情報処理学会論文誌 Vol. 50 No. 1 14–30 (Jan. 2009)

無線センサネットワークにおける複数プログラムの動的配備

末永 俊一郎[†1,†2] 吉岡 信和[†1,†3] 本位田 真一[†3,†4]

無線センサネットワーク（Wireless Sensor Networks: WSN）は，センサデータを取得できる多数のノードが無線によって接続されるネットワークである．従来，WSNは特定のアプリケーションのために敷設されるのが一般的であったが，今後は，あらかじめ敷設されたWSNを利用して複数のアプリケーションを運用する形態が考えられる．敷設されたWSNで複数のアプリケーションを運用する場合，アプリケーションを構成するプログラムを動的にノードに配備する手法が有効であると既存研究により報告されている．ただし，既存研究は，WSN内部で複数のプログラムが連携するアプリケーションを構築する際のアーキテクチャを提案しておらず，アプリケーションを構成するプログラムの機能分担が課題となる．また，複数プログラムで構成されるアプリケーションが，稼動場所を変更する際，全プログラムを動的に配備できない可能性があり，アプリケーションを継続して実行できないという課題がある．本論文では，これらの課題を解決するために，コンポーネント化されたプログラムによって構成されるアーキテクチャと，1つのプログラムが他のプログラムを生成し，複数プログラムを動的に配備する手法を提案し，シミュレーション環境で提案手法の有効性を示した．

Dynamic Deployment for Programs in Wireless Sensor Networks

Shunichiro Suenaga,[†1,†2] Nobukazu Yoshioka[†1,†3] and Shinichi Honiden[†3,†4]

Wireless Sensor Networks (WSN) are wireless networks consisting of a high number of spatially distributed nodes that monitor physical conditions. Most WSNs so far have been designed with a very specific application in mind. However, it has become desirable and necessary these days to support the operation of multiple applications simultaneously in a single sensor network. One aspect of executing multiple applications in a WSN is to deploy them dynamically - an issue that has already been addressed in prior work. These existing solutions suffer from two problems however that keep them from being practical for deploying applications that consist of multiple distributed program components. For one, they lack a proper architecture to coordinate redeployment, and furthermore they offer no practical process to deploy multiple program components. We address these issues by proposing an architecture that gives program component deployment specific roles and by incorporating a generative approach for the dynamic deployment process of such components.

1. はじめに

現実世界の現象を把握する技術としてセンサがある．代表的なセンサに温度センサやRFIDがあり，センサの利用によって“火災”や“物の存在有無”などの現実世界の現象を把握することができる．現在，センサを無線によって接続する無線センサネットワーク（WSN: Wireless Sensor Networks）が着目されている．WSNはセンサデータを取得可能な複数のノードによって構成され，各ノードは，取得したセンサデータを無線で転送するマルチホップ・アドホックネットワークを構成できる．WSNの代表的なアプリケーションに，環境のモニタリング，対象物の検知と追跡，ヘルスモニタリング，構造物のモニタリングなどがある[1),2)]．

WSNのアプリケーションは，個々のノードにプログラムを配備し，ノードを物理的に配置することによって実現される．これまで，WSNは特定のアプリケーションのために敷設されるのが一般的であり，アプリケーションの運用前に個々のノードにプログラムを配備し，ノードを配置して，アプリケーションを運用することができた．しかし，今後のWSNの1つの運用形態として，WSNが事前に配備された環境下で，システム運用者の要求に応じて複数のアプリケーションを運用することが考えられる．この運用形態では，稼働中のアプリケーションを停止させずに，新規アプリケーションを構成するプログラムをノードに配備することや，配備したプログラムを変更することが要求される．また，WSNを構成する

†1 総合研究大学院大学
The Graduate University for Advanced Studies
†2 日本ユニシス株式会社
Nihon Unisys, Ltd.
†3 国立情報学研究所
National Institute of Informatics
†4 東京大学
The University of Tokyo

ノードはプログラムメモリが非常に限られていることから，複数のアプリケーションを構成するプログラムを全ノードに配備することができず，個々のノードに必要なプログラムだけを配備することが必要になる．

あらかじめ敷設されたWSNを利用して複数のアプリケーションを追加・変更しながら運用する際，ノードを回収してプログラムを追加・変更すると，アプリケーションの実行を妨げ，回収の手間も要する．そのため，無線を利用して特定のノードに配備されたプログラムを変更する手法としてリプログラミング[3]が提案されている．リプログラミングの代表例にDeluge[4]やAgilla[5]があり，ベースステーションから追加・変更されるプログラムを特定のノードに配備することができる．

本研究では，WSNの限られた範囲で計測を実施し，WSN内部で計測結果に応じて処理を実施するアプリケーションを複数構築することを想定する．このアプリケーションは空間的な計測を必要とするため，複数のノードに計測を行うプログラムを配備する必要がある．それに加え，取得された計測結果を判定し，要求に応じた処理を実行するためのプログラムが必要であり，これらのプログラムの連携が必要になる．本研究では，こうしたアプリケーションを*LCA*（Locally Centralized Application）と定義する．個々の*LCA*は，システム運用者の要求によって計測場所を変更することがあり，*LCA*を構成するプログラムの配備場所を変更することが必要になる．

ある*LCA*の稼動場所の変更が必要になった場合，Delugeでは，ノード上に静的にプログラムを配備しているため，*LCA*を構成するプログラムを配備先のノードから消去し，新配備先のノードにベースステーションから改めて配備することが必要になる．プログラムの配備場所の変更回数が多い環境下では，ベースステーションからプログラムを配備する回数の増加によって，ベースステーション付近のノードがプログラムの転送によって電力を消費し，バッテリが枯渇する恐れがある．それに対してAgillaでは，ノード上のプログラムを旧配備場所から新配備場所に移動させ，WSN内部で動的にプログラムを配備することができる．この手法は，ベースステーションからプログラムを配備する手法より，プログラムの平均移動距離が短く⋆1，特定のノードの電力を消費しないという長所があり，プログラムの配備場所の変更回数が多い環境下での運用に適している．

Agillaは，プログラム間通信やプログラムの動的配備の基本機能を提供するミドルウェアであり，アプリケーションの処理を記述したプログラムを複数同一ノードに配備することができる⋆2．ただし，複数のプログラムがWSN内部で連携するアプリケーションを構成する際に適したアーキテクチャを提案していない．そのため，*LCA*を構築する際にプログラムをアドホックに開発した結果，多様な処理を1つのプログラムとして実行することとなり処理とコード容量が増加し，定期的な計測処理が他の処理によって中断したり，軽量に作成すべきプログラムのコード容量が大きくなったりする可能性がある．また，*LCA*の稼動場所の変更が必要になった場合，計測を実行するプログラムの配備場所を変更する必要が生じ，*LCA*を構成する複数のプログラムを新配備場所に移動させる必要がある．Agillaでは，複数のプログラムの配備場所を変更する場合，統率を行うプログラムが通信を行って他プログラムに移動の呼びかけを行い，複数のプログラムが個々に移動を行う．この動的配備手法は，通信への依存度が高く，移動呼びかけの失敗や移動失敗によって，すべてのプログラムが移動を成功できない可能性がある．すべてのプログラムが移動して自身を配備できない場合，*LCA*の実行を継続できないという課題がある．

本論文では，これらの課題を解決するために，*LCA*を構成するためのアーキテクチャと，複数のプログラムを動的に配備する手法を提案する．前者については，*Master*，*Slave-S*，*Slave-M*というコンポーネント化されたプログラムによって構成されるアーキテクチャを提案する．後者については，既存手法の課題の原因となる通信への高い依存度を低下させるため，*Master*がすべてのプログラムのコードを持って移動を行い，複数のプログラム（*Slave-S*，*Slave-M*）を生成・配備する手法を提案する．

本論文の構成は次のとおりである．2章で事例と課題，3章で提案手法を説明する．4章で評価，5章で議論を行い，6章で関連研究を述べる．7章で本論文をまとめる．

2. 事例と課題

2.1 事　　例

本論文では，**図1**のように，WSNのインフラが事前に敷設された平置き倉庫を想定する．倉庫には入荷・出荷によって物品が倉庫に入ること，倉庫から出ることがある．加えて，他物品の出荷のためや，倉庫内移動のために物品が移動される場合⋆3があり，物品の現在位置が変更される可能性がある．本論文では，図1に示すように，物品が移動されたの

⋆1 ベースステーションと旧配備場所を結ぶ線の垂直二等分線でWSNを区切った場合，旧配備場所に近い領域に移動する場合は旧配備場所から移動した方が移動距離が短い．空間的な平均をとればベースステーションからつねに配備するより平均移動距離は短くなる．

⋆2 Agillaはミドルウェアに潤沢な機能を持たせることで動的配備されるプログラムを軽量化することにも成功している．

⋆3 物品は倉庫内でつねに移動をするということでなく，必要に応じて移動される可能性がある．

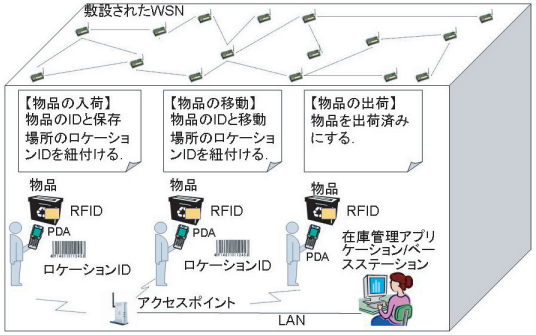

図 1　想定環境
Fig. 1　Assumed environment.

