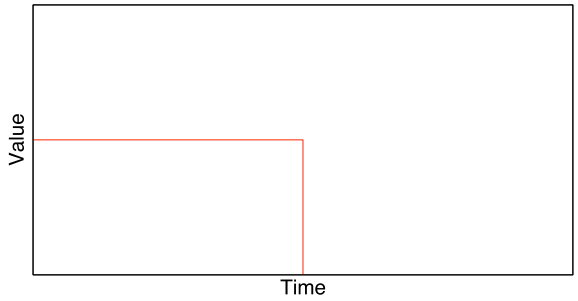

図 3.1　ハードリアルタイム

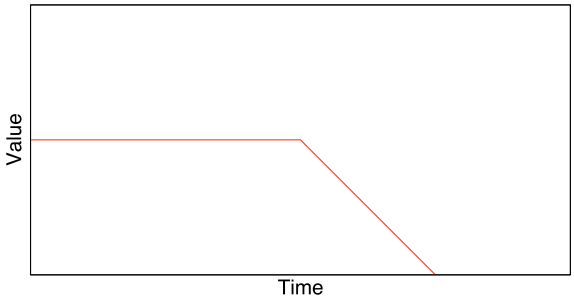

図 3.2　ソフトリアルタイム

3.2　イベントモデル

イベントモデルで構築されたオペレーティングシステムは全てのタスクをイベントによって起動し，run-to-completion で実行する形態のオペレーティングシステムである [16]．イベントモデルは図 3.3 に示されるように，ひとつのイベントループと多数のイベントハンドラから構成される．イベントループはイベントの到着を待ち，イベントが届くとそのイベントに関連付けられているイベントハンドラを実行する．イベントモデルではイベント駆動型プログラミングによってアプリケーションが記述される．イベントハンドラは寿命の短い run-to-completion で記述され，プリエンプションされることがない．つまり，イベントモデルにおいてタスクは関数呼び出しと等価であり，実行ストリームがひとつで実現されるため各タスクでローカル変数の領域を共有可能であることから，消費電力を抑えつつ，並列性を実現できる．また，各タスクが不可分に実行されるため共有資源に対する排他制御が不要となり，安全性が高い．さらに，CPU の特殊な機能を用いなくても実装できることから移植性も高い．低消費電力の実現から，無線センサネットワークにおけるオペレーティングシステムとしてイベントモデルが選択される場合が多く，現在最も主流なオペレーティングシステムとなっている．本節では，TinyOS と Contiki をイベントモデルの例として提示し，それぞれの特徴について言及する．

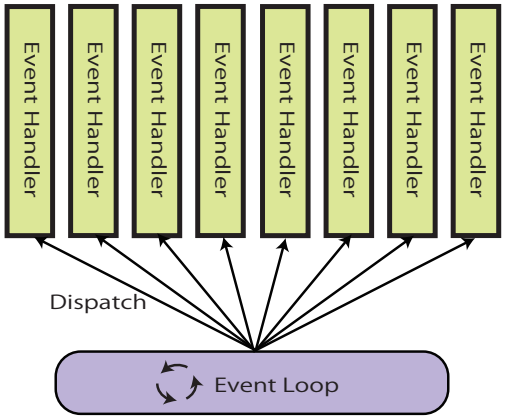

図 3.3　イベントモデル [16]

3.2.1　TinyOS

イベントモデルのオペレーティングシステムの中でも，最も代表的なものが TinyOS[3][4] である．TinyOS はカリフォルニア大学バークレー校のスマートダストプロジェクト [9] で開発されたオペレーティングシステムで，現在無線センサネットワークの標準的なオペレーティングシステムとして扱われており，Crossbow 社から発売されている Mica2 や MicaZ[7]，Telos[17]，iMote[18] 上で動作する．TinyOS は CPU の特別な機能を使用せずに実装可能であるため，移植性が高く，ATMEL の AVR128L や Texsus の MSP430，ARM7 などさまざまな CPU に移植されている．TinyOS のアーキテクチャを図 3.4 に示す．

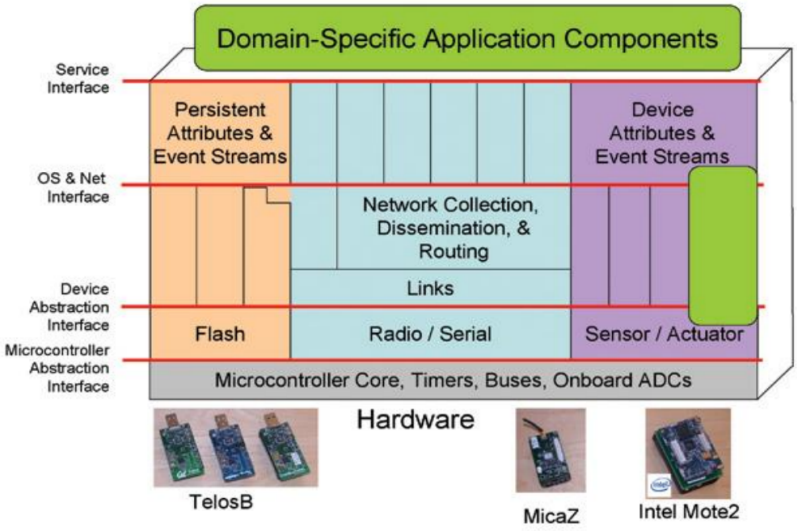

図 3.4　TinyOS アーキテクチャ [19]

TinyOS では nesC[20] と呼ばれるイベント駆動型の新しい言語で複数のイベントハンドラをひとつのモジュールとして設計可能な機能を提供している．nesC 言語は nesC コンパイラによって C 言語のコードに変換された後，C コンパイラによって実行形式へと変換される．nesC 言語を採用することによって，イベントモデルの持つプログラムの開発のし辛さを提供し，新しい言語を学ばなければならないという手間がかかってしまうものの，nesC はイベントモデルに特化した最適化を行っているため省資源性を実現している．

3.2.2 Contiki

Contiki[1] は C 言語で記述された，イベントモデルを採用したオペレーティングシステムである．Contiki では動的モジュールを実現するために Executable and Linkable Format（ELF）形式のファイルをロード可能な仕組みを提供している [21]．Contiki のアーキテクチャを図 3.5 に示す [22]．

Contiki において，イベントモデルの利点を失わずにスレッド形式でプログラムの書きやすさを実現したのが Protothreads[2] である．Protothreads ではイベントハンドラの中でスレッド的に動作させたい部分を PT_BEGIN と PT_END で囲み，PT_WAIT_UNTIL で条件付きブロックを行うことで，イベントハンドラ実行中でも他のタスクに切り換えることを可能にしている．この Protothreads については，4.1 において詳細に述べる．

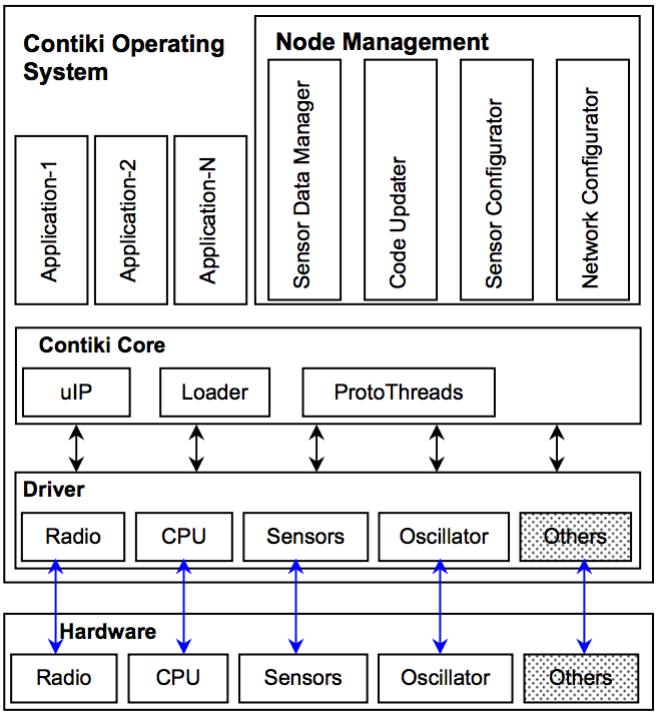

図 3.5　Contiki アーキテクチャ

3.3 スレッドモデル

スレッドモデルは図 3.6 に示されるように，複数のスレッドから構成され，各スレッドはそれぞれ独立に実行ストリームを持っており，低い優先度のスレッドは高い優先度のスレッドにプリエンプションされるという特徴を持つ．スレッドモデルではユーザはあたかも CPU を占有しているかのように一連の処理をひとつのスレッドとして記述することができるため，プログラムが書きやすい．またプリエンプションを行うことも想定しているため，リアルタイム処理をサポートすることができる．本節では，スレッドモデルの例として，MANTIS OS と Nano-RK を紹介し，その特徴について考察する．

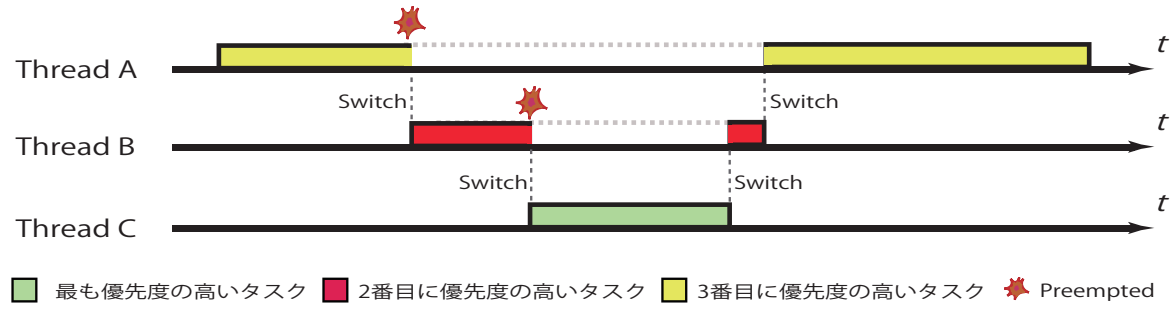

図 3.6　スレッドモデル

3.3.1 MANTIS OS

MANTIS OS[23] は Linux や FreeBSD などで用いられているスレッドと同様の機能をセンサノード上で実現したスレッドモデルのオペレーティングシステムである．開発者は Linux や FreeBSD などで用いられているソフトウェアを大きな変更なく MANTIS OS 上に移植することができる．また，MANTIS OS 上の 1 つのスレッドとして TinyOS[3][4] を実装することも可能であり [24]，さまざまなソフトウェアリソースを MANTIS OS 上で動作可能であることも特徴的である．MANTIS OS は C 言語で実装されており，アプリケーション開発者も C 言語を用いて開発を行うことが可能である．図 3.7 は MANTIS OS のアーキテクチャである．

