情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

センサネットワークアプリケーションの実装支援 API の実装と評価

稲垣彰祐[†1]　森駿介[†1]　梅津高朗[†1,†2]
山口弘純[†1,†2]　東野輝夫[†1,†2]

ワイヤレスセンサネットワーク (WSN) アプリケーションでは，多数ノード間のマルチホップ通信やルーティング処理が頻繁に発生する．実装レベルでは，これらの処理を，送信ノード，中継ノード，受信ノードの内部処理とノード間の送受信処理に変換する必要があるが，一般にそれらの処理に相当するプログラム開発は煩雑である．そこで，本研究ではそれらの送受信処理を簡易に記述できるように幾つかの API を提供すると共に，それらの API を含む記述から NesC 言語のプログラムに変換するトランスレータを開発した．また，開発した API を用いて著名な位置推定アプリケーション DV-Hop やルーティングプロトコル GPSR の開発を行い，プログラム記述の簡潔さや提案手法の有効性の評価を行った．

Design and Development of APIs for wireless sensor network applications

Akihiro Inagaki,[†1] Shunsuke Mori,[†1]
Takaaki Umedu,[†1,†2] Hirozumi Yamaguchi[†1,†2]
and Teruo Higashino[†1,†2]

Implementing wireless sensor network (WSN) applications needs considerable efforts. It usually contains a series of tasks of collecting information and exchanging packets, which should be specified as the behavior of senders, receivers and relay nodes. To alleviate the workload of writing low-level codes, we have designed, implemented, and evaluated APIs for WSNs. They are designed as high-level APIs that hide the detailed behavior of collaborative nodes. We have also designed a translator which converts each API description into the corresponding NesC description used in real environments. Using those APIs, we have implemented DV-Hop and GPSR, well-known position estimation routing protocols, respectively, and evaluated the usefulness of the proposed method.

1. はじめに

災害活動や気象観測など，様々な用途において利用可能な技術として注目を集めているワイヤレスセンサネットワーク (WSN) におけるプロトコルやアプリケーションの開発では，ノードごとに複雑な動作の規定が必要となるなどの実装の煩雑さや，シミュレーションで良い結果が出ても実環境実験を行うと全く異なる振る舞いが見られるなど，実環境実験とシミュレーションの差異などの性能評価の困難さが問題となっている．

そこで，我々は，WSN アプリケーションの設計開発から性能評価までを支援する統合開発支援環境 D-sense[1)] を提案している．D-sense では，既存の様々な WSN プロトコルにおいて頻繁に用いられる処理を API として提供し，実装時にかかる時間と労力を軽減する．これにより，開発者はプロトコルやアプリケーションのアルゴリズムレベルの動作の設計に注力することができる．これに加え，実機用コードからシミュレータ用コードへの変換機能，実機実験で得た情報をシミュレータに反映する機能，実行時にモニタリングを可能とする実環境を対象とした性能評価支援環境などを提供することで，実機とシミュレーションのシームレスな評価実験環境を実現している．しかし，現状の D-sense ではデータルーティング用の API しか提供しておらず[1)]，位置推定など様々な協調型アルゴリズムに対しても利用できる汎用的な API の提供が望まれる．そこで本稿では，D-sense におけるアプリケーション実装支援 API の設計と実装，及び API を用いて記述されたコードを実機用言語である NesC に変換するトランスレータの設計について述べる．これらを利用することで，例えばマルチホップ通信のように複数ノードが協調して行う処理を関数として指定可能となる．

設計した API は，パケット送信 API，ネットワーク情報取得 API，トポロジ構成 API の 3 種類に分類される．パケット送信 API は，パケットを目的地に配送するために各ノードが行うべき一連の処理を，ノードがパケット送信ノード，中継ノード，宛先ノードそれぞれになった場合の処理内容を記述したプログラムコードとして実現している．例えば，マルチホップでのパケット送信処理の実装を行う場合，パケット送信ノード，中継ノード，宛先ノードそれぞれの動作を，各ノードが正しく連携するように詳細に規定するため実装が煩

†1 大阪大学　大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
†2 独立行政法人科学技術振興機構，CREST
Japan Science Technology and Agnecy, CREST

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

雑になるが，WSN アプリケーションではこのような処理は頻繁に用いられる．そこで，このような一連の処理をノード間の通信の詳細を隠蔽して記述可能な API として提供し，実装工程の大幅な削減を実現する．ネットワーク情報取得 API は，パケットの送受信により他ノードの情報などを取得する一連の処理を同様にノード役割ごとのプログラムコードとして実現している．トポロジ構成 API は，複数ノードの協調により木構造などのトポロジを構築する処理，及び構築したトポロジ上で使用できる送受信を伴う処理を記述したプログラムコードとして実現する．さらにこれらの実装支援 API を用いて記述したプロトコル，及びアプリケーションを実機用言語である NesC のコードに変換する API トランスレータの設計を行った．このトランスレータにより，API を用いて記述したプロトコルやアプリケーションの実行コードを得ることができる．本稿では，いくつかの API について API 記述から NesC コードへの変換方法を示している．

