JAIST Repository
https://dspace.jaist.ac.jp/

Title	大規模ネットワーク実証実験設備における実験用資源の最適利用に関する研究
Author(s)	吉岡, 慎一郎
Citation	
Issue Date	2011-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/9744
Rights	
Description	Supervisor:知念賢一, 情報科学研究科, 修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

大規模ネットワーク実証実験設備における実験用資源の最適利用に関する研究

北陸先端科学技術大学院大学
情報科学研究科

吉岡 慎一郎

2011 年 3 月

修 士 論 文

大規模ネットワーク実証実験設備における実験用資源の最適利用に関する研究

指導教員　知念賢一 特任准教授

審査委員主査　知念賢一 特任准教授
審査委員　篠田陽一 教授
審査委員　丹康雄　教授

北陸先端科学技術大学院大学
情報科学研究科

0910071 吉岡 慎一郎

提出年月: 2011 年 2 月

Copyright © 2011 by Yoshioka Shinichiro

概 要

インターネットは、多くサービスが存在し人々の生活には欠かせないものとなっている。それらのサービスは、ネットワーク利用するソフトウェアによって構成されている。そのようなソフトウェアを新たに導入する場合、動作に異常がないか確認をする必要がある。なぜなら、インターネットによってサービスの利用者へ、重大な影響を及ぼす可能性があるためである。そこで、ソフトウェアの開発者は、繰り返し検証を行い品質を高めることによって、そのような自体を避ける。検証は、想定するネットワーク環境で行うことが望ましい。しかし、インターネットのような大規模なネットワーク環境を用意することは困難である。なぜなら、人的コストや費用の面で現実的ではないためである。そこで、ソフトウェアの検証では、大規模なネットワークを構築し検証が可能な設備を利用する。本稿では、このような設備のことネットワーク実験設備と呼ぶ。ネットワーク実験設備では、多数のノードとネットワークが用意してある。本稿では、これらのことを実験資源と呼ぶ。ソフトウェアの検証を行う実験者は、実験資源の一部を借りて想定するネットワーク環境を構築する。ネットワーク実験設備では、実験資源を管理・制御するために支援ソフトウェアを利用する。そして、支援ソフトウェアによって、実験資源を管理・制御する。ネットワークを利用するソフトウェアの検証では、想定するネットワークの環境を構築することが望ましい。しかし、ネットワーク実験設備には、実験資源の利用用途や数に限りがある。実験者は、限られた資源を利用し、より想定と近いネットワークを構築する必要がある。想定と近いネットワークを構築することで、より正確な検証が可能となるためである。

実験資源の用途や数が制限される理由に、ネットワーク実験設備の構成に起因する問題がある。支援ソフトウェアによる管理システムは、多数存在する実験資源の管理を容易にするため静的に設置してある。そのため、管理システムに関連する実験資源の情報は、実験者が勝手に変えることはできない。さらに、異なる実験者が、同じ管理システム上で実験の管理や制御を行っている。そのため 管理システムを共有することになり、他の実験者へ影響を及ぼすような恐れがある。

本研究では、ネットワーク実験設備における管理システムやその構成を見直すことで、より最適に実験資源を利用する。そして、実験者が、より円滑にネットワークを利用するソフトウェアの検証行える環境を実現する。そこで、本研究では入れ子型の実験環境を提案する。管理システムを入れ子型の構成に配置することで、静的な管理システムに起因する制限を取り払う事が可能となる。

本研究では、提案するネットワーク実験設備を実現する管理システムを考案した。提案する管理システムは、既存の支援ソフトウェアを利用・拡張することで実現する。入れ子型の管理システムは、既存の管理システムを、実験者ごとに提供することで可能とした。また、実験資源を追加する機能として、仮想ノードへ対応した。仮想ノードは、実験者が仮想化ソフトウェアにより生成するノードである。仮想ノードを実験資源化することで、

実現可能なネットワーク実験の幅が広くなる。そして、動作検証実験を行うことで、提案するネットワーク実験設備は有用であることを示す。また、それらの実験結果をもとに、提案システムについて考察した。

目 次

第 1 章　はじめに **1**
1.1 背景 1
1.2 本研究の目的 1
1.3 構成 2

第 2 章　ネットワーク実験設備 **3**
2.1 StarBED 3
2.2 SpringOS 4
2.2.1 ERM(Experiment Resource Manager) 4
2.2.2 DMAN(Directory Manipulator) 4
2.2.3 Kuroyuri 5
2.2.4 NI(Node Initiattor) 5
2.2.5 SWMG(SWitch ManaGer) 6
2.2.6 PWMG(PoWer ManaGer) 6

第 3 章　柔軟な実験環境作成の提案 **7**
3.1 既存研究における制約 7
3.1.1 StarBED における制約の原因 7
3.2 制約の少ない実験環境の提案 8

第 4 章　設計 **10**
4.1 全体像・構成 10
4.2 NEE の生成・制御 10
4.2.1 NEE Installer：NEE Manager の導入 13
4.2.2 NEE Manager：NEE の生成 13
4.3 ソフトウェアの設計 13
4.3.1 NEE Installer の設計 15
4.3.2 NEE Manager の設計 17
4.3.3 NEE スクリプトインタプリタ 17
4.3.4 DCM(Dhcpd Configration Manager) 18
4.3.5 仮想マシンを用いた実験ノードの生成 18
4.3.6 SWMG Proxy 20

4.3.7 PWMG Proxy 20

第 5 章 実装 **23**
5.1 実装環境 23
5.2 SpringOS との協調 23
5.2.1 ERRP クライアントモジュールの実装 24
5.2.2 DMAN クライアントモジュールの実装 24
5.2.3 SWMG クライアントモジュールの実装 25
5.2.4 PWMG クライアントモジュールの実装 25
5.3 NEE Installer の実装 26
5.3.1 管理ノード群の導入 26
5.4 NEE Manager 実装 26
5.4.1 DCM 26
5.4.2 SWMG Proxy 27
5.4.3 PWMG Proxy 27
5.4.4 ノード情報の登録・更新 27
5.4.5 DHCP サーバの設定変更 29
5.4.6 仮想ノードの追加 29
5.4.7 NEE スクリプトインタプリタ 32

第 6 章 評価 **34**
6.1 動作検証 34
6.1.1 実験ノード追加 34
6.1.2 NEE の隔離 39
6.2 既存研究との比較 41

第 7 章 おわりに **43**
7.1 今後の課題 43
7.1.1 高度な入れ子型の実験環境の生成 43
7.1.2 仮想ノードへの高度な対応 43
7.2 将来の展望 44
7.2.1 実験環境の保存と復元 44
7.2.2 実験環境の移送 44

第1章　はじめに

1.1　背景

インターネット上には、多くのサービスから存在する。それらサービスは、ネットワークを利用するソフトウェアにより実現されている。そして、新しいサービスを実現するために、多くのソフトウェアが開発されている。ソフトウェアの開発は、正常に動作しているかを確認するために検証を行う必要がある。ネットワークを利用するソフトウェアの場合、想定するネットワークで検証することが重要である。特に、インターネットのような大規模なネットワーク上で運用するサービス場合、そのような検証は重要である。しかし、インターネットのような大規模なネットワークを検証のために準備することは、困難である。なぜなら、人的コスト・費用の面で現実的ではないためである。そこで、大規模なネットワークが構築可能な、ネットワーク実験設備を利用する。ネットワーク実験設備は、実験ノードやネットワーク機器が設置されている。さらに、それらを有効に利用するための支援ソフトウェアがある。このような設備を利用することにより、ネットワークを利用するソフトウェアの検証を行うことができる。

ネットワーク実験設備では、ソフトウェアの検証を行う実験者へ実験資源の一部を貸し出す。実験資源とは、実験ノードや論理的なネットワークのことである。論理的なネットワークは、スイッチの制御により動的にネットワークを作成することができる。実験者は、これらの実験資源により検証を行う。複数の実験者がいる場合、お互いに影響を及ばさないための機構が必要である。そのため、ネットワーク実験設備では、支援ソフトウェアによって実験資源を管理・制御する。

以上のように運用されている実験設備だが、いくつかの制限が存在する。制限とは、実験ノードやネットワークの利用用途や数である。これらの制限がなくなれば、実験者はより効率よくソフトウェアの検証を行うことが可能となる。

1.2　本研究の目的

ネットワーク実験設備は、実験者が効率よく検証を行うことのできる環境を用意する必要がある。本研究では、実験資源の制限の少ないネットワーク実験設備を提案する。そこで、従来のネットワーク実験設備の管理システムを見直す。従来の設備は、運用の容易さから、管理システムを静的に設置してある。管理システムが静的に設置されているため、実験者は実験資源の利用用途や数を変更することができない。そこで、管理システムを実

験者がごとに分割することで、それらの制限をなくすシステムを提案する。実験者は、自分の環境の管理システムをカスタマイズすることで、実験資源の用途や数を変更することができる従って、実験者は、より幅広い実験を行うことができ、ソフトウェアの検証を円滑に進めることができる

1.3 構成

２章では、既存の実験環境について述べる。３章では、既存研究の問題点と、それを解決する実験環境の提案を行う。４章では、既存のネットワーク実験設備の制限を解決するシステムの設計について述べる。５章では、提案したシステムの実装について述べる。６章では、提案システムを利用して実験を行い、その結果から提案システムについて考察する。７章では、まとめと、今後の課題と将来への展望について述べる。

第2章　ネットワーク実験設備

本章では、本研究で想定するネットワーク実験設備である StarBED[1, 2] について節 2.1 で述べる。また、StarBED で用いられている支援ソフトウェアである SpringOS[3] について節 2.2 で述べる。

2.1　StarBED

StarBED は、情報通信研究機構北陸リサーチセンター [4] によって提供されているネットワーク実験設備である。約 1000 台の PC とスイッチ、支援ソフトウェアから構成されている StarBED では、実験資源であるノードやネットワークが、設備の中に収められている。ノードは、いくつかグループに分けて設置してある。グループは、ネットワークのメディアやノードのスペックによって分けられている。ネットワークは、VLAN[5] により論理的なネットワークを構築する。実験者、これらの実験資源の一部を利用することができる。そして、それらを用いてネットワーク実験を行う。多くのネットワーク実験設備では、利用できる OS やネットワークが制限されている。しかし、StarBED では、それれらの制限が少ない。ノードには、任意の OS を導入することができる。また、ネットワークは、スイッチを制御することで、幅広いネットワーク環境を構築できる。

StarBED では、多数ノードの制御や実験実行を補助するために、支援ソフトウェア群である SpringOS を利用する。SpringOS は、大規模なネットワーク実験環境向けの支援ソフトウェア群である。SpringOS は、設備全体の実験資源の管理からネットワーク実験の実施に有用なソフトウェアを提供している。実験者は、実験をシナリオをとして記述し実行する。そのため、ネットワーク実験環境の導入から、実験の実施までを一貫して行うことが可能である。

StarBED の構成を図 2.1 へ示す。ノードやネットワークは、管理ノード群により管理されている。ノードは、管理ネットワークと実験ネットワークに接続されている。管理ネットワークは、ノードや実験の管理のためのネットワークである。また、管理ネットワークは設備の運用に必要であり、実験者が利用用途を変更することができない。実験ネットワークは、実験者が実験を実施するためのネットワークである。利用用途は、制限されていない。StarBED は、以上の設計により実験資源の管理や実験を行う。

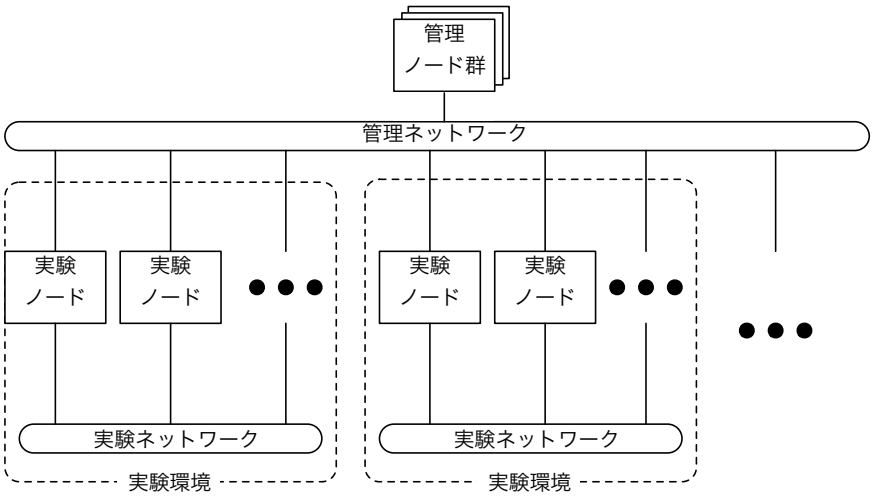

図 2.1: StarBED の構成

2.2 SpringOS

本節では、支援ソフトウェア群である SpringOS について述べる

2.2.1 ERM(Experiment Resource Manager)

ERM は、実験資源の管理や割り当てを行う。そのため、ノードや VLAN ID の実験資源情報を保持している。実験資源情報とは、利用状況やディスク種類、ネットワークインタフェースの数や用途等である。SpringOS のソフトウェアは、ERM へを問い合わせることで実験資源情報を取得する。また、ERM は、実験者への実験資源の割り当て情報も管理している。実験者は、提供されている以外の実験資源は、利用できないようにする。SpringOS の多くのソフトウェアは、ERM へ問い合わせることで実験資源の情報を取得する。

