第 XXXI 部

実ノードを用いた大規模なインターネットシミュレーション環境の構築

第 31 部
実ノードを用いた大規模なインターネット シミュレーション環境の構築

第 1 章　はじめに

Deep Space One WG では、実ノードによる大規模ネットワーク実験環境の構築技術や利用方法の研究に取り組んでいる。現在、大規模なネットワーク実験環境である StarBED および AnyBed を対象として議論を進めている。

本報告では、第 2 章で StarBED とその実験支援システムを、第 3 章で AnyBed について述べる。そして、このような実ノードによる大規模実験環境上での実験トポロジ構築手法の 1 つとして、無線環境のエミュレーション手法を第 4 章で述べる。

第 2 章　StarBED

2.1 実験支援システム

2.1.1 はじめに

StarBED[301, 361] は、512 台の PC クラスタとそれらを接続するスイッチで構成された大規模なネットワーク実験環境である。StarBED のネットワークは、管理用と実験用に分離されており、管理用のトラフィックによる実験への影響を防ぐよう設計されている。

StarBED 利用者は実験を行うために、実験に必要な機能を持つノードにソフトウェアを導入し、ノードが接続されているスイッチを設定することで実験トポロジを構築する必要がある。StarBED では物理配線を変更せずに、仮想的に実験トポロジを構築する。この手順をすべて人の手により行った場合、利用者は、実験環境の構築および遂行のために長時間拘束される。また、実験に利用するノード数が多いほど、実験環境全体を思い通りに制御することは困難である。そこで我々はこのような施設で利用者の実験の遂行を支援するシステムを開発している。

2.1.2 支援システム

利用者は、実験トポロジおよび各ノードに利用するソフトウェア、実験の手順（シナリオ）をファイルに記述する。実験の遂行は我々のシステムが自動的に行うため、利用者は設定を記述するだけでよい。

具体的に支援システムは以下の動作を行う。

1. 利用者による設定記述を読み込む
2. ネットワークインタフェースの数やその種類など実験用ノードに必要な機能を認識する
3. 利用できるノード群から必要な機能を持つノードを選択し、実験ノードに割り当てる
4. 各ノードへ OS やアプリケーションを導入する
5. スイッチの VLAN を設定し実験トポロジを構築する
6. 指定されたシナリオ通りに実験を遂行する

ネットワーク実験は、トポロジやパラメータに小さな変更を加えながら複数回行われることが多い。本システムを利用すれば、利用者は設定記述に変更を加えるのみでよい。また、実行されるコマンドは設定記述として残されるため、実験後の結果解析が容易になるとともに、同様の実験を行う際の再現性も向上する。支援システムを利用する場合と利用しない場合の概念図を図 2.1 と図 2.2 に示す。

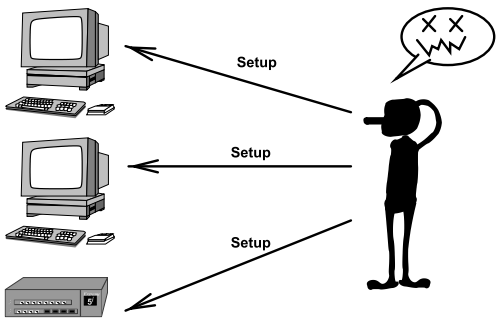

図 2.1.　手動での実験遂行

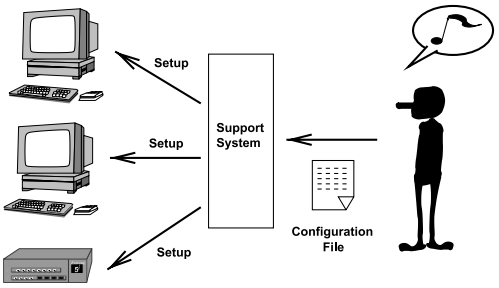

図 2.2.　支援システムを用いた実験遂行

2.1.3 利用例

StarBED ではこれまでにさまざまな研究に利用されている。例として iptraceback に関する研究 [230, 343]、マルチキャストに関する研究 [217, 313] などが行われ、その成果を挙げている。また、本節で述べた通り、StarBED の利用支援システムも充実してきており、利用者は容易に実験を行える。

2.2 実験ノードへの OS およびアプリケーションの導入

概要

インターネットで実際に利用されている機器を用いて、インターネットをシミュレーションすることができる施設として北陸 IT 研究開発支援センター（StarBED）がある。このような施設で、実ノードを用いた大規模なネットワークシミュレーションを行う際には、そこに存在するノードの数も増え、ノードやネットワークの設定に多くの人的資源が費やされる。このようなシミュレーションを行うための環境の構築を、より容易に行うことができれば、ネットワークシミュレーション自体に、より多くの人的資源を割当てることができる。

StarBED にはすでに各ノードに対して、OS を導入するしくみが存在するが、利用者は常にそのシステムを操作する必要があり、人的資源を節約するといった面では、有用であるとはいえない。そこで、我々は、ノードへの OS の導入やネットワーク設定および、ノードに個別なパラメータの設定を自動的に行う機構である、Node Supervisor System（NSS）の設計と開発を行った。NSS により、人的な資源が必要となる作業をまとめ、利用者の拘束時間を短縮することにより、人的資源の削減を図る。

本節では、NSS の設計とその実装について述べ、既存の StarBED への OS の配布システムと比較し NSS の評価を行う。

2.2.1 はじめに

インターネットに導入されるソフトウェアやハードウェアの動作を検証するためには、インターネットに導入されるソフトウェアやハードウェアそのもの（実ノード）を利用してインターネットを模倣した環境（ネットワークシミュレーション環境）を構築し、さらにその上で実際にインターネットへ導入される実装を被検証対象として評価を行わなければならない。

実ノードを用いネットワークシミュレーション環境を構築する際に、多くのノードを適切に設定する必要がある。ネットワークシミュレーションの規模が大きくなるにしたがい、設定が必要なノードの数も増加し、シミュレーション環境の構築に多くの人的資源を費やさなければならない。そこで本研究では、実ノードを用いた大規模ネットワークシミュレーション環境構築の自動化を行うことにより、それに要する人的資源の削減を図る。

2.2.2 実ノードを利用した大規模ネットワークシミュレーション施設

大規模なネットワークシミュレーション環境を構築する際には、多くのシミュレーション環境の要素である PC など（ノード）と、ネットワーク機器に対し操作を行う必要がある。シミュレーションに要するノードの数が増えるにつれ、IP アドレスのような個別のノードに対する特別な設定も増え、ネットワークシミュレーションに用いるネットワーク機器への設定も複雑になる。

StarBED

実ノードを用いて大規模なネットワークシミュレーションを実現する施設として、StarBED がある [361]。図 2.3 に StarBED の概念的なトポロジと、実際に物理トポロジ上にどのようにシミュレーション環境が構築されるかを示した。図 2.3(a) が StarBED の概念的なトポロジである。シミュレーション用の 512 台のノードが用意されており、シミュレーション専用のネットワークと制御用のネットワークへ接続されたインタフェースを持っている。

図 2.3(b) および図 2.3(c) は、StarBED の物理ネットワーク上に、どのようにシミュレーション環境が構築されるかを示している。利用者は各ノードのシミュレーション用のネットワークインタフェースが

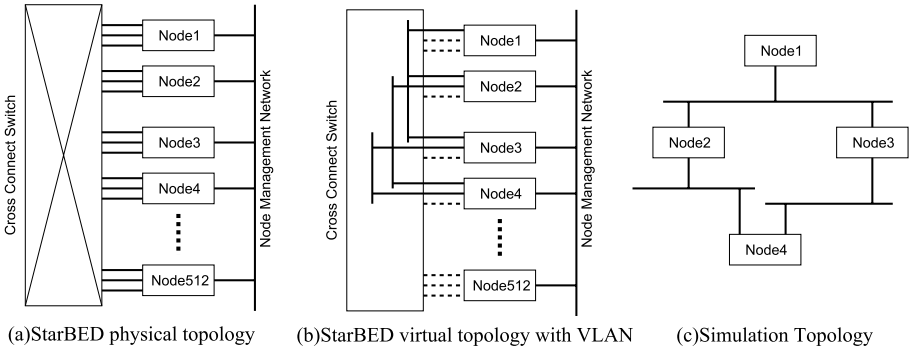

図 2.3. StarBED の概念的トポロジとシミュレーション環境への適応

収容されているクロスコネクトスイッチの VLAN の設定を変更することで、必要なトポロジを構成する。たとえば、図 2.3(c) 中には、3 つのネットワークが存在し、それぞれ Node1 と Node2 と Node3、Node2 と Node4、Node3 と Node4 が属する。これらのネットワークは図 2.3(b) 中のクロスコネクタの VLAN の設定によって仮想的に構成され、その結果図 2.3(c) のような VLAN が構築される。

StarBED のノードには、標準でいくつかの OS がインストールされており、利用者はこの OS を利用することが可能である。ノードは標準で PXE[136] により起動し、Hard Disk Drive（HDD）内のいずれの OS を用いるかは、PXE にて取得するブートローダで選択している。利用者は、施設をグループ単位で時分割利用しており、ネットワークシミュレーション終了後、標準状態に戻す必要がある。

利用者がシミュレーション用にカスタマイズした OS を利用する場合には、それらを必要な台数のノードへ導入する必要がある。また、それらを標準状態へ復帰させる際にも、利用したノードすべてに対して操作を行わなければならない。StarBED を利用し数百台規模のシミュレーション環境を構築した場合には、OS の導入や、標準状態への復帰という作業は非常に大きなコストを必要とする。

標準状態復帰システムの問題点

現在、StarBED にはノードの HDD を標準状態に戻す手段が用意されている。グループごとに標準状態のパーティション単位で用意されたディスクイメージをネットワークを介してコピーするもので、その手順は、以下のようになる。

1. ディスクイメージ配布プログラムを動作させるため配布先ノードへ FD を挿入
2. 個々のノードを操作し、任意のディスクイメージをどのパーティションに導入するかを選択
3. FD を取出し、ノードを再起動
4. ディスクイメージの配布終了後、ノードの個別設定

また、このシステムは、人間が逐次 GUI により操作することしかできないため、ディスクイメージおよびパーティション選択の自動化は不可能である。この既存の標準状態復帰システムは単一のノードに対しては有用であるが、数百台規模のノードに対してこのシステムを利用するのは、同等の操作を複数のノードに対して行う必要があるためコストが大きい。また、起動に FD を利用することから物理的にノードにアクセスする必要があり、多くの時間と、労力を要する。

本節では、ネットワークシミュレーション環境を構築する際に、人が拘束される時間を人的資源の尺度とする。

既存のシステムを利用すると、利用者はノードに対して、雛形ノードのディスクイメージを導入する際には、OS のコピー設定の時点から、個別の設定を行うまで、連続的に操作を要求される。したがって、人的資源の観点からは非常にコストが大きいといえる。

我々は、ノード設定の自動化を図り、人的資源の削減を図るために Node Supervisor System を提案し実装した。

2.2.3 NSS の設計

本項では、Node Supervisor System（NSS）の設計について述べる。

ノードへの OS の配布

ノードへ OS を導入する際に、複数のノード間で利用する OS が同一であれば、OS の基本部分を構成するファイル群と、個々のノード間で異なるパラメータを設定するためのファイル群に分けて管理する。ここでパラメータとは、該当ノードで、動作していなければならないアプリケーションや、IP アドレスやホスト名などのノード固有の設定を指す。これらを、別に管理することで、保存しておかなければならないファイルの合計サイズなどが小さくなり、ファイルサーバに必要なディスク容量や、転送しなければならないファイルサイズが小さくなるというメリットがある。したがって、各ノードに基本となる OS を導入後、各ノードで個別の設定を行うファイル群をその上に適用し、個別のパラメータを設定を行う方法を採用した。

各ノードに個別のパラメータを設定を行うためのファイル群には、各 OS に用意されている設定ファイルや、コマンドを実行し各パラメータを設定するためのシェルスクリプトおよび、さまざまなアプリケーションの本体やその設定ファイルが含まれる。これらのファイルを実行または、特定の位置に配置することで、ノード個別のパラメータの設定を行う。

メモリのみ（HDD 無し）で動く OS の導入

前述の通り StarBED では、利用したノードはシミュレーション終了後に初期状態に戻しておく必要がある。したがって、HDD を用いると、初期状態に戻すというコストがかかる。また、標準で用意されている OS 以外の OS を利用する場合も、OS の導入というオーバーヘッドは非常に大きい。

HDD の利用なしにシミュレーションを行うため、メモリのみで動く OS を用意した。この手法を用いた場合にはノードは起動する際に TFTP[285] を利用して、OS の動作に必要なカーネルとディスクイメージを取得する。TFTP は、ファイルを転送する際そのファイルをブロックごとに転送する。TFTP のブロック番号は、2 bytes で表されるので 65535 のブロック数をカウントすることが出来る。512 bytes が TFTP のブロックの単位であるので、TFTP で転送できる最大のファイルサイズは 65535 blocks × 512 bytes = 33553920 bytes ≒ 32 Mbytes 程度となる。よって、この上限以下のディスク容量で十分なノードならばこの手法は有用であるが、より大きなディスクスペースを要求するノードの場合は HDD を用いなければならない。