開発した API を用い，著名な位置推定アプリケーションである DV-Hop[2]，及び著名なルーティングプロトコルである GPSR[3] の実装を行い，API を用いることで実行用コードが容易に得られることを示した．また，DV-Hop については，センサノードを利用した実験を行い，実験結果と理論値を比較することでその動作の正しさを検証した．これにより，トランスレータによる API 記述から実行用コード記述への変換，及び API の動作が正当であり，API による実装工程の削減が重要であることを示した．

2. 関 連 研 究

Abstract Regions[4] では，アプリケーション開発を単純化するために通信を抽象化し，電波のノード間の接続性，地理的配置，その他のノード詳細によって定義される木構造やクラスタなどの領域をベースとした集団の通信インタフェースを提供している．本稿で説明する API 群の中には，領域 (トポロジ) をベースとした API もあるが，領域などの構築を必要とせずに使用できる API に加え，よりプリミティブな API も提供しており，様々な規模のアプリケーションで利用可能な，より汎用的で実装に近いレベルでの開発サポートを目指している．

Kairos[5] は，複数ノードを含む集合に対し，ノード ID の管理，隣接ノード集合の構築，指定ノードからのデータの取得を行うプリミティブを提供し，集中型の実装を可能とすることで，分散コンピューティングの大域的な振る舞いの表現を可能としている．本稿で提案する API 群は，複数ノードが関わるハイレベルな処理を対象としている点で類似するが，ノード間分散型プログラミングコードとして実現している点で大きく異なる．

SenQ[6] では，スケーラビリティや再利用性の欠如，実用的な様々なシナリオの表現が不可能といったテストベッドにおけるの問題，及びシミュレーションにおける再現の忠実性の問題に対し，センサネットワークのオペレーティングシステムと高い忠実性を持つシミュレーションをエミュレーションにより統合し，様々なイベントやバッテリー，クロックドリフトの正確なモデルや効率的なカーネルの提供を行うことで解決を図っている．D-sense では，実環境実験とシミュレーションの差異を埋めるために，実環境実験で得た情報を反映したシミュレーションの設定ファイルを作成することでより実環境実験に近いシミュレーションを実現している．

3. 統合開発支援環境 D-Sense の概要

D-sense は 1 章で述べたような WSN アプリケーションの開発の際の種々の課題を解決し，少ない手間でアプリケーション開発や性能評価を行うことを可能とする開発支援環境の提供を目標としている．アプリケーション開発者は，D-sense にて提供される実装支援 API を用いてアルゴリズムレベルのプログラムを記述するだけで，シミュレーション，実環境実験の双方でそのアプリケーションの性能評価が可能となる．さらに，実環境実験のログを収集し，シミュレーションの設定に反映することで，実環境実験に近いシミュレーションを行うことができる機能を有している．実環境実験では，モニタリング記述言語を用いて監視条件を記述することで，開発者が行いたいモニタリングを行うことができる．

3.1 シミュレーションと実環境実験のシームレスな連携による実験支援

D-sense では，シミュレーションと実環境実験との間の連携を行い，性能評価を支援するための機能の提供を行っている．連携機能として，(1) 2 者間でのコード共有，(2) アニメータによる実環境実験結果の可視化，及び (3) 実環境のログを元にしたシミュレーションのパラメータ設定，が挙げられる．

D-sense を構成する API トランスレータにより生成された NesC のコードはそのまま無線通信を行う端末 Mote 上で実行可能である上，NesC トランスレータによって無線端末用シミュレータである QualNet[7] 用の C++コードを生成することも可能である．可視化機能を用いて，実環境のセンサノード群の動作や状態を QualNet アニメータで表示できる．

また，パラメータ設定機能は，実環境で得たノードの位置情報や電力残量のログを元に，実環境での設定を継承したシミュレーション環境へのシームレスな移行を実現する．これにより，小規模ネットワークでは実環境で評価し，中規模から大規模ネットワークでは同様の設定でシミュレーションにより評価するといったことが容易に行える．

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

3.2 プロトコルの監視および管理支援

分散環境におけるセンサノード上のプログラムのデバッグは多くの困難を伴う．D-senseはそれにかかる作業負荷を軽減するために，ノードの情報の取得やデバッグの簡易化を可能とするための監視支援を行う．開発者は，無線接続関係，位置関係，センシングイベントといった条件を元にノードの集合を定義し，その集合が行うべき処理を規定する抽象レベルの記述を行うことにより容易に WSN の監視を行うことができる．この機能を利用することで，モニタリングや保守などをより容易に運用できる．

