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

Title	サービスの段階的開発検証環境の構築
Author(s)	松井，大輔
Citation	
Issue Date	2010-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/8942
Rights	
Description	Supervisor:篠田陽一，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

サービスの段階的開発検証環境の構築

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

松井 大輔

2010 年 3 月

修 士 論 文

サービスの段階的開発検証環境の構築

指導教官　篠田陽一 教授

審査委員主査　篠田陽一 教授
審査委員　知念賢一 特任准教授
審査委員　敷田幹文 准教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

610079 松井 大輔

提出年月: 2010 年 2 月

Copyright © 2010 by Daisuke Matsui

概 要

インターネット上では、様々なサービスが展開されている。これらは、他のサービスによるトラフィックや、経路中の障害、伝送遅延など、様々な外部要因の影響を受ける。このため、新しいサービスの検証においては、それらによる影響を確認する必要がある。本研究では、サービスの開発を対象に、多様な検証を行える実験環境の構築を目指した。

ソフトウェアを開発する際には、意図どおりに動作するかの検証が必要不可欠である。開発初期の段階においては、サービスが遅延などによる影響によって、予想外の動作を行っても原因の切り分けが困難になる。その動作が、遅延などの外部要因の影響よる挙動なのか、サービス自身の問題なのかを判断しにくいため、事前にサービスの動作検証が必要になる。サービスへの影響を検証するには、サービスの動作検証を行った後、同じ環境に外部要因を追加して検証を行う必要がある。このようにサービスの検証においては、検証段階に応じて、適切な環境を構築して検証を行う必要があり、段階的に行うことが効果的である。本研究では、このような検証を段階的検証と呼ぶ。

本研究では段階的検証において想定する段階として、3 つの段階を定義した。サービスのみの段階、外部要因を追加した段階、インターネット上で検証を行う段階を想定する。検証を支援するための既存技術として、テストベッドと呼ばれる実験用の設備がある。テストベッドには、それぞれ適した用途があり、分散配置されていて、インターネット上で試験を行う段階に適したものや、実験用の施設として設置されていて、サービスのみの段階に適したものがある。また、そのようなテストベッドは、遅延などを追加することにより、外部要因を追加した段階も実現できる。このように、テストベッドを用いることで各々の検証段階が実現できる。しかし、全ての検証段階を１つのテストベッドで実現することは困難である。

段階的検証においては、複数のテストベッドを使い分けることでの、実現が考えられる。しかしテストベッドごとに、使用する OS や、管理システム、使用のための設定などが異なるため、使い分ける場合は、使用のための設定といった準備作業を、テストベッドごとに行わなければならない。これはテストベッドごとに、想定している用途が異なり、それぞれ制御方法や設定項目が異なるためである。また異なるテストベッドを使い分ける場合、移行に伴う作業が必要になり、さらに、前段階のテストベッドで行っていた設定や管理情報を共有することができない。複数のテストベッドを、一つの大きなテストベッドとしてまとめて扱うことができれば、同一の管理システムによって制御できるため、透過的な移行が見込める。また制御方法などはテストベッド間で統一することが好ましい。使用するテストベッド全てで、同じ OS や管理システムを使用できるように、実験環境を構築しなければならない。

本研究では、実験用の施設として設置されているテストベッドである StarBED を対象に、提案システムのプロトタイプを実装した。その上で、インターネット上での検証を行う段階に適したテストベッドである PlanetLab と、同じ OS や管理システムを使用する

MyPLC を用いて、実験環境を構築した。MyPLC の機能に拡張を加えることで、StarBED と PlanetLab を、一つの大きなテストベッドとして扱うことを可能とした。また、試験段階の移行方法として、ネットワーク接続の柔軟な変更を実現した。トンネルプロトコルなどを用いて、施設内部のネットワークとインターネットの接続を切り替えることにより、外部要因の追加や、インターネットとの接続を行うことが可能となった。また、PlanetLab と StarBED の計算機を併用することによって、より大規模な実験環境の構築を実現した。更に、そのようにして構築された環境での、実験の実行を支援するため、既存技術の適用を検討し、拡張を行った。

本研究の評価として、StarBED と PlanetLab、そして閉じた環境上で MyPLC を用いて構築した環境と、提案システムで構築した環境を比較した。その結果、本研究の手法で想定する試験段階を、容易に実現できることを確認した。特に、インターネット上で検証を行う段階は、PlanetLab を使用することと、トンネル接続の、二通りの方法で実現できることを確認した。

本研究では、円滑な段階的検証を行うため、これらの機能からなるテストベッドアーキテクチャを実現した。本研究の成果として、特性の異なる複数のテストベッドを組み合わせることにより、各検証段階に適した実験環境を容易に構築できることと、それらを透過的に利用することが可能となり、幅広い実験が実現できた。これにより、円滑な段階的検証が可能となった。

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　ネットワークサービスの開発と検証 **3**
2.1 ソフトウェアの開発サイクル 3
2.2 ネットワークサービスの開発における考慮事項 4
2.2.1 分散サービスの例 5
2.2.2 分散サービスの開発と検証 5

第 3 章　既存の実験支援技術 **7**
3.1 インターネットを使用するテストベッド 7
3.1.1 PlanetLab 7
3.1.2 PlanetLab Japan 9
3.1.3 OneLab 9
3.2 実験用の施設を使用するテストベッド 9
3.2.1 StarBED 9
3.2.2 XENebula 10
3.2.3 GARIT/AnyBed 10
3.2.4 NetBED/Emulab 10
3.3 まとめ 10

第 4 章　段階的検証のためのテストベッドアーキテクチャの提案 **11**
4.1 要素と試験環境 11
4.1.1 サービスのみの環境 11
4.1.2 周辺要素を再現した環境 12
4.1.3 インターネットに接続された環境 13
4.2 各試験環境に求められる特性 13
4.2.1 pure stage に求められる特性 13
4.2.2 emulated stage に求められる特性 14
4.2.3 Internet stage に求められる特性 14

4.3 移行に伴う作業 14
4.3.1 周辺要素の追加・削除 14
4.3.2 インターネットとの接続・切断 14
4.3.3 アプリケーションの設定・実験内容の変更 15
4.4 実験環境の構築 15
4.4.1 複数のテストベッドの併用 15
4.4.2 実験用ノードの OS の統一 16
4.4.3 OS のメディアレスインストール 16
4.4.4 実験環境の管理 17
4.4.5 実験環境の拡張 17
4.4.6 実験環境の初期化 19
4.4.7 実験環境の保持 19
4.4.8 時刻の同期 19
4.4.9 試験対象サービスとの分離 21
4.5 実験実行支援 21
4.5.1 実験の記述と実行 21
4.5.2 サービス独自のプロトコルの制御 22
4.5.3 ログの収集 22
4.5.4 ログへの時刻の挿入 23
4.5.5 グループ化による制御 23
4.6 ネットワーク接続の変更 23
4.6.1 周辺要素を模倣したネットワークへの接続 25
4.6.2 インターネットへの接続 27
4.7 テストベッド間の協調 29
4.8 まとめ 29

第 5 章 既存技術の使用の検討 31
5.1 テストベッドの選択 31
5.1.1 PlanetLab 31
5.1.2 StarBED 31
5.1.3 本研究で使用するテストベッド 31
5.2 実行実験支援ソフトウェア 32
5.2.1 CoDeploy 32
5.2.2 pssh 33
5.3 実験環境の構築 33
5.3.1 OS のメディアレスインストール 33
5.3.2 SpringOS 33
5.3.3 時刻の同期 36
5.4 トンネルプロトコル 37

5.4.1 L2TP 37
5.5 テストベッド間の協調 38

第 6 章 提案するアーキテクチャの適用例と実装 40
6.1 Local PlanetLab の構築 40
6.1.1 Local PlanetLab の通常の構築方法 40
6.1.2 通常の構築方法における問題 43
6.2 構築用プログラムを用いた Local PlanetLab の構築 44
6.2.1 PLC ホストを構築 44
6.2.2 SpringOS によるノードの確保 44
6.2.3 PLCAPI によるノードの登録 45
6.2.4 インストールメディアの PXELinux への置き換え 45
6.2.5 plnode.txt の配布方法 47
6.2.6 ノードの起動・インストール 52
6.2.7 実験環境の拡張 52
6.3 実験実行支援 52
6.3.1 グループごとの制御 54
6.3.2 ログの収集 54
6.3.3 ログへの時刻の挿入 54
6.3.4 サービス独自のプロトコルの制御 55
6.4 ネットワーク接続の変更 55
6.4.1 施設内のネットワークへの接続 55
6.4.2 インターネットへの接続 56
6.4.3 トンネル接続の制御機構の実装 56
6.5 Global PlanetLab との連携 60
6.5.1 共有しないノードの設定方法 60
6.5.2 Peer の登録 61
6.5.3 Local ノードの一時的な提供 63
6.5.4 Local 専用 PLC 63

第 7 章 評価 65
7.1 想定する stage の構築の容易性 65
7.1.1 pure stage 65
7.1.2 emulated stage 66
7.1.3 Internet stage 66
7.2 実験環境の構築に関する評価 67
7.2.1 OS のメディアレスインストールについての評価 67
7.2.2 実験環境の初期化に関する評価 68
7.2.3 実験環境の管理に関する評価 69

7.2.4 実験環境の拡張に関する評価 69
7.2.5 時刻の同期に関する評価 70
7.2.6 試験対象サービスとの分離に関する評価 70
7.3 実験実行支援の評価 70
7.3.1 実験内容の記述 71
7.3.2 グループ化による制御に関する評価 71
7.3.3 ログの収集に関する評価 71
7.3.4 ログへの時刻の挿入に関する評価 72
7.3.5 サービス独自のプロトコルの制御に関する評価 72
7.4 テストベッド間の協調に関する評価 72
7.4.1 非共有ノードの設定に関する評価 73

第 8 章 今後の可能性 74
8.1 閉じた施設間の接続と協調 74
8.2 周辺要素同士の協調 74
8.3 MyPLC の開発検証 74
8.4 StarBED の新しい利用形態 74
8.5 他のテストベッドへの応用 75

第 9 章 おわりに 76

第1章　はじめに

1.1　背景

インターネットは社会において重要なインフラとなっており、多くのサービスがインターネット上で提供されている。インターネット上で提供されるサービスの形態も多様化しており、単体でサービスを提供するもの以外にも、複数台で協調して動作してサービスを提供するものも存在する。本論文ではそのようなサービスを分散サービスと呼ぶ。

サービスを開発する際には、各機能が意図通りに動作するか、どの程度の性能が出せるかといった仮定を確認するために、検証が行われる。分散サービスのような、多数のノードが協調して動作を行うものは、協調した動作の検証も行う。このため、検証の際に複数台の計算機を用いる必要があり、管理・操作の面でも作業量が多くなる。効率よく検証を行うためには、多数の計算機を同時に操作するための工夫が必要不可欠になる。また検証の際には、実際に使用する環境を再現した環境で検証を行うことで、より妥当な実験結果を得られる。しかしインターネットのような複雑な環境を再現することはほぼ不可能であり、実際にインターネット上で動作させた際に、検証時には再現していなかった要素により、想定外の動作をする可能性がある。検証環境と、インターネットが異なる環境である以上、挙動が同一のものになるとは限らない。インターネット上で検証を行うことで、この点は解消される。しかし開発初期においては、予期せぬ動作をしても原因の特定が困難になる。そのため、小規模な環境で検証を行ってから、環境を徐々に分散配置された環境に近づけて検証するように、段階的に環境を変更、あるいは移行しながら検証を行う必要がある。本論文ではこのような検証を段階的検証と呼ぶ。小規模な環境での検証後、大規模な環境で検証を行い、問題が起こった場合、問題の原因を特定・修正してから、意図通りに修正されているかを小規模な環境でまた検証する。このため段階的検証においては、何度も実験環境を構築することになる。しかしそれぞれの検証段階に応じて、何度も実験環境を構築することは容易ではない。

1.2　本研究の目的

分散サービスの検証には、段階的検証が効果的である。しかし、段階ごとに実験環境を何度も構築する必要がある。このため検証以外の作業に大きく時間をとられてしまう。

本論文ではこのような段階的な環境構築を容易にするために、考えられる検証段階と、各検証段階に求められる特性、移行に必要な作業を挙げる。そして実験環境を構築して、

各検証段階を円滑に移行するための仕組みを提案する。

1.3　構成

2 章では、ネットワークサービスの開発と、分散サービスの開発における問題点について述べる。3 章では、検証のための既存研究として、テストベッドと呼ばれる実験用の施設や技術を紹介する。4 章では、段階的検証における段階を定義し、テストベッドアーキテクチャを提案し議論する。5 章では、提案技術の実現にあたり、既存技術の使用を検討する。6 章では、既存技術だけでは実現できず、本研究で実装した提案技術について述べる。7 章では、本研究の評価として、既存技術との比較を行う。8 章では、本研究の成果を元に、今後の課題や応用について述べる。

第2章　ネットワークサービスの開発と検証

2.1　ソフトウェアの開発サイクル

インターネット上へサービスを展開する場合、実現するためのソフトウェアの開発が必要不可欠である。本論文では開発とは、図 2.1 のように提案・設計・実装・評価・運用といった工程全体を指す。まずどのようなサービスを実現するかというアイディアの提案から始まり、既存の技術で使えるものは無いかといった検討を行う。次にどのような機能を持たせるか、ユーザインターフェイスはどのようにするかといった設計を行い、仕様書を作成する。設計によって、ソフトウェアの仕様書ができた後、実際にソフトウェアとして実装が行われる。実装後は、意図通りの動作を行うか、想定する性能が出るかといった評価が行われる。

オブジェクト志向に代表されるように、複雑なシステムの実装においては、システムの機能を細かく分け、部品化して設計・実装を行う。検証においても、システムの個々の機能を検証し、挙動を把握することで、問題の切り分けを容易にできる。

予想外の動作をするなどの問題が発生した場合は、前の段階に戻って修正や検証を行う。例えば、図 2.2 のように動作検証を行った際に、意図通りに動作しない、あるいは想定する速度で処理が終わらないなどの問題が起きた場合には、実装の段階に戻って問題の修正を行う。原因が設計上の問題で、実装の段階で修正できない問題であれば、設計の段階に戻って修正を行うことになる。また検証の段階で、設計上の問題であることが自明であれば、検証の段階から設計の段階に戻る。開発は、こうしたサイクルを繰り返して行われる。また、前の段階へ戻らないように、各手順を念入りに行うウォーターフォールモデルと呼ばれる手法もある。

ソフトウェアの開発を終了した後は、問題が発生しても修正を行うことは困難である。開発の段階が終了すると、既にソフトウェアを公開しているため、既に使用されているソ

図 2.1: 開発における段階

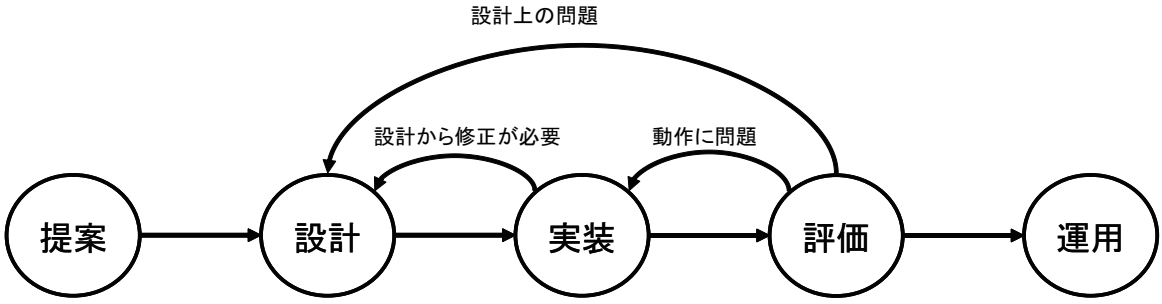

図 2.2: 各段階で行う作業

フトウェアは、開発者の手による変更ができない。問題点を修正し、新しいバージョンのソフトウェアを公開したとしても、利用者が更新しなければ問題は残り続ける。新しいバージョンと古いバージョンが混在することを見据えて、新しいバージョンを、古いバージョンと通信可能なように実装する方法もある。ただしこの場合は、古いバージョンとどこまで共存可能かという、互換性の検証が求められる。

運用中のサービスは停止することが難しい。更新作業のために停止すると、その間サービスは利用できなくなるため、事前に利用者への告知などが必要となる。このためサービスを公開・運用する前に、入念に検証を行うことが求められる。更新のための仕組みをソフトウェアに実装しておく方法もあるが、更新に伴う負荷や、負荷を分散させるための仕組みも踏まえて、更新のための仕組みも検証を行う余地がある。

2.2 ネットワークサービスの開発における考慮事項

前述のように、アプリケーションの開発から公開までは、いくつかの段階を経て行われる。ネットワークサービスの場合は、通信機能の検証も必要になる。通信機能の検証であれば、プログラムを複数起動することや、プロトコルをテストするためのプログラムなどを作成することで試験が行える。しかし単一の計算機での試験では、問題が起きた際に、どのプログラムに問題があったかを判断しにくい。あるいは計算機自体の問題という可能性もある。このように計算機だけではなく、通信相手やネットワークによる影響も考慮しなければならない。ネットワークを使用するアプリケーションは、様々な要素の影響を受ける。例えば、ファイヤーウォールの影響や、経路中の障害によってパケットが届かないことがありえる。更に、必要な処理が問題なく行われていても、通信による遅延や、計算機の負荷によって、十分な性能が出ない場合もある。

このため、通信相手やネットワークを再現して検証を行うことが一般的である。しかし、それらを再現することは難しい。まず機材の確保が必要になるが、開発者が利用可能な機材は限られているため、再現できる規模には限界がある。次に、機材を確保できても、それらを適切に設定してネットワークなどを再現することは、時間的・作業的なコストが高い。このため、機材の確保も含めて、小さな規模でしか再現ができない。そして

機材を確保してネットワークなどを再現しても、インターネットのような複雑な環境を、再現すること自体が難しく、再現した内容の妥当性が問われる。このような問題に対しては、テストベッドと呼ばれる検証用の設備を利用するという方法がある。

2.2.1 分散サービスの例

既存の分散サービスの例としては、CDN や DNS が挙げられる。P2P はサーバやクライアントという概念を持たないが、分散した計算機が協調して動作するネットワークアプリケーションに挙げられる。このような分散サービスの、分散配置や協調動作のメリットには、複数の計算機で処理を行うことによる負荷分散が挙げられる。また CDN では、クライアントの近くに存在する計算機が、サービスを提供することによってパフォーマンスが向上する。更に何らかの理由で、ある計算機がサービスを提供できなくなった場合でも、他の計算機でサービスを継続するといった、耐故障性の向上などのメリットもある。

2.2.2 分散サービスの開発と検証

分散サービスの場合は、更に検証が難しくなる。サービスを構成する計算機の台数が増えるため、検証に伴う作業量が増える。例えば、プログラムに修正を加えた場合、反映させるためには修正版を各計算機にコピーする必要がある。コピー後、検証を行うためには各計算機上で実行する必要がある。このように検証に伴う作業が、台数分増えてしまう。

分散サービスならではの機能として、協調動作の確認や、それによる効果の検証といった作業も必要になる。前述のように、ネットワークによる影響を受ける点も、台数が増えた場合は複雑化する。それらの影響は、計算機が接続されているネットワークによって異なる。分散サービスの場合は、各計算機が同じネットワークに接続されているとは限らず、様々なネットワークに分散配置されていると考えられる。このため、計算機ごとに状況が異なり、ネットワークによる影響も多様なものになる。このようなことから、問題が発生した際に、原因の特定が困難である。このため、分散サービスの動作確認等を行う段階では、遅延やパケットロスといった要素がない環境で行うことが好ましい。他の要素の無い環境で試験を行い、サービスのみの環境での動作を確認してから、他の要素を再現した環境で、他の要素による影響を確認・測定するべきである。また、他の要素の無い環境と、ある環境の試験結果を比較することにより、他の要素による影響を確認することができる。

このように試験内容ごとに、要素を再現するかどうかで異なる試験環境を構築する必要がある。また負荷試験のために大量のクライアントを用意するなど、他の要素を追加しなくても、台数を増やす場合もある。このように、試験ごとに適切な試験環境を構築する必要がある。更に、インターネットの経路や、他のサービスに関連するトラフィックなどは、日々変化しており、忠実に再現することは難しい。このため、インターネットでの試験も視野に入れるべきである。

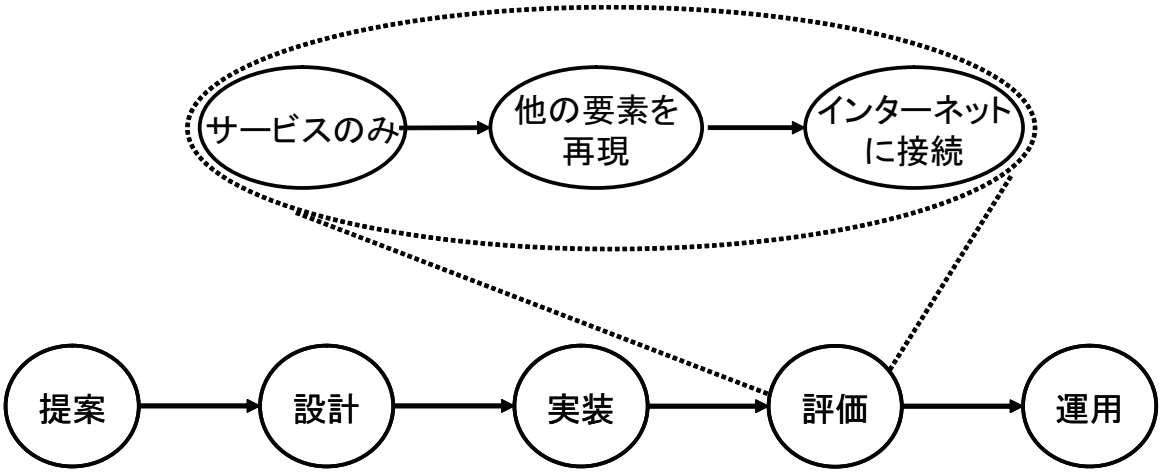

図 2.3: 段階的な検証

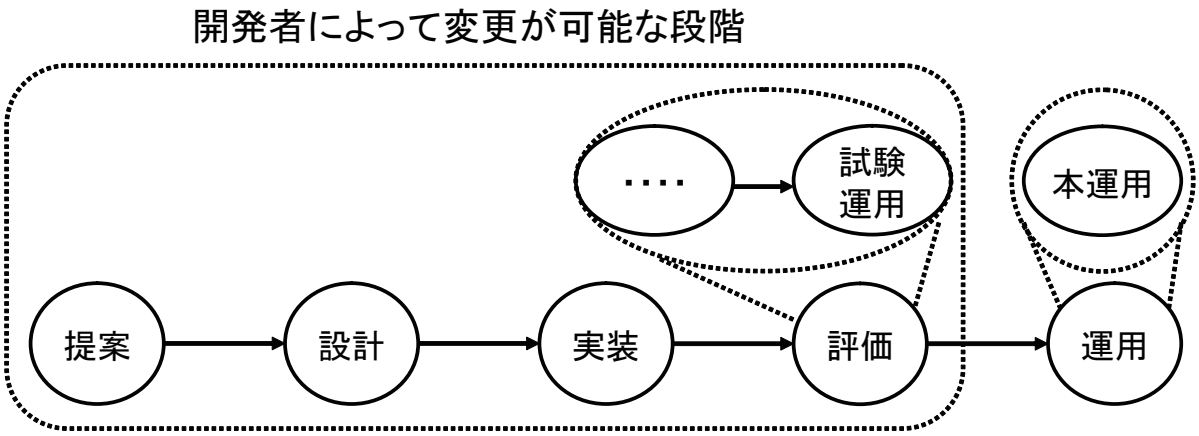

図 2.4: 開発者によって変更可能な範囲

本論文では図 2.3 のように、検証もまた段階に分けて行うと考え、そのような検証を段階的検証と呼ぶ。サービスの運用に関しては、本論文で考える開発の工程には含まないものとする。ただし β テストのように、試験の一環として運用を行う場合があると考え、そのような工程は試験運用と呼ぶ。本論文上の定義では、試験運用は開発の工程に含む。また、開発者の手によって停止などの制御が行えない運用は本運用と呼び、本論文で考える開発の工程には含まない。本論文では、本運用とは開発が終了してから行われる運用を指す。図 2.4 に、開発者によって行われる段階の範囲を示す。