か，出荷されたのかといった在庫管理情報は，WSN とは別の在庫管理アプリケーションが把握していると仮定する．また，WSN を構成するノードにおいても物品の存在有無は検知できると仮定する．

本想定環境において，システム運用者である倉庫業者は，管理を要する高級な物品の周辺で，物品固有の計測と計測された状態に応じた処理を必要とする．たとえば，高級ワインの周辺では，周辺温度を 15°C 未満に保って保存を行い，15°C 以上の温度が計測された場合，ベースステーションや PDA に通知し状況を報告する．また，美術品の周辺には，夜間（光度が一定の閾値を下回った場合）に赤外線センサが反応した場合に侵入者と見なすプログラムを配備し，侵入者が検知された場合，侵入者を追跡するプログラムを WSN に散布し，緊急時のインフラとして WSN を利用する．

これらのアプリケーションは，物品の周辺で空間的な計測を実施するため，物品周辺の複数のノードに計測を行うプログラムを配備する必要がある．また，物品周辺での異常や平均値などの空間的な情報を判定するため，計測を行うプログラムと連携して，空間的な情報を判定し WSN 内部で要求に応じた処理を行うプログラムが必要になる[⋆1]．本論文では，こうしたアプリケーションを *LCA*（Locally Centralized Application）と定義する．

LCA は，物品が移動を行った場合，物品の新現在地周辺に移動を行って継続して実行できる必要がある．たとえばワインがシステム運用者によって他の場所に移動された場合，ワイン周辺で計測を行うプログラムは配備場所を変更する必要がある．既存技術を用いた場合，ワインの周辺で温度を計測していたプログラムが移動の呼びかけの受信に失敗した場合や，移動命令を受付けたが移動に失敗した場合，*LCA* を構成するプログラムの動的配備に失敗し，ワインの移動先では *LCA* を継続して実行することができない．

2.2　要求と課題

本研究で想定する *LCA* は，物品の周辺で計測や処理を行って，物品が移動した場合に，再度物品の周辺で計測や処理を実施できる必要がある．*LCA* 実現にあたる要求は次のとおり整理できる．

要求 A) 計測の実行

物品周辺の複数ノードで計測を実行できる．

要求 B) 計測結果の判定と処理

計測結果を判定し，要求に応じた処理を行う．

要求 C) 情報の収集・交換

WSN 内外の情報交換や，LCA を構成するための情報を収集する．たとえば，ベースステーションや PDA で稼動するアプリケーションと通信を実行する．

要求 D) 新現在地におけるプログラムの動的配備

LCA を構成するプログラムを新現在地周辺に移動させ，プログラムを配備して *LCA* を継続して構成できる．

Agilla は MOTE[6] で稼動するミドルウェアであり，WSN 内部でプログラムを動的に配備することを可能にするが，抽出された要求 A)〜D) を満たす場合，次の課題をかかえる．

課題 1：アーキテクチャ

課題 2：複数プログラムの動的配備

課題 1：Agilla は，個々のプログラムに対して通信や動的配備の機能を提供するが，*LCA* のような複数のプログラムで空間的なアプリケーションを構成する際のアーキテクチャがなく，*LCA* を構築する際に複数のプログラムをアドホックに開発することになる．このとき，多様な処理を 1 つのプログラムに実行させると，軽量に作成すべきプログラムのコード容量が増加して移動効率が悪くなる可能性や，定期的な計測を実行するべきプログラムが他の処理の実行によって，定期的な計測を中断する可能性がある．*LCA* の構成に適したアーキテクチャが必要となる．

課題 2：Agilla では，プログラム間通信を分散タプルスペース[7] を介して行っている．

⋆1 複数のアプリケーションを 1 つの WSN を用いて実現する場合，取得されたデータを，そのつどベースステーションに送信すると，ベースステーション付近のノードの電力が枯渇する可能性があるため，WSN 内部で可能な限り処理を実行する．

WSN に配備された複数のプログラムが配備場所の変更による移動を行う際，統率を行うプログラム（本論文では，リーダと呼ぶ）が他プログラムに移動の呼びかけを行う手法として，次の 2 つがある．1 つは，リーダが他プログラムの位置を取得しておいて，移動時に直接通知する手法である．もう 1 つは，リーダが，特定の分散タプルスペースに移動命令を書いておき，他プログラムは定期的に特定の分散タプルスペースを読み込み移動を検知する手法である．後者の手法では，他プログラムが移動の呼びかけを常時チェックする必要があり，呼びかけが通知されていない場合にもポーリングを行って電力を消費する．そのため，複数のプログラムが移動を行う際，移動を行いながらお互いの位置を通知しあう前者の手法が適している．WSN ではノード間の距離に依存してパケットロスが生じることが知られている[8]．プログラムの移動時に，通信に高く依存したアプローチをとると，次の課題によって，*LCA* を継続して構成できない可能性がある[⋆1]．

課題 2-(a) 移動の呼びかけに失敗し，一部のプログラムが移動できない．

課題 2-(b) 一部のプログラムが移動に失敗する．

3. 提 案 手 法

本論文では，2.2 節で述べた要求のうち，要求 A)，B)，C) を満たすためのアーキテクチャを 3.1 節で提案し，要求 D) を満たすための複数プログラムの動的配備手法を 3.2 節で提案する．

3.1 アーキテクチャ

LCA が，物品周辺で計測・処理を実行する際の要求は A) 計測の実行，B) 計測結果の判定と処理，C) 情報の収集・交換となる．本論文では，要求 B)，A)，C) を専任する *Master*，*Slave-S*，*Slave-M* というコンポーネント化されたプログラムにより構成されるアーキテクチャを提案し，*LCA* を構成する．*Master*，*Slave-S*，*Slave-M* の機能を**表 1** に示す．

Master，*Slave-S*，*Slave-M* でアーキテクチャを構成する理由を以下に示す．A) は，物品周辺のノードで計測を行うために，複数で物理的に異なるノードで動作する必要があり，1 つのプログラムとしてコンポーネント化するのが妥当である．B) は，複数の A) のプログラムと通信を行って物品周辺での事象を把握するため，同様に 1 つのプログラムとしてコンポーネント化する．C) は，A) または B) に機能を負わせることが可能だが，対象物を追跡する場合や，通信失敗回数が多い場合など，移動を行って情報を収集する方が適する場合がある．A)，B) のどちらかに C) の機能を負わせると，A) は計測を実施しているため，移動を行うと物品周辺での計測ができなくなる可能性がある．B) は A) からの報告を受付けているため，移動を行うと B) と距離が離れて通信の劣化や通信の転送などが生じ WSN では効率が悪い．加えて，移動を考慮した場合，プログラムは軽量な方が効率が良い．そこで，C) も 1 つのプログラムとしてコンポーネント化する．

表 1　アーキテクチャ
Table 1　Architecture.

プログラム	説明	要求
Master	*Slave-S*，*Slave-M* の動的な生成・配備を行い，ライフサイクルを管理する．*Slave-S* から報告される計測結果の判定を実施し，各 *LCA* の要求に応じた処理を行う．WSN 内外の通信を *Slave-M* に依頼する．*LCA* に 1 つだけ存在し，配備完了後は移動まで特定のノードで稼動する．	B
Slave-S	計測処理を実行し，異常の検知などの計測結果を *Master* に送信する．*LCA* に複数存在することができ，配備完了後は *Master* の移動まで特定のノードで稼動する．	A
Slave-M	*Master* からの命令によって，必要に応じて移動を行って情報収集や情報交換を行う．（例：アプリケーションとの通信を実行）*LCA* に 1 つ存在する．	C

表 1 に示すとおり，*Master* は，B) 以外に *Slave-S*，*Slave-M* の動的な生成・配備や，ライフサイクルの管理を行う（3.2 節で詳細を記述する）．*Slave-S* は物品周辺での計測処理を実施し，*Slave-M* は *Master* から依頼によって，情報収集と情報交換を実施する．

本アーキテクチャを事例にあてはめた場合，次の動作を行う．

(1) ワインの周辺に各プログラムが配備される．
(2) ワイン周辺の隣接ノードで *Slave-S* 合計 4 体が計測を行い，異常があれば *Master* に通知を行う．
(3) 計測対象が移動した場合，*Master* は *Slave-M* をベースステーションや PDA で稼動する在庫管理アプリケーションに派遣する．
(4) Master は Slave-M 経由で在庫管理アプリケーションから新配備場所を取得する．
(5) 新配備場所に移動し，*LCA* を構成する．

⋆1 これを回避するために，プログラムの一部が移動できなかった場合に備えて，移動後にプログラム間で通信を実施したり，プログラムが存在しない場合に検索をしたり，プログラムが消失した場合に複製を行う対策があるが，通信が必ずしも信用できない WSN では状況を正確に把握することに限界がある．

