第 XXVI 部

実ノードを用いた大規模なインターネットシミュレーション環境の構築

第 26 部
実ノードを用いた大規模なインターネット シミュレーション環境の構築

第 1 章　Deep Space One WG 2006 年度の活動

Deep Space One は大規模かつ現実的なネットワーク実験を行うための研究に取り組むワーキンググループである。実ノードを用いた StarBED や GARIT、仮想機械を利用した、セキュリティ実験のための VM Nebula などの実験設備の開発と運用を通して、実験の体系的分類、その支援手法などについての議論を進めている。

本年度の主な活動について報告する。

- StarBED および SpringOS の概要
- SpringOS Ver. 1.1 の現状
- GARIT と AnyBed の概要
- 次世代の大規模実証環境に必要な機能整理
- 既存のリンクエミュレータの評価

第 2 章　StarBED および SpringOS の概要

概要

現在のインターネットはすでに社会基盤として機能しており、この上で技術開発のための実験を行うことは不可能である。一方で、さまざまな目的のための実験を行いたいという要求がある。このような要求を満たすために、インターネットとは別の環境として、ソフトウェアシミュレータや実ノードによる実証環境が利用されている。

実環境との同一性を重視すると、実ノードにより大規模な実験トポロジを構築するのが理想的であるが、そのコストは非常に大きい。我々は、実ノードを用い大規模な実験トポロジを容易に構築できる環境として、大規模実証環境の要件を検討し StarBED を構築した。

本報告書では、各実証環境の特性について言及し、StarBED の概要と StarBED で利用されている実験支援システムである SpringOS について述べる。また、StarBED で行われた実験から、その有効性を明らかにする。

2.1 はじめに

従来のインターネットは実験的な側面を非常に強く持っており、実証環境としても利用されていた。しかし、現在のインターネットは社会基盤として利用されており、すでにさまざまな重要なサービスが運用されている。このようなサービスに影響をおよぼす恐れがあるため、もはや一利用者の判断によりインターネットを実証環境として利用できない。その一方、新たな技術の導入のための動作試験や、インターネット運用技術者の育成を目的として、インターネット上でさまざまな実験を行いたいという要求は高まってきている。

この矛盾を解決するため、インターネットと分離された疑似インターネット環境が求められている。このような環境を提供する手法の一つであるソフトウェアシミュレータはインターネット上のノードやネットワークの挙動を巨視的にシミュレートするソフトウェアである。ソフトウェアシミュレータを用いれば、さまざまな規模のネットワークを容易に構築することができ、構築したネットワーク上での各要素の制御も容易であるが、実際にインターネット上で利用される実装を利用できないことが多い。

一方、実ノードを用いて実験トポロジを構築すれば、実際にインターネットに投入される機器やソフトウェアを利用できるため、インターネットの挙動に近い環境を構築できる。しかし、機器の調達、物理的な接続や設定、ノード制御のためのコストがソフトウェアシミュレータを利用した場合と比較して大きい。このため、数台から十数台程度の実ノードを用いた小規模な実証環境が利用されることが多い。しかし、実環境との同一性を考えた場合、実ノードを用いた大規模な実証環境で容易に実験が行えることが理想的である。我々はこの要求を満たす大規模な実ノード

による実証環境である StarBED および、StarBED での実験を支援するためのシステムである SpringOS を実装した。計算を目的とした計算機クラスタは多数存在しているが、StarBED は実験を行うことを目的とした計算機クラスタである。StarBED を利用することで小規模な実証環境上では観測できなかった挙動の観測が可能になる。また規模的に実環境に近い実証環境を構築することができるため、実験の信頼性および実環境との同一性の向上が期待できる。

本報告書では、大規模な実証環境の必要性とその実装である StarBED および SpringOS について述べる。

2.2 大規模実証環境

本節では、ソフトウェアシミュレータや小規模な実ノードによる実証環境といった既存の実証環境について述べ、大規模実証環境の必要性を議論する。

2.2.1 既存の実証環境とその特性

ソフトウェア開発の各段階において、それぞれの環境の適応性は異なる。アイデアの検討など初期段階では、巨視的に新たな技術の性質を確認できるため、ソフトウェアシミュレータは有用である。プロトコルの手順の検証などは、実験対象以外の要素の挙動を隠蔽し、実験遂行者の想定する環境で検証が行えるため、ソフトウェアシミュレータが効果を発揮できる実験である。ある程度ソフトウェアシミュレータでの検証を行った後は、実環境に近い環境での実装の検証を行うことが望ましい。実ノードによる実証環境はこの段階で非常に有用である。これにより、実環境へ導入する実装の正当性や性能評価を行える。