第3章　既存の実験支援技術

本章では既存の実験支援技術として、テストベッドと呼ばれる、実験用の設備や枠組みについて説明する。

3.1　インターネットを使用するテストベッド

3.1.1　PlanetLab

PlanetLab[1] は新しいサービスの開発者と、利用者の両方を支援するためのテストベッドである。PlanetLab は管理システムである PlanetLab Central(以降 PLC とする) と、実験用ノードから構成されている。実験用ノードは、PlanetLab に参加している組織から提供されている。PlanetLab に参加する条件として、実験用ノードを 2 台提供する必要がある。その組織に所属している実験者は、実験用ノードを提供することで、他の組織から提供されているノードを含めて自由に利用できる。数としては、現在 493 の組織が参加しており、提供されているノードは 1000 台を超えている。

実験用ノードは Linux VServer[2] によって多重化し、複数の実験者が同時に利用出来るようになっている。このため、PlanetLab 上での実験で使用するソフトウェアは、Linux 用の実装でなければならない。また、root 権限が必要な操作にも制約がある。

実験用ノードはインターネットを直接利用するため、実験中のサービスをインターネット利用者へ公開できる。また、実際のトラフィックや遅延の影響下で実験も可能である。性能評価や動作検証以外にも、長期にわたる運用実験も想定されている。

Slice

PlanetLab で、実験者に割り当てられるリソースは Slice と呼ばれる。Slice を１台のノードに対して割り当てた単位は Sliver と呼ばれる。Sliver は Linux VServer を用いて実装されており、１つの Sliver が１つの仮想マシンとして実現されている。またメモリなどの資源は、1 つの Sliver が大量に消費しないように制限が設けられている。Sliver の概念図を図 3.1 に示す。

Sliver の操作は SSH[3] を用いて行えるようになっており、実験者が自分の使用する公開鍵を PLC に登録すると、各実験用ノードに対して公開鍵が配布される。Slice の名前がユーザ名として使用されるため、Slice@hostname のようにして Sliver が指定でき、ssh

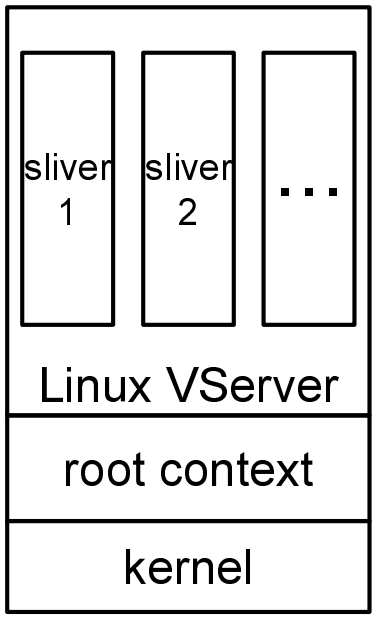

図 3.1: Sliver の構成

Slice@hostname　のように実行することで ssh を用いて Sliver への接続が行える。また SSH を使用するソフトウェアによる、実験支援が可能となっている。実験支援ソフトウェアの例としては、後述する CoDeploy や pssh がある。

MyPLC

公式な PlanetLab とは別の PlanetLab 網を構築するために、MyPLC というソフトウェアが PlanetLab の開発元である Princeton 大学より公開されている。これを用いて複数のサイトからノードを提供し合い、PlanetLab 同様に分散環境を構築することができる。そのような PlanetLab の亜種としては PlanetLab Japan[4] などがある。また 1 人の実験者が、私的に使う環境を構築することもできる。本論文では、実験者が MyPLC を用いて閉じたネットワーク上に構築した環境を Local PlanetLab と呼ぶ。また公式な PlanetLab や PlanetLab Japan のように、インターネットを介し、複数の実験者で共有して使用するための環境を Global PlanetLab と呼ぶ。

PlanetLab はネットワークを経由して操作するため、インターネットの遅延や障害が操作に影響を及ぼす。また実験用ノードは他の実験者のサービスなども動作している。そのため、十分な性能が出ない可能性がある。このようなことから、PlanetLab へ展開する前にサービスを開発する環境として、Local PlanetLab を利用することが考えられる。

MyPLC は 1 人の実験者が大規模な環境を構築することは想定されておらず、インストールの方法などの点から、大規模な Local PlanetLab の構築は困難である。構築方法や問題点については後述する。

3.1.2 PlanetLab Japan

MyPLCで構築されたPlanetLabの亜種に、PlanetLab Japanがある。PlanetLab Japanは2005年に東京大学を中心に発足された。PlanetLab Japanはより手軽に使えるPlanetLabを目指しており、実験者はノードを提供せずとも利用できる。

3.1.3 OneLab

PlanetLab Europeを拡張したテストベッド。無線LANで接続しているノードのための拡張等が行われている。現在、26の組織が参加している。また、PlanetLab EuropeやPlanetLab Japanと連携しており、実験者は連携先のノードも合わせて利用できる。

3.2 実験用の施設を使用するテストベッド

3.2.1 StarBED

PlanetLabのような、多くの組織が計算機を提供し合うテストベッドとは逆に、一つの施設内に全ての資源があるテストベッドがStarBED[5]である。StarBEDは約1000台の計算機を備えており、全てL2スイッチに接続されている。L2スイッチのVLANを変更することによって、仮想的にトポロジを変更する。また操作・管理を行うための管理用ネットワークと、実験対象が実際に通信をするための実験用ネットワークに分かれている。これによりトラフィックの計測などの実験では、操作・管理に伴うトラフィックを分離することができる。実験者はノードを自由に利用することができ、OSなどについても好きなものを使用することができる。また、計算機は何種類か用意されており、計算機の構成ごとにグループとして分けられている。実験者は用途に合わせて、ネットワークインターフェイスの数や性能などを元に使用するノードを選択できる。グループごとに同じ構成の計算機を多数用意しているため、均一な性能のノードが利用できる。

大規模な実験を支援するため、SpringOSという実験支援ソフトウェア群が用意されている。実験用の計算機以外の構成要素として、DHCP[6]サーバやTFTP[7]サーバがある。また後述するSpringOSに含まれている、管理用のサーバプログラムもある。施設としてのStarBEDは、北陸リサーチセンターのことを指すが、同じ構成要素を用意することで、北陸リサーチセンター以外の場所でも構築できる。

StarBEDの利用の形態としては、予約制である。これは時分割多重といえる。実験者は利用期間を決めて申請し、その期間中、割り当てられた機材を用いて実験を行う。利用期間が過ぎた後は、別の実験者に資源が割り当てられる。このため利用に際しては、所要時間も重要な要素になる。

3.2.2 XENebula

XENebula[8] はクラスタ環境を対象に、各計算機を xen によって多重化し、より大きな規模の実験環境を構築する。XEN のイメージを変更することにより、様々な OS を起動させることができる。

3.2.3 GARIT/AnyBed

GARIT は奈良先端科学技術大学院大学で運用されているテストベッドである。異なる組み合わせの計算機を使用するための仕組みとして、AnyBed がある。AnyBed は L3 ネットワークの構築を目的としており、トポロジ記述ファイルを元に構築を行う。物理的な配線は変更せず、L2 スイッチを操作し、VLAN を変更して実験用トポロジを構築する。各ノードではルーティングデーモンが起動し、経路情報を交換して動的に経路設定が行われる。CAIDA の AS データを元にトポロジ記述ファイルを作ることによって、AS の単位で抽象化してインターネットを再現できる。また AnyBed は GARIT 以外の環境も想定しており、StarBED や XENebula 上でも使用が可能である。

3.2.4 NetBED/Emulab

NetBED[9] はシミュレータと実機の両方の長所を取り入れたテストベッドである。ns[10] の記述にしたがってネットワークを構築することが可能である。資源が空き次第、実験が実行され、実験終了後、自動的に資源が解放される。このため、ある状態で実験環境を保持しておくことはできない。また、施設間をインターネットで接続しているため、インターネット中の背景トラフィック等の影響を取り入れることができるが、それらを選択することはできないため、実験内容に応じて背景トラフィック等を取り入れるかどうかは選択することができない。

3.3 まとめ

既存のテストベッドでは、いくつかの段階には対応できる。しかし、全ての段階に対応できるテストベッドは存在しない。段階的検証のためには、複数のテストベッドを使い分ける必要がある。テストベッドごとに、使用方法等が異なるため、移行する度に新しい方法を覚える必要がある。また、前段階のテストベッドと同じ内容の実験をして比較する場合、実験の制御方法が異なると、実験内容の解釈が変わる可能性がある。

第4章　段階的検証のためのテストベッドアーキテクチャの提案

本研究では、段階的検証を支援するためのテストベッドを提案する。本章ではそのアーキテクチャについて議論する。実験のサイクルとして、以下の流れを想定する。

1. 実験内容の決定
2. 実験環境の構築
3. 実験の実行
4. 実験結果の収集

2 の、実験環境の構築については 4.4 節で議論する。3 と 4 については実験の実行支援として、4.5 節で議論する。

4.1　要素と試験環境

本節では要素の組み合わせを元に、段階的検証において想定する試験環境を定義する。各段階のことを stage と呼ぶ。

4.1.1　サービスのみの環境

サービスのみの環境とは、試験対象となるサービスのみの環境で、他の要素は存在していない環境とする。サービスの動作検証が、この環境で行われる。サービスのみという表現の定義として、サービスを提供するためのソフトウェア以外にも、クライアントソフトウェアなど、サービスの動作や試験に必要な要素も含む。本論文では、サービスのみの環境で行う試験段階を pure stage と呼ぶ。

- 最小台数での試験

  最小台数の環境とは、サービスの動作に最低限必要な要素のみを用意した環境とする。この環境では、サービスの動作確認が行われる。また、後に大規模な試験を予

定している場合は、実験を支援するための工夫が必要となる。そのため、大規模な試験を見越して、実験支援ソフトウェアを用いた場合の動作確認等もこの段階で行う。実験支援ソフトウェアなどの技術については、5 章で説明。

- 台数を増やしての試験

  負荷試験や拡張性の試験などでは、多数の計算機を使用することになる。このような試験を行う場合は、実験支援ソフトウェアなど、多数の計算機を操作するための工夫が必要不可欠となる。実験支援ソフトウェアを用いるメリットとして、操作性の向上や、作業コストの軽減以外にも、実験の実行を自動化することにより、操作ミスを防ぐことや、実験の再現性を確保することに繋がる。

  本論文では、具体的な台数は定義しないが、実験を実行・操作するために、専用の仕組みを用いるかどうかを基準とする。宮地らの研究 [11] では、10 台程度とされている。

4.1.2 周辺要素を再現した環境

インターネットには、開発対象のサービス以外の要素が数多く存在する。本論文では、サービスの構成要素に含まない要素は**周辺要素**と呼ぶ。

サービスのみの環境での試験結果と比較することにより、他の要素による影響を確認できる。本論文では、周辺要素を再現した環境で試験を行う段階を、emulated stage と呼ぶ。周辺要素の例を、以下に挙げる。

- リンク特性

  遅延・帯域・パケットロスなど、通信を行うリンクが持つ要素が挙げられる。本論文では、これらの要素を総称してリンク特性と呼ぶ。これらの要素を再現する方法については、既存研究がある。遅延や帯域といったリンク特性の再現については、Netem[12] や Dummynet[13] といった技術がある。また、それらを用いたネットワークの模倣に関しては明石らの研究 [14] がある。

- 背景トラフィック

  インターネット上では、対象のサービス以外のサービスもあり、それらの通信によるトラフィックが流れている。このような、対象サービスと直接関係の無いトラフィックを、背景トラフィックと呼ぶ。背景トラフィックの再現ついては梅木らの研究 [15] がある。またトラフィックを生成する装置としては SmartBit がある。

- ネットワークトポロジ

  トポロジとは、ノードの接続形態を意味する。ここでのネットワークトポロジとは、L3 のトポロジを指す。ネットワークトポロジの再現に関しては、AnyBed による模倣インターネットなどの研究がある。

- 他のサービス

  影響を与える可能性のあるサービスや、互換性のあるサービスがこれにあたる。例えばDNSであれば、サーバプログラムとして異なる実装でも、上位や下位のDNSとして互換性がある。

本論文では周辺要素そのものの再現方法については提案しない。既存の技術によって再現された周辺要素を柔軟に追加・変更するための方法を考える。

4.1.3 インターネットに接続された環境

周辺要素を再現する場合、要素の再現性や、再現する内容の妥当性も考慮しなければならない。更にインターネット上の要素を全て再現することは困難であり、想定していなかった要素の影響を受ける可能性がある。このため、インターネット等の実環境上でも試験を行うべきであり、試験運用によってそのような影響を確認することができる。本論文では、インターネットに接続された環境で試験を行う段階をInternet stageと呼ぶ。また、インターネット上のホストであっても、接続されているネットワークによって状況は異なる。例えば背景トラフィックなどは、接続されているネットワークによって、異なるものが流れている。分散サービスにおいては、多くのネットワーク上での動作を確認できることが好ましい。

同一の環境で、インターネット上であるかどうかのみを切り替えることは難しい。このため、異なる計算機へ移行させる必要がある。

4.2 各試験環境に求められる特性

4.2.1 pure stageに求められる特性

pure stageを実現するためには、対象となるサービスが動作する環境を構築できれば良い。ただし、pure stageの場合でも、数百という規模になれば実験が困難になる。そのため、大規模実験を支援するためのソフトウェアなどが必要になる。また支援ソフトウェアには、emulated stageやInternet stageでも使用できるように、汎用性が求められる。周辺要素など、サービス以外の要素が無い環境であることが求められる。またノードの性能が均一であることが望ましい。例えば性能の異なるノードが混在している環境で、負荷が原因で問題が発生した場合、性能の問題なのか、負荷が集中したためなのかを判断しにくいためである。

4.2.2 emulated stage に求められる特性

emulated stage を実現するためには、周辺要素を再現する必要がある。周辺要素の再現は、既存の技術で実現が可能である。そのため本論文での要件としては、これらの技術で再現した周辺要素を、構築する実験環境に取り入れられることが挙げられる。

4.2.3 Internet stage に求められる特性

Internet stage を実現するためには、インターネットに接続されたノードで実験できれば、要件を満たせる。ただし試験運用を行うためには、利用者から接続できる必要がある。しかしインターネットに接続されているノードを、実験的な用途に使用できるケースは少ない。また、利用できるとしても台数が限られる。Internet stage では、実際のインターネットの影響下での試験が行われる。接続されているネットワークによって、状況が異なるため、多様なネットワークが使用できることが求められる。

4.3 移行に伴う作業

本節では、各 stage の移行に必要な作業を定義する。また、移行に伴うアプリケーションの設定や、実験内容の変更作業についても述べる。

4.3.1 周辺要素の追加・削除

pure stage と emulated stage の違いは、周辺要素の再現を行うかどうかである。つまり、pure stage に周辺要素を追加すれば emulated stage になる。同様に、emulated stage から周辺要素を取り除けば pure stage になる。

4.3.2 インターネットとの接続・切断

pure stage、emulated stage と、Internet stage との違いは、インターネット上のノードであるかどうかである。このため、インターネットへ接続すれば Internet stage へ移行でき、切断後、実験用のネットワークに接続すれば pure stage へ移行できる。そして周辺要素を再現した実験用ネットワークに接続すれば emulated stage への移行になる。これらを踏まえた stage の遷移を図 4.1 に示す。

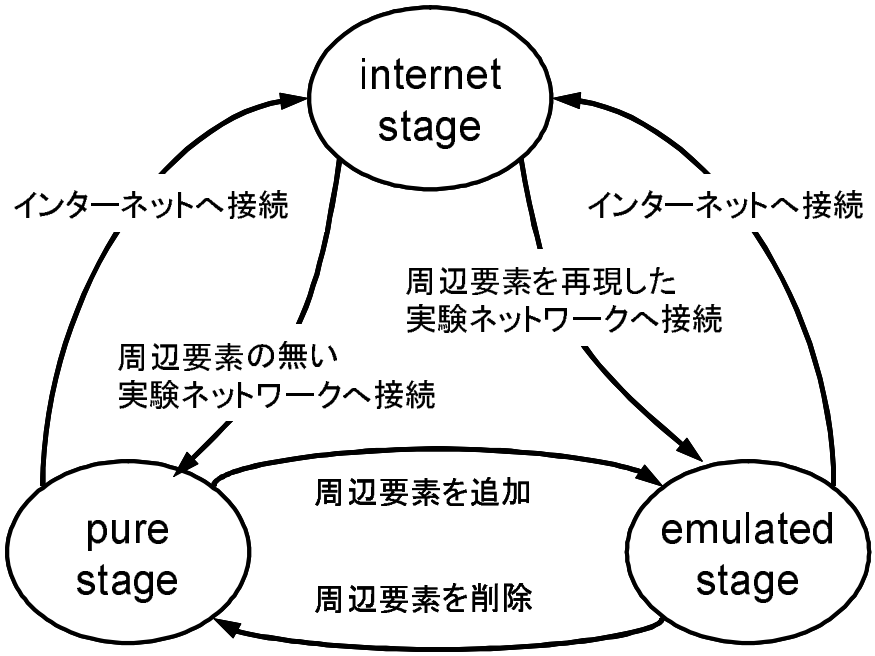

図 4.1: stage の遷移

4.3.3 アプリケーションの設定・実験内容の変更

試験環境の変化以外にも、試験環境を変更することによる、アプリケーションの設定変更や、実験支援ソフトウェアの設定変更、実験内容の記述変更が必要となる。特にテストベッド単位で移行した場合、管理用のサーバのアドレスなど、テストベッドを利用するために必要な設定も変更する必要がある。

異なるネットワークに接続した場合は、経路制御の都合上、IP アドレスを変更する必要がある。実験内容として、各ノードが他のノードの IP アドレスを参照して通信を行う場合等は、合わせて変更を行う必要がある。このようなアドレスの変化は、ホストの参照時に FQDN で指定するなどの方法で吸収できる。

4.4 実験環境の構築

4.4.1 複数のテストベッドの併用

提案手法を実現する方法として、既存のテストベッドを使用する。既存のテストベッドでは、いくつかの検証段階には対応できる。しかし全ての段階には対応しているテストベッドはない。これは、インターネット上のノードであるか、閉じた環境上のノードであるかのように、両立のできない特性があるためである。このため本研究では、複数のテストベッドを使い分けることを前提とする。

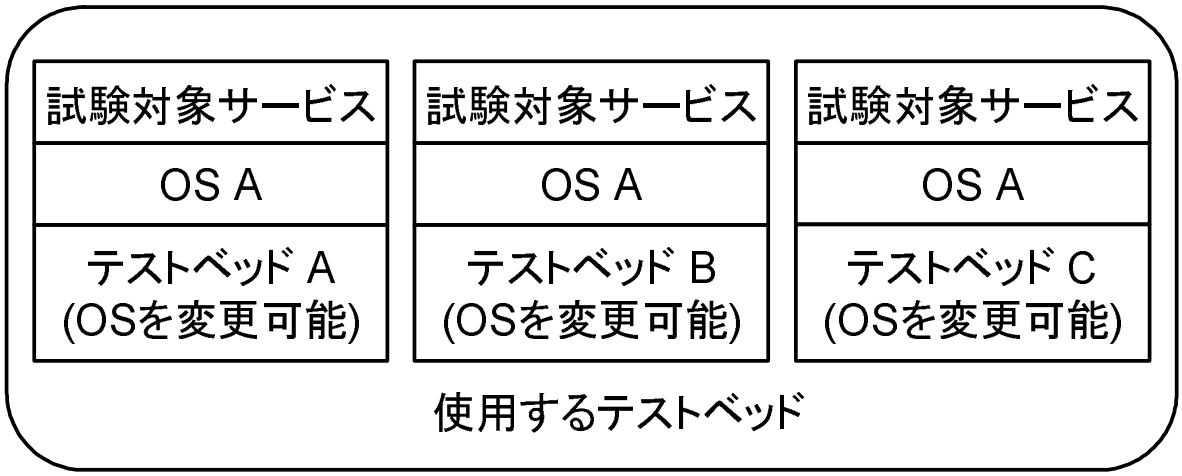

図 4.2: OS を変更可能なテストベッドのみでの構成

4.4.2 実験用ノードの OS の統一

本研究では、段階的検証において複数のテストベッドを使い分けることを前提としている。テストベッドによって使用する OS が異なる場合は、試験対象のサービスが動作しない、あるいは OS に合わせて変更が必要になる可能性がある。このため各テストベッドにおいて使用する OS を、本運用で想定している OS に統一できることが好ましい。

テストベッドと OS の組み合わせとして、OS を変更可能なテストベッドのみでの構成が、最も自由度が高いと言える。概念図を図 4.2 に示す。図 4.2 では、OS を変更可能なテストベッドのみを使用し、各テストベッドで OS A をインストールした場合の概念図である。この組み合わせであれば、テストベッド間の OS を統一することができる。

OS を自由に変更できないテストベッドが混在していても、他のテストベッドでその OS を使用できれば、各段階で同じ OS が使用できる。また、同じ OS を使用しているテストベッド同士であれば組み合わせられる。概念図を図 4.3 に示す。図 4.3 では、テストベッド B の OS は変更できないが、使用している OS B はテストベッド A にインストールができる。またテストベッド C も OS を変更することはできないが、同じ OS である OS B を使用しているため、混在していても同じ OS が使用できる。このため本研究では、テストベッドを組み合わせる基準の一つとして、OS が統一できるかを検討する。

4.4.3 OS のメディアレスインストール

実験の準備として、実験用ノードへの OS のインストールが必要になる。特に前述の理由により、テストベッド間で OS を統一する場合、OS を変更できるテストベッドではインストール作業を行うことになる。

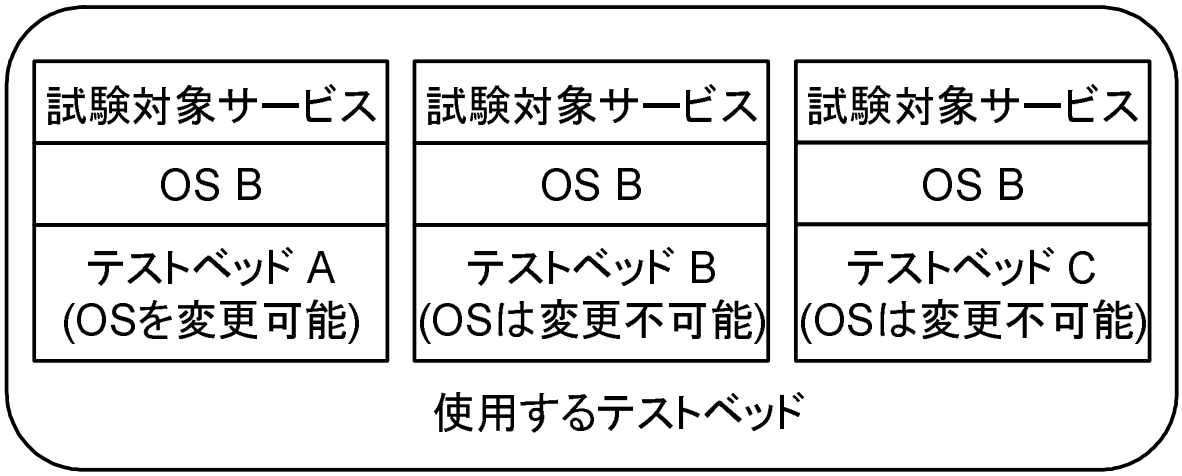

図 4.3: OS を変更できないテストベッドを含めた構成

多数のノードへ OS をインストールする場合、台数に比例して作業量が増えてしまう。一般的に OS をインストールする際には、CD-ROM 等のインストールメディアを用意する必要がある。またインストールには、HDD のフォーマットなど時間のかかる作業が含まれている。フォーマット中などは実験者が操作を行わなくても良いため、並列してインストール作業を進めることはできる。しかし、その場合は多数のインストールメディアを用意する必要がある。インストールメディアの作成やドライブへの挿入などの作業を、ソフトウェアから行うことはほぼ不可能であり、自動化は見込めない。よって、インストールメディアを用いずにインストールする方法が必要となる。

4.4.4 実験環境の管理

構築した実験環境の状態を、実験者が確認できることが望ましい。確認したい情報としては、IP アドレスや現在接続されているネットワーク等が挙げられる。

4.4.5 実験環境の拡張

段階的検証においては、次の段階へ移行するため、実験環境の拡張が考えられる。実験環境を構築するシステムを使用する場合、大きな環境への移行方法として、以下の二つが考えられる。