Slave-S は，計測対象を計測するためのプログラムである．このプログラムは，物品の周辺で空間的なデータを取得するため *LCA* に複数個存在し，複数のノードに 1 体ずつ配備される．*Slave-M* は，*Master* からの通知に応じて情報収集・交換を実行する．たとえば，在庫管理アプリケーション[⋆1]との通信を実施し，WSN 内部および WSN 外部で把握できない事象を報告・取得する．この例では，WSN 内部ではプログラム側で計測対象の存在有無は計測できるが，計測対象が移動したのか，出荷されたのか判断できない．在庫管理アプリケーションと通信を行って，計測対象の新現在地を取得する．出荷された場合には，*LCA* の活動を停止する結果を *Master* に通知する．*Master* は，*Slave-S* から報告された計測結果に合わせた判定を実施する．たとえば，計測結果が異常であった場合に，在庫管理アプリケーションとの通信を *Slave-M* に依頼する．

3.2　複数プログラムの動的配備

LCA の稼動場所の変更によって，WSN 内部で複数プログラムを動的に配備する場合がある．既存手法を用いると，2.2 節であげた以下の課題 2-(a)，(b) によって *LCA* を継続して実行できない場合がある．

課題 2-(a) 移動の呼びかけに失敗し，一部のプログラムが移動できない．

課題 2-(b) 一部のプログラムが移動に失敗する．

提案手法では，これらの課題を解決するために，*LCA* を継続実行することに着眼した手法をとり，課題 2-(a)，(b) が起こる確率を減らす手法を提案する．

課題 2-(a) の確率をなくすため，特定のプログラムが，他のプログラムを生成する手法を提案する．この手法は，移動先で他のプログラムを生成し，配備することができるため，他のプログラムに移動の呼びかけを行う必要がないが，移動前に配備したプログラムを消去する必要がある[⋆2]．課題 2-(b) は通信が 100%信頼できない WSN では完全に解決することはできないが，確率を下げることはできる．既存手法を用いてプログラム N 体を移動させる場合，個々のプログラムが移動に失敗する確率を P とすれば，N 体すべてのプログラムが移動に成功する確率（*Probability*）は以下の式で表現できる．

$$Probability = (1 - P)^N \tag{1}$$

提案手法では，この確率（*Probability*）を向上させるために N を減らすアプローチをとり，*Master* が *Slave-S* と *Slave-M* のコードをすべて保持する（つまり N が 1 になる）．以上のことから，提案手法では，課題 2-(a)，(b) が起こる確率を減らすことができる[⋆3]．

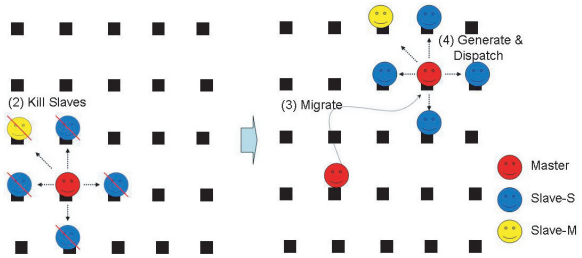

図 2　複数プログラムの動的配備
Fig. 2　Dynamic deployment of multiple programs.

提案手法は以下の手順で複数プログラムの動的配備を行う．

(1)　*Master* に移動が通知される．
(2)　*Master* は配備した *Slave* を消去する命令をミドルウェアに通知する．
(3)　*Master* は移動を実施する．
(4)　*Master* は移動先で *Slave* を生成し配備する．

図 2 に (2)〜(4) に対応した例を示す．(2) *Master* は移動が必要になった際に，配備された *Slave* の消去をミドルウェアに通知する．*Slave* の消去はミドルウェアが実行する．(3) *Master* は *Slave* の消去確認を行わずに，次の配備場所に移動する．(4) *Master* が移動後，*Master* の配備先のノードのミドルウェアを介して *Slave* を生成し，*Slave* を周辺ノードに配備する．これを繰り返すことで，*LCA* を継続して実行することができる．

3.3　実　　装

表 2 に実装環境を示す．本研究では Agilla をベースとして，以下の機能を拡張した．

拡張機能 1　*Master* が *Slave* を生成する機能
拡張機能 2　*Master-Slave* 間のロケーション管理機能
拡張機能 3　*Slave* の消去機能

本研究で実装したミドルウェアを**図 3** に示す．プログラムはミドルウェアに ISA（Instruction Set Architecture）により処理を依頼する．*Instruction Set Handler* は，実行さ

⋆1 ベースステーションや PDA で稼動を行っている在庫管理アプリケーション
⋆2 4.2 節で消去の成功率を評価し，評価環境において 100%を達成している．
⋆3 提案手法では，*Master* が移動に失敗すると *LCA* を継続して構成できないという欠点があるが，既存手法でもリーダが 1 体存在することから，この欠点は既存手法と同じである．

表 2　実装環境
Table 2　Development tools.

項目	説明
OS	TinyOS 1.2.2 [9]
開発言語	NesC 1.2.7 [10]
ランタイム	Agilla 3.0.2 [11]
開発環境	Windows XP Professional SP2, Cygwin 1.5.23

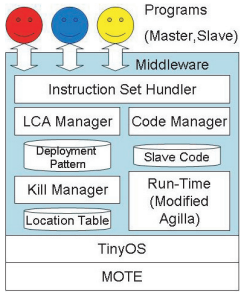

図 3　ミドルウェア
Fig. 3　Middleware.

れる命令を振り分け，拡張された機能と Agilla で提供される機能を振り分ける．なお，本研究で作成した ISA を付録 A.3 の表 11 に示す．

拡張機能 1：ミドルウェアは新たなプログラムが送られてきた場合，*Master* であるか否か，移動命令とプログラムの ISA から判別する．*Master* であった場合，*Code Manager* が *Master* のコードから *Slave* を生成し，*Master*，*Slave* の各コードをそれぞれのコードバッファに格納する．このとき，*Master* のコードは，実行コードバッファに格納され実行されるが，*Slave* のコードは *Slave Code* に格納された後，配備場所に送信される．*Slave* の配備場所は，*Master* コード中に記載された *Deployment Pattern* により決定され，*Deployment Pattern* に従って *LCA Manager* が *Slave-S*，*Slave-M* をそれぞれの配備場所に定められた数だけ送信する．*Slave-S*，*Slave-M* を受信したミドルウェアは，*Slave* のコードを実行コードバッファに格納し，*Slave-S*，*Slave-M* が実行される．

拡張機能 2：ミドルウェアは，*Slave* を送信し Ack を受け取った際に，*Slave* の送信先の

表 3　プログラムの記述
Table 3　Programming.

```
// Master Code ————————
        (Master の処理の記述)
// Slave-S Code ————————
        startslaves //Slave-S の開始
        (Slave-S の処理の記述)
        endslaves //Slave-S の終了
// Slave-M Code ————————
        startslavem //Slave-M の開始
        (Slave-M の処理の記述)
        endslavem //Slave-M の終了
```

ノード ID と *Slave* のプログラム ID を *Location Table* に記録する．同様に *Slave* を受信した際には，*Master* の存在するノード ID と *Master* のプログラム ID を *Location Table* に記録する．*Location Table* に記録された情報を用いて *Master-Slave* 間の通信が可能になる．なお，*Master-Slave* 間の通信は Agilla で提供されている分散タプルスペースを介して行われる．

拡張機能 3：プログラムの移動が必要になった場合，*Master* は *Slave* を消去する命令をミドルウェアに下す．*KillManager* は，*Location Table* の情報をもとに消去対象となる *Slave* の存在するノードに消去命令を送信し，命令を受信したミドルウェアは，指定された *Master* の ID を親に持つ *Slave* の消去を行う．

Master，*Slave* のプログラミングは，Agilla で提案されている ISA と，本研究で拡張した ISA（付録 A.3 の表 11）を用いて記述し，**表 3** のように *Master* の内部に *Slave* の処理を記述する形をとる．*Slave-S* の処理は，startslaves，endslaves という ISA の間に記述する．同様に，*Slave-M* の処理は，startslavem，endslavem の間に記述する．

なお，*Slave* の配備場所を決定する *Deployment Pattern*（配備パターン）は，ミドルウェアの中に保存された *Slave* の空間的な配備情報である．本研究で使用した配備パターンを付録 A.1 の図 10 に示す．配備パターンの充実，状況に合わせた配備パターンの取得は，本論文の提案範囲外である．

4. 評　　価

4.1 シミュレーション環境

本論文では，TOSSIM [8] において Lossy モデル[⋆1]を使用して評価を行った．なお，評価時のルーティングプロトコルとして Geographic Routing を用い，ノード間の通信は単純な Greedy Forwarding [12] で行われる．評価における共通の計測条件を**表 4** に示す．なお，隣接ノードのみ通信可能としているのは，効率的にマルチホップ通信を評価するためであり，提案手法の制約ではない．

本論文の評価で用いる 5 feet，8 feet，10 feet はグリッド上に配置されるノード間の距離を示し，それぞれ約 1.5 m，2.4 m，3.0 m に相当する．TOSSIM の Lossy モデルは，ノード間の距離に応じたノード間リンクのビットエラー率を記述したファイルをツールを用いて生成し，このファイルをシミュレーション時に読み込むことによって，実測値と同等なパケットロスを再現する．表 4 に，生成されたファイルにおける，全隣接ノード間（対角線を含む）の平均ビットエラー率と全隣接ノード間（対角線を含む）の非対称リンクの割合を示す[⋆2]．

表 4　共通の計測条件
Table 4　Common envirornment among items of evaluation.