図 2.1 にネットワークアプリケーション開発手順の一例を示す。開発者は、まずアイデアをソフトウェアシミュレータで確認し、ある程度効果があると認められれば、実環境用に実装し、小規模な実ノードによる実証環境で検証を行う。この段階で効果が十分あり、また既存のサービスに影響を与えないことを確認後、実環境に導入する。実環境に導入後、なんらかの問題が発生した場合は、その環境を小規模な実証環境上に再現後、問題に対する対応を行い、修正した実装を実環境に導入するということを繰り返す。しかし、検証用に利用した実ノードによる実証環境と、インターネットのような実環境との差異は、

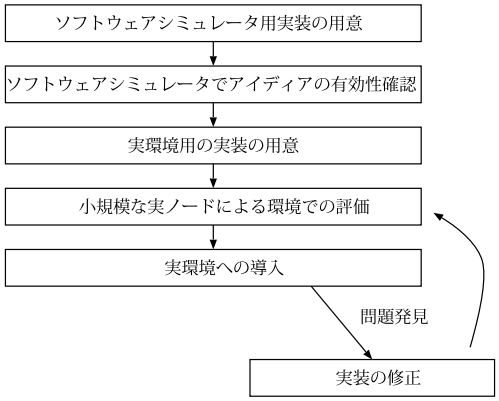

図 2.1.　ネットワークアプリケーション開発ステップの一例

規模性、多様性ともに大きい。

インターネットのような実環境に新たな技術を導入する際には、ノードやその集合であるネットワークのどのような性質が技術に影響するのか、また新たな技術が既存の技術に及ぼす影響の種類や規模の予想は困難である。ソフトウェアシミュレータは実験遂行者が前もって想定した環境で、決められた手順にしたがって正確に動作するが、各要素の動作を十分に予測できていなかった場合は、その結果が実際のインターネット上での挙動と異なる恐れがある。実ノードを用いたネットワークであれば、インターネットと同様に、それぞれのノード上のプロセスが自律的に動作し、それが集合することで、実験前には想定できなかった状況が生まれる場合もある。

しかしこれまではさまざまなコストの問題から、十数台から数十台規模以上の大規模な実証環境を構築するのは、コストなどの観点から困難であった。しかし、このような小規模な実ノードによる実証環境は、実環境と規模性、多様性においての乖離が大きい。このため小規模な実証環境での実験を行ったあと、実環境にアプリケーションを導入した際に、実験段階では予測できなかった問題が発生することがあった。このような問題は、小規模な実ノード実証環境と実環境の中間に位置する大規模実証環境で実験を行うことにより、事前に発見できる可能性がある。また、ソフトウェア開発のより早い段階で実ノード環境での実験を行えることも利点の一つである。実環境で用いられる実装を早い段階から評価できるため、実環境での新技術の特性をより詳細に認識でき、その結果を開発にフィードバックできる。

我々は、数百台の実ノードを一カ所に集めた環境

を構築し、それらのノードを用いた実証環境の構築および、実験遂行のための支援ソフトウェアを充実させることを考えた。これにより、実ノードによる実証環境の実環境との同一性と、ソフトウェアシミュレータの実験のしやすさという長所をあわせ持つ実証環境を構築できる。このような多数の実験設備と支援ソフトウェアが存在する実証環境を利用することにより、既存の実証環境構築技術よりも大規模な実験トポロジを柔軟に構築できる。また、支援ソフトウェアによりある程度ノードの設定を自動化できるため、多様性を向上させることも可能である。本報告書では、大規模な実験設備に実験トポロジの構築と実験の遂行を自動化するソフトウェアが用意された環境を**大規模実証環境**と呼ぶ。