2.2.2 DMAN(Directory Manipulator)

SpringOS の想定するネットワーク実験環境では、ネットワークブートによりブートローダを取得し起動する。起動するディスクパーティションごとにブートローダが準備されている。ネットワークブートは、DHCP サーバ [6] によって IP アドレスとブートファイル名、TFTP サーバ [7] のアドレスを取得する。次に、TFTP サーバからブートファイルを取得しブートローダを起動する。DHCP サーバにより設定されているブートファイルは、シンボリックリンクになっている。DMAN は、そのシンボリックリンクを変更することで、ブートローダを切り替える。

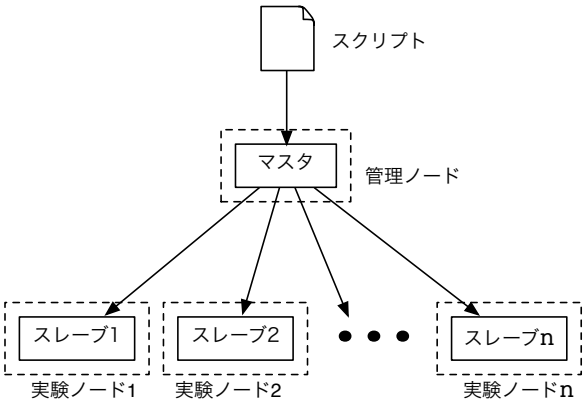

図 2.2: マスタ・スレーブモデル

2.2.3 Kuroyuri

Kuroyuri[8] は、ネットワーク実験記述言語処理系である。K 言語と呼ばれる実験記述言語により実験シナリオを記述する。実験シナリオは、以下を含む

- 実験ノードの作成
- 実験ネットワークの作成
- 実験の実施

K 言語は、実験の準備から実施までを一貫して記述することができる。

Kuroyuri は、マスタ・スレーブモデルを採用している。Kuroyuri によるマスタ・スレーブモデルの概要を図 2.2 へ示す。マスタとは、実験シナリオ実行するノードである。スレーブは、マスタによって命令された実験シナリオを実行する。マスタから、複数のスレーブに実験シナリオを投入することで多数のノードを利用した実験を行う。

2.2.4 NI(Node Initiattor)

ネットワーク実験環境では、多数のノードを扱う。そのため、各ノードへの手動で OS 導入した場合、多くの時間や労力を必要とする。そこで、OS 導入を支援するソフトウェアである NI[9] を利用する。

NI の機能は２つである。

1) ノードのディスクイメージの保存

2) ノードへのディスクイメージの書き込み

NI は、ディスクレスシステムの OS を、ネットワークブートにより起動する。そのため、DMAN よって、NI が起動するように切り替える。そのため、1) や 2) のように、ディスクを操作することが可能である。1) により、OS の入ったディスクごとディスクイメージとして保存する。そして、2) により、多数のノードへディスクイメージを書き込み、OS 導入する。

2.2.5 SWMG(SWitch ManaGer)

SpringOS の想定するネットワーク実験環境では、VLAN により論理的なネットワークを構築し実験を行う。VLAN の操作は、スイッチを制御することで可能である。しかし、ネットワーク実験環境では、異なったメーカのスイッチが混在して利用されている。スイッチは、メーカによって操作方法が異なる。実験者には、実験に要する手間を省くため、それらを意識しないスイッチの制御方法を提供する必要がある。SWMG 制御を抽象化し、同じインターフェースで複数のスイッチの制御を実現する。そのため、実験者は、スイッチの制御の違いに時間を費やすことなく、実験を遂行することができる。

SWMG は、ERM へ問い合わせることで各ノードが接続されているスイッチやポートの情報を取得する。さらに、ERM と協調することで、実験者がスイッチ制御可能な範囲を制限する。ネットワーク実験設備では、運用に利用しているネットワークや、他者の実験ネットワークが存在する。そして、スイッチの制御した場合、これらのネットワークを破壊してしまう恐れがあるためである。

2.2.6 PWMG(PoWer ManaGer)

ネットワーク実験環境では、多数のノードが存在し起動・停止を手動で行うことは困難である。そのため、それらの手間を省くために、遠隔から一括で起動・停止を行う機構が必要である。PWMG は、IPMI[10] を利用することで遠隔から起動・停止を行う。また、多数ノードを制御可能である。IPMI は、特定のハードウェアや OS に依存することなく、マシンを管理するためのインタフェースである。IPMI はによって管理されたマシンは、ネットワーク上から電源の起動・停止が可能である。

PWMG は、ERM に問い合わせることで各ノードの IPMI の情報を知る。また、SWMG と同様、ERM と協調することで、他の実験者への影響を防ぐ。

第3章　柔軟な実験環境作成の提案

3.1　既存研究における制約

実験者は、実験設備から一部の資源を借りて実験環境を作成する。しかし、既存研究による実験環境では、作成困難な構成がある。そのため、実験者から見た場合、実験環境による制約が存在していることになる。これらは、実験環境の運用ポリシーや管理システムに起因することが多い。本研究では、そのような実験環境の制約の中の一部に注目する。実験資源に関する項目として以下がある。

- 実験ノード情報の変更
- 実験ノードの追加
- ネットワーク用途の変更

実験ノードは、実験設備によって一括に管理されており、それらを追加したり情報を変更することは困難である。もし、実験者が実験ノードを追加・変更することが可能となった場合、情報の一貫性が取れなくなる。

既存研究による実験環境では、ネットワークはあらかじめ用途を決定されていることが多い。なぜなら、運用や管理を行うためのネットワークが、最低限必要となるためである。このような、重要なネットワークは、動的に設定するより静的に設置しておいたほうが運用面において安全かつ容易である。

3.1.1　StarBED における制約の原因

既存研究である StarBED に注目し、実験環境の制約の原因を議論する。節 2.1 で、StarBED の構成を図 2.1 に示した。このような構成の StarBED だが、 節 3.1 で述べた制約がある。その原因として以下が挙げられる。さらに、原因となる箇所を強調し、図 3.1 へ示す。

- 管理ノード群の共用
- 管理ネットワークの共用

管理ノード群は、複数の実験環境によって共用されている。また、管理ノードを利用するためのネットワークである管理ネットワークも共用されている。管理ノード群が特定に実

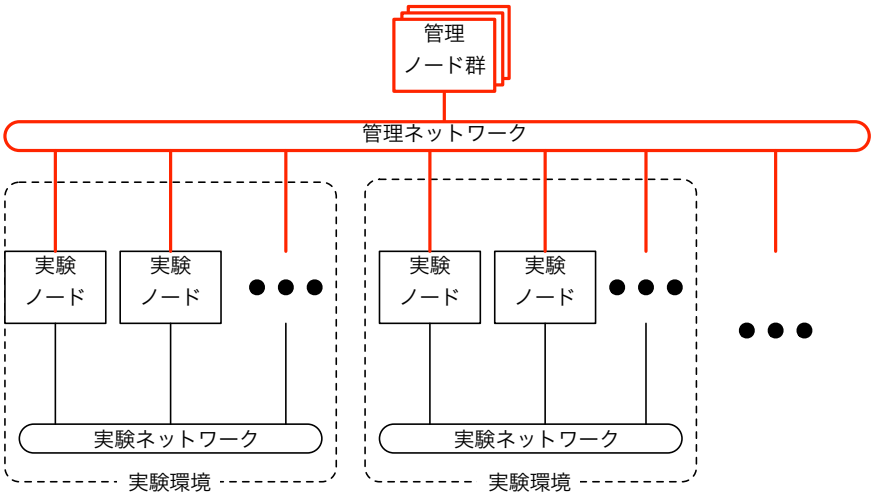

図 3.1: StarBED の構成：制約の原因となる構成箇所

験者によって情報や設定を変更された場合、実験設備全体の情報の一貫性が無くなってしまう。そのため、管理ノード群における、実験者の利用権限は読み出しのみである。また、これを利用するための管理ネットワークも、実験者が用途を変更してしまうと、実験ノードの管理ができなくなる。そのため、実験者が用途を変更して利用することはできない。

3.2 制約の少ない実験環境の提案

節 3.1 で述べた制約を取り除く実験環境を提案する。そこで、StarBED の構成を基に、実験資源の管理権限や構成について検討する。従来の実験環境の管理権限と提案する実験環境の管理権限の概要を図 3.2 へ示す。節 3.1 を述べたように、従来の実験資源の管理は、実験設備よって行われていた提案する実験環境では、実験者が割り当てられた実験資源の管理を行う。

提案する実験環境の構成について、以下の構成を提案する。

1) 実験環境ごとに追加・設定が可能な管理ノード群

2) 実験環境ごとに、動的に生成可能な管理ネットワーク

1) について、実験環境ごとに、追加・設定可能な管理ノード群が存在すれば、実験者はそれらを利用して実験資源の追加や変更が可能となる。2) について、管理ネットワークも実験者が管理を行うために、実験環境内に生成する。管理ネットワークの場合は、実験資源が追加・変更されるとき、それに合わせて再構築する。そのため、動的に生成可能である必要がある。このような構成によって、割り当てられた実験資源の管理権限を実験者へ移譲する。

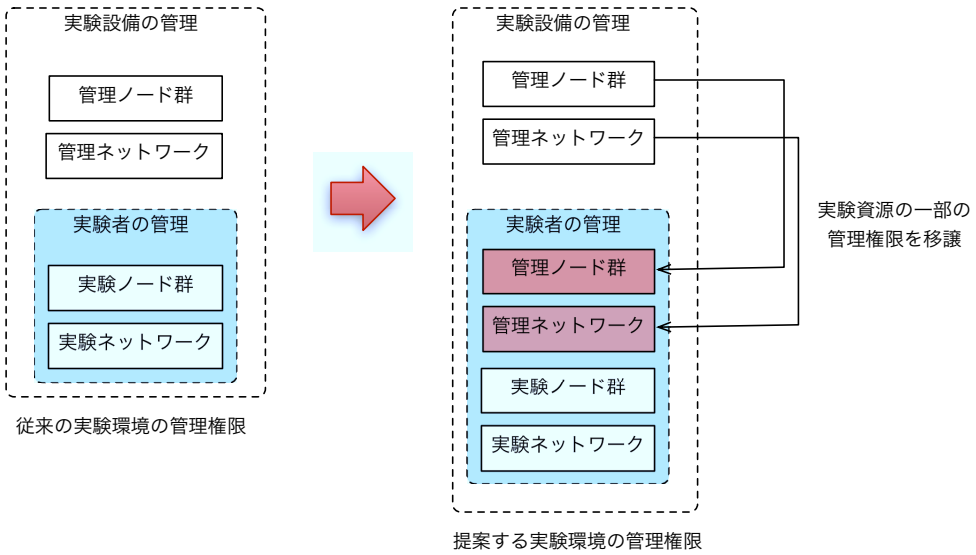
実験設備の管理
管理ノード群
管理ネットワーク
実験者の管理
実験ノード群
実験ネットワーク
従来の実験環境の管理権限
実験設備の管理
管理ノード群
管理ネットワーク
実験者の管理
管理ノード群
管理ネットワーク
実験ノード群
実験ネットワーク
実験資源の一部の
管理権限を移譲
提案する実験環境の管理権限

図 3.2: 管理権限の移譲

第4章　設計

節 3.2 で述べた実験環境を設計する。設計する実験環境の全体像・構成を節 4.1 で述べる。実験環境の生成と制御に関して節 4.2 で述べる。ソフトウェアの構成を節 4.3 で述べる。

4.1　全体像・構成

提案する実験環境の全体構成を図 4.1 へ示す。提案する実験環境の構成は、節 2.1 で述べた StarBED の構成を基に設計した。そのため、管理ノード群や管理ネットワークの役割は、StarBED と同様である。

提案する実験環境では、全体を 1 つの実験環境と見なす。そして、その中にいくつかの実験環境が存在する。それぞれの実験環境中には、管理ノード群と管理ネットワークといくつかの実験ノード・実験ネットワークが存在する。このように、提案する実験環境では、実験環境が入れ子状になって構成されている。提案する実験環境では、1 つの実験環境の単位を NEE(Network Experiment Environment) と呼ぶ。NEE が入れ子状に構成されていることから、このような構成・システムを用いる実験環境を Nested-NEE と呼ぶ

Nested-NEE では、図 4.1 よりも、さらに多段になった実験環境も可能となる。NEE が 3 段なった実験環境構成の例を図 4.2 へ示す。Nested-NEE では上位の管理者に影響を受けない実験環境を、実験者が生成可能となる。1 段目 NEE が管理しているのは 2 段目 NEE である。また、2 段目 NEE が管理しているのは 3 段目 NEE である。そのため、1 段目 NEE からは、3 段目 NEE は不可視となる。このような構成により、実験者が生成可能な実験環境の幅はさらに広がる。

4.2　NEE の生成・制御

本節では、節 4.1 で述べた NEE の生成や制御について述べる。実験者が Nested-NEE を運用するには、節 4.1 で設計した構成を管理するシステムが必要となる。そこで、NEE を生成・制御するシステムを設計する。NEE を新たに生成し、それらを制御することで Nested-NEE を管理する。

これらを実現するために、生成・制御するシステムを持ったノードを追加する。それらを追加した Nested-NEE の構成を図 4.3 へ示す。NEE を生成するために上位の NEE に存在