項目	説明
トポロジ	グリッド（8 × 8）
Lossy モデル	5 feet，8 feet，10 feet
平均ビットエラー率	0.18% (5 feet)，0.29% (8 feet)，0.96% (10 feet)
非対称リンクの割合	63.86% (5 feet)，77.14% (8 feet)，75.24% (10 feet)
ノード間通信	対角線を含む隣接ノードのみ可
通信速度	40 Kbps
メッセージサイズ	36 byte (Active Message)

⋆1 TOSSIM では，TinyOS の実装言語である NesC で記述されたプログラムを稼動させることができる．また，Lossy モデルを用いることでノード間の非対称リンクとパケットロスをシミュレーションできる．

⋆2 パケットロス率や，パケットエラー率の詳細，算出式については文献 8)，13) を参照のこと．なお，本論文中では，ノード間の距離が増加した場合に “パケットロスが多い” と記述しているが，文献 8)，13) において，ノード間の距離が増加した場合にパケットロスが多くなることが示されているため，妥当な表現である．

4.2 アーキテクチャの評価

LCA を実現する場合，*Master* が移動を実施しながら *Slave* を配備して，*LCA* を構成することが必要になる．そこで，提案手法を用いて以下のシナリオを評価した（具体的なコードは付録 A.2 を参照のこと）[⋆3]．

(1)　*Master* が配備場所に移動する．
(2)　*Master* が *Slave-S* を隣接ノードに 1 体ずつ合計 4 体配備する．
(3)　*Master* が *Slave-M* を隣接ノードに 1 体配備する．
(4)　*Master* の命令に応じて *Slave-M* がベースステーションに行く．
(5)　*Slave-M* がベースステーションに報告を行う．
(6)　ベースステーションより新しい場所を取得し移動を行う．

この測定を 6 回繰り返すことを 1 回とし，5 feet，8 feet，10 feet で各 15 回ずつ実施した結果を**表 5** に示す．なお，本計測では，*LCA* の移動間隔として 2 ホップ移動させた．成功率は (1) から (6) までのすべてに成功した確率を示す，2 回移動する成功率は，各環境でそれぞれ，100%，93.3%，73.3%であった．移動回数が増えるに従って，パケットロスが多い環境では，移動をともなう *LCA* 構成の成功率が顕著に低下した．5 回移動する成功率は，各環境でそれぞれ，86.6%，60.0%，33.3%であった．本評価により，2 回の移動をともない，初期配備も含めて合計 3 カ所で計測を実施する *LCA* は，5 feet，8 feet の環境において 90%以上の確率で継続して運用できることが分かる．ただし，ノード間隔が広がりパケット

表 5　アーキテクチャの評価
Table 5　Evaluation of the architecture.

シミュレーション環境	5 feet	8 feet	10 feet
2 回移動–成功率（提案手法）	100%	93.3%	73.3%
2 回移動–成功率（Agilla）	73.3%	60.0%	33.3%
3 回移動–成功率（提案手法）	93.3%	73.3%	40.0%
3 回移動–成功率（Agilla）	53.3%	40.0%	20.0%
4 回移動–成功率（提案手法）	93.3%	66.7%	40.0%
4 回移動–成功率（Agilla）	33.3%	20.0%	6.8%
5 回移動–成功率（提案手法）	86.6%	60.0%	33.3%
5 回移動–成功率（Agilla）	26.7%	13.3%	0%

⋆3 本評価では，図 10 に示す配備パターンの 1 を利用している．なお，付録 A.2 に示すコード容量は 92 byte である．

ロスが多い環境における *LCA* の継続運用や，稼動場所の変更の多い *LCA* の継続運用は今後の課題となる．

Agilla の手法を用いて，複数のプログラムが移動を実施して *LCA* を連続して構成する場合の 5 feet，8 feet，10 feet における評価を，表 5 に示す（15 回の測定）．既存手法より，提案手法のほうが *LCA* を継続運用する際に有効な手法であることが分かる．

提案手法の欠点として，以下が考えられる．

欠点 (a)：*Master* の移動失敗により *LCA* が構成できない．

欠点 (b)：*Slave* の配備失敗により *LCA* を構成できない．

欠点 (c)：*Slave* の消去失敗により *Slave* が残存する．

Slave-S の消去に失敗した場合，*Slave-S* は，*Master* の旧存在場所の分散タプルスペースに継続して計測結果を通知する．そのため，利用者のいないデータを送受信するために *Slave-S* が存在するノードと *Master* の旧存在場所のノードの電力を無駄に消費し続ける[⋆1]．

これらの欠点を評価するため，付録 A.2 に示すコードをシミュレーション環境において，2 ホップ移動させる評価を各環境で 60 回実施した．**表 6** に評価結果を示す．

欠点 (a)：*Master* の移動失敗率は，*Master* が移動に失敗した確率を示す．各シミュレーション環境で，5.0%，8.3%，18.3%計測された．パケットロスが多い環境での *Master* の移動失敗は今後の課題となる．

欠点 (b)，(c)：本ミドルウェアでは，配備の失敗や，消去の失敗時に再送[⋆2]を行うことで対応している．再送無–配備成功率は，再送を実施しなかった場合に *Slave-S* を 4 体，*Slave-M* を 1 体配備することに成功した確率である．再送を実施しない場合，各環境では，80.0%，54.9%，19.0%しか配備に成功しない．再送無–KILL 成功率は，再送を行わない場合に配備した *Slave* をすべて消去できる確率である．各環境で，90.9%，68.6%，52.4%しか消去できない．再送を行うことで，評価環境では配備成功率，KILL 成功率ともに 100%を達成している．以上の結果から，提案するアーキテクチャは *LCA* を構成する際のアーキテクチャとして妥当と判断できる．

4.3　複数プログラムの動的配備の評価

提案手法，Agilla の手法で，3 つのプログラムを一定の距離を保ちながら移動させ，複数のプログラムを動的に配備する際の成功率を計測した．以下を 4 回繰り返すことを 1 回の計測として，15 回の計測を実施し，各手法によって 3 つのプログラムが移動しながら自身を何回配備できるかを評価した．

- 1 つのプログラムに移動を通知する．
- 3 つのプログラムが隣接ノードに移動を行い自身を配備する

Agilla の手法では各プログラムのコード容量は，73 byte，45 byte，45 byte[⋆3]である．提案手法では，*Slave-S*，*M* を 1 体ずつ準備し 3 体の動的配備を実現した．*Master* のコード容量は 60 byte であり，*Slave-S*，*M* のコード容量は 12 byte である．

計測結果を**表 7**，**表 8** に示す．表 7 はプログラム間の距離が 1 ホップ[⋆4]のときの結果であり，表 8 はプログラム間の距離が 2 ホップ[⋆5]のときの結果である．平均配備可能数は，Agilla の手法では，3 つのプログラムのどれか 1 つが移動に失敗したり，移動の呼びかけの通信が失敗して取り残されたりするまで 3 つのプログラムがすべて配備された回数の平均である．提案手法の場合は，*Master* が移動に失敗するか，*Slave*2 体のうち 1 体でも配備に失敗[⋆6]するまで配備できた回数の平均である．成功率は，4 回移動を成功させ，かつ 3 つのプログラムがすべて配備される確率である．

表 7 は，プログラム間の間隔が 1 ホップのときの計測結果である．提案手法では，10 feet の Lossy 環境において，成功率が 66.7%であるのに対し，Agilla の手法では，40.0%であっ

表 6　欠点の評価
Table 6　Evaluation of drawbacks.

シミュレーション環境	5 feet	8 feet	10 feet
Master の移動失敗率	5.0%	8.3%	18.3%
再送無–配備成功率	80.0%	54.9%	19.0%
再送有–配備成功率	100%	100%	100%
再送無–KILL 成功率	90.9%	68.6%	52.4%
再送有–KILL 成功率	100%	100%	100%

⋆1 *Slave-S* のコードはたかだか十数 byte〜数十 byte であり，メモリに対する制約よりも，電力を消費し続ける方が課題と考えられる．本評価における *Slave-S* のコード容量は 20 byte であり，メモリの空き容量（60 Kbyte）に対する割合は 0.03%となる．

⋆2 本評価では，*Slave* の配備失敗時の再送回数は 2 回，消去失敗時の再送回数は 1 回として評価を実施した．

⋆3 Agilla の手法では，お互いの位置を把握するためのコードが煩雑になり，提案手法よりコード容量が増える．

⋆4 1 つのプログラムを中心として，隣接ノードのうち 2 つのノードにプログラムが存在している．図 10 に示す配備パターンの 2 を利用．ただし，図 10 における *Slave-S* の位置に *Slave-S* と *Slave-M* を配備し，図 10 における *Slave-M* の位置には *Slave* を配備しない．

⋆5 1 つのプログラムを中心として，2 ホップ隣のノードのうち 2 つのノードにプログラムが存在している．図 10 に示す配備パターンの 3 を利用．ただし，図 10 における *Slave-S* の位置に *Slave-S* と *Slave-M* を配備し，図 10 における *Slave-M* の位置には *Slave* を配備しない．

⋆6 提案手法ではある場所で *Slave* の配備に失敗しても，次の配備場所で *Slave* の配備に成功できる可能性があるが，この場合は失敗としている．

表 7　複数プログラムの動的配備（1 ホップ）
Table 7　Dynamic deployment (1 hop).

プログラム	5 feet	8 feet	10 feet
Agilla–平均配備可能数	3.46	3.00	2.66
Agilla–成功率	80.0%	60.0%	40.0%
提案手法–平均配備可能数	4.00	3.73	3.60
提案手法–成功率	100%	80.0%	66.7%

表 8　複数プログラムの動的配備（2 ホップ）
Table 8　Dynamic deployment (2 hop).