2.2.2 大規模実証環境の設計

実ノードによる実証環境の構築および、実験の遂行コストが大きいことはすでに述べた。実証環境の規模が大きくなるほど、ノードの用意や物理接続に関するコストは大きくなる。我々はこれらのコストを軽減するため、多数の実験用実ノードをもつ施設を構築することで、実証環境を構築する度に実ノードを調達するコストの低減を図った。この施設を複数の実験遂行者で時分割および空間分割により共有利用することで、その経済的および人的な管理コストを低減できる。

大規模実証環境には、非常に多くのノードが存在するため、そのノードを一台ずつ管理し、実験トポロジの構築を行うことは困難である。この問題を解決するため、外部からの設定入力により、自動的にノード間接続を変更できる機器を利用する。このような機器には、VLAN や ATM により仮想的にトポロジを変更できるスイッチや、物理的に接続を変更できるスイッチが利用できる。

これまでで挙げた手法により、実ノードの用意およびトポロジ構築のためのコストは軽減できる。しかし、ノードへのソフトウェアの導入や実験シナリオの遂行にも、大きな人的および時間的リソースが必要となる。また人為的ミスにより、実験精度の低下を招く可能性は高い。これを解決するため、実験遂行者の意図する実験トポロジの自動的生成および、実ノードへ自動的に OS やアプリケーションを導入する支援システムを用意し、実験遂行者は実験の遂行の大部分を支援システムにまかせることで、人為的な実験精度の低下を防ぐ。

本節では、このような環境を想定した実験の遂行手順について考察し、実験設備に必要なハードウェア要件および、支援ソフトウェア要件について議論する。

2.2.3 実ノード環境での実験の遂行手順

一般的な実ノードによる実証環境での実験の遂行手順について、支援システム無しの場合を図 2.2 a)（アラビア数字）に、支援システム有りの場合の手順を図 2.2 b)（ローマ数字）に示す。

それぞれの手順について以下に示す。

1）**実験トポロジやシナリオの検討** 実験の目的により適切なネットワークトポロジや各ノードの性能要件および導入するソフトウェアを検討し、さらにその環境上で実行されるべきシナリオを検討・決定する。

2）**必要な実ノードなどの用意** 検討の結果から必要な実ノードやネットワーク機器を用意する。

3）**各機器の物理的な接続** 用意したネットワーク機器を物理的に接続する。

4）**実ノードへのソフトウェアの導入** 実験用ノードへ必要な OS やアプリケーションを導入し、必要な設定を行う。

5）**スイッチなどネットワーク機器の設定** 必要であれば、スイッチやネットワーク機器の設定を行う。

6）**構築した環境上でのシナリオ実行** 構築された環境上で実験シナリオを遂行し、実験データを収集する。

7）**ログの解析** 得られた実験データを解析する。

2 の手順についてはすでに提案した、前もって多数のノードを用意しておくという方法で解決できる。支援ソフトウェアは何らかの形で実験実行者による

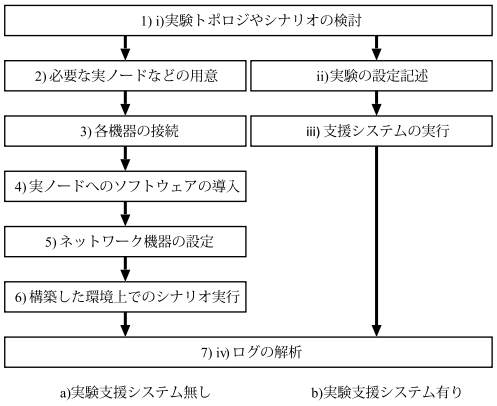

図 2.2. 実験の遂行手順

実験設定を、入力として受け取り、それにしたがい実験を遂行する。この時の実験手順は、図 2.2 b) に示した通りであり、実験遂行者は実験の大部分を支援システムにまかせることで、実験による拘束時間を短縮することができる。これにより、実験遂行者は行う実験の検討およびその結果の解析に注力できる。また、実験は前もって記述された手順にしたがって機械的に遂行されるため、人為的なミスの低減および、再現性の向上にも効果がある。