4. 提供する実装支援 API 及びその実現方法

D-sense の提供する主な機能の 1 つが，アルゴリズム設計の支援である．アプリケーション，あるいはプロトコルの開発者は D-sense の実装支援 API を利用してアルゴリズムレベルでコードを記述し，API トランスレータが API を実装コードに置き換えることで，センサノード (Mica Mote) 上で実行可能な NesC によるアプリケーション，あるいはプロトコルの実装を得られる．以降では，API，及び API トランスレータについて述べる．

4.1 実装支援 API

WSN におけるルーティング，及び位置推定などのアプリケーションにおいて頻繁に行われる一般的な処理として，パケットを送信する処理，他ノードの情報を取得する処理，木構造などのトポロジを構成しそのトポロジ上で行う処理などの送信，中継，宛先などの各ノードが協調して行う一連の処理を考慮し，各ノードの処理のタイミングや順序関係に応じてパケット送信 API，ネットワーク情報取得 API，トポロジ構成 API の 3 つに分類を行い，これに基づいて API を設計した．設計した実装支援 API を表 1 に示す．

これらの API はプログラム中でモジュール (関数) として利用でき，後述するトランスレータによって送信，中継，受信の各処理を含む一連の処理のコードに変換される．受信ノードでの多くの API では，処理終了後に受信ノードが実行すべき処理 (事後条件) もあわせて指定することができる．表 1 中の多くの API の引数に指定されている “event”がこれに相当し，その内容をユーザが定義する．以下に主要な API を説明する．

4.1.1 パケット送信 API

表 1 中のパケット送信 API では，マルチホップによりパケット送信を行う処理を API として提供している．これらの API では，送信ノードによるパケット生成と送出，中継ノードでの転送，宛先ノードでの受信，といった動作フローは共通しているが，その動作内容が API ごとに異なる．以下で説明するパケット送信 API はいずれも何も値を返さない．

パケット送信 API	
指定された 1 つのノードへパケットを送信	-> send_by_unicast(destID,pkt,event)
指定された複数のノードへパケットを送信	-> send_by_multicast(destIDs[],pkt,event)
指定ホップ数分パケットをブロードキャスト	-> send_by_broadcast(ttl,pkt,event)
指定領域へパケットを送信	-> send_by_geocast(left,right,top,bottom,pkt,event)
ネットワーク情報取得 API	
隣接テーブルを構成	-> set_neighborTable(time_out_period,event)
隣接テーブルを構成し，定期的に更新	-> periodically_update_neighborTable(cycle)
指定ノードまでの経路を取得	-> get_route(destID,event)
指定ノードまでのホップ数を取得	-> get_hop(destID,event)
指定ノードの位置を取得	-> get_position(destID,event)
指定した 2 ノード間の距離を計算	-> get_distance(destID1,destID2,event)
指定ノードの電池残量を取得	-> get_residual_energy(destID,event)
指定したノードと自身との間の遅延を取得	-> get_delay(destID,event)
指定ノードのパケットロス率を取得	-> get_packet_loss_rate(destID,event)
1 ホップあたりの平均距離を計算	-> get_distance_per_hop(destID,event)
トポロジ構成 API	
指定ノードを根とした全域木を構築	-> build_spanning_tree(rootID,event)
祖先ノードの ID を取得	-> get_ancestor(event)
根ノードへパケット送信	-> send_to_root(pkt,event)

表 1 実装支援 API

“send_by_unicast(destID,pkt,event)”は，宛先ノード ID“destID”によって指定されたノードに対してパケットへのポインタ “pkt”で指定されたパケットをユニキャスト送信する API である．少量のデータ送信を想定し，指定ノードまではデータリンク層ブロードキャストを繰り返すことで到達するシンプルな方針を採用している．これに対し，各ノードは，もし自身が送信ノードならパケットを送出し，中継ノードなら受信パケットの宛先 ID とパケット ID を判定し，自身が宛先でなく，かつ初めて受信するパケットであれば転送し，宛先ノードはパケット受信後，ユーザが定義する関数 “event”で指定された処理を実行するようなコード片として実現する．“send_by_multicast(destIDs[],pkt,event)”は，“send_by_unicast()”で行われる処理を複数回行い，複数の宛先ノード ID を格納する配列 “destIDs[]”で指定される宛先ノード群へ指定パケットを送信する API であり，現状では “send_by_unicast()”で行われる処理を複数回行うことで実現している．“send_by_broadcast(ttl,pkt,event)”は，パケットへのポインタ “pkt”で指定されたパケットのブロードキャストをホップ数 “ttl”によって指定されたホップ回数分だけ行う API である．“send_by_geocast(left,right,top,bottom,pkt,event)”は，位置を表す “left”，“right”，“top”，“bottom”により四隅の座標を指定された領域内