- 前段階の実験環境を破棄し、大きな実験環境を新しく構築する

  実験環境を毎回構築しなおす方式の概念図を図 4.4 に示す。この方法では、前の段階で構築した実験環境を一度破棄する。このため、前の段階において、実験者が加えた変更が保存されない欠点がある。実験支援ソフトウェアを用いることで、それら

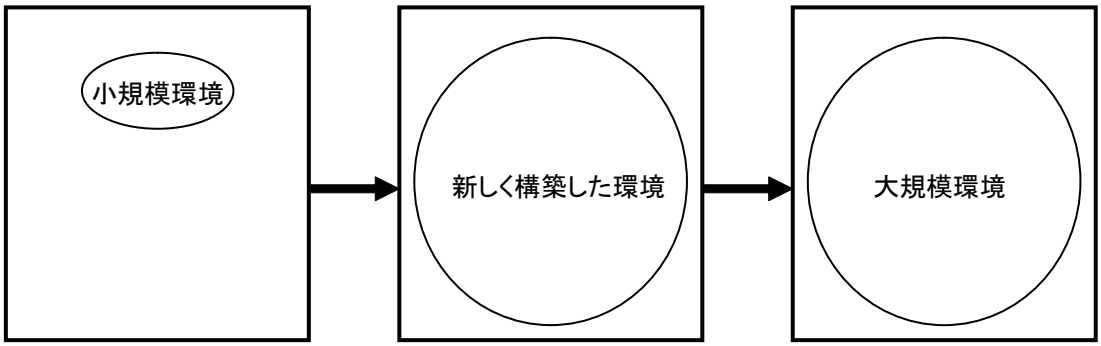

図 4.4: 実験環境を新たに構築する方法

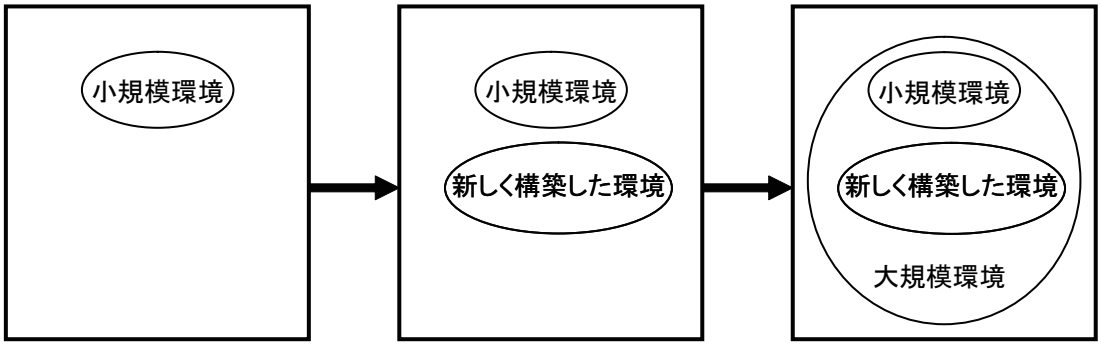

図 4.5: 実験環境を追加で構築する方法

の変更も再利用が可能になる。しかし実験者が全ての操作を、実験支援ソフトウェアや、環境構築ソフトウェアで行わなければならず、自由度が低下する。

- 実験環境を構築して、前段階の実験環境と結合する

  実験環境を追加で構築し、現在の環境と結合する方法の概念図を図 4.5 に示す。この方法では、小規模環境がそのまま残るため、実験支援ソフトウェアなどを用いずに行った変更もそのまま残る。実験者が手作業で変更を加えることがあるため、小さな実験環境からの移行では、こちらの方法が好ましい。既に作った実験環境を識別し、新しく構築した環境と結合する方法が必要となる。あるいは、既にある小規模環境に結合するように、新しい環境を構築する必要がある。またこの方法は、pure stage から emulated stage への移行にも適用できる。図 4.6 に示すように、新たに構築した周辺要素と pure stage を結合することで、emulated stage を実現できる。

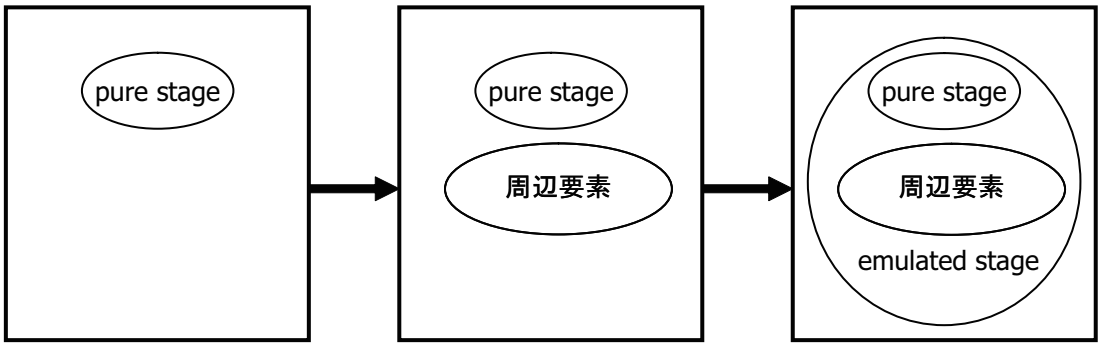

図 4.6: 周辺要素の導入

4.4.6 実験環境の初期化

共有して使用するテストベッドの場合、実験終了後、ログの収集をしてから資源を解放し、初期の状態に戻す作業がある。大規模な環境を構築した場合は、初期化のための作業にも大きく時間を費やすことになる。このため、実験後、元の状態に戻す機構が必要になる。また、何か問題があって実験をやり直すために、初期の状態に戻す用途も考えられる。このような、実験環境の保存・復元に関しては野中らの研究 [16] がある。

4.4.7 実験環境の保持

前述のように、実験後には実験環境を元の状態に戻す必要がある。実験後、初期化を行うように処理を自動的に実行することが考えられるが、本研究では採用しない。実験が全て意図通りに実行された場合は良いが、何かしらの問題で予想外の動作をした際には、問題のあった状態が残っていなければ、原因を解析できなくなるためである。

4.4.8 時刻の同期

各実験ノードの、時刻の同期も必要である。例えば実験のログを収集した際に、時刻がずれていた場合は、実験結果の分析に支障が出るためである。例えば、図 4.7 のように、10:00 に全てのノードでイベントが発生した場合には、各ノードの間で時刻がずれていると、解釈が変わってしまう。ノード B の時刻が 1 時間進んでいて、ノード C の時刻は 1 時間遅れていると仮定する。その場合は図 4.8 のように、異なるタイミングで発生したイベントであると解釈されてしまう。

分散サービスのように、複数のノードにまたがってサービスを提供する場合は、特に時間の同期が必要になる。

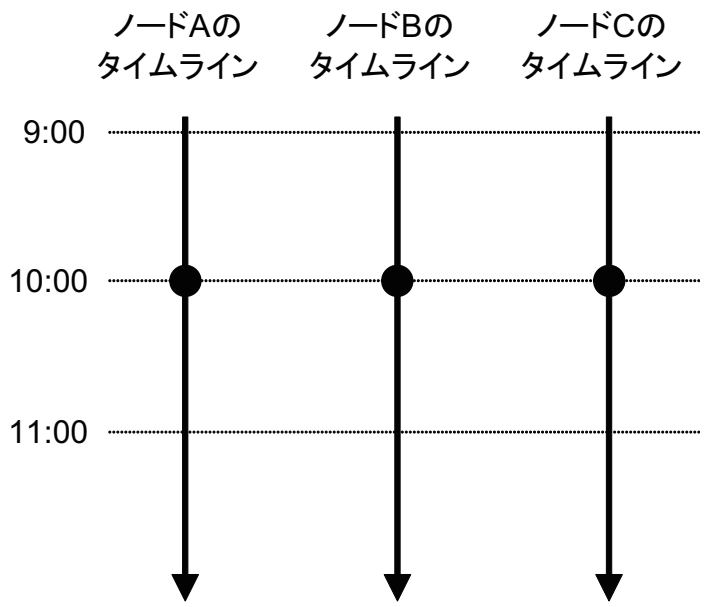

図 4.7: イベントのタイムライン

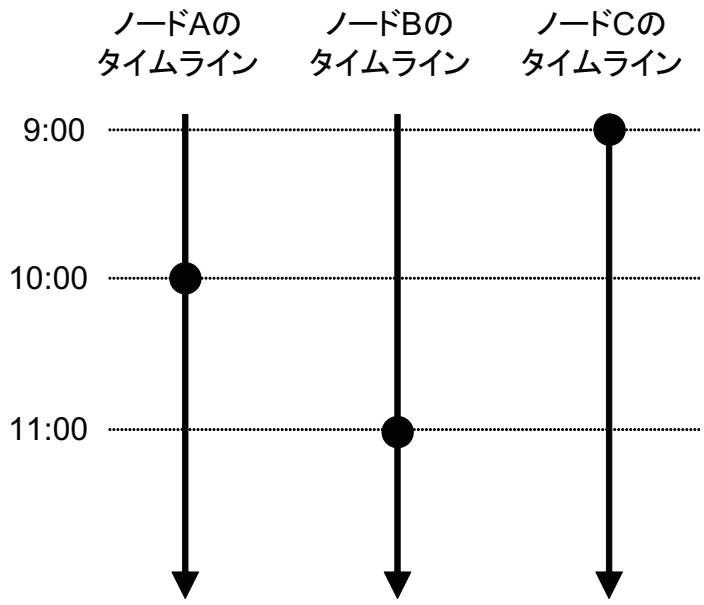

図 4.8: 時刻がずれていた場合の解釈

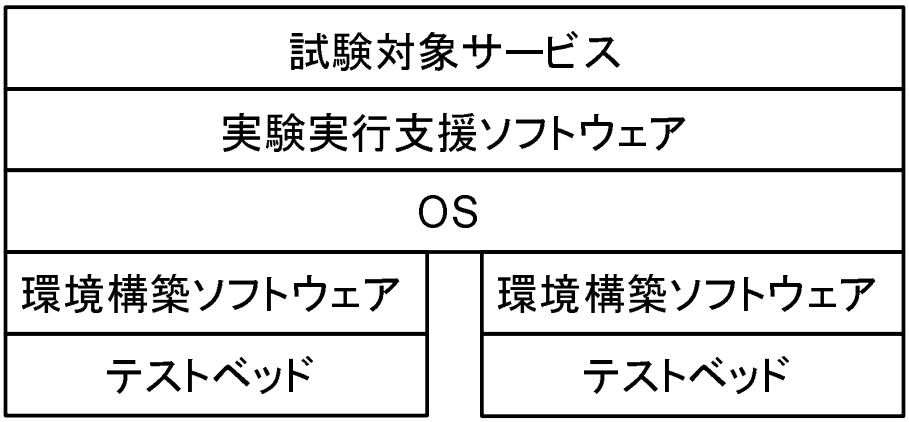

図 4.9: 本研究で提案するアーキテクチャ

4.4.9 試験対象サービスとの分離

これらの、実験環境の構築・維持に必要な機構が、試験対象のサービスに影響を与えないことが好ましい。環境構築のための仕組みは、テストベッドごとに異なるため、テストベッドに依存せざるを得ない。しかし実験の実行に関しては、どのテストベッドでも共通の仕組みを使えることが好ましい。そのため本研究では、環境構築のための仕組みと、実験の実行を支援するための仕組みを分離して設計する。

4.5 実験実行支援

本節では、実験の実行を支援するための枠組みの設計を行う。サービスの実装によって制御方法は異なる。特に、サービス独自のプロトコルなどは汎用的な仕組みから制御することは困難である。このため、実験者が対象サービスのために、制御の仕組みを作る必要がある。そのため本研究での設計の範囲では、実験実行の際に多数のノードをどのように扱うか程度に留める。本研究ではそのような支援を実験実行支援と呼ぶ。

コマンドの自動実行など、技術的には実験環境の構築と同じ技術で実現可能である。しかし本研究の設計では、実験環境を構築するための技術と、実験の実行を支援するための技術は違うものとする。これは前述の、試験対象サービスと実験環境を分離するためでもある。

実験実行支援ソフトウェアを含めた構成図を図 4.9 に示す。

4.5.1 実験の記述と実行

大規模な実験においては、１つ１つのノードを操作することは困難になる。同じコマンドを複数のノード上で実行させる程度でも、実験実行の効率は大きく向上する。実験内容

を予め記述し、内容に沿って実行する機構があれば、以下のようなメリットがある。

- 実験内容の再利用

  実験においては、同じ内容の試験を何度も行うことが考えられる。例えば測定のために、何度か同じ実験をして平均値や分布を求めたり、値を変更してどの程度の差が見られるかを調べるなどである。このため、実験内容を再利用できることが求められる。実験内容を予めファイルに記述する方式であれば、そのような再利用も容易になる。

- 時間的コストの軽減

  実験に伴うコストとして、時間的コストも挙げられる。特に段階的検証においては、多くの試験が行われるため、１つ１つの実験にかかる時間は無視できないものになる。ただし実験そのものに時間がかかる場合があり、実験の所要時間を大幅に短縮することは難しい。例えば、トラフィックの計測であれば、実際に想定する時間だけ通信を行わなければならない。このため、実験自体の所要時間ではなく、実験者の作業時間を対象に軽減を目指す。実験内容を手順化・自動化しておけば、実験者は実験終了まで待つだけで良く、その間に他の作業を行える。

4.5.2 サービス独自のプロトコルの制御

サービス独自のプロトコルを扱うためには、そのサービスに対応するサーバやクライアントのソフトウェアが必要になるが、それらが汎用的な仕組みから制御できるとは限らない。しかし、サーバやクライアントが送信するメッセージと、同じ形式のメッセージを扱うことができれば、サービス独自のプロトコルを扱うことができる。

4.5.3 ログの収集

分散サービスの場合、実験結果となるログは各ノード上に出力される。ログの解析などを行う場合には、図 4.10 のように各ノード上にあるログファイルを 1 つのノード上に集める必要がある。台数が増えると、ログの収集による作業量も増えてしまう。ネットワークを経由して、ログを収集する方法もあるがネットワークに問題が発生した場合は、ログに漏れが発生する。また大規模な環境では、収集するノードへの負荷の懸念される。収集するノードを複数用意して、負荷分散を行うことが考えられるが、その場合はログを収集したノードにあるログを、更に収集しなければならない。このため本研究では、各ノード上にファイルとして存在しているログを、制御用ノードが収集する方法を採用する。

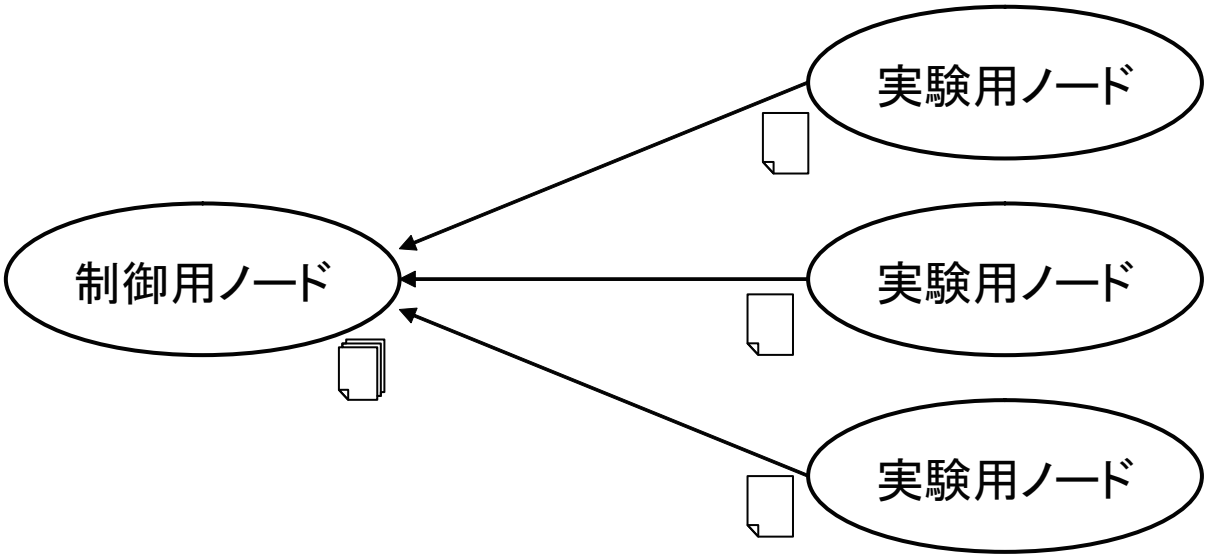

図 4.10: ログの収集

4.5.4 ログへの時刻の挿入

複数のノードからログを収集した場合、時刻を基準に比較する。しかしログに時刻が含まれるかは、サービスの実装に依存する。このため本研究では、ログに時刻を挿入する仕組みを提案する。

4.5.5 グループ化による制御

一回の実行で、複数のノードへ同じ命令を送信できれば、多数のノードを扱う実験も効率良く行える。しかし実験によっては、全てのノードで同じ動作を行わせる必要はない。例えばサーバとクライアントのように、いくつかの役割に分かれて動作することが考えられる。図 4.11 のように、サーバとして制御するノード群、クライアントとして制御するノード群のようにグループに分け、それぞれに対して同じ命令を送信して実験の制御を行う。このため、ノードをグループに分けて、グループ毎に制御できる機構が必要になる。

4.6 ネットワーク接続の変更

emulated stage および Internet stage への移行方法として、ネットワーク接続の変更を提案する。概念図を図 4.12 に示す。それぞれの stage に対して、対応するネットワークに接続させることによって、stage の移行を実現できる。

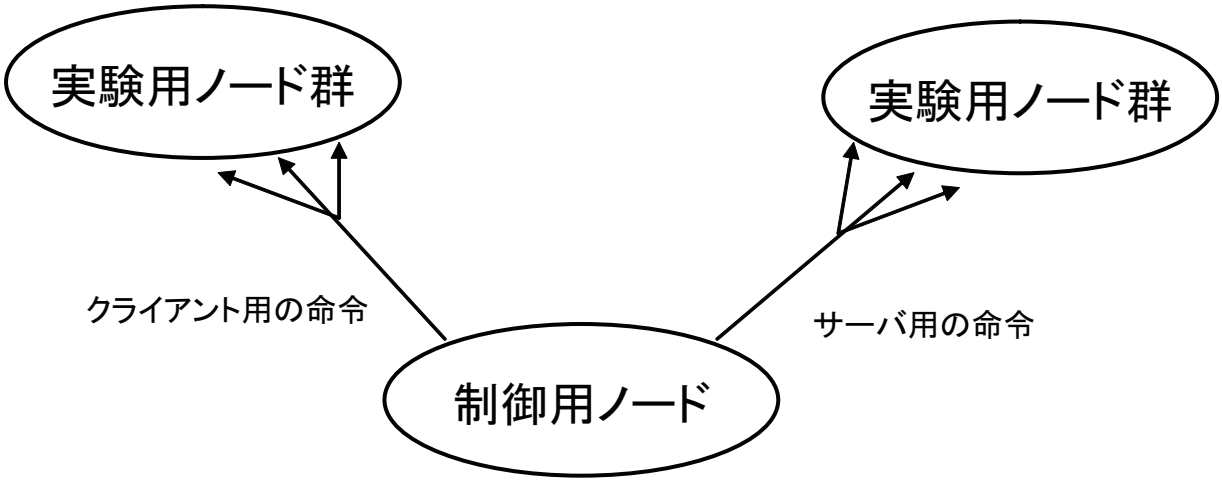

図 4.11: グループごとの制御

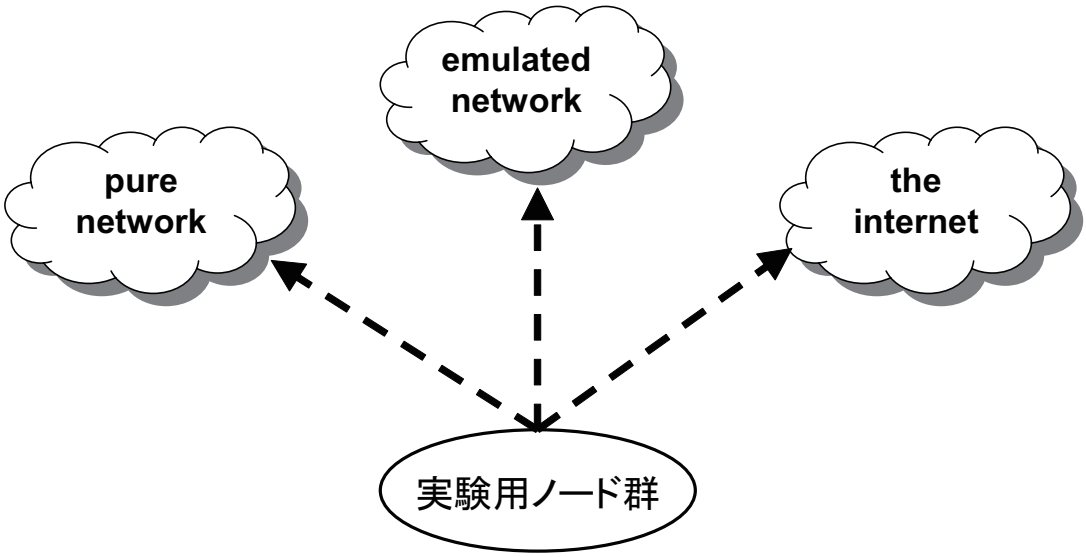

図 4.12: ネットワークの接続変更による stage 移行

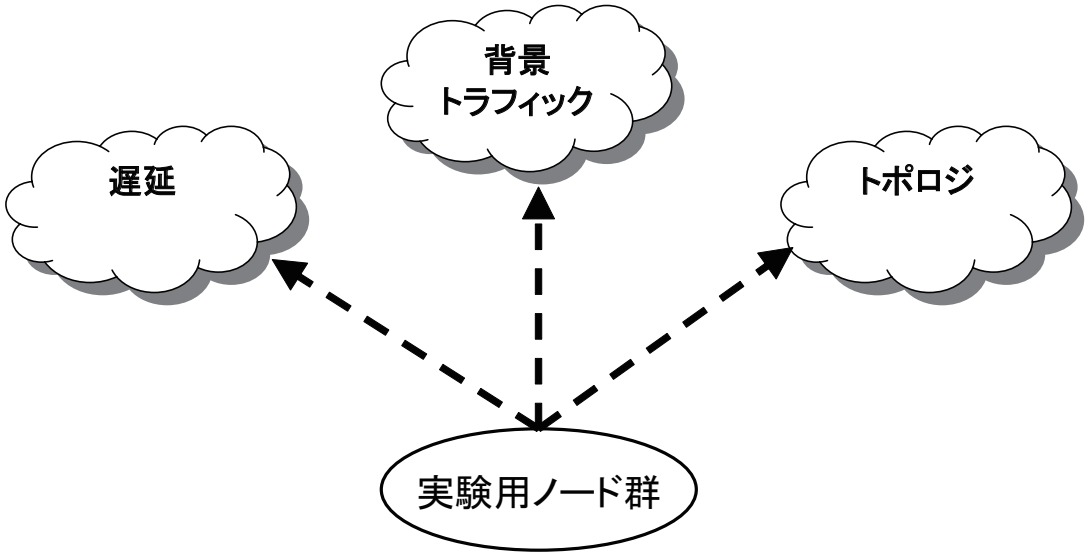

図 4.13: ネットワークの接続変更による周辺要素の導入

4.6.1 周辺要素を模倣したネットワークへの接続

emulated stage を実現するためには、周辺要素を再現し、実験環境に取り入れられる必要がある。周辺要素の再現については 4.1.2 節で説明したように、既存の技術で実現できるため、本研究では再現方法については提案しない。

図 4.13 のように、周辺要素を再現したネットワークに、実験対象のサービスが動作しているノードを接続させることで、実験環境に、周辺要素を取り入れることができる。このため、特に実験環境を分離しなければ、協調は容易である。

千装らの研究 [17] では、VLAN を組み合わせることで、複数の機能を導入していた。これを周辺要素の再現にも適用する方法が考えられるが、本研究ではその手法は使わない。この方法では、追加した要素が相互に影響を受けるとは限らないためである。例えば、遅延と背景トラフィックを再現する場合を考える。この場合は、図 4.14 のようになり、遅延のあるネットワークと、背景トラフィックが流れているネットワークがそれぞれ別に構築される。