3.3.2 Nano-RK

Nano-RK[5] は，無線センサネットワークにおけるマルチタスク処理機能を備えたリアルタイムオペレーティングシステムであり，リソースの使用が制限された無線センサネットワークにおける，マルチタスク処理や優先度順位スケジューリング，マルチホップネットワークのサポートを実現する．2Kbits の RAM と 18Kbits の ROM を使用し，Rate Monotonic Scheduleing[25] と Rate Harmonized Scheduling[26] を利用することによって，ハードリアルタイムアプリケーショ

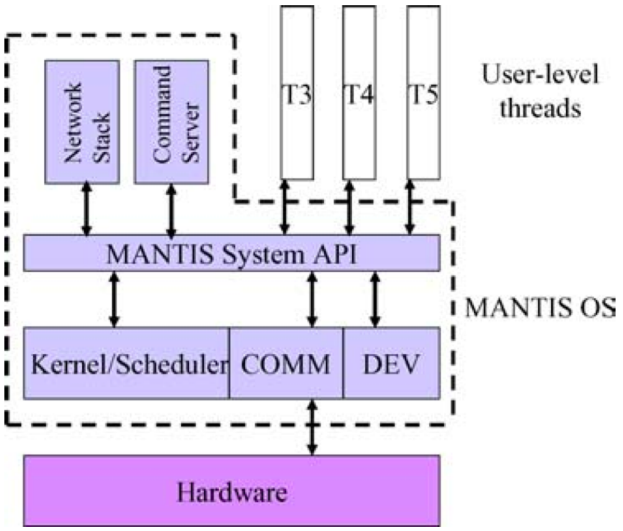

図 3.7　MANTIS OS アーキテクチャ

ンとソフトリアルタイムアプリケーションの両方をサポートする．現在動作確認がとれているセンシングプラットフォームとして，MicaZ[7] と FireFly[27] が挙げられる．Nano-RK のアーキテクチャは図 3.8 のとおりである．

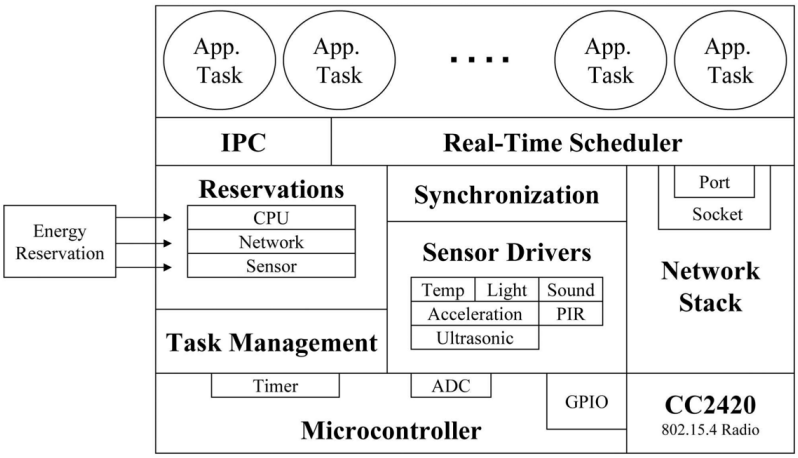

図 3.8　Nano-RK アーキテクチャ

3.4　イベントモデルとスレッドモデルの比較

表 3.1 にイベントモデルとスレッドモデルにおけるメリットとデメリットを示す．既に述べたように，イベントモデルではプリエンプションされることを前提としていないため，実行ストリーム

がひとつで実現され，ローカル変数の領域がそれぞれのタスクで共有することができるようになることから，消費電力の少ないシステムを構築することが可能になる．しかしながら，イベントモデルではユーザが一連の処理を細かい処理に分割しなければならないため，アプリケーションの開発が困難になってしまい，リアルタイム処理のサポートもできない．

イベントモデルに対して，スレッドモデルでは複数のスレッドがそれぞれ独立に実行ストリームを保持しているため，プリエンプションを可能とする．結果として，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．また，それぞれの処理をひとつのスレッドとみなし記述をすることが可能となるため，開発者がアプリケーションを開発することを容易にする．しかしながら，プリエンプション時の電力の消費が激しいことや，スレッド間の共有資源へのアクセス制御が必要となるために安全性が損なわれるなどの欠点を兼ね備えている．

表 3.1　オペレーティングシステムの比較

	メリット	デメリット	
イベントモデル	低消費電力	バッチ処理の採用	開発が困難である
スレッドモデル	リアルタイム性のサポート	高消費電力	

3.5　まとめ

本章では，まず，無線センサネットワークにおけるオペレーティングシステムについて導入した．次いで，無線センサネットワークにおけるオペレーティングシステムをイベントモデルとスレッドモデルの 2 つのモデルに大別し，各モデルにおけるオペレーティングシステムの概要を述べた．最後に，イベントモデルとスレッドモデルを比較し，イベントモデルではタスクの中断をしないことを前提に設計されているため，現在のセンサネットワーク用のオペレーティングシステムでは低消費電力であることと，リアルタイム性のサポートはトレードオフの関係にあるということを示した．

第 4 章

問題提起

本章では最初に，イベントモデルである Contiki において，スレッドモデルと同様にタスクを切り替えることを可能にした Protothreads について紹介する．その後，既存研究における問題意識について言及し，本研究における目的について述べる．

4.1 Protothreads

3.2 で述べたように，イベントモデルは組み込みシステムやセンサネットワークにおいて，主流なオペレーティングシステムとなっている．しかしながら，イベントモデルは消費電力を低く維持できる一方で，ユーザが一連の処理を細かい処理に分割しなければならないため，プログラムが書き辛いという問題が発生する（Listing4.1）．Contiki[1] における，Protothreads[2] はイベントモデルでありながらもスレッドモデルと同様の挙動を行うことができるため，電力の消費を抑えつつ，イベントモデルにおける開発を行うことが困難であるという欠点を補いながら（Listing4.2），実行中のタスクを切り替えることを可能にしている．本節では，Protothreads の機能について詳しく述べる．

Listing 4.1　Protothreds を使用せずに記述した場合

```
1  struct pt {
2      unsigned short lc;
3  };
4
5
6  static int counter = 0;
7  static
8  char example(struct pt *pt) {
9      char PT_YIELD_FLAG = 1;
10
11     if(PT_YIELD_FLAG) { ; }
12
13     switch((pt)->lc) {
14         case 0:
15             while(1) {
16                 do {
17                     (pt)->lc = __LINE__;
18
19         case __LINE__:
20                         if( !(condition) ) { // counter != 1000
21                             return PT_WAITING;
22                         }
23                 } while(0);
24
25                 printf("Threshold reached\n");
26                 counter = 0;
27             }
28     }
29     PT_YIELD_FLAG = 0;
30     (pt)->lc = 0;
31
32     return PT_ENDED;
33 }
```

Listing 4.2　Protothreds を使用した場合

```
#include"pt.h"


static int counter = 0;
static
PT_THREAD(example(struct pt *pt)){
    PT_BEGIN(pt);

    while(1) {
        PT_WAIT_UNTIL(pt, counter == 1000);
        printf("Threshold reached\n");
        counter = 0;
    }

    PT_END(pt);
}
```

4.1.1　メモリ

一般的なスレッドモデルのオペレーティングシステムでは，図 4.1 のようにそれぞれのスレッドにそれぞれのスタックを必要とする．しかし，メモリが制限されているセンサネットワークのようなシステムでは，スタック用のメモリは静的に保持されなければならないため，このメモリを他の目的で使用することはできない．

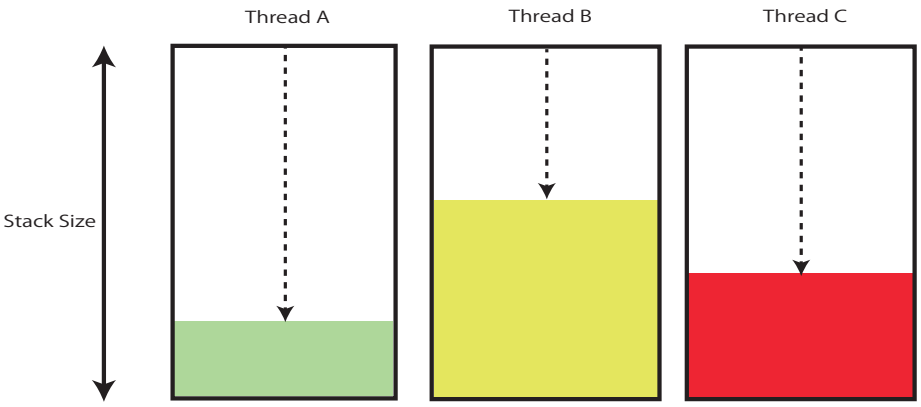

図 4.1　一般的なスレッドモデルにおけるスタック

それに対して，Protothreads におけるスタックを図 4.2 に示す．Protothreads を使用した場合，すべてのプログラムは同じスタックを共有し，実行されることとなる．つまり，Protothreads を利用しているオペレーティングシステムではそのような事態を防ぐために，マルチスレッドを実現しつつも，ひとつのスタックをあたかも複数個あるように見せかけている．

4.1.2　イベント

イベントモデルである Contiki では，イベントを受け取るとプロセスが実行される．ここでは Contiki で扱われる非同期イベント，同期イベント，そしてポーリングの違いについて説明する．

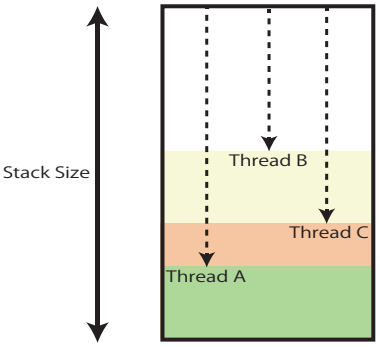

図 4.2　Protothreads におけるスタック

非同期イベント

非同期イベントの概念図を図 4.3 に示す．非同期イベントが発生すると，そのイベントはカーネルのイベントキューに挿入され，後ほどイベントとして受け取られる．非同期イベントはポストされてから受諾されるまでの間，Contiki カーネル内のイベントキューに保持される．イベントキュー内のイベントはカーネルによってイベント受信プロセスに送られ，カーネルはイベントキュー内を確認し，イベントを呼び出しているプロセスへとイベントを届ける．Contiki では特に接頭語を付けずにイベントと表現される場合，そのイベントは非同期イベントを指す．また，イベントループ実行中に処理されるイベントはこの非同期イベントである．

非同期イベントの受信プロセスはある特定のプロセスである場合と，すべての実行中のプロセスである場合がある．イベントの受信者がある特定のプロセスだった場合，カーネルはイベントを届けるためにこのプロセスを呼び出す．イベントの受信者がシステム内のすべてのプロセスだった場合，カーネルは同じイベントをすべてのプロセスへと順次届けていく．

