修 士 論 文

大規模センサネットワーク環境における
時間同期プロトコル

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

滝澤 啓

2010 年 3 月

修 士 論 文

大規模センサネットワーク環境における時間同期プロトコル

指導教員　Defago Xavier 准教授

審査委員主査　Defago Xavier 准教授
審査委員　青木 利晃 准教授
審査委員　井口 寧 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

810035 滝澤 啓

提出年月: 2010 年 2 月

概 要

本稿では，隣接ノード同士での最小限のメッセージ量でネットワーク全体の時間を同期させるプロトコルを提案する．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景

近年の無線通信技術の発展と小型端末の低コスト化により，無線通信機能を搭載した小型のセンサデバイスによって構成されるネットワーク[1]が注目されている．このセンサノードを広範囲に分散配置させ，それぞれのセンサが取得したデータを集約することで災害予測や防犯，あるいは交通制御などの科学用途が想定され．広義の意味ではロボットによる動作制御もその活用例の 1 つであり，その用途も規模も多岐に及んでいる．

例えば，各センサノードの多点同時計測による環境モニタリングを考えると計測する規模の大きな対象領域へセンサを数百，あるいは数千ノード配置するケースも想定される．

この複数のセンサノードによって構成されるネットワークを分散処理システムと捉え，既存の分散アルゴリズムを適応させる研究が盛んに行われている．その中でも前述のようなアプリケーションを構築する際には，時間的な整合性を保証することが必要となり，何らかの手段によって各ノードの時間を合わせなければならない．

しかし，この問題を分散システムとして考える場合，各ノードが絶対的な時間を保持するグローバルな時計の参照を行うことが出来ないという前提が存在する，そのため，必然的にシステム内で各ノードが参照出来る対象は自分以外のノードの時間となる．つまり，GPS や電波時計などの外部装置を用いずにネットワークコミュニケーションを利用して，それぞれのノードが持つ時計を合わせるメカニズムが必要となる．現実世界でも，衛星からの電波が物理的に届かない屋内や，そもそも衛星が存在しない月など，時間を合わせるためのインフラストラクチャが整っていない局所的な環境での同期手法の工夫は重要な問題であろう．それに加えて，高い信頼性への要求から，規模の大きさによるスケーラビリティや長期運用に耐えうる耐故障性を考慮し，自律的に安定した時間同期システムの実現が求められている．

1.2　研究目的

本稿の研究目的は，大規模なセンサネットワーク上での長期的な環境計測や協調動作を想定して，ネットワークコミュニケーションを利用した自己安定型時間同期手法の提案である．この提案プロトコルは三つのプロトコルによって構成される．定期的に送信を行う

[1]以降センサネットワークと呼ぶ

pulse プロトコル，時間同期アルゴリズム，リファレンスとなるリーダを選定するリーダエレクションプロトコルの三つである．最終的に，これらのプロトコルを統一する．時間同期のプロセスを行う過程で一時故障が発生した際にも安定してシステム全体が動作し，リファレンスノードの動的決定による自律的に安定状態に収束することを実現する．最終的な目標として三つのプロトコルを 1 つのシステムとして提案し，Java 向け分散フレームワーク Neko によるシミュレーションによる実験と評価を行い，長期的な運用に耐えられるシステムを目指すための考察を行う．

1.3 研究の流れ

第 2 章はセンサネットワークの特徴を述べる。第 3 章で時間同期の関連研究を述べ，本研究の立ち位置を明確にする．第 4 章でシステムモデルを定義し，第 5 章で提案プロトコルについて述べる．第 6 章で Neko によるシミュレーションを行い，7 章でまとめと今後の指針を述べる．

第2章　センサネットワークの特徴

本章ではセンサネットワークにおける時間同期の既存研究について述べる．まず，対象とするセンサーネットワークを取り巻く環境や制約について述べ，関連研究として，GPS や電波時計などの外部装置を用いた時間同期の手法と，ネットワークコミュニケーションを用いた場合の代表例として本研究のベースとなる NTP のそれぞれに関して述べることで本研究の立ち位置を明確にする．

2.1　センサネットワークについて

センサネットワークとは，多数のセンサノードによって構成されるネットワークである．センサノードは小型の筐体に通信機能，計算機，そして電源ユニットを備えたデバイスのことを指す．近年，その中でも特に強い注目されているのがワイヤレス通信機能をもつワイヤレスセンサネットワーク (Wireless Sensor Network) である．このセンサノードは，搭載されたセンサーによって現実世界のデータを収集し，自身の通信レンジ内のノードと互いに通信を行いながら自律的にネットワークを構成する．それぞれのノードが収集したデータは自分以外のノードを経由する（マルチホップ機能）ことでシンクノードに集約され，外部のユーザに運ばれる．(2.1 図参照)

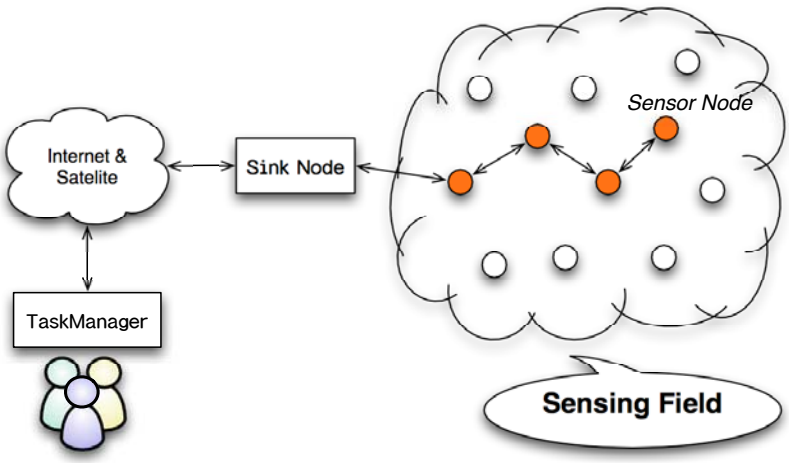

図 2.1: ワイヤレスセンサネットワークの概要 [3]

シンクノードとは，実際にセンシングするセンサノードとは異なり，外部のネットワークや送信施設にたいしてのゲートウェイの役割を果たす．本稿では，このワイヤレスセンサネットワークを前提とする．

特徴

Estrin ら [1] はワイヤレスセンサネットワークを構成するノードが期待される対応特性を以下のように述べている．

- ケーブルレス (Untethered)
  ノードに搭載される電源ユニットと電源供給源とケーブルで接続されていないため，バッテリー容量は有限である。そのため，限られたノードの電力を効率的に利用し，長期活用するためにも計算処理や通信機能を最適化し，最小限のコストで行う必要がある．

- アドホックな設置 (Ad Hoc deployment)
  ノードを設置する対象領域に既存のネットワーク基盤がある保証がない．例えば，上空から散布する場合など，落下地点が予測出来ないため，ノード間での自律的なネットワーク形成が必要とされる．

- 動的変化 (Dynamic Topology changes)
  不安定な環境での活用が想定されるため，ノードの移動や故障によるトポロジの動的な変化に柔軟に対応させることが必要とされる．

- メンテナンスフリー (Unattended operation)
  一度配備されたノードに対してユーザが直接手を加えることは難しいため，ノード自身が変化に応じた挙動の制御を行う必要がある．

加えて，ハードウェアとしての進化により，U.C.Berkley で進められている Smart Dust プロジェクト[1]のような，ミクロサイズのセンサノードの研究も行われている，それに従い，低コストで大量のノードを配置するアプリケーションへの活用が進めらており，上記の特性が更に強く求められている．具体的には，環境や生態系の観測や物流や工場の生産管理など，農業や工業などの産業分野への応用が進められてきたが，現在では，超小型のセンサノードを人間の身体に装着するヘルスチェッキングや防犯・防災などの社会基盤に位置づけられ，ユビキタスコンピューティングにおける必須技術である，

[1]SmartDust: http://robotics.eecs.berkeley.edu/ pister/SmartDust/

MANETとの差異

これまで述べたセンサネットワークを実際に応用するには，既存のネットワーク技術では実現できない問題点が存在する．これまで，センサネットワークの研究領域と比較的類似するものとしてマルチホップ間の P2P(ピアツーピア) 通信によるモバイルアドホックネットワーク，通称 MANET (Mobile Ad-hoc Networks)[4] の研究が盛んに行われている．例えば，インターネットの利用技術の標準化団体である IETF は MANET の Working Group[2]を設立し，プロトコルや標準化について議論を行っている．この MANET とセンサネットワークを明確に区別し本研究の立ち位置を明確にするために，本稿においては Akyildiz ら [3] が述べている区別を採用する．以下に MANET との違いを述べる．

- センサネットワークはノード数が膨大な場合が多い
- センサネットワークは配置密度が高い場合が多い
- センサノードは故障する可能性が高い
- トポロジーが頻繁に変化する
- ユニキャストによる P2P 通信ではなく，ブロードキャスト中心の通信を行う
- センサノードはハードウェアの制約が多く，電力やリソースが有限である

以上が MANET と異なる点であるが，ノードの Mobility によるトポロジの変化など一部重複する点においては対象アプリケーションによって異なるため，議論の余地がある．(本稿におけるモデルは第 4 章にて述べる)

考慮すべき要因

これまでセンサネットワークの特徴と MANET との差異を明確にした．これらを踏まえて，本研究の対象であるセンサネットワークを実際に設計して実現する際に考慮すべき要因を挙げる．

1. スケーラビリティ (scalability)
   アプリケーションが対象とする領域に応じて，ノードの設置数が爆発的に多くなる可能性がある．マルチホップ通信によって相互通信するセンサネットワークにおいては，全ノード数と，ある地点におけるノードの設置密度の要求がそれぞれ異なる．そのため，ノード数の増加に柔軟に対応した通信制御プロトコルが必要である．