本来のインターネットでは、背景トラフィックが遅延の影響を受けているはずであり、図 4.15 のようになる。

トポロジの再現も合わせた場合は、更に再現性が落ちることになる。遅延網を、トポロジの一部に接続させるだけでは、局所的にしか遅延が発生しない。

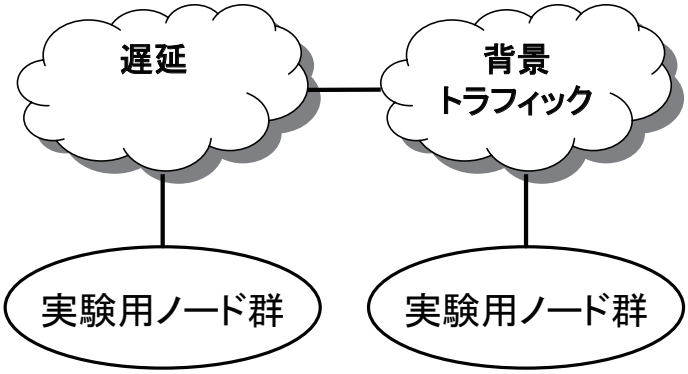

図 4.14: 遅延と背景トラフィックを別々に再現

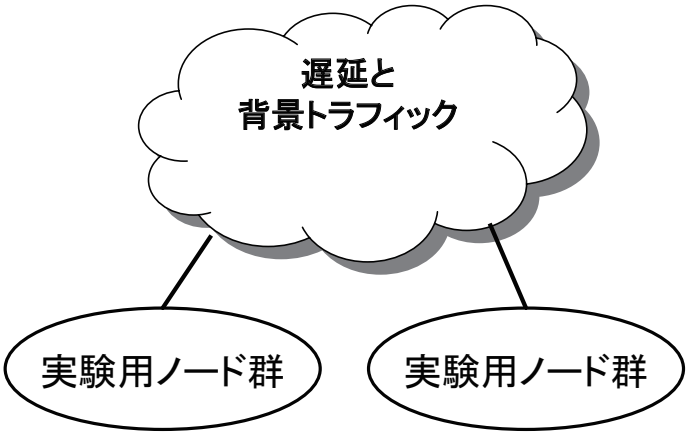

図 4.15: 遅延と背景トラフィックを同じネットワークで再現

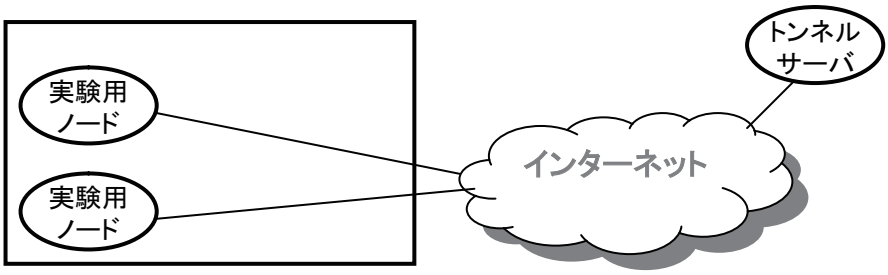

図 4.16: 各ノードが接続する方法

4.6.2 インターネットへの接続

インターネットへ接続することで、Internet stage を実現できる。しかし実験用ノードを、インターネットと繋がったネットワークへ接続変更することは、作業コストが高く、現実的ではない。そこで、本研究ではトンネル接続による、インターネットとの接続を提案する。施設のネットワークによる接続だけでは、接続できるネットワークが限られる。しかしトンネル接続であれば、接続先のネットワークを柔軟に変更できる。このため、分散配置も容易になる。またテストベッド施設がプロバイダなどと契約しておらず、利用できないネットワークであっても、実験者がプロバイダと契約していれば利用できる。更に運用を想定しているネットワークに接続させることで、実際に運用した時に受ける影響を見ることができる。ただし遅延等については、想定する環境と同一にはできない。実際のネットワークとは異なり、トンネル接続先のネットワークから、試験用の環境までの遅延が加算されるためである。また、その間のパケットロスや背景トラフィックの影響も受ける。

トンネル接続の方式としては、以下の方法が考えられる。

- 各ノードで接続させる方法

  概念図を図 4.16 に示す。各ノードがトンネル接続プロトコルに対応している必要がある。

- ブリッジノードが接続する方法

  1. ブリッジノードを用意し、トンネル接続を行う。
  2. ブリッジノード上でブリッジを構成する。
  3. 実験用ノードがブリッジ接続されたネットワークに接続する。

  概念図を図 4.17 に示す。実験用ノードが対応している必要はなく、接続するネットワークを変更することで実現できる。接続に必要な情報は、代表ノードにだけ持たせれば良い。クライアントが固定のグローバル IP アドレスを指定するプロトコルに

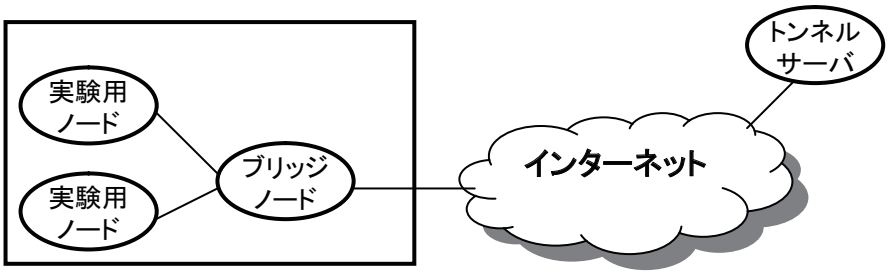

図 4.17: ブリッジノードが接続する方法

も適用できる。デメリットとして、実現可能なトンネルプロトコルが限られる点が挙げられる。また多数のノードが接続する場合は、負荷分散も視野に入れる必要がある。

使用するトンネルプロトコルは、以下の要件を満たすことが望ましい。

- NAT を経由しての接続が可能であること

  インターネットとの接続には、グローバルな IP アドレスが必要になる。しかしグローバルな IP アドレスには限りがあり、実験者が自由に使用できるとは限らない。特に本研究で想定する用途にグローバルな IP アドレスを使う場合、常時接続ではなく、実験時にのみ使用するアドレスとなる。閉じたネットワークでの試験を想定しているテストベッドで、そのような用途のグローバル IP アドレスを用意しているとは限らない。このため、NAT を経由しても接続可能なプロトコルが求められる。NAT であればグローバルな IP アドレスの数は、実際に接続するノードの数より少なくて良い。またテストベッド施設の制約上、NAT でしか外部と接続が行えないことが考えられる。

- 想定する OS で動作すること

  本研究では、使用する OS は、使用するテストベッドの組み合わせに依存する。このため、テストベッドで使用されている OS に対応していることが望ましい。

- ノードがグローバルな IP アドレスを取得できること

  外部と通信をするだけであれば、NAT 等でも実現はできる。しかし NAT の場合は、外部からコネクションを張ることができず、試験運用などが実現できなくなってしまう。また、サービスの実装によっては自身の IP アドレスを参照するため、NAT 等で外部からみたアドレスと内部からみたアドレスが一致していなければ、問題を起こす可能性がある。このため、ノード自身にグローバルな IP アドレスが割り当てられるプロトコルを使用する。

- IPv4 以外のプロトコルにも対応していること

  インターネットは IP アドレスの枯渇等の問題から、IPv6 への移行が考えられている。トンネルプロトコルが IPv6 に対応していれば、IPv6 でのサービス提供や、IPv6 対応版の検証などの用途にも使用できる。また、トンネル接続サービスが IPv6 でのみ提供されている場合等にも対応できる。

- トンネルサーバの設定が固定的であること

  どのノードが接続するかは不確定であるため、トンネルサーバの設定として、クライアントの IP アドレスを指定しなくて良いプロトコルでなければならない。

4.7 テストベッド間の協調

本研究では、複数のテストベッドを使用することを前提としている。OS や実験実行支援ソフトウェアを統一することにより、円滑な移行が見込める。しかし、それでも別々のテストベッドを使用しているため、グループ化などの管理はそれぞれのテストベッドに委ねられる。このため、テストベッドを移行した際には、グループを再度定義するなどの移行作業も必要になる。

図 4.18 のように、複数のテストベッドのノードを合わせて、一つのテストベッドとして扱うことができれば、実験環境の管理情報などを共有できる。どのテストベッドのノードであるかを意識せず、透過的に使用できるようになり、利便性が向上する。また使用可能なノードの数が増えるため、より大規模な実験が可能になる。

4.8 まとめ

本章では、段階的検証のためのテストベッドアーキテクチャを提案した。実現のために必要なテストベッドや、時刻の同期方法等については既存技術も適用可能である。5 章では、実現にあたり既存技術の使用を検討する。既存技術だけでは適用できず、本研究で実装した部分については、6 章で述べる。

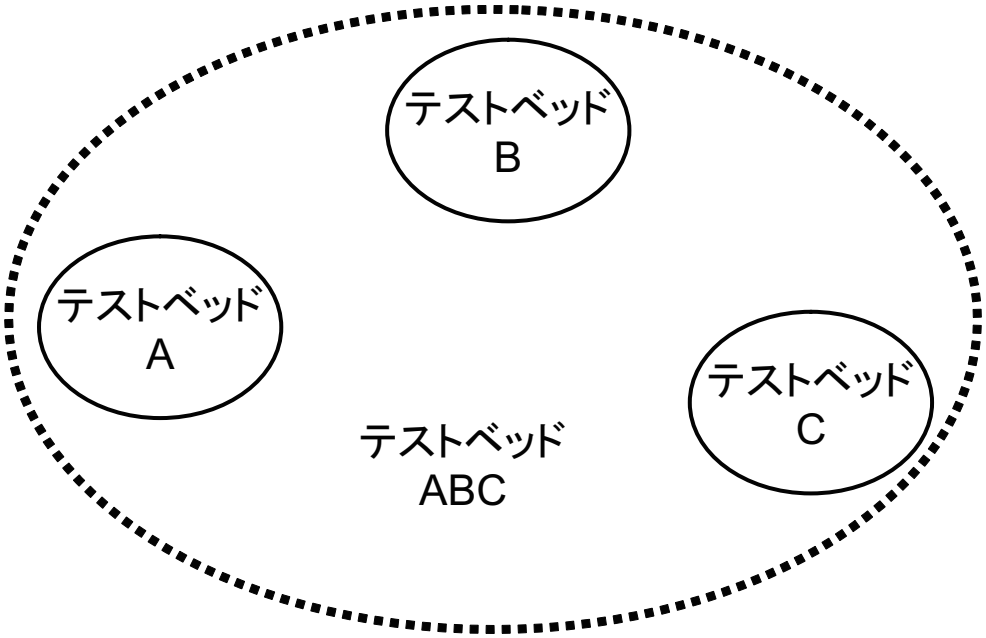

図 4.18: テストベッド間の協調

第5章　既存技術の使用の検討

4章では、段階的検証のためのテストベッドアーキテクチャを提案した。本章では、実現のために既存技術の使用の検討を行う。

5.1　テストベッドの選択

5.1.1　PlanetLab

PlanetLab はインターネットを直接利用していることから、Internet stage に適したテストベッドといえる。構築に必要なソフトウェアが MyPLC として公開されているため、pure stage や emulated stage でも PlanetLab を構築することができれば、OS や実験実行支援ソフトウェアを統一することができる。しかし構築にインストールメディアが必要な点などから、大規模な Local PlanetLab を構築することは困難である。

Slice の概念により、テストベッドの運用に必要な機能と、実験者に提供される環境が分離されている。更に Federation という機能があり、PlanetLab Europa などの亜種と協調している。これにより、テストベッド間の協調も実現できる。これらの機能については後述する。

5.1.2　StarBED

StarBED は実験者がノードを占有して利用できる。このため、安定したノードが利用でき、OS なども変更が可能である。ネットワークについても、VLAN で分けられているため、外乱の無いネットワークが使用可能である。このため、pure stage に適したテストベッドといえる。実験環境において、周辺要素を再現するための研究には、StarBED を対象にしている研究がいくつかある。よってそれらの技術を適用することで、emulated stage も実現可能である。

5.1.3　本研究で使用するテストベッド

PlanetLab は多くの組織からノードが提供されており、Internet stage を実現するテストベッドに適している。PlanetLab への移行を見越し、pure stage には Local PlanetLab

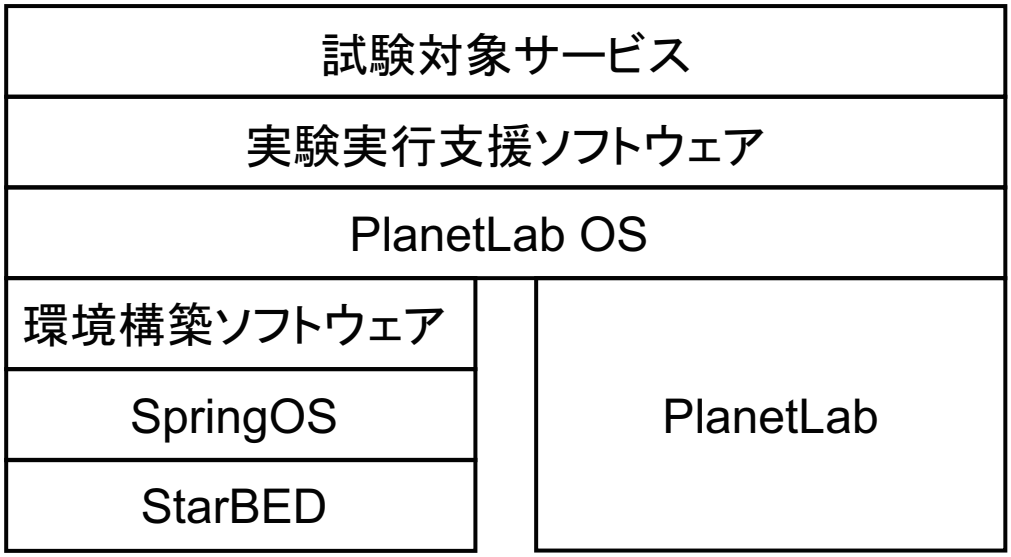

図 5.1: StarBED と PlanetLab による構成

を使用する。これにより同一の OS と実験実行支援ソフトウェアが使用可能となる。Local PlanetLab の構築には StarBED を用いた。これにより、大規模な Local PlanetLab の構築が可能になる。更に、StarBED を対象とした周辺要素の模倣技術を使用できるため、emulated stage を実現することができる。概念図を図 5.1 に示す。

5.2 実行実験支援ソフトウェア

5.2.1 CoDeploy

PlanetLab に適した実験支援ソフトウェアとして CoDeploy がある。CoDeploy には multicopy、multiquery というプログラムが含まれており、multicopy は複数のノードに対して同じファイルをコピーし、multiquery は図 5.2 のように、複数の実験用ノード上で同じコマンドを実行させる。SCP、SSH を用いて実装されているため、PlanetLab のように、同一の公開鍵が設置されているノード群であれば、同じようにファイル転送と操作が行える。ファイルの配布には CoDeeN を利用することもできる。その場合は codeploy というプログラムを用いる。

使用する Sliver の指定方法として、環境変数を用いる。環境変数として、Slice の名前と、ノードのリストを記述したファイルを指定する必要がある。このため、Slice 名が異なる Sliver が含まれている場合は、それらを同時に操作することはできない。

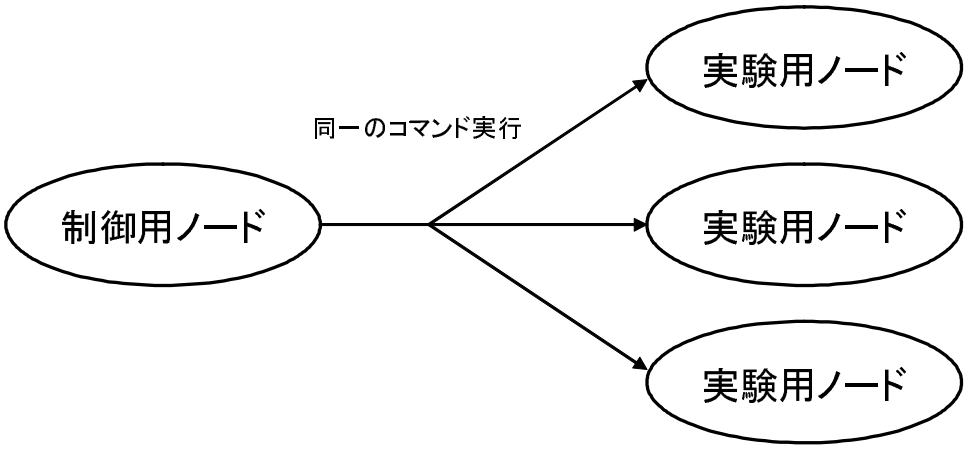

図 5.2: 複数のノードへ同じ命令を送信

5.2.2 pssh

CoDeploy に含まれるプログラムのように、複数台のノードに対して、SSH を用いて制御を行うプログラムとして pssh がある。ファイルの配布についても、pscp というプログラムを用いて行うことができる。CoDeploy とは実装上の違いがいくつかあり、ノードやユーザ名の指定方法が異なる。pssh では、ノードのリストをファイルとして用意し指定する。このリストファイル中で、ノードごとにユーザ名を指定できるため、CoDeploy と違って複数の Slice を使用している場合でも同時に操作が行える。

5.3 実験環境の構築

5.3.1 OS のメディアレスインストール

多数の計算機への OS のインストール方法として、メディアを使用しない方法が必要となる。メディアレスインストールを実現するための方法として、Preboot eXecution Environment(以下 PXE とする) がある。PXE はネットワークブートの規格であり、BIOS の設定では CD-ROM や HDD からの起動と同様に、PXE による起動が選択できる。PXE はディスクレスシステムの起動にも使用されており、ネットワークを経由して、OS の起動に必要なファイルを取得することができる。

5.3.2 SpringOS

StarBED を対象とした、実験支援ソフトウェアとして SpringOS がある。SpringOS は単一のプログラムではなく、機能ごとに複数のプログラムに分かれている。

- ERM

  ERM は施設の資源を管理するためのソフトウェアであり、サーバとして動作する。クライアントが利用する際には、ERRP というプロトコルを使用する。管理している情報としては、下記のようになっており、ノードの名前、ネットワークインターフェイス、接続されている L2 スイッチのポートなどの情報を管理している。

```
node node001 [
    bootdisk,IDE
    health,ICMP,SSH
    power,SNMP-NECMIB
    net,manage,FastEthernet,xx:xx:xx:xx:xx:xx,"mgsw1:1/1",10.0.0.1,
"Linux""eth0""FreeBSD""fxp0"
    net,empty,FastEthernet,xx:xx:xx:xx:xx:xx,,,
"Linux""eth1""FreeBSD""fxp1"
    net,experiment,GigabitEthernet,xx:xx:xx:xx:xx:xx,"exsw1:1/1",,
"Linux""eth2""FreeBSD""fxp2"
]
```

  ERM にはユーザの概念があり、ユーザに対して割り当てるべき資源の管理も行っている。条件を指定して資源を探すことが可能であり、FIND というメッセージを用いることで実験者が使用可能な資源から、条件に合うものを検索できる。検索する条件としては、ディスクのタイプや、ネットワークインターフェイスの速度や数が挙げられる。また、ノードが実験で使用中であるかの管理が可能であり、既に使用している資源に対しての排他処理も行っている。排他処理では、HOLD という処理を行っており、FINDHOLD というメッセージを用いることで、条件に合う資源を HOLD できる。既に HOLD されているリソースは、別の機会の FINDHOLD によって HOLD されることはない。SpringOS の他のプログラムによって、情報を参照するために使用されている。

  ERM は施設の情報を管理しているが、実験者が構築・設定した環境については管理を行っていない。このため、本研究で提案する実験環境の管理については、別途行う必要がある。

- SWMG

  StarBED ではネットワークトポロジの変更には、物理的な配線作業を行わず、VLAN を用いて仮想的に行っている。そのために VLAN の変更を行うソフトウェアが SWMG である。通常、L2 スイッチは telnet 等を用いて操作が可能である。SWMG は telnet プロトコルを使用し、L2 スイッチに対してコマンドを送ることで、VLAN を変更する。SWMG へのリクエストには SWCP というプロトコルを使用する。対象になるポートは、予め ERM で HOLD されている必要があり、ERM と連動して使用する。

- DMAN

  DMAN は、ファイルのシンボリックリンクを操作するためのサーバである。クライアントからは DMP というプロトコルで使用する。StarBED では、DMAN を使用し、TFTP サーバ上にあるファイルのシンボリックリンクを操作している。StarBED では、全ての計算機は PXE を用いてネットワーク経由で起動する。起動後、TFTP サーバからブートローダを取得して、起動する OS を選択している。このブートローダを切り替えるために、シンボリックリンクを変更している。

  PXE のためにクライアントに渡されるファイルの名前は、DHCP サーバによって設定されている。変更するためには DHCP サーバの再起動が必要となるが、StarBED では DHCP サーバを利用者全体で共有しているため、影響が大きい。そこで、PXE で渡すファイルの名前を固定にし、シンボリックリンクの張り替えによって変更している。このメカニズムは PXE Linux など、設定ファイルの名前が決まっているシステムにも応用できる。

- wolagent

  電源の操作をソフトウェアで行うため、Wake On LAN[18] を使用してノードを起動する。電源が投入されていない状態の計算機には、IP アドレスが割り当てられていない。このため Wake On LAN では、宛先の指定にブロードキャストアドレスを指定する。ブロードキャストは通常、異なるセグメントに対して転送されない。このため Wake On LAN を送るホストと、対象の計算機は同一セグメント上に存在している必要がある。しかし実験者が操作する計算機が、対象の計算機と同じセグメントに存在しない場合でも、Wake On LAN による起動が行えるようにするための仕組みが wolagent である。IPMI であれば、起動と停止も可能であり、StarBED には IPMI に対応した機器も導入されている。しかし IPMI に対応していない機器もある。それらの機器の再起動や電源の停止には snmp[19] を使用する。実験用ノードには、snmp が入っていることが前提となる。

- pickup/wipeout

  StarBED での OS のインストールの支援として、dd を用いて既存のノードの HDD の内容をコピーする方法がある。このため、実験者が行う作業として、1 台のノードに目的の OS をインストールすれば、二台目以降は dd によるコピーを待つだけでよい。本論文では、dd によってノードから抽出したファイルをディスクイメージと呼ぶ。ディスクイメージを作成するためのプログラムは pickup、ディスクイメージの内容を書き込むためのプログラムは wipeout である。ディスクイメージの書き込みや作成の際には、NI というディスクレスの OS が起動する。wipeout は一回の実行で、複数のノードに対してディスクイメージの書き込みを行うことができる。このため、インストールするノードの数が増えても、実験者が行う作業としては wipeout を一回実行するだけでよい。ただし、インストールに伴う時間は軽減できない。

- kuroyuri

  実際に実験を実行するためのソフトウェアがkuroyuriという枠組みである。kuroyuriはシナリオ実行という概念を持っており、実験の内容をあらかじめファイルに記述しておき、kuroyuriの枠組みに含まれるプログラムを実行するときに読み込ませることで、記述した内容の実験を行う。シナリオの記述にはK言語という言語を使用する。kuroyuriにはシナリオ記述の解釈と制御を行うmasterというプログラムと、実験用ノードで実際に実験内容のコマンド等を実行するslaveというプログラムに分かれている。masterを実行するノードは、ENCDと呼ばれる。使用するノードの割り当てや、L2スイッチの設定などは、ERRPやSWCPを用いて行う。masterとslaveの間で通信を行いながら、実験を遂行することができる。masterとslaveの間の通信は管理用のネットワークを使用し、実際の実験には実験用のネットワークを使用する。このため、トラフィックの分離が行える。また、接続性の無い状況の試験も行える。実験用のネットワークからは、他のノードと接続されていなくても、管理用ネットワークさえ接続されていれば、制御を行えるためである。シナリオには、実験環境の構築も含めることができ、OSのインストールを含めてmasterから実行することができる。

  K言語の仕様では、実験内容をclassとして記述することができる。同じclassのノードを複数台構築し、実験を行わせることができる。このため、グループ化による制御も行うことができる。しかし、本研究では実験環境の構築と、実験の実行については分離することを提案している。kuroyuriでは両者が分離されていないため、本研究では使用しない。

- NI

  pickup/wipeoutにおいて、ddを実行するために用いられるディスクレスのOS。master/slaveでも、記述次第ではNIが起動してノードのインストールを行ってから、実験が行われる。

- fncp

  シナリオ実行の際に、slaveノードがmasterノードを探すのに使用する。これはディスクイメージのインストールを含めたシナリオ実行を行った場合に、新たにディスクイメージがインストールされたslaveノードは、masterノードのIPアドレス等の情報を取得できないためである。