非同期イベントは *process_post* 関数によってポストされる．*process_post* 関数ははじめにキュー内にイベントのためのメモリ空間があるかどうかを確認するために，現在のイベントキューのサイズを調べる．もしメモリに空きがなければエラーを返し，メモリに空きがあった際にはイベントをキューの最後に挿入する．

同期イベント

図 4.4 は同期イベントにおける概念図である．非同期イベントに対して，同期イベントはポストされたとき，即座にイベントとして受諾される．同期イベントはある特定のプロセスにのみポストされる点も非同期イベントとは異なっている．同期イベントはすべてのタスクを初期化する際にのみカーネルによって呼び出され，イベントループ内で同期イベントが発行されることはない．

同期イベントは即座に実行されるため，同期イベントがポストされることは関数を呼び出すことに等しい．すなわち，同期イベントを届けられたプロセスは直接よびだされ，同期イベントをポス

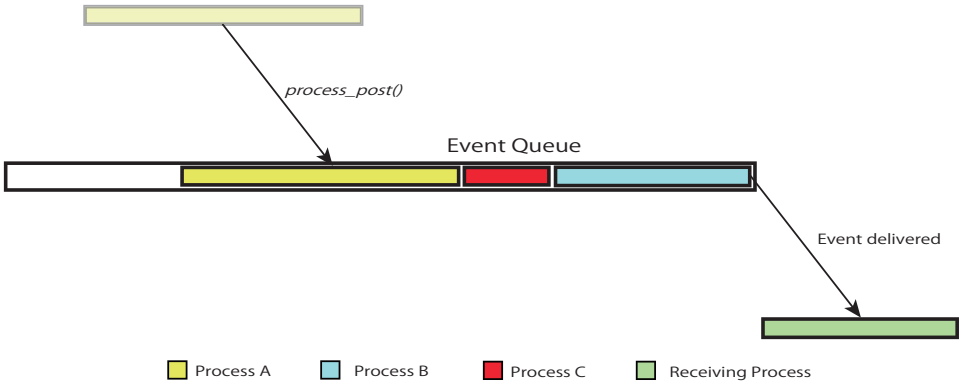

図 4.3　非同期イベントの実行

トしたプロセスは受信プロセスによる同期イベントの処理が終了するまで他の処理を行うことはできない．しかしながら，受信プロセスはイベントが同期的にポストされたか，非同期的にポストされたかを知らされない．

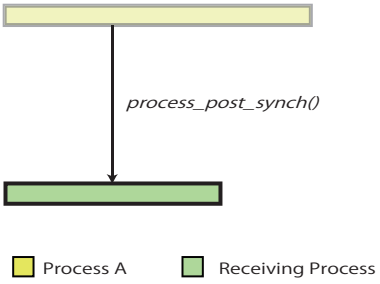

図 4.4　同期イベントの実行

ポーリング

ポーリングリクエストはその他のイベントと異なり，プロセスは *process_poll* 関数によって呼び出され，この関数がプロセス上で呼び出されるとそのプロセスは可能な限り早急にスケジューリングされる．ポーリングはインタラプトからプロセスを実行させる手法のひとつであり，*process_poll* 関数はプリエンプティブモードから安全に呼び出される，プロセスモジュールにおける唯一の関数である．

4.1.3　タスクの切り替え

一般的なスレッドモデルのオペレーティングシステムは，タスクの割り込みがあった際には，レジスタの状態を保存することで変数の値を保持し，割り込まれたタスクの実行が終了したときに再びレジスタから変数を読み込み，処理を途中から再開する．しかしながら Protothreads を使用

した場合，状態をレジスタに保存することはせずに，戻り値を利用することでタスクの CPU の利用を放棄する．コンテキストスイッチごとにレジスタ保持の処理を行わないため，エネルギーの節約を実現している．戻り値をもとにしたタスクの切り替えを行う際には，各タスクがプログラムのファイルの行数を保持し，スケジューラに対してそれぞれのタスクの状態を return することで，各タスクが CPU 利用権限を再度取得したとき，行数を条件分岐し，前回中断した箇所から実行を再開することとなる（Listing4.1）．アプリケーション内の変数は静的な変数を採用しているため，電力の消費を抑えつつ，一貫性を保っているが，現在実行中のタスクよりも優先度の高いタスクが実行待ちになった際に，他のマルチスレッドモデルのオペレーティングシステムで実現されているように，ループ内の処理を実行しているタスクに割り込みをし，タスクを切り替えて実行することはできない．これはタスクが return を発行するまでスケジューラが実行されないためである．

また現在では，タスクを切り換える際にタイマー割り込みを目的とした利用をされていない．タイマーと現在時刻を比較し，発火時刻を過ぎている場合，タスクを実行待ちのキューへと加える．タイマーを実行するタスクは他のタスクと同じ優先度で周期的に実行される．

4.2 既存システムにおける問題意識

3.4 で言及したように，イベントモデルではタスクの中断をしないことを前提に設計されているため，現在のセンサネットワーク用のオペレーティングシステムでは省資源性で低消費電力であることと，リアルタイム性のサポートはトレードオフの関係にある．しかしながら，環境モニタリングやターゲットトラッキングなどの状況においては，通常時は省資源性で低消費電力を実現し，イベントが生じた際にはリアルタイム処理を行うことが必要となってくる．それにも関わらず，現在双方を両立可能としたセンサネットワーク用のオペレーティングシステムは存在していない．

また 4.1 から，Protothreads がスレッドモデルのオペレーティングシステムとの親和性があると推測できるが，既存の Protothreads の実装ではイベントに優先度をつけることなく，First In First Out（FIFO）の要領でイベントを呼び出している（図 4.3）．しかし，環境モニタリングやターゲットトラッキングを想定環境とした場合，イベントの到着順でスケジューリングすることは好ましくない．例えば，優先度の高いタスクが到着したときに既に多量のタスクが実行待ち状態となっている場合に，高優先度のタスクが実行される頃には，既にそのタスクに実行する価値はない可能性がある．

さらに 4.1.3 にて，Protothreads では現在実行中のタスクよりも優先度の高いタスクが実行待ちになった際に，他のマルチスレッドモデルのオペレーティングシステムで実現されているように，ループ内の処理を実行しているタスクに割り込みをし，タスクを切り替えて実行することはできないことについて述べた．これは実行中のタスクの CPU の利用が放棄されるまで，リアルタイム処理の必要なタスクの実行が開始されないことを示しており，タスクの実行が長期にわたって行われる場合，リアルタイムタスクの実行が遅れる原因となり得る．

4.3 本研究の目的

本研究では 4.2 に述べたことを踏まえ，環境モニタリングやターゲットトラッキングのような時間的制約を伴ったタスクの処理が必要なアプリケーションを想定した際に，適切なタスクを省エネルギーで選択，実行することを可能にしたシステムの構築を目指す．

4.4 まとめ

本章では，まず，主流となるイベントモデルのオペレーティングシステムである Contiki に実装されている Protothreads を紹介し，メモリ，イベント，タスクの切り替えというそれぞれの観点から考察を行った．次に，既存の無線センサネットワークにおけるオペレーティングシステムの各モデルについて問題点を挙げ，続いて Protothreads を使用する際に注意すべきことについて言及し，最後に本研究の目的について記した．

第 5 章

設計と実装

本章では，本システムにおける機能要件，概要，設計，さらに実装について説明する．

5.1 設計

5.1.1 機能要件

ここでは本システムの設計にあたって，4.3 で述べた本研究の目的を達成するために必要となる機能について言及する．

1. リアルタイム性
   時間的制約を伴ったタスクの処理が必要なアプリケーションを想定した場合，オペレーティングシステムとしてリアルタイム処理を行えるものを選択すべきであり，無線センサネットワークを用いた環境モニタリングでは，イベント発生に応じてタスクを優先度を変化させる必要がある．また，ターゲットトラッキングにおいても同様に，対象の検知は適時にゲートウェイまで届けられなければならない．

2. 低消費電力
   森林火災検知システムにおけるネットワークライフタイムは，火災シーズンを上回ることが望ましく，野生動物の生態調査を行う際にも，現地調査を複数回行うことが困難となる場合もあるため，資源の限られたセンサノードにおいて低消費電力を維持することは極めて重要である．監視システムアプリケーションを用いた任務でも，任務期間中に資源の制限されたセンサデバイスの手動による充電はできないことがほとんどであることから，同様の機能が必要であることが考えられる．

5.1.2 システム概要

前述のとおり，Contiki[1] はイベントモデルであるため，電力消費を低く維持することが可能である．また 4.1.3 で示したように，一般的なスレッドモデルではレジスタの状態を保存しておくことによりタスクの切り替えを実現していることに対して，Contiki における Protothreads[2] は，戻り値を利用することでタスクの CPU の利用を放棄しつつ，一貫性を保つことが可能であるため，他のスレッドモデルのオペレーティングシステムと比べて，エネルギーの消費を軽減できる設計となっている．しかしながら，Protothreads を使用した場合，タスクの実行を中断させるプリエンプションができないため，本システムでは適時処理が必要なイベントが発生した際には，コンテキストを保存し，スレッドを切り替えることができるような設計が必要となる．

本システムの全体像を図 5.1 に示す．本システムは，ひとつのイベントループと多数のイベントハンドラから構成される．イベントループはイベントの到着を待ち，イベントが届くとイベントに関連付けられているイベントハンドラを実行する．しかしながら，通常のイベントモデルのオペレーティングシステムと異なり，本システムではリアルタイム処理の必要なイベントが発生する場合を想定している．イベントループ実行中に，リアルタイムイベントが発生した場合には，現在実