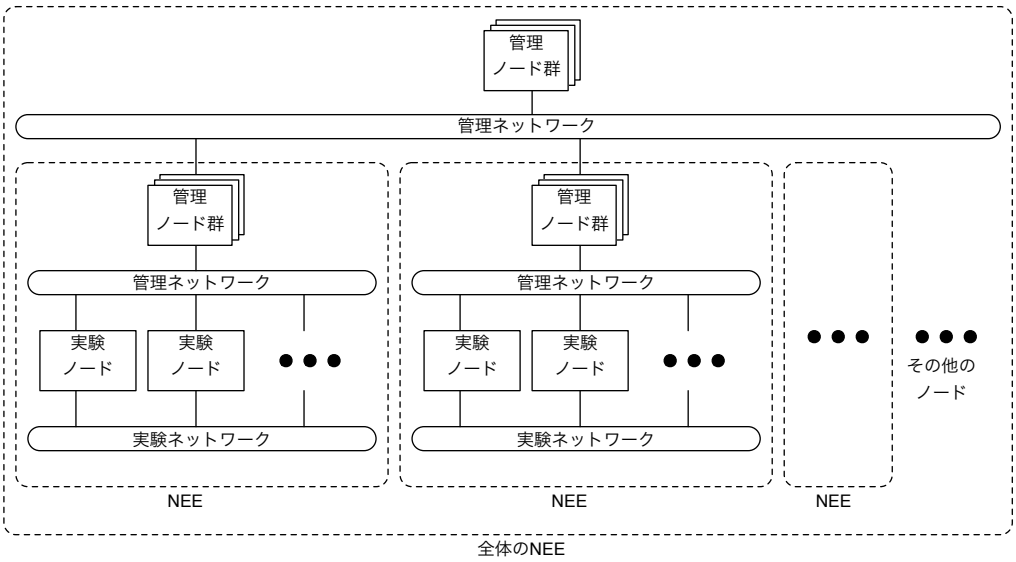

図 4.1: 提案する実験環境 (Nested-NEE) の構成

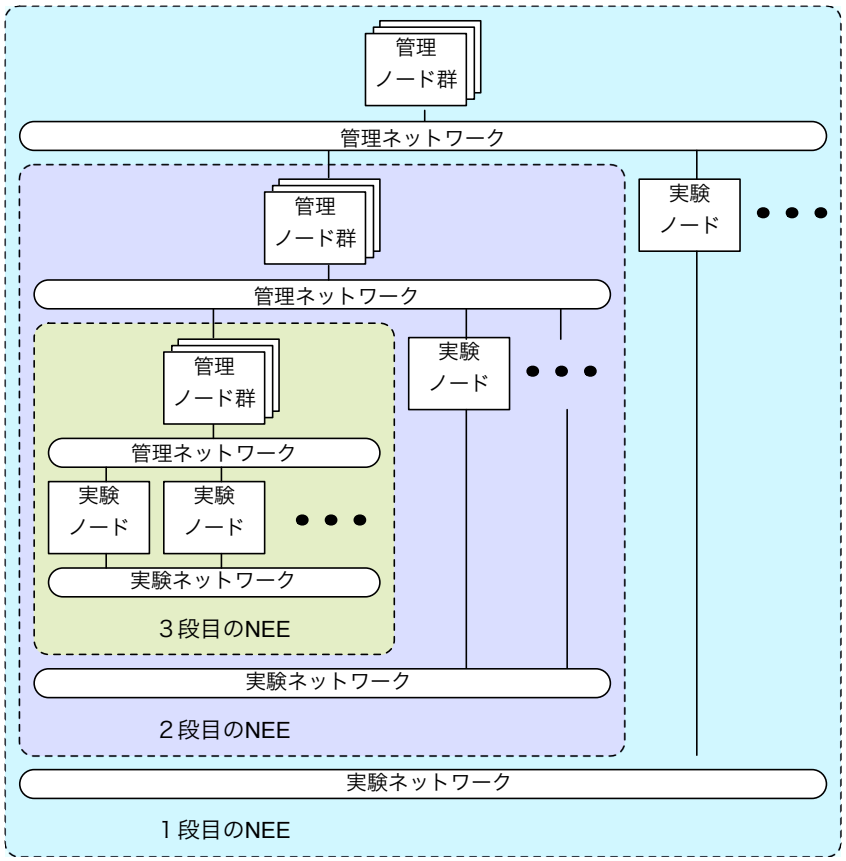

図 4.2: 多段の Nested-NEE

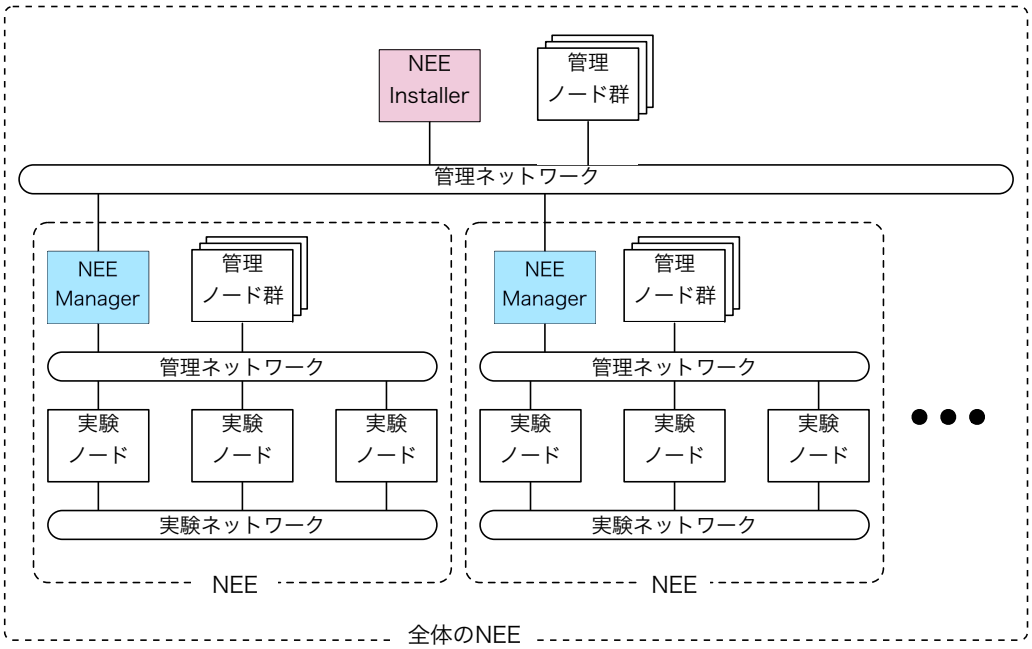

図 4.3: 生成・制御用のノードを追加した Nested-NEE

するノードである NEE Installer を設置する。また、NEE を生成・制御するために実験環境内に NEE Manager を設置する。NEE Installer と NEE Manager の機能を以下へまとめる。

- NEE Installer
  - 実験ノードの取得
  - 取得した実験ノードを管理ネットワークへ接続
  - 実験ノードへ、NEE Manager を導入
- NEE Manager
  - VLAN・実験ノードの取得
  - 管理ノード群の生成・制御
  - 管理ネットワークの生成・制御
  - 実験ノードの生成・制御
  - 実験ネットワークの生成・制御
  - 実験シナリオの実行

Nested-NEE 上で、NEE Installer と NEE Manager を利用して、NEE を生成・制御する手法を解説する。Nested-NEE では、NEE を生成する前の状態を図 4.4 へ示す。上位の管理

ノード群と管理ネットワークが存在する。そして、実験ノードはそれらに接続されていない状態である。Nested-NEE では、この状態から NEE を生成する。

4.2.1 NEE Installer：NEE Manager の導入

最初に、NEE Installer を利用して NEE Manger を導入する。この動作を図 4.5 へ示す。NEE Installer は、管理ノード群の ERM へ接続し実験ノードを取得する。取得した実験ノードを上位の管理ネットワークへ接続する。次に、実験ノードへ NEE Manager を導入する。NEE Manager の導入には、SpringOS の機能を用いる。このように NEE Installer は、NEE Manager を生成する。

4.2.2 NEE Manager：NEE の生成

次に、NEE Manger を利用して NEE の生成を行う。NEE Manager の動作を図 4.6 へ示す。実験者は、生成する実験環境の構成を NEE スクリプトへ記述する。NEE スクリプトは、実験環境の構成を記述した設定ファイルである。NEE Manager は、NEE スクリプトの記述通りに VLAN や実験ノードを取得する。VLAN や実験ノードの取得は、上位の管理ノード群の ERM へ接続し実行する。上位の管理ノード群から取得するのは、新たな資源は上位の管理下にあるためである。このようにして実験者は、利用可能な範囲の実験資源を取得することができる NEE Manager は、取得した VLAN や実験ノードの情報を利用して管理ノード群と管理ネットワークを生成する。実験ノードの OS 導入や実験ネットワークの生成は、SpringOS の機能を用いる。NEE Manager は、SpringOS のソフトウェアを呼び出すことができるため、NEE スクリプトにそれらの機能を直接記述することも可能である。さらに、NEE スクリプト上で SpringOS の Kuroyuri を呼び出すことで実験シナリオを実行する。このような機能により、NEE スクリプトを記述することで、NEE 生成から実験シナリオの実行を行うことができる。つまり、NEE スクリプトを記述することにより、実験環境構築から実験シナリオ実施を通して行うことが可能となる。

4.3 ソフトウェアの設計

Nested-NEE のソフトウェア構成を図 4.7 へ示す。NEE Installer は、SpringOS を包括し利用・制御するソフトウェア群となる。NEE Manager は、SpringOS・DHCP サーバを包括し利用・制御する。また、ファイル転送に用いるソフトウェア (FTP サーバ・TFTP サーバ・HTTP サーバ・NFS サーバ) と協調する。NEE Installer の設計を節 4.3.1 で述べる NEE Manager の設計を節 4.3.1 で述べる

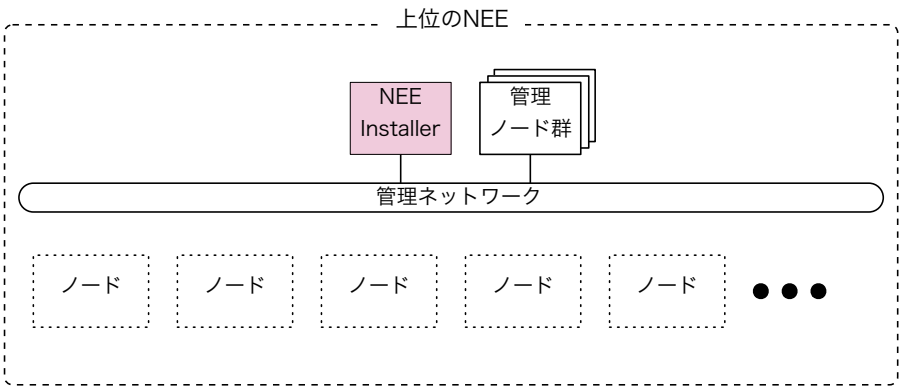

図 4.4: 上位の NEE のみ存在する状態

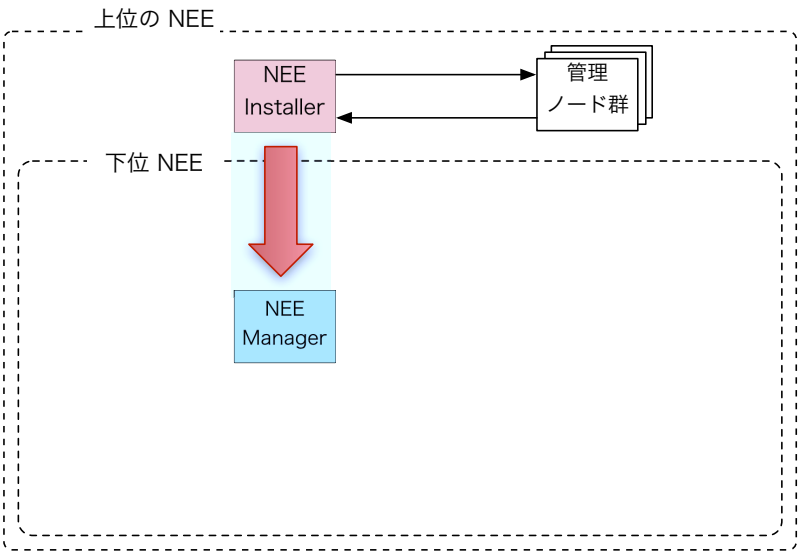

図 4.5: NEE Installer の動作

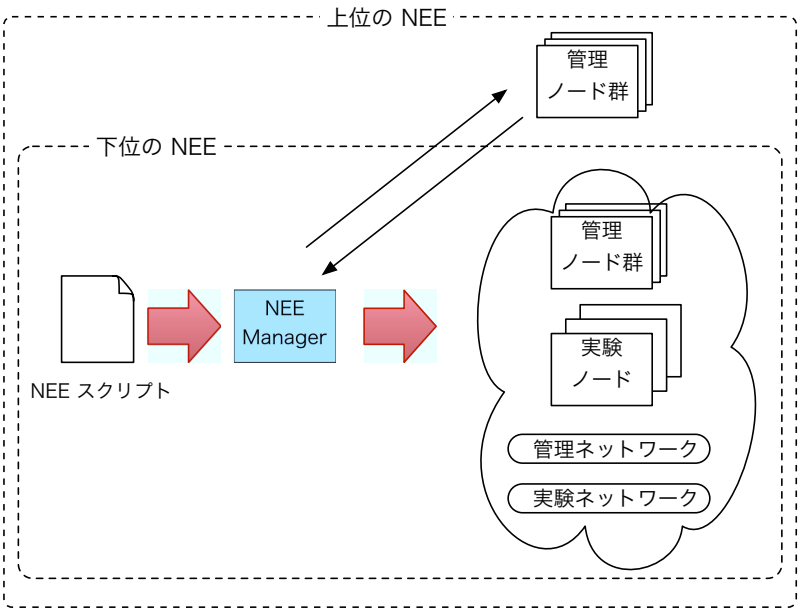

図 4.6: NEE Manager の動作

4.3.1 NEE Installer の設計

節 4.2 でまとめた機能を満たす設計を行う。

実験ノードの取得

実験ノードは、ERM と通信することで確保することができる。ERM と通信するには、ERRP(Experiment Resource Reservation Protocol) を利用する機能が必要である。NEE Installer が、ERRP を利用することで直接、管理ノード群へアクセスしノードの取得が可能となる