5.3.3 時刻の同期

時刻の同期に関しては、NTP[20]がある。PlanetLabは、ノードが起動時にNTPを用いて時刻を同期する。このため、本研究でもこれを利用する。ただしNTPサーバについては、決められたものを使用する。このため、pure stageやemulated stageのように、外

部の NTP サーバを利用できない状況では、施設内のものを使用するように変更しなければならない。

5.4 トンネルプロトコル

5.4.1 L2TP

トンネリングプロトコルとして、本研究では L2TP[21] を採用した。以下、L2TP が本研究での要求と合致するかを述べる。

本研究での要求

- NAT を経由しての接続

  StarBED では JGN 等の回線を用いて外部と接続を行うことはできるが、グローバル IP アドレスには限りがあるため、同時に接続可能な数は限られる。NAT を用いることで、グローバル IP アドレスを節約することができる。

  L2TP は UDP を用いてセッションを確立し、仮想的な回線を設ける。L2TP サーバからは、IP アドレスとポート番号でクライアントを識別することになる。このため、NAT を経由してもトンネル接続を行うことができる。

- 想定する OS 上での動作

  本研究では、OS として PlanetLab ノードを想定している。現行のバージョンでは、PlanetLab ノードの OS は Fedora Core 8 をベースにしている。Fedora Core 8 用の PPP[22] の RPM パッケージと、rp-l2tp のパッケージを追加するだけで使用できる。

- IPv4 以外の L3 プロトコルへの対応

  L2TP は、仮想回線上で PPP を用いて設定を行う。このため PPP が対応していれば、IPv4 以外の L3 プロトコルを使用できる。PLNode は、IPv6 への対応が考えられており、IPv6 の広いアドレス空間を利用し、Sliver ごとにグローバルな IPv6 アドレスを割り当てる機構が提案されている。本研究で提案するトンネル接続でも、IPv6 に対応した PLNode が将来的に活用できるように、IPv6 が適用可能なトンネルプロトコルを採用した。

- 接続先のノードの設定が固定的であること

  本研究では、試験の段階に応じて必要なときにのみ、トンネル接続を行うことを想定している。このため、常時接続を想定しているプロトコルは適さない。接続時に、接続相手となるノードを操作せずに接続できることが好ましい。L2TP サーバの実装である xl2tpd というソフトウェアは、この条件を満たしている。

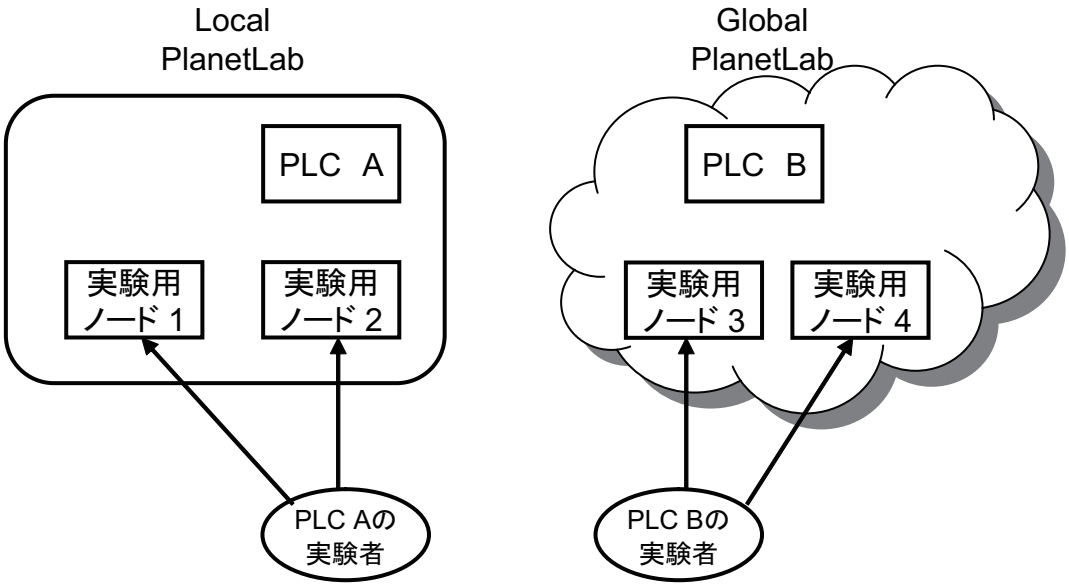

図 5.3: 所属している PLC のノードのみ利用可能

L2TP を使うデメリット

L2TP によって確立された仮想回線上では、接続したノードが送受信するトラフィックのみが流れる。このため、遠隔地のネットワークに接続したとしても、トラフィックの計測等の用途には使用できない。

5.5 テストベッド間の協調

PlanetLab には Federation という機能があり、これによって複数の PlanetLab を利用することができる。複数の PlanetLab の間で、所属しているノードや Slice の情報を共有することができる。Federation を結ぶ相手の PLC ホストは、Peer と呼ばれる。通常は、図 5.3 のように、実験者は自分の所属している PlanetLab の実験用ノードしか利用できない。しかし、Federation 機能を使うことで、PLC 同士が情報を交換し、図 5.4 のように、Peer に所属しているノードを利用することができるようになる。共有される情報としてはノード、サイト、Slice、Person が挙げられる。共有されない情報としては NodeNetwork や NodeTag が挙げられる。また、Federation によって共有されたノード等は、更に Federation を組んでいる他の PLC に対しては共有されることはない。例えば図 5.5 のような構成で Federation を組んだ場合、PlanetLab A の実験者からは PlanetLab A、B のノードが利用でき、PlanetLab B の実験者からは A、B、C 全てのノードが使用できる。そして B は C と Federation を組んでいるが、A のノードを C に提供することはないため、PlanetLab C の実験者は B、C のノードのみを利用できる。

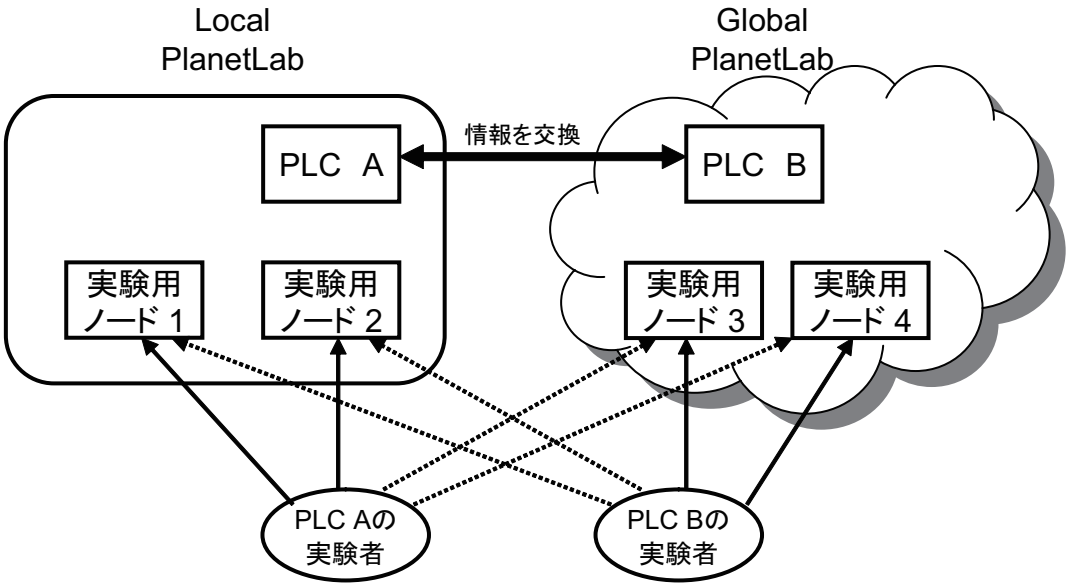

図 5.4: お互いの PLC に所属しているノードを利用可能

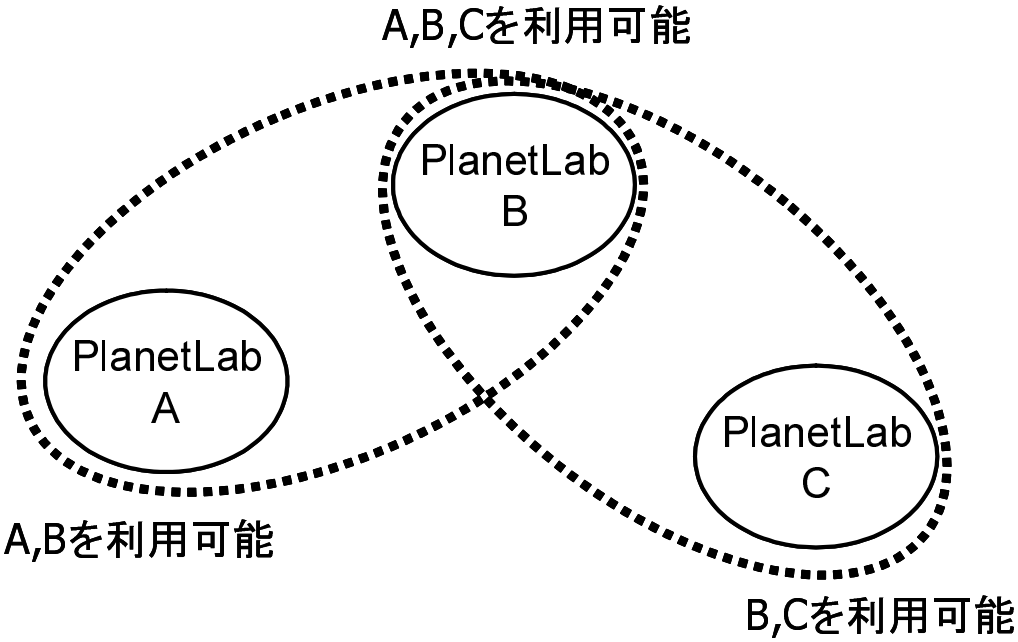

図 5.5: Federation によって共有される範囲

第6章　提案するアーキテクチャの適用例と実装

提案する枠組みの適用例として、本研究ではPlanetLabとStarBEDを選択した。StarBEDを対象とした環境構築の枠組みと、PlanetLabの機能を利用した実験実行支援・テストベッド間の協調について説明する。

6.1　Local PlanetLabの構築

本節ではまず、MyPLCを用いてLocal PlanetLabを構築する手順について説明する。次に大規模なLocal PlanetLab環境を構築する上での問題点を挙げ、本研究で実現した解決方法を説明する。

6.1.1　Local PlanetLabの通常の構築方法

MyPLCを用いたLocal PlanetLabの構築方法について説明する。

PLCホストの構築手順

PLCとしてサービスを提供するためには、いくつかの構成要素が必要となる。MyPLCはLinux系OSのRPMパッケージとして提供されており、これらは全てMyPLCのパッケージに含まれている。構成要素として、以下のサーバが挙げられる。

- Webサーバ

  Webサーバは、実験者向けの情報を提供する。実験に使用する公開鍵の登録なども行える。PLCAPIサーバを経由してデータベースを参照しており、ノードの情報の取得やSliverの割り当てなども行える。

- bootサーバ

  Webサーバが、実験者向けのサービスであるのに対して、bootサーバは、実験用ノード向けのサービスである。bootサーバでは、実験用ノードをインストールするためのファイルなどが提供されている。

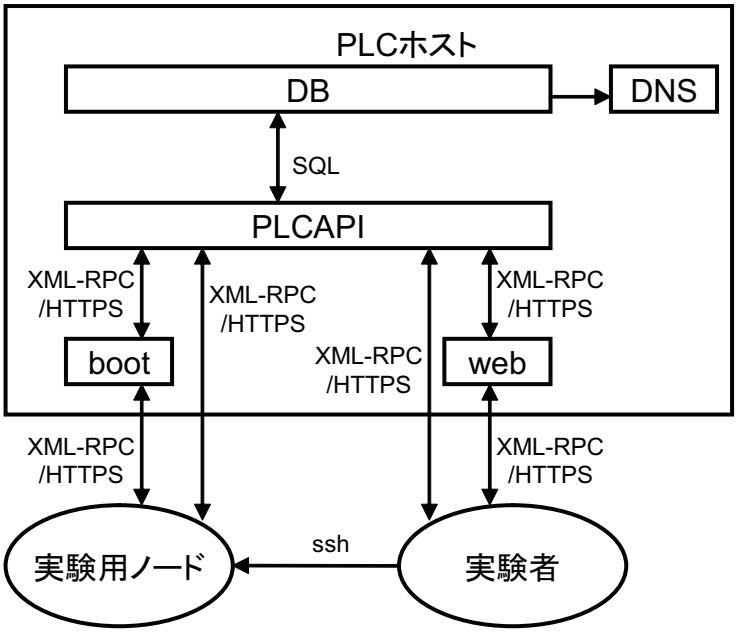

図 6.1: MyPLC の構成図

- PLCAPI サーバ

  PLCAPI サーバは、PLC を操作するための API を提供する。通常は Web サーバなどを経由して操作されるが、XML-RPC を用いて PLCAPI 関数を直接操作することが可能である。

- データベースサーバ

  PLCAPI サーバを経由し、ノードやネットワーク設定、起動の状態や、Slice、実験者のアカウントなどの各種情報を格納している。

- DNS サーバ

  実験用ノードの IP アドレスと名前の情報を提供する。データベースの情報を元にエントリを作成している。

構成図を図 6.1 に示す。実験者は Web サーバを経由して、使用するノードへの、公開鍵の割り当てなどを行える。boot サーバによって、公開鍵が実験用ノードに割り当てられる。実験者は、割り当てた公開鍵に対応する秘密鍵を用いて、実験用ノードを使用することができる。実験者が実験用ノードを使用する際には、SSH を使用する。

設定の手順としては、MyPLC のインストール後に plc-config-tty というコマンドを実行することで、インタラクティブに設定を行える。設定項目としては以下のものが挙げられる。

- 管理者アカウント
- PLC のサイト識別子
- 各種サーバの指定

各種サーバのプログラムが MyPLC のパッケージに含まれているため、1 台のノードで PLC ホストとして動作可能である。そのような設定を行う場合は、PLC ホストの IP アドレスまたはホスト名を登録すればよい。また設定される値は BootCD にも渡されるため、実験用ノードから到達できるアドレスを指定する必要がある。そのため、ループバックアドレスなどは指定してはならない。設定された情報は XML 形式のファイルとして、所定の場所へ保存される。

PLC へのノードの登録

実験用ノードは、事前に PLC へ登録しなければならない。通常は Web インターフェイスを用いて登録する。登録の際には、IP アドレス・ホスト名等を入力する。登録した際には plnode.txt というファイルが生成され、登録されたことの証明に用いられる。PlanetLab では遠隔地のノードにインストールし、PlanetLab 網に参加させることを想定している。このため不正なノードが参加しないよう、インストールおよび起動の際に認証を行い、登録されたノードであることを確認している。plnode.txt には、認証のための鍵の情報が記述されている。また、後述するネットワークの設定にも用いられる。

実験用ノードの用意

インストールの際に PLC との通信が行われる。そのため実験用のノードは、予めネットワークに繋がっている必要がある。plnode.txt には、このための設定も保存されており、使用するインターフェイス、IP アドレスを動的に取得するか、静的に設定するか等を指定できる。また実験用ノードには専用の OS をインストールする必要がある。

インストールメディアの用意

実際にノードへ PlanetLab ノードをインストールするためには CD-ROM などのメディアにインストーラを入れる必要がある。インストーラプログラムは BootCD と呼ばれ、CD-ROM に入れることを想定している。また CD-ROM 以外にも、USB メモリ用のイメージも用意されている。認証のため、BootCD はインストール時に PLC ホストと通信を行う必要がある。PLC ホストのアドレスなどの情報が必要になるため、PLC ごとに異なる BootCD が生成される。またインストール前に実験用ノードを、PLC へ登録する必要があるが、このときに生成される plnode.txt もインストール時に必要になる。plnode.txt はフロッピーディスクか USB メモリに入れることを想定している。PLC ホストが同じで

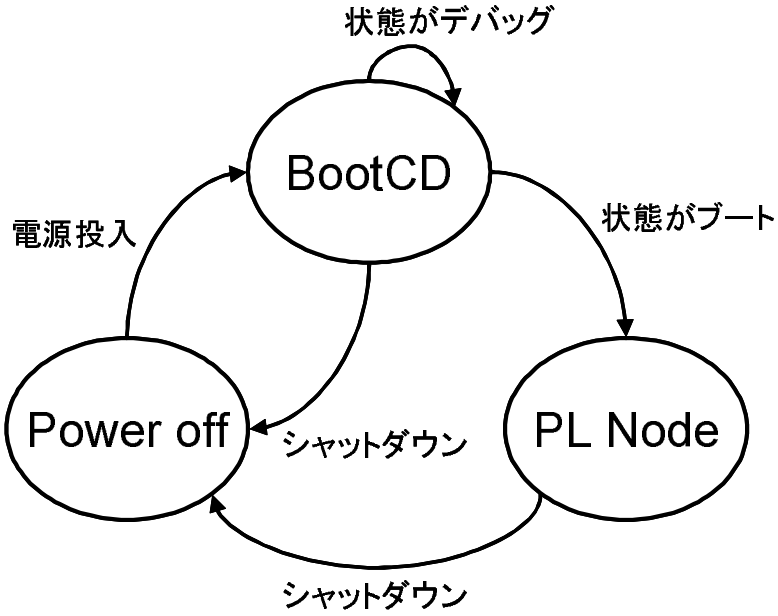

図 6.2: PlanetLab ノードの状態遷移

あれば BootCD は同じものを使用できるが、plnode.txt は実験用ノードごとに固有のものが生成される。BootCD はインストール以外にも、起動の際にも使用され、PLC と通信し、認証以外にも再インストールなどの状態確認を行ってから実験用の PL Node が起動する。また何か問題がある場合、状態はデバッグモードになり、PL Node としては起動せずに BootCD のプログラムが起動したままになる。BootCD と実験用ノードの状態遷移を図 6.2 に示す。このため BootCD は起動のたびに必要となり、ノードの数だけ用意しなければならない。

ノードの起動・インストール

ノードが起動し、BootCD が起動するとインストールが始まる。インストール中にはユーザが操作を行う必要はなく、インストールが進行する。

6.1.2 通常の構築方法における問題

大規模な Local PlanetLab を構築するにあたり、インストールメディアが必要な点が問題となる。節 4.4.3 で述べたように、メディアを用いたインストールでは、大規模な環境を構築することは困難である。またノードごとに異なるファイルが必要になるため、メディアを使わない方法でインストールする場合にも、工夫が必要となる。

6.2 構築用プログラムを用いたLocal PlanetLabの構築

Local PlanetLabの構築には、メディアが必要である点、ノードごとに異なるファイルが必要である点から、大規模な環境を構築することが困難であった。そこで本研究では、メディアレスインストールを実現するため、構築用プログラムとしてpl_autobuildを作成した。本節では、構築プログラムによる構築方法を述べる。

6.2.1 PLCホストを構築

PLCホストの構築に関しては、構築するPlanetLabの規模が大きくなっても、1台を構築するだけである。このため自動化しても、軽減される作業量は少ない。しかし設定ミスの防止や、後述するFederationのための設定、インストーラの取り出し等を見越すと、自動化する余地がある。

- pickup/wipeoutによる構築

  PLCはLinux上で動作するため、pickupを用いることでPLCをインストール・設定したノードのディスクイメージを作ることができる。しかし、複数のPLCホストを構築する場合には、ホストごとの設定を後から行う必要がある。また、インストールイメージは同じものを使い続ける必要がある。wipeoutによる構築後、PLCホストの設定を変更した場合はインストーライメージの更新作業が必要となる。

- スクリプトによる構築

  Linuxが既にインストールされているものとし、MyPLCの動作に必要な設定を行う。PLCが動作するための環境設定や、実験用ノードのインストールのために、インストーラのイメージを取り出す作業も合わせて行う。取り出したインストーライメージには、PLCホストのホスト名などを元にファイル名をつけることで、複数のLocal PlanetLabを構築する際に識別できる。

6.2.2 SpringOSによるノードの確保

実験用ノードの準備として、PLCと通信可能な状態である必要がある。ノードの確保と選択、およびネットワークの設定が必要となる。また、IPアドレスを動的に割り当てる場合は、DHCPサーバも必要になる。

ノードの確保

ノードの確保に関しては、ERMを使用する。また後述するノードの登録や、PXE Linuxのために必要となる情報もERMを用いて取得する。

ネットワークの設定

実験用ノードをインストールするには、予めネットワークにつながっている必要がある。ここでは、L2 以下のネットワークの設定を行う。StarBED では、実験用ノードは全て L2 スイッチに接続されており、VLAN を設定することで仮想的にトポロジを変更している。このため、実験用ノードを PLC ホストと通信可能な状態にするには、適切な port を VLAN に所属させる必要がある。VLAN の変更に関しては SWCP を用いる。また、設定に必要な情報は ERRP を用いて取得する。さらに、DHCP で IP アドレスを取得できるように、DHCP のための設定ファイルをあわせて生成する。IP アドレスも実験用のものを使うため、pl_autobuild が管理し、割り当てを行う。また MAC アドレスも、ERRP で取得する。

6.2.3 PLCAPI によるノードの登録

インストールのためには、予めノードを PLC へ登録する必要がある。通常、Web インターフェイスを用いて行われるが、XML-RPC を用いて、登録に必要な関数を呼び出すことでも行える。本研究では PLC を操作するための関数郡を PLCAPI と呼ぶ。本研究では、ノードの登録には XML-RPC を用いて PLCAPI を操作する。登録に必要な情報として、ホスト名は ERRP を用いて取得した名前を元に作成する。MAC アドレスも、ERRP から取得したものを PLC に登録し、IP アドレスは pl_autobuild が割り当てを行ったものを登録する。

6.2.4 インストールメディアの PXELinux への置き換え

大規模な Local PlanetLab を構築するには、大量のインストールメディアが必要になる。しかしメディアの作成やドライブへの挿入など、人手で行わなければならない作業があり、ノードの数だけそれらの作業を行うことは現実的ではない。そこで本研究では BootCD プログラムを、PXE Linux へ置き換えた。これにより多数のノードを構築することが容易になった。

BootCD の構成

BootCD は ISOLinux という、CD 用の Linux の規格として実装されている。以下のファイルから構成されている。

- isolinux.bin

  BIOS によって起動されるファイル。isolinux.cfg を読み込んで起動する。

- isolinux.cfg

  ISOLinux の設定ファイル。どのファイルを、何のために読み込ませるかを定義している。

- pl_version

  BootCD のバージョン情報が記述されている。

- boot.cat

  ブートカタログファイル。

- kernel

  ディスクレスの Linux として起動した際に、使用されるカーネル。

- bootcd.img

  ディスクレスの Linux として起動した際に、ファイルシステムとして利用されるファイル。どの PLC ホストが作成した BootCD でも、共通のイメージが含まれる。

- overlay.img

  ディスクレスの Linux として起動した際に、ファイルシステムとして利用されるファイル。bootcd.img とは違い、作成した PLC ホストごとに固有の情報が含まれる。

.img の付くファイルは、initrd という形式であり、RAM DISK に展開するために用いられる。ファイルの中身としては、Linux のファイルシステムである。ネットワーク経由で起動する PXE Linux でも同様の形式のファイルが使用されている。このため、構成ファイルをそのまま使用し、CD からの起動を PXE に置き換えることができる。

PXELinux の設定ファイルの命名規則

PXE Linux で参照される設定ファイルは、以下の優先順位で読み込まれる。

1. 対象ホストの MAC アドレスを元に生成した名前

   MAC アドレスの区切り文字は、通常:が用いられるが、PXE Linux での設定ファイル名では、-が区切り文字となる。MAC アドレスを元に決定しているため、IP アドレスが不確定な場合でも設定を適用できる。ただしインターフェイスカードを交換するなどで、MAC アドレスが変化した場合は対応できなくなる。

2. 対象ホストの IP アドレスを 16 進数にした名前

   IP アドレスを元に適用する設定ファイルを決定する。TFTP を使用する場合は DHCP を併用するため、DHCP での設定では、固定的に IP アドレスを割り当てる必要がある。

3. 対象ホストのネットワークを 16 進数にした名前

   ネットワークアドレスを元に適用する設定ファイルを決定する。DHCP によって設定されるネットマスクの長さとは無関係であるため、厳密にはネットワークアドレスではない。対象ホストの IP アドレスの先頭ビットが設定ファイルの名前と一致していれば適用される。マッチングに使用するビット長は、ルーティングテーブルのように最長マッチであり、適用する設定ファイルは、少しづつ名前の短いものを探す。