支援ソフトウェアが行う処理は、実験用に用意されているリソースから、実験トポロジを構成するために必要なリソースの選択および獲得、実験トポロジの生成、実験シナリオの実行である。

2.2.4 実験設備への要件

まずは実証環境として動作させるための基本的な機能として、実験設備への要件をあげる。

管理用ネットワークの分離　実験用のネットワークと、管理用のネットワークを分離する。これにより実験の通信と制御用通信の相互の影響を防ぐ。

リンク特性の模倣　リンクの遅延やパケット損失、ディレイなどのリンク特性を生成するための機能が用意されていることが好ましい。専用の機材を利用する方法や、広帯域の対外線を用意し、実環境の特性を取り入れる方法がある。

ノードへのコンソール操作　ネットワークが切断された際に、実験ノードへ接続し、問題点の特定や復旧を行うため、各ノードのコンソールを一カ所から、容易に操作する機能が必要である。

自動的な実験トポロジの自動構築　すでに述べたように、自動的にネットワークの構築を行える機材が必要である。

2.2.5 支援ソフトウェアへの要件

実験支援ソフトウェアへの要件を以下にまとめる。

実験遂行者による設定の認識・解析　実験遂行者による実験ノード設定および実験トポロジについての設定を認識・解析する。この結果から他の機能と連携し、実験トポロジを構築し、実験シナリオを遂行する。

リソースの状態・属性管理　利用可能な実ノードや実験トポロジを構築するためのリソースの状態およびそれらのリソースの属性を管理しておき、その情報をもとに実験遂行者へ実験に必要なリソースを割り当てる。属性にはノードのネットワークインターフェイスの種類や数、アーキテクチャや標準でインストールされている OS などがある。実験トポロジを構築するためのリソースとして、VLAN 番号などがある。複数の実験遂行者による空間分割利用のため、排他処理が必要である。

実験ノードの自動設定　実ノードに OS やアプリケーションを導入する機能である。これにより各実ノードは実験用ノードとして振る舞う。

実験トポロジの自動構築　自動的に実験トポロジを構築する機能である。設定記述にしたがって、ネットワーク機器の設定を行う。

シナリオに従った実験の自動遂行　設定記述にしたがって、実験を遂行するための機能である。実験遂行者による各ノードのプログラムの起動の人的・時間的コストを削減できる。また人為的な実験ミスを低減する効果もある。

実験ノードの状態管理/表示　各実験ノードがどのような状態にあるかを理解しやすい形式で実験遂行者に知らせる。

実験ログ収集　実験終了後、各実験ノードからログを収集する。実験の規模が大きくなるほど、ログ収集のための人的および時間的なコストが高くなる。

リンク特性の模倣　ハードウェアの項ですでに述べたが、Dummynet[201] や NIST Net[175] などのソフトウェアでの実現も可能である。

また、何らかの方法で実ノードの電源管理を行える手法が必要である。IPMI[97] や Magic Packet Technology[142] を用いた Wake on LAN などを利用して、ハードウェアレベルで実ノードの電源を ON/OFF できる機器はさまざまなベンダによって発表されている。またソフトウェア的に電源の管理を行うためには SNMP などを利用することができる。

2.3 StarBED

我々は、前節の内容をふまえて大規模実証環境の一実装として StarBED[247] を提案した。StarBED は通信・放送機構[1]により北陸 IT 研究開発支援センターとして 2002 年に開所され、2006 年 4 月から同

1　現情報通信研究機構

機構北陸リサーチセンター[2]として運営されている。本節では StarBED について述べる。

2.3.1 StarBED の構成概要

StarBED は 512 台の実ノードと、VLAN および ATM の VC/VP を用いて仮想的に変更することで、実験トポロジを構築する。StarBED の概念的なトポロジを図 2.3 に示す。