取得した実験ノードを管理ネットワークへ接続

上位の管理ネットワークへ接続するには、実験ノードのスイッチのポートを管理ネットワークの VLAN へ所属させる。スイッチ制御し VLAN を設定するには、SWMG を用いる。そのため、SWMG と通信する機能が必要となる。NEE Installer では、SWMG のクライアントである bswc を呼び出して利用する。bswc の設定ファイルは、NEE Installer が自動的に生成する。

実験ノードへの NEE Manager 導入

実験ノードへ NEE Manager を導入するには、NI を利用する。予め NEE Manager のディスクイメージを作成しておき、それを NI によって、実験ノードへ導入する。NI を利用するために、SpringOS のソフトウェアの一部である wipeout を利用する。NEE Installer は、wipeout を呼び出すことにより NEE Manager のディスクイメージを導入することが可能となる。

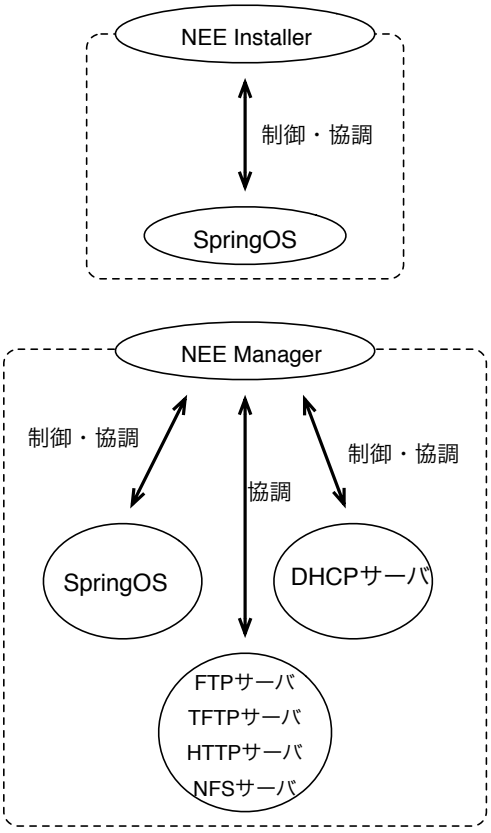

図 4.7: Nested-NEE のソフトウェア構成

4.3.2 NEE Manager の設計

以下へ、各機能の設計を述べる。

VLAN・実験ノードの取得
NEE Installer と同様に ERM と通信することで可能になる

管理ノード群の生成・制御
予め管理ノード群のディスクイメージを作成しておき、NI を利用して、実験ノードへ導入する wipeout を呼び出して利用する。

管理ネットワークの生成・制御
上位の管理ノード群から取得してきた VLAN を利用する。さらに、SWMG と通信することで管理ネットワークを作成する。DHCP サーバを制御し IP アドレスの管理を行う。

実験ノードの生成・制御
従来の SpringOS と同様に NI を用いて、ディスクイメージを導入する。

実験ネットワークの生成・制御
管理ネットワークと同様に VLAN を取得し、SWMG で設定を行う。

実験シナリオの実行
実験シナリオのファイルを取得し、Kuroyuri を呼び出し、実験シナリオを実行する。

4.3.3 NEE スクリプトインタプリタ

NEE 内の構成の制御は NEE スクリプトによって行う。そのため、NEE Manager が必要な情報を設定したり、制御するには NEE スクリプトを用いる。NEE スクリプトを解釈し、実行するソフトウェアとして NEE スクリプトインタプリタを用意する。NEE スクリプトに必要な構文を以下にまとめる。

- 設定用の構文
- 生成用の構文
- 制御用の構文

設定用の構文により、NEE Manager へ値を設定する。例えば、上流の ERM の IP アドレスとユーザ名・パスワード・プロジェクト名である。生成用の構文により、新たにノードやネットワークを生成する。例えば、実験ノードの追加や管理ネットワークの生成である。制御用の構文により、すでに存在するノードやネットワークの情報を制御する。例えば、実験ノードのネットワークインターフェースの情報を変更して利用したい場合や、管理ネットワークの IP アドレスを変更したい場合である。

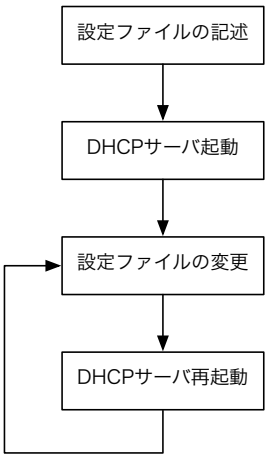

図 4.8: DHCP サーバ設定変更の手順

4.3.4 DCM(Dhcpd Configration Manager)

管理ネットワークを生成・制御したとき、IP アドレス管理の設定を適切に設定する必要がある。IP アドレス管理に影響するのは、ERM のノード情報と DHCP サーバである。ERM のノード情報は、ERM と通信することで変更可能である。多くの DHCP サーバは、動的な設定変更ができない。DHCP サーバの起動と設定変更の手順を図 4.8 へ示す。最初に DHCP サーバを起動するときには、設定ファイルを記述し DHCP サーバを起動する。起動中の DHCP サーバが存在している場合、設定ファイルを変更し DHCP サーバを再起動する。このように、管理ネットワークを生成・制御する場合、IP アドレスの管理に関して手順を踏むことになる。

そこで、これらの作業を自動的に行うシステムを提案する。システムの概要を図 4.9 へ示す。NEE Manager は、ERM と問い合わせ、実験ノードの情報を受け取る。そして、受け取った実験ノードの情報から設定ファイルのデータを生成する。生成した設定ファイルのデータを DCM へ送信する。DCM は、ネットワークから設定ファイルのデータを受信する。そして、設定ファイルのデータを引数として dhcpd を再起動する。これらのシステムにより、DHCP サーバの設定を動的に変更することができる。

4.3.5 仮想マシンを用いた実験ノードの生成

実験ノードの追加として、仮想マシンによるノードを生成する。仮想マシンによるノードの生成・追加の動作を図 4.10 へ示す。はじめに、NEE Manager が仮想マシンの導入を行う。仮想マシンの導入には、以下の機能が含まれる

1) 仮想マシンの導入

2) 仮想マシンによるノードの配布

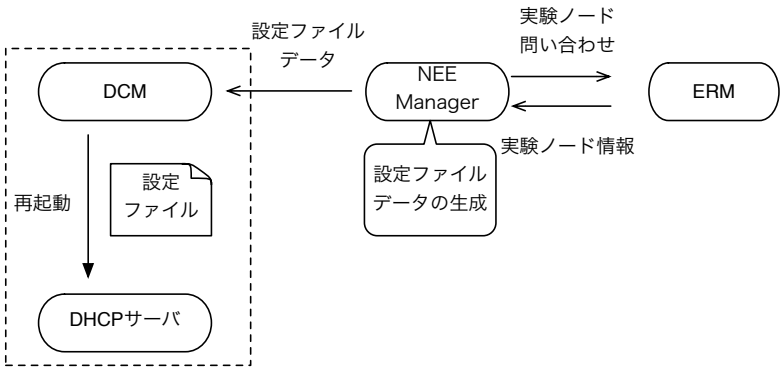

図 4.9: DHCP サーバの動的な設定変更システム

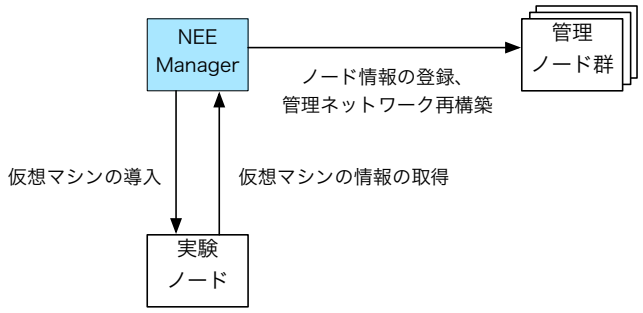

図 4.10: 仮想マシンによる実験ノードの追加

3) 仮想マシンによるノードの複製

4) 仮想マシンによるノードの起動

1) について、仮想化ソフトウェアの導入は、ネットワークブートにより自動的に行う。2) について、仮想ノードは、通常のファイルと同様に扱うことができる。そのため、通常のファイルとして各ノードへ配布する。3) について、仮想ノードを多重化して利用したい場合は、各ホストで複製する。4) について、仮想マシンノードを起動し利用可能な状態にする。

次に、NEE Manager は導入された仮想マシンを制御し、仮想マシンによるノードの情報を取得する。そして、取得した情報から実験ノードの情報を生成し、管理ノード群の ERM へ登録する。

最後に、NEE Manager はアップデートされた ERM の情報を反映させるため、管理ネットワークの再構築を行う。

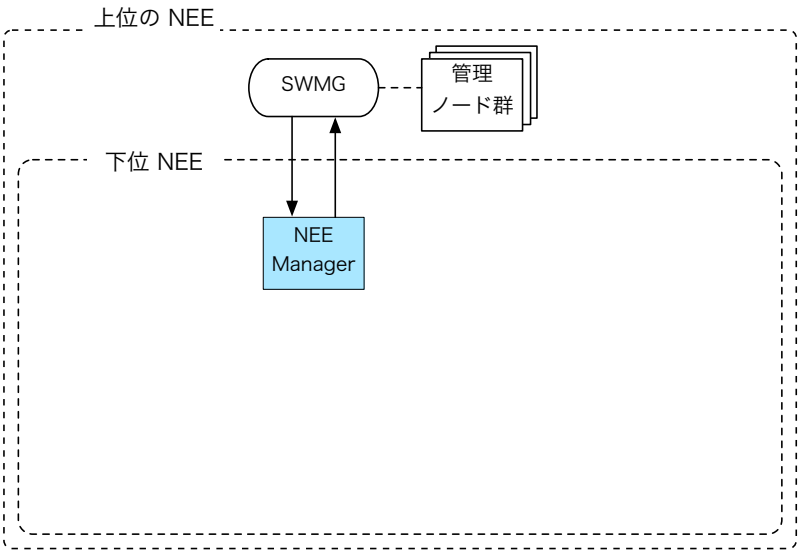

図 4.11: SWMG の利用：上位 NEE と下位 NEE の場合

4.3.6 SWMG Proxy

上位の NEE に設置されている SWMG を利用するためのソフトウェアを設計する。スイッチの設定を行うことのできるノードは限られている。SWMG は、TELNET によってネットワークからスイッチの設定を行う。そのため、スイッチの制御用のネットワークが存在し、SWMG が接続されていなければならない。Nested-NEE の場合、最上位の NEE には SWMG が存在しており利用可能な状態にある。上位の NEE と下位の NEE が存在している場合の構成を図 4.11 へ示す。この場合、下位の NEE は、上位の NEE の管理ネットワークと接続しているため SWMG との通信が可能である。最上位の NEE から数えて、3 段目の NEE が存在している場合の構成を図 4.12 へ示す。この場合、3 段目の NEE の NEE Manager からは、最上位の SWMG に通信することができない。このような仕組みをとった場合、3 段目以降の NEE ではスイッチの制御が不可能となる。Nested-NEE は、ネットワークを動的に作成する必要があるため、スイッチの制御がなければ成り立たない。そこで、SWMG の通信を中継するソフトウェアとして SWMG Proxy を設置する。図 4.12 へ、SWMG Proxy を設置した場合の構成を図 4.13 に示す。このように最上位の意外の NEE で SWMG Proxy を設置することで多段になった場合もスイッチの制御が可能となる。

4.3.7 PWMG Proxy

上位の NEE に設置されている PWMG を利用するためのソフトウェアを設計する。IPMI は、ネットワークからマシンを監視・制御するためのプロトコルである。PWMG は、IPMI 等によって実験ノードの電源を制御する。IPMI のネットワークは、管理ネットワークや実