メモリのみで動くシステムを利用する際には、起動時に用いるカーネルとシミュレーションに必要な機能を持つディスクイメージを、あらかじめ用意しておく必要がある。個々のノードに依存するファイルは、OS の起動後 FTP[258] を用いて取得する。

HDD を用いて動作する OS の導入

HDD を用いるノードに対し OS を導入する際には、雛形のノードを用意し、そのディスクイメージを取得し、それを対象ノードへ配布する。

HDD に読み書きを行うにあたり、HDD のディスクイメージを取得し HDD のパーティションへの書き込みを実現するシステム動作は、HDD から独立したもので無くてはならない。そのため、メモリのみで動く OS をネットワークブートすることで HDD から独立した環境を実現する。この、メモリのみで動作する OS を用いてディスクの読み書きを行う。

ノードへの OS 導入の手順

ノードへ OS を導入する際に必要な設定作業を事前に行い、OS の導入が終了するまで、人間がその場に拘束されないような手順を採用した。この手順を以下に示す。

1. シミュレーションで利用する OS を 1 台のノードに導入し、雛形となるノードを作成
2. 上述のノードで、ディスクイメージを作成するための OS を動作させるため、メモリのみで動く OS が起動するように設定
3. ディスクイメージを作成しファイルサーバへ転送
4. NSS の設定ファイルを生成
5. NSS を実行

2.2.4 NSS の実装

ここでは、前述の設計にしたがって、実際に行った実装についての詳細を述べる。

HDD を用いて動作する OS のための実装

メモリのみで動く OS 上で、dd コマンド、ftp コマンドおよび gzip コマンドを利用し、パーティション単位のディスクイメージ配布を実現する。

パーティション単位のディスクイメージ配布の概略を図 2.4 に示す。指定された HDD のパーティショ

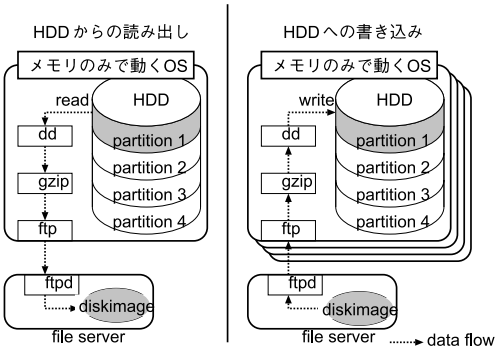

図 2.4. HDD への OS 導入

ンのディスクイメージを読み取る場合、dd で読み取ったデータを gzip で圧縮しつつ ftp を用いてそのディスクイメージをファイルサーバに保存する。一方、HDD にディスクイメージを書き込む場合、FTP サーバから取得した gzip により圧縮されたディスクイメージを伸長しながら、dd で HDD の任意のパーティションにデータを書き込む。

NSS の動作説明

NSS は、管理ホストで動作する Node Manager（NM）と、ノードで動作する Node Agent（NA）からなる。

NM はノードごとの起動方式、OS の種類、OS を導入するパーティション、HDD に導入するディスクイメージファイル名、メモリのみで動作させる際に用いるカーネルのファイル名など、前もって利用者により設定された内容から、ノードで動く NA に対し指示を出し、ノードへの OS の配布、ノード個別のパラメータを設定する。

また、NM はシミュレーションに用いるノードの状態を保持し、ノードで動作する NA に対し指示を出す。これにより、ノードへの OS 導入の自動化を実現する。

NSS が HDD を用いるノードに対し、OS の導入および個別のパラメータの設定を行う一連の流れを図 2.5 に示す。NM は、HDD のディスクイメージを書き込むノードを Wake on LAN を用いて起動させる。HDD を用いるノードはディスクイメージを導入するため、まずメモリのみで動作する OS で起動し、メモリのみで動作する OS を起動するためのブートローダを PXE が TFTP を利用して取得する。続いてブートローダがメモリのみで動作する OS の動作に必要なカーネルとディスクイメージを取得する。HDD に対しディスクイメージを書き込むための OS が起動すると、その上で動作している NA から NM に対して、ディスクイメージ取得の準備が整ったことが報告される。この報告を受け取った NM は、NA に対し、どのパーティションに、どのファイルを取得し書き込むかを指示する。NA は、それにしたがい HDD にデータを書き込む。ディスクイメージの書き込み終了後、ノードは再起動し先ほど HDD に書き込んだパーティションからノードを起動するためのブートローダを取得する。HDD を用いて動作する OS の起動後、その上で動作している NA が、OS が起動したことを NM に報告する。NM は、その報告を受け取ると、NA に対しノード個別の設定を記述したファイル（差分ファイル）を取得し適用するよう指示を行う。

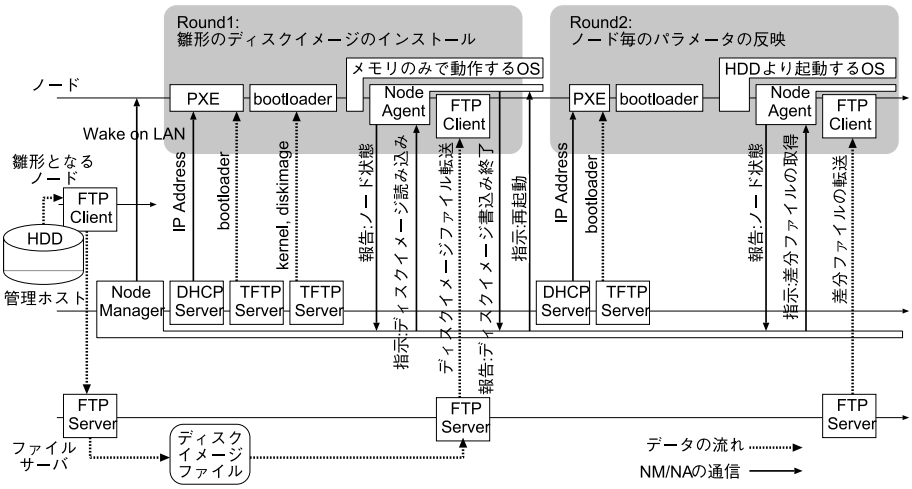

図 2.5. NSS による HDD を用いて動作するノードへの OS 配布

2.2.5 NSS の評価

ここでは、我々が提案、実装した NSS を利用した場合と、StarBED の既存の OS 配布システムの比較および NSS の評価を行う。

NSS と StarBED の既存 OS 配布システムの比較

StarBED の既存の OS 配布システムと、NSS の機能の比較を行う。この結果を表 2.1 に示す。

表 2.1.　既存 OS 配布システムと NSS の比較

	StarBED 既存	NSS
起動媒体	FD	netboot
local での操作	○（GUI）	○（CUI）
ネットワークごしの操作	×	○
ノード個別のパラメータ設定	×	○

StarBED の既存の OS 配布システムは、FD を利用して起動する。そのため、利用者は OS を導入するノードに物理的にアクセスする必要がある。また、操作を行う際、個々のノードごとに操作を行う必要があり、ネットワーク経由での操作も不可能である。NSS は、ノードへの OS の導入を自動化するために、その起動はネットワーク経由で行い、また、ネットワーク越しの操作を行うことで、ノードの集中管理を行う。

NSS を用いた場合、シミュレーションに用いるノードは、PXE により起動するため、個々ノードが起動の際にネットワーク経由で取得するブートローダを、ファイルを配布するサーバ側で選択可能となる。これにより、個別ノードの HDD のいずれのパーティションから起動するか、メモリのみを用いて起動するかの選択を一元管理できる。

NSS と StarBED の既存 OS 配布システムが要する時間の比較

NSS を利用した場合と、StarBED の既存 OS 配布システムを利用した場合の、人間の拘束時間を計測した。両システムでも、それぞれのシステムが用いるディスクイメージはすでに存在するものとする。それぞれのディスクイメージをノードの HDD に書き込み、ノードの設定を行うまでの過程について計測を行った。なお、個々のノードに対し書き込むディスクイメージはそれぞれのシステムごとに全ノードで同様のディスクイメージを用いた。NSS のディスクイメージファイルは 4 GB のものが 1.5 GB に圧縮されたものを用いた。同様に、StarBED 既存の OS 配布システムのディスクイメージファイルは、4 GB のパーティションのものが 1.5 GB に圧縮されているものを用いた。NSS の場合は NSS の設定ファイルを利用者が記述し、NM を実行するまでの時間を拘束時間とし、StarBED 既存の OS 配布システムは FD 挿入から全てのノード個別の設定が終わるまでを拘束時間とした。その際の拘束時間の結果を図 2.6 に示す。

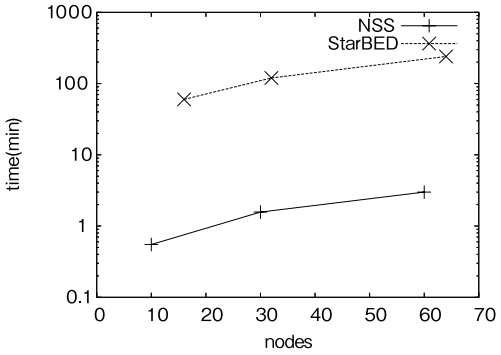

図 2.6.　処理するノード数に対する拘束時間

2.2.6 考察

HDD を用いるノードの構築について StarBED 既存の OS 配布システムと NSS で比較を行った。計測結果を見ると、設定を行う利用者の拘束時間は既存の StarBED の OS 配布システムと NSS を用いた場合の両システムとも、扱うノードの数が増えれば増加する。しかし、図 2.6 からわかるように、その増加の傾きが既存の StarBED の OS 配布システムと比べ NSS の方が小さい。これは、NSS が人の判断を要する部分を初期の NSS の設定を作成する個所に集中させ、一定の処理ごと（HDD へのディスクイメージ導入終了後のノードの再起動など）に人の判断を仰がずともよいためである。これにより、StarBED 既存の OS 配布システムに比べ、NSS を利用した場合はユーザが拘束される時間を大幅に節減できることが、明らかである。

2.2.7 今後の課題

現在の NSS は、ノード共通部分の転送に FTP を用いているため、その転送はユニキャストである。そのため、ネットワーク帯域の上限の都合上、現状

3 並列が上限となる。

シミュレーション環境の設定に要する時間の短縮を実現する 1 つの手法として共通部分配布のマルチキャスト化を行う。

2.2.8 おわりに

本節では、実ノードを用いたシミュレーション環境の構築は、その規模とともに困難になることをまず確認した。その上で、実ノードを用いて数百台規模のシミュレーション環境を構築するための施設である、StarBED について紹介し、そこで利用されているノードへの OS 配布システムの詳細と問題点を挙げ、これを解決するため NSS の提案および実装についての詳細を述べた。また NSS を実装し、既存のシステムとの比較を行い、その有効性を確認した。最後に、現状の NSS の問題点について考察し、これを回避する方法について今後の課題として述べた。今後は、現状の NSS の問題点を回避するために、開発を進めていく予定である。

2.3 実験の自動実行

概要

インターネットに導入されるソフトウェアを開発するためには、その挙動を検証するための環境が必要である。実験環境の構築のためソフトウェアシミュレータが広く利用されているが、多くのソフトウェアシミュレータではインターネットに導入される実装そのものを利用することはできず、実ノードによる実験環境が必要となる。

実ノードによる実験環境で、利用者が想定する実験手順を手動で実現するのはさまざまなコスト面から困難である。我々は、利用者が指定した実験手順にしたがって自動的に実験を駆動するためのシステムの設計を行った。またこのシステムを実装し StarBED 上で動作検証を行うことで、その挙動を確認した。

2.3.1 はじめに

新たな技術をインターネットに導入する際には、すでに運用されているサービスへの影響を防ぐため、最低限の検証を行う必要がある。現在のインターネット上ではさまざまなサービスが行われており、未検証の技術の導入によるサービスの停止といった危険は回避されなければならない。新たな技術の検証用途にはソフトウェアシミュレータが広く利用されているが、多くの場合ソフトウェアシミュレータは、シミュレーション専用のプログラムコードを要求する。したがってソフトウェアシミュレータでの検証は、実際にインターネットに導入されるソフトウェアの検証ではない。実際にインターネットに導入される実装の、プログラムのバグや仕様に記述されていない動作までを含めた検証が必要があり、このような検証を行う場合は、実ノードを用いて実験環境を構築し、その上に被検証対象を組み込んで実験を行う必要がある。

一般的に実験は、実験の手順（シナリオ）にしたがって進められる。ソフトウェアシミュレータを利用する場合には、利用者は前もってシナリオを記述しておくことで、実験中は別の作業を行うことができ、その拘束時間を回避できる。また、これにより、再度同様の実験を行えることと、実験の再現が可能であるというシミュレーションの長所の要因にもなる。

実ノードを用いた実験環境を利用する際にも、利用者の指定通りに実験ノード上でプログラムを起動するしくみが必要である。手動で各プログラムを実行することもできるが、利用者は実験が実行されている間、常にシナリオに沿ってコマンドを入力せねばならないことになり、実験に拘束されることになる。また、実験ノードの数が多ければ多いほど、各実験ノードの制御は複雑になり、プログラムの実行ミスなどが当然発生する。また、実験の再現も困難である。