   ネットワークアドレスを元にグループを作成し、同じ設定を適用する場合に使用できる。

4. default

   上記のいずれのファイルも存在しなかった場合に読まれるファイル。全てのノードで同じ設定を適用する場合に使用できる。

どの名前のファイルを使用するかは、施設のポリシー等を踏まえて決定する必要があるが、StarBED のように複数の実験者で共有している施設では、ノードごとに決定できる 1 か 2 を使用することになる。どちらの場合でも、ERRP を使用することで、必要な情報を取得できる。

6.2.5 plnode.txt の配布方法

BootCD プログラムを PXELinux として起動することにより、CD-ROM などのメディアを使わずにインストールが行える。しかし実験用ノードのインストール・起動には、PLC へ登録されたノードであることを証明する plnode.txt が必要になる。このファイルはノードごとに固有のものであるため、BootCD のように同じものを共有することは出来ない。

本研究では plnode.txt の取得方法として、インストーラを改造する方法と、PXELinux の構成ファイルに含める方法の 2 つを実現した。

インストーラを改造する方法

BootCD に含まれるファイルを変更し、plnode.txt をネットワーク経由で取得する方法について説明する。ネットワークの設定を行う処理の後に、設定ファイルを取得する処理を追加した。自身の設定ファイル名を識別する方法として、IP アドレスに対応するように命名規則を設けた。plnode.txt を配布するサーバには PLC ホストを用い、PLC ホストの Web サーバ上からダウンロードする形で実装した。概念図を図 6.3 に示す。PXE boot を想定しているため、TFTP サーバから plnode.txt を取得することも考えられる。しかしその場合、TFTP サーバを変更する度に BootCD イメージを変更する必要がある。PLC

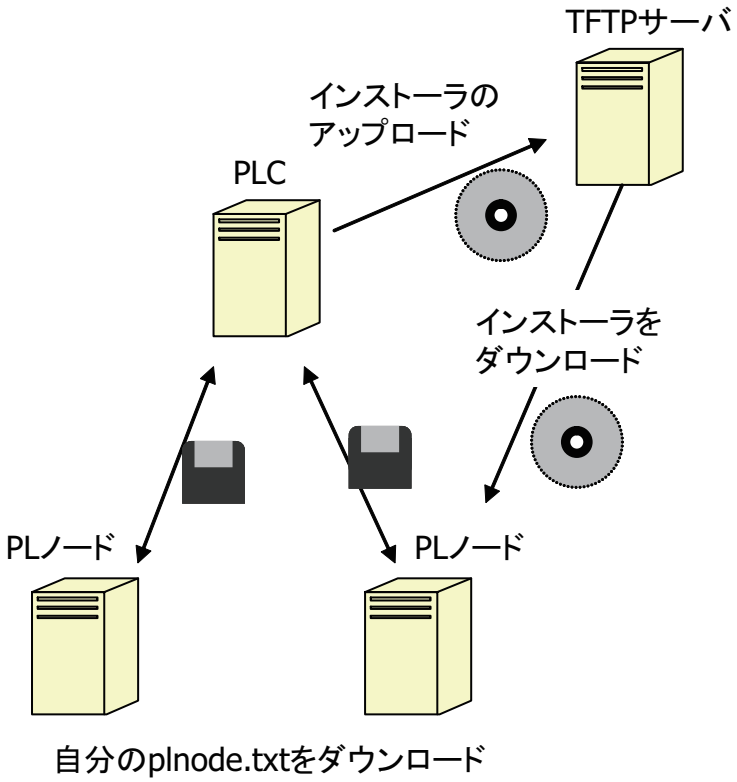

図 6.3: plnode.txt をダウンロードする方式

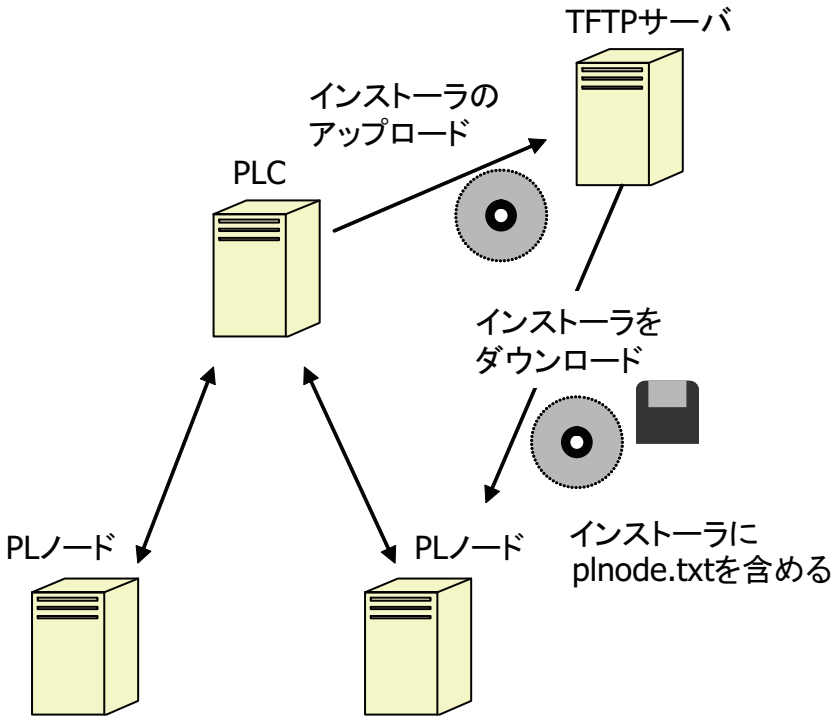

図 6.4: PXELinux の構成ファイルに含める方式

ホストのアドレスであれば、BootCD が変数から参照できるため、BootCD イメージの移植性が高まる。

メリットとしては、PXE Linux の設定ファイルの内容を単純化できる。この方式では同じ PLC に所属するノードであれば、設定ファイルの内容は全て同じになるためである。

デメリットとしては、plnode.txt でのネットワークの設定を行えなくなる点が挙げられる。このため、必ず DHCP を用いてアドレスを取得しなければならない。PXE Linux で起動するため、DHCP は用意されている環境ではあるが、使用するインターフェイスの指定が行えなくなる。StarBED のように、ノードが複数のインターフェイスを持っている前提の環境では、BootCD へ更に変更を加え、インターフェイスを選択するようにしなければならない。また BootCD に対して変更を行う必要があるため、BootCD のバージョンが変わる度に対応が必要になる。

PXELinux の構成ファイルに含める方法

plnode.txt は、BootCD のファイルシステム上での、特定の場所から参照される。このため、plnode.txt を PXE Linux 用のイメージとして用意することで BootCD と共に、各ノードに配布できる。概念図を図 6.4 に示す。

メリットとしては従来通りに、plnode.txt を用いてネットワークの設定が可能になる点が挙げられる。また BootCD のバージョンが変わっても、plnode.txt の場所や名前に関す

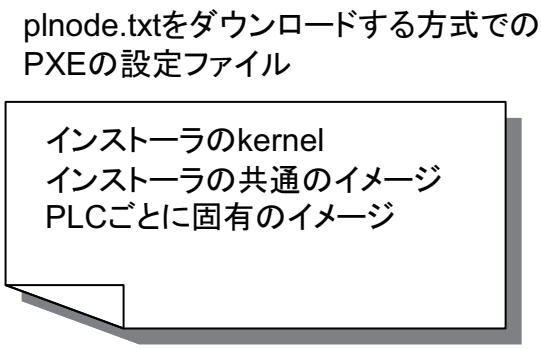

図 6.5: ダウンロードする方式での PXE の設定ファイル

るルールが変わらない限り対応可能である。

デメリットとしては、ノードごとに違う内容のイメージで起動しなければならない点が挙げられる。このため、PXELinux 用の設定ファイルもノードごとに異なる内容で作成しなければならない。ただし設定ファイルの内容は、ノードと PLC の組み合わせが同じであれば一意に決められる。

PXELinux 用の設定ファイルの適用

DMP を利用して、シンボリックリンクを設定する。前述の、plnode.txt の取得方法によって、設定ファイルの内容と命名規則が異なるため、それぞれの場合での規則も合わせて説明する。設定ファイルの名前については、PXE Linux の制約上、ノードごとに対応する名前でなければならない。しかし StarBED では、このように名前を変更できないファイルを扱うために、DMAN を用いてシンボリックリンクを変更する方式をとっている。これにより、ファイルに内容と関連性のある名前をつけることが可能となる。このため、煩雑なファイルの内容を参照せずとも、適切な名前を実体となるファイルにつけることで、識別することができる。

- インストーラが plnode.txt をダウンロードする方式

  同じ PLC に所属しているノードであれば、全て同じ設定ファイルが使用できる。このため、ファイル名の命名規則として、PLC のホスト名を元に決定する。ファイルの内容としては図 6.5 のようになる。所属している PLC が同じであれば、同じ設定ファイルを異なるノードで使用できるため、TFTP サーバ上では図 6.6 のような構成になる。図 6.6 中の L と書かれたファイルは、シンボリックリンクである。PXE Linux では設定ファイルの名前が決まっているため、各ノードに対応するものを用意しておく必要がある。しかし、ファイルの内容は PLC ごとに共通のものを使用するため、PLC 単位で共有できる設定ファイルへのシンボリックリンクを使用している。

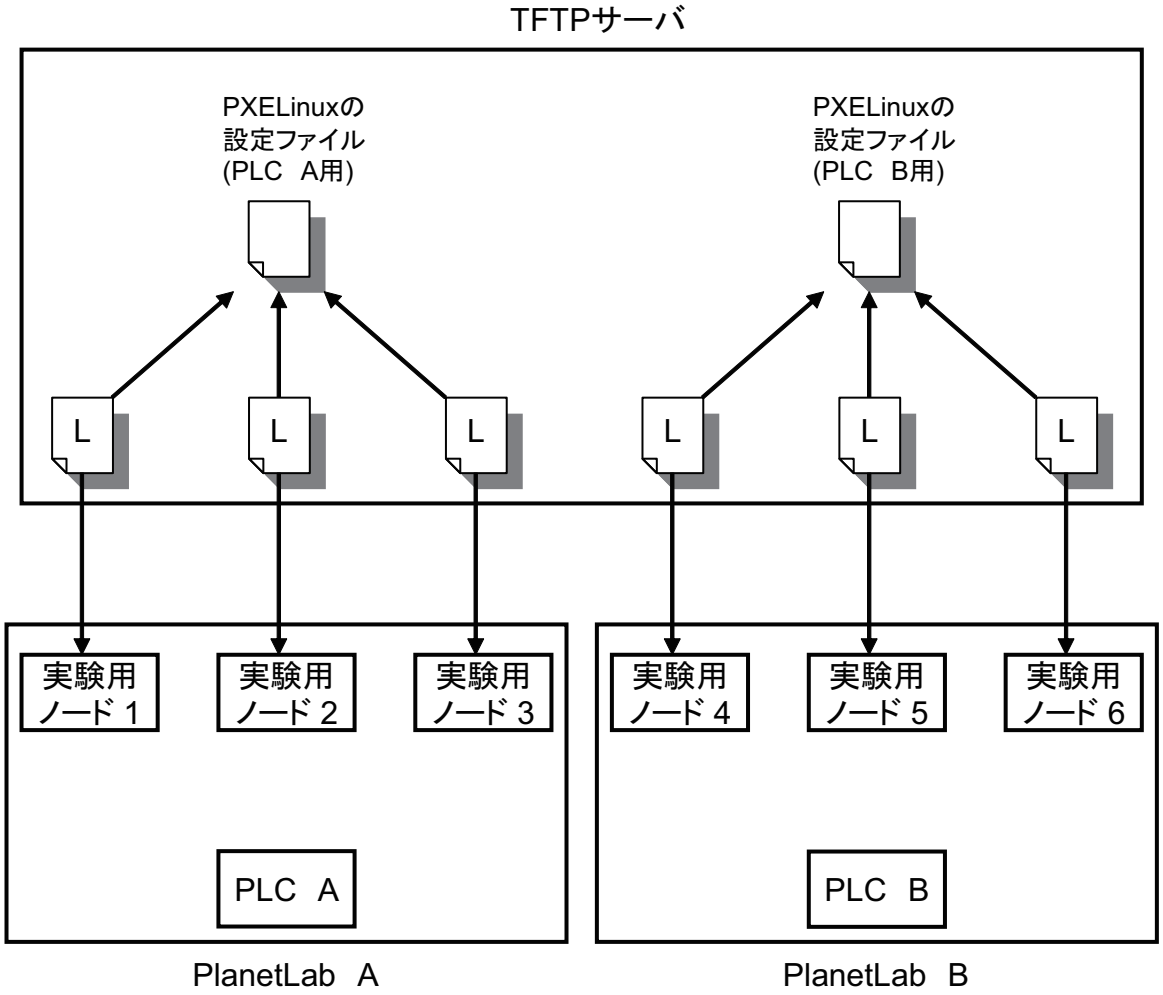

図 6.6: ダウンロードする方式での TFTP サーバ上のファイル

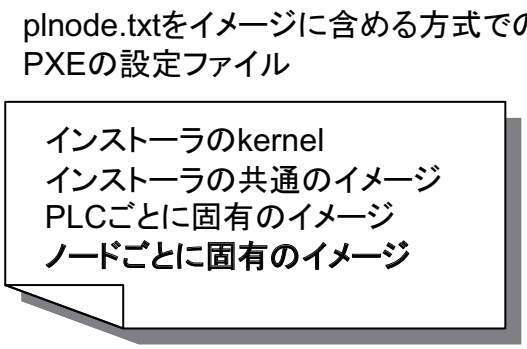

図 6.7: イメージに同梱する方式での PXE の設定ファイル

- インストーラに plnode.txt を同梱する方式

  ノードごとに異なる設定ファイルが必要になる。ファイルの内容に関しては、所属する PLC と、ノードの組み合わせによって決まる。ファイルの内容としては図 6.7 のようになる。このため、ノードごとに固有のイメージが必要になるため、TFTP サーバ上の PXE Linux 用の設定ファイルも、図 6.8 のように、ノードごとに必要になる。

6.2.6 ノードの起動・インストール

ノード・ネットワーク・BootCD の用意ができれば、ノードを起動するだけで起動・インストールが行われる。電源の操作も通常、手動で行わなければならないが、Wake on LAN を用いることで、ソフトウェアで行うことができる。SpringOS では、Wake On LAN のための仕組みとして、wolagent が用意されており、本研究でもこれを使用する。

6.2.7 実験環境の拡張

pl_autobuild は、追加での構築にも対応している。使用する VLAN やネットワークアドレスを、ファイルに保存している。再度実行した際には、ファイルを参照し、同じ VLAN やネットワークアドレスを使用するように実装した。これにより図 6.9 のように、既に構築した Local PlanetLab に対して、ノードを追加して拡張することができる。

6.3 実験実行支援

Local PlanetLab が構築できるため、実験実行支援には CoDeploy や pssh が適用できる。

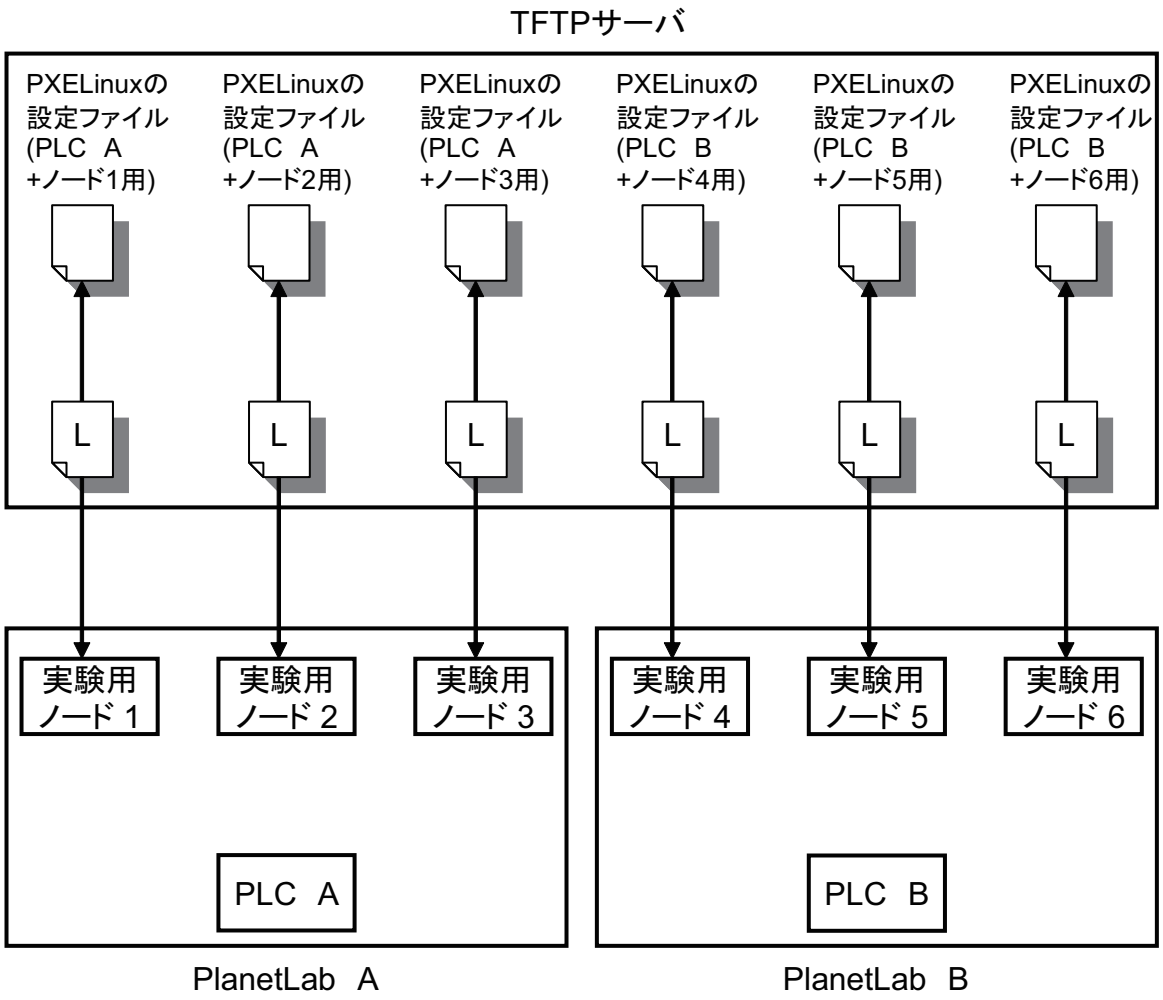

図 6.8: インストーラに含める方式での TFTP サーバ上のファイル

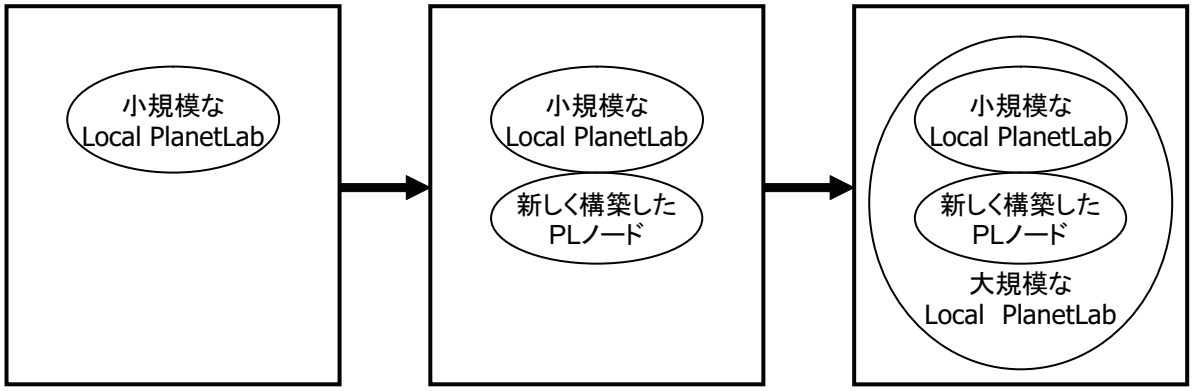

図 6.9: Local PlanetLab の拡張

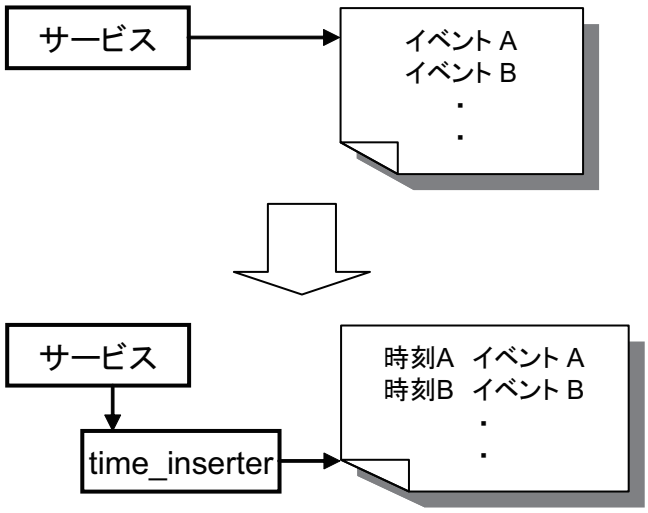

図 6.10: 時刻の挿入

6.3.1 グループごとの制御

pssh であれば、ノードのリストを実行時に指定できる。このため、グループごとのノードリストをファイルとして用意することで、グループごとの制御が可能になる。また PLC では、データベース上でノードのグループを作成することができる。

6.3.2 ログの収集

pssh や CoDeploy を使用することで、各ノードが持っているファイルを、収集することができる。同じ名前のファイルをコピーするため、収集するノード上で、ファイル名が衝突することが考えられる。しかし pssh も CoDeploy も、ファイル名が衝突しないようにディレクトリに分けるなどの処理を行っている。ノードの名前を利用しているため、どのノード上のログだったかもファイル名から判断することができる。

6.3.3 ログへの時刻の挿入

複数のノードからログを収集した場合、時刻を基準に比較する。ログに時刻が含まれるかは、サービスの実装に依存する。そこで本研究では、時刻を挿入するためのスクリプトとして time_inserter を作成した。概念図を図 6.10 に示す。time_inserter はサービスが出力するイベントログを受け取り、時刻を挿入した上でファイルに出力する。厳密には出力するタイミングで時刻が挿入されるため、イベントが発生した正確な時刻ではない。

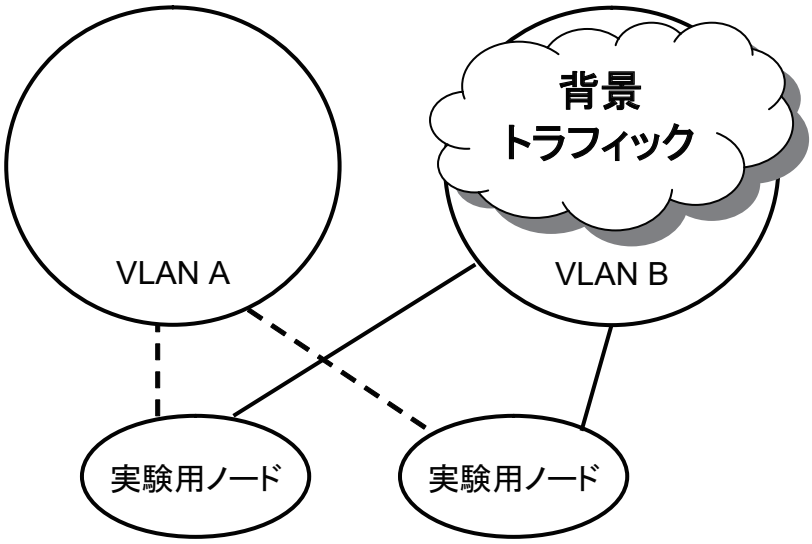

図 6.11: VLAN を変更して周辺要素を導入

6.3.4 サービス独自のプロトコルの制御

サービス独自のプロトコルを制御するため、指定したメッセージを送信するスクリプトとして、tcp_writer を作成した。tcp_writer はテキストベースのプロトコルを前提としており、指定した IP アドレスとポート番号に対して、指定したメッセージを送信する。このような、アプリケーション層でのプロトコルテストには telnet も使用できる。しかし telnet との違いとして、tcp_writer はインタラクティブな操作は行わない点が挙げられる。このため、スクリプト等に組み込みやすい。