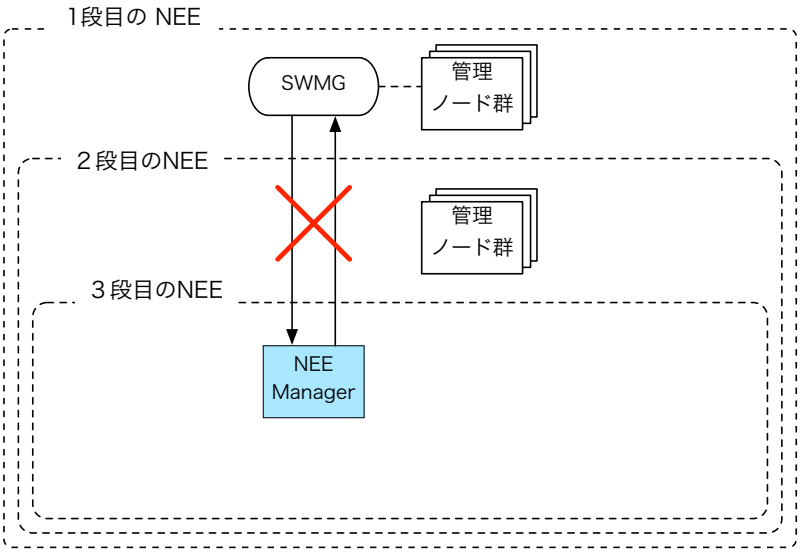

図 4.12: SWMG の利用：3 段構成 NEE の場合

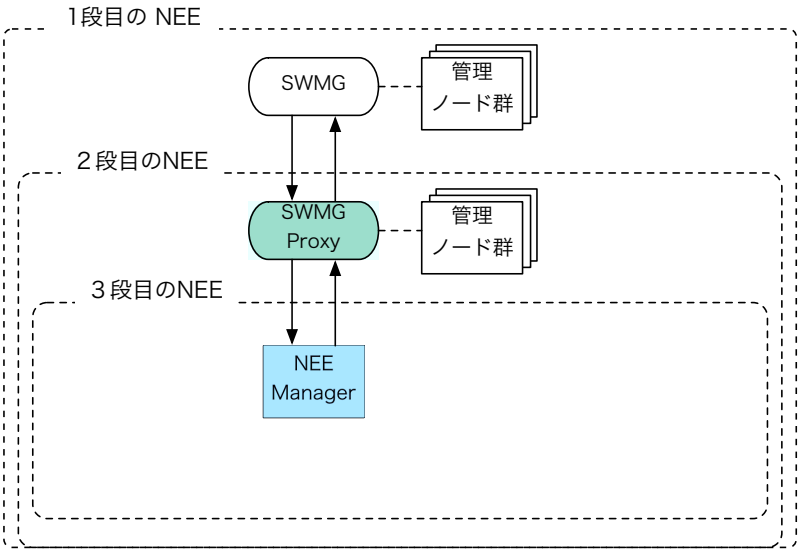

図 4.13: SWMG Proxy の設置

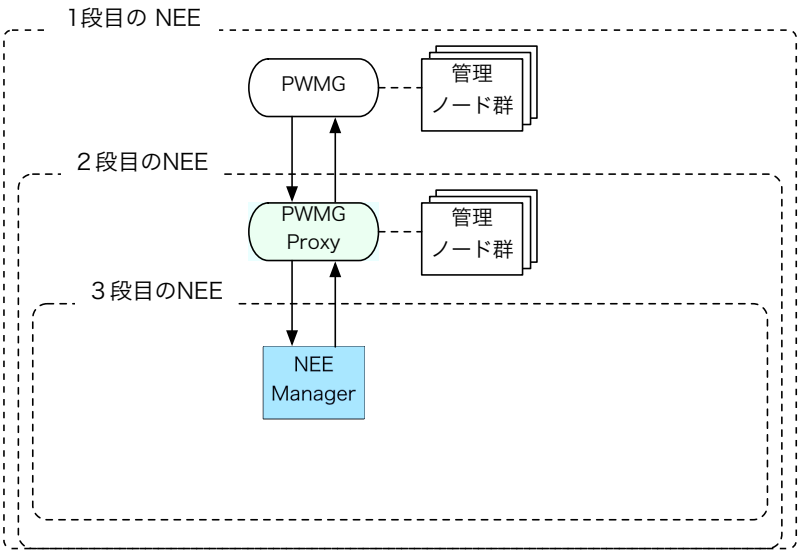

図 4.14: PWMG Proxy の設置

験ネットワークとは別の専用ネットワークを利用する。そのため、節 4.3.6 で述べた SWMG の場合と同じ理由により、多段になった場合利用できなくなる。実験ノードの電源制御がリモートにより行えない場合、実験の利便性が大幅に下がる。そこで、SWMG と同じく PWMG Proxy を設置する。多段の NEE において、PWMG Proxy を設置した時の構成を図 5.3 へ示す。このように、SWMG と同様に多段に対応することが可能となった。

第5章　実装

5.1　実装環境

実装に利用した環境やソフトウェアについて述べる。実装に利用したソフトウェアを表 5.1 へ示す。実装する OS には、CentOS 5.5[11] を採用した。CentOS は、Linux ディストリビューションの 1 つである。実装する OS に CentOS を採用したのは、SpringOS の開発環境と合わせるためである。SpringOS は、開発版のものを利用する。SpringOS 開発版は、最新の機能が備えられている点とより多くのスイッチへ対応している。

実装するためのプログラミング言語として Python[12] を利用する。また、Python のモジュールとして paramiko[13] と IPy[14] を利用する。paramiko は Python 上で SSH[15] を利用するためのモジュールである。Python 上で SSH を利用することで、プログラム上でマシンの制御することができる。IPy は、Python 上で IP アドレスを扱うためのモジュールである。IPy を利用することで、IP アドレスを簡単に生成・管理することができる。

その他、IP アドレス管理やファイル転送を利用するためのソフトウェアを CentOS のパッケージから導入する。CentOS はパッケージ管理システムの yum[16] を用いて、簡単にソフトウェアを導入することができる。ISC DHCP[17] は、ISC[18] により実装された DHCP サーバである。Apache[19] は、広く利用されている HTTP サーバである。vsftpd[20] は、CentOS のパッケージで利用可能な FTP サーバの 1 つである。仮想マシンによる実験ノードのために、仮想化ソフトウェアとして VMware ESXi[21] を利用する。仮想ノードを利用するための仮想化ソフトウェアには VMware ESXi を用いる。VMwareESXi は、スーパバイザ型の仮想化ソフトウェアである。最後のに、実装したソフトウェアを管理するソフトウェアとして Mercurial[22] を利用した。

5.2　SpringOS との協調

Nested-NEE では、SpringOS のソフトウェアを協調・制御する。そのために、SpringOS のソフトウェアと通信するための機構が必要である。SpringOS のソフトウェアは、それぞれのプロトコルによって通信している。そこで、それらのプロトコルによって通信できるクライアントモジュールを実装する。このモジュールは、Python 上で利用するものである。

表 5.1: 実装に利用したソフトウェア

OS	CentOS 5.5
SpringOS	SpringOS 開発版
プログラミング言語	Python
Python モジュール	paramiko
	IPy
CentOS のパッケージ	DHCP server (ISC DHCP Server V3.0.5-RedHat)
	TFTP server
	HTTP server (Apache 2.2.3)
	FTP server (vsftpd 2.0.5)
	NFS server
仮想化ソフトウェア	VMware ESXi
バージョン管理システム	Mercurial

表 5.2: ERRP コマンドの一部

コマンド	機能
LIST	実験資源を列挙する
INFO	実験資源の情報を表示する
REGIST	実験資源を登録する
UPDATE	実験資源の情報を更新する
FINDHOLD	実験資源を検索し取得する。

5.2.1 ERRP クライアントモジュールの実装

ERM は、実験資源の情報を保持するソフトウェアである。ERM は、ERRP(Experiment Resource Reservation Protocol) を用いて、通信を行う。ERM と通信するために、ERRP クライアントモジュールを実装する。ERRP のコマンドの一部を表 5.2 へ示す。このようなコマンドを用いることで、ERM に直接、情報の問い合せや登録を行う。

5.2.2 DMAN クライアントモジュールの実装

DMAN は、NI や OS 起動を切り替えるためにソフトウェアである。DMAN は、ブートファイルへのシンボリックリンクを作成する、ネットワークブートのファイルを切り替える。DMAN との通信には、DMP(Directory Manipulation Protocol) を通信を利用する。そのため、DMP を利用するためのプログラムとして DMP クライアントモジュールを実装

表 5.3: DMP コマンドの一部

コマンド	機能
SYMLINK	シンボリックリンクを作成
RMSYMLINK	シンボリックを削除
MKDIR	ディレクトリを作成
REMOVE	ファイルを削除

```
rm 172.16.1.241 1234
rmuser shin
rmpasswd ***
rmproject proj
sm 127.0.0.1 1240

activate node001-002:0
deactivate node003-004:1

joinvlan 100 node001-002:0

exit
```

図 5.1: bswc の設定ファイルの例

した。DMP コマンドの一部を表 5.3 へ示す。このようなコマンドにより、DMAN を制御する。

5.2.3 SWMG クライアントモジュールの実装

本研究では、任意スイッチの設定を変更し、VLAN によるネットワークを構築する。そのため、SWMG を利用する機構が必要である。SWMG は、SWCP(SWitch Configuration Protocol) を用いて、通信を行う。提案するネットワーク実験環境では、SWMG のクライアントプログラムである bswc を呼び出し、設定を反映するプログラムを実装した。bswc の設定ファイルの例を図 5.1 へ示す。最初に ERM の IP アドレスやユーザ名を設定する。activate は、任意のノードのポートをオンする構文である。また、deactivate は、任意のノードのポートをオフにする構文である。joinvlan は、任意のノードのポートを VLAN へ所属させるための構文である。このような設定ファイルを動的に生成し、bswc を呼び出し実行することで SWMG クライアントモジュールとする。

5.2.4 PWMG クライアントモジュールの実装

PWMG は、IPMI を用いることにより、ノードの電源操作を行う。PWMG は、PWCP(PoWer Configuration Protocol) を用いて通信を行う。PWCP のコマンドの一部を図 5.4 へ示す。本

表 5.4: PWCP コマンドの一部

コマンド	機能
PWON	電源オン
PWOFF	電源オフ
PWFORCEOFF	強制的に電源オフ
REBOOT	再起動

研究では、任意にノードの電源操作を行うために PWCP を利用するためのプログラムとして PWCP クライアントモジュールを実装した。

5.3 NEE Installer の実装

5.3.1 管理ノード群の導入

管理ノード群の導入には、NEE Installer を用いる。NEE Installer は、上位の ERM へ利用可能なノードを問い合わせる。利用可能なノードに問い合わせには、ERRP の FINDHOLD コマンドを用いる。NEE Installer に備わっている ERRP クライアントにより実行する。

次に、取得したノードを上位の ERM の管理ネットワークへ接続する。管理ネットワークへの接続には、SWMG とそのクライアントである bswc を利用する。NEE Installer には、bswc へ設定を直接入力し実行する機能が備わっている。最後に、NI によりディスクイメージを取得したノードへ書きこむ。

5.4 NEE Manager 実装

5.4.1 DCM

管理システムを構築するために必要な DCM の実装について述べる。節 4.3.4 では、DCM が DHCP サーバを管理することを述べた。DCM が想定する DHCP サーバのプログラムは、ISC DHCP(以下、dhcpd と呼ぶ) である。dhcpd は、起動する際に引数としてネットワークインタフェースと設定ファイルが必要である。DCM では、これらの情報をネットワークから受信し、dhcpd を再起動する。DCM の通信プロトコルで利用するコマンドを表 5.5 へ示す

表 5.5: DCM コマンド

コマンド	機能
RELOAD	設定ファイルを受信し dhcpd を再起動する。
RELOADI	設定ファイルとネットワークインターフェース名を 受信し再起動する。
IFCONFIG	dhcpd を起動させるためのネットワークインターフェースへ IP アドレスを設定する。

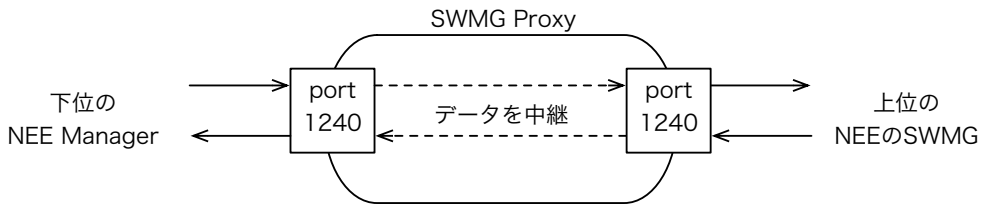

図 5.2: SWMG Proxy

5.4.2 SWMG Proxy

SWCP は TCP によって通信し、ポート番号 1240 番を利用するそのため、TCP の通信を中継するプログラムを実装した。SWMG Proxy の動作を図 5.2 へ示す。SWMG Proxy は、起動すると下位の NEE Manager からの接続が待ち受けた状態となる。接続があった場合、SWMG Proxy が上位の NEE の SWMG に接続しに行く。そして、リクエストやレスポンスのデータを中継する。このような仕組みにより SWMG Proxy を実装した。

5.4.3 PWMG Proxy

PWCP は TCP によって通信し、ポート番号 1242 番を利用する。SWMG Proxy のポート番号を変えただけのプログラムである。PWMG Proxy の動作を図 5.3 へ示す。仕組みについても SWMG Proxy と同様である。

5.4.4 ノード情報の登録・更新

ノード情報の登録・更新について、ERRP クライアントを用いて実装した。ノード情報の登録を行うまでの手順を図 5.4 へ示す。ノード情報の書き換えは、状態の属性など登録できない値を適切に変えるために必要である。ノード情報の更新を行うまでの手順を図 5.5 へ示す。更新におけるノード情報の書き換えは、実際に属性の値を書き換えること指す。