本節では、ユーザにより指定された通りのタイミングで自動的にプログラムを起動するシステムの 1 つとして、StarBED[301, 361] で利用されている実験の自動遂行プログラムの設計について述べる。

2.3.2 実ノードによる実験環境

まず本項で、我々の提案する自動実験遂行機構が前提とする実ノードによる実験環境について述べる。

実ノードによる大規模な実験環境の必要性

ソフトウェアシミュレータを用いれば、大規模な実験環境を容易に模倣することができるが、ソフトウェアシミュレータは、ネットワークや各実験ノードの挙動を巨視的に模倣するものであり、ネットワーク上のさまざまな要素の詳細な動作の検証は行えない。また、実際にネットワーク上で動作している実装を利用できないことが多く、ソフトウェアシミュ

レータ上で観測された実験の結果と実ネットワーク上での挙動が異なる可能性がある。

実際にインターネットに導入するソフトウェアやハードウェアの実装の検証を行うためには、実ノードにより構築された実験用のネットワーク上で、対象となる実装を動作させる必要がある。

NSE[225] のように、ソフトウェアシミュレータと実ノードによるネットワークを接続し、実験環境を構築する手法も提案されている。しかし、検証対象のアルゴリズムや実装がどのような外部要因により影響を受けるかを予測することは困難であるため、両者を実験ネットワークのどの部分に割り当てるかを決定することも困難である。

一般的に、実験対象となる実装の挙動や、その実装による他のサービスへの影響を前もって知ることが困難なため、できるだけインターネットに近い環境で実験を行うことが重要である。対象となる環境としてインターネットを想定した場合は、非常に多くの実ノードにより構築された環境が必要となる。

また、ソフトウェアシミュレータではハードウェア実装の検証は当然行えないが、実ノードによる実験環境ではこれも可能となる。

StarBED

実ノードによる大規模な実験を行うための施設の 1 つとして StarBED がある。StarBED には 512 台の実験用ノードが用意されており、それぞれのノードは VLAN や ATM のパスが扱えるスイッチにより接続されている。実験用のトラフィックと管理用トラフィックを分離するため、各ノードには最低 2 つのネットワークインタフェースが用意されており、実験用ネットワークと管理用ネットワークに接続されている。StarBED の概念図を図 2.7 に示す。管理側のネットワークには実験ノードを管理、観測するための管理用ノードが接続されており、我々の提案する実験遂行機構も管理用ネットワークに接続されているノードで動作することを想定する。

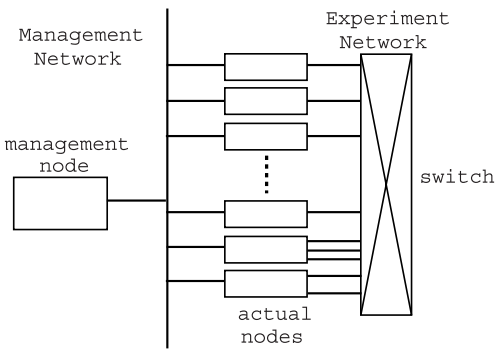

図 2.7.　StarBED の概念図

StarBED では、実験トポロジは、物理的なトポロジに変更を加えることなく、VLAN などを利用し仮想的に構築される。物理的なトポロジを変更しないことで、実験環境構築に必要な手順を節減し、施設を利用できる時間が長くなる。

実ノードによる実験環境での実験遂行手順

一般的な実ノードによる実験環境での実験の遂行手順を以下に示す。

1) **ノードの物理配線** 実ノードを用意しケーブルの配線など物理的な接続作業を行う。
2) **実験ノードとネットワークの設定** 実験に必要な OS やアプリケーションを実ノードにインストールし、IP アドレスの設定など、実験に必要な設定を行う。また、必要であればスイッチの設定なども行う。
3) **実験の遂行** 実験者が意図するシナリオに沿って実験を遂行する。
4) **ログの収集および解析** 実験の結果ログを各実験ノードやスイッチから収集し解析を行う。

本節では、1)、2) についてはすでに完了していることを想定し、3) の実験の遂行部分に着目し、指定されたシナリオに沿って自動的に実験を行うシステムの設計について述べる。

2.3.3 実験の自動駆動

本項では、実験を自動的に遂行するためのシステムについて議論を行う。

現状での問題点

多くのソフトウェアシミュレータでは、利用者によって計画されたシナリオの記述にしたがって実験が遂行される。しかし、実ノードを用いた実験環境には、このようなシステムが存在しない。実験をシナリオ通りに遂行するシステムが存在しない場合、利用者がそれぞれの実験ノードにログインしてプログラムを起動する方法や、リモートノードから rsh[159] や ssh[331] を利用してプログラムを起動することになる。以下にその問題点を挙げる。

利用者の時間的コストの増大 利用者は実験実行中、常にコマンドの入力をする必要があるため、実験終了までの間、実験に拘束される。

再現性の低下 利用者の手によってコマンドを実行するため、全く同じ実験を行う場合でも、同じ時刻にコマンドを実行することは困難である。したがって 2 回の実験での差異が大きくなり、実験の結果へ影響を与える可能性がある。

実行誤り 利用者が 1 つずつコマンドを入力するため、予定していた手順を誤る可能性がある。この場合、誤りの存在自体を知ることも困難である。

ノード間同期 あるノードの挙動をトリガとして、別のノードで処理を開始したい場合に手動やスクリプトでの実行では、各ノード間でタイミングを合わせることは困難である。

我々は以上の問題点を解決するためのシステムを設計した。

モデル設計

実験のシナリオは前もって利用者により記述されることを前提とした。利用者は実験環境を構築およびシナリオの記述のみ行い、実験の遂行自体は我々のシステムにまかせることで、実験に拘束される時間を削減できる。

シナリオにしたがって各実験ノードのプログラムを起動するための方式について検討した。以下で検討した各方式について述べる。また、シナリオの設定ファイルを読み込み、シナリオ全体を把握しているプログラムをシナリオマスタと呼ぶ。

ノード自律モデル 実験前にシナリオをシナリオマスタから各実験ノードに配布し、各実験ノードは配布されたシナリオに沿って自律的にプログラムの起動を行うモデルである。このモデルでは、各実験ノードが自律的にプログラムを起動するため、各実験ノード間の同期が困難である。ノード間でメッセージを交換し同期をとる方法も考えられるが、それぞれのノードへのメッセージ送出のタイミングや、受け取ったときの処理をシナリオにまとめて記述することになり混乱の元になりやすい。また、この方法では各実験ノードがメッセージ交換が必要となる別ノードへのセッションをその台数分保持する必要があり、実験自体に影響を及ぼす可能性がある。図 2.8 にノード自律モデルを示す。

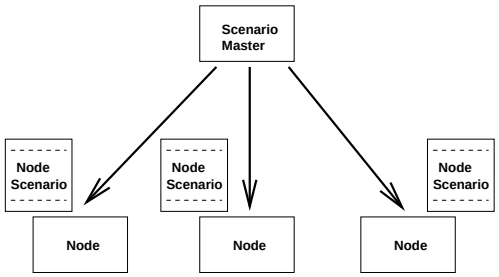

図 2.8. ノード自律モデル

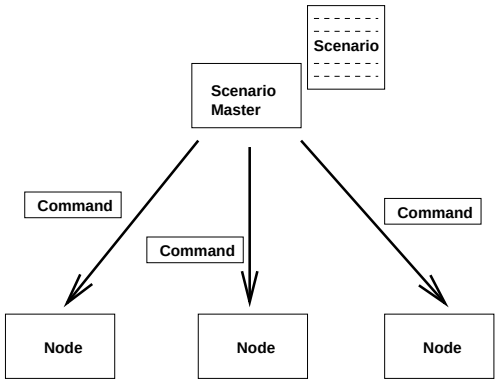

図 2.9. コマンド送信モデル

コマンド送信モデル シナリオマスタからプログラム実行のタイミングごとに、実行するコマンドを各実験ノードに送信する。各実験ノードは受信したコマンドを単に実行するというモデルである。このモデルでは、サーバは実験ノードの台数だけのセッションを制御しなければならず、大規模な環境になるほどシナリオマスタのセッションの管理のための処理負担が大きくなり実験へ影響をおよぼす可能性がある。また、大規模でなくてもあるタイミングに処理が集中した場合には、シナリオマスタの負荷が大きくなり、設定されたタイミングと実際にコマンドが実行されるタイミングに無視できない差異が生じる可能性がある。シナリオマスタと、実験ノードがメッセージを交換することで、実験ノード間の同期をとることもできるが、この処理を行った場合はさらにシナリオマスタの負荷が高くなる。各ノードが直接メッセージを送り同期を図る方法もあるが、ノード自律モデルと同様の問題が発生する。このモデルを図 2.9 に示す。

これらを踏まえて、我々は前述した 2 つの手法の折衷案を採用した。我々のモデルでは、シナリオマスタは実験前に各実験ノード用のシナリオを配布し、

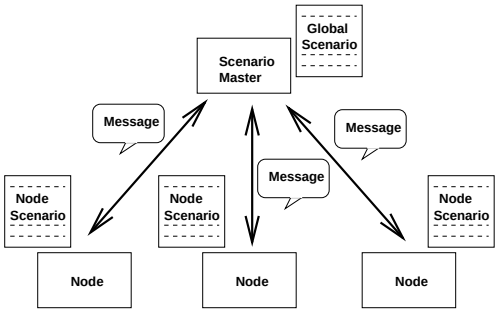

図 2.10.　提案モデル

実験ノード間の同期のみシナリオマスタを通して行う。OS の導入や設定が済んだ実験ノードは、シナリオを受け取るとそれを実行し、シナリオ記述に沿ってシナリオマスタへのメッセージの送信や、シナリオマスタからメッセージを受信するまで、シナリオ遂行を停止するといった処理を行う。また、シナリオマスタは専用のシナリオを持っており、実験ノードからのメッセージを受信するまでのシナリオ遂行の停止や、実験ノードへのメッセージの送信を行う。シナリオマスタのシナリオを実行することにより、各実験ノード間の同期をとる。

このモデルでは、実験ノード間同期が可能となるばかりではなく、実験ノード間の同期が必要な時のみ、メッセージ交換を行うことで、毎回コマンドを送信する場合よりもシナリオマスタの負荷が軽減できる。

また、シナリオをそれぞれの実験ノードへ個別に入力するのは非常に時間がかかる作業であるが、設定ファイルをシナリオマスタで一括して読み込み、各実験ノードごとのシナリオに分割して実験ノードへ送ることでこの時間も短縮できる。このモデルを図 2.10 に示す。

シナリオの同期機構を考慮に入れ設定言語の設計を行った。基本的には設定ファイルに実行するプログラムと実行されるタイミングを列挙する形とする。タイミングは別ノードの挙動をトリガとする方式および、時刻をトリガとする 2 方式をサポートすることとする。これにより、より柔軟な実験遂行が期待できる。

ns-2[298] は広く利用されているソフトウェアシミュレータであり、ns-2 に採用されている設定記述を用いれば、これまでソフトウェアシミュレータにて行われていた実験をそのまま実ノード環境に移行できるが、この記述方法ではトリガにノードのイベントを利用できない。このため、新たな独自の言語の設計を行った。

言語設計

利用者にわかりやすくシンプルであるということを念頭におき言語の設計を行った。シナリオ記述には 2 種類あり、各実験ノードで実行するためのシナリオとシナリオマスタのシナリオがある。実験ノード用のシナリオをノードシナリオ、シナリオマスタ用のシナリオをグローバルシナリオと呼ぶ。基本的に、ノードシナリオは実験の遂行のためのものであり、グローバルシナリオは実験ノード間の協調のために利用する。

ある程度の規模のネットワーク実験を行う場合、各実験ノードはその役割により、グループにわけられると考えた。同じグループに属する実験ノードに同様のシナリオを個別に記述するのは冗長であるため、基本的にノードの属性はグループごとに定義することとする。ノードシナリオでは基本的に、プログラムの実行と別の実験ノードとの同期のためのシナリオマスタへのメッセージ送出、およびノードマスタからのメッセージを待ち、シナリオ遂行の停止を行う。またグローバルシナリオでは、実験ノードへのメッセージの送信および、複数の実験ノードからのメッセージを待機してのシナリオ実行の停止を行う。

ここで、実験ノード 2 台を用意し一方からもう一方の実験ノードに対して ICMP echo request を送信するというだけの簡単な例を示す。この例の動作手順を以下に示す。

1. 実験ノードがシナリオを受け取るとシナリオマスタに対して、“setupdone” というメッセージを送る。
2. これを受信したシナリオマスタは、実験ノードに対して “start” というメッセージを送信する。
3. このメッセージを受信した実験ノードは、別の実験ノードに対し ping request を 10 回送信する。
4. 実験ノードは ping が終了するとシナリオマスタに対して “finished” というメッセージを送出する。

図 2.11 にこの例のノードシナリオを示す。ただし、前述の通り本システムは、あらかじめ実ノードで構築された環境上に、VLAN などで仮想的に実験環境を構築し、この実験環境でシナリオを実行すると

```
 1 nodeclass clclass {
 2   scenario {
 3     send "setupdone"
 4     recv msgfromsrv
 5     msgswitch msgfromsrv {
 6       "start" {
 7         wakewait "ping" "ping" \\
 8           "-c" "10" "somehost"
 9         send "finished"
10       }
11     }
12   }
13 }
```