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

のノードへパケットを送信する API である．指定領域に含まれる各隣接ノード，そのような隣接ノードがない場合は指定領域の中心座標に最も近い隣接ノードへ送信し，これを繰り返す．

4.1.2 ネットワーク情報取得 API

ネットワーク情報取得 API では，パケット送信 API と同様の方法により対象ノードに対してクエリパケットを送信し，クエリパケットを受信した対象ノードが要求された情報を載せたレスポンスパケットをマルチホップで返信することでネットワーク情報の取得を行う一連の処理を API として提供している．このような処理では，パケット送信 API と同様にクエリが宛先ノード群に転送され，各宛先ノードは処理内容に応じたレスポンスを返す．中継ノードがこれを転送し，送信ノードが受信することでネットワーク情報を取得する．パケット送信 API と同様にこれらの各ノードごとに行われる，送信，転送，受信といった処理の詳細を API ごとに規定している．

“set_neighborTable(time_out_period,event)”は，隣接テーブルを構築する API である．まず，送信ノードがデータリンク層ブロードキャストを行い，これを受信した隣接ノードは，ノード ID や位置など自身の情報を載せたレスポンスを送信ノードへユニキャストで返す．送信ノードにおいてはブロードキャスト実行から，タイムアウト時間 “time_out_period”によって指定された時間が経過した時点で届いている返信を集め，隣接テーブルを作成し，それを返り値とする．なお，“time_out_period”の値が 1024 の時，経過する時間は 1 秒となる．“periodically_update_neighborTable(cycle)”は，“set_neighborTable()”と同様に隣接テーブルを構築するが，こちらは周期 “cycle”で指定される周期ごとにブロードキャストを行い，定期的に隣接テーブルの更新を行う．そのため，何も値を返さない．隣接テーブルの構築完了を待たずに API の次に記述されている処理に移るため，構築完了前の隣接テーブルへのアクセスによるエラーが発生する可能性がある．また，一定周期ごとに動作する仕様のため，他の API とは異なり終了時処理を行う関数の定義を行う必要がない．

“get_route(destID,event)”は，宛先ノード ID“destID”によって指定された宛先ノードまでの経路を取得し，返す API である．“send_by_unicast()”と同様の方法で宛先ノードへクエリを送信し，その経路を ID の集合としてパケットに保存しておく．その保存しておいた経路を列として含むパケットを送信ノードに (逆経路で) 返信する．複数の経路がある場合はそれらが全て返信される．“get_hop(destID,event)”，“get_position(destID,event)”，“get_distance(destID1, destID2,event)”，“get_residual_energy(destID,event)”は，“get_route(destID,event)”と同様に宛先ノードまでのホップ数，宛先ノードの位置，送信ノードとの 2 点間の距離，宛先ノードの電池残量をそれぞれ取得し，返す API である．“get_delay(destID,event)”は，宛先ノード ID“destID”により指定される宛先ノードと自身との間のパケット送受信に伴う予測遅延を返す API である．宛先ノードに対し，“get_route()”と同様の方法でパケットの送受信を行うことでラウンドトリップタイムを計測し，それを遅延として返す．“get_packet_loss_rate(destID,num,event)”は，宛先ノード ID“destID”により指定される宛先ノードに対して送信したパケットのロス率を返す．届いた回数については，ユニキャストで宛先ノードに問い合わせる．“get_distance_per_hop(destID,event)”は，1 ホップあたりの距離を見積もって返す API である．この API では，自分と宛先ノードはそれぞれ自身の位置を知っていることを前提としている．まず，宛先ノード ID“destID”によって指定された宛先ノードのホップ数と位置を，“get_hop()”や “get_position()”と同じ方法で取得し，1 ホップあたりの距離を見積もる．

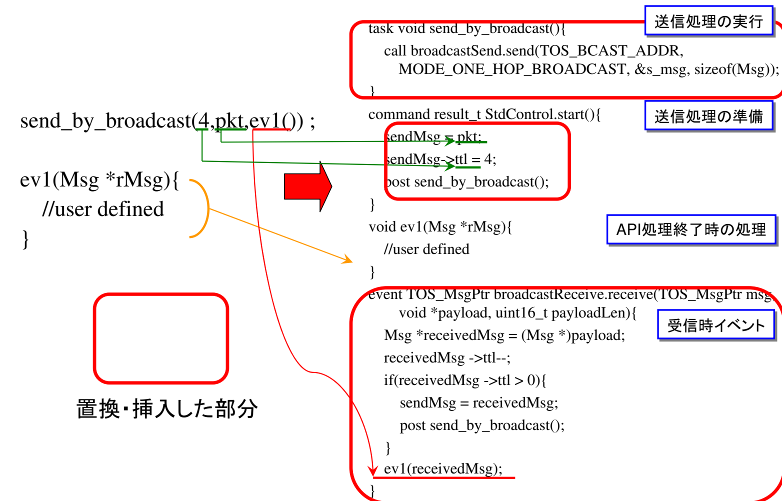

図 1 send_by_broadcast() の変換概略図

4.1.3 トポロジ構成 API

トポロジ構成 API は，複数ノードが協調して木構造などのトポロジを構築する．

“build_spanning_tree(rootID,event)”は，根ノード ID“rootID”によって指定されたノードを根とした，ネットワーク的に連結しているノードを全てカバーする全域木を構築する