6.4 ネットワーク接続の変更

6.4.1 施設内のネットワークへの接続

StarBED であれば、ネットワークは VLAN によって接続を切り替えているため、再現した周辺要素と、同じ VLAN に所属させることにより、周辺要素を取り入れられる。このため、VLAN を単位として周辺要素を扱うことができる。背景トラフィックを取り入れる場合の概念図を図 6.11 を示す。実験用ノードが元々所属していた VLAN への繋がりを点線、所属 VLAN を変更した後の繋がりを実線で表す。周辺要素を再現していないネットワークに繋がっている VLAN A から、背景トラフィックを再現したネットワークに繋がっている VLAN B へ接続を変更することで、実験用ノードは、背景トラフィックの影響をうけるようになる

6.4.2 インターネットへの接続

トンネル接続により、インターネットとの接続が可能になる。またトンネル接続先を変更することで、ノードのインターネット上での場所を変更できる。StarBED の場合は、WIDE Project のネットワークや JGN 等を経由してインターネットとの接続が可能である。しかし施設的な手続きが必要であり、実験者の権限で頻繁に切り替えることはできない。

SpringOS でも、外部との接続を行うことは想定されていない。そこで本研究では、外部との接続を含めたネットワーク接続の変更方法を提案する。

L2TP を用いた接続方法

L2TP サーバへトンネル接続する方法について説明する。トンネル接続を行うためのクライアントソフトウェアには rp-l2tp を採用した。rp-l2tp を用いたトンネル接続の手順は以下のようになる。

- 設定ファイルの作成

  ここで述べる設定ファイルとは、rp-l2tp のための設定ファイルと、PPP のための設定ファイルの両方を指す。それぞれ接続先に応じた内容で作成する必要がある。

- l2tpd の起動

  l2tpd は L2TP サーバのプログラムでもあるが、rp-l2tp の場合はクライアントとしても使用される。

- セッションの確立

  セッションの確立には l2tp-control というコマンドを使用する。l2tp-control はセッションの切断にも使用される。

設定ファイル中では、接続先となる L2TP サーバを指定する必要がある。また接続の際に用いる PPP のオプションをファイルに定義し、呼び出す必要がある。

6.4.3 トンネル接続の制御機構の実装

PLNode には、実験者に提供される VServer の領域とは別に、root context と呼ばれる領域がある。概念図を図 6.12 に示す。これにより、実験者に提供される領域に影響を与えずに、トンネル接続のための機能を追加できる。

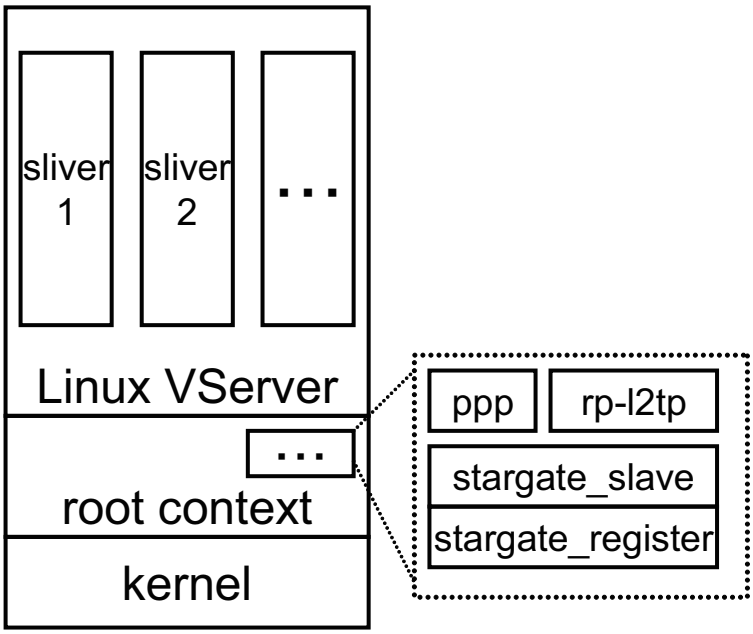

図 6.12: root context に接続用のプログラムを追加

接続用のパラメータの指定

トンネル接続に必要な、パラメータの指定方法について説明する。L2TP の場合は、L2TP によって仮想的な回線が設けられ、その仮想回線上で PPP を用いて接続が行われる。このため、L2TP 自体の設定以外にも、PPP の設定が必要になる。

設定内容として以下のものがあげられる。

- 暗号化の有無
- 暗号化アルゴリズム
- 圧縮の有無
- MTU・MRU
- ユーザ名・パスワード
- 認証の要求

これらはファイルに記述して指定する。本研究では、これらのパラメータは接続先ごとに固定的なものであると想定し、接続先ごとに命名規則を設けたファイルを用意して管理する。

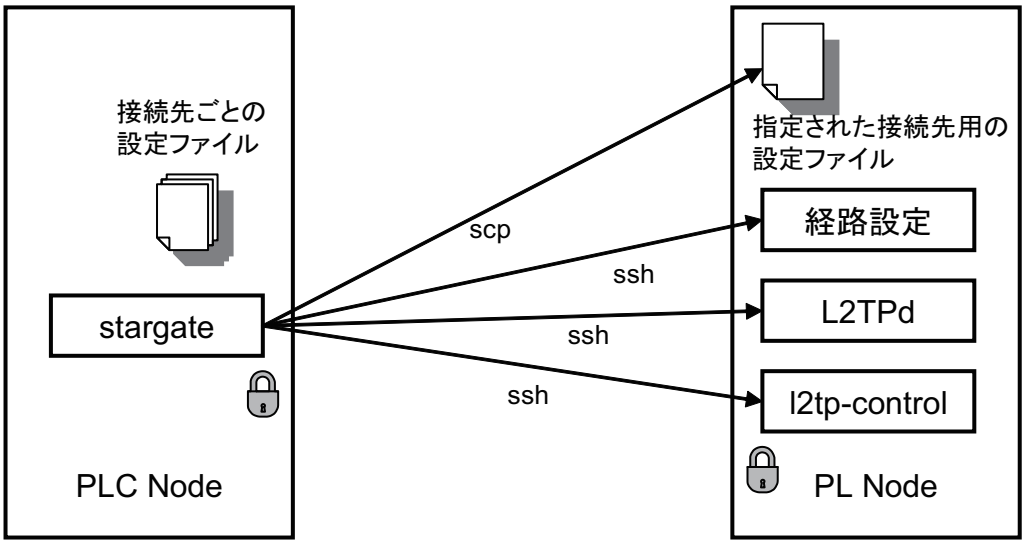

図 6.13: SSH を用いたトンネル接続の実行

外部との接続性の確保

トンネルサーバと接続するために、適切なネットワークに接続する必要がある。StarBED の場合は VLAN を変更し、NAT などの外部との接続性があるネットワークに接続させる。

接続に伴う変更

L2TP によって、グローバルな IP アドレスが取得できる。このため、PL ノードが実験で使用する IP アドレスが変化する。通信を行う実験の場合、通信相手となるノードは、IP アドレスで指定することになる。IP アドレスが変化した場合、実験内容の記述でも合わせて変更する必要がある。ただし DNS に IP アドレスを登録し、FQDN で指定するなどで、IP アドレスの変化は吸収できる。

SSH/SCP を用いた実装

本研究では、接続を実行するノードは、PLNode を対象にしている。PLNode の root context には、PLC の持つ秘密鍵を使用することにより、SSH で操作を行うことができる。これを利用して、設定ファイルの配布やトンネル接続の実行などの処理を、SSH を用いて実装することができる。

構成を図 6.13 に示す。

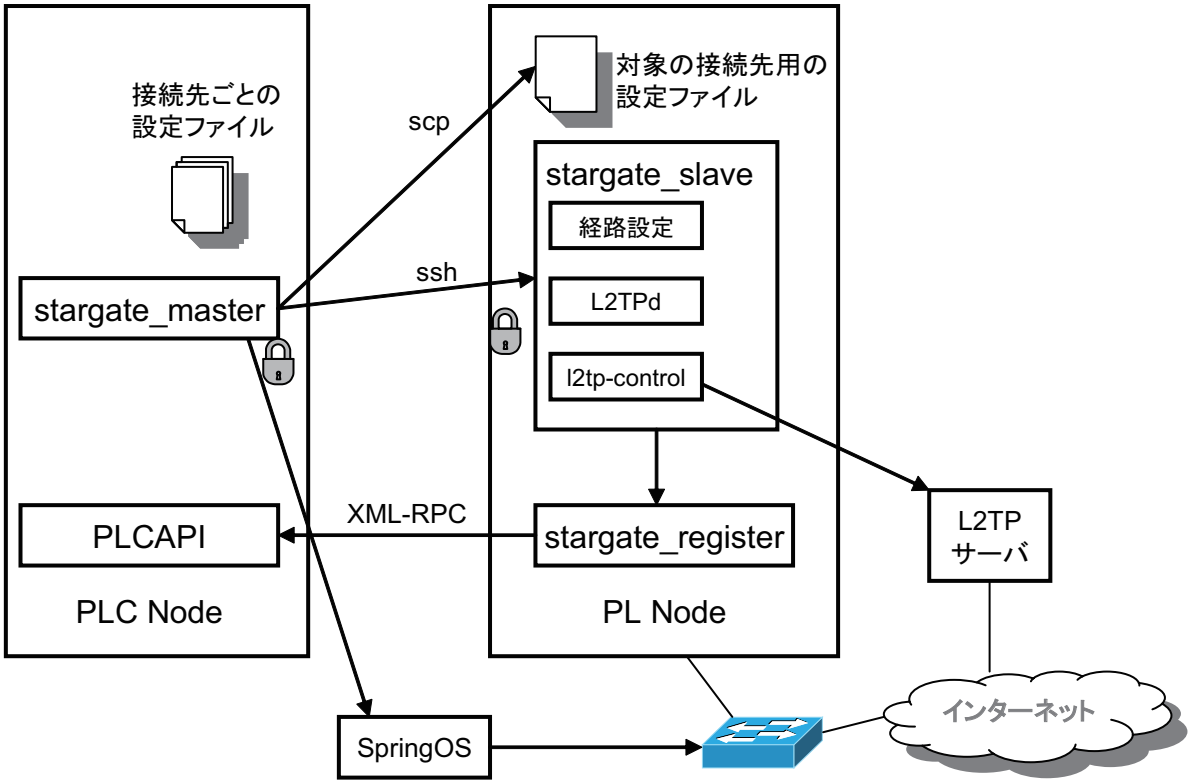

図 6.14: 3 分割構成

3 分割構成

VLAN の変更や、グローバル IP アドレスの取得といった処理も合わせたモデル。構成図を図 6.14 に示す。

- stargate_master

  設定ファイルの配布、所属 VLAN の変更を行う。設定ファイルの配布には、scp を使用する。

- stargate_slave

  dhclient 及びトンネル接続を実行する。経路の変更などの準備的な処理も行う。

- stargate_register

  接続後、取得した IP アドレスを PLC のデータベースに反映させる。L2TP の場合は、PPP を用いているため、ppp0 というインターフェイス名で識別される。register は ppp0 を参照し、新たに取得した IP アドレスを PLC ホストに報告・更新する。

PLC のデータベースに、新しい IP アドレスが反映される。PLC は DNS サーバとしても動作しており、データベースを元に、エントリを作成している。よって PLC を DNS サー

バとして利用するように設定し、ノードの指定に IP アドレスを用いず、FQDN を使用することで、アドレスの変化を吸収することができる。

ファイルサーバを用意する方法

設定ファイルの配布方法として、HTTP や FTP 等のサーバを用いる方法が考えられる。この方法であれば、PlanetLab ノード以外でも使用できる。しかし Web サーバや FTP サーバを別途用意する必要がある。PLC ホストでは Web サーバも動作しているため、それを流用する方法もあるが、PlanetLab ノードが対象であれば、SCP で実現できる。

6.5 Global PlanetLab との連携

PlanetLab には、Federation という機能があり、複数の PlanetLab 同士が協調できるようになっている。本研究でも、その機能を使用する。

6.5.1 共有しないノードの設定方法

通常、Federation は Global PlanetLab 同士で行われる。そのため、お互いに保持するノードは、全てインターネットに接続されており、実験者はインターネットを経由して使用することができる。しかし本研究では、Local PlanetLab と Global PlanetLab の間で Federation を組むことを想定している。Local PlanetLab のノードには、外部との接続性の無いノード、つまり Global PlanetLab に所属する実験者から利用できないノードが含まれる。

このようなノードは、Peer の PLC に所属する実験者にとっては不要なノードであり、共有しないことが望ましい。また共有したノードは、Peer の実験者によって利用が可能となる。そのため、共有した場合は占有ができなくなり、性能を求める試験には適さなくなる。

このため、ノードごとに共有するかどうかを設定できることが好ましい。

サイト名による設定

共有されない条件の一つに、所属サイトが同じノードがある。所属サイトが同じであるかの判断は、PLC 上での識別子で判断される。これを利用し、通常はデフォルトのサイトに所属させて共有させず、外部と接続したノードは接続先のサイトに所属を変更することで、共有可能にする方法で実現できる。ノードの所属しているサイトは通常、登録したものから変更することはできない。所属サイトはノードを提供しているサイトであり、他のサイトに変わるケースは少ないためと考えられる。サイト自体の情報を変更し、名前を変えることでも、共有のための制御は行える。しかし、サイトに所属しているノード全て

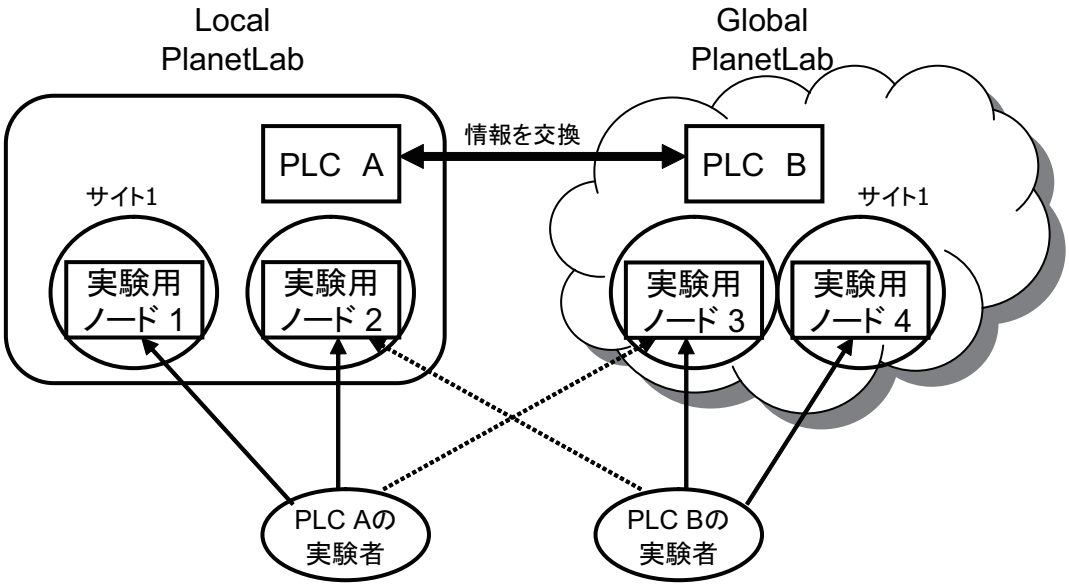

図 6.15: 同じサイトのノードは共有されない

が共有する対象となるため、細かな制御はできない。この方法を使用する場合、所属サイトを変更できるようにする必要がある。

NodeTag による設定

PLC のデータベース上の、ノードに関する情報を拡張することで共有しないノードであるかどうかの指標になる。PLC には、このような独自の情報を記述するために NodeTag という情報を作成できる。NodeTag にはタグの名前と値を保存することができる。本研究では NodeTag を付け加えた。タグの名前は is_local とし、値は共有しないノードならば True、共有するノードであれば False とした。Federation で情報を交換する際に、NodeTag の is_local を参照し、値が True であるノードは交換するリストから除外するように変更を加えた。Peer には既にフィルタリングされたリストが交換されるため、Peer のデータベース上では is_local が定義されていなくてもよい。このため、変更を加えていない PLC とも Federation を組むことができる。

6.5.2 Peer の登録

Federation を結ぶ相手の PLC ホストは、Peer と呼ばれる。Peer を登録するには、以下の情報及びファイルが必要となる。

- URL

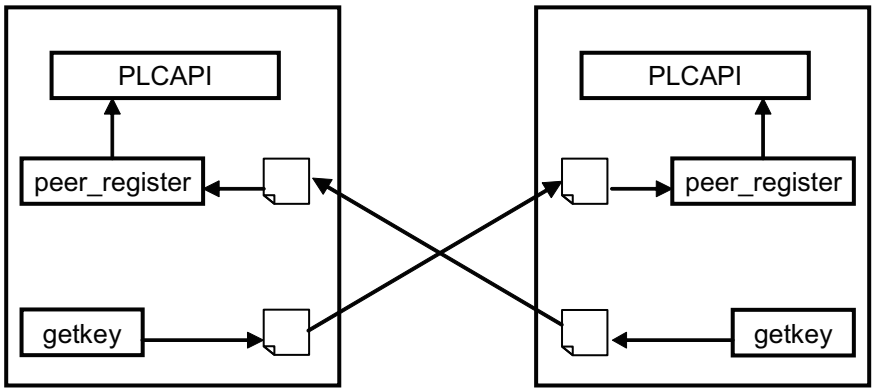

図 6.16: Peer の登録

情報を交換するため、Peer となる PLC ホストの、PLCAPI サーバとしての URL が必要になる。

- peername

  Peer の名前。データベースへの登録と、識別に必要となる。

- PLCAPI サーバの証明書

  PLC 同士での情報の交換には、HTTPS を用いて通信を行っているため、PLCAPI サーバの証明書も登録に必要となる。

- 公開鍵

  Peer としての認証には、PGP を用いている。このため、公開鍵をお互いに交換しておく必要がある。

本研究では、Peer の登録を支援するため、Peer 登録用スクリプトとして peer_register を作成した。また、登録に必要なファイルを取り出すスクリプトとして、getkey を作成した。getkey では必要な情報とファイルを、tar コマンドによってまとめ、tar.gz 形式で保存する。命名規則としては、peername.tar.gz のように、peername を元に決める。URL のように、元々ファイルとして存在していない情報は、peername.url 等の独自の拡張子で保存する。PLC 管理者同士で、tar.gz ファイルを交換した後、peer_register に tar.gz ファイルを読み込ませることで、Peer を登録する。動作を図 6.16 に示す。これによって、登録処理を簡略化できる。また PLC 管理者同士によって交換すべき情報やファイルを、tar.gz ファイル一つにすることができる。

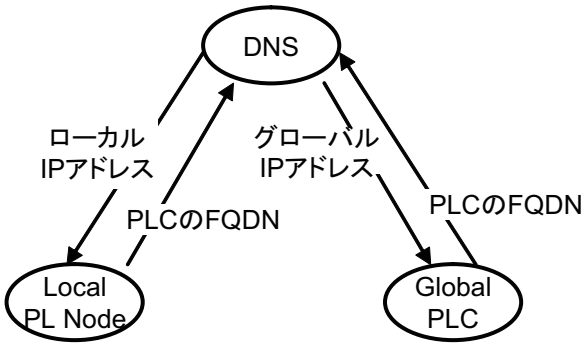

図 6.17: FQDN によって PLC の IP アドレスの違いを吸収

6.5.3 Local ノードの一時的な提供

Federation を組むことにより、Local PlanetLab のノードのうち、外部との接続性があるノードを Global PlanetLab に提供することができる。このようなノードは、Global PlanetLab として提供するノードに比べて、提供される期間が短くなることが予想される。また、一時的に提供するノードとなる場合もある。

このような形でノードを提供する場合、提供する期間を通知できることが好ましい。そこで、本研究では Life time の概念を提案する。Life time とは、一時的に提供されているノードに対して設定される情報であり、提供が終了する時間を定義する。実装方法として、PLC のデータベースを拡張することで行える。PLCAPI には、前述の NodeTag という概念があり、独自の情報を付与することができる。Local 専用ノードの場合とは異なり、Global PlanetLab 側で対応が必要となる。このため Federation を結ぶ際に、管理者同士で相談して設定すれば良いだろう。

6.5.4 Local 専用 PLC

本研究では、Local PlanetLab の PLC と Global PlanetLab の PLC とで、Federation を組むことを想定している。PLC 同士が通信を行える必要があるため、Local PLC にはグローバル IP アドレスが必要になる。しかし Local PLC は、プライベート IP アドレスを持った PL ノードとの通信が想定されている。このため、閉じた環境でしか使用できないプライベート IP アドレスが設定されている。予めグローバル IP アドレスと同じ値を実験用のネットワークで使用することや、プライベート IP アドレスとして設定しておくという方法も考えられる。しかし、この方法では PL ノードとの通信に支障が出る可能性がある。また、実験用のネットワークで使用する IP アドレスに制約ができてしまう。この問題も FQDN で設定を行うことによって、吸収することができる。図 6.17 に示すように、Local PlanetLab に対してはプライベート IP アドレスを、Global PlanetLab に対してはグローバル IP アドレスを返すようにすることで実現できる。

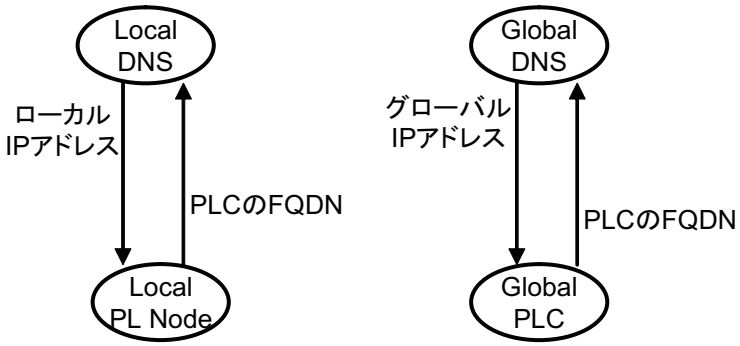

図 6.18: FQDN によって PLC の IP アドレスの違いを吸収

このような設定を DNS サーバに行う必要があるが、実際には図 6.18 のように、Local PlanetLab ノードが使用する DNS と、Global PlanetLab が使用する DNS は異なる。このため、それぞれの DNS サーバに、適切なエントリが登録されていれば実現できる。

第7章　評価

本研究では段階的検証において、想定する試験段階を stage として定義した。本章では、各 stage の構築の容易性を評価する。また StarBED 上での Local PlanetLab の構築を対象に、実験環境の構築と、実験実行の支援を実現した。

本研究では各 stage を実現するため、StarBED 上での大規模な Local PlanetLab の構築、トンネル接続によるインターネットとの接続、Federation 機能による StarBED と PlanetLab の間の協調を実現した。

7.1　想定する stage の構築の容易性

本節では、想定する stage の実現性を評価する。比較対象として、本研究で土台として使用しているテストベッドである StarBED と PlanetLab を比較し評価する。また、小規模な Local PlanetLab も合わせて比較し評価する。

7.1.1　pure stage

pure stage は、サービス以外の要素がない環境で試験を行う段階である。このため要件としては、周辺要素の無い環境を構築できることが挙げられる。また、各ノードの条件を同じにするため、ノードの性能は均一な環境であることが好ましい。

StarBED で環境を構築することにより、実験用ネットワークとして、周辺要素のないネットワークを作ることができる。また StarBED であれば、同じ構成の計算機を複数台利用することができる。同一の性能のハードウェアで環境を構築できるため、ノードの性能差による挙動の違いはなくなる。このため、StarBED の評価は○とした。PlanetLab は

表 7.1: 各 stage の実現

	StarBED	PlanetLab	Local PlanetLab	本研究
pure stage	○	×	△	○
emulated stage	○	×	×	○
Internet stage	×	○	×	◎

インターネットに接続されており、接続を変更することができないため、周辺要素を排除できない。また、提供されているノードの性能は同一ではなく、他の利用者による負荷の影響がある。このため、評価は×とした。Local PlanetLab は、大規模な環境構築が困難であるため、△とした。本研究は、StarBED 上で大規模な Local PlanetLab を構築している。StarBED 同様、実験用ネットワークとして構築するため、周辺要素の無い環境を構築することができる。また、StarBED のノードを使用するため、同一の性能のノードを使用できる。このため、評価を○とした。