行中のタスクをプリエンプションし，スレッドを切り替え，リアルタイムタスク実行後再びイベントループの実行を再開する．

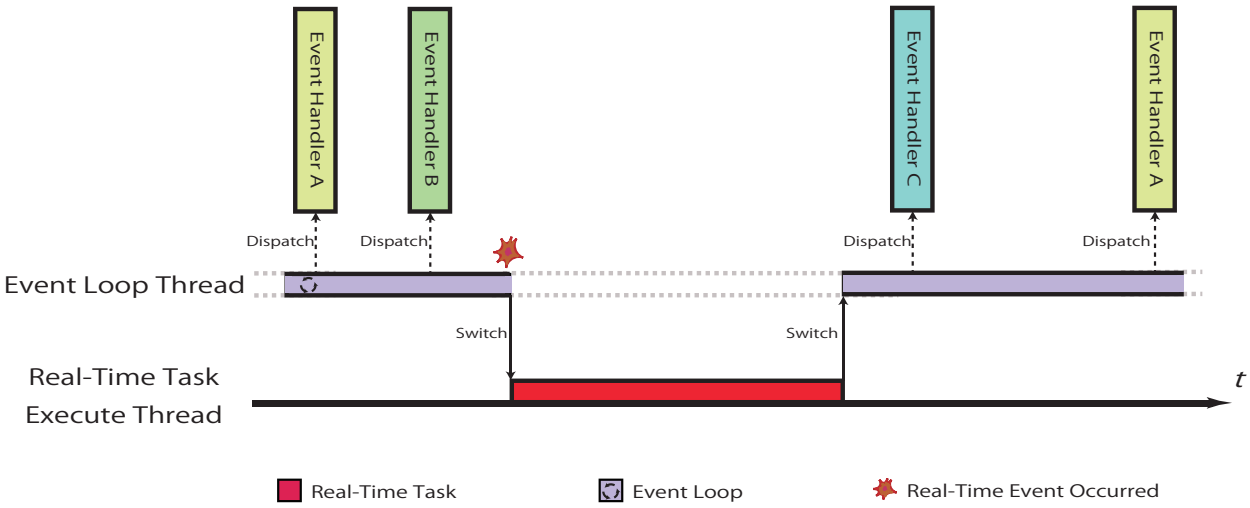

図 5.1　システム概要

5.1.3　システム構成

本節では，本システムのアーキテクチャ（図 5.2）を示し，各モジュールについて詳しく説明する．

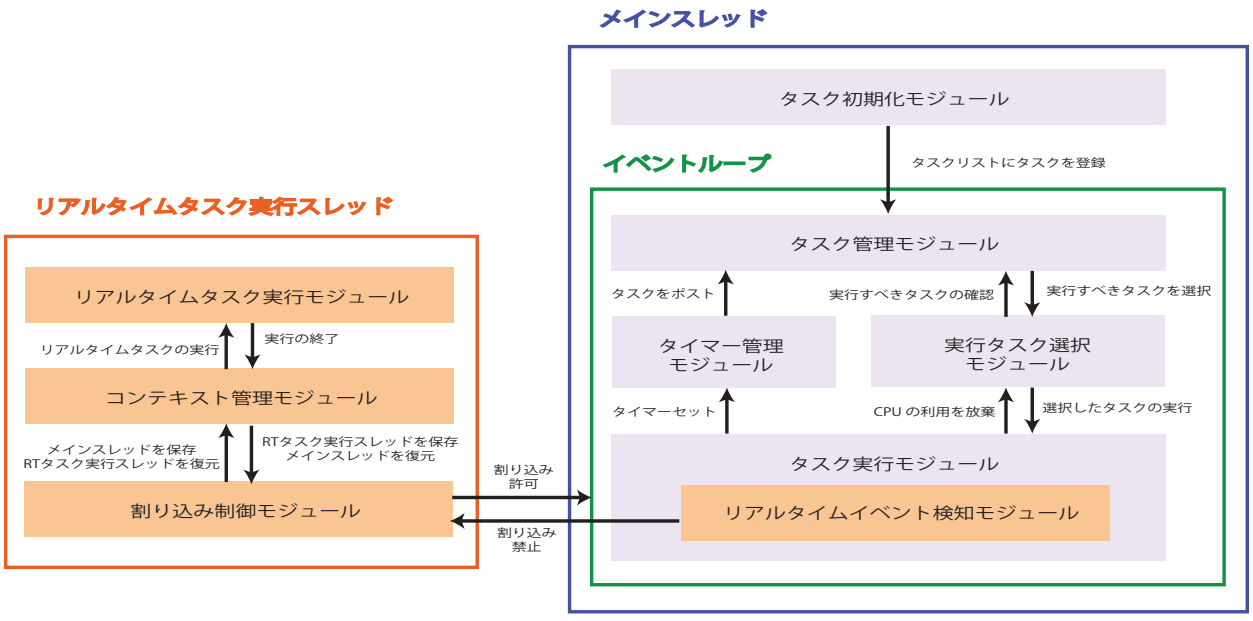

図 5.2　システムアーキテクチャ

タスク初期化モジュール

タスク初期化モジュールは，アプリケーション開発者によって初期化されたタスクをカーネルが保持するタスクリストに登録する．リアルタイムタスクを初期化する際には，そのタスクのコンテキストを保存するためのスタックも同時に初期化しておく．本モジュールはアプリケーションがデプロイされたときにのみ実行され，イベントループ内で実行されるモジュールと異なり，複数回実行されることはない．

タスク管理モジュール

ここではスケジューリングアルゴリズムに従って，タイマー管理モジュールによってポストされるタスクを実行される順番に並び替える．実行タスク選択モジュールからの問い合わせがあった際には，スケジューリング後の最も優先度の高いタスクのアドレスを渡す．

実行タスク選択モジュール

タスク実行モジュールにおいて，実行中のタスクが CPU の利用を放棄した場合，実行タスク選択モジュールにて，次に実行するタスクを選択するために，タスク管理モジュールに実行が最優先とされるタスクについて尋ねる．タスク管理モジュールからの応答を待ち，受け取ったタスクを次に実行するべく，タスク実行モジュールへと引き渡す．

タスク実行モジュール

タスク実行モジュールでは，タスク選択モジュールにて選択されたタスクを実行する．実行中のタスクが CPU の利用を放棄した場合には，次に実行するタイミングを決めるために，タイマー管理モジュールに次の実行時間を知らせ，タスク選択モジュールによって選ばれた次に優先度の高いタスクを実行する．

タイマー管理モジュール

実行中のタスクが自ら実行を中断した場合，タスク実行モジュールにて次に実行される時間が指定される．タイマー管理モジュールでは，タスク実行モジュールによってセットされたタイマーが発火した際には，それに対応するタスクをポストするため，タスク管理モジュールにタスクのポストを知らせる．

リアルタイムイベント検知モジュール

本システムの想定環境である，環境モニタリングやターゲットトラッキングのようなアプリケーションでは，イベントが発生した際に，他のどのタスクよりも優先してそのタスクの実行を行う必要がある．本モジュールでは，対象とするイベントが発生した場合に，イベントループに割り込み

をし，他のスレッドにスイッチした際に割り込みが発生しないよう，割り込み制御モジュールにて割り込みを禁止するシグナルの送信を依頼する．

割り込み制御モジュール

割り込み制御モジュールでは，リアルタイムタスク実行中に他のタスクによる割り込み処理が発生しないように制御を行う．リアルタイム処理が終了した際には，イベントループ実行中のメインスレッドにタスクの完了を知らせ，割り込みを許可する．

コンテキスト管理モジュール

タスクの実行中に割り込みが発生した場合，割り込みが発生しなかったときと同様の処理が行われなければならないため，レジスタの値をスタックに退避させる．それに対して，割り込み発生により実行されるタスクは，以前の状態を取り戻すべく，スタックからコンテキストを復元する．リアルタイムタスクの処理が完了し次第，メインスレッドのコンテキストを復元し，割り込み制御モジュールに割り込みを許可するように求める．

リアルタイムタスク実行モジュール

本モジュールは，コンテキスト管理モジュールでの中断されたタスクの復元が完了し次第，リアルタイム処理の必要なタスクの実行を行う．タスクの実行が完了すると，その旨をコンテキスト管理モジュールに伝え，割り込み発生前の状態を復元する．

5.1.4 リミテーション

本研究では，リアルタイム処理実行中に割り込みをし，新しいリアルタイムタスクの実行をする多重割り込みをサポートしていない．また，多数のリアルタイムイベントが頻繁に発生するような環境も今回は想定しない．

5.2 実装

5.2.1 実装環境

本システムは MicaZ 上で実装した．表 5.1 に MicaZ 及びアプリケーション作成に用いたコンパイラの詳細を示す．開発する際に使用した言語は C 言語である．

5.2.2 メモリ

4.1.1 で一般的なスレッドモデルにおけるスタックと Protothreads におけるスタックの違いについて述べたが，ここでは本システムにおけるスタックについて言及する．本システムでは，通常

表 5.1　実装環境

項目	環境
CPU	Atmega128L
Clock	7.37 MHz
メモリ	4 KB
コンパイラ	avr-gcc 4.5.3
無線チップ	CC2420
オペレーティングシステム	Contiki 2.6
プラットフォーム	MicaZ

のタスクに関しては Protothreads を用いて省電力なタスクの切り替えを実現し，リアルタイムタスクは割り込みを発生させタスクを切り替えることを 5.1.2 にて述べた．したがって，本システムにおけるスタックは図 5.3 のようになり，一般的なスレッドモデルのオペレーティングシステムよりもメモリを節約することに成功している．

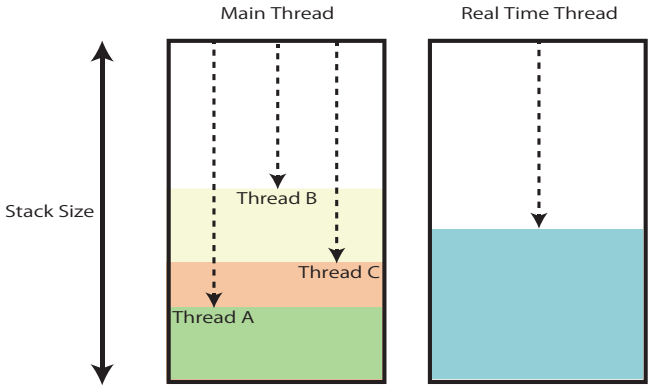

図 5.3　D-Switch におけるスタック

5.2.3　タスクの状態遷移

本システムで実行されるタスクには図 5.4 のような 4 種類の状態があり，関数呼び出しによって，それぞれのタスクの状態は遷移する．ここではタスクの状態の変遷について説明する．

アプリケーションがシステムにデプロイされると，*process_start* 関数が実行され，状態は PROCESS_STATE_NONE から，PROCESS_STATE_RUNNING へと変化する．すべてのタスクは PROCESS_STATE_NONE 状態から PROCESS_STATE_RUNNING へと切り替えられることで，実行される準備が整えられる．つまり，カーネルの保持するタスクリストに登録されているタスクはすべて，PROCESS_STATE_RUNNING 状態で実行までの間待機することとなる．

タスクが実行される際には，*call_process* 関数により，スケジューリングされたタスクリス

トの中から最も優先度の高いタスクが実行され，PROCESS_STATE_RUNNING から PROCESS_STATE_CALLED となる．PROCESS_STATE_CALLED 状態のタスクは実行を終えると自ら CPU の利用を放棄するため，PROCESS_STATE_RUNNING 状態に戻り，次に優先度の高いタスクの状態を PROCESS_STATE_CALLED へと変え，そのタスクの実行を行う．