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

API である．各ノードが親ノードと子ノードを隣接テーブル内に記録しておくことで木構造を実現する．従って，この API は隣接テーブルが作られていることを前提としている．構築方法は，まず，送信ノードが根ノードへクエリを送信し，根ノードは自身の ID を載せたパケットのブロードキャストを行う．これを受信したノードは，送信ノードを親として隣接テーブルに登録し，自身の ID を載せたパケットをブロードキャストする．ただし，同じシーケンス番号のパケットを 2 回受信した場合はそれらは転送しない．あるノードからパケットを受け取った場合，そのパケットに応答パケットを返す．これにより，子ノードが隣接テーブルに登録される．以上のブロードキャストを繰り返すことで連結しているノードを全てカバーした全域木を構築する．“send_to_root(pkt,event)”は，パケットを根ノードへ送信する API である．“get_ancestor(event)”は，木構造において自身よりも “祖先”にあたる全てのノードの ID を返す API である．根ノードにクエリを送り，経由した全ての ID を載せたパケットを元のノードまで木にそって返信する処理を行う．

4.1.4 API の利用制約

本稿で提案している API 記述を元に出力する言語 NesC はマルチタスク処理が行えない仕様であるため，応答パケットを待ちながら他の処理を行うことができない．そのため，NesC ではパケット受信時にイベント関数が呼び出され，受信後の処理はそのイベント関数に記述する．また，表 1 中の実装支援 API には，送受信を含む一連の処理を提供している API も存在するが，これらの多くの API の動作はパケット送信に対する応答パケットが返ってきた時点で完了する．そのため，これらの API で情報を取得した後の処理は，“event”というテンプレートとして記述し，NesC の受信イベント内で呼び出される関数として変換される．API での一連の処理の完了直後に “event”が呼び出される．つまり，ユーザは API での処理完了を前提とした処理は “event”内，もしくはそれ以後に実行される部分に記述しなければならない．そのため，関数の引数などのようにネストしての API の使用，for 文や while 文などによる繰り返しの中での API の使用についても正しく処理が行われることは保証されない．例えば，ネットワーク情報取得 API により他ノードまでのホップ数などの情報を取得し，それを利用した処理を行う場合，“event”より先にそのような処理が行われた場合，正常な動作は保証されない．

4.2 API トランスレータ

API トランスレータは，4.1 節で述べた API を用いて記述された部分を検出し，あらかじめ用意されているコードを調整して置き換えや挿入を行うことで，NesC で記述された実行プログラムを生成する．