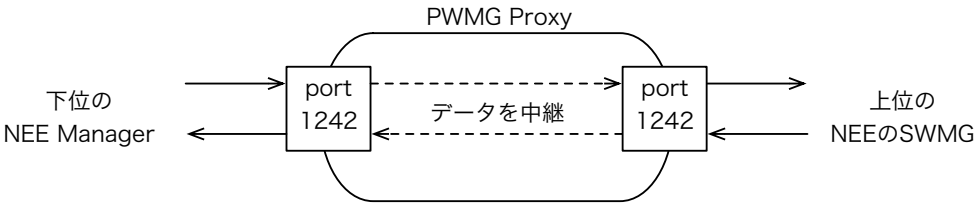

図 5.3: PWMG Proxy

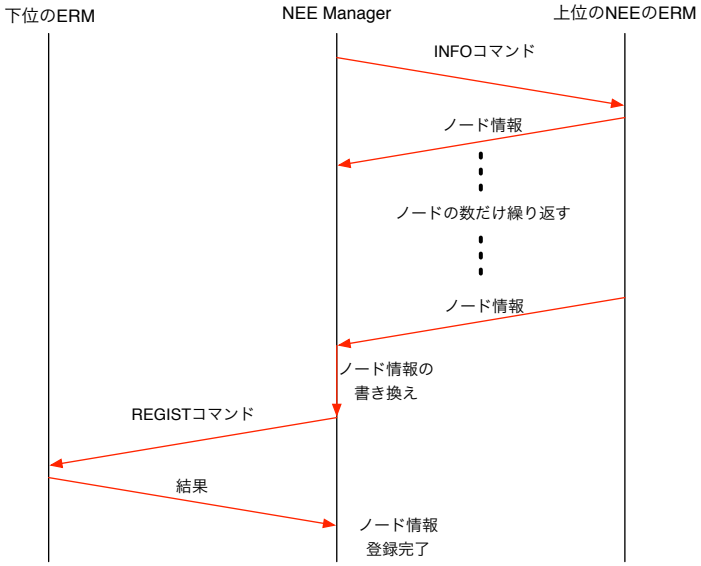

図 5.4: ノード情報登録の手順

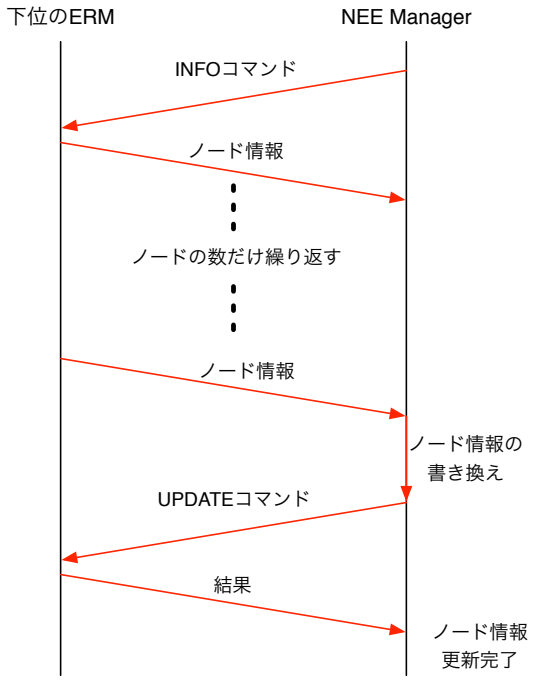

図 5.5: ノード情報更新の手順

5.4.5 DHCP サーバの設定変更

DHCP サーバの設定を変更し反映するまでの手順を図 5.6 へ示す。NEE Manager は、ERM へノードの情報を問い合わせる。そして、取得したノード情報から DHCP サーバの設定ファイルのデータを作成する。作成した設定ファイルのデータを DCM に送信することで、DHCP サーバの設定変更を行う。

5.4.6 仮想ノードの追加

5.4.6.1 仮想化ソフトウェアの選択

また、この節では、仮想マシンと実機を区別するために、実機を利用したノードのことを実機ノードと呼ぶ。

以下の理由により VMwareESXi を選択した。

1) ゲスト OS の移動・複製

2) ゲスト OS の制御

3) 柔軟なネットワーク設定

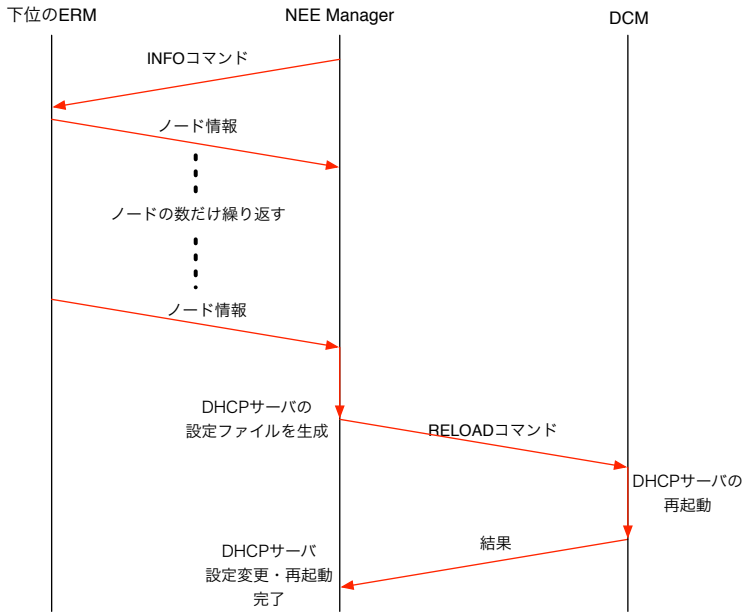

図 5.6: DHCP サーバの設定変更手順

Nested-NEE では、複数のゲスト OS を用意し仮想ノードとして利用する。そのため、ゲスト OS の移動・複製が容易にできる必要がある。VMware ESXi 上のゲスト OS は、ハイパーバイザ上のファイルとして構成されている。そのため、通常の OS 上で扱うことがが可能である。また、ファイルを複製することにより、ゲスト OS が複製できる。

実機ノードは、PWMG によって電源を制御し OS の起動や停止を行う。しかし、仮想ノードは、IPMI のような電源を制御する機構は備えられてない。仮想ノードの電源の制御は、ハイパーバイザ上で行う。そのため、ネットワークからハイパーバイザに接続し、電源を制御する機構が必要となる VMware ESXi は、SSH によりログインしゲスト OS 制御用のソフトウェアを利用することができる。それらのソフトウェアによって、仮想ノードの制御を行う。

仮想ノードは、ネットワーク設定をハイパーバイザ上で行う。Nested-NEE では、仮想ノードと実機ノードを同じように扱う。そのため、実機ノードに近いネットワーク環境を用意する。VMwareESXi では、仮想的なスイッチである vSwtich が利用可能である。vSWitch を設定することで、実機ノードに近いネットワーク環境をすることできる。

5.4.6.2 仮想化ソフトウェアの導入

Nested-NEE では、仮想化ソフトウェアを複数の実機ノードへ自動的に導入する。VMwareESXi は、ネットワークブートを用いた導入が可能である。VMwareESXi の自動インストールに必要な構成を以下へ示す。

1) 実機ノード：PXEBOOT が可能なネットワークインタフェース

2) NEE Manager：DHCP サーバ

3) NEE Manager：TFTP サーバ

4) NEE Manager：HTTP サーバ

PXEBOOT(Preboot Execution Environment)[23] は、ネットワークブート環境の 1 つである。実機ノードのネットワークインターフェースが対応している必要がある。Nested-NEE は、実機ノードへの OS 導入等に、ネットワークブートを用いる。そのため、1)・2)・3) は、これらの環境は既に利用可能である。4) について、VMwareESXi の自動インストールでは、ハイパーバイザのイメージファイルと、インストールスクリプトを HTTP サーバから取得するため必要である。

また、NEE Manager にインストール状況を把握する機能を実装した。インストールスクリプトには、3 つの段階でスクリプトを記述することができる。3 つの段階は以下である。

- インストール前
- インストール後
- 最初の起動

それぞれの段階で、NEE Manager へメッセージを投げることによりインストール状況を把握することができる。また、インストールスクリプトへ vSwitch の構成を記述することで、起動時に反映させることができる。

5.4.6.3 仮想ノードの作成

実験者は、仮想ノードとなる雛形のゲスト OS を準備する。そして、ハイパーバイザ上のゲスト OS のファイルを取り出し、NEE Manager で保存する。VMwareESXi におけるゲスト OS を保存するのに必要なファイルは、以下である。

- vmx ファイル：ゲスト OS の構成を記述したファイル
- vmdk ファイル：ゲスト OS のディスクイメージ

5.4.6.4 仮想化ソフトウェアの制御

仮想化ソフトウェアの制御は、SSH によるリモート接続で行う。VMwareESXi のハイパーバイザは、SSH によるリモート接続が可能である。しかし、標準では有効になっていない。ため、インストールスクリプト中に SSH を有効にするスクリプトを記述した。従って、起動時より SSH によるリモート接続が可能となる。SSH の接続には、Python のモジュールである Paramiko を利用した。

VMwareESXi のハイパーバイザは、通常の UNIX 系の OS ようにコンソールへログインすることができる。また、基本的なコマンドやゲスト OS を制御するためのツールが備わっている。

5.4.6.5　仮想ノードの配布・複製

前節で述べた、ゲスト OS のファイルを、複数の実機ノードへ配布する。ゲスト OS を複数に配布する様子を図 5.7 に示す。ゲスト OS の配布は、NFS を用いて行う。NEE Manager 側で、NFS サーバを用意する。ハイパーバイザは、NFS サーバをファイルシステムにマウントする。vmx ファイルは、NFS サーバをマウントしたディレクトリから複製する。vmdk ファイルも同様だが、複製時にはハイパーバイザに用意された専用のソフトウェアを用いる。さらに、ハイパーバイザ上に仮想ノードを複数用意する場合、取得したゲスト OS を複製する。

vmx ファイルには識別子が存在し、ゲスト OS ごとに異なっている必要がある。また、MAC アドレスが記述されており、ゲスト OS ごとに異なっている必要がある。そのため、複製する際に、vmx ファイルからこれらの記述を取り除き複製する。vmdk の複製には、ディスクイメージを操作するためのコマンドである vmkfstools を利用する。

配布・複製したゲスト OS のファイルは、ハイパーバイザへ登録する必要がある。登録は、ゲスト OS を制御するコマンドである vim-cmd によって行う。vim-cmd コマンドは、ゲストの OS を制御するコマンドである。ゲスト OS の状態確認や起動・停止に利用する。

5.4.6.6　仮想ノードの実験資源化

以下の手順で仮想ノードを実験資源化する。

1) ゲスト OS の情報の取得

2) ERM への登録

3) 管理システムの再構築

最初に、vmx ファイルから取得する。次に、仮想ノードの情報を作成し、ERRP の REGIST コマンドを用いて ERM へ登録する。最後に管理システムを再構築する。

5.4.7　NEE スクリプトインタプリタ

節 4.3.2 では、管理システムの構築や実験資源の追加は、NEE スクリプトインタプリタによって実行することを述べた。NEE スクリプトは、行指向の設定ファイルとして設計した。処理の流れを図 5.8 へ示す。このような流れにより文法を評価していく。

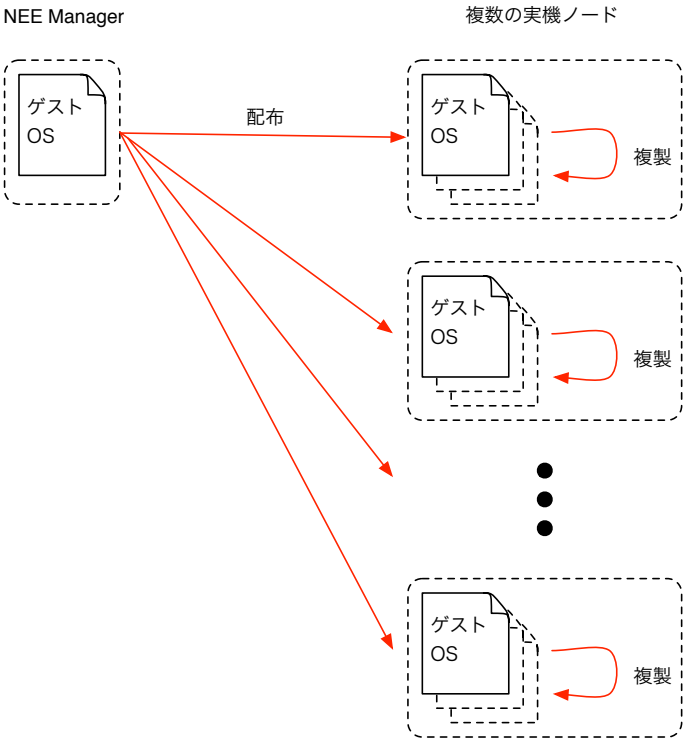

図 5.7: ゲスト OS の配布・複製

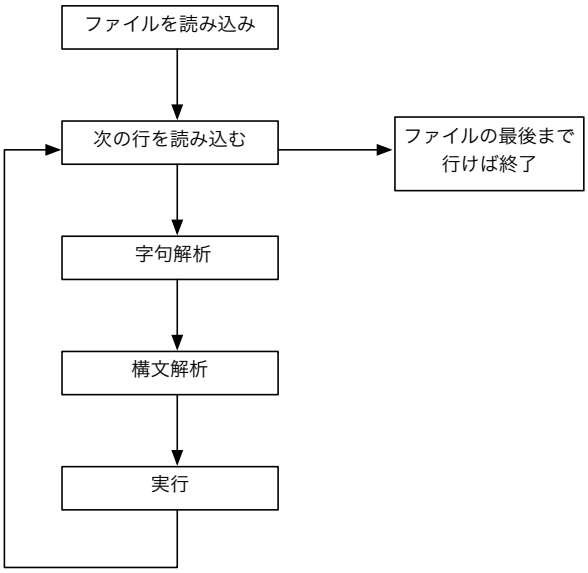

図 5.8: NEE スクリプトインタプリタの処理の流れ

第6章　評価

本章では、Nested-NEE を評価する。節 6.1 では、Nested-NEE の動作を実験により検証する。そして、節 6.2 では、Nested-NEE を既存研究と比較し評価する。