[2]http://www.ietf.org/dyn/wg/charter/manet-charter.html

2. 耐故障性 (fault tolerance)
   センサノードの故障や設置環境の変化により，ノードの位置や存在が変化してしまう危険性が高い．具体的には電力消費による動作停止や風や天候による局所的なノードの移動が発生した場合でもネットワークとして性能を最大限維持することが求められる．

3. ネットワークトポロジ (network Topology)
   センサネットワークのトポロジーが頻繁に変化することを考慮すべきである．設置時の物理的配置，設置後の故障や電波到達範囲の変化，ノード追加時の再設定などに対応できる必要がある．

4. ハードウェアの制約 (hardware constraints)
   限られたリソースによる計算のオーバヘッドを考慮した設計を行うべきである．

5. コスト (production costs)
   1. と関連して，ノード数のスケールによって一台あたりのコストが大きければ大きいほど，指数関数的にコストは増大する．そのため，一台あたりのコストをできるだけ抑える必要がある．

以上の制約的要因を踏まえながら，応用領域によって異なる要求に汎用的に最適な設計を行う必要がある．本稿で提案する時間同期プロトコルは，上記のスケーラビリティと耐故障性，それに伴うトポロジの変化という問題点に焦点を当てたものである．すなわち，ノード数の増加に対応し，故障によるネットワーク構造の変化が発生する環境を前提とする．

2.2 対象アプリケーション

本研究は森林や氷河，あるいは火山などの大規模な自然環境の多点同時計測を対象とする．すべてのノードが同じタイミングでセンシングによるデータ収集において，時間の誤差が生じているとその時間におけるデータに整合性がなくなってしまうことからも時間同期は非常に重要な技術である．

観測領域が巨大であり，センサノードを高密度かつ広範囲に配置したい場合，配置による人的コストを考慮し，センサノードをヘリコプターや飛行機の高所から空中散布する方法が一般的である．図 2.1 が空中散布による配置のイメージである．一点からの単純な拡散配置によっても落下する位置が一様になるような確率的配置が数多く提案されている．

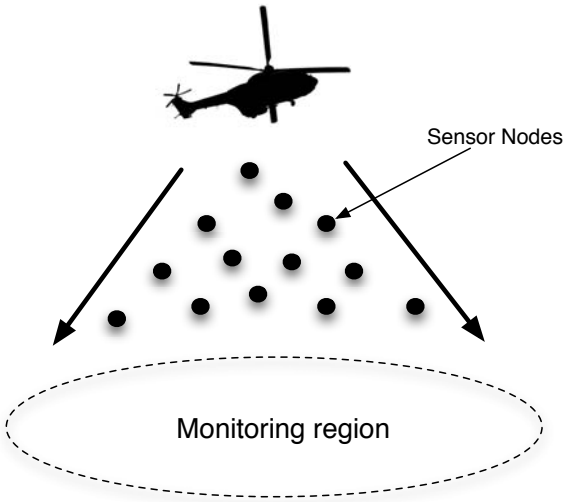

図 2.2: 空中散布による単純拡散配置のイメージ

この方法は空中から観測対象に対して散布するため，ノードが落下する位置を特定できず，初期状態が把握しにくい．さらに，落下によるセンサノードの欠落や自然環境の変化によるノードの移動，あるいは故障が発生する．これはネットワークトポロジの変化を意味し，それに伴う最適なルーティング情報が変化してしまう．

本研究の動機は，こうした自然環境によるネットワーク構造の変化の影響を最小限にするために，ノードの位置やルーティング情報に依存することなく，自然環境による動的なネットワーク構造の変化に対応することである．従って，本研究ではセンサノードの配置法には言及せず，対象領域にランダムに配置されているものとしている．これによって，センサノードの位置情報や配置方法を限定しないことで問題点を明確にしている．

第3章　関連研究

時間同期手法に関しては，冒頭で述べた通り分散システムの信頼性やセキュリティの面でも重要な要素である．しかし，前項で述べたセンサネットワークの特徴と制約やアプリケーションによって許される同期精度が異なることから，最適な同期手法を選択する必要があり，多くのワイヤレスセンサネットワークのための同期プロトコルが提案されている [5]〜[10]．時間同期の手法は大きく分けて，特別な外部装置を用いた手法とネットワークコミュニケーションによるノード間の相互通信によって実現する手法に大別される．本章ではそれぞれの特徴と問題点を分析し，本稿の提案手法の意義を明確にする．

3.1　外部装置を用いた時間同期

Global Positioning System

自律的に各ノードが正確な時間を知るための方法はGPS(Global Positioning System) による時間同期が一般的である [5]． GPS は地球を周回する 4 つ以上の人工衛星からの測位信号を利用した逐次近似法によって正確な位置と時間の誤差を算出する．GPS 衛星にはセシウム原子時計と呼ばれる時計が搭載され，誤差のない正確な時間を所持している．この原子時計と同期させるために複数の GPS 衛星からの信号を受信機に送ることで広範囲に渡る同期を実現する．受信機の位置を (x, y, z)，衛星 i の位置を (x_i, y_i, z_i) ，光速 c，そして GPS 受信機と衛星 i との距離と誤差をそれぞれ r_i，δ とすると 式 (3.1) が導かれる．

$$r_i = \sqrt{(x_i - x)^2 + (y_i - y)^2 + (z_i - z)^2} + c\delta \tag{3.1}$$

式 (3.1) によってそれぞれの (x, y, z, δ) が求められ，位置だけでなく衛星が持つ原子時計との誤差を通信遅延の影響を排除して算出することができる利点がある．

電波時計

近年では安価で小型化された電波時計をセンサノードに搭載することでネットワークコミュニケーションを用いずに時間同期を行う手法も実用化されている．この手法では送信

局から受信された時間をそのまま利用するため，送信局からの通信による遅延を修正できないため，環境による遅延の影響を直接受けやすい問題点がある．

3.2 ネットワークコミュニケーションを用いた時間同期

本研究の提案手法が提案するNTPおよびSNTPについて，通常の有線によるインターネット環境とセンサネットワーク環境の違いを踏まえて示す．

NTP - Network Time Protocol -

ネットワーク経由でコンピュータ同士で同期させるプロトコルの代表例がNTP(Network Time Protocol)[1] であり，インターネット上で最も一般的に広く利用される方式である．また，NTPを時間調整の機能に特化させて簡略化されたプロトコルがSNTPである．本稿ではこのSNTPをベースととする．SNTPは基本的な構想と時間調整アルゴリズムはNTPと同様のものである．

まず，NTPを定義するものとして，Unix系OSにおける”deamon”，Windowsにおける”Service”と呼ばれるソフトウェアプログラムとされ，サーバとクライアントの間で時間の値を交換するプロトコルやシステムクロックを進めるか遅らせるかを判断するための時間の値を処理するアルゴリズム群によって構成される．SNTPは前述のように，複雑なアルゴリズムを用いずに，簡易にシステムを開発可能である．

以下にインターネットにおいて活用される際の主な特徴を述べる．

- 標準時間となる大域時計にクライアントが問い合わせることで，ネットワーク上の全てのコンピュータの時計を同期させることが目的である
- プライマリサーバは衛星や原子時計によってUTCと同期している
- 2wayによるサーバとクライアント間での時間情報の交換によって，誤差や端末間の遅延を計算する
- サブネットのアーキテクチャはStratumと呼ばれる階層的構造になっており，Stratum-0である最上位のプライマリサーバは正確な時間を保持していること
- 不安定なネットワーク上でも階層構造による高いスケーラビリティと耐故障性を保証していること

NTPにおける階層化構造を図3.1に示す．

[1] デラウェア大学の David Mills 博士が中心となって20年以上の間開発プロジェクトを進めており，最新は version4 が公開されているが現在のインターネットの標準は version3 である．

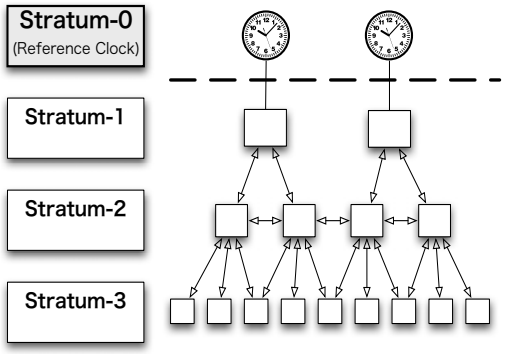

図 3.1: NTP における階層化構造

NTP は UTC に同期したプライマリサーバと，1 つ下の階層をセカンダリサーバ以下のサブネットをクライアントとした木構造を形成する．時刻源のサーバからどれほど離れているかを Stratum と呼ばれる端末間でのホップ数によって定義し，その値によって各クライアントが同期を行うサーバを選択する仕組みになっている．また，非常に複雑なアルゴリズムにより，フィルタリング機能や誤差，あるいはドリフト値の一時的な修正などの機能をもつ．

SNTP

SNTP(Simple Network Time Protocol)[7] は，NTP を簡易化したものである．特に，組込み機器などのハードウェアの性能の制限が多く，NTP の仕様の複雑な処理によって処理時間のオーバーヘッドが大きいものを対象にし，時間調整の機能のみに特化したプロトコルである．基本的な時間，オフセットや往復の通信遅延の算出に関する処理は同一のものであるが，Stratum による階層化構造をなくし，長期間に渡る統計的な誤差修正の機能を排除することで処理の軽量化を図っている．