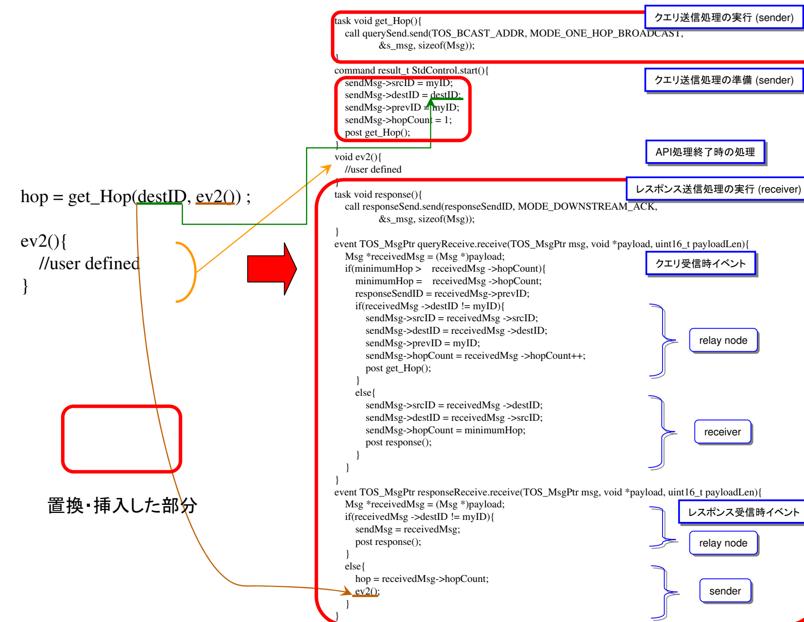

図 2 get_Hop() の変換概略図

NesC では送信準備処理，送信処理の実行関数，受信時イベントによって通信を記述する．本手法では，それぞれの段階で適切な処理を行うことで API の機能を実現する．具体的な変換方法は，以下の通りである．まず，API が使用する関数や参照している大域変数の宣言が必要なので，それらを挿入する．次に送信準備処理として，API 記述の部分を，最初のパケット送信以前に実行すべき処理に対応したコードと置き換える．その後，送信処理関数の呼び出しを挿入する．それに合わせて，送信処理の実行関数及び受信時に実行されるイベント関数など，API としての処理を動作させるために必要な関数の挿入を行う．受信時イベント関数には，転送処理なども含まれる．以上の置換や挿入などによって，全て NesC で記述されたプログラムが生成される．1 つの API が複数使用されている場合は，それぞれの呼び出しごとに API に ID を割り振り，変換後のコード中のパケット送信処理，中継処理，受信処理において ID を元に処理を分けることで処理を実現する．

また，NesC においてはメッセージ受信時にどのイベントが実行されるかはインターフェー

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

スと呼ばれる識別子によって決まる．例えば，図 1 においては，変換後の上部の "broadcastSend"，および末尾の event の "broadcastReceive" がインターフェースであり，これらが外部のコンフィグレーションファイルで接続されている．そのため，"broadcastSend.send()" で送信したメッセージの受信時には "broadcastReceive()" が実行される．トランスレータは一般に複雑になるこの外部ファイルの生成も行う．

図 1 は，指定ホップ数分ブロードキャストを行うパケット送信 API である send_by_broadcast() のトランスレータによる NesC 記述のコードへの変換概略図を示している．

図 1 を含め，以降に例示する概略図においては，関数や変数の宣言部は省略している．API で記述されている部分を，送信パケットの内容の定義と送信処理を行う関数の呼び出しの処理で置き換えており，API の引数の値の適切な場所への反映も行っている．さらに，置き換えた箇所を含む関数の上部に送信処理を行う関数を挿入し，受信処理を行うイベント関数をプログラムの下部に挿入している．受信ノードが宛先ノードでない場合は，転送処理を行い，宛先ノードの場合は API 処理完了時の処理を行う．

図 2 では，指定ノードまでのホップ数の取得を行うネットワーク情報取得 API である get_Hop() のトランスレータによる NesC 記述のコードへの変換概略図を示している．図 2 においても，図 1 と同様に置換，挿入を行っている．パケット送信 API では大まかな一連の処理の流れがパケット送信，中継，受信であったのに対し，ネットワーク情報取得 API はそれに加え，レスポンスパケットの送信，中継，受信の処理の記述もあるため，挿入コード量が図 1 と比べて多くなっているが，変換方針は同じである．

図 3 では，1 つの API が複数用いられている例として，パケット送信 API である send_by_broadcast() が複数記述されている場合の変換概略図を示している．図 3 中では if 文中で API が用いられているが，API 部分をそのまま送信処理までのコードに置き換えることで正常な処理の流れを維持している．使用されている 2 つの send_by_broadcast() にそれぞれ ID を振り，ID ごとに送信，中継，受信の各処理を分けている．

5. API を用いた実装と実環境実験

設計した実装支援 API，及び API トランスレータによりアプリケーションの実環境実験による性能評価が容易に行えることを示すために，既存の著名な位置推定手法 DV-Hop[2]，及びルーティングプロトコル GPSR[3] について，API を用いた実装を行った．また，API トランスレータによる変換，及び API の動作の正当性，及び API を用いた実装プロセスの労力の削減の重要性を示すために，DV-Hop について，API を用いた実装から API トランスレータを用いて得た NesC コードを用いた実環境実験を行った．

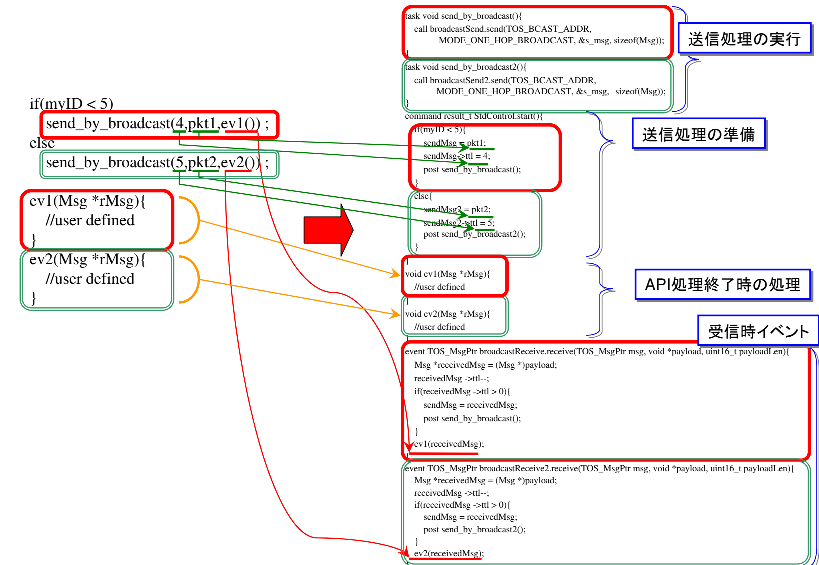

図 3 API が複数含まれる場合の変換例

5.1 API を用いた実装のコード量検証

DV-Hop 及び GPSR のそれぞれの実装支援 API を用いて実装したコードの行数，それを API トランスレータによって変換した後の NesC コードの行数，実装に用いた API の種類と総数を表 2 に示す．