プログラム	5 feet	8 feet	10 feet
Agilla–平均配備可能数	2.86	2.06	1.20
Agilla–成功率	53.3%	26.7%	13.3%
提案手法–平均配備可能数	4.00	3.40	3.20
提案手法–成功率	100%	73.3%	60.0%

た．平均配備可能数は，それぞれ 3.60 と 2.66 であった．また，パケットロスが少ない 5 feet の Lossy 環境において，提案手法が 100%の確率で配備ができるのに対して，Agilla の手法では 80.0%にとどまった．

表 8 は，移動を行うプログラム間の間隔が 2 ホップのときの計測結果である．提案手法では，10 feet の Lossy 環境でも，成功率が 60.0%であるのに対して，Agilla の手法では，13.3%であった．平均配備可能数は，3.2 と，1.2 であった．また，パケットロスが少ない 5 feet の Lossy 環境において，提案手法が 100%の確率で配備ができるのに対して，Agilla の手法では 53.3%にとどまった．以上の結果から，提案手法の動的配備手法は Agilla で動的配備を実現するよりも有効であると判断できる．

4.4　オーバヘッドの評価

提案手法は *Master* が移動先で *Slave* を配備するため，トラフィックが *Master* の移動先のノードに集中し，動的配備時間のオーバヘッドが生じる．そこで，本節で**動的配備時間**と動的配備に必要な**トラフィック**の計測結果を示す．

動的配備時間：動的配備時間を，プログラムが動的配備を開始する時間（*T_Start*）から，すべてのプログラムが動的配備を終了して実行可能となる時間（*T_End*）の差分と定義する．**図 4** に，提案手法，Agilla の動的配備時のシーケンス図を示す．なお，図 4 には，Agilla における他プログラム，提案手法における *Slave* は簡単のため各 1 体のみ示す．

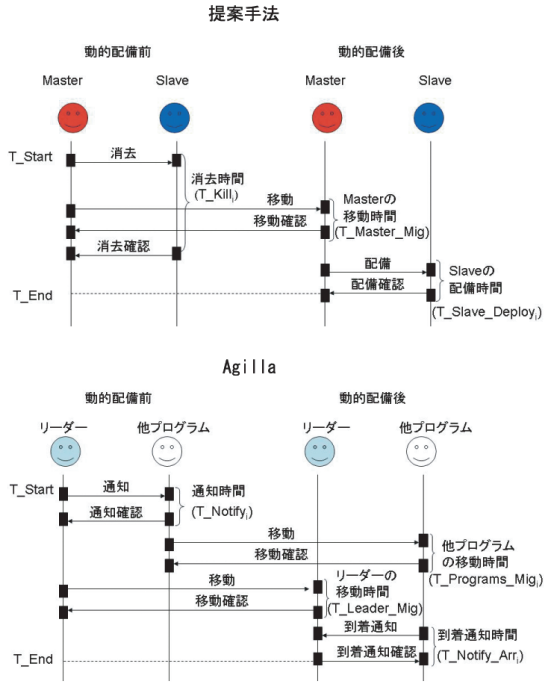

図 4　動的配備時のシーケンス図
Fig. 4　Sequence for dynamic deployment.

表 9 に 5 feet，8 feet，10 feet における，各パターン（付録 A.1 図 10）で 1 ホップ *LCA* を移動させた際の平均動的配備時間（計測回数 15 回）を示す[⋆1]．表 9 より，以下 2 つの傾向が分かる．

傾向 1：提案手法の動的配備時間は，パケットロスが少なく（5 feet，8 feet），プログラム数が少ない環境（パターン 2，3）では Agilla と同程度の動的配備時間となる．

傾向 2：提案手法の動的配備時間は，パケットロスが多く（8 feet，10 feet），プログラム数

⋆1 動的配備に失敗した場合は除く，なお配備パターン 1 のコード容量は提案手法の Master が 92 byte（Slave-S：20 byte，Slave-M：31 byte），Agilla のリーダは 139 byte，他プログラムは 45 byte である．配備パターン 2，3 は，提案手法の Master が 60 byte（Slave-S：12 byte，Slave-M：12 byte），Agilla のリーダは 73 byte，他プログラムは 45 byte である．

表 9　平均動的配備時間
Table 9　Average time for dynamic deployment.

パターン	手法	5 feet	8 feet	10 feet
1	Agilla	3.78(sec)	4.47(sec)	7.94(sec)
1	提案手法	4.33(sec)	11.51(sec)	21.19(sec)
2	Agilla	1.35(sec)	1.50(sec)	1.76(sec)
2	提案手法	1.07(sec)	2.28(sec)	3.48(sec)
3	Agilla	1.40(sec)	1.83(sec)	2.15(sec)
3	提案手法	1.32(sec)	2.62(sec)	4.01(sec)

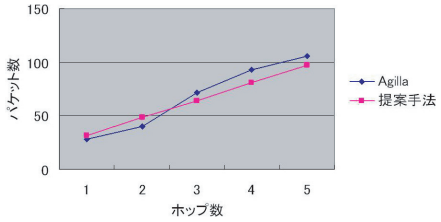

図 5　動的配備時のパケット数（5 feet）
Fig. 5　Packet count between dynamic deployment (5 feet).

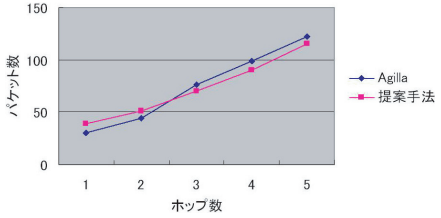

図 6　動的配備時のパケット数（8 feet）
Fig. 6　Packet count between dynamic deployment (8 feet).

が大きな環境（パターン 1）では，Agilla に比べて 2 倍〜3 倍と長くなる．

トラフィック：各シミュレーション環境における，提案手法と Agilla の動的配備時のパケット数を評価した結果を，**図 5**，**図 6**，**図 7** に示す．提案手法は *LCA* 構成に最適化されているため，コード容量を Agilla に比較して削減することができるが，本評価ではコード

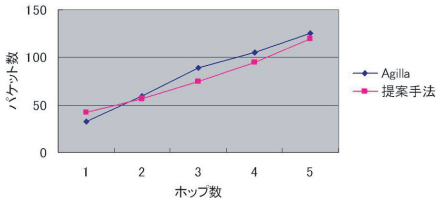

図 7　動的配備時のパケット数（10 feet）
Fig. 7　Packet count between dynamic deployment (10 feet).

表 10　最大トラフィックを持つノードの平均パケット数
Table 10　Average packet count on a node which has max traffic.

手法	5 feet	8 feet	10 feet
Agilla	10.0	10.6	14.1
提案手法	17.9	21.3	22.2

容量[⋆1]を同等にして評価を行った．図 5，図 6，図 7 のパケット数は，提案手法，Agilla の手法でプログラム数を 3 体とし，各シミュレーション環境で *LCA* を 1 ホップから 5 ホップ移動させた場合に各ノードが送信するパケットの合計数の平均値（計測回数 5 回）を示す．図 4 に示すとおり，提案手法のパケットの合計数は，各ノードで送信される消去・消去確認，移動・移動確認，配備・配備確認に要する送信パケット数の和に相当し，Agilla のパケットの合計数は，各ノードで送信される通知・通知確認，移動・移動確認（リーダ，他プログラム），到着通知・到着通知確認に要する送信パケット数の和に相当する．図 5，図 6，図 7 より，提案手法，Agilla の動的配備に要するパケット数は，ほぼ同等であることが分かる．

提案手法は *Master* の移動先のノードが *Slave* の配備を実施するために，当該ノードが最大のトラフィックを持つ．**表 10** に，両手法における最大のトラフィックを持つノードが送信するパケット数の平均値（計測回数 25 回）を示す．表 10 より，提案手法は Agilla に比べてトラフィックが集中するノードが存在し，約 2 倍程度のパケット数を送信する負荷を持つことが分かる．

⋆1 提案手法では Master を 163 byte，Slave-S，Slave-M を 45 byte とし，Agilla ではリーダを 73 byte，他プログラムを 45 byte とした．

5. 議　　論

5.1 提案手法の限界

提案手法には，以下の限界がある．

限界 1　*Slave* の実行状態の維持

限界 2　同種 *Slave* のプログラムは均一

限界 3　メモリが溢れる場合

限界 4　動的配備時間中の計測

上で列挙した限界について，以下に詳細を示す．

限界 1：本ミドルウェアはモバイルエージェントのランタイムである Agilla を拡張しているため，プログラムは実行状態を保ったまま移動を行うことができる．提案手法では，*Master* は実行状態を保って移動はできるが，*Slave* は，*Master* の移動のたびに破棄され，移動後生成されるために，実行状態や内部変数を維持することはできない．ただし，本論文で提案する，*Slave-S* は計測処理を繰り返し，*Slave-M* は *Master* からの通知を待つプログラムであり，*Master* 移動の前後で実行状態や内部変数を維持する必要がないため，提案手法の限界とはならない．