3.2.1 無線センサネットワーク用プロトコル

センサネットワーク環境での時間同期プロトコルは近年盛んに行われている．中でも RBS（Reference Broadcast Synchronization）[8] は本稿のアプローチに近いリファレンスノードとの時間同期である．しかし，同メッセージのやりとりがリファレンスからメッセージを受信したタイムスタンプを受信者同士で交換することで同期を実現する点で異なる．また，FTS（Flooding Time Synchronization）[10]，TPSN（Timing-sync Protocol for Sensor Networks）[?] など様々なプロトコルが研究されている．

3.3 まとめ

外部装置を用いる手法では複数の衛星からの信号を受信する必要がある点である．つまり，電波の届きにくい屋内や月などの衛星が充分に存在しない局所的な環境においては適応出来ないため，汎用的な手法とは言えない．また，衛星からの信号を受信するためのGPS受信機をすべてのノードに配備することは大規模環境においてはコストの面で優れているとはいえない．加えて，対象のアプリケーションによっては，原子時計，GPS，あるいは電波時計が使えない局所的な環境が想定されることからも，ネットワークコミュニケーションを用いた手法による時間同期手法が重要であることは明白である．その中でもセンサネットワーク環境のための時間同期手法も数多く研究されているが，揺らぎや同期誤差の削減が主なトピックである．そのため，本稿のようなアプローチは理論的な分散アルゴリズムの分野において活発に議論されているが，理論と実践の間に隔たりが存在する．そのため，本稿ではその中立的な立ち位置で時間同期のプロトコルを提案し評価することで安定に推移することを確認する．

第4章　システムモデルと定義

本章では本研究において対象とするシステムのモデルを示す．

4.1　システムモデル

本節では本研究のシステムの仮定や前提を述べる．
前提として，センサネットワークにおける各ノードの実行体であるプロセスの集合を$\Pi = p_0, p_1, p_2 ... p_n$と表し，システムの最大ノード数（プロセス数）を$n$とする分散システムの枠組みで考える．以下にその特徴を挙げる．

- 時間非同期システム
  分散システムには大きく分けて，タスク実行時間，通信遅延，そして内部時計の変動に関する制限が存在する同期システムと，上限や下限の制限をまったく置かない非同期システムの二つのモデルに分けられる．
  同期システムモデルはシステムの挙動に制約をもたせることで曖昧さが少ないためシステムの設計は容易になるが，現実的な状況が制限されてしまう．一方，非同期システムは制約が緩いため様々な状況を想定した現実的なモデルとされるが，多くのアルゴリズムに解が存在しないことが知られている．そのため，本研究ではセンサネットワークの不安定な状況を想定した非同期システムでありながら，時間の概念に制約を持たせる時間非同期システム [11] を採用する．

- ノードは内部時計を持つ
  従来の非同期システムと時間非同期システムの最も大きな違いはプロセスを実行するノードはそれぞれ固有の時計を持ち，プロセスはそれぞれ自身のノードの時計を参照することである．時間非同期モデルにおける概念図を図 4.1 に示す．
  環境内の時間とはグローバルな実時間とノードがそれぞれ持つ内部時間が存在する．実時間tにおけるノードiの内部時計を$C_i(t)$と定義する．実時間tにおける異なるノードの内部時計の値は必ずしも一致せずそれぞれ異なる．
  一方，GPS や電波時計による外部から参照できる大域時計の存在が存在しない環境を想定するため，グローバルな時間情報を取得するのが困難である．そのため，環境内で時間の同期は，それぞれのノードは自分以外のノードがもつ内部時計の値を交換することにより実現する．

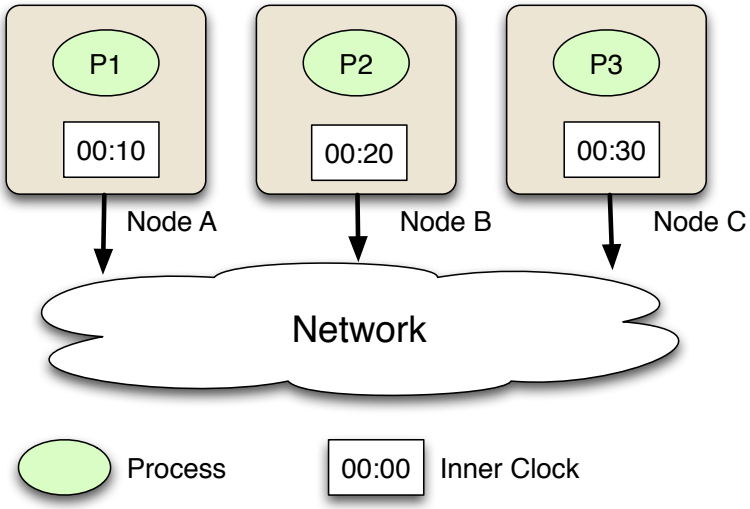

図 4.1: 時間非同期システムにおける内部時間

4.2 センサノードモデル

本節では本研究におけるシステムを構成するセンサノードがどのようなものであるかを述べる．

- ポジショニングシステムによる位置情報や時間情報は持たない
  通信インフラが整っていない局所的な環境を想定する．したがって，各ノードは自身の時間を GPS 衛星や電波時計による外部補正を受けることができない．そのため，ノード自身の位置や実時間を知ることができない．あくまで環境内での相対的な時間の同期を行うものとする．

- ノード固有の ID を持つ
  ノードはそれぞれ固有の識別番号を持つ．

- 故障モデル
  ノードの故障を想定する．本研究ではノード自体のクラッシュ故障を対象とし，各ノードのの故障率を α と定義する．故障した場合は隣接するノードとのチャネルが喪失し，メッセージの送受信は行わないものとする．

- ワイヤレスコミュニケーション
  ノード同士はコミュニケーションチャネルを通じてお互いにメッセージを交換することで逐次的に処理を進めるメッセージパッシングモデルである．通信が行えるコミュニケーションレンジ R は各ノードで共通の値を持ち，レンジ内の隣接ノードを $neighbor$ と定義する，ノード間での通信は無線によるブロードキャスト全方位型通

信を行い，コミュニケーション範囲内に存在する全ノードとコミュニケーションチャネルを形成する．なお，本研究ではユニキャストによる単一通信は想定しない．各ノードが持つ情報の交換はブロードキャスト通信によってのみ行われるものとする．次節においてノードの通信モデルを定義する．

4.3 通信モデル

本節ではノード間の通信のモデルと定義を述べる．ブロードキャストによる通信の概要図を図 4.2 に示す．

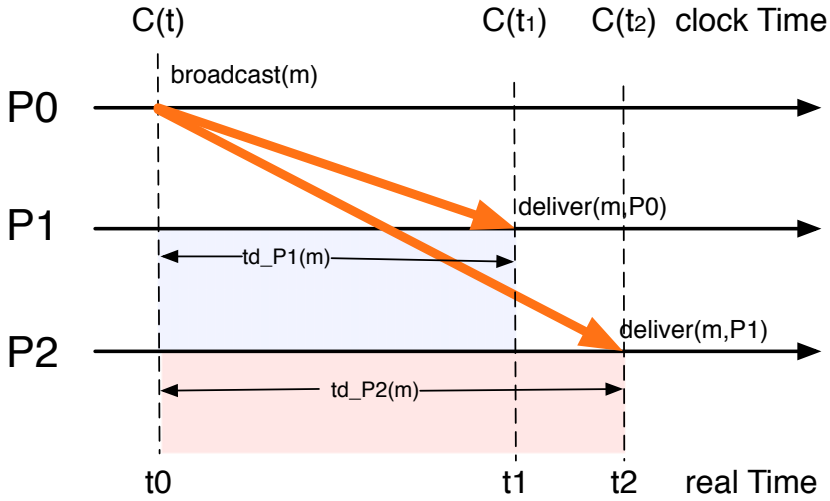

図 4.2: ブロードキャストによる通信

- $broadcast^r_{p_i}(m)$：プロセス p_i が内部時間 t においてメッセージ m をブロードキャスト．
- $deliver^r_{p_j}(m, p_i)$：プロセス p_i により送信されたメッセージ m をプロセス p_j が受信．

ノード間通信は基本的にこの仕組みによって行う．加えて，プロセス p_i がメッセージ m の送信した時間を $st_{p_i}(m)$，プロセス p_{p_j} が受信した時間を rt_{p_j} と定義すると，通信によって生じる遅延 $(transimissiondelay)$ を示す td_{p_i} は以下の式で導かれる．

$$td_{p_j} = rt_{p_j}(m) - st_{p_i}(m) \tag{4.1}$$

さらにこの td_{p_j} の遅延の下限を $td_{p_j} \geq \delta_{min}$ とする．この δ_{min} は Cristian らは [11] においては $\delta_{min} = 0$ としている．これは，ネットワークの Bandwidth やメッセージサイズなどの流動的要因よって左右されるためであり，本研究でも同様に設定する．一方で，δ の

上限に関しては故障したプロセスを検出するために非常に重要な要因である．図 4.3 のように，δ をメッセージ受信におけるタイムアウト値に設定することで送信側の故障を検出することが出来る．