6.1　動作検証

Nested-NEE の動作を検証するために 2 つの実験を行った。小節 6.1.1 では、NEE の隔離に関する実験を行う。また、小節 6.1.2 では、実験ノードの追加に関する実験を行う。

実験は、本学に設置されている Murubushi[24] を利用した。Murubushi は、SpringOS を導入した実験設備である。Murubushi の機材のうち、実験用に提供されたものを利用する。それらの構成を図 6.1 へ示す。また、それぞれの機器の仕様を表 6.1 へ示す。exppc001〜020 は、実験ノードとして利用可能な PC である。expsw1 は、IPMI のネットワークと管理ネットワークを構成するためのスイッチである。expsw2 と expsw3 は、実験ネットワークを構成するためのスイッチである。

2 つの実験のために、実験シナリオを作成した。実験シナリオは、ノードをランダムに 5 つ選択しネットワークの帯域を測定するものである。以降からは、この実験シナリオのことを” 実験シナリオ A” と呼ぶ。実験シナリオ A のプロセスとその流れを図 6.2 に示す。K 言語による実験シナリオ A の記述を図 6.3 に示す。kuma は、Kuroyuri のマスターのプロセスである。また、kusa は、Kuroyuri のスレーブのプロセスである。kusa は、kuma から実験シナリオを受け取り実行する。iperf.py は、５つのノードをランダムに選択する。そして、選択したノードに対して iperf[25] を実行する。全てのノードへの iperf を実行し終えたら実験終了である。

6.1.1　実験ノード追加

6.1.1.1　実験目的

実験ノードが追加できることを検証する。追加する実験ノードは、仮想マシンによる実験ノードである。また、追加された実験ノードが、実験に利用可能か検証する。付加的な目的として、NEE の生成から実験の終了までの時間を調査する。

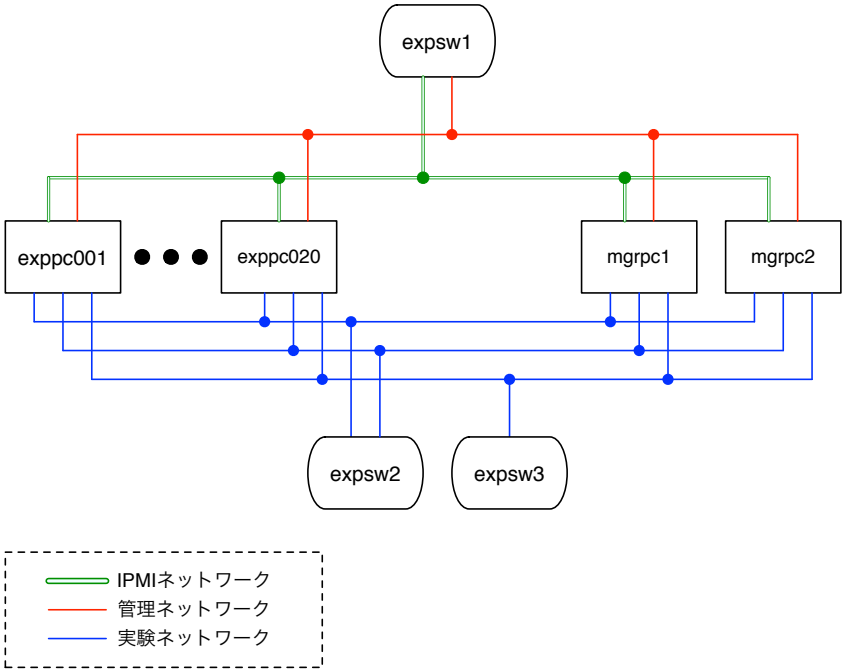

図 6.1: Murubushi の構成

表 6.1: Murubushi の機器仕様

機器	仕様
exppc001〜020	FUJITSU PRIMERGY RX100 S6 Intel(R) Xeon(R) X3430 2.40GHz 8Gbyte Memory SATA 160G 7,200RPM HDD
mgrpc1、mgrpc2	FUJITSU PRIMERGY RX200 S6 Intel(R) Xeon(R) E5506 2.13GHz 8Gbyte Memory ”SAS 146GB 10,000RPM HDD” × 3 (RAID5)
expsw1	D-link DGS-3450
expsw2、exppc3	Brocade FCX648

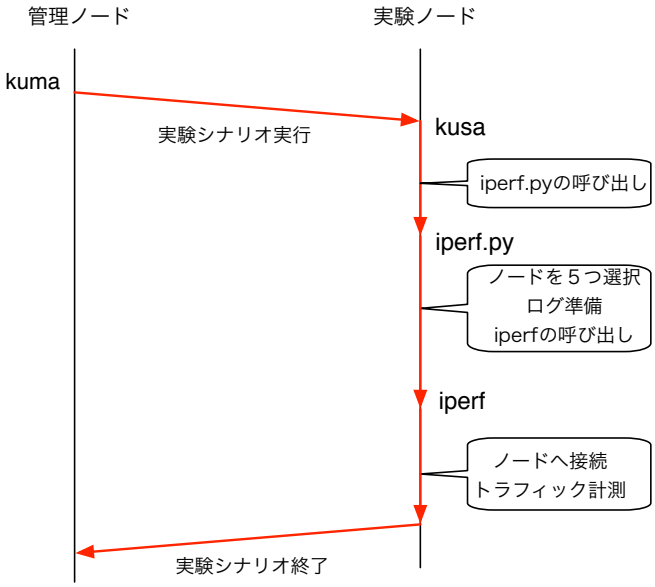

図 6.2: 実験シナリオ A のプロセスとその流れ

```
1       rmanager ipaddr "172.17.0.241" port "1234"
2       fncp ipaddr "172.17.0.241"
3       tftpdman ipaddr "172.17.0.241"
4       encd ipaddr "172.17.0.241"
5
6       user "shin"
7       project "proj"
8
9       ipaddrrange "172.17.0.0/16"
10
11      sparenodemin 0
12      sparenoderatio 100
13
14      nodeclass clclass {
15              method "thru"
16              netif media fastethernet
17              scenario {
18              callw "/usr/bin/python" "/home/shin/iperf.py" "eth0" "eth1" "172.17.0.241"
19              send "done"
20              }
21
22      }
23
24      nodeset client class clclass num 4
25
26      scenario {
27              sync {
28              multimsgmatch client "done"
29              }
30
31      }
```

図 6.3: 実験シナリオ A の K 言語記述

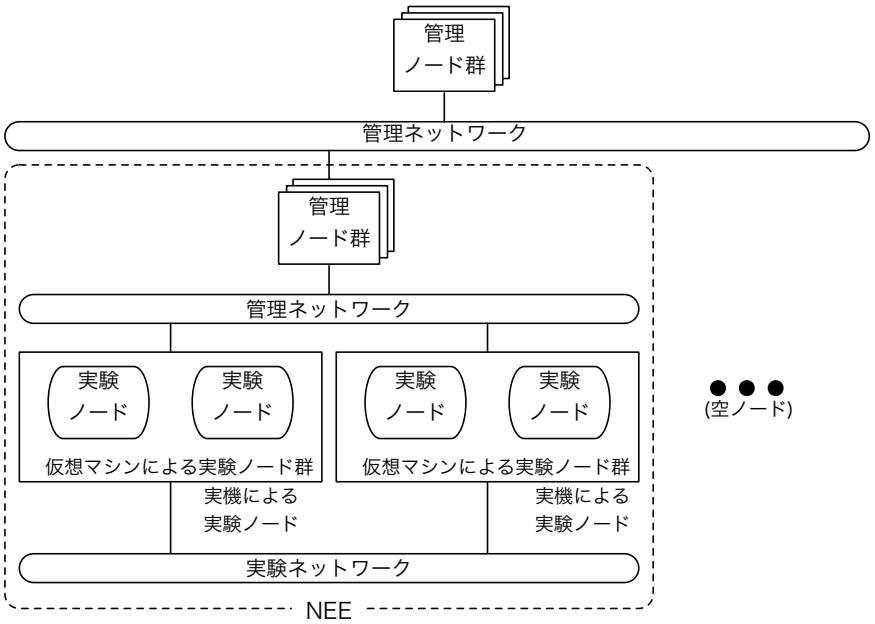

図 6.4: 実験ノード追加：実験環境の構成

6.1.1.2 実験内容

仮想マシンによる実験ノードを生成し、実験資源として利用する。実機による実験ノード毎の仮想マシンによる実験ノードの数の組み合わせをいくつか用意し実験する。また、それらの組み合わせごとに、NEE の生成から実験終了までの時間を計測する。

実機による実験ノードを 2 台利用し、その上に仮想マシンによる実験ノードを 2 台作成する場合を例として、実験環境の構成と NEE スクリプトの記述について述べる。実験環境の構成を図 6.4 へ示す。実験に利用する実験ノードは、仮想ノードによるものである。そのため、計 4 台の実験ノードを利用することになる。

NEE スクリプトを 6.5 へ示す。利用した NEE スクリプトについて詳しく解説する。最も左の数字は、解説のために付加した行番号である。

```
1       set upper_erm ipaddr 172.16.1.241
2       set upper_erm user shin
3       set upper_erm password ***
4       set upper_erm project proj
5
6       set erm user shin
7       set erm password ****
8       set erm project proj
9
10      set dhcpd next_server 172.17.0.241
11
12      set ESXi user root
13      set ESXi password ****
14
15      node num=2
16
17      management_network if=1 172.17.0.0/255.255.0.0
18      experiment_network if=2
19      experiment_network if=3
20
21      install_esxi num=2
22      create_vnode esxi node_num=2 vm_copy=2 debain_exp
23
24      call kuma /home/shin/Dev/scenario/iperf.sc
```

図 6.5: 実験で利用する NEE スクリプト

1-4 行目　　上位 NEE の IP アドレスとユーザ・パスワード・プロジェクト名である。
6-8 行目　　下位 NEE の IP アドレスとユーザ・パスワード・プロジェクト名である。
10 行目　　DHCP サーバの設定である。今回は、next server のみ設定した。
12-13 行目　　VMwareESXi のパスワードである。
15 行目　　取得するノードの数である。
上位 NE から 2 台の実機ノードを確保することを示す。
17 行目　　管理システムの構築を示す。
ノードの 1 番目のネットワークインタフェースを利用し、
172.17.0.0./255.255.255.0.0 サブネットで管理システムを構築する。
18-19 行目　　2 番目と 3 番目のネットワークインターフェースを利用して構築する。
21 行目　　ノードから 2 つ選び出し VMwareESXi をインストールすることを示す。
22 行目　　VMware ESXi のインストールされたノードを 2 つ選び、
仮想ノードを 2 つずつ準備することを示す。
debian_exp とは、仮想ノードのファイルが置いてあるディレクトリを示す。
24 行目　　Kuroyuri によって実験シナリオ A(iperf.sc) が、
実験シナリオを実行することを示す。

表 6.2: 実験結果

実機ノード	仮想ノード	総ノード数	実験時間 [s]
2	2	4	548.930
2	4	8	644.844
2	8	16	893.358
4	2	8	595.930
4	4	16	694.683
4	8	32	950.437
8	2	16	691.643
8	4	32	791.363
8	8	64	1033.167

6.1.1.3 実験結果

実験結果から、Nested-NEE の実装が動作し、実験ノードが追加できることを確認した。実験したノードの組み合わせと実験時間を表 6.2 へ示す。さらに、グラフ化したものを図 6.6 へ示す。グラフにより、実験に要する時間は、実機によるノードの数が増加しても実験時間への影響が少ないことがわかる。また、仮想マシンによるノードの数が増加した場合、実験時間に大きく影響することがわかる。

6.1.2 NEE の隔離

6.1.2.1 実験目的

Nested-NEE の設計では、生成した NEE はネットワークが隔離された状態にある。そのため、NEE 内の管理ネットワークや実験ネットワークのトラフィックは、他の NEE に影響を及ぼすことはない。これらを確認するために、複数の NEE を生成し、お互いの影響を検証する。

6.1.2.2 実験内容

今回の実験では、2 つの NEE を生成する。そして、それらのうち 1 つを NEE(A)、NEE(B) と呼ぶ。実験環境の構成を図 6.7 へ示す。NEE(A) では、実験シナリオ A を利用した実験を行う。NEE(B) では、パケットキャプチャによってトラフィックを監視する。そして、NEE(A) のトラフィックが流れていないかを確認する。