タスクが実行されている最中に，リアルタイムタスクによる割り込みが発生した場合，*process_suspend* が実行され，状態は PROCESS_STATE_SUSPENDED へと遷移する．再びタスクが再開されるとき，*process_resume* により PROCESS_STATE_CALLED 状態に復帰し，中断した処理の続きを行う．

タスクが終了されるとき，*exit_process* 関数は対象とされるタスクの状態を PROCESS_STATE_NONE に変更し，状態の変わったタスクはタスクリストから消去される．

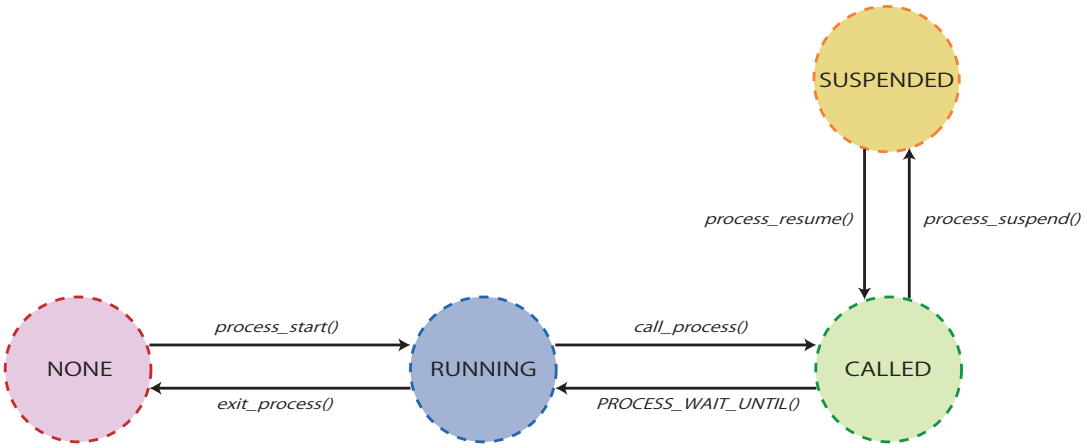

図 5.4　状態遷移図

5.3　まとめ

本章では，最初に本システムに必要な要件についてまとめ，システムの概要について述べた．そして，システムアーキテクチャを各モジュールに分解しながら説明し，システム構成について考察した．最後に実装環境など実装に関する内容について言及した．

第 6 章

評価

本章では本システムの評価を行い，結果について考察する．

6.1 評価方針

本節では，本システムが 5.1.1 で示した機能要件を満たしているかどうかを実験を行うことで評価する．具体的な方法について以下に述べる．

1. リアルタイム性
   本研究は環境モニタリングやターゲットトラッキングのような無線センサネットワークにおけるアプリケーションを対象としているため，イベントモデルである拡張する前の Contiki[1] と本システムを比較し，リアルタイムイベント発生から実行までの時間を評価する．

2. 消費電力
   本システムではリアルタイム処理が必要なイベントが発生しない場合には，イベントモデルを採用している．消費電力を抑えるという要件が満たされているかどうかを確認するために，ランダムにリアルタイムイベントを発生させ，イベント発生後現在の電圧をベースステーションへと送信するアプリケーションを，拡張前の Contiki と本システム上で実行し，それぞれの消費電力を比較し，考察する．

6.2 リアルタイム性における評価

6.2.1 導入

タスクの総数を一定にした場合とタスクの総数を変化させる場合についてそれぞれ，イベント発生の頻度を変化させながら，リアルタイムイベントが発生してから対象となるタスクが実行されるまでの時間（Latency）を評価する．イベント発生の頻度を変化させるために，Contiki におけるイベントタイマーによってセットされる値を変化させる．イベントタイマーは，タイマーと現在時刻を比較し，発火時刻をすぎている場合，タスクをキューへと加える．また，システムの安定性を評価するために，累積確率密度関数（Cumulative Distribution Function）を評価に取り入れ，その結果について 6.2.4 にて考察する．

6.2.2 評価環境

本評価では Contiki のために設計された仮想マシン上のシミュレータ Cooja[28] を利用する．仮想マシンの詳細を表 6.1 に記す．また 5.2.1 で示したように，本実験では対象とするノードとして MicaZ[7] を利用する．

表 6.1　評価環境

項目	環境
オペレーティングシステム	Ubuntu 12.04
メモリ	1024 MB

6.2.3　実験結果

タスクの総数を一定にした場合とタスクの総数を変化させる場合についてそれぞれ，実験によって得られた結果を示す．以降より，Contiki は拡張前の Contiki を，D-Switch は拡張された後の Contiki を示すこととする．

タスクの総数を固定した場合

タスクの総数を固定し，実験ごとにイベントの発生頻度を増やしながら 300 秒ずつ，3 回行った．

1. 実験 1
   表 6.2，図 6.1 からわかるように，D-Switch と Contiki を比較してみると，最小値，中央値の値に差異は見られない．しかしながら，Contiki における Latency の最大値が 163ms を記録することが数回あったため，平均値，標準偏差ともに D-Switch を上回る結果となった．また図 6.2 より，Contiki では最小値を記録した回数が D-Switch よりも多いが，D-Switch では Latency が 81ms 以下となる場合が 98% を占めているため，D-Switch の方が安定して低い Latency を維持することに成功している．

2. 実験 2
   表 6.3，図 6.3 より実験 1 と同様に Latency を比較したところ，最小値，中央値の値に差異はなく，最大値は実験 1 で得られた値に等しい．しかしながら，実験 1 と比較して，Contiki における Latency が最大となる場合が多く，平均値，標準偏差の値が増加した．図 6.4 により得られた累積密度確率関数の結果も図 6.2 と類似している．

3. 実験 3
   表 6.4, 図 6.5 が示すとおり，最小値は Contiki が D-Switch を下回ったが，最大値は Contiki が D-Switch を上回る 245ms を記録しているため，Contiki における Latency の平均値，標準偏差がそれぞれ D-Switch を上回る結果となった．さらに図 6.6 より，D-Switch では 94% が 81ms の Latency でリアルタイムタスクの実行を行うことができており，安定性において D-Switch の優位性を示している．

表 6.2　実験結果 1

	最小値（ms）	最大値（ms）	中央値（ms）	平均値（ms）	標準偏差
D-Switch	80	82	81	81	0.17
Contiki	80	163	81	97.97	33.39

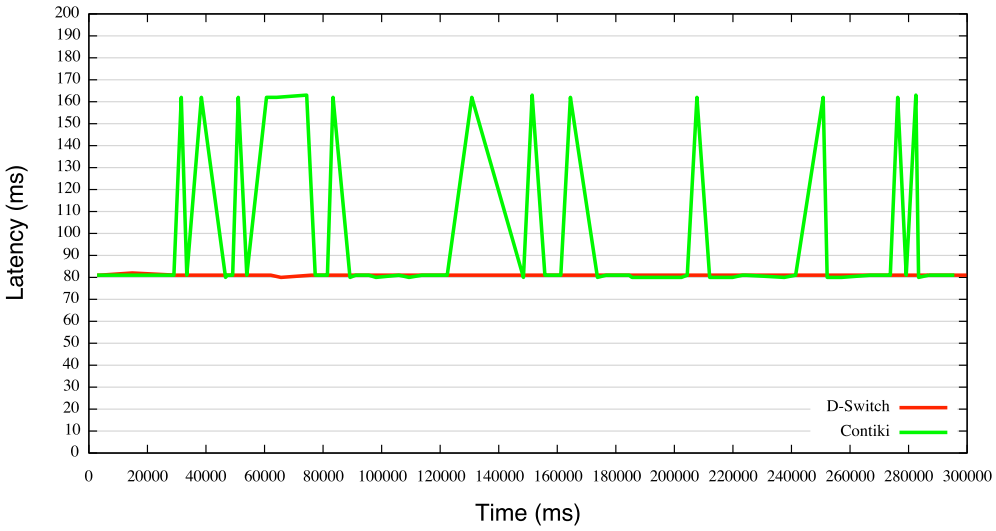

図 6.1　Latency1

表 6.3　実験結果 2

	最小値（ms）	最大値（ms）	中央値（ms）	平均値（ms）	標準偏差
D-Switch	80	82	81	81.02	0.19
Contiki	80	163	81	108.37	38.49

表 6.4　実験結果 3

	最小値（ms）	最大値（ms）	中央値（ms）	平均値（ms）	標準偏差
D-Switch	81	82	81	81.02	0.24
Contiki	80	245	81	98.36	49.61