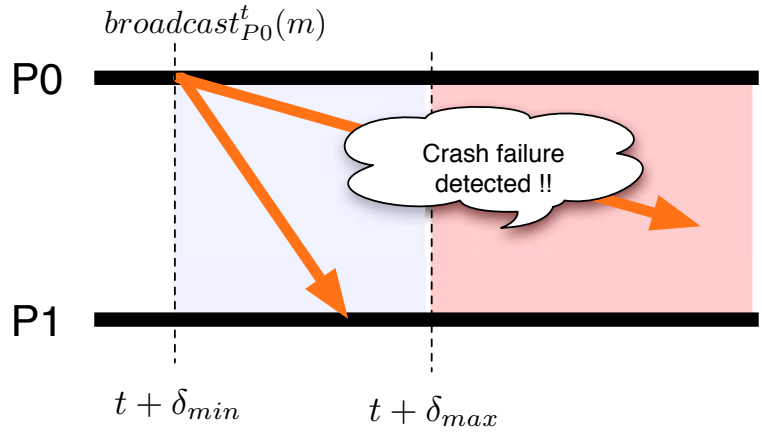

図 4.3: δ によるタイムアウト型故障検出器

時間の同期

本節では時間の定義を示す．各ノードの持つ実時間 t における内部時間を $C_{p_i}(t)$ とする．時間を同期させるとは，ある実時間 t において二つ以上の内部時間をノード間で同じ値に合わせることである．つまり，ある実時間 t において $C_{p_i}(t) = C_{p_j}(t)$ であることを指す．

しかし，内部時計は様々な要因によって進み方に誤差が生じるため，時間の誤差は少しずつ拡大していく．一般的に内部時計は発振器による規則正しい振動を計測器が振動数にカウントすることにより時間を進ませる．つまり，発振器の振動の誤差が時間の進み方の違いとなってあらわれる．この誤差の割合を $drift$ 率と定義する．この $drift$ が発生することで各内部時計は．一般的な発振器の drift 率は 1 秒間に $2\mu sec(20^{-6})$ であり，主に経年や環境の変化によって発生する．ネットワークコミュニケーションによる同期を行う際には，不定な通信の遅延と $drift$ によって誤差が生じる．仮に同じ通信遅延であると仮定した場合でも，同様であることが知られている．図 4.4 に drift 率の概要を示す．$C_{p_i}(t) = t$ は理論的に完璧な時間である．なお，本稿における環境では実際のノードは自分自身で外部補正装置にアクセスできないため，この実時間を知ることはできないものとする．

Eventual Leader Election

ネットワークによって接続されたノードの集合に対して，リーダーとなる 1 つのノードを選定するための問題である．その中でも特に，自律的にいつか 1 つのリーダーを選定す

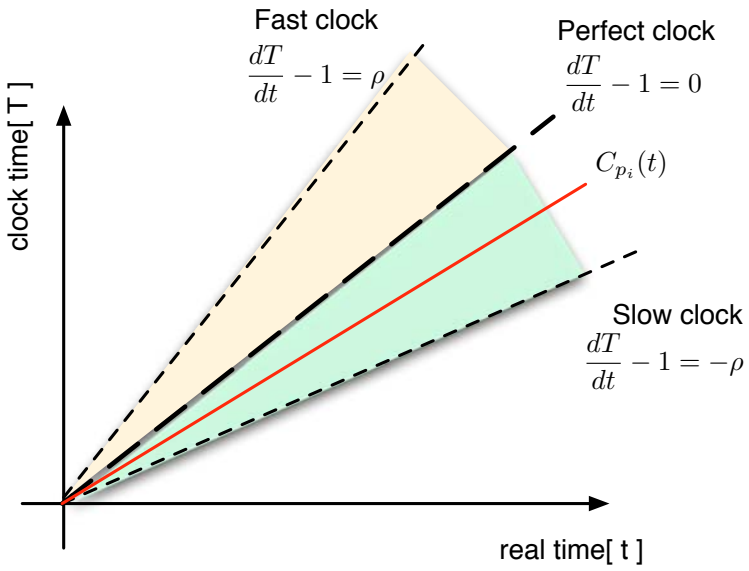

図 4.4: drift の概要

る手法を Eventual Leader Election とされる．本稿においては，動的な環境の変化に対応した自己安定する手法として時間同期と組み合わせたプロトコルを提案する．

第5章　提案プロトコル

本章では，我々が提案する自己安定型時間同期プロトコルについて述べる．前述の通り，時間同期はノード間でのネットワークコミュニケーションを通じて交換することで実現する．本稿では定期的な $pulseMessage$ の通信のみで隣接ノードと時間同期を行いながら，参照先とネットワーク全体の時間の参照元であるリファレンスノードを自律的に決定する手法を提案する．

5.1　概要

本節では，提案する時間同期プロトコルの全体像について述べる．本稿で提案するプロトコルは以下の二つのアルゴリズムによって構成される．

時間同期アルゴリズム：ノード間での内部時間調整

SNTP の 2-way プロトコルをブロードキャスト仕様に改良し複数のノードとの同期を行う．

リファレンスノード選定アルゴリズム：リファレンスノードを中心とした生成木の作成と階層化および故障検出器．

この二つのアルゴリズムを pulse メッセージの定期的なやりとりだけで実現する．状態遷移によって変わる挙動をすべて一種類のメッセージタイプで実現する．ノード数の増加に対応し故障の発生によるトポロジの動的変化が発生しても自律的に安定状態に収束するプロトコルとして提案する．

5.2　時間調整方法

本項では内部時間の交換による時間同期手法に関して述べる．SNTP における 2-way の算出方法をベースに本稿における条件に最適化させる．

5.2.1　単一ノードに対する同期

ノード間の双方向通信によって内部時計のタイムスタンプを交換する．図5.2 に 2 ノード間でのメッセージの往復による同期の流れを示す．

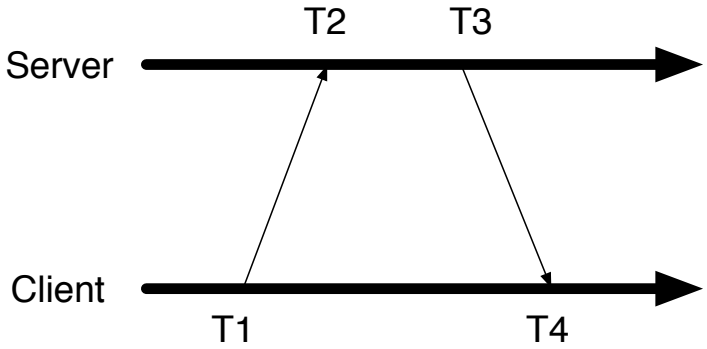

図 5.1: SNTP おける 2-way 時間同期

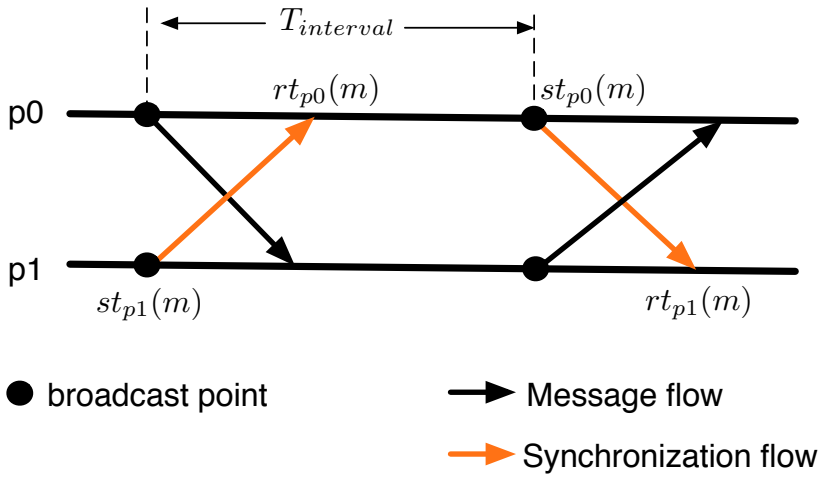

図 5.2: 本稿における 2-way 時間同期

図5.1 はSNTP の Server と Client の時間同期のためのメッセージフローである．Client からの要求に対して，Server が返答する即時的なメッセージ交換が特徴である．つまり，

リクエストに対して，一時的な保存を行わず，Server 側からの即時返答によって同期を行う過程を

図 5.2 は，同様にノード A に対してノード B が時間を問い合わせているシーンである．最も大きな違いは，それぞれのノードは定期的に pulse メッセージを隣接ノードに対して送信を行っている点である．pulse メッセージを送信する間隔を $T_{interval}$ の値でそれぞれのノードで設定され，B は自分の同期参照先が A であることは知っているものとする（同期先の選定に関する詳細は後述する）．

時間同期はノード間の時間差 $(offset)$ を求めることが目的となる．B は pulse メッセージを送信する際に内部時計のタイムスタンプ st_B を送り，A はメッセージを受信した時間である rt_A とする．このとき，A の内部時間は $rt_B + td_A + offset$ で表すことができるが，各ノードの drift 率を含めた正確な offset はこの時点では知ることが出来ない．そのため，pulse メッセージを受信した A による返信時のタイムスタンプ (st_A) と B の受信側のタイムスタンプ (rt_B) を合わせた 4 つのタイムスタンプにより，以下の式で時間差 $offset$ と通信の遅延 $delay$ を算出する．

$$offset = \frac{(rt_A - st_B) - (rt_B - st_A)}{2} \tag{5.1}$$

$$delay = \frac{(rt_A - st_B) + (rt_B - st_A)}{2} \tag{5.2}$$

B 側ではこの $offset$ の値を修正することで A と同期を実現する．

5.2.2 問題点 1: 複数の隣接ノードとの同期

この場合のように，単一ノードを対象としたユニキャスト通信においては SNTP と同様のプロトコルを適応することができる．しかし，センサノードの通信は無線によるブロードキャストが基本であり，複数の隣接ノードに対してメッセージを一斉送信するため，同一メッセージ内にリクエストが到着しているノードに対してのタイムスタンプを含めなければならない．次項ではブロードキャスト仕様に最適化した手法を述べる．