限界 2：本ミドルウェアでは，*Slave* を指定された数だけ生成することはできるが，生成されたプログラムは各 *Slave* ごとに均一のプログラムである．同種の *Slave* が連携を要する用途を想定する場合，機能分担と同種 *Slave* 間の通信が必要になる．機能分担を実施する場合，すべての機能を持つコードを 1 つの *Slave* 中に持つ必要が生じ，新規 *Slave* を作成する際とのトレードオフを考慮する必要がある．*Slave* 間の連携は今後の課題となる．

限界 3：本ミドルウェアのコード容量は 66.3 Kbyte である[⋆1]．MOTE には，128 Kbyte のプログラムメモリがあるため，本ミドルウェアは約 60 Kbyte を *LCA* を構成するプログラムの領域のために利用することができる．本ミドルウェアは *LCA* を構成するプログラムの領域が溢れてしまった場合，*LCA* を構成することができず，提案範囲外となる．

溢れる可能性については次のとおり考察できる．たとえば，付録 A.2 に示す *Master*[⋆2]が図 10 における配備パターン 1 で，ある特定ノードおよびその隣接ノードに 1 体ずつ存在する場合，特定ノードに存在する *Master*，*Slave-S*，*M* のコード容量の合計値は 203 byte となる[⋆3]．したがって，付録 A.2 に示す規模のコード容量を持つ *Master* は，ある地域に密集しても各ノードに約 300 個存在することができる[⋆4]．本評価で用いた最大の空間領域を持つ 10 feet（3 m）四方の領域において 300 個程度の計測対象はプログラムメモリの制約上は実現できる[⋆5]．より大きなコード容量を持つ *Master*，*Slave* を配備する場合は，メモリが溢れる可能性があり，溢れた場合の対策が必要になる．

限界 4：提案手法や Agilla は WSN 内部でプログラムを動的配備するため，計測対象が移動を行い，動的配備を実行する際のオーバヘッドとなる動的配備時間における計測が不可能である．

上で列挙した限界以外に，提案手法は，*Master* が移動に失敗すると，継続して *LCA* を構成できない．本限界については，4.2 節で評価を実施しているが，対策として *Master* の冗長化を行うことが考えられる．なお，この問題は提案手法固有の問題ではなく，Agilla も同等である．

5.2 メモリに対する提案手法の妥当性

提案手法は *Master* が *Slave* をかかえて動的配備を実行する．そのため，Agilla に比してメモリを消費する可能性がある．提案手法，Agilla が *LCA* を構成する際の，両手法における全プログラムの合計コード容量（*T_CodeSize_P*，*T_CodeSize_A*）は以下の式で表現できる（算出根拠は付録 A.4 参照）．

$$T\_CodeSize\_P = Master\_Process + 6 + (N+1) * Slave\_CodeSize \quad (2)$$

$$T\_CodeSize\_A = Master\_Process + (N-1) * (Slave\_CodeSize + 56) \quad (3)$$

なお，式 (2)，(3) における *Master_Process*，*Slave_CodeSize* は *Master* の処理部，*Slave* のコード容量を，*N* は全プログラム数を示す．**図 8**，**図 9** に提案手法，Agilla によって記述されるプログラムの合計コード容量を示す．図 8 では，*Master_Process* を 30 byte，*Slave_CodeSize* を 100 byte とし，*Slave* のコード容量が *Master* より大きな場合の合計コード容量とプログラム数の関係を示す．同様に，図 9 では，*Master_Process* を 50 byte，*Slave_CodeSize* を 30 byte とし，*Slave* のコード容量が *Master* より小さい場合の合計コード容量とプログラム数の関係を示す．なお，Agilla のリーダはプログラム数によってコード容量が可変となるため，図 8，図 9 に *Master* のコード容量（固定）とともに示す．図 8，図 9 より，*Slave*

⋆1 Agilla のミドルウェアのコード容量は 57.9 Kbyte である．
⋆2 92 byte
⋆3 203=92+4*20+31:Master:92 byte，Slave-S:20 byte，Slave-M:31 byte
⋆4 296 = 60*1000/203
⋆5 データメモリ，バッテリなどの計算資源に対するプログラム間の競合解消は提案範囲外となる．これらは，TinyOS に依存する．これは既存研究（Agilla，Deluge）も同じである．

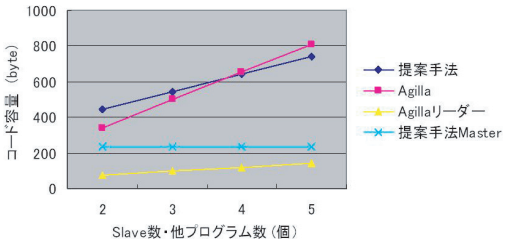

図 8　全プログラムの合計コード容量 1
Fig. 8　Total code size of deployed programs 1.

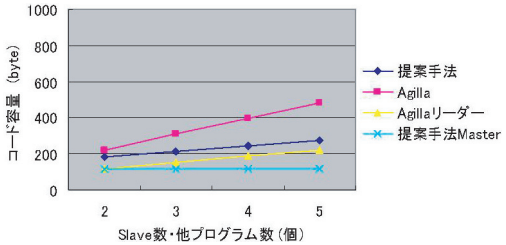

図 9　全プログラムの合計コード容量 2
Fig. 9　Total code size of deployed programs 2.

のコード容量が小さく，プログラム数が多い場合，提案手法の合計コード容量は Agilla に比べ小さいことが分かる．*Slave* のコード容量が大きく，プログラム数が少ない場合に，提案手法の合計コード容量は Agilla に比べ大きくなる．

図 8 において，*Slave*，他プログラム数が 2 の場合（合計 3 つのプログラムで *LCA* を構成する場合），*Master* のコード容量は 236 byte となり，リーダのコード容量は 74 byte となる．この差（162 byte）は，メモリの空き領域（60 Kbyte）の 0.27%に相当し，メモリに対して大きな制約とならない．加えて，この差は，図 9 のような *Slave* が小さなアプリケーションによって相殺される可能性がある．以上のことより，*Master* が *Slave* を所持する提案手法は妥当といえる．

5.3　オーバヘッド

本節では，4.4 節において評価された**動的配備時間**，**トラフィック** について論じる．

動的配備時間：提案手法の動的配備時間（*Time_P*），Agilla（*Time_A*）の動的配備時間は以下の式で表現できる（図 4 参照）．なお，式 (4)，(5) における N は全プログラム数を示す[⋆1]．

$$Time\_P = MAX\left(\sum_{i=1}^{N-1} T\_Kill_i,\ T\_Master\_Mig + \sum_{i=1}^{N-1} T\_Slave\_Deploy_i\right) \tag{4}$$

$$Time\_A = MAX\left(\left(\sum_{i=1}^{N-1} T\_Notify_i + T\_Leader\_Mig\right),\right.$$
$$\left. MAX_{1<i<N-1}\left(\sum_{j=1}^{i} T\_Notify_j + T\_Programs\_Mig_i + T\_Notify\_Arr_i\right)\right) \tag{5}$$

4.4 節では，平均動的配備時間に以下の傾向があった．

傾向 1：提案手法の動的配備時間は，パケットロスが少なく（5 feet，8 feet），プログラム数が少ない環境（パターン 2，3）では Agilla と同程度の動的配備時間となる．

傾向 2：提案手法の動的配備時間は，パケットロスが多く（8 feet，10 feet），プログラム数が大きな環境（パターン 1）では，Agilla に比べて 2 倍〜3 倍と長くなる．

傾向 1 については，N が少なく，パケットロスが少ない環境では，*Slave* の配備時間を要さず，すべての *Slave* を配備する時間を意味する式 (4) の第 2 引数の第 2 項（$\sum_{i=1}^{N-1} T\_Slave\_Deploy_i$）の影響が小さく，提案手法の動的配備時間が短くなると考えられる．

傾向 2 については，N が多く，パケットロスが多い環境では，*Slave* の再送が *Master* の存在するノードで集中・連続して起こることにより，上に示した式 (4) の第 2 引数の第 2 項の影響が大きく，動的配備時間が長くなる．Agilla の場合は，式 (5) に示すとおり，再送が起こってもプログラムが存在するノードが別であり，再送は複数のノードで分散して生じるのでリーダと他プログラムの移動は大きなオーバヘッドとはならない．

提案手法の動的配備時間は，最悪の場合 Agilla の動的配備時間の約 3 倍かかる（表 9 における配備パターン 1，10 feet）が，本研究では倉庫を想定し，計測対象の物品は通常は移動を行わず，物品移動時の動的配備速度に対する要求を想定していない．たとえば，2 日間倉庫に存在する物品が，配備パターン 1，10 feet 環境で 2 回移動を行ったとしても，動的配

⋆1 式 (4) に示すように，提案手法は消去確認を行わずに *Master* が移動を行い，*Slave* を移動先で配備する．式 (5) に示すように，Agilla のリーダはすべての他プログラムに順番に通知を行ってから移動を行う．Agilla の他プログラムは通知を待ってから移動と到着通知を実施する．

備時間は約 0.025% [⋆1] となり，無視できる範囲となる．

トラフィック：プログラムのコード容量，プログラム数が同等な条件では提案手法，Agilla ともに WSN 内部でのトラフィック量はほぼ同等であった．動的配備手法以外に違いがないことから，提案手法，Agilla の消費電力は同等と理解できる．提案手法は *Slave* が *Master* の存在するノードから配備されるために一部のノードに負荷がかかる．本制約は，本研究の想定環境である倉庫において，物品がある程度均質に分散することを考慮すると，*Master* も均質に分散するために，特定のノードに *Slave* を配備する負荷が集中することにはならず，WSN 内部のトラフィックも Agilla と同等であることから，想定環境における課題とならない．