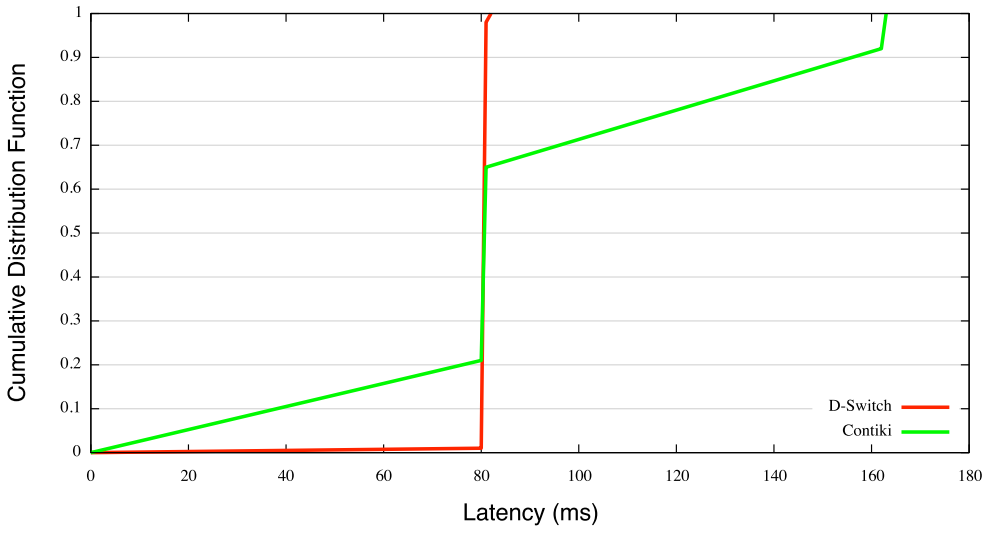

図 6.2　累積確率密度関数 1

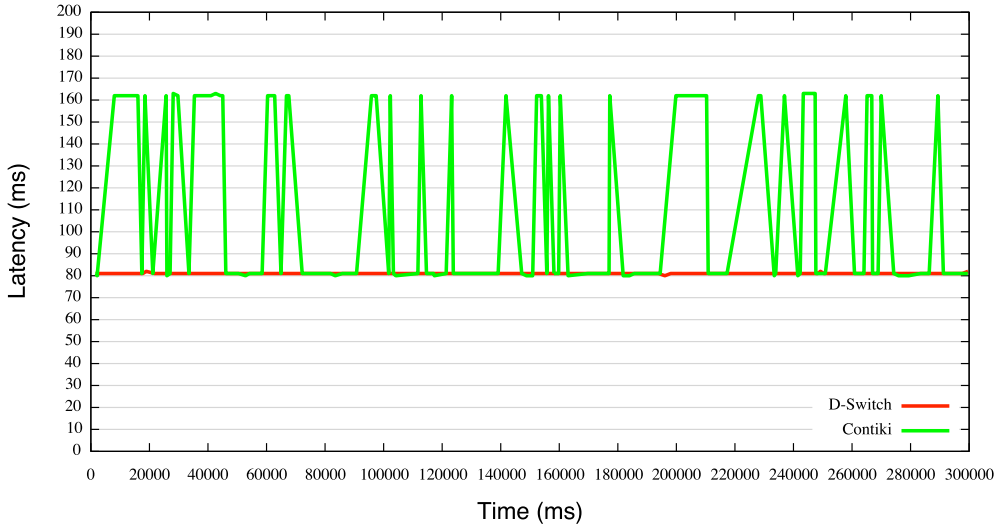

図 6.3　Latency2

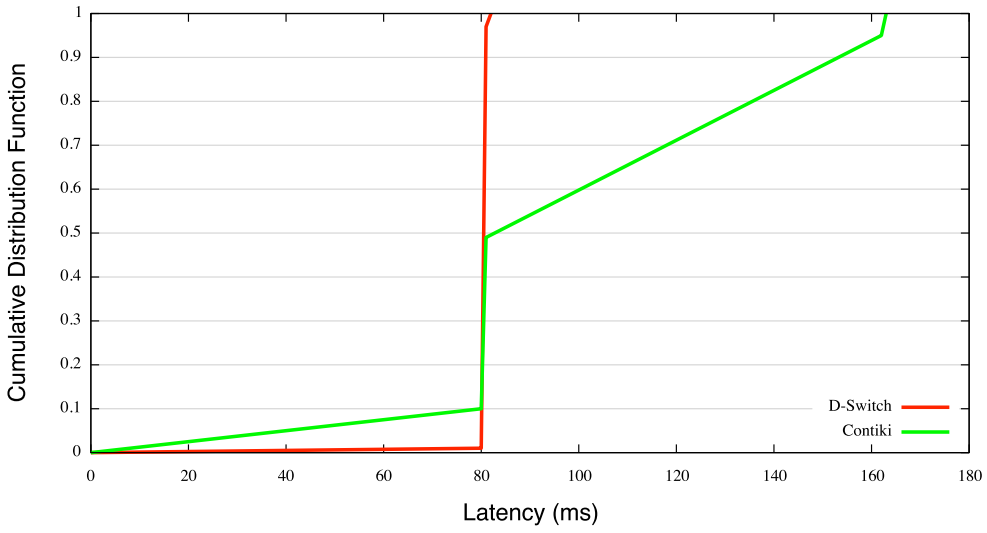

図 6.4　累積確率密度関数 2

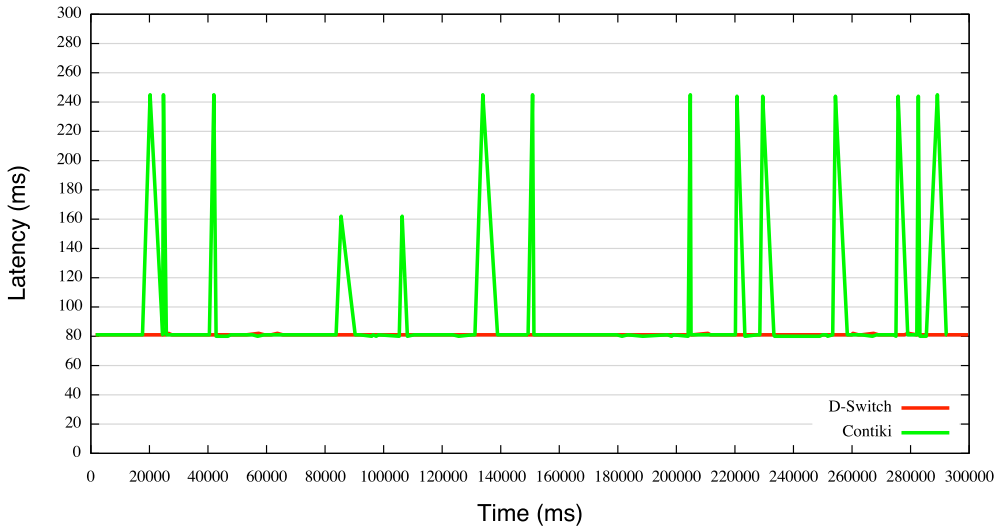

図 6.5　Latency3

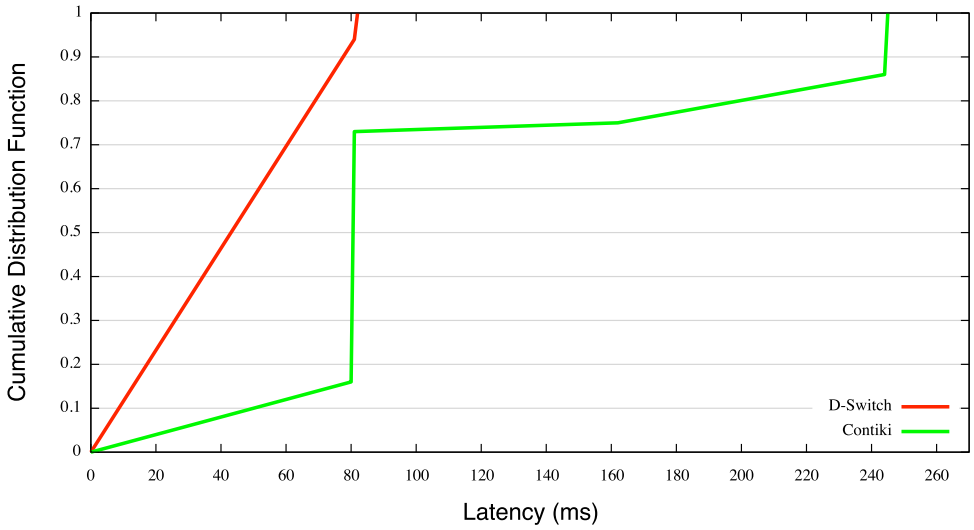

図 6.6　累積確率密度関数 3

タスクの総数を変化させた場合

タスクの総数を増やしながら実験を 300 秒ずつ，2 回行った．

1. 実験 4

   表 6.5，図 6.7 が示すとおり，Contiki では Latency の最小値が 80ms を記録し，両者における最小値となったが，D-Switch では観測されたすべての値が 81ms を記録しており，標準偏差 0 を示したことや，100% が 81ms の Latency で実行されていること（図 6.8）からも，本システムの安定性を裏付ける結果となった．

2. 実験 5

   表 6.6，図 6.9 より，Contiki における最大値，平均値，標準偏差は全実験中最大の値となった．それに対して，D-Switch は図 6.10 にも示されるように，97% が 81ms 以内にリアルタイムタスクの実行を許可されている．