図 2.11. ノードシナリオの例

```
 1 scenario {
 2   recv pingclient[0] msgfromcli
 3   msgswitch msgfromcli {
 4     "setupdone" {
 5       send pingclient[0] "start"
 6       sync {
 7         msgmatch pingclient[0] "finished"
 8       }
 9     }
10   }
11   exit
12 }
```

図 2.12. グローバルシナリオの例

いう大きなシステムの一部として設計を行った。したがって、本来ノードクラスの設定にはシナリオ以外にもさまざまな設定が存在するが、本節はシナリオについてのみ着目し、その他の部分については省略している。

実験ノードの集団は nodeclass を用いて宣言される。ブレースで囲まれた部分が nodeclass の属性の定義である。ノードシナリオ部分で、send を用いると自動的にシナリオマスタ向けにメッセージを送信する。また recv でシナリオマスタからのメッセージを待ち、メッセージを受け取るとその後に指定した変数にメッセージを格納する。msgswitch で格納されたメッセージ内容により挙動を変更できる。wake および wakewait はプログラムを起動するために利用され、wake が子プロセスを起動しプログラムを実行させるのに対し、wakewait は自分自身でプログラムを起動しその終了を待つ。実験ノードは 2 台必要であるが、1 台は ping に応答し ICMP echo reply を返すだけでいいため、とくにシナリオは用意していない。この例のグローバルシナリオを図 2.12 に示す。

シナリオマスタは、実験ノードから “setupdone” というメッセージを受信するまでシナリオの遂行を停止し、このメッセージが届くと “start” というメッセージを実験ノードに対して送信する。その後、実験ノードから “finished” というメッセージが届くまで待機する。

図 2.12 の 6 行目から 8 行目で宣言されている、sync と msgmatch を用いると、複数の条件が揃うまでシナリオの遂行を停止できる。この 2 つのキーワードはグローバルシナリオに記述される。sync はその後にブレースで囲まれたブロック構造を持ち、ブロック内に記述された条件がすべて整うまでシナリオを停止する。msgmatch は sync のブロック構造中に記述されることが多く、実験ノードからのメッセージを待つ。

2.3.4 検証

我々は 2.3.3 項で提案したモデルにしたがった自動実験遂行機構を実装し、StarBED 上で評価実験を行った。結果が容易に想像できる実験を行い、その結果が正しいかを検証した。この実装は UNIX 系 OS 上で動作し、スイッチやルータなどで動作する専用ファームウェアは、開発環境が提供されていないことが多いため、現在は対象外としている。しかし本モデルのしくみ自体は簡単なものであるため、実装さえできればスイッチやルータ上でも利用できると考えている。

検証実験内容

帯域ベンチマークプログラムである netperf[300] を用いて 2 台の実験ノード間の帯域を測定した。この時利用した設定ファイルを図 2.13 に、また実験の流れを図 2.14 に示す。この例でも、図 2.13 中の実験ノードの IP アドレスやディスクイメージをインストールするための設定部分は省略した。

25、26 行目のキーワード nodeset で nodeclass で定義したノードを実際に何台用意するかを指定する。25 行目のクライアントの宣言文を例にとると、clclass のノードを client という名前で 1 台生成している。nodeset によって生成されたノードは、宣言された際に設定された名前とブラケットで囲まれた数字によって表現される。たとえば 31 行目のように client[0] と指定される。その他の、シナリオの記述はすでに説明したため省略する。

この実験では、netperf のサーバ 1 台とクライア

```
 1 nodeclass svclass {
     ...
 2   scenario {
 3     wake "/sim/netserver" "/sim/netserver"
 4     send "serverstarted"
 5     recv msg
 6     msgswitch msg {
 7       "quit" {
 8        wakewait "/usr/bin/pkill" \\
 9          "pkill" "netserver"
10          exit
11        }
12     }
13   }
14 }
15
16 nodeclass clclass {
     ...
17   scenario {
18     send "clisetupdone"
19     recv dst
20     wakewait "/sim/netperf" "/sim/netperf" "-H" dst
21     send "cdone"
22   }
23 }
24
25 nodeset client class clclass num 1
26 nodeset server class svclass num 1
     ...
27
28 scenario {
29   sync {
30     msgmatch server[0] "serverstarted"
31     msgmatch client[0] "clisetupdone"
32   }
33
34   send client[0] haddr(server[0].netif[0].ipaddr)
35   sync {
36     msgmatch client[0] "cdone"
37   }
38   send server[0] "quit"
39   exit
40 }
```

図 2.13.　検証用設定ファイル例

ント 1 台の計 2 台を用いる。サーバが起動すると netperf のサーバプログラム netserver を起動し、実験ノードが起動すると netperf のクライアントプログラムである netperf を実行する。この時 netperf を実行するタイミングなどを、シナリオマスタを通したグローバルシナリオを用いて調整している。このシナリオでの動作を以下に示す。

1. サーバが起動し netserver を起動するとシナリオマスタに対して “serverstarted” という文字列を送信する（4 行目）。
2. クライアントが起動するとシナリオマスタへ “clisetupdone” という文字列を送信する（18 行目）。
3. シナリオマスタはこの 2 つのメッセージの受信を待ち（29 行目から 32 行目）、クライアントにサーバの IP アドレスを通知するメッセージを送信（34 行目）する。
4. これを受信したクライアントは指定された IP アドレスのサーバへ向かって netperf を実行する（19 から 20 行目）。
5. netperf が終了するとクライアントはシナリオマスタに “cdone” という文字列を送信する（21 行目）。
6. この文字列を受信したシナリオマスタはサーバへ対して “quit” という文字列を送信（38 行目）する。
7. このメッセージを受信したサーバは netserver プロセスを終了する。

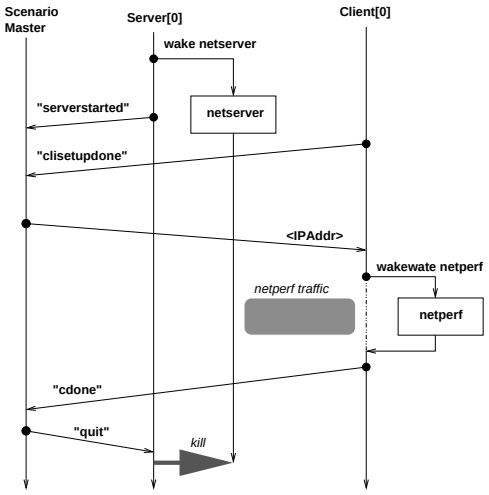

図 2.14.　検証実験の流れ

検証結果

利用した実験ノードは 100Base-TX のリンクで接続されており、帯域が 90 Mbps 程度という実験結果が得られた。このことから、各実験ノードでのプロセスの起動が成功しており、また実行結果に対する影響も小さいと考えられる。この実験は 100 台規模でも行い、我々のシステムが正しく動作することを確認している。利用者は設定ファイルを記述するだけで良いため、拘束時間も大幅に削減できた。

2.3.5 今後の課題と制限事項

我々のシステムは基本的に実験環境上の PC を操作することを想定しており、一般的な OS が動作しないようなノードを操作することはできない。しかし、ノードの協調は簡単なメッセージパッシングによって行われているため、PC 以外のノード上でも簡単に実装できると考えられる。

現在、本機構により、一般的な実験の駆動は行えるが、シナリオマスタで一度に 1 つの同期のみしか扱えず、複数のコンテキストによる同期を同時に行えないため、複数のコンテキストを同時に扱う場合は、シナリオ記述が複雑になってしまう。また、実験実行中にノードが故障した場合などの、例外処理についても未検討である。今後は以上を扱えるよう言語の拡張を進めていく。また、その他の実験支援システムとの協調についても、今後拡張を続けていく予定である。

2.3.6 まとめ

インターネットに投入される技術は、最終的に投入される実装自体の検証が行われなければならない。このような実験を行う際には実ノードによる実験環境に、実ノードの検証対象を組み込んで実験を行う必要がある。このような要求を満たすため、実ノードによる大規模なネットワーク実験環境および、その環境での実験を支援するためのシステムが求められている。

本節では支援システムのうち、前もって用意された実験のシナリオを自動的に遂行するためのシステムのモデルについて議論を行った。その結果、我々は各実験ノードのシナリオを前もって各実験ノードに配布、実験ノードは配布されたシナリオを実行し、シナリオマスタにクライアントがメッセージを送信することにより、クライアント間の協調をとるモデルを採用した。またシナリオ実行の提案モデルを充足するための、設定言語を提案した。最後に提案した手法を実装し、StarBED において実験を行いその有効性を確認した。

2.4 実験環境の資源管理

2.4.1 はじめに

大規模なネットワーク実験を行う際には、ネットワーク実験に利用できるノードの管理が重要である。実験に利用可能なノードは、基本的に、実験に利用されていない状態と利用されている状態という 2 つの状態を遷移する。しかし、ノードは故障する可能性があり、多くのノードを取り扱うほど、存在する故障ノードは多くなる。

故障ノードを実験トポロジの一部に利用した場合、正しい実験結果が得られない可能性がある。さらに、故障ノードが存在すること自体の認識、および認識した後の故障ノードの特定に長い時間がかかり、実験の遂行自体に大きな影響を及ぼす可能性がある。したがって、故障検出の手法と、発見された故障ノードを実験から排除する手法の確立が必要である。

2.4.2 資源管理モデル

資源管理システムは、基本的に利用者から要求された条件を満すノードを検索し利用者へ貸し出す。またそれに加え、ノードの故障を検知し、故障ノードを利用不可の状態へ移動した後、故障を復旧するための処理を行う。あるノードが複数の利用者の実験に属することを防ぐため、資源管理システムは、1 つのネットワーク実験環境で 1 つだけ動作する。

ノード状態の遷移

我々はノードの状態には、待機、貸出し、利用禁止、利用停止の 4 つを設けた。ノードの状態遷移を図 2.15 に示す。ノードの資源管理システムの管理下に置かれる際の初期値は待機状態とする。資源管理システムは、利用者からノードの貸し出し要求を受けると、利用者に貸出し可能で待機状態にあるノードから、要求された条件を満たすノードを必要数利用者に貸し出す（貸出し状態）。利用者により実験利用に好ましくないと判断されたノードは、故障の可能性がある場合は利用停止状態に移され、実験の性格に合わない場合は待機状態へ戻される。利用停止状態のノードに対しては、管理者による検証が行われ、その結果、故障が発見された場合は利用禁止状態

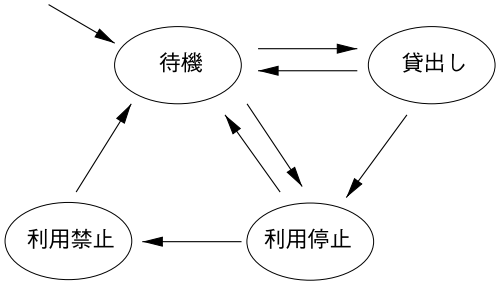

図 2.15. ノードの状態遷移図

に移行し、修理が完了後、待機状態へ遷移する。また、健全であることが確認されたノードは再び待機状態へ遷移する。

ノードの故障検知とその復旧

ノードの故障検知は、貸し出された利用者が利用停止状態に遷移させる場合と、何らかの健全性確認システムと協調し、その結果を反映させる場合がある。健全性確認システムにもさまざまな種類があり、現在考えている確認手法には、ping を利用しての疎通確認や各ノードの vmstat や netstat の結果、特定ノードのトラフィック監視による状態確認、SNMP を利用した状態管理などがある。

このようなチェックを定常的に行うことで、ある程度の故障を検知できる。また、実験遂行システムと協調することで、利用者の実験中に不審な動作をするノードを発見した場合は、利用者へ通知するとともに、実験によってはノードの置き換えなどを行う。システムが故障を検知した際に行う処理は、利用者が指定できる必要がある。

基本的に利用者による実験中の検証は、実験による影響を受ける可能性があるため、利用者が不審な動作をするノードを発見した場合は、ノードを利用禁止状態に遷移させるのではなく、利用停止状態へ遷移させるにとどめ、実際に故障しているかどうかの検証は管理者が行う。また、夜間や週末に大規模な検証を行うことも可能である。この管理者による一斉検証は、検証対象のノードを利用停止状態へ遷移させた後に行い、その結果、検証対象のノードは待機状態または利用禁止状態へ遷移する。これらの検証の結果、故障が発見された場合は、ノード群の管理者に通知し修理を促す。

2.4.3 今後の方針

今後は、本節で提案した設計をもとに、StarBED [301, 361] における資源管理システムを実装し運用する。これにより定義した状態と健全性の確認方法の妥当性や、健全性確認システムや実験遂行システムとの協調方法について検討する。さらにこの結果を新たな設計実装に反映することで、より良いシステムを構築する。