5.2.3 ブロードキャストによる複数ノードの同期

前節の図 5.2 では，同期先に対して要求を行うノードは 1 つの場合である．図 5.3 にブロードキャストの特性に着目した手法を示す．

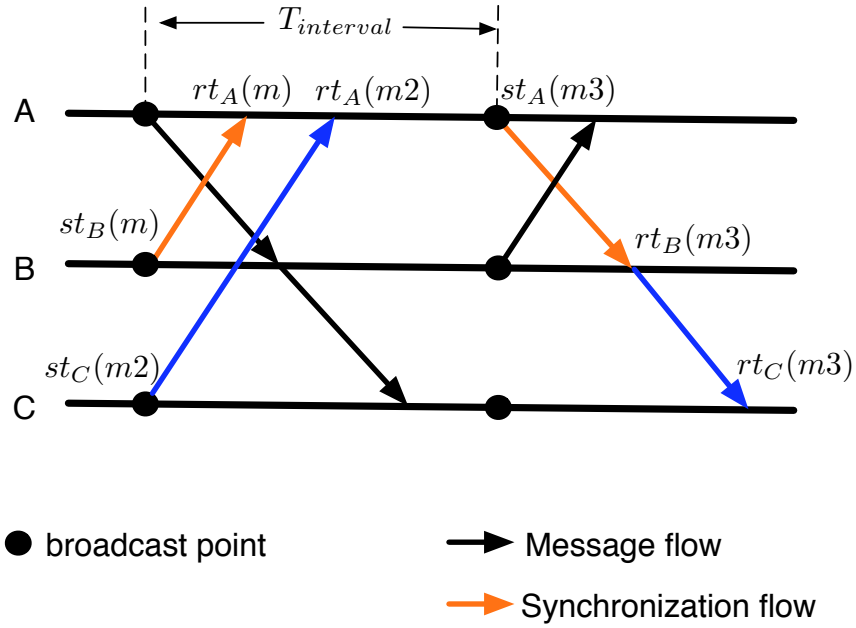

図 5.3: 複数の隣接ノードとのメッセージ交換

neighbor が複数の場合，$T_{interval}$ の待ち受け時間の間に複数のメッセージが可能性がある．大規模であればあるほど，それだけ 1 つのノードの通信範囲 R 内に存在する通信可能なノード数が増える．この密度の高さに応じたタイムスタンプの処理も大きな問題の 1 つである．

図 5.3 の例では，A に対して B,C からリクエストが届いている．そのため，A は $T_{interval}$ の間，つまり次の送信タイミングであるブロードキャストポイントまでそれぞれのノードからのメッセージを送信時と A が受信した際のタイムスタンプのセットを保持する必要がある．表 5.1 に各ノードが持つ受信済みのデータを格納する *receivedTable* を示す．

表 5.1: リクエスト受信時のデータ構造

Received Table		
NodeID	sendTime	receiveTime
B	$st_B(m)$	$rt_A(m)$
C	$st_C(m2)$	$rt_A(m2)$

そして，このIDと二つのタイムスタンプのセットを自分の*neighbor*に対してブロードキャストを行う．同期先からのメッセージを受信したノードは，自身のIDと等しいタイムスタンプのセットを取り出す．その後，式5.1と式5.2により，内部時間の差と通信遅延の値を算出することで内部時計を同期させることができる．

なお，複数の*neighbor*，つまり，この場合親ノードが通信範囲内にいるときに行われる．その際，内部時計を修正する$offset$を一時的に保存し，閾値であるneighbor数だけ集まったらそれらを平均して修正している．後述する階層化された構造では，親の階層のノード

この手法の利点は以下の二つである．

- 一斉送信で複数の*neighbor*との効率的な同期が実現できる
- 要求と返答が対になる即時的なSNTPのプロトコルとの異なり，定期的なメッセージ送信により安定した時間同期を繰り返すことが出来る

5.2.4 問題点2: 閉路による同期ループ

前項の手法で*neighbor*と同期を行う場合に生じる問題点として，ネットワークのトポロジ内に閉路（cycle）が存在する場合を想定する．閉路とは，グラフ理論においてノード間の1つの辺から成り，言い換えれば1つの円状にループしている構造を指す．ネットワークを双方向の無向グラフとして考えた場合を図5.7と図5.8を図に示す．

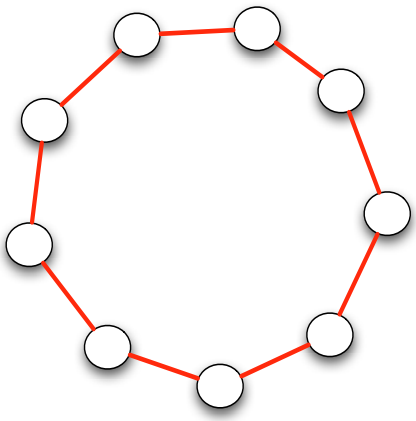

図 5.4: 完全閉路グラフ

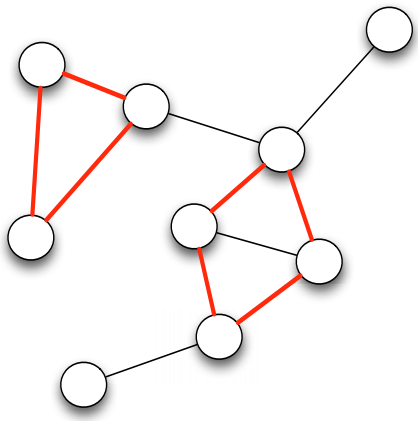

図 5.5: 部分的閉路を含むグラフ

ネットワークに部分的な閉路が存在する場合，同期のループが発生してしまう．つまり，指向性のない論理的な経路のため，閉路内でお互いに同期し合ってしまう．それによって，ネットワーク全体の時間の統一的な同期を行うことが出来ない可能性が生じる．この問題点を解消するためには各ノードの同期先を一意に決定し，根となるリファレンスノードを

中心とした木構造が必要となる．次節ではシステム内で論理的な全域木 (spaning tree) を自律的に生成し，すべての内部時間の中心となるリファレンスノードを決定する手法について述べる．

5.3　リファレンスノード選出方法

本節では，システム内のリーダーノードを決定しトポロジを安定化させるプロトコルを述べる．

5.3.1　自己安定

前項で述べた閉路が生じる問題点に対して，リファレンスとなるノードが必要であることを述べた．本稿ではネットワーク全体で全域木を形成することで各ノードの同期先を一意に決定しながら，最終的に 1 つのリファレンスノードに同期先が収束する手法を採用する．

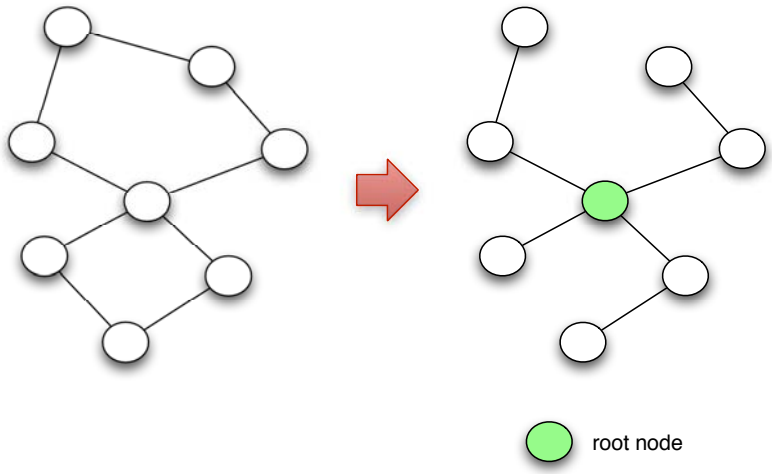

図 5.6: 全域木への変換

木構造へ変換する必要性として以下の点を挙げる．

- リファレンスノードの決定
  リファレンスとなるリーダーノードはシステム内に 1 つであることが望ましい．最終的に 1 つのノードに決定される必要がある．以下にこのアルゴリズムの要求を挙げる．

- 各ノードの参照先の決定
  各ノードが自分の同期先となる親ノードを決定したい．しかし，リファレンスとの相対距離が判別出来ないと安定しない．(詳しくは後述) リファレンスと繋がっている木構造のため，末端のノードであってもリファレンスとの同期を実現する．

- 故障が発生した際のトポロジの維持を自律的に行う自己修復能力
  木構造のためノードの故障により，トポロジの動的な変化が生じる可能性がある．そのため，各ノードは自分の同期先に対して，生存しているかどうかの判定を行う必要がある．そのため，ノードの故障を検出後メッセージの送受信が行われないと判断した場合，自身の同期先を再設定する．

- 時間同期プロトコルとの融合性
  この全域木構築のためだけに別種類のメッセージを送るのは効率的ではない．したがって，定期的に送信する pulse メッセージに必要な情報を含ませるだけで，ネットワーク全体にマルチホップで Flooding させることなくノードからノードへ伝播する仕組みを考慮する．つまり，シンプルな値でデータ量を抑えた値を時間同期のためのメッセージ交換に加えつつ，リファレンスノードを選出する仕組みが必要である．

これらの要素を踏まえた生成木を構築する Eventual leader Election アルゴリズムを提案する

初期状態

全ノードの初期状態は以下の通りである．

- 状態: IDLE
- 隣接ノード数 ($neighbors$) の初期値は 0 とする（隣接ノードに関する情報は持たない）
- 固有のノード ID を持っている
- 自分の同期先を判定する値としてローカル変数 $parent$ を持つ．初期値は自身の ID とする．