NEE(A) の実験は、仮想マシンによる実験ノードを利用し、実験シナリオ A を実行する。NEE スクリプトは、 6.1.1 で利用したものと同様である。

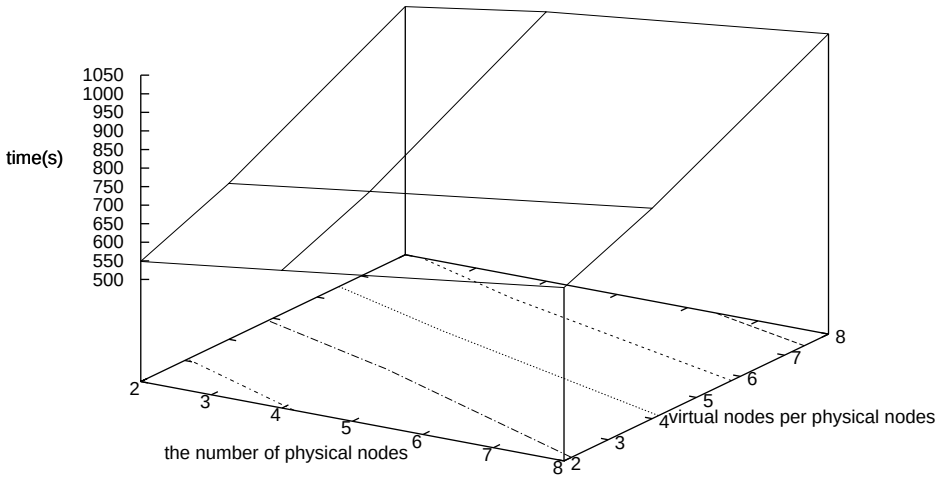

図 6.6: 仮想ノード数と実験時間

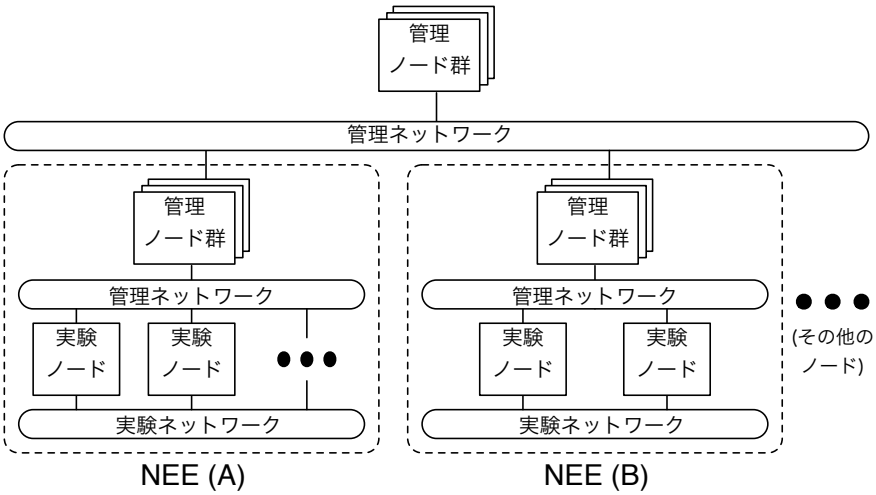

図 6.7: NEE の隔離：実験環境の構成

表 6.3: 確認したパケットの一覧

パケットの種類
Iperf によるパケット
DHCP パケット
ARP パケット
ディスカバリプロトコルのパケット (CDP、LLDP 等)

NEE(B) は、tcpdump[26] を利用してパケットを確認した。確認したパケットのプロトコルの一覧を表 6.3 へ示す。Iperf のパケットは、NEE(A) の実験トラフィックが流れていないかを検証するために確認する。DHCP パケットは、DHCP サーバ・クライアントのパケットが流れていないかを検証するために確認する。ARP やスイッチのディスカバリプロトコルが、NEE(A) のセグメントから流れてきていないかを確認する。スイッチのディスカバリプロトコルとは CDP(Cisco Discovery Protocol)[27] や LLDP(Link Layer Discovery Protocol)[28] 等を指す。CDP や LLDP の他にスイッチの各ベンダーによるディスカバリプロトコルが存在する。それらも含めて、NEE(A) からパケットが流れていないか確認した。

6.1.2.3 実験結果

NEE(B) でトラフィックを監視した結果、NEE(A) のトラフィックが流れていないことを確認できた。NEE(A) と NEE(B) のネットワークは、影響を及ぼしていない。そのため、Nested-NEE では、生成された NEE が隔離されていることが確認できた。

6.2 既存研究との比較

実験環境の作成に関して、既存研究である StarBED と Nested-NEE を比較する。比較する項目とその可否をまとめたものを表 6.4 に示す。

実験ノードの導入は、StarBED では支援ソフトウェアである SpringOS を用いることで可能である。Nested-NEE でも、SpringOS を用いるため実験ノードの導入が可能である。実験ノードの追加に関して、StarBED では実験者が新たに実験ノードを追加することは困難である。困難な理由は章 3 で述べたとおり、実験環境の構成に起因する制約が存在するためである。しかし、Nested-NEE ではそれらを克服するための実験環境を構成すること実験ノードの追加を可能とした。実験ノード情報の変更に関して、実験ノードの追加と同様の理由で StarBED では困難だったが、Nested-NEE では可能とした。管理ネットワークの作成に関して、StarBED では静的に設置してあり、実験者が作成するような機構はな

表 6.4: 実験環境の作成に関する比較

	StarBED	Nested-NEE
実験ノードの導入	◯	◯
実験ノードの追加	×	◯
実験ノード情報の変更	×	◯
管理ネットワークの作成	×	◯
実験ネットワークの作成	◯	◯
他の実験環境との分離	×	△

かった。しかし、Nested-NEE では、実験環境ごとに管理ネットワークを作成して利用するため可能となった。実験ネットワークの作成に関して、StarBED では VLAN を動的に切り替えることで可能としている。Nested-NEE でも、同じ機構を採用しており、実験ネットワークの作成が可能である。

最後に他の実験環境との分離に関して、StarBED では共通の管理ネットワークを利用していた。そのため、実験環境が分離されているとは言えない。Nested-NEE は実験環境ごとにネットワークを隔離されている。そのため、NEE 同士は分離された状態となる。ただし、Nested-NEE では、管理ネットワークや実験ネットワークは分離されているものの、NEE Manager が上位の管理ネットワークと接続されている。そのため、厳密には分離されていない状態とみなし△の評価とした。

第7章　おわりに

本研究では、最適に実験資源を利用するための実験環境を検討した。既存のネットワーク実験設備は、管理システムに起因する実験資源の利用制限が存在した。そのため、存在する実験資源の制限を取り払うことで、より自由度の高い実験環境を目指した。そこで、既存のネットワーク実験設備の管理システムを再検討した。その結果、分割型の管理システムを提案した。提案するシステムでは、従来では静的に設置してあった管理システムを、動的に構築する。そのため、実験者ごとに利用可能な管理システムを実現した。提案するネットワーク実験設備では、実験資源の用途の変更や追加が可能となる。また、提案する実験環境を実装し、評価を行った。その結果、入れ子型の実験環境によって、利用可能な実験資源の中から、実験環境を生成することが可能となった。生成した実験環境は、互いに隔離された状態にあり他の実験環境へ影響を及ぼす可能性やトラフィックの負荷を低減させることが可能となった。また、設備の管理に因われない実験環境を生成することで、実験資源の追加や実験資源の用途変更が可能となった。

7.1　今後の課題

7.1.1　高度な入れ子型の実験環境の生成

Nested-NEE は、入れ子型に対応する設計を行った。そのため、さらに高度な入れ子型の実験環境を生成することも可能である。高度な入れ子型の実験環境の例としては、

例えば、6 章で挙げた 2 段だけではなく多段になったものがある。その他には、構成が同じでパラメータを変化さえた環境がある。現在の設計でも、このような環境自体を生成すること自体は可能である。しかし，パラメータを代入し、実験を行うような機構は考慮されていない。このような、柔軟な実験環境を生成するためには、Nested-NEE を拡張する必要がある。

7.1.2　仮想ノードへの高度な対応

仮想ノードを実験資源として追加し、通常のノードと同様に管理システムを実現した。しかし、SpringOS 自体は実機ノードと仮想ノードを区別しない。そのため、実機ノードと仮想ノードが混在した環境において、ノードの選択が不便となる。例えば、ERM のノード情報へ実機ノードか仮想ノードかを判別可能な属性を加える。あわせて、K 言語に実機

ノードと仮想ノードを区別する機能を加えることで便利になる。今回は、VMwareESXi へ対応したが、より多くの選択肢が存在するべきである。様々な仮想化ソフトウェアに対応することで、さらに柔軟に実験環境を生成できる。

7.2 将来の展望

7.2.1 実験環境の保存と復元

実験環境の保存・復元に関して野中ら [29] の研究により成功している。本研究で提案した入れ子型の実験環境にあわせて、実験環境の単位である NEE を保存と復元することで研究対象にすることが可能になる。NEE により実験環境を抽象的に扱うことができ、より保存と復元が柔軟かつ容易に行うことが可能になると考えられる。さらに、NEE スクリプトを拡張することで制御に関しても、容易に行うことが可能と考えられる。

7.2.2 実験環境の移送

将来の展望として実験環境の移送がある。実験環境の移送とは、実験環境を異なる施設へ移して利用することである。実験環境の移送を行うことで、様々な利便性が生まれる。例えば、小規模の実験環境から大規模な実験環境への移送することで、より実践的な実験を行うことができる。本研究では、入れ子型の実験環境を設計し、1 つの単位を NEE とした。入れ子型の設計によって、実験環境の構成への依存が少なくなった。構成に関する依存が少なることで、実験環境そのものを抽象的に扱うことができ、移送が行い易くなる。

謝辞

本研究を進めるにあたり、数多くのご指導を頂いた指導教員の知念賢一特任准教授、副指導教員の篠田陽一教授に深く感謝致します。また、多くの助言を頂いた主テーマ審査員の丹康雄教授、副テーマ指導員の長谷川忍准教授に感謝致します。

研究や設備利用に関して多く助言を頂いた情報通信研究機構北陸リサーチセンターの三輪信介氏、Razvan Beuran 氏、宮地利幸氏、中田潤也氏、太田悟史氏、中川岳史氏、中井浩氏、下口宗裕氏、石崎淳氏、竹中ゆかり氏に感謝致します。

研究室ゼミ等にて活発な議論や助言を頂いた、情報科学センターの宇多仁助教、小原泰弘助教に感謝致します。

研究や生活面において、お世話になった本研究室の LATT Khin Thida 氏、高野祐輝氏、安田真悟氏、井上朋哉氏、Nguyen Lan Tien 氏、Nguyen Nam Hoai 氏、Muhammad Imran Tariq 氏、松井大輔氏、明石邦夫氏、川瀬拓哉氏、立花一樹氏、中村祐輔氏、橋本将彦氏に感謝致します。また、繁忙期に数々の協力を頂いた本研究室の山田悠介氏、鍛治祐希氏に深く感謝致します。

参考文献

[1] Toshiyuki Miyachi and Ken-ichi Chinen and Yoichi Shinoda. StarBED and SpringOS large-scale general purpose network testbed and supporting software. *International Conference on Performance Evaluation Methodlogies and Tools (Valuetools) 2006*, Oct 2006.

[2] StarBED Project. http://www.starbed.org/.

[3] StarBED Project - SpringOS. http://www.starbed.org/softwares/springos.html.

[4] 北陸リサーチセンター. http://starbed.nict.go.jp/.

[5] IEEE Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. IEEE Std 802.1Q-1998.

[6] R. Droms. Dynamic Host Configuration Protocol, RFC2131. March 1997.

[7] K.R. Sollins. TFTP Protocol (revision 2), RFC783. June 1981.

[8] 知念賢一, 宮地利幸, 篠田陽一. Kuroyuri: ネットワーク実験記述言語処理系. コンピュータソフトウェア, Vol. 27, No. 4, pp. 4_43-4_57, 2010.

[9] 三角真, 宮地利幸, 知念賢一, 篠田陽一. 実ノードを利用したネットワークシミュレーションにおけるノードへの OS の導入及びパラメータ設定機構の開発. 情報処理学会研究報告書 DSP-116, pp. 95-100, 2004.

[10] Intel Corporation. IPMI - Intelligent Platform Management Interface. http://www.intel.com/design/servers/ipmi/.

[11] CentOS. http://www.centos.org/.

[12] Python. http://www.python.org/.

[13] paramiko - SSH2 protocol for Python. http://www.lag.net/paramiko/.

[14] IPy - A python Module for handling IP-Addresses and Networks. http://c0re.23.nu/c0de/IPy/.

[15] T. Ylonen, C. Lonvick, and Ed. The Secure Shell (SSH) Connection Protocol, RFC4254. January 2006.

[16] yum. http://yum.baseurl.org/.

[17] DHCP — Internet Systems Consortium. http://www.isc.org/software/dhcp.

[18] Internet Systems Consortium. http://www.isc.org/.

[19] The Apache Software Foundation. http://www.apache.org/.

[20] vsftpd - Secure, fast FTP server for UNIX-like systems. http://vsftpd.beasts.org/.

[21] VMware vSphere Hypervisor. http://www.vmware.com/jp/products/vsphere-hypervisor/.

[22] Mercurial SCM. http://mercurial.selenic.com/.

[23] Intel Corporation. Preboot Execution Environment (PXE) Specification Version 2.1, 1999.

[24] Murubushi. http://www.jaist.ac.jp/ k-chinen/pj/murubushi/.

[25] Iperf. http://iperf.sourceforge.net/.

[26] Tcpdump/libpcap public repository. http://www.tcpdump.org/.

[27] Cisco Systems. Cisco discovery protocol.

[28] IEEE Standard for Local and Metropolitan Area Networks–Station and Media Access Control Connectivity Discovery, IEEE 802.1AB, 2005.

[29] 野中雄太. ネットワーク実験環境の保存と復元に関する研究. Master's thesis, 北陸先端科学技術大学院大学, 2008.