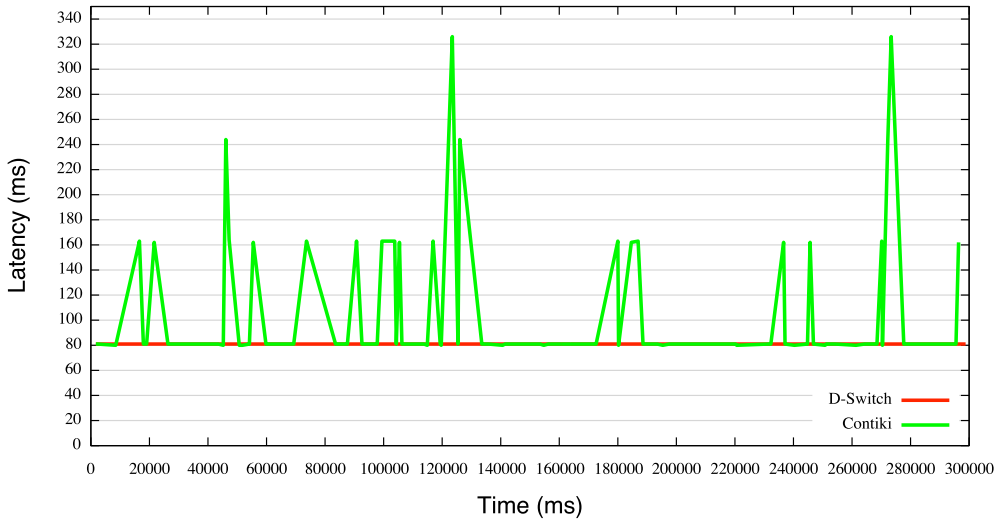

図 6.7　Latency4

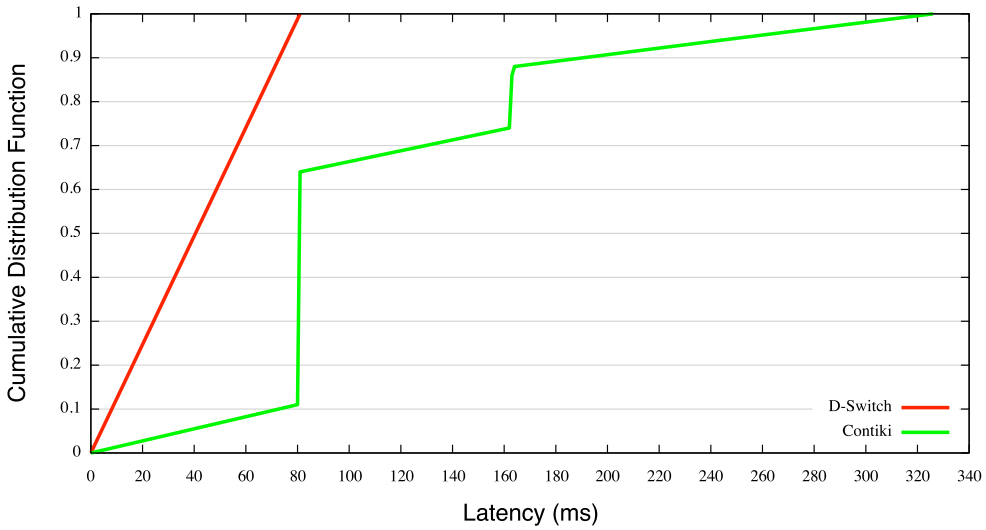

図 6.8　累積確率密度関数 4

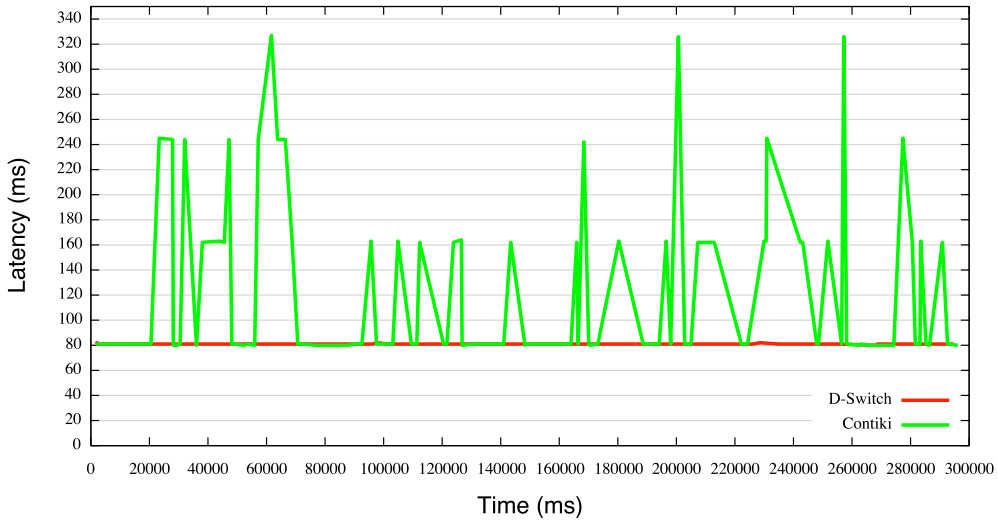

図 6.9　Latency5

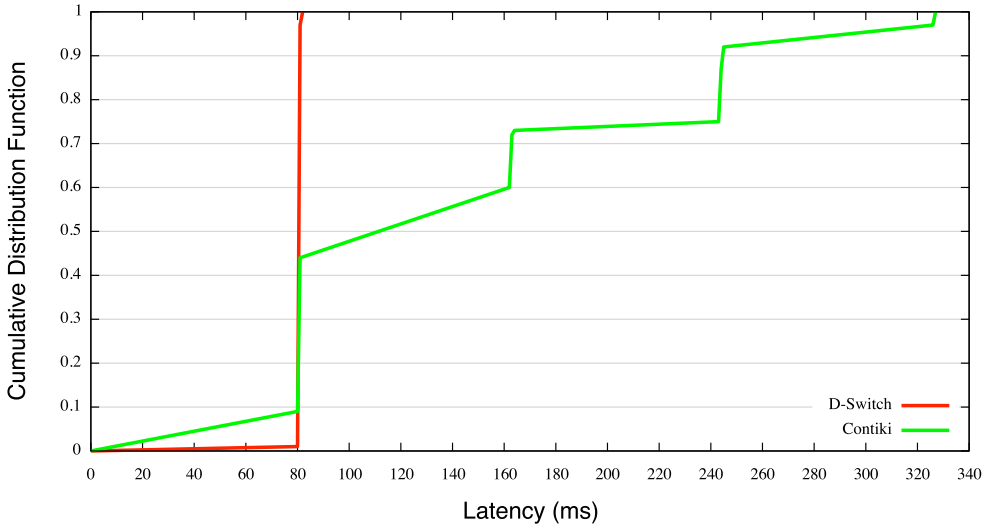

図 6.10　累積確率密度関数 5

表 6.5　実験結果 4

	最小値（ms）	最大値（ms）	中央値（ms）	平均値（ms）	標準偏差
D-Switch	81	81	81	81	0
Contiki	80	326	81	102	48.52

表 6.6　実験結果 5

	最小値（ms）	最大値（ms）	中央値（ms）	平均値（ms）	標準偏差
D-Switch	80	82	81	81.02	0.21
Contiki	80	327	81	121.26	63.95

6.2.4　考察

図 6.11，図 6.12 はそれぞれ，D-Switch，Contiki の Latency に関するすべての実験結果を，図 6.13，図 6.14 はそれぞれ，D-Switch，Contiki の累積確率密度関数に関するすべての実験結果を示している．実験 1 から実験 3 にてわかるようにイベントの発生頻度が増加した場合，Contiki では Latency が増加することが伺える．同様に実験 4 から実験 5 のようなタスクの総数が増加した場合にも，Contiki における Latency は増加する傾向にある．

4.1.2，4.2 で示したように，Contiki では発生したイベントは FIFO の要領でポストされていくため，イベントが発生された際にキューにすでに実行待ちのイベントが挿入されていた場合，実行される時間が遅くなってしまう．それに対して，D-Switch では，リアルタイムイベントが発行された際には，実行中のタスクがあった場合でも，そのタスクの実行を一時中断して，リアルタイムタスクの実行を行うことができるため，Latency を低く維持することを可能にしている．

D-Switch における Latency が 80ms に収束しているのは，イベント発生から現在のコンテキストを保存するまでの時間だと推測される．それに対して，Contiki には実行中のタスクをプリエンプションしてリアルタイムタスクを行うことができないため，リアルタイムイベントが発生した場合でも，実行中のタスクの終了を待たなければならない．Contiki における Latency の最小値が 80ms となっているのは，リアルタイムタスクが発生してから，前のタスクの実行が終了するまでの時間であると考えられる．

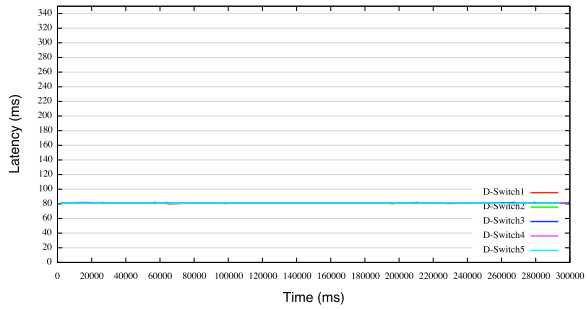

図 6.11　D-Switch Latency

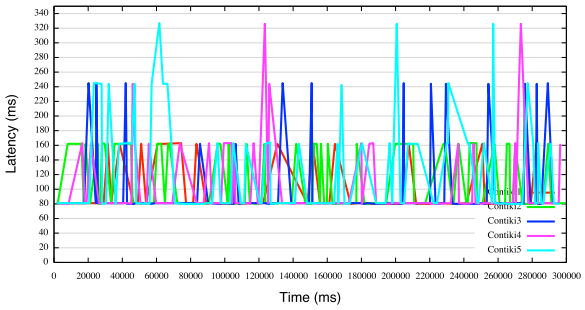

図 6.12　Contiki Latency

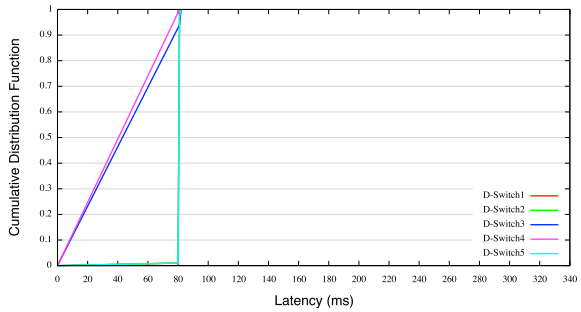

図 6.13　D-Switch 累積確率密度関数

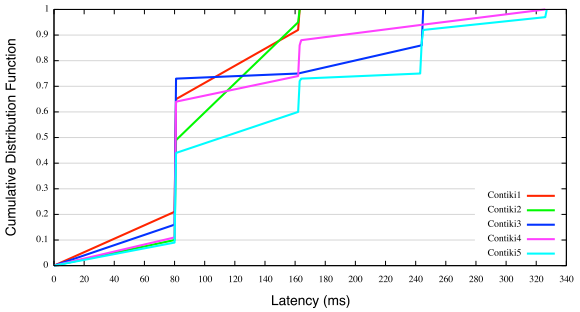

図 6.14　Contiki 累積確率密度関数

6.3　消費電力における評価

6.3.1　評価環境

実機の MicaZ を使用して，ランダムにリアルタイムイベントとして，ベースステーションへと現在の電圧を送信するアプリケーションをデプロイし，実験を行った．その際，Contiki と D-Switch とで比較を行い，それぞれのシステムでタスクの総数を変化させる．

6.3.2　実験結果

図 6.15 は消費電力に関する実験結果を表している．本システムにおける電力消費は拡張前の Contiki とほとんど差分がないことがわかる．

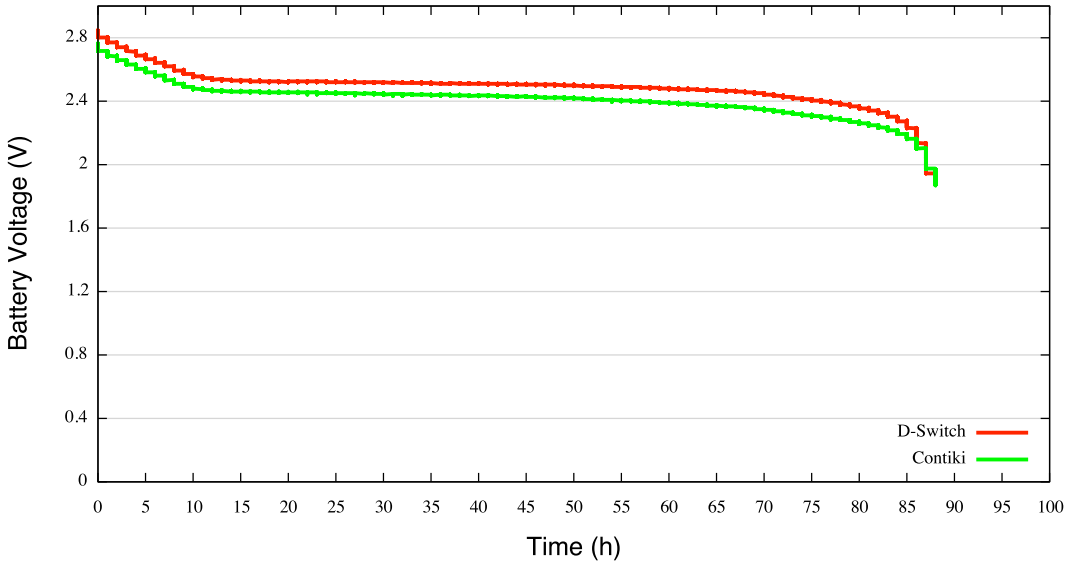

図 6.15　消費電力

6.3.3　考察

ここでは，6.3.2 にて得られた結果について考察する．無線センサノードにおいては無線が最も電力を消費するモジュールのひとつであり，消費電力を抑えるためには単位時間あたりの無線利用を抑える必要がある．無線利用時間を低減するためのアプローチとして，無線センサネットワークに向けた，様々な MAC プロトコルが提案されており，無線センサネットワーク向け MAC プロトコルにおいて，定期的に無線を起動し，受信すべきデータが発信されているかを確認することで，無線利用時間を削減する手法を用いることが多い [29]．6.2.3 に示したように本システムでは，リアルタイムタスクの実行には約 80 ms もの時間を要してしまってはいるものの，リアルタイムイベント発行から実行までの時間はほぼ一定であるといえる．したがって，本システムを用いることで，効率的に無線資源を利用し，無線の起動を定期的に行うことが可能となるため，それに伴って無線の起動時間を短縮することも可能となる．以上より，コンテキストスイッチングによる電力消費は大きいものの，周辺デバイスのオンとオフを素早く切り替えることができるようになった結果，電力の消費を抑えることができ，拡張前の Contiki と同等の省エネルギー性を実現できたと考えられる．