第 3 章　AnyBed

概要

ノード数にともなう物理資源割り当ての難しさ、設定項目数の多さにより、クラスタ環境上への実験ネットワーク構築は困難である。また、ソフトウェア・ハードウェアごとの記述形式の違いにより、別環境への設定ファイルの再利用が難しい。本章で提案する実験ネットワーク構築支援機構では実験ネットワークトポロジを論理トポロジと物理トポロジに分離した XML 記述を用いる。これら 2 つの XML 記述を基に支援機構はクラスタ環境の差異によらず論理トポロジを反映する。また本章では実装の評価を通して本提案の有用性を示す。

3.1 はじめに

近年、インターネットに関する研究では、実運用されているネットワーク以外の環境で検証・評価を行う手法が利用されている。その中の一手法として、計算機とレイヤ 2 スイッチを用いて、実運用されているネットワークのエミュレーションを行い、そのネットワーク上で実験を行うものがある。このようなネットワークエミュレーションを行う環境はネットワークエミュレーションテストベッドと呼ばれる。近年、計算機の低価格化・省スペース化により、ネットワークエミュレーションテストベッドを用いて大規模なネットワーク実験を行うことが可能となっている。

ネットワークエミュレーションテストベッドにおける一般的な実験手順は次の通りである。

第一に、実施しようとする実験を計画する。実験の目的により目標となるレイヤ 3 の実験ネットワークトポロジを決定する。また、実験ネットワークトポロジ上のどのノードにおいてプログラムを動かし、結果を収集するかを決定する。

第二に、実験計画に基づき実験ネットワークトポロジのノードに対して、ネットワークエミュレーションテストベッドに存在する物理的なノードを割り当てる。その後、実験ノードのネットワークインタフェースに対して物理的なネットワークインタフェース、

IP アドレスを割り当てる。最後に、実験ネットワークトポロジ中のサブネットに対し VLAN とサブネットアドレスを割り当てる。この際、物理配線、ネットワークインタフェース数・帯域や CPU 能力といったノードの能力を考慮し、的確な割り当てを行う必要がある。

第三に、実際のノード、レイヤ 2 スイッチ設定を行う。各ノードに対してネットワークインタフェース、経路制御、名前解決の設定を行い、レイヤ 2 スイッチに対して VLAN の設定を行う。このことにより、実験用ネットワークを構築する。また、実験に必要なプログラムを各ノードに配布し、設定を行う。そして、各ノードでプログラムを動作させ、実験を行う。この際、実験の種類によっては、各ノードでのプログラム実行の時間、順序を考慮する必要がある。プログラムの実行終了後、実験結果の収集を行う。実験結果とは、実験に用いるプログラムの出力、実行結果である。実験結果は、各ノードのハードディスクに保存されるか、syslog によりリモートホストに転送される。最後に実験の後処理を行う。各ノードやレイヤ 2 スイッチに対して行った設定を初期状態に戻す。また、各ノードのハードディスクに書き込んだプログラム、データの消去を行う。

上述した手順で実験を行う際に実験者は実験ネットワークを構築するためにクラスタの各ノードとレイヤ 2 イーサネットスイッチに対して設定を行う必要がある。しかし、ノード数が増えるにしたがって、この設定に要する作業量が増加し、実験者の負荷となる。たとえば、我々が過去に大規模ネットワークエミュレーションテストベッドの StarBED[361] において行った IP Traceback 実験 [343] では、64 台の PC クラスタを用いて実インターネットの BGP トポロジを再現するために、延べ 4 日を費やした。

また、特定の PC クラスタ環境において作成した設定ファイルはその PC クラスタ環境のハードウェア構成に依存している。このため、ハードウェア構成の異なった PC クラスタ環境において、同一トポロジで実験を行う際に過去に使用した設定ファイルの再利用が困難である。大規模なネットワークエミュレーションテストベッドにおいては、多人数で同時に PC クラスタを共有して使用する。このため、実験日により利用できる PC クラスタが変化し、ハードウェア構成が異なる PC クラスタを利用しなければならない場合がある。このような場合を考えると、過去に使用した設定ファイルを再利用出来ることが必要である。

本研究では、これらの問題点を解決するためにクラスタ環境での実験ネットワーク構築支援機構 AnyBed を提案する。

AnyBed は第一に資源割り当ての自動化により実験者の作業量と人的ミスを減らし、短時間での実験ネットワーク構築を目的とする。第二に実験に利用する PC クラスタのオペレーティングシステム、ハードウェア構成が異なる場合でも、実験ネットワークトポロジの再利用性を確保することを目的とする。

本章においては、まず 3.2 節で本研究が提案する実験ネットワーク構築機構である AnyBed の設計について述べ、3.3 節で今回実装を行った部分である論理トポロジへの資源割り当て機構の詳細について述べる。その後、3.4 節において実装した機構の評価を行い、その結果と考察を述べる。3.5 節においては関連研究について述べる。3.6 節ではまとめと今後の課題について述べる。

3.2 AnyBed の設計

AnyBed 全体の設計を図 3.1 に示す。AnyBed は 3 つの層から構成される。1 つ目は実験ネットワーク構築のための情報を集める層であるデータ収集層（designing layer）、2 つ目は資源割り当てを行う資源割り当て層（assigning layer）、3 つ目は設定ファイルを配布し、実際の実験ネットワークを構築する設定反映層（reflecting layer）である。各層を構成するコンポーネントは容易に交換可能である。データ収集層と資源割り当て層のデータ交換には XML で記述されたファイルを用いる。

AnyBed では実験ネットワークのトポロジを論理トポロジと物理トポロジに分離する。論理トポロジ

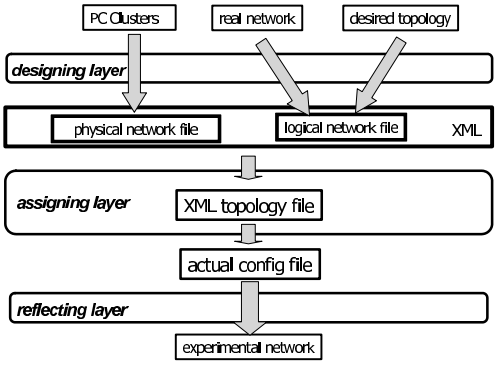

図 3.1. AnyBed 概念図

は実験ネットワークのレイヤ 3 トポロジを記述する。このため、論理トポロジは特定のクラスタ環境には依存しない。一方、物理トポロジにはクラスタ固有の情報である物理ノード、配線、ネットワークインタフェースの帯域などの情報を記述する。

実験を行う際には、物理トポロジを元に論理トポロジに対して資源割り当てを行い、その割り当ての結果生成された設定ファイルをクラスタ環境の各ノード・レイヤ 2 スイッチに配布して設定を行い、実験ネットワークが構築される。

3.2.1 実験ネットワーク構築の流れ

実験者が AnyBed を用いて実験ネットワークを構築する際の流れを図 3.1 を参照して示す。

1. 実験者は論理トポロジ（logical network file）を作成する。
2. 作成した論理トポロジとあらかじめクラスタごとに準備されている物理トポロジ（physical network file）から資源割り当て機構を用いて XML で記述された XML トポロジ（XML topology file）を生成する。
3. XML トポロジを元にノードの種類により異なる実設定ファイル（actual config file）を生成する。
4. 実設定ファイルを各ノードに配布し、設定を反映する。

このように、実験者は論理トポロジファイルを作成し AnyBed に与えるだけで、実験ネットワーク構築のための大半の作業は AnyBed が自動的に行ってくれる。

3.2.2 データ収集層

データ収集層はハードウェアに関連する情報や、実際に運用されているネットワークからトポロジ情報を収集する層である。ハードウェアに関する情報とは、具体的には PC クラスタノードの MAC アドレス、ネットワークインタフェースの情報である。トポロジ情報とはネットワークインタフェースとレイヤ 2 スイッチとの接続情報、実運用されているレイヤ 3 ネットワークの情報である。これらの情報を収集し、収集した情報を元に物理トポロジと論理トポロジを作成する。

3.2.3 物理トポロジと論理トポロジの記述

物理トポロジには機器の物理的接続状況と可能な能力を記述する。現在のところ、ノードの能力として記述できるのはそのノードが持つインタフェースの 802.1q 対応の有無である。物理トポロジの構造は次の通りである。まずノードの集合を表現する `<nodes>` 要素がある。`<nodes>` 要素はその子要素としてノードを表現する `<node>` 要素を持つ。`<node>` 要素は子要素としてノードが持つネットワークインタフェースを表現する `<interface>` 要素を持つ。`<interface>` 要素はその子要素として、物理的リンクを表現する `<link>` 要素を持つ。

物理トポロジの例を図 3.2 に示す。この例では

```
<nodes>
    <node name="mc12" os="FreeBSD">
        <interface name="bge0" bandwidth="1000"
                dot1q="yes" purpose="management"
                managementip="172.16.1.12">
        </interface>
        <interface name="bge1" bandwidth="1000"
                dot1q="yes" purpose="experiment">
           <link tonode="mc1-sw2" toint="ethernet 1/2"/>
        </interface>
        <interface name="bge2" bandwidth="1000"
                dot1q="yes" purpose="experiment">
           <link tonode="mc1-sw2" toint="ethernet 1/7"/>
        </interface>
    </node>
</nodes>
```

図 3.2.　物理トポロジの例

```
<nodes>
    <node name="NodeA">
        <interface name="NodeA-Int1">
            <network name="Net1"/>
        </interface>
        <interface name="NodeA-Int2">
            <network name="Net2"/>
        </interface>
    </node>
</nodes>
```

図 3.3. 論理トポロジの例

mc12 という名前のノードが bge0 という名前の管理用ネットワークインタフェースと bge1、bge2 という名前の 2 つの実験用ネットワークインタフェースを持ち、実験用ネットワークインタフェースはそれぞれ mc1-sw2 というスイッチの ethernet 1/2、ethernet 1/7 という名前のポートに接続されている。

論理トポロジは実験者が構築しようとする実験ネットワークの論理的トポロジを記述する。論理トポロジの構造は次の通りである。まずノードの集合を表現する <nodes> 要素がある。<nodes> 要素はその子要素としてノードを表現する <node> 要素を持つ。<node> 要素は子要素としてノードが持つネットワークインタフェースを表現する <interface> 要素を持つ。<interface> 要素はその子要素として、ネットワークインタフェースが属するネットワークを表現する <network> 要素を持つ。

論理トポロジの例を図 3.3 に示す。この例では NodeA が NodeA-Int1、NodeA-Int2 というインタフェースを持ち Net1、Net2 というネットワークに属している。

3.2.4 資源割り当て層

資源割り当て層ではデータ収集層により生成された 2 つのファイルを元に、実験者の意図する論理トポロジへ資源割り当てを行い、実際の実験ネットワークを構築する機構について説明を行う。資源とは、物理ノード、物理ネットワークインタフェース、IP アドレス、VLAN、帯域である。資源割り当て層に関する機構図を図 3.4 に示す。資源割り当て層は以下で述べる 3 つの機構からなる。

第一の機構は資源割り当て機構（dispatcher）である。本機構は、物理トポロジ、論理トポロジを入力として取り、XML トポロジを出力する。

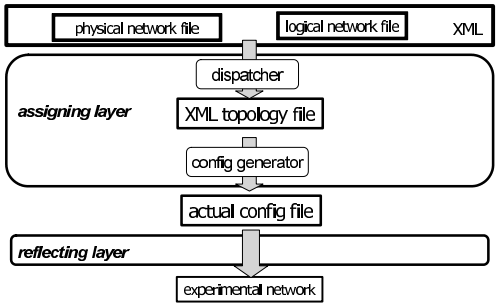

図 3.4. AnyBed 機構図

本機構の入力ファイルである物理トポロジは node – interface – link という node を根とした木構造を持ち、論理トポロジ は node – interface – network という node を根とした木構造を持つ。このため、dispatcher により link と network を対応付けることにより、適切に論理トポロジにおけるノードを物理的なノードに対応付けることができる。

第二の機構は実設定ファイル生成機構（config generator）である。本機構は、XML トポロジを入力として取り、実設定ファイルを生成する。

3.2.5 設定反映層

設定反映層では資源割り当て層の機構により生成された実設定ファイルをクラスタの各ノード、レイヤ 2 スイッチに反映させる。

設定反映層唯一の機構は実設定ファイル反映機構である。本機構は実設定ファイルを入力として取り、各ノード、レイヤ 2 スイッチに設定を反映させる。

3.3 AnyBed の実装

本節では、AnyBed のうち、核となる資源割り当て層について実装を行った。資源割り当て機構の実装を行った理由は、実験手順のうち、資源割り当てに関する作業量が最も多く、この部分を自動化することにより飛躍的に実験ネットワーク構築の作業効率が向上するためである。

AnyBed のうち資源割り当て層に存在する機構は図 3.4 の通り dispatcher と config generator である。

3.3.1 資源割り当て機構の実装

本項では、資源割り当て機構の実装について述べる。

実装のための言語には、Ruby を使用した。この理由は、XML やノードの設定ファイルといったテキストファイルの処理を多く行うため、テキストファ

```
# 実験ネットワークトポロジにおけるノードの配列
BedNodes nodes
# 物理ノードの配列
BedPhysicalNodes pnodes
# 最も残り帯域の多い物理インタフェース
BedPhysicalInterface maxleftbwpint

# 論理ノードをネットワークインタフェースの数が多い順に並べる
sort(nodes)
# 物理ノードをネットワークインタフェースの帯域の合計が多い順に並べる
sort(pnodes)
# 論理ネットワークインタフェースを帯域が多い順に並べる
sort(nodes.int)
# 物理ネットワークインタフェースを帯域が多い順に並べる
sort(pnodes.int)

nodes.each {|n|
  p = pnodes.shift
  n.int.each {
    average_bandwidth = p.int.bandwidth_total /
        n.int.bandwidth_total
    p.int.bandwidth_left = p.int.bandwidth_total

    maxleftbwpint = search_max_max_bandwidth_left(p.int)
    assign_vlan(maxleftbwpint, n.int)
    maxleftbwpint.bandwidth_left -= average_bandwidth
  }
}
```