5.4　対象とするアプリケーション

本論文では，*LCA* の構成を対象とし，ノードに *Slave-S* を配備して，周辺環境の計測を実施し，必要に応じてアプリケーション独自の処理を行うことを目的にしている．そのため，本論文で事例として用いた物品の周辺の計測や，ある限られた範囲における異常の監視などのアプリケーションが対象領域となる．

本論文では，物品の現在位置をベースステーションや PDA で稼動する在庫管理アプリケーションが管理していると仮定しているが，*Slave-M* が，物品が移動した後に新移動先を取得することができれば，*Master* は在庫管理アプリケーションとの通信を行わずに移動する手法を実現できる[⋆2]．加えて，他プログラム，他 *Master* からの通知によって，*Master* が移動を行うことも可能であり，在庫管理アプリケーションの仮定は提案手法の制約とはならない．

なお，*Slave-M* は移動に適した軽量なプログラムであり，2.1 節で事例としてあげた侵入者を追跡するプログラムとしても転用できる．この場合，*Slave-S* に美術品の周辺で侵入者を検知させ，*Master* が侵入者と判定した場合に，*Slave-M* に侵入者を追跡させるモデルが実現できる．追跡を行う際に，*Slave-M* は，*Master*，*Slave-S* と離れて，侵入者を追跡し，*Master* による動的配備を必要としない．この際，*Slave-M* は *Master* との通信を実施せずに，独立したプログラムとして処理を実施する．本モデルの実現にあたっては，*Slave-M* が侵入者を逃さず追跡するアルゴリズムを開発する必要があり，*Slave-M* がすべてのノードに自身を複製して，WSN 全体で侵入者を追跡するアルゴリズムが適すると考えられる．

⋆1 100(21.19*2)/2(24*60*60)

⋆2 *Slave-M* が物品の新現在位置を検知するための効率的なアルゴリズムが必要になる．現在の実装ではランダム移動を行うことになる．

提案手法が不向きなアプリケーションとして，移動頻度がきわめて頻繁なアプリケーションが考えられる．たとえば，オフィスに多くの人が存在して，人ごとにアプリケーションを構成したい場合は，提案手法の動的配備の成功率ではアプリケーションの要求を満たさないと考えられる．複数のベースステーションを敷設して LAN と WSN で構成されるネットワークを作成することや，信頼性の高い無線の通信プロトコルを利用するアプローチが考えられる．また，5.1 節の限界 4 で述べたように，提案手法は動的配備中の計測が不可能であり，厳密な定期計測を行うアプリケーションを想定する場合，提案手法は不向きとなる．こうしたアプリケーションを実現する場合，すべてのノードに定期計測を実施できるプログラムをあらかじめ配備しておくことが必要になる．

6. 関 連 研 究

WSN ノードに配備されたプログラムを無線波で更新し，ノードにプログラムを動的に配備する研究に，Agilla [5]，ActorNet [14),15)]，SensorWare [16)] がある．Agilla は，本研究の実装のベースとしたが，複数プログラムの動的配備をサポートしていない．ActorNet は，actor モデル[17)] を WSN に取り入れた研究例であり，プログラムが移動せずに周辺ノードのセンサデータを取得するモデルを提案している．ただし，計測条件を所持した measure actor を事前に各ノードに配備することが求められ，提案手法で解決している複数プログラムの動的配備について解決していない．SensorWare は，Agilla と ActorNet と異なり，MOTE に比較して資源が潤沢な iPAQ で稼動する．想定環境が異なり，複数プログラムの動的配備については提案していない．

WSN ノードに配備されたプログラムを無線波で更新し，ノードにプログラムを静的に配備する研究に Deluge [4]，Mate [18)]，FireCracker [19)] がある．Deluge は，プログラムの更新を行うノードの ID を指定して，ベースステーションから，各ノードにプログラムを配備することができる．ROM に書き込みを要する容量の大きなプログラムを変更できるが，ベースステーション周辺の特定のノードが，プログラムの送受信によって集中して電力を消費するという課題がある．また各ノードで稼動できるプログラムの種類は 1 種類であり，複数のプログラムを段階的に更新・追加する用途に向かない．Mate は，Agilla のベースとなったバーチャルマシンであり，軽量なプログラムでノード上のプログラムを更新することができる．ただし，Deluge と同様に各ノードで稼動させられるプログラムは 1 種類であり，複数のプログラムを段階的に更新できないという課題がある．加えて，特定ノードのプログラムを更新できないという課題もある．FireCracker [19)] は，ある特定のノードにプログラ

ムを送り込み，周辺ノードのプログラムを特定ノードから更新する手法を提案しているが，提案手法のように異なる機能のプログラムを周辺ノードに配備することができない．

WSN 以外のネットワークでは，Mobile Space[20]，石川ら[21] が階層型のモバイルエージェントを提案し，複数のプログラムを一緒に移動させることを実現している．これらの研究は，機能の異なるプログラム間での連携を目的として，階層関連を持つプログラム間でお互いの機能を利用することができる．これらの研究は，計算資源が潤沢な環境と，帯域が広く安定したネットワークを想定し，プログラム間の協調を目的としている．本研究とは，想定環境と，アプリケーションの継続的な実行を目的とした点が異なる．

7. まとめと将来研究

本論文では，プログラム連携を行うアプリケーションを複数構成する際の手法として，1) コンポーネント化されたプログラムによって構成されるアーキテクチャと，2) 複数プログラムを生成して動的に配備する手法を提案し，これらを実現するミドルウェアを構築した．また，シミュレーション環境において，提案アーキテクチャによるアプリケーション構成の有効性と，複数プログラムの動的配備の有効性を示した．提案手法は，複数のプログラムで構成されるアプリケーションを 1 つの WSN 内で複数運用する際の有効な手段の 1 つとなると考えられる．

将来研究として，配備パターンの充実や，配備パターンの状況に合わせた取得がある．本ミドルウェアは，*Master* を受信したノードは *Slave* を配備パターンに従って単純に配備先に送信している．本ミドルウェアは，隣接するノードを管理しているので，隣接ノードの有無は知ることができるが，マルチホップを要する隣接ノードは管理していないため，マルチホップするノードに *Slave* を送信する場合には，そのノードの存在を仮定している[⋆1]．WSN ではノードの周辺環境が変化することが考えられるため，*Slave* の配備先のノードの存在有無を確認することや，状況（配備先のノードの隣接ノード数やバッテリの残量など）を把握してから *Slave* を送信できると WSN の特性に対する適応力が高まる．また，5.1 節で述べたが，*Slave* 間の連携を想定していない．今回提案した *LCA* を構成する *Master*，*Slave* は基本プログラムであり，より複雑なプログラム間連携は将来研究となる．

⋆1 4.3 節の評価時に 2 ホップはなれた隣接ノードにプログラムを送っている．

参 考 文 献

1) Kuorilehto, M., Hannikainen, M. and Hamalainen, T.D.: A survey of application distribution in wireless sensor networks, *EURASIP J. Wirel. Commun. Netw.*, Vol.5, No.5, pp.774–788 (2005).
2) Culler, D.E., Estrin, D. and Srivastava, M.B.: Guest editors' introduction: Overview of sensor networks, *IEEE Computer*, Vol.37, No.8, pp.41–49 (2004).
3) Wang, Q., Zhu, Y. and Cheng, L.: Reprogramming wireless sensor networks: Challenges and approaches, *IEEE Network*, Vol.20, No.3, pp.48–55 (2006).
4) Hui, J.W. and Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale, *SenSys '04, Proc. 2nd International Conference on Embedded Networked Sensor Systems*, pp.81–94, ACM Press, New York, NY, USA (2004).
5) Fok, C.-L., Roman, G.-C. and Lu, C.: Rapid development and flexible deployment of adaptive wireless sensor network applications, *Proc. 25th IEEE International Conference on Distributed Computing Systems* (*ICDCS'05*), Washington, DC, USA, IEEE Computer Society, pp.653–662 (2005).
6) Crossbow mote. http://www.xbow.jp/zigbee-smartdust.html
7) Gelernter, D.: Generative communication in linda, *ACM Trans. Program. Lang. Syst.*, Vol.7, No.1, pp.80–112 (1985).
8) Levis, P., Lee, N., Welsh, M. and Culler, D.: Tossim: Accurate and scalable simulation of entire tinyos applications, *SenSys '03, Proc. 1st International Conference on Embedded Networked Sensor Systems*, pp.126–137, ACM Press, New York, NY, USA (2003).
9) Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D. and Pister, K.: System architecture directions for networked sensors, *SIGPLAN Not.*, Vol.35, No.11, pp.93–104 (2000).
10) Gay, D., Levis, P., von Behren, J.R., Welsh, M., Brewer, E.A. and Culler, D.E.: The nesc language: A holistic approach to networked embedded systems, *PLDI*, pp.1–11 (2003).
11) Pre-packaged versions of agilla. http://mobilab.wustl.edu/projects/agilla/download/pre3/index.html
12) Karp, B. and Kung, H.T.: GPSR: Greedy perimeter stateless routing for wireless networks, *Mobile Computing and Networking*, pp.243–254 (2000).
13) Tossim: A simulator for tinyos networks. http://www.cs.berkeley.edu/~pal/pubs/nido.pdf
14) Kwon, Y., Sundresh, S., Mechitov, K. and Agha, G.: Actornet: An actor platform for wireless sensor networks, *AAMAS '06, Proc. 5th International Joint Conference*