評価基準

各ノードの親となるノードを決定するための評価基準 RootInfo は以下の 2 つの値を用いる Set である．

- $nodeID$: ノードが持つ個別のユニークな ID
- $neighbors$： ノードの通信範囲内に存在する隣接ノード数

各ノードの親となるノードは，隣接ノード内から評価基準である各ノードの ID と $neighbors$ によって一意に決定される．ノードの次数である $neighbors$ が多いほどネットワーク全体に対する影響度が大きいためである．$neighbors$ のメッセージを受信する際に自身の通信範囲内の隣接ノードかどうかのカウントを行うため，この値は動的に更新される．さらにノード ID は $neighbors$ が等しい場合に一意に親を決定するために用いる．そのため，重複を許さないユニークな非負の整数とする．図 5.7 にメッセージ受信時の処理を示す．

リファレンスとの相対距離

これに従い，メッセージ交換を続けて $RootInfo$ をアップデートし続けることで最も評価基準の高いノードがリファレンスノードとして 1 つに収束する．

しかし，このアルゴリズムだけではリファレンスの相対的な位置が判明しないため，時間を同期させる対象のノードがリファレンスに近いノードかどうか判断することができない．本稿では木構造における下位のノードがリファレンスノードに向かって辿るように同期することで，リファレンスとの遠隔的な同期を図ることでシステム全体の時間の同期を行うことが目的である．したがって，リファレンスとの相対的な距離を明らかにし，自分がどのノードと同期を行えば良いかの依存関係を明確にする必要がある．

以上の問題点を解決するために，リファレンスとなるノードは自分がリファレンスであると認識した段階で各ノードの自分からの距離，つまり最短のホップ数 $hopCount$ を pulse メッセージに含ませ，下の階層のノードへ伝播させる．この値を受信したノードが $hopCount$ の値をカウントアップさせ，その他のノード経由で伝播させることで各ノードが $hopCount$ を知ることができる．また，その過程で自分の親となるノードの情報を知ることで各ノードの同期先が確定する．具体的には，自分の $hopCount$ を i とすると，時間の問い合わせ先は $i-1$ となり，リファレンスから自分より遠い（下の階層の）ノードとは同期を行わない．つまり，これによってノード間の依存関係によって生じる閉路の同期ループを抑止することができる．以上の理由から，リファレンスからの $hopCount$ による階層構造を構築する．

5.3.2 問題点 3:故障ノードの検出

故障ノードが発生しても自己安定する全域木を構築することためには，故障をどう定義し，どのように検出するかが重要な問題となる．つまり，故障を速やかに検出しそのノードをネットワークから切り離すことが求められる．特にリファレンスノードが故障した際にはネットワーク全体にその影響が及ぶ危険性があるため，最も考慮すべき要件である．次節で本稿における故障検出手法に関して述べる．

5.4 耐故障性

5.4.1 故障検出器

一般的に，分散システムで用いられる故障検出器のタイプは pull 型と push 型に分けられる．この区別は，故障を検出する側のアプローチの仕方による区分けである．具体的には，前者が観察者が生存メッセージを送信し，それに対する返答を待つ．そして，被観察者から返答が一定時間帰ってこなければ故障と検出する．後者は被観察者が生存確認メッセージを一方的に送信し，観察者は被観察者からの生存メッセージの受信する間隔によっ

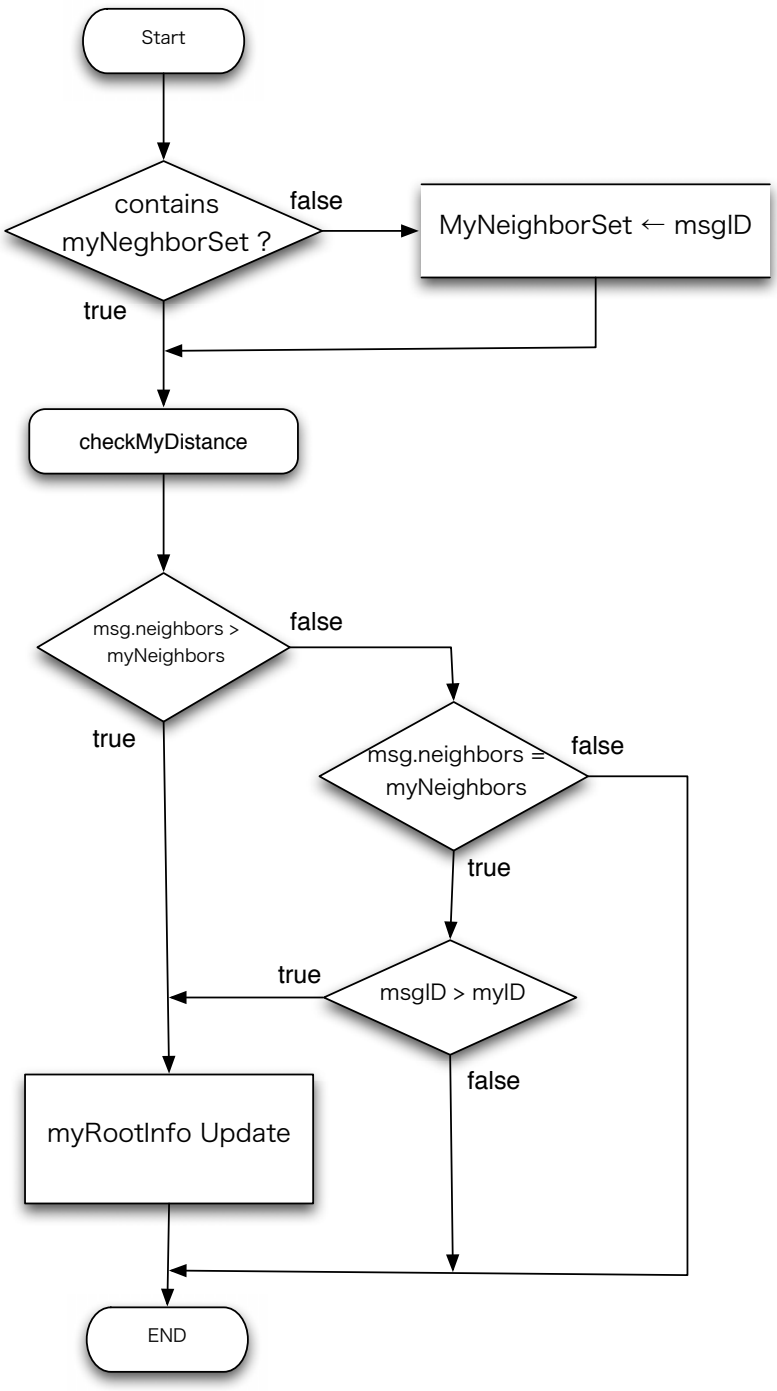

図 5.7: Leader election の流れ

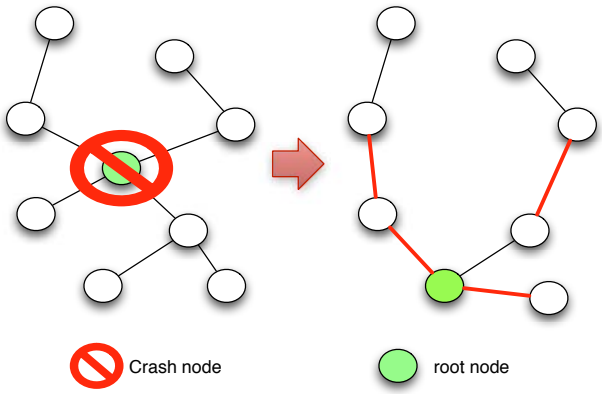

図 5.8: 故障発生時における木の再構築

て，故障かどうか判断する．言い換えれば，pull 型の双方向による相互検出と push 型の一方通行な通信によるものである．以下に pull 型と push 型の違いを明確にし，本稿において適応するタイプを選定する．

- Pull 型：生存メッセージに対して即時的な返答が求められる．
- Push 型：定期的なメッセージの送信のみで，返信メッセージを必要としない．

本稿の通信モデルは定期的なブロードキャストによる一斉送信のみを行うことから，即時的な返答を行うことは想定していない．さらに，全域木を構築することでノードの同期先（各親ノード）が一意に決定することから，観察者と被観察者との間に双方向の対話型通信は必要ない．つまり，それぞれの子ノードが観察者となり，親ノードからのメッセージを監視することで故障の検出が可能である．したがって，本稿では pull 型のアプローチによる故障検出器が最適であると考える．なお，末端のノードに関しては子となるノードが存在しない上，故障の影響が他のノードに伝播しないことから検出を行わないものとする．

pull 型故障検出器

前節で述べたように，pull 型の故障検出器を考える．検出の流れを図 5.9 に示す．

$p0$ が親ノード，$p1$ を子ノードとする．検出の際は子ノードが観察者となり，被観察者である親ノードの故障を監視する．

前述の通り，各ノードはお互いに $T_{interval}$ の間隔で pulse メッセージを送信している．この場合 $p0$ から $p1$ にメッセージが到着している間は $p0$ が生存していることを意味する．つまり，メッセージが $p1$ へ届かなくなった際に $p0$ が故障と判断することができる．そのためのタイムアウト値を $T_{timeout}$ とする．この値によってメッセージが受信までの規定値を設けることで故障によるメッセージの不通を検出することができる．このとき，

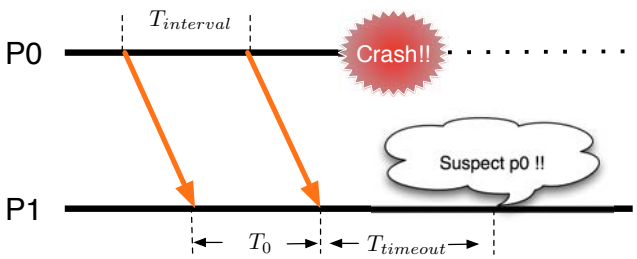

図 5.9: 故障発生時の Pull 型故障検出器の例

$T_{timeout} \geq T_{interval}$ にするべきあり，このアプローチによって発生する $T_{timeout}$ の設定は検出までの時間と正確さの間にトレードオフが発生する．