また Nano-RK[5] のような，一般的なスレッドモデルのオペレーティングシステムはそれぞれの

タスクに優先度が割り振られており，実行中のタスクよりも優先度の高いタスクが実行待ちになった場合，プリエンプションを行い，コンテキストスイッチングしていくこととなる．ゆえに優先度の高いタスクが実行待ちになる度にコンテキストを切り替えなければならないため，3.4 で述べたように，イベントモデルに比べて電力の消費が激しい．しかしながら，本システムはすべてのタスクにおいてスレッドモデルのような挙動を行うのではなく，リアルタイムタスク以外の通常のタスクはイベントモデルのように扱われる．したがって，プリエンプション及び，コンテキストの切り替え回数を抑えることができたため，一般的なスレッドモデルで消費される電力よりも電力の消費を抑えることができ，このような結果に至ったと推測できる．

6.4 まとめ

本章では，まず，本研究の評価方針について述べ，次に，リアルタイム性における評価に関する評価環境を記し，実験結果について考察を行った．次いで，消費電力における評価に関しても同様に評価環境について述べ，実験結果に基づく考察を行い，拡張する前の Contiki における電力消費率にを大幅に変化を加えることなく，82ms 以内のタスクの実行を可能にしたことを示した．

第 7 章

結論

本章では，本システムの結論を述べる．

7.1 今後の課題と展望

無線センサネットワークは複数の無線センサノードによって構成される分散システムであるため，本研究で想定しているような環境モニタリングやターゲットトラッキングなどの時間的整合性が要求されるアプリケーションの構築には，何らかの手段によって時刻同期を行うことが求められている．代表的なオペレーティングシステムである TinyOS[3][4] における Flooding Time Synchronization Protocol（FTSP）[30] と呼ばれる高精度な時刻同期プロトコルでは，平均約 1μs の時刻同期の実現に成功している．しかしながら，Contiki[1] が独自に採用している時刻同期プロトコルの精度は FTSP と比較してかなり劣っているのが現状である．また，通常時は無線を休止し，無線の起動を一定に保つことに加えて，受信すべきデータがネットワークに存在しなければすぐに無線を休止状態に戻す MAC プロトコルが提案されているが，高精度な時刻同期をすることにより無線の起動時間を短くすることができるため，消費電力を抑えることも可能となる．したがって今後の展望として，Contiki における高精度な時刻同期プロトコルを実装することで，より厳密なリアルタイム処理ができるようにすることを目指す．

また，Contiki は MicaZ[7] 以外にも，Iris Mote[6] や Tmote Sky[31] などの多様なプラットフォームへの対応が確認されているが，現状として本システムは今回実装として採用した MicaZ 上でしか動作を確認することができていない．CPU のアーキテクチャが異なるプラットフォームに移植した際に，正常に動作できるようにすることも今後の課題として挙げる．

7.2 本論文のまとめ

本論文では，時間的制約の伴った環境モニタリングやターゲットトラッキング時における，省エネルギーなリアルタイム処理の必要性について述べ，既存のオペレーティングシステムを採用した際の問題点について言及した．次いで，無線センサネットワークにおけるオペレーティングシステムに対する動的なスケジューリングスイッチング機構である D-Switch を提案し，通常時はイベントモデルとしてイベントに応じた処理を行うとともに，リアルタイムイベント発生時には現在のタスクを中断し，リアルタイムタスクの実行が完了し次第，中断したタスクの処理を再開するシステムを設計し，評価を行った．実験の結果，本システムを拡張する前の Contiki における電力消費を大幅に増加させることなく，82ms 以内のタスクの実行を可能にした．

謝辞

本論文の執筆にあたり，親身になって丁寧にご指導して頂きました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します．また，貴重なご助言を頂きました慶應義塾大学環境情報学部准教授高汐一紀博士，慶應義塾大学環境情報学部准教授中澤仁博士，慶應義塾大学環境情報学部米澤拓郎特任助教，慶應義塾大学環境情報学部陳寅特任助教，慶應義塾大学政策・メディア研究科研究員伊藤友隆氏，慶應義塾大学政策・メディア研究科博士課程小川正幹氏に深く感謝致します．

慶應義塾大学徳田研究室の諸先輩方には折に触れ貴重なご助言を頂き，また多くの議論の時間を割いて頂きました．特に，政策・メディア研究科修士課程伊藤瑛氏，安形憲一氏には，本研究に対し，多くの時間を割いて頂きました．ここに多大なる感謝と尊敬の意を表します．

また，MEMSYS，Link 研究グループにおいて，研究活動だけではなく，公私に関わらず，親しく接していただいただいた江頭和輝氏，早川侑太朗氏，宮川眞海子氏，佐田真穂氏，井上美菜子氏，寺山淳基氏，皆川昇子氏，荻野メリッサ氏，池田貴匡氏，神崎亜実氏，水谷正芳氏を始めとした諸先輩，後輩方，毎週のように食事をともにした榊原寛氏，日吉ハンドボール同好会に所属する皆様，立川高校ハンドボール部 OB の皆様に深く感謝致します．

最後に，数少ない同学年として，研究活動に切磋琢磨した古川侑紀氏，卒論提出が迫り，氷点下付近まで気温が下がっているにも関わらず，ユニークなギャグでより一層周りの気温を低下させ，さらなる精神修行を課してくれた宇佐美真之介氏，嫌いと言われつつも，残留生活においておそらく一番長く時間を共有し，数々の罵詈雑言を浴びせるという精神修行を課してくれた高木慎介氏，毎週のように楽しみにしていた週刊少年ジャンプのネタバレをいつも楽しそうに語り，生きていく目標とは何かを考えさせてくれるような精神修行を課してくれた豊田智也氏に深く感謝し，謝辞と致します．

2015 年 2 月 10 日

小町 芳樹

参考文献

[1] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pp. 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[2] Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, SenSys '06, pp. 29–42, New York, NY, USA, 2006. ACM.

[3] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. *SIGPLAN Not.*, Vol. 35, No. 11, pp. 93–104, November 2000.

[4] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[5] Anand Eswaran, Anthony Rowe, and Raj Rajkumar. Nano-rk: An energy-aware resource-centric rtos for sensor networks. In *Proceedings of the 26th IEEE International Real-Time Systems Symposium*, RTSS '05, pp. 256–265, Washington, DC, USA, 2005. IEEE Computer Society.

[6] Crossbow Technology. `http://www.xbow.com/`.

[7] Jason L. Hill and David E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, Vol. 22, No. 6, pp. 12–24, November 2002.

[8] Sun SPOT World. `http://www.sunspotworld.com/`.

[9] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for “smart dust”. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, pp. 271–278, New York, NY, USA, 1999. ACM.

[10] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Comput. Netw.*, Vol. 52, No. 12, pp. 2292–2330, August 2008.

[11] David M. Doolin and Nicholas Sitar. Wireless sensors for wildfire monitoring. Vol. 5765, pp. 477–484, 2005.

[12] Carl Hartung, Richard Han, Carl Seielstad, and Saxon Holbrook. Firewxnet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In Per Gunningberg, Lars-ke Larzon, Mahadev Satyanarayanan, and Nigel Davies, editors, *MobiSys*, pp. 28–41. ACM, 2006.

[13] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, pp. 88–97, New York, NY, USA, 2002. ACM.

[14] Tian He, Sudha Krishnamurthy, John A. Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, and Lin Gu. Energy-efficient surveillance system using wireless sensor networks. In *In Mobisys*, pp. 270–283. ACM Press, 2004.

[15] L. Nazhandali, M. Minuth, and T. Austin. Sensebench: toward an accurate evaluation of sensor network processors. *2013 IEEE International Symposium on Workload Characterization (IISWC)*, Vol. 0, pp. 197–203, 2005.

[16] 猿渡, 鈴木, 大原, 南, 森川. 無線センサネットワークにおけるオペレーティングシステムの研究動向. In *Technical Report 2008003*. 森川研究室, 2009.

[17] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling ultra-low power wireless research. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[18] Lama Nachman, Ralph Kling, Robert Adler, Jonathan Huang, and Vincent Hummel. The intel®; mote platform: A bluetooth-based sensor network for industrial monitoring. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, IPSN '05, Piscataway, NJ, USA, 2005. IEEE Press.

[19] CAPSIL Wiki. `http://www.capsil.org/capsilwiki/index.php/TinyOS`.

[20] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pp. 1–11, New York, NY, USA, 2003. ACM.

[21] Adam Dunkels, Niclas Finne, Joakim Eriksson, and Thiemo Voigt. Run-time dynamic linking for reprogramming wireless sensor networks. In *ACM SenSys*, pp. 15–28. ACM Press, 2006.

[22] A. K. Dwivedi, M. K. Tiwari, and O. P. Vyas. Operating systems for tiny networked sensors: A survey. *IJRTE 2009*, pp. 152–157.

[23] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. Mantis os: An embedded multi-

threaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.*, Vol. 10, No. 4, pp. 563–579, August 2005.

[24] Eric Trumpler and Richard Han. A systematic framework for evolving tinyos. In *In proc. 3 rd Workshop on Embedded Networked Sensors*, pp. 61–65, 2006.

[25] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, Vol. 20, No. 1, pp. 46–61, January 1973.

[26] Anthony Rowe, Karthik Lakshmanan, Haifeng Zhu, and Ragunathan Rajkumar. Rate-harmonized scheduling for saving energy. In *Proceedings of the 2008 Real-Time Systems Symposium*, RTSS '08, pp. 113–122, Washington, DC, USA, 2008. IEEE Computer Society.

[27] Anthony Rowe, Rahul Mangharam, and Raj Rajkumar. Firefly: A time synchronized real-time sensor networking platform.

[28] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. Cross-level sensor network simulation with cooja. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pp. 641–648. IEEE, 2006.

[29] Joseph Polastre, Jason Hill, and David Culler. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 95–107. ACM, 2004.

[30] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys '04, pp. 39–49, New York, NY, USA, 2004. ACM.

[31] MEMSIC. Tmote sky, 2011. `http://www.snm.ethz.ch/Projects/TmoteSky`.