図 3.5.　ノード・インタフェース割り当てアルゴリズム

イルの処理が容易に実装できるスクリプト言語を用いたほうが効率よく実装できると考えたためである。

実装を行った資源割り当て機構は、物理トポロジと論理トポロジを読み込み、XML トポロジを出力せずに直接実設定ファイルを出力する。

資源割り当て機構の現在のバージョンが動作するためには以下に示す条件が必要である。

- 各ノードは必ず 1 つ以上の LAN スイッチに繋がっている。
- 各レイヤ 2 スイッチは VLAN 機能を備える。
- 各レイヤ 2 スイッチは必ず 1 つ以上の 802.1q 対応のリンクで繋がっている。
- 各レイヤ 2 スイッチには共通の VLAN ID が設定できる。

この条件を置くことにより、実装を簡略化することにした。

資源割り当て機構の処理内容について次に述べる。資源割り当て機構は、論理トポロジの内容を読み込み、データを格納する。次に、物理トポロジの内容を読み込み、同様にデータを格納する。次に、実験ネットワークトポロジ構築に用いるネットワークに対して VLAN の ID とサブネットアドレスを割り当てる。このサブネットアドレスは 10.0.0.0/8 のプライベートアドレス空間から割り当てる。その後、そのサブネットに属するネットワークインタフェースに対してアドレスを割り当てる。そして、物理ノード、実験インタフェースと物理インタフェースの対応付けを行う。その後、レイヤ 2 イーサネットスイッチ間の接続に用いられているポートを検出し、必要な VLAN 設定を計算する。

最後に、対応づけられた結果を元に、実設定ファイルを出力する。実設定ファイルの内容は、FreeBSD 上の rc.conf、hosts、zebra.conf、ospfd.conf とレイヤ 2 イーサネットスイッチの設定を記したファイルである。本実装で対応したレイヤ 2 イーサネットスイッチは、DELL 社製 PowerEdge 1655MC Inte-

grated Switch、ExtremeNetworks 社製 Summit48、Summit5i である。

実験ネットワークトポロジのノードと物理ノードの対応付け、ノードのインタフェースと物理ノードの物理インタフェースの対応付けに関しては、本プロトタイプ実装では、簡単のために制約条件をインタフェースの帯域のみに絞り、実験ノードのインタフェースに対して、物理ノードのインタフェースが持つ帯域を少ない計算量で公平に割り当てるアルゴリズムとした。このアルゴリズムを図 3.5 に示す。

3.4 AnyBed の検証・評価

3.4.1 再利用性・作業量低減に対する検証

最初に、均質な PC クラスタ環境と異なったハードウェアによる不均質な PC クラスタ環境において動作するか検証を行った。つまり、1 つの実験ネットワークトポロジファイルをハードウェア構成の異なる PC クラスタ環境に再利用できるということを確認した。同時に、資源割り当て機構の実行、実設定ファイル反映機構による実設定ファイルの配布、反映にかかる時間を計測した。この時間計測により短時間で実験ネットワークトポロジが構築できるかどうかを検証した。また、構築された実験ネットワークトポロジの正当性を確認することにより、実装した機構が設計通りに動作しているかを検証した。そして、実験者が記述したファイルのサイズと資源割り当て機構により生成されたファイルの合計サイズを比較することにより、実験者の設定ファイル記述に関する作業量が軽減されていることを確認した。

検証環境

異なるクラスタ環境において AnyBed が正常に動作するかを検証するため、構成するハードウェアが均質なクラスタ環境と不均質なクラスタ環境の 2 種類を準備した。

均質なクラスタ環境のハードウェア構成を表 3.1 に示す。

均質なクラスタによる検証実験は、表 3.1 のハードウェア構成を持つ DELL 社製ブレードサーバを 17 台用い、1 台を実験用サーバとし、残りの 16 台を実験ネットワーク構築用ノードとして行った。実験用サーバ、実験ネットワーク構築用ノードは、6 台のレイヤ 2 スイッチにより相互接続されていた。

均質なクラスタでは config reflector の実装として PXE boot[136] による OS 起動・設定ファイル配布機構を元に改良した機構を利用する。この配布機構は PXE により FreeBSD の TFTP サーバ上に存在する kernel image と disk image を読み込んで各ノードを起動させ、起動時に各ノードが FTP サーバ上に存在する設定ファイルを取得するというしくみである。この機構を用いて各ノードを起動させ、各ノードに設定ファイルを配布し設定を行うことにより、実験ネットワークトポロジを構築する。

表 3.1. 均質な実験用 PC クラスタの構成

CPU	Intel Pentium3 1.4 GHz
メモリ	1024 MB
NIC	Broadcom BCM5703X 2 台
レイヤ 2 スイッチ	DELL PowerEdge1655MC Switch

表 3.2. 不均質な実験用 PC クラスタの構成

1 〜 3	CPU	Intel Pentium3 450 MHz
	メモリ	256 MB
	NIC	Intel Pro 10/100B/100+
		3Com 3c905B-TX
4 〜 7	CPU	Intel Pentium3 900 MHz
	メモリ	256 MB
	NIC	3Com 3c905B-TX
		Netgear GA620
レイヤ 2 スイッチ		FoundryNetworks EdgeIron4802F
		ExtremeNetworks Summit48
		ExtremeNetworks Summit5i

不均質クラスタ環境の各ノードのハードウェア構成を表 3.2 に示す。

不均質なクラスタ環境のノードの内、1 から 3 台目のノードと 4 から 7 台目のノードではネットワークインタフェースカードの種類が異なるため、オペレーティングシステムでのネットワークインタフェース名と利用できる帯域が異なる。

不均質なクラスタ環境においては、ノードのネットワークインタフェースの種類によっては PXE Boot が不可能な場合がある。このため、資源割り当て機構を動作させ、実設定ファイルを NFS により公開する管理サーバを準備し、各実験ノードが NFS により取得した個々の設定ファイルを個々のハードディスクに複製し、設定の反映を行う。

検証内容

これらの 2 種類のクラスタ環境で同一の論理トポロジを用いた場合、同様の実験ネットワークが構成

されるかを検証した。検証に用いる論理トポロジは 7 個のルータ間でフルメッシュリンクを構成するトポロジを用いた。2 種類のクラスタ環境で同様の実験ネットワークトポロジが構築できているかどうかを検証するために、双方の環境で各ノードの OSPF ルーティングテーブル内容を比較し、同様のネットワーク経路が存在するかどうかを調べた。比較するルーティングテーブルの内容は Zebra OSPF デーモンに telnet でログインし、show ip ospf route コマンドにより取得した。

この検証を行う際、実験ネットワーク構築を開始してから完了するまでの時間を計測する。計測は 10 回行い、その平均値を得た。更に、両クラスタ環境において、実験者が記述した論理トポロジ、物理トポロジのファイル数・サイズと資源割り当て機構により生成された実設定ファイルの合計ファイル数・サイズを比較した。

検証結果

検証結果を表 3.3 にまとめる。均質・不均質どちらの環境においても AnyBed を利用せず手作業で実験ネットワークを構築する場合は表 3.3 に示されている設定ファイル数・サイズのファイルを手作業やスクリプトを用いて作成せねばならない。

ファイルサイズでは、どちらの環境においても論理トポロジのサイズは 3377 バイトであり、物理トポロジのサイズは均質環境では 7294 バイト、不均質環境では 3172 バイトであった。つまり、論理トポロジと物理トポロジの合計サイズは、均質環境では 3377 + 7294 = 10671 バイト、不均質環境では 3377 + 3172 = 6549 バイトであった。これに対し、実設定ファイルの合計サイズは均質環境では 33957 バイト、不均質環境では 39191 バイトであった。このことから、AnyBed を用いることにより実験者が実験ネットワークを構築するために必要とされる記述量が減少していることがわかる。

表 3.3.　検証結果

	均質環境	不均質環境
AnyBed 設定ファイル数	2	2
AnyBed 設定ファイルサイズ	10671 バイト	6549 バイト
設定ファイル数	31	30
設定ファイルサイズ	33957 バイト	39191 バイト
平均実験ネットワーク構築時間	137 秒	135 秒

ファイル数については論理トポロジは 1 ファイルからなり、実設定ファイルは各ノードごとに rc.conf、zebra.conf、ospfd.conf、hosts の 4 ファイルと各スイッチごとに 1 ファイルからなる。本検証でのノード数は均質、不均質環境ともに 7 ノード、スイッチ数は均質環境では 3 台、不均質環境では 2 台であった。このため、均質環境でのファイル数は $4 \times 7 + 3 = 31$ ファイル、不均質環境でのファイル数は $4 \times 7 + 2 = 30$ ファイルである。このことから AnyBed を用いることにより実験者が記述するファイル数が減少していることがわかる。

実験ネットワークの構築時間は、どちらの環境においても 140 秒以下であった。この結果は、7 台のノードにより実験ネットワークを構築する時間としては、十分短い時間であると言える。

これらの結果から明らかに実験者の作業量は AnyBed により削減されていると言える。

OSPF ルーティングテーブルの内容はどちらの環境においても同質であったことから、同質の実験ネットワークが構築できたと言える。このことから、同じ実験ネットワークトポロジを異なる PC クラスタ環境において再利用可能であると言える。

3.4.2 実験ノード数に対するスケーラビリティ評価

次に、提案システムが大規模のネットワーク数、ネットワークインタフェース数において実用に耐えうるかどうかの検証を行った。

実装を行った資源割り当て機構が、多数のノードを有する PC クラスタ環境で動作するかどうかの検証を行った。資源割り当て機構に入力するファイルのうち、物理トポロジを固定し、論理トポロジ中に記述されているノード数を変化させ、資源割り当て機構の各部分における実行時間と資源割り当て機構全体におけるメモリ使用量を測定した。最大ノード数は StarBED での PC クラスタ環境を想定し、512 ノードとした。

評価用の実験ネットワークトポロジはできる限り各ノードがフルメッシュリンクを持つ構造とした。これは、ノード数が同じ場合、フルメッシュ構造とするのが最もネットワーク数が多くなり、ネットワーク数と関連して設定量が多くなるためである。しかし、実際の実験においては VLAN の最大数制限のため、512 台の各ノードがフルメッシュのリンクを持つことはできない。よって本実験では、IEEE802.1Q

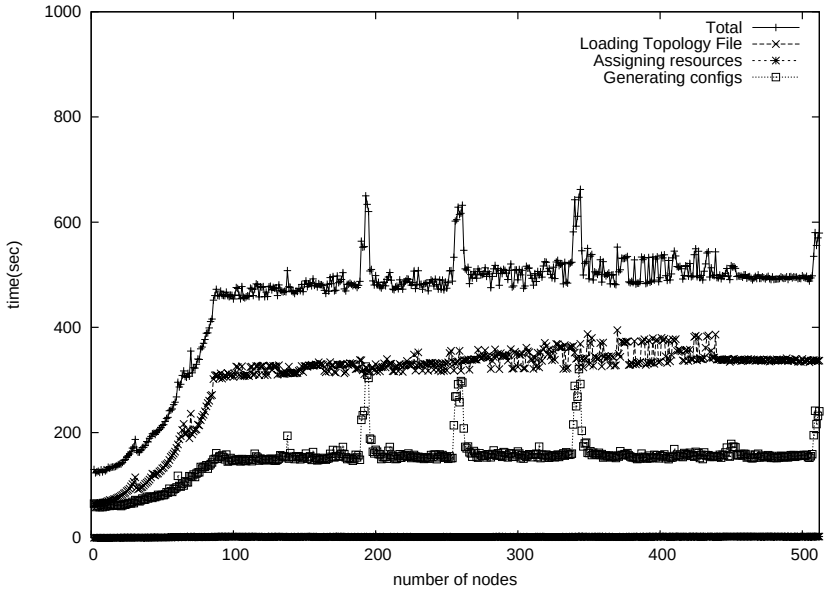

図 3.6. ノード数に対する資源割り当て機構の処理時間

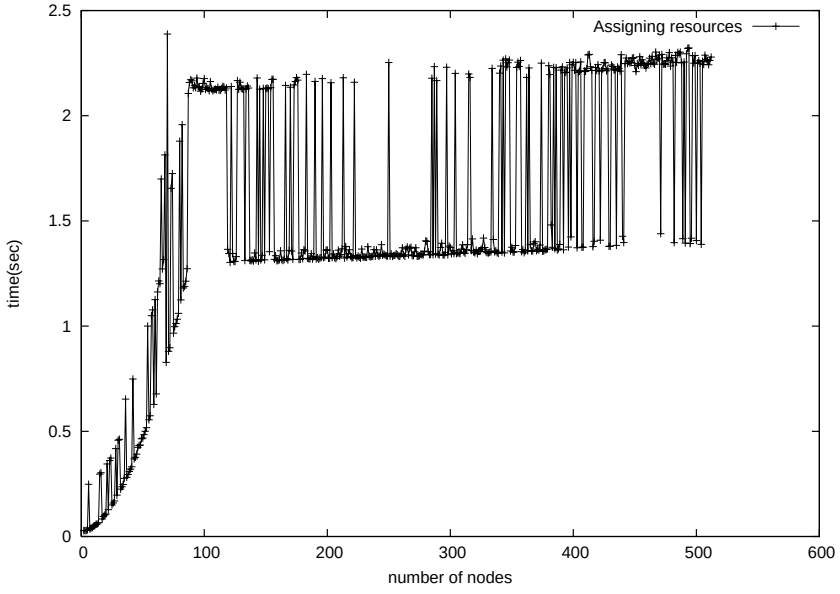

図 3.7. ノード数に対する資源割り当て機構の処理時間（アルゴリズム処理部）

での最大 VLAN 数 4096 から管理用 VLAN を除いて、実験に利用できる VLAN 数を 4000 と想定し、その VLAN 数において出来る限り複雑なネットワークトポロジを考えた結果、ノード数が 88 台まではフルメッシュ構成とし、それ以上のノード数ではフルメッシュを構成する各ノードに対してぶら下がる形でリンクを持つノードからなる構成とした。この実験ネットワークトポロジを表 3.4 の示す環境においてノード数を 2 から 512 まで変化させ、実験を行った。