7.1.2 emulated stage

emulated stage では、周辺要素を再現し、実験環境に取り入れられることが求められる。本研究では、周辺要素の再現については提案しなかった。しかし StarBED を対象とした、周辺要素の再現に関する研究がいくつかある。StarBED では、それらの技術で周辺要素を再現したネットワークと、試験対象のノードを同じ VLAN へ所属させることで、周辺要素を実験環境に取り入れることができる。

StarBED はこれらの周辺要素の再現が行えるため、評価は○とした。PlanetLab は接続が固定的であり、再現した周辺要素を取り入れることはできないため、評価は×とした。Local PlanetLab であれば、ノードの IP アドレスなどは変更でき、周辺要素にあわせて、所属するネットワークセグメントを変更することはできる。しかし、小規模な Local PlanetLab は、周辺要素を再現するための資源が無いと考えられる。このため、評価は×とした。本研究は、StarBED 上で大規模な Local PlanetLab を構築している。このため、StarBED を対象とした周辺要素の再現技術を取り入れることができる。よって評価を○とした。

7.1.3 Internet stage

Internet stage を実現するためには、インターネット上のノードを使用するか、ノードをインターネットへ接続させる必要がある。本研究では、トンネル接続によって、閉じた施設上のノードを、インターネットへ接続させることが可能になった。

トンネル接続によって、擬似的にインターネット上の任意のネットワークに配置できるようになった。しかし、遅延に関しては、実際のネットワークと同一とはいえない。施設からトンネルサーバまでへの遅延が上乗せされるためである。このため、遅延の測定などには使用できない。しかし Federation 機能によって、Global PlanetLab のノードも合わせて使用できるため、そのような用途には、Global PlanetLab のノードを用いることで実現できる。

トンネル接続により、仮想的な配置が可能になった。また、同じノードを様々なネットワークに接続させることで、ネットワークごとの挙動の違いを調べることができる。接続

表 7.2: 実験環境の構築に関する評価

	StarBED	PlanetLab	Local PlanetLab	本研究
OS のメディアレスインストール	○	-	×	○
実験環境の初期化	○	○	△	◎
実験環境の管理	×	△	△	○
実験環境の拡張	△	-	△	○
時刻の同期	△	○	△	○
試験対象サービスとの分離	○	○	○	◎

できるネットワークの多様性は、利用可能なトンネルサーバの数に依存する。また、Global PlanetLab のノードを使用することにより、更に選択肢を広げることができる。

StarBED では、外部との接続そのものは可能であるが、施設的な手続きが必要であり、実験者の権限で変更はできない。また、利用可能なネットワークには限りがある。このため、評価は×とした。PlanetLab であれば、数多くの組織から提供されているノードを使用できる。このため、様々なネットワークに接続されているノードを使用できる。よって、評価を○とした。Local PlanetLab は、外部との接続性が無い環境を想定しているため、評価は×とした。本研究では、トンネル接続によるインターネットへの接続と、Federation 機能による PlanetLab との協調を実現した。トンネル接続と、PlanetLab ノードの利用の２通りの方法で Internet stage を実現できるため、評価は◎とした。

7.2 実験環境の構築に関する評価

本節では、実験環境の構築について評価を行う。比較対象として、本研究で土台として使用しているテストベッドである StarBED と PlanetLab を比較し評価する。また、小規模な Local PlanetLab も合わせて比較し評価する。

7.2.1 OS のメディアレスインストールについての評価

本研究では、Local PlanetLab のメディアレスインストールを実現した。通常の構築方法では、CD-ROM などのインストールメディアを使用するため、メディアの作成時間、ドライブへの挿入の手間などの点から、多数のインストールは困難であった。更に、BootCD の作成に使用した CD-ROM は、再利用ができないため、毎回メディアを作成しなければならなかった。設定ミスなどによって、メディアの作成をやり直す場合は、更に時間を消費してしまう。

本研究ではこれをPXE Linuxに置き換えることにより、時間的・作業的コストを大幅に軽減し、大規模なLocal PlanetLabの構築を実現した。

更に、SpringOSを用いることで、PXEブートで必要になるTFTPサーバ上のファイルにも、適切なシンボリックリンクが生成される。これによって、StarBEDのように複数の実験者で共有しているTFTPサーバでも、ファイル名の衝突などを回避できる。ネットワーク接続などもSpringOSで設定できるため、実験者はPLCホストを用意した後、pl_autobuildを実行するだけで、自分の使用するLocal PlanetLabを構築できるようになった。これにより、作業コストを大幅な軽減が見込める。また設定ミスなども防止することができる。

SpringOSでもpickup/wipeoutにより、メディアレスインストールが行えるため、評価は○とした。PlanetLabは、OSがインストールされた状態で提供されているため対象外とし、評価は-とした。Local PlanetLabは前述のように、メディアを使わないインストールは困難であったため、評価は×とした。本研究は、Local PlanetLabのメディアレスインストールを行えるため、評価は○とした。

7.2.2 実験環境の初期化に関する評価

共有して利用するテストベッドでは、実験後、他の実験者が利用出来るようにするために、実験環境の初期化が必要な場合がある。また、何か問題があり実験をやり直す場合など、実験のために実験環境を初期化することも考えられる。StarBEDでは、ノード単位の初期化であればpickup/wipeoutによって、施設が用意しているOSをインストールすることで、初期化することができる。Local PlanetLabの状態に初期化する場合は、より手軽になり、再インストールまたは、Sliverの割り当てなおしによって、実験環境の初期化を実現できる。

StarBEDはpickup/wipeout等によって、初期化が行えるため、評価は○とした。PlanetLabは、元々他の実験者と共有して使う前提であり、他の実験者のために初期化する必要はない。実験者が再構築のために初期化する場合は、Sliverの割り当て直しで実現できる。このため、評価は○とした。ただしある状態を保存し、その状態に復元することはできない。Local PlanetLabは、実験者が再構築のために初期化する場合は、Sliverの割り当て直すことで、容易に初期化できる。また、PLノードのOSを再インストールすることも容易である。PLCを操作し、該当ノードの状態を「再インストール」にしてから該当ノードを再起動すれば、再インストールが行われる。しかし他のOSがインストールされていた状態に戻す機能はないため、評価は△とした。本研究では、他の実験者が利用出来るように初期化する場合は、pickup/wipeoutによって初期化できる。実験のために、使用する実験環境を初期化する場合は、Sliverの割り当て直し、あるいはPlanetLabノードの再インストールによって初期化できる。このため、評価は◎とした。

7.2.3 実験環境の管理に関する評価

段階的検証では、実験環境を何度も構築・変更して実験を行う。このため、実験者が現在の実験環境の情報を参照できることが好ましい。実験者が構築・設定した環境は、PLC によって管理されている。また IP アドレスを変更した場合も、stargate_register によって新しい IP アドレスが通知され、更新される。このため PLC ホストを経由して、実験環境の情報を取得できる。ただし再現した周辺要素などについては、PLC によっては管理されない。

StarBED では、施設の情報を管理するための機能はあるが、実験者が構築した環境を管理するための方法は特に提案されていない。このため、評価は×とした。PlanetLab では、PLC によって管理されている。しかし、実験者が IP アドレスなどを変更することはできないため、実験者が構築した環境という概念が無い。このため、本研究で想定する用途での管理は該当しないので、評価は△とした。Local PlanetLab であれば、IP アドレスなどを変更できる。このため、PLC による管理には意味がある。また、PLC の管理権限があれば、PLCAPI を用いてデータベースを拡張することができる。このため、実験者が管理のために必要な情報を拡張することができる。しかし変更した情報を更新するための仕組みは、別途容易する必要がある。このため、評価は△とした。本研究では stargate_register によって、変更された IP アドレスを反映させるようにした。これにより、PLC を用いて実験環境の管理が行えるため、評価は○とした。

7.2.4 実験環境の拡張に関する評価

段階的検証のためには、実験環境を拡張できる必要がある。本研究で大規模な Local PlanetLab を構築するために実装した pl_autobuild は、既に構築した Local PlanetLab を拡張できる。一度構築した場合、情報をファイルに保存している。二回目以降の実行ではファイルを参照し、同じ VLAN に所属するように構築を行う。また ERRP で、HOLD し続けることにより、既にインストールされているノードは、二回目以降にインストールする対象にはならない。また DHCP サーバにも、追加したノードの情報を反映させ、設定ファイルに追加する。

StarBED では、特に拡張のための仕組みはないが環境構築のための仕組みで実現できる。しかし実験者が適切に構築する必要があるため、評価は△とした。PlanetLab では、実験者が実験環境を構築しないため対象外とし、評価は-とした。Local PlanetLab は StarBED と同様、拡張しての構築を行うことができるが、実験者がそのように構築する必要があるため、評価は△とした。本研究では、拡張での構築に対応できるように pl_autobuild を実装したため、評価は○とした。

7.2.5 時刻の同期に関する評価

分散サービスの開発検証のように、複数のノードを用いて実験を行う場合、時刻は実験結果を解析する指標となる。このため、使用するノードの時刻は同期されていることが求められる。時刻の同期のためには、NTP というプロトコルがある。

StarBED では、施設の中で NTP サーバが提供されているが、NTP の設定は実験者が行わなければならない。SpringOS の枠組みには NTP サーバはないが、SpringOS によって NTP サーバの構築は行える。このため、評価は△とした。PlanetLab では、起動時に NTP で時刻の同期が行われており、起動後の PL ノードとしても、root context で ntpd が定期的に時刻を同期している。このため、評価は○とした。Local PlanetLab では、NTP サーバとの通信が行えるかによって、時刻の同期が行えるかが変わる。設定されている NTP サーバは外部のものであるため、NAT などで外部と接続が可能でなければ利用できない。外部と接続できない場合は、内部に NTP サーバを構築する必要がある。このため、評価は×とした。本研究では、大規模な Local PlanetLab を使用しているため、実験用ノードが NTP を使用するための設定は行われている。外部との接続性が無い環境で、内部の NTP サーバを使用する場合も、DNS に適切な IP アドレスを登録することで、実験用ノードに設定を行わなくても、NTP による時刻の同期が行える。このため、評価は○とした。

7.2.6 試験対象サービスとの分離に関する評価

時刻の同期など、実験環境の構築・維持に必要な機能は、試験対象のサービスに影響を与えないことが好ましい。

StarBED では、ネットワークを管理用と実験用に分けている。これにより、トラフィックとして試験対象のサービスと、実験環境の管理を分離している。このため、評価は○とした。PlanetLab では、実験者に提供する環境を Sliver として独立させており、時刻の同期や Sliver の割り当てなどの機能と分離されている。これにより試験対象と、実験環境の運用に必要な機能を分離している。このため、評価は○とした。この仕組は Local PlanetLab でも同様であるため、Local PlanetLab の評価は○とした。本研究では、StarBED 上で Local PlanetLab を構築しているため、ネットワークを実験用・管理用に分けることによる分離と、Sliver による分離の両方が実現できる。またトンネル接続などの、本研究で独自に加えた変更も、root context に対して変更を加えた。このため、評価は◎とした。

7.3 実験実行支援の評価

本節では、実験実行支援について評価を行う。比較対象として、StarBED の実験支援ソフトウェアである SpringOS と、PlanetLab に適した支援技術である pssh と比較して評

価を行う。本研究でも pssh を使用するが、時刻の挿入などの目的で作成したソフトウェアがあるため、合わせて評価を行う。

7.3.1 実験内容の記述

実験の再利用性や、ミスの防止、作業コストの軽減などの面から、実験内容を記述し、実行する機能が求められる。

StarBED では、kuroyuri によって実験内容を予め記述しておき、自動的に実行できる。このため、評価は○とした。PlanetLab にはシナリオ実行のための仕組みはない。しかし、シェルスクリプトなどで記述しておけば、pssh などのソフトウェアから複数台のノード上で、まとめて実行できる。このため、シナリオ実行の評価は×、実験内容の記述としては△とした。本研究では、kuroyuri は使用せず、pssh などを想定しているため、同様の評価とした。

7.3.2 グループ化による制御に関する評価

サーバとクライアントに分かれて、それぞれで実験に必要な処理を行うように、実験においては、各ノードをグループに分けて実験を実行することが考えられる。このため、グループ化による制御が必要になる。

StarBED では、kuroyuri を用いることで、class として定義したグループを制御することができる。このため、評価は○とした。PlanetLab でも、pssh を用いることで、グループに分けて制御を行える。pssh で使用するノードのリストは、ファイルに記述して定義できる。グループ毎にファイルを作成し、実行する度に指定することで、グループに分けての制御が行える。ただし、異なるグループへの制御を同時に行うことはできないため、スクリプトを作成するなどの工夫が必要となる。このため、評価は○とした。本研究でも、pssh による制御を想定しているため、評価は○とした。

7.3.3 ログの収集に関する評価

実験後、結果を元に解析や評価を行うことが考えられる。実験結果を収集する必要があり、本研究では実験結果は、ログファイルとして各ノード上に保存されていることを想定している。

StarBED では、特にログの収集のための仕組みはない。しかし、シナリオ内でログの収集処理を記述することで実現はできる。このため、評価は△とした。PlanetLab では、pssh などを使用することにより、複数のノードから同じ名前のファイルを収集できる。収集したノード上で、名前が衝突しないように、ディレクトリ分けや名前の変更を適切に行なっている。このため、評価は○とした。本研究でも pssh などによる収集を想定しているため、評価は○とした。

7.3.4 ログへの時刻の挿入に関する評価

本研究では、実験結果は各ノード上にログファイルとして保存する方式をとっている。複数のノードからログを収集した場合、時刻を基準に比較し、解析を行う。しかしログに時刻が挿入されるかは、サービスの実装に依存している。time_inserter を用いることで、時刻を挿入して、ログを出力できるようになった。前述のとおり、time_inserter がログを出力するタイミングで時刻が挿入される。このため、正確にはイベントが発生した時刻ではない。ただし実験においてログの時刻は、ノード間でイベントの相関を調べるための指標に過ぎない。このため正確な時刻である必要はなく、全てのノードが time_inserter でログに時間を挿入していれば、条件は同じになる。

time_inserter で時刻を挿入したサービスと、元々ログに時刻を挿入するサービスでは、time_inserter の処理が入る分のずれが生じる。

StarBED や PlanetLab では、ログへ時刻を挿入する仕組みは提案されていない。このため、評価は×とした。本研究では、ログへ時刻を挿入するための仕組みを実現したため、評価は○とした。

7.3.5 サービス独自のプロトコルの制御に関する評価

サービス独自のプロトコルを扱うには、そのサービスに対応するサーバやクライアントのプログラムが必要である。シェルスクリプトなどからそれらを実行することはできるが、特定のプロトコルメッセージのみを扱えるとは限らない。本研究では、独自のプロトコルメッセージを扱うために、tcp_writer を実装した。これによって、サービス独自のプロトコルを、シェルスクリプトなどから扱うことが可能になった。しかし、tcp_writer で送信したメッセージへの応答などは取り扱えない。このため、クライアントやサーバの挙動までは模倣できない。また tcp_writer の実装上、TCP を使用するテキストベースのプロトコルしか扱えない。

StarBED や PlanetLab では、サービス独自のプロトコルを扱うための仕組みは提案されていない。このため、評価は×とした。本研究では、サービス独自のプロトコルを扱うための仕組みを実現したため、評価は○とした。

7.4 テストベッド間の協調に関する評価

PlanetLab の機能を用いることによって、Local PlanetLab と Global PlanetLab の協調が可能になった。Local PlanetLab として StarBED 上で構築した環境を、Global PlanetLab として本学の研究室で構築した環境を使用し、確認を行った。

表 7.3: 実験実行支援の比較表

	SpringOS	PlanetLab(pssh)	本研究
実験内容の記述	○	△	△
シナリオ実行	○	×	×
グループ化	○	○	○
ログの収集	△	○	○
ログへの時刻の挿入	×	×	○
独自プロトコルの制御	×	×	○

7.4.1 非共有ノードの設定に関する評価

Federation による情報交換で、is_local が設定されているノードは、共有されないことを確認した。本研究での実装では、共有しないノードを、取り除いた情報を交換する。このため Peer は、特に非共有のための設定に対応している必要はない。

第8章　今後の可能性

8.1　閉じた施設間の接続と協調

本研究では、段階的検証のため、Local PlanetLab と Global PlanetLab の協調を実現した。しかしこれ以外にも、異なる施設にある Local PlanetLab 同士の協調に利用できる。閉じた施設同士であっても、トンネル接続で施設間を結ぶことができれば Federation 機能を用いて情報を交換し、協調して利用することができる。もちろん、ポリシー上共有したくないノードは、共有対象から除外することができる。

8.2　周辺要素同士の協調

本研究では、特に周辺要素の再現については既存技術で可能であるため、提案しなかった。既存の技術を用いて、各周辺要素を再現し、組み合わせることで複数の周辺要素を再現することができる。しかし大規模なネットワークの模倣は、各要素を細かに設定するだけでも大きな時間がとられてしまう。複数の周辺要素の再現と協調方法については、検討の余地がある。

8.3　MyPLC の開発検証

展開が容易になることにより、MyPLC 自体の開発検証にも利用できる。例えば、MyPLC は IPv6 への対応および、Sliver ごとに IPv6 のアドレスプレフィックスを割り当てる機構が提案されている。本研究の仕組みを用いることで、そのような機能の動作検証や負荷試験、試験運用などが可能になる。また PL-VINI のように、独自の改良を加える亜種の開発にも利用できる。

8.4　StarBED の新しい利用形態

StarBED は利用期間で分割することにより、複数の実験者での共有を可能にしている。本研究の成果の一つとして、StarBED 上で大規模な Local PlanetLab を構築することが可能になった。これにより、StarBED のノードを複数の実験者で同時に利用するという、今までになかった利用方法が可能になる。論文を提出する時期などは、StarBED も予約が

集中し、十分な資源を確保できないことがある。しかし Local PlanetLab によって、同時期で共同で利用することにより、利用可能な台数を増やすことができる。また StarBED の予約や資源の割り当ても、PlanetLab の枠組みを用いて行える。ノードの初期化などの作業も、Slice 単位で行える。ただし、各計算機の性能を限界まで使用するような用途には向かない。また、PlanetLab と同様の制約があり、IP アドレスなどを実験者の権限で変更することはできなくなる。

8.5 他のテストベッドへの応用

本研究ではテストベッドの組み合わせとして、StarBED と PlanetLab を選択し、大規模な Local PlanetLab 用に、実験環境構築ソフトウェアを実装した。

PlanetLab 以外のテストベッドに適用するには、そのテストベッド用に実験環境構築ソフトウェアを実装する必要がある。また、テストベッド間の協調や実験環境の管理など、PlanetLab の機能を利用した部分は、別途実装する必要がある。

第9章　おわりに

本研究ではサービスの段階的開発検証環境を提案した。分散サービスの開発検証を対象に、段階的検証の必要性を述べた。段階的検証において、試験環境に求められる要素を議論した。試験環境の持つ要素を元に、試験段階に対応する試験環境を stage として定義した。各 stage 間の移行方法を議論し、ネットワークの接続変更による移行を提案した。テストベッド間の協調と、環境構築と、実験実行支援からなるテストベッドアーキテクチャを提案した。提案アーキテクチャの実現のために既存技術を検討し、StarBED と PlanetLab を対象に適用した。既存技術では実現できない部分として、大規模な Local PlanetLab の構築、トンネル接続の制御を実現した。またテストベッド間の協調では、共有しないノードを設定できるように、拡張を行った。既存のテストベッドと比較し、定義した stage の構築が容易にできることを確認した。本研究の成果を応用した、今後の可能性を議論した。

謝辞

本研究を行うにあたり、主指導教員である本学 篠田 陽一教授には数多くの助言と指導を頂きました。深く感謝し、心からをお礼を申し上げます。主テーマ審査委員丹 康雄教授、知念 賢一特任准教授、敷田 幹文准教授、副テーマ指導教官井口 寧准教授には、本研究を始めるにあたり、助言をいただきました。深く、心から感謝を申し上げます。

情報通信研究機構の研究員、三輪 信介氏、太田 悟氏、Razvan BEURAN 氏、宮地 利幸氏、にはStarBEDをはじめとするテストベッドに関する情報や本研究についての意見など、多くの助力をいただきました。情報通信研究機構 北陸リサーチセンターの利用の際には、中井 浩氏、石崎 淳氏、竹中 ゆかり氏にお世話になりました。

WIDE Projectの方々には、研究に関して多くの意見を頂きました。また特に、村本 衛一氏、河口 信夫氏、今井 裕二氏、櫻井 覚氏、管 文鋭氏には、本研究のさきがけとなった共同研究にご協力いただきました。

本学中川 晋一客員准教授、宇多 仁助教授、小原 泰弘助教授には、研究および普段の生活の両面でお世話になりました。

本研究室LATT Khin Thida氏、高野 祐輝氏、安田 真悟氏、井上 朋哉氏、Nguyen Lan Tien氏、芳炭 将氏、NGUYEN, Nam Hoai氏、Muhammad Imran Tariq氏、佐川 喜昭氏、石渡 優祐氏、川瀬 拓哉氏、栗原 良尚氏、明石 邦夫氏、立花 一樹氏、中村 祐輔氏、橋本 将彦氏、山田 悠介氏、吉岡 慎一郎氏には、研究以外にも、普段の生活の面で、支えていただきました。

参考文献

[1] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. Planetlab: An overlay testbed for broad-coverage services. *ACM SIGCOMM Computer Communication*, Vol. 33, pp. 2–13, July 2003.

[2] W. Zhang and et al. Linux virtual server. http://linuxvirtualserver.org/.

[3] Tatu Ylonen. Secure login connections over the internet. *Sixth USENIX Security Symposium*, pp. 37–42, March 1996.

[4] Planetlab japan. http://www.planet-lab-jp.org/.

[5] Toshiyuki Miyachi, 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.

[6] S. Alexander and R. Droms. DHCP Options and BOOTP Vendor Extensions. RFC 1533 (Proposed Standard), October 1993. Obsoleted by RFC 2132.

[7] K.R. Sollins. TFTP Protocol (revision 2). RFC 783, June 1981. Obsoleted by RFC 1350.

[8] Shinsuke Miwa, Mio Suzuki, Hiroaki Hazeyama, Satoshi Uda, Toshiyuki Miyachi, Youki Kadobayashi, and Yoichi Shinoda. Experiences in emulating 10k as topology with massive vm multiplexing. *In Proceedings of The First ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures(VISA 2009)*, August 2009.

[9] Brian White, Shashi Guruprasad, Mac Newbold, Jay Lepreau Leigh Stoller, Robert Ricci, Chad Barb, Mike Hibler, and Abhijeet Joglekar. Netbed: an integrated experimental environment. *ACM SIGCOMM Computer Communications Review*, Vol. 33, p. 27, July 2002.

[10] Daniel Mahrenholz and Svilen Ivanov. Real-time network emulation with ns-2. *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pp. 29–36, 2004.

[11] 宮地利幸. 大規模実証環境の実現と実験支援によるネットワークサービスの検証技術. PhD thesis, 北陸先端科学技術大学院大学, March 2007.

[12] S Hemminger. Network emulation with netem. *Linux Conf*, April 2005.

[13] LuigiRizzo. Dummynet:asimpleapproachtothe evaluation of network protocols. *ACM SIGCOMM Computer Communication Review*, Vol. 27, No. 1, pp. 31–41, Jan 1997.

[14] 明石邦夫, 井上朋也, 安田真悟, 村本衛一, 知念賢一, 宇多仁, 篠田陽一. リンクエミュレータの多重実行の限界値測定. *Internet Conference 2009*, pp. 33–39, Oct 2009.

[15] 梅木孝志. マルチレベル背景トラフィック生成技術に関する研究. Master's thesis, 北陸先端科学技術大学院大学, 2008.

[16] 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.

[17] 千装利幸. 次世代エンタープライズネットワーク統合運用管理の実現手法に関する研究. Master's thesis, 北陸先端科学技術大学院大学, 2009.

[18] Philip Lieberman. Wake on lan technology, July 2002.

[19] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP). RFC 1098, April 1989. Obsoleted by RFC 1157.

[20] D.L. Mills. Network Time Protocol (NTP). RFC 958, September 1985. Obsoleted by RFCs 1059, 1119, 1305.

[21] W. Townsley, A. Valencia, A. Rubens, G. Pall, G. Zorn, and B. Palter. Layer Two Tunneling Protocol "L2TP". RFC 2661 (Proposed Standard), August 1999.

[22] D. Perkins and R. Hobby. Point-to-Point Protocol (PPP) initial configuration options. RFC 1172 (Proposed Standard), July 1990. Obsoleted by RFCs 1331, 1332.