(1) DV-Hop

DV-Hop は，各ノードが，座標が既知である基地局 (以下，ランドマーク) との距離をホップ数から推定し，自身の座標を求めるアプリケーションである．

DV-Hop では，ランドマークの位置やホップ数を取得したり，1 ホップあたりの距離を計算し送信する処理が行われるため，表 2 で示すような API を用いて実装を行った．上記のような情報取得を行う処理は複数ノードに対して行うため，用いた API の総数が多くなっている．従って，それに伴い変換後のコード量が大幅に増加しており，API 記述によりコード量が大きく削減されていることが分かる．

© 2010 Information Processing Society of Japan

Vol.2010-SLDM-144 No.52
Vol.2010-EMB-16 No.52
Vol.2010-MBL-53 No.52
Vol.2010-UBI-25 No.52
2010/3/27

	API 記述のコード量	変換後のコード量	用いた API	用いた API の総数
DV-Hop	123 行	1613 行	send_by_broadcast(), get_distance_per_hop() get_hop(), get_position()	12
GPSR	162 行	427 行	periodically_update_neighborTable() send_by_unicast()	3

表 2 API による実装の検証

(2) GPSR

GPSR は，位置情報ベースのルーティングプロトコルである．隣接ノードのうち，宛先ノードに最も近いノードへパケットを送信し，これを繰り返して宛先ノードへ到達するグリーディ法を用いるが，隣接ノードに自身よりも宛先ノードに近いノードが無い場合があり，この時に「右手の法則」によって，平面グラフ上を時計回りに回るように次のノードを選択していくことで宛先ノードへ到達する．

GPSR において通信を伴う処理は隣接テーブルの構築，及び指定ノードへのパケット送信のみであるため，表 2 で示すように用いた API の総数は少ない．それでも，変換後のコード量と API 記述のコード量を比較すると 2 分の 1 以下に抑えられており，実装の工程が削減されていると言える．

5.2 実環境実験

API を用いて実装した DV-Hop を API トランスレータによって NesC 記述に変換したものを用いて実環境実験を行った．ノード配置については，ノードの数に限りがあったため，16 ノードを図 4 に示すような配置で実験を行った．16 ノード中ランドマークは 4 ノードとした．推定を行うノードが 3 つのランドマークで構成される三角形の外部にあると結果の座標が不正確になりやすいため，ランドマークは，正方領域の四隅に配置した．残りのノードは図 4 のように均一に配置し，各ノードはランドマークから定期的に送信される情報を基に位置推定を行った．そして，図 4 で示すような，理想的なノード間の接続関係であった場合の各ノードの DV-Hop による推定位置をあらかじめ理論値として計算しておき，実験結果の推定位置をそれと比較することで評価を行った．

表 3 は実験結果の推定位置，理論上の推定位置，実験結果の推定位置と理論上の推定位置の誤差を示している．表 4 は実験時に各ノードが計算に使用したランドマークとそこまでのホップ数を示している．あらかじめ計算しておいた理論上の推定位置は理想的な条件で計算を行ったため，4 つのランドマークで構成される正方形の中心に近い 4 つのノードは実

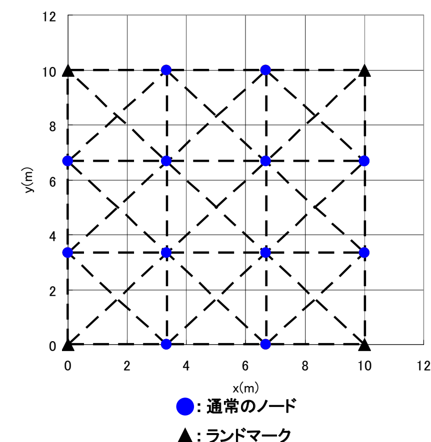

図 4 ノード配置およびノード間の想定接続関係

ノード ID	実験結果の推定位置	理論上の推定位置	実験結果と理論値の誤差
1	(5, -9.52)	(3.33, 0.56)	10.22
2	(5, 2.22)	(6.67, 0.56)	2.36
3	(1.5, 5)	(0.56, 3.33)	1.92
4	(5, 5)	(3.33, 3.33)	2.36
5	(7.04, 2.96)	(6.67, 3.33)	0.53
6	(9.44, 5)	(9.44, 3.33)	1.67
7	(1.3, 6.98)	(0.56, 6.67)	0.8
8	(0.56, 5)	(3.33, 6.67)	3.24
9	(13.92, 5)	(6.67, 6.67)	7.44
10	(10.83, 8.5)	(9.44, 6.67)	2.29
11	(5, 11.48)	(3.33, 9.44)	2.63
12	(5, 10.83)	(6.67, 9.44)	2.17
平均値			3.13