実験では論理トポロジの読み込みに要する実時間、

表 3.4. スケーラビリティ評価環境

ハードウェア	
CPU	UltraSPARCIII 900 MHz × 24
メモリ	64 GB
ソフトウェア	
オペレーティングシステム	Solaris 8
言語	Ruby 1.8.1
XML パーサ	REXML 2.4.8

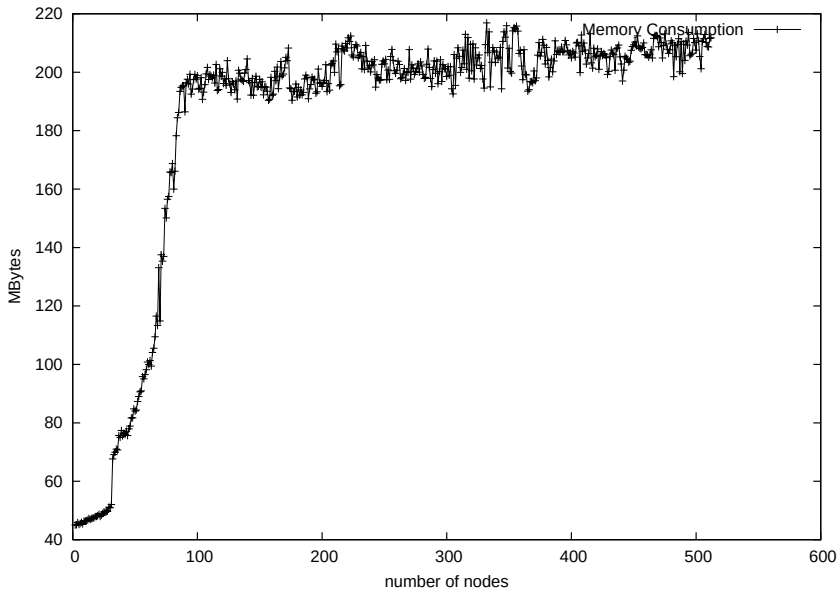

図 3.8.　ノード数に対する資源割り当て機構のメモリ使用量

物理トポロジの読み込みに要する時間、資源割り当てに要する時間、実設定ファイルの出力に要する時間を測定した。また、資源割り当て機構が使用するメモリ量を測定した。計測した結果を図 3.6、図 3.7、図 3.8 に示す。

512 ノードでの実行時間は 579 秒であり、そのうちアルゴリズム処理部は 2 秒であった。また、512 ノードでのメモリ使用量は 211 メガバイトであった。

図 3.6、図 3.7、図 3.8 のグラフでは 88 ノードまでは値が急激に増加している。これは評価用の実験ネットワークトポロジが 88 ノードまではフルメッシュ構成であるためネットワーク数が急激に増加するが、それ以上のノード数では各ノードに対してぶら下がる形のリンクになるため、88 ノード以上ではネットワーク数が急激に増加しないためであると考えられる。

以上の結果から、本提案機構は検証環境で用いたような研究室レベルの組織に存在する数 10 台程度の規模を持つクラスタにおいては、十分実用に耐えると考えられる。

3.5 関連研究

ネットワークエミュレーションテストベッドにおける資源割り当てに関する関連研究にはユタ大の NetBed[325] で行われている研究 [259] やデューク大の ModelNet[311] がある。これらの研究は特定のクラスタ環境に対して実現したい実験ネットワークトポロジを与え、適切な資源割り当てを行い、実験ネットワークを構築するという手法である。それに対して、本研究ではさまざまな構成のクラスタ環境での利用を前提とし、そのクラスタ環境上で同一の実験ネットワークを構築することを目的の 1 つとしている。このことにより関連研究とは研究の方向性が異なる。

3.6 おわりに

本章では、ネットワークエミュレーションテストベッドでの実験ネットワーク環境構築に関する問題点を指摘し、その問題点を解決する実験ネットワーク構築支援機構である AnyBed を提案した。そして AnyBed のうち、資源割り当て層を実装し評価することにより、実験ネットワークトポロジの再利用性を確保し、資源割り当てを自動化することで短時間で実験ネットワークを構築可能なことを示した。

よって、本提案によりネットワークエミュレーションテストベッドでの実験ネットワーク構築の省力化が可能となったと言える。

今後の課題として、未実装となっている各機構の実装、資源割り当てアルゴリズムの改良、アプリケーション設定ファイルへの対応、クラスタが物理的に分散している場合の実験ネットワークの構成方法などについて考えていく。

第 4 章　有線ネットワークにおける無線ネットワークのデータ到達性エミュレーション

概要

本章では、無線ネットワークで用いられるアプリケーションやプロトコルの実行環境として広く用いられている有線ネットワークで無線ネットワークをエミュレーションする方法を提案する。本手法はイーサネットフレームレベルのエミュレーションを実現しているため、アドホックネットルーティングプロトコルに対応するなど、W-NINE などの従来手法に比べエミュレーション対象が広い。

4.1 はじめに

近年、無線通信は広く普及し、携帯電話や無線 LAN のユーザ数が著しく増加している。また、モバイルコンピューティングの発達により、自律的にネットワークが構成されるアドホック無線ネットワークが注目されている。そのため、多種多様な無線通信を用いたアプリケーションやプロトコルが開発されている。これらのアプリケーションやプロトコルは最終的に目標となる実際の機材をプラットホームとして稼働することになるが、対象となる機材やソフトウェアの整備が必要である。これらの手間が開発の妨げとなっており、手間を省く簡易な実験環境が求められている。

仮想的な無線環境が実現できれば、これらの機材やソフトウェア整備の手間を軽減、あるいは時期を移すことができる。本研究では汎用的な有線インターネットを用いて、無線イーサネット上の TCP/IP 通信のエミュレーションを実現する。ネットワーク層の挙動も含めたエミュレーションを想定し、イーサネットフレームのフィルタリングによって無線イーサネットの到達性を変化させる。

先行研究には IP レベルのエミュレーション W-NINE[50] がある。これはアプリケーション層やトランスポート層の実験を目的としており、アドホックネットワークのルーティングプロトコルなどのエミュレーションの実現は困難である。

本研究は、開発したアプリケーションやプロトコルを稼働させる環境を構築することが目的である。ns-2[3] に代表されるような、通信行為をすべてシミュレーションするソフトウェアシミュレータとはアプローチが異なる。

本章では、有線ネットワークと無線ネットワークの比較、無線ネットワークのエミュレーション技法とその実装を述べる。そして、到達性変更の検証実験、アドホックルーティングプロトコルの動作例を紹介する。

4.2 無線ネットワーク

本節では、本研究のエミュレーション対象である無線ネットワークについて述べる。

4.2.1 想定する無線ネットワーク

今回想定する無線ネットワークは、現在普及している IEEE 802.11b とし、ネットワークの形態は、基地局などを必要としないアドホックネットワークを想定する。この無線アドホックネットワークではノードとノードが直接通信を行う。想定する無線ネットワークのトポロジの例を、図 4.1 に示し、以下に特徴を挙げる。

- ノード A の無線の伝送可能範囲にノード B、D が含まれ、ノード A からのデータはノード B、D に届く。ノード C は伝送範囲外でデータは届かない。表 4.1 は、図 4.1 の無線環境における各ノード間のデータの到達性を表す。送信ノー

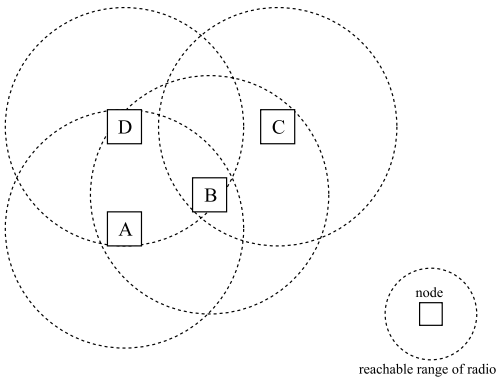

図 4.1.　無線ネットワークのトポロジ例

表 4.1.　ノード間のデータの到達性

	A	B	C	D
A	−	○	×	○
B	○	−	○	○
C	×	○	−	×
D	○	○	×	−

ドからデータが到達する組み合わせは○、データが到達しない組み合わせは×で表す。

- ノード A の無線の伝送範囲内に存在するノード B、D にのみノード A からの ARP などのデータを到着させる必要がある。これは、データの到達性がネットワーク層に関係するのではなく、リンク層に関係するためである。
- ノードは移動する。そのため、表 4.1 では、ノード A–C 間でデータは直接到達できないが、ノードが移動することで通信できるようになる可能性がある。そのため、全ノードで他のノードへの到達性を再計算し、伝送可能範囲に達すると通信が可能になるシステムの実装が必要である。

4.2.2 有線ネットワークとの相違点

無線ネットワークのエミュレーション環境を構築するため、有線ネットワークとの相違点を考えねばならない。以下に相違点を示す。

データの到達性の偏在 電磁気学的特性（位置、距離、障害物、大気状態）によりデータの到達性が変化する。

帯域 無線通信に使用できる周波数は限られているため、実行帯域は比較的狭い。

コリジョン コリジョン検知・再送の方法が異なる。

これらのうち、無線ネットワークのエミュレーション環境として最も重要なのはデータ到達性の変化であろう。以下、データ到達性の変化を中心に説明する。

4.3 設計

システムは到達性算出と到達性制御で構成される（図 4.2）。到達性算出部は各ノードや全体の情報を元に、各時刻における各ノード間の到達性を算出する。前述のように、無線ネットワークにおける到達性変化のエミュレーションを実現する実験環境構築が本研究の目的であり、到達性の厳密な算出は本研究の次の段階の目的である。そして、ネットワーク層の視点ではイーサネットフレームが到達できるかどうかの 1 ビットの情報量のみを必要とする。したがって、今回は単純な到達性計算で目的を達成できる。そこで、位置からユークリッド距離を算出して、到達性を導出した。

到達性制御部は算出された到達性にしたがってイーサネットフレームを破棄する。

各ノードを同じネットワークに接続すると、あるノード X の送出するイーサネットフレームが別のノード Y に直接届く（制御なし）可能性がある。そこで、ノード X と Y が同じネットワークに所属しない形態にする必要がある。そして、あるノードから見てイーサネットフレームが実際に到達しない状況を作り出すためには、ノードの外にフィルタリング機構を設置せねばならない。したがって、図 4.3 のような無線ネットワークは、図 4.4 のような装置構成でエミュレーションを行う。図中の Link Control Node（LCN）がイーサネットフレームのフィルタリングによりリンクを制御するプログラムである。各ノードに対して LCN が 1 つずつ割り当てられる。

例えば、表 4.1 の場合、ノード A は、ノード B、D にはイーサネットフレームを送信可能だが、ノード C には送信不能である。ノード A に接続された LCN E