StarBED において実験トポロジは、VLAN および ATM の VP/VC を用いて設定可能である。実験トポロジの設定や、実ノードへの設定は管理用ネットワークを通して行われる。管理用ネットワークは、管理用トラフィックの実験への影響を防ぐため実験用ネットワークと分離されており、各実験用ノードは管理用と実験用の 2 つのネットワークに接続されている。実験を支援するためのファイルサーバや DHCP サーバ、支援システムの一部のモジュールは管理ネットワークに接続されたノードで動作する。WIDE プロジェクトの 10 Gbps のネットワークおよび、Japan Gigabit Network（JGN）の 10 Gbps のネットワークに接続されており、必要であればこれらのネットワークを通じてインターネットへ接続できる。このネットワークを利用して別のサイトへの接続や、実トラフィックの導入に利用できる。各ノードのコンソールは Raritan 社 [196] の製品により、一台の端末から操作できるため、ネットワークからの制御が不可能になった場合にも対応できる。また、さまざまな実験を行うため、仮想トラフィック生成装置や、パケットスニファなどのハードウェアも用意されている。実験遂行者の持ち込みハードウェア専用のラックも用意され、柔軟に実験に対応できる。

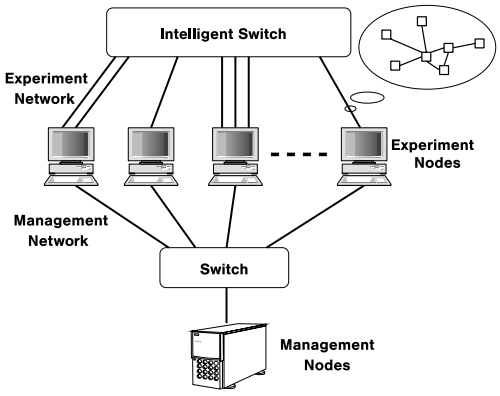

図 2.3. StarBED の概念的トポロジ

2 http://starbed.nict.go.jp/

前述の通り初期の StarBED はの 512 台の実験専用の PC が用意された。その後 2006 年 4 月にさらに 168 台の PC が追加し、現在 680 台の PC を実験に利用できる。StarBED では、これらの PC のネットワーク接続を VLAN および ATM の VC/VP を用いて仮想的に変更することで、実験トポロジを構築する。StarBED の概念的なトポロジを図 2.3 に示す。

2.3.2 仮想機械の利用

我々の想定する実証環境は基本的に多数の計算機を集め、1 計算機を実験トポロジ上の 1 ノードとして動作させることで大規模な実験トポロジを構築する。存在する計算機では実現できない規模の実験トポロジを構築するために、その都度、新たにノードを用意することは、経済的費用、設置するための空間、そして計算機の保守費用などさまざまなコストが必要となるため現実的ではない。この解決策として、1 台の計算機を仮想的に多重化することで、さらに大規模な実験トポロジを構築する手法がある。すでに用意されている計算機を仮想的に多重化すれば、新たな計算機を用意するためのコストは発生せず、計算機の保守費用は増加しないため、新たな計算機を用意する場合よりも、さまざまな点でコストが小さい。

我々が想定する実証環境は実環境用の実装を大規模な実験トポロジ上で検証することが目的である。したがって、実験遂行者が要求するソフトウェアが変更を加えることなく動作することが非常に重要である。汎用的な OS が動作しない場合は、実験遂行者が意図するアプリケーションの動作に支障をきたす場合や、実験遂行者が変更を加えた OS を利用できないなど、問題が生じる場合がある。仮想ノード実現技術の一つである仮想機械は、ハードウェアレベルで計算機を模倣する。したがって、汎用的な OS が変更なしに動作し、ソフトウェアレベルでは実環境と同一の挙動を示す仮想ノードを利用できる。ただし、仮想機械の性質や性能に影響を受けるため、性能検証や、実ノードのデバイスドライバなどといった計算機自体の機能に関する検証には適さない。

一般的に、仮想機械より高い抽象度で仮想ノードを実現する技術では、専用の OS を利用し仮想ノードの動作速度を向上させているか、OS の模倣を行わずプロセスなどのレベルでの多重化を可能にする。こ

のような技術を利用すれば、更に大規模な実験トポロジを構築できるが、実環境との同一性は低下するため、実験遂行者は実験の性質を十分に吟味し、仮想ノードを利用できるかどうかを決定する必要がある。