もし，$T_{timeout}$ 値を長く設定すれば正確に故障を判断することができるが，それだけ検出までの時間が長くかかってしまう．反対に，$T_{timeout}$ 値の設定を短くすれば，検出までの時間が短くなるが，故障ではなく通信による遅延によりメッセージ到着までの時間がかかっているだけで故障ではないかもしれない．つまり，誤検出の可能性があがってしまう．このトレードオフはアプリケーションによって，Quality of Service の要求によって設計が異なると考えられる．本稿においては故障の発見を正確に行うことに主眼をおいて $T_{timeout}$ 値を設定する．

タイムアウト値の設定

本稿では，$T - timeout$ 値は動的に変化する独自の 2 つの値によって随時更新することでネットワークのトラフィックやノードの状態に対応する．2 つの値とは，定期的な pulse メッセージの到着間隔である T_{last}，そして $p0$ からメッセージが送信されて $p1$ が受信するまでの通信による遅延時間 T_{tr} である．$T_{timeout}$ の値を以下の式 5.3 で算出する．

$$T_{timeout} = T_{last} + T_{tr} \tag{5.3}$$

T_{last} と T_{tr} は親ノードからの最新の pulse メッセージの値を用いるため，メッセージを受信する度に更新される．子ノードはこの $T_{timeout}$ を親からのメッセージ待ち受けの閾値内に到着しない場合，故障と自身の親ノードの情報を消去する．

第6章　シミュレーション

本章では，提案プロトコルを実装し，シミュレーションによって大規模環境での有効性を評価する．

6.1　目的

本稿で述べるプロトコルを性能評価を行う際には，実際にセンサノードを配置して評価を行う手法が考えられる．しかし，ノード数が膨大な大規模環境を想定しているため，充分な数のセンサノードを利用するコストや配置する場所の制約があり困難である．そのため，本稿では分散アルゴリズムの開発や評価に用いられる分散フレームワーク Neko の仮想ネットワーク上でセンサネットワークを構築し，シミュレーションを行った．その上で提案プロトコルを実際に動作させることで生じる時間の誤差や挙動が自律的に安定することを分析しその有効性を評価する．

6.2　シミュレーション環境

Neko について

Neko とは，分散アルゴリズムの開発および性能評価を行うためのフレームワークである．アーキテクチャは上位のアプリケーションと仮想ネットワーク部によって成り立ち，send という基本的な命令によってネットワークに送り出し，ネットワークは deliver 命令によってメッセージをそれぞれのプロセスへ dispach する．

また，Java 言語による離散的なイベント駆動のエンジンを持っており，アプリケーション層のプロトコルを実装する際に必要となる機能を API で公開している．これらにより，開発者はアプリケーション層で複雑な分散アルゴリズムを API を用いて記述し，ネットワーク層によるシンプルなメッセージパッシングモデルによる実装と検証を同時におこなうことができる．さらに，Neko の API を用いて実装された分散アルゴリズムを複数の計算機を用いた実ネットワークでの実験と単一計算機でのシミュレーションにも対応している．つまり，開発者が異なる環境で検証を用いる必要がなく，Neko による統合的な環境が提供されているのである．

シミュレーションを行う際には，Neko のメッセージを伝搬するための基盤であるネットワークはユーザが独自に定義する config ファイルを用いて指定することができる．しか

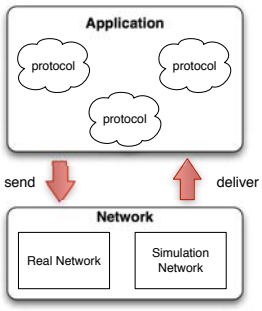

図 6.1: Neko の概念図

し，今現在 Neko の想定するネットワークはそれぞれのプロセスが完全結合した構造のみであり，センサネットワークのような動的な挙動や不特定な要因によるノード状態の変化によってトポロジが変化するネットワークは実装されていない．そのため，センサネットワーク環境のための分散アルゴリズムやそれらを用いたアプリケーションを開発および評価を行うことができない．Neko を用いたセンサネットワーク環境のシミュレートは必要不可欠である．以上の理由から，本節では Neko におけるネットワークシミュレータを擬似的なセンサネットワーク環境に最適化する．

センサネットワークへの適応

Neko のネットワークは予め実装されている Basic Network や Random Network などを config ファイルから区別する．これにより，開発者はネットワークを意識することなくアプリケーションを開発することが出来る．さらに，ユーザが独自に定義したネットワークを追加可能である．これらに従い，シミュレートした独自のセンサネットワークを以下のように Neko に追加した．

トポロジ管理

ノードとアプリケーション層のプロセスの ID によりマッピングし，ネットワーク層でそれぞれのノードの情報を管理する機能を追加した．ノードが持つ初期座標，通信範囲を config ファイルから設定可能とすることで動的な更新による neighbor ノードや故障ノードの判定をアプリケーション層で意識することなく開発を行うことが出来る．例えば，メッセージを送信する際に通常はアプリケーション側で send メソッドの引数として宛先プロセスの ID を指定するが，センサネットワークのように動的なトポロジの変化が頻繁に発生する環境ではブロードキャスト通信が基本である．そのため，アプリケーション側で送信受信可能な対象を意識する必要がない．つまり，環境の情報とアプリケーション層でのアルゴリズムを切り離す必要がある．以下にメッセージの送受信の過程を示す．

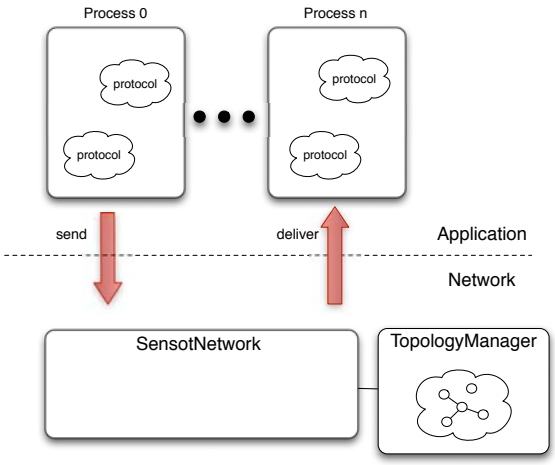

図 6.2: センサネットワークのシミュレート

STEP1 アプリケーション側で send() が呼ばれる．宛先の情報は対象のプロトコル ID のみ．

STEP2 ネットワーク側でメッセージの送信元のノード情報を問い合わせ，現在の座標からノード間の距離を算出し通信範囲内の隣接ノードを特定する．

STEP3 送信元から隣接ノードとの相対距離から通信の遅延時間を算出する．

STEP4 timer スレッドを起動し，遅延時間経過するまで Queue に入れる．

STEP5 遅延時間経過後，対応するプロセス ID 内のプロトコルへメッセージを deliver() する．

初期情報として座標を個別に設定する場合は以下の情報を config ファイルに記述する必要がある．

- ノード ID
- (x，y) の 2 次元座標
- 通信範囲（全ノード共通）

6.2.1 コンフィグファイルの設定方法

Apache の org.java.util パッケージの configuration クラスを利用して動作する．Neko の config ファイルは ”ユーザ定義のパラメータ = 値 or 文字列 or 真偽値” の形式で記述される．開発者は実装したアルゴリズムに対応して自由な値を設定可能である．本稿にお

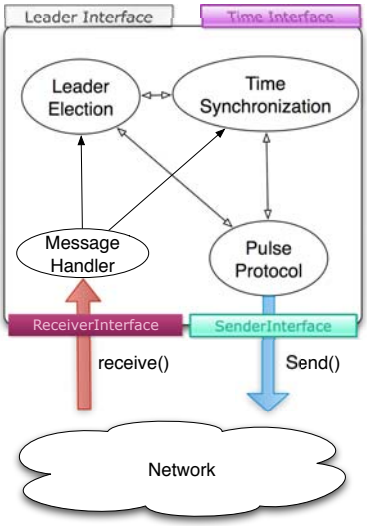

図 6.3: 提案プロトコル overView

いては，前述の各ノードの情報の設定に加えて，規模の大きいネットワークを構成する場合は config ファイルからノードの初期位置のランダムに設定できる．

こうした環境により，各ノードが隣接ノードへブロードキャストを行うセンサネットワークをシミュレートしている．以下に Neko のプロトコルに対応させた全体像を示す．

6.3 シミュレーションシナリオ

本稿では前述のシステムモデルを基に作成したシミュレーションのシナリオを作成し．前節で構築した Neko 用のセンサネットワーク環境でシミュレートする．

・仮想空間

センサノードを配置する環境は一辺が 100m の正方形の空間とする．なお，空間内にはセンサノードのみが存在し障害物などによるメッセージの遅延や消失は考慮しない．加えて，GPS や電波時計などのインフラストラクチャが存在しない局所的な環境を想定するため，

・センサノード

ノードが通信可能な範囲は 20m とする．それぞれユニークなノード ID を所持する．ノードの配置は空中からの散布を想定するため，初期位置は仮想空間内にランダムに配置されるものとする．