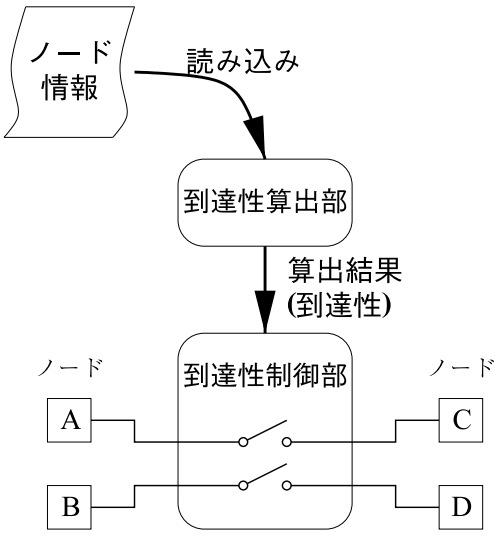

図 4.2.　システム構成

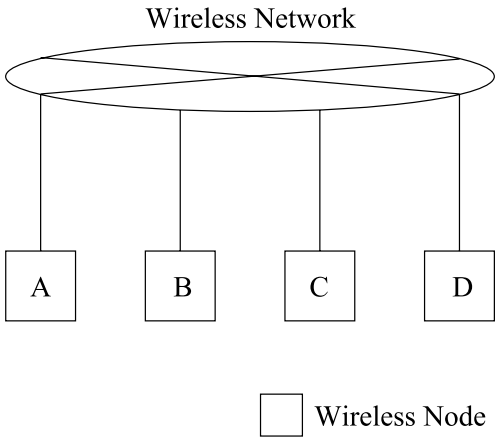

図 4.3.　想定する無線ネットワークトポロジ

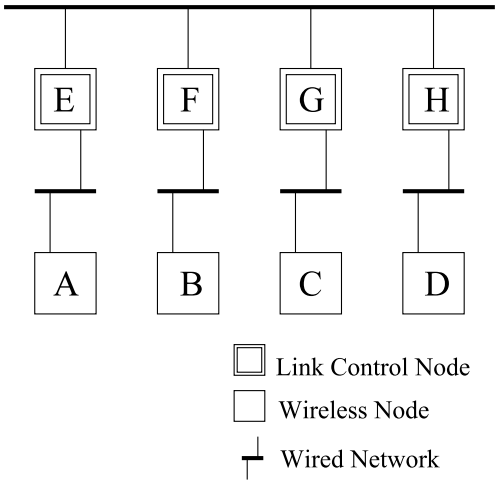

図 4.4. エミュレーション装置のトポロジ

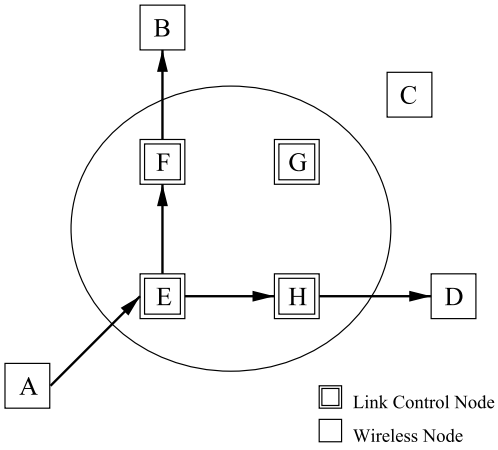

図 4.5. フレームの流れ

が、表 4.1 に基づいたノード A と他ノードとの到達性の情報を持ち、その情報に基づいてイーサネットフレームの転送を行う。この場合では、LCN E が LCN F、H を介してノード A と B へイーサネットフレームを送信する（図 4.5）。この処理を LCN で行うことにより、無線ネットワークのエミュレーションが実現される。

また、前節で述べたように、本実験環境の仕様として、送信ノードと受信ノード間をイーサネットフレームによって通信するシステムとする必要がある。イーサネットフレームをすべてのノードに届ける方法として、今回は UDP トンネルを用いた。イーサネットフレームが UDP でカプセル化される様子を図 4.6 に示す。ノード A からノード B にデータを送信する場合、LCN E でイーサネットフレームを UDP でカプセル化し、LCN F へ送信する。LCN F

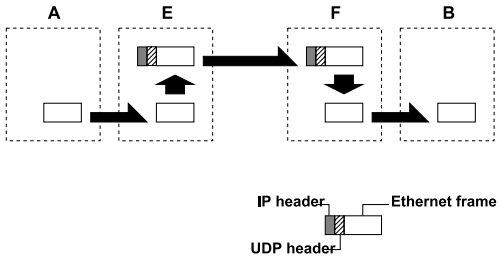

図 4.6. カプセル化

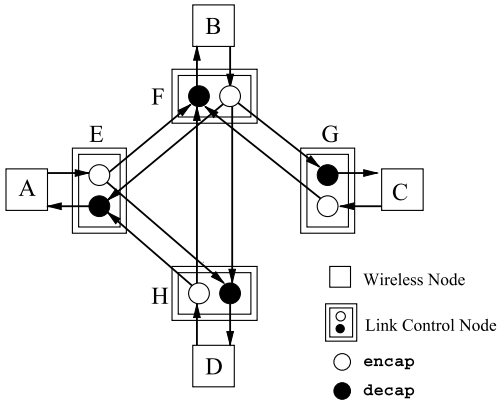

図 4.7. ソフトウェア配置

で UDP ペイロードからイーサネットフレームを取り出し、ノード B にイーサネットフレームで送信する。これにより、イーサネットフレームによってすべてのノードと通信できる。また、UDP トンネルは、特定のノードに向けて、イーサネットフレームを含めたデータグラムをユニキャストで送信する。そのため、特定のノードにのみイーサネットフレームの転送が可能となり、イーサネットフレームが到達できるノードを変化させることができる。上記の LCN E と F での処理を行うプログラムをそれぞれ encap、decap と呼ぶ。エミュレーション全体では図 4.7 のような配置となる。encap でイーサネットフレームを UDP ペイロードへのカプセル化を行い、decap で UDP ペイロードからイーサネットフレームを取り出す。これにより、ノード間でイーサネットフレームを送受信でき、またそのイーサネットフレームが受信できるノードを変化させる環境を構築できた。

次に、エミュレーションの手順について示す。

1. 各時刻による各無線ノードの位置を決定
2. 位置から各無線ノード間の距離を時刻ごとに算出
3. 算出した距離からデータの到達性を算出
4. 各 LCN へ通信ノードの到達情報を送信
5. 各 LCN の時刻を 0 にする

6. 各 LCN で decap を起動
7. 各 LCN の時刻を 1 つ進める
8. その時刻の到達情報をもとに encap を起動
9. 7 へ戻る

これにより、ノードの到達性は、時刻ごとに変化させることができる。

4.4 実装

到達性算出部は perl で作成した。

到達性制御部の LCN の要素となる encap と decap は UNIX 上で C 言語を用いて実装した。イーサネットフレームの受信には libpcap[181] を用いた。libpcap の拡張を行うことにより、libpcap でイーサネットフレームの送信を行えるようにした。なお、この 2 つのプログラムは FreeBSD と Linux で動作確認済である。実ノードのエミュレーションの進行は、kuroyuri[301] を用いて行う。kuroyuri は、StarBED Project[301] により開発された実験支援システムのひとつである。

4.5 評価

4.5.1 データの到達性の変化

無線ノードの移動にともなった到達性の変化を確認する実験を行った。無線ノードを移動させ、その到達性を計測する（図 4.8）。ノードは、矢印の向きに移動し、ノード間の距離を変化させることでデータの到達性も変化する。実験諸元を表 4.2 に示す。

ノード A からノード B、C、D に ICMP を送信し、それぞれのノードで通信を観測することで、到達性の変化は確認した。そのため、前述のエミュレーション手順に ICMP を送出する以下のステップを追加して行った。

9. 各通信ノードから全通信ノードへ向けて ping を実行（ICMP 送出）: 10 ms 間隔
10. ping の結果を記録
11. 7 へ戻る

図 4.9 は、送信ノード A から受信ノード B へ ICMP を送信した結果である。同様にノード A からノード C と D に送信した結果を、それぞれ図 4.10 と図 4.11 に示す。実線は想定値を示し、破線は測定値を示す。3 つの結果において、想定値と測定値がほぼ同じ値を示しているので、ノード間の距離による無線イーサネットの到達性変化のエミュレーションが成功したとみなせる。しかし、最初にノード B、C、D がノード A の伝送範囲内に入った時の想定値と実測値の時刻の間にずれがあった。

これは、最初の一部の ICMP が届かなかったことを示す。このずれは、送信元ノードが ARP による

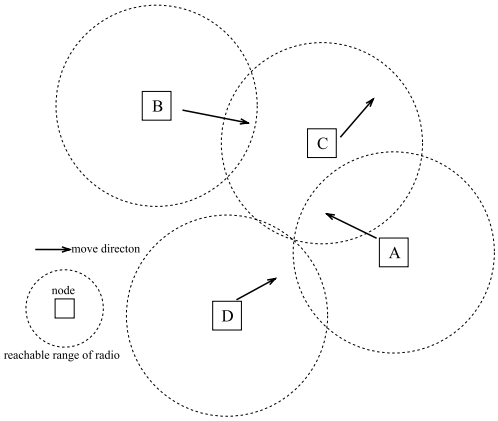

図 4.8.　テストトポロジ

表 4.2.　実験諸元

ノード数	4 [台]
データの最大到達距離	150 [m]
移動範囲	500 × 500 [m]
ノードのスピード	0〜5 [m/s]
移動モデル	random walk
実験時間	100 [s]

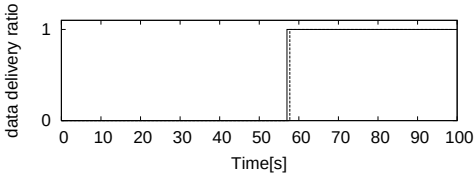

図 4.9.　A–B 間の ICMP 応答

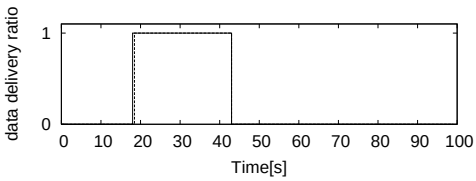

図 4.10.　A–C 間の ICMP 応答

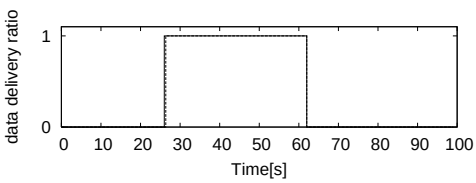

図 4.11.　A–D 間の ICMP 応答

アドレス解決に要した時間である。想定している無線ネットワークでも、ARP による受信先の MAC アドレス解決が行われるので、このずれは正常であり、ずれを除去する工夫は不要である。

4.5.2 アドホックルーティングプロトコルを用いた実験

この実験では、今回提案した実験環境で、実際に使用されているプロトコルやアプリケーションの動作の検証を行う。アドホックネットワークのルーティングプロトコルは、Ad-hoc On Demand Distance Vector（AODV）[246] を用い、図 4.12 のような、トポロジで実験を行う。表 4.3 に実験諸元を表す。

本実験では、ノード A に FTP（File Transfer Protocol）サーバを起動し、ノード B からノード A のファイルを FTP によって取得する。ノード A は移動するため、最初はノード D を介してファイルを取得する。途中でノード D と接続が切れるため、経路が変わりノード C を介してファイルの取得を行う。すべてのノードで、tcpdump を動作させ、結果を取得する。この結果よりファイルの取得経路の変化を確認した。これにより、今回提案する実験環境で、ネットワーク層以上のプロトコルやアプリケーションの動作について、確認できた。

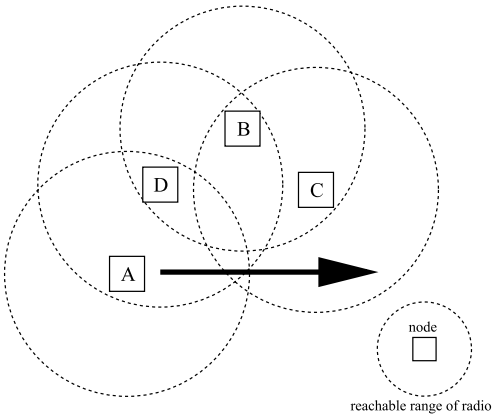

図 4.12. アドホックネットワークの実験トポロジ

表 4.3. アドホックネットワークの実験諸元

ルーティングプロトコル	AODV
ノード数	4 [台]
データの最大到達距離	150 [m]
ノードのスピード	4 [m/s]
実験時間	75 [s]

4.6 議論

本章では、電波の到達性の変化の要因を距離のみに限定したが、それ以外の要因として、フェージング、障害物、気象条件、移動にともなうドップラ効果が知られている。フェージングへの対応は今後の課題である。無線イーサネットは主に屋内の近距離用途に限られるため、気象条件やドップラ効果は考慮する必要はない。現実の障害物の種類や数はさまざまであり、一般的な障害物モデルを設けることは困難である。多くの研究では障害物がない状況を想定しており、本研究でも障害物の影響は除外した。一般的な障害物モデルが確立されれば適用できるであろう。

また、有線ネットワークと無線ネットワークでは帯域やコリジョン検出に違いがある。これらも今後の課題である。

4.7 まとめ

無線ネットワーク向けに開発された実際のアプリケーションやプロトコルを容易に稼働させる実験環境が求められている。本研究では、その 1 つの解として有線イーサネットによる無線イーサネットのエミュレーション手法を提案した。イーサネットフレームのフィルタリングにより、データ到達性が変化するという無線ネットワークの特性を模倣する。本手法を組み込んだエミュレーション環境で、ICMP の到達性の変化や、アドホックルーティングプロトコルの動作など、本手法の効果を確認した。本手法は無線ネットワークの簡易な実験環境として有用である。

フェージングやコリジョンなどの無線ネットワークの特性を忠実に再現することが今後の課題である。