StarBED にはノード多重化のため、仮想機械の実装の一つである、米 VMware 社の VMware[261] を導入し、数千台規模の実験を可能とした。

2.4 SpringOS

SpringOS は StarBED で利用されている実験支援システムの総称であり、個別の機能を持つモジュール群により構成されている。各種設定を記述したファイルを用意するだけで、実験遂行者の意図する実験が、SpringOS により自動的に遂行される。SpringOS の主な機能を以下にあげる。

- 実験遂行者による設定の記述の認識・解析
- リソースの状態および属性管理
- ノードへの OS の導入および各種設定
- スイッチへの VLAN 設定による実験トポロジ構築
- シナリオを管理するシナリオマスタと、シナリオを実行するシナリオスレーブとが協調することによるシナリオ実行

電源管理については Wake on LAN および SNMP を用いて実現している。また、シナリオ遂行機構を用い、Dummynet や NIST Net を起動することによるリンク特性の模倣を、ファイル転送用プログラムを起動することで実験ログの収集を実現している。

2.4.1 ノードへのソフトウェアの導入

ノードのソフトウェア導入には、前もって実験遂行者が作成したディスクイメージをノードのハードディスクに書き込むことで行う方法と、ディスクレスシステムで各ノードを動作させる方法に対応している。ディスクイメージを生成する支援モジュールも用意しており、一台のノードに必要なシステムを構築すれば、その環境を多数のノードに複製できる。ディスクレスシステムとして起動する場合は、起動が非常に短時間で行える。ディスクレスシステムを構築する方法には、ディスク部分を TFTP を利用し取得後メモリ上におく方法および、NFS を利用する方法が利用できる。

実験ノードへのソフトウェア導入の詳細については、[313] にまとめられている。

2.4.2 シナリオの自動遂行

実験のシナリオも設定ファイルに記述されることを前提とする。また、シナリオが記述された設定ファイルを読み込み、シナリオ全体を管理する要素をシナリオマスタと呼ぶ。

シナリオマスタは実験遂行者の記述したファイルを読み込み、実験前に各実験ノード用のシナリオを配布する。各実験ノードが別のノードの挙動をトリガとしてイベントやプログラムを実行するといったノード間同期が必要な場合はシナリオマスタを通じ、メッセージパッシングを用いて行う。OS やアプリケーションの導入および設定が済んだノードでは、実際にノード上でコマンドを実行するシナリオスレイブが起動する。シナリオマスタは定期的にノードへ接続を試み、シナリオスレイブが起動後、接続が成功すると、シナリオスレイブはシナリオマスタから実行すべきシナリオを受け取りシナリオ遂行を開始する。

シナリオスレイブはシナリオに記述されたプログラムの起動や、ノードの同期のためシナリオマスタへメッセージの送信および、シナリオマスタからのメッセージを受信するまで、シナリオ遂行を停止するといった処理を行う。また、シナリオマスタは各ノード同期のためのシナリオを実行する。シナリオの実行についての詳細は [31, 315] にまとめられている。

2.4.3 設定記述

一般的に一つの実験で同様の設定のノードが多数利用されることが多いため、我々が実装した設定記述ではオブジェクト指向的なクラスの概念を導入した。これにより同一の設定の多数のノードを容易に宣言することが可能である。

設定記述は設定部とシナリオ部に別れており、設定部ではノードのブート方法や、ディスクを利用する場合はディスクイメージの指定、ディスクイメージを書込むハードディスクのパーティション、必要なネットワークインターフェイスの種類、ノードで実行されるシナリオなどが記述される。

シナリオ部にはノードのクラス設定部に記述されるノードシナリオと、ノードシナリオの協調のためにシナリオマスタが実行するグローバルシナリオがある。シナリオ設定部は senario 宣言で始まるブロックに記述される。ノード間協調は実験ノード上で動作するシナリオスレーブと、管理用ノード上で動作

するシナリオマスタのメッセージ交換によって実現される。ノードシナリオは、基本的にノードで実行されるコマンドのリストであり、他のノードとの協調が必要な場合のみ、メッセージ交換のための記述がされる。グローバルシナリオはノードシナリオの協調のために利用されるため、メッセージ交換の制御が主な内容となる。