表 3 実験結果

© 2010 Information Processing Society of Japan

ノード ID	LM1	hop	LM2	hop	LM3	hop
1	L1	1	L3	5	L4	5
2	L2	2	L3	3	L4	3
3	L1	1	L2	2	L4	2
4	L1	2	L3	2	L2	2
5	L2	2	L4	3	L1	3
6	L2	1	L4	1	L3	3
7	L3	1	L1	3	L4	4
8	L1	1	L4	3	L2	3
9	L4	1	L1	4	L3	4
10	L4	1	L2	2	L1	3
11	L4	1	L4	1	L1	3
12	L4	2	L1	3	L2	3

表 4 実験時に各ノードが計算に用いたランドマーク情報 (LM はランドマーク)

際の配置と一致するなど，いずれのノードも実際のノード位置のすぐ近くの座標となった．しかし，表 3 で示すように実験結果の推定位置と理論上の推定位置の誤差はやや大きい値となった．原因を調べたところ，表 4 に示すようにランドマークから 5 ホップで受信したデータを用いていたり，通信できないはずのノードと通信していたりするなど想定外の通信がなされている例が多くあることが分かった．このような想定外の通信は，実環境であるために電波到達距離が不安定であり，かつ各端末の電波到達距離には差があることと，複数の通信が干渉を起こしていることが要因となり発生していると考えられる．

想定していた最短経路ではなく，電波到達距離の不安定さや通信の干渉による迂回経路が選択されたが，そのもとで DV-Hop 全体の動作が正しいものであると確認できた．従って，API 記述のトランスレータによる変換は問題なく行われていると言える．また，この性能評価で得られた結果のように，電波伝搬状況に応じて起こる電波到達距離の不安定さや通信干渉といった問題は実験によって判明するものであり，性能評価実験を行うために，実装のプロセスの労力を削減することは重要である．

6. あとがき

本稿では，マルチホップ通信を行う WSN のプロトコル及びアプリケーション開発を支援するための API，及びその API を無線端末用言語 NesC のコードへ変換するトランスレータの設計及び実装について述べた．特に，メッセージ送受信による複数ノードの協調処理を対象に，パケット送信 API，ネットワーク情報取得 API，トポロジ構成 API といった上位レベルの API の設計，実装を行った．

評価については，著名なプロトコルの実装を通して API を用いた場合のプログラム記述の簡潔さならびに動作の検証を行った．これにより，トランスレータによる API 記述から実行用コード記述への変換，及び API の動作が正当であり，API による実装工程の削減が重要であることを示した．今後は，設計対象のアプリケーション分析をさらに行い，クラスタなどの様々なトポロジの構成やデータ収集などを行う API を設計し，より多くの WSN プロトコルの設計開発を支援することを目指していく．

参 考 文 献

1) 森 駿介, 梅津 高朗, 廣森 聡仁, 山口 弘純, 東野輝夫：ワイヤレスセンサネットワークの設計開発支援環境 D-sense，情報処理学会論文誌， Vol.50, No.10, pp.2556–2567 (2009).
2) D. Niculescu and B. Nath: DV Based Positioning in Ad Hoc Networks, *Journal of Telecommunication Systems*, Vol.22, pp.267–280 (2003).
3) B. Karp and H. T. Kung: GPSR: Greedy Perimeter Stateless Routing for Wireless Networks, *Proc. of the 6th Annual International Conference on Mobile Computing and Networking(MobiCom 2000)*, pp.149–160 (2000).
4) M. Welsh and G. Mainland: Programming Sensor Networks Using Abstract Regions, *Proc. of the 1st Conference on Symposium on Networked Systems Design and Implementation(NSDI 2004)*, pp.29–42 (2004).
5) Ramakrishna Gummadi, Omprakash Gnawali, and Ramesh Govindan: Macroprogramming Wireless Sensor Networks Using Kairos, *Proc. of the IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2005)*, pp.126–140 (2005).
6) Maneesh Varshney, Defeng Xu, Mani Srivastava and Rajive Bagrodia: SenQ: A Scalable Simulation and Emulation Environment for Sensor Networks, *Proc. of the 6th IEEE International Conference on Information Processing in Sensor Networks(IPSN 2007)*, pp.196–205 (2007).
7) Scalable Network Technologies, Inc.: QualNet Simulator. http://www.scalable-networks.com/.

© 2010 Information Processing Society of Japan