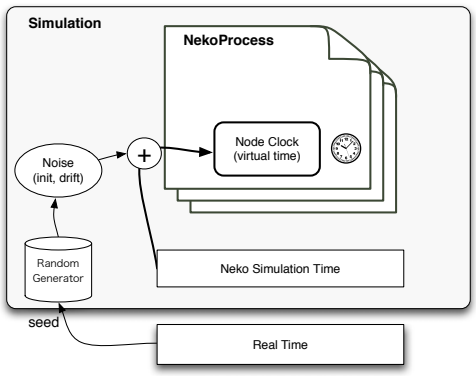

図 6.4: 仮想時間のモデル

・仮想時計

ノードはそれぞれ個別の内部時計を持っている．そのため，Neko によるシミュレーションを行う際にノードそれぞれが持つ内部時計を擬似的にシミュレートする必要がある．その際，離散事象型シミュレータである Neko 上のイベント（send もしくは deliver などの通信遅延による時間の推移）はすべてシミュレーション時間に従う timer によって管理されることを考慮しなければならない．例えば，シミュレーション内の時間はあるイベント A の発生した時間から次のイベント B の発生時刻まで進める方式であり，実際の実時間のように均等な進み方をする時間の概念とは異なる．つまり，実時間を基準にノードの時間を測定するのは Neko のシミュレーションを行う上で意味を為さない．したがって，シミュレーション時間を基準にした独自の時計を実装する仕組みをする必要がある．

本稿では Neko におけるシミュレーション時間を元に仮想の時計をノード毎にインスタンス化させ，図 6.4 のようにそれぞれ時間が異なるように設定する．具体的に各ノードにおける node clock の値は式 6.1 で表す．なお，$SimulationTime$ とは Neko 上でのイベント発生時の論理時間，$init$ は初期化された段階でそれぞれの仮想時計でランダムな時間差を持たせるための値である．$drift$ は仮想時計の単位時間における時間の進み方を表し，時計は退行しない一般的な前提から非負の値とする．

$$vTime_(n) = (SimulationTime + (init)) * (1 + drift) + offset \tag{6.1}$$

この値を $vTime$ とする．それぞれの仮想時計から getTime() メソッドでコールすることでイベント発生時の $vTime$（現在の時間）を得ることができる．$drift$ や $init$ は乱数として正規分布に従うものとする．

6.4　シナリオ

6.4.1　予備調査 1: 小規模環境での適応

トポロジの形状を指定し，提案プロトコルが正常に動作していることを確かめる．ノード数 6 の単純なリング型のトポロジを想定する．メッセージの同期を行うに従い，安定状態によって標準偏差と変動係数の推移を確かめる．

6.5　評価

予備調査

Number of message	Standard Deviation	Coefficient of variation
0	18.79097085	0.283758295
10	17.43503803	0.1084753
100	14.91233752	0.014142335
1000	14.55168118	0.001441057

通信の回数が増えるに従い，標準偏差が 0 に近づいていることが分かる．つまり，各ノードの時間が平均値前後に向かって推移していることが予測できる．さらに相対的なばらつきをあらわす変動係数も減少し続けていることが分かる．これはリファレンスノードを中心に安定した同期が行われているためであろう．これにより，リファレンスノードが 1 つに決定しトポロジの変化がない安定した状態になっていることが分かる．仮に，故障が発生した場合は不安定な状態を経ても，安定状態に収束する．不安定な状態では同期先が安定して決定されず，正しくないノードと同期してしまう危険性がある．本稿のプロトコルでは，リファレンスノードが複数存在する不安定な過程から最終的に 1 つのノードに収束するため，不安定な状態では時間の各ノードがもつ時間のばらつきの発生は避けられない問題であるが，その影響も次第に最小限に留まる．

加えて，通信レンジ内のすべてのノードが定期的にブロードキャストする環境であるため，大規模な環境においてはリファレンスとなるノードの情報が，すべてのノードに伝播するまで時間がかかる．予備調査で行った 6 ノードのリング状のトポロジは，リファレンスまでの最長 hop 数が 2 と少ないためすぐにリファレンスノードが 1 つに決定し，それぞれの階層のレベルが知らされるため，安定したものと考えられる．そのため，大規模な環境では不安定な状態が長く続く可能性がある．しかし，本稿の階層判別アルゴリズムにより常に自分と近いリファレンスとの距離に更新することで解消し，常に最新の同期先となる親階層の情報が伝播するため問題はないと思われる．

6.5.1 考察

大規模環境に適応させるにあたっての問題点となる要素について述べる．このプロトコルの問題点は，前述のように隣接ノード同士による情報の伝播のみを対象にするため，規模のスケールが大きくなればなるほど，すべてのノードに情報が行き渡るまでの伝達速度が犠牲になってしまう点であろう．つまり，階層的なネットワーク構造が決定的になるまで不安定な状態が継続されることであり，その状態ではリファレンスノードが決定されるまでネットワーク全体の時間を近似させるのは困難である．そのため，提案プロトコルのようにメッセージを交換しつづけて安定するまでの時間的なコストが必要な自律的なプロトコルか，あるいはGPSのような外部装置を用いるかどうかはコストと正確さの要求のトレードオフを考慮する必要がある．

本稿で述べた定期的な単一メッセージのみでの時間同期は小規模なセンサネットワークでのシミュレーションでの確認に留まった．しかし，大規模環境における安定状態までの時間に制限がなければ，提案プロトコルでも同様に有効であると思われる．さらに発展させるために．全体への伝播時間の遅れを考慮し，リファレンスノードを決定することで実現する中心的な構造だけでなく，自律分散的に各自が近似的な時間を見積もるようなアルゴリズムを考えることもできる．本稿は neighbor 同士の複数の offset を算出し，その平均をとることで時間の近似を試みた．おそらく，わずかな揺らぎである時計のドリフト値を段階的に修正することでより正確な値を見積もることができると思われる．ただ，その際には通信の遅延も考慮しなければならない．場所を問わないワイヤレス通信であるがゆえに，障害物や環境的な制約に影響される不定な遅延が重要な要素となる． また，上位層だけでなくMAC層の揺らぎも遅延の原因の1つである．また，メッセージを送る間隔を短くすることで正確な同期が可能であるが，電力消費や無線チャネルの衝突などのネットワークトラフィックに負荷がかかってしまうことが課題となる．

第7章　まとめと今後の指針

本稿ではセンサネットワーク環境における単一メッセージのみで自己安定する時間同期プロトコルを提案した．2-way によるノード間のタイムスタンプを交換するし複数のノードと同期を行う手法，さらに自律的に同期先を選定しリファレンスとなるリーダを最終的に 1 つに収束させるためのプロトコル，さらに故障の検出方法を示した．これらの提案プロトコルを統合し，Neko によるシミュレーション環境を整え，小規模な環境での有効性を示した．さらに，大規模環境においても同様のアルゴリズムで適応可能である．今後，シミュレーションプログラムのデータ構造に問題があると思われるため修正を行う．なお，本稿に関連した今後の指針を以下に挙げる．

- シミュレーションプログラムの改良と大規模な環境への適応
- 実際のセンサノードを用いた誤差の計測
- ドリフト値の見積もりアルゴリズムの開発
- センサネットワークにおける分散アルゴリズム適応支援環境の構築

本稿ではシミュレーションを用いて実験を行った．さらに正確な検証を行うためには実際のセンサノードを用いた現実的な誤差の計測が必要であると考えられる．そのためには，定期的な単一メッセージによるアプローチはメッセージ内のデータ量の効率化など，より一般的な問題への適応が考えられる．上位層のアルゴリズムだけでなく MAC 層の遅延や揺らぎを考慮した実用に耐えうるプロトコルに最適化する必要があると考えられる．それに伴って，センサネットワークは適応対象が幅広いことから，野外や局地的な環境でのケースでの活用も多く，アプリケーションによって最適なアルゴリズムやシステムが異なる．そうした中でも時間同期は必須の技術であり，アプリケーションによって異なる制約のもとで大規模な環境でも自律的に安定する高精度なプロトコルが今後も求められている．本稿の自己安定を目的としたプロトコルがその開発の一助となると思われる．

7.1　謝辞

本研究を行うにあたり、日頃より懇切丁寧な御指導を賜りました Defago Xavier 准教授に厚く御礼申し上げます．また，日頃より岸研究室，青木研究室の皆様方にはお世話になりました．この場を借りてお礼申し上げます．

参考文献

[1] D. Estrin, L. Girod, G Pottie, and M. Srivastava, Instrumenting the world with wireless sensor networks, In Proc. of International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001), pp.2033-2036, 2001.

[2] F. Sivrikaya and B.Yener, Time syncronization in sensor Network: A survey, IEEE Network Magadine ' s special issue on Ad Hoc Networking: Data Communications and Topology Contorol, vol. 18, no.4, pp. 45-50, July 2004,

[3] I.F. Akyildiz, S. Weilian, Y. Sankarasubramaniam, and E. Cayirci, Wireless sensor networks: a survey, Computer Networks, Vol. 38, No. 4, pp. 393-422, 2002.

[4] C. Perkins, Ad Hoc Networks, Addison-Wesley, Reading, 2000.

[5] E Kaplan, C Hegarty, Understanding GPS: Principles and Applications Second Edition, Artech House, 2006.

[6] David L. Mills, Internet Time Synchronization: the Network Time Protocol, Global States and Time in Distributed Systems. IEEE Computer Society Press, 1994.

[7] David L. Mills, Simple Network Time Protocol (SNTP) Configuration Version 4 for IPv4, IPv6 and OSI, RFC4330, 2006.

[8] J. Elson, L. Girod and D. Estrin, Fine-Grained Network Time Synchronization using Reference Broadcasts, Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI ' 02), 2002.

[9] S. Ganeriwal, R. Kumar and M. B. Srivastava, Timing-sync Protocol for Sensor Networks, Proceedings of the 1st ACM Conference on Embedded Network Sensor Systems 2003.

[10] J. Elson, L. Girod and D. Estrin, The Flooding Time Synchronization Protocol, Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems, 2004.

[11] F. Cristian, C. Fetzer, The Timed Asynchronous Distributed System Model IEEE Transactions on PARALLEL AND DISTRIBUTED SYSTEMS, vol. 10, No. 6, June 1999.