on Autonomous Agents and Multiagent Systems, pp.1297–1300, ACM Press, New York, NY, USA (2006).
15) Kwon, Y., Sundresh, S., Mechitov, K. and Agha, G.: Actornet: An actor platform for wireless sensor networks, Technical Report UIUCDCS-R-2005-2595, Department of Computer Science, University of Illinois at Urbana-Champaign (2005).
16) Boulis, A., Han, C.-C. and Srivastava, M.B.: Design and implementation of a framework for efficient and programmable sensor networks, *MobiSys '03, Proc. 1st International Conference on Mobile Systems, Applications and Services*, pp.187–200, ACM Press, New York, NY, USA (2003).
17) Agha, G., Mason, I.A., Smith, S.F. and Talcott, C.L.: A foundation for actor computation, *J. Funct. Program.*, Vol.7, No.1, pp.1–72 (1997).
18) Levis, P. and Culler, D.: Mate: A tiny virtual machine for sensor networks, *ASPLOS-X, Proc. 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.85–95, ACM Press, New York, NY, USA (2002).
19) Levis, P. and Culler, D.: The firecracker protocol, *EW11*: *Proc. 11th Workshop on ACM SIGOPS European Workshop*, New York, NY, USA, ACM, p.3 (2004).
20) Satoh, I.: Mobilespaces: A framework for building adaptive distributed applications using a hierarchical mobile agent system, *Proc. 20th International Conference on Distributed Computing Systems* (*ICDCS 2000*), Washington, DC, USA, IEEE Computer Society, pp.161–168 (2000).
21) 石川，吉岡，田原，本位田：階層構造制御に注目したモバイルエージェントフレームワークとそのマルチメディア応用，「ソフトウェアエージェントとその応用」特集号，電子情報通信学会，pp.1402–1417 (2005).

付　　録

A.1　配備パターン

図 10 に本論文で使用した配備パターンを示す．図 10 では，*Master* と *Slave*，*Slave-S* と *Slave-M* を別のノードに配備しているが，同じノードに配備することも可能である．

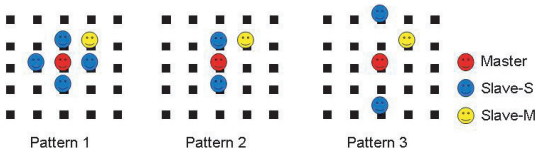

図 10　配備パターン
Fig. 10　Deployment patterns.

A.2　プログラム例

```
// Master Code ---------------------
//リアクション登録（新現在地が報告された場合発動）

BEGIN       pushn ido
            pusht  LOCATION
            pushc 2
            pushc RXN1
            regrxn
//リアクション登録（ID が取得できない場合発動）
            pushn rfi
            pushc 1
            pushc RXN2
            regrxn
//アイドル
IDLE        （アイドル中の任意の処理を記述）
            pushc IDLE
            jumps
//リアクションの発動（新現在地が報告された）
RXN1        pop
            setvar 0
            pushc IDO
            endrxn
//新現在地へ移動
IDO         killslave   //Slave の削除
            getvar 0
            gsmove 1   //  配備パターンの指定
            pushc IDLE
            jumps
//リアクションの発動（Slave-S が ID を取得できない）
RXN2        pushc RFI
            endrxn
//Slave-M を派遣する
RFI         pushn gob
            pushc 1
            slavemloc
            rout
            pushc IDLE
            jumps
// Slave-S Code ----------------------
            startslaves
//ID が取得できない場合の擬似処理
            pushn rfi
```

```
            pushc 1
            masterloc
            rout
//定期的な観測を実施
SENSE       pushc 16
            sleep
            pushc TEMPERATURE
            sense
            pushc THRESHOLD
            cgt
            rjupmc EME
            pushc SENSE
            jumps
//異常時の処理
EME         (温度異常時の任意の処理を記述)
            endslaves
// Slave-M Code ----------------------
            startslavem
//リアクションの登録
BEGM        pushn gob
            pushc 1
            pushcl CLN
            regrxn
//Master から指示があるまで待機
WAIT        wait
//リアクションの発動
CLN         masterloc
            setvar 0
            pushcl CCC
            endrxn
//ベースステーションに報告
CCC         pushloc 1 1
            smove
            pushn loc
            getvar 0
            pushc 2
            out
            endslavem
```

A.3　追加した ISA

表 11 に本研究で追加した ISA を示す.

A.4　プログラムの合計コード容量

提案手法の *Master* のコード容量（*Master_CodeSize*），Agilla のリーダのコード容量（*Leader_CodeSize*）は以下の式 (6)，(7) で記述できる.

表 11　追加した ISA
Table 11　Extended instruction set architecture.

ISA	説明
gsmove	Master の実行状態を保存して，Master のコードを移動させる．引数で配備パターンを指定する.
gwmove	Master の実行状態を保存せずに，Master のコードだけを移動させる．引数で配備パターンを指定する.
killslave	Master の ID を親に持つ Slave を消去する.
masterloc	Master の位置をスタックに積む（Master との通信が可能になる）.
slavemloc	Slave-M の位置をスタックに積む（Slave-M との通信が可能になる）.
startslaves	Master コード中で Slave-S の開始を宣言する.
endslaves	Master コード中で Slave-S の終了を宣言する.
startslavem	Master コード中で Slave-M の開始を宣言する.
endslavem	Master コード中で Slave-M の終了を宣言する.
gettime	時間とプログラム ID を取得（評価とデバッグのため）.
spawnslaves	Slave-S を所定の場所に生成する.
spawnslavem	Slave-M を所定の場所に生成する.

$$Master\_CodeSize = Master\_Process + SlaveS\_CodeSize + SlaveM\_CodeSize + 6 \quad (6)$$

$$Leader\_CodeSize = Leader\_Process + 22 * (N - 1) \quad (7)$$

式 (6) における *Master_Process* は *Master* の処理部分のコード容量，*SlaveS_CodeSize*，*SlaveM_CodeSize* は，*Slave-S*，*Slave-M* のコード容量，6 は提案手法独自の ISA による追加分[⋆1]を示す．式 (7) における *Leader_Process* はリーダの処理部分のコード容量，N は全プログラム数，22 はリーダと他プログラム間の通信に要するコード容量を示す.

動的配備される全プログラムを考慮し，提案手法，Agilla のプログラムのコード容量の合計（*T_CodeSize_P*，*T_CodeSize_A*）は，以下の式 (8)，(9) で表現できる.

$$T\_CodeSize\_P = Master\_CodeSize + (N - 2) * SlaveS\_CodeSize + SlaveM\_CodeSize \quad (8)$$

$$T\_CodeSize\_A = Leader\_CodeSize + (N - 1) * Programs\_CodeSize \quad (9)$$

式 (9) における *Programs_CodeSize* はリーダを除く他プログラムのコード容量を示

⋆1 *Slave* の消去，*Slave* の宣言

す．簡易化のため，*Slave-S* と *Slave-M* のコード容量を同一と仮定し，*SlaveS_CodeSize*, *SlaveM_CodeSize* を *Slave_CodeSize* と定義する．Agilla の他プログラムに *Slave* と同様な動作を行わせる場合，34 byte のリーダとの位置交換と到着通知確認を行うための処理が必要になる．Agilla の他プログラムと *Slave* の処理を同等とすると，式 (8)，(9) は以下の式 (10)，(11) に変換できる．

$$T\_CodeSize\_P = Master\_CodeSize + (N-1) * Slave\_CodeSize \quad (10)$$

$$T\_CodeSize\_A = Leader\_CodeSize + (N-1) * (Slave\_CodeSize + 34) \quad (11)$$

式 (6)，(7) における *Master_Process*，*Leader_Process* は同等な処理を実施することから式 (7) における *Leder_Process* を *Master_Process* で置き換え，式 (6)，(7) を式 (10)，(11) に代入すると，式 (2)，(3)（5.2 節）が導出される．

(平成 20 年 3 月 31 日受付)
(平成 20 年 10 月 7 日採録)

末永俊一郎（正会員）

1997 年東北大学理学部・地球物理学科卒業．1999 年同大大学院理学研究科・地球物理学専攻修了．2001 年，（株）東芝を経て日本ユニシス（株）入社，現在に至る．RFID・センサ等のミドルウェア開発，ユビキタスビジネス開発に従事．

吉岡　信和（正会員）

1971 年生．1993 年富山大学工学部電子情報工学科卒業．1998 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（情報科学）．同年（株）東芝入社．2002 年より国立情報学研究所に勤務，現在，同研究所准教授，エージェント技術の研究，ソフトウェア工学の研究に従事．日本ソフトウェア科学会会員．

本位田真一（正会員）

1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2004 年より同研究所アーキテクチャ科学研究系研究主幹を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．2002 年 5 月〜2003 年 1 月英国 UCL ならびに Imperial College 客員研究員．2005 年度パリ第 6 大学招聘教授．早稲田大学客員教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．ソフトウェア工学，エージェント技術，ユビキタスコンピューティングの研究に従事．IEEE，ACM 等各会員，日本ソフトウェア科学会理事，情報処理学会理事を歴任．日本学術会議連携会員．