これらの設定記述の例を図 2.4、図 2.5、図 2.6 に示す。この例はノードを 2 台用意し、一台で netserv を起動し、もう一台のノードで netserv が起動しているノードに対して netperf を実行するだけという簡単なものである。図 2.4 はノードクラスの宣言とインスタンスの生成、図 2.5 はネットワークに関する設定、そして図 2.6 はグローバルシナリオの例である。

図 2.4 では、svclass という名前のノードクラスを宣言している。このクラスのノードのディスクイメージである img.gz は、FTP を利用して 172.16.1.1 から取得され、パーティション 2 に書込まれる。また Ethernet のネットワークインターフェイスが一つだけ利用される。シナリオ部には、ノード起動後 netperf のサーバプログラムである、netserver を起動し、シナリオマスタに “serverstarted” というメッセージを送信、その後シナリオマスタから “quit” というメッセージを受け取ると netserver のプロセスを終了する内容のシナリオが記述されている。最終行で、server という名前のインスタンス群を 1 台生成している。

図 2.5 では、ethclass という名前のネットワークのクラスを宣言している。IP アドレスのレンジをここで指定することで、参加したネットワークインターフェイスに自動的にアドレスを設定できる。netset 宣言で ethnet という ethclass のインスタンスを生成している。attach でノードのインターフェイスをこのネットワークに参加させることで、ネットワークを形成する。

図 2.6 はグローバルシナリオの一例である。このシナリオでは、開始後、server[0] からの “serverstarted” というメッセージと client[0] からの “clisetupdone” というメッセージを待つ。この 2 つのメッセージを受信すると、client[0] へ server[0] の IP アドレスをメッセージとして送信する。その後 client[0] から “cdone” というメッセージを待ち、受信すると、server[0] へ “quit” というメッセージを送信し終了する。

SpringOS は実ノードおよび VMware による仮想機械を制御可能であり、ある実験に必要な数の実ノードが確保できない場合や、実環境との同一性を吟味し、仮想機械で十分要求が満たせる場合は、仮想ノードを利用し実験トポロジを大規模化できる。

SpringOS の設計および詳細については、[150] にまとめられている。

```
nodeclass svclass {
  partition 2
  ostype "FreeBSD"
  diskimage \\
    "ftp://usr:pas@172.16.1.1/img.gz"
  netif media fastethernet
  scenario {
    wake "/sim/netserver" \\
      "/sim/netserver"
    send "serverstarted"
    recv msg
    msgswitch msg {
      "quit" {
        wakewait "/usr/bin/killall"\\
          "killall" "netserver"
        exit
      }
    }
  }
}
nodeset server class svclass num 1
```

図 2.4. ノード定義

```
netclass ethclass {
  media fastethernet
  ipaddrrange "192.168.3.0/24"
}
netset ethnet class ethclass num 1
attach server.netif[0] ethnet
```

図 2.5. ネットワーク定義

```
scenario {
  sync {
    msgmatch server[0] "serverstarted"
    msgmatch client[0] "clisetupdone"
  }
  send client[0] \\
    haddr(server[0].netif[0].ipaddr)
  sync {
    msgmatch client[0] "cdone"
  }
  send server[0] "quit"
  exit
}
```

図 2.6. グローバルシナリオ定義

2.5 StarBED で行われた実験の考察

StarBED は多くの研究に利用され、さまざまな成果を残している。我々は、これまでの運用の結果から実証環境に求められる性質について議論した。本節では、まず実証環境の性質について述べ、行われた実験について分析し、実験の特性と StarBED との適応性について考察する。

2.5.1 実験の特性

まず実証環境の特性について述べる。

2.5.1.1 実環境との同一性

実ノードを用いた環境の最大の長所は実環境と同様の実装を利用できることである。実ノードを利用しているため、実環境との同一性はある程度確保されてはいるが、より実環境に近づけるための工夫が必要である。

実時間性 実ノードを用いた環境では実験は実時間で進む。実験の遂行時間自体は、実験の複雑さに関わらず利用者が想定した時間となる。

規模追従性 大規模検証環境には、少なくとも数百台から数千台程度のノードが必要であると考えている。この数字は一組織が実験のためだけに保有する環境よりも大規模であり、さらにある程度の組織のネットワーク環境を模倣するだけに足る環境である。

実験ノードすべてを実ノードを利用する方法が基本であるが、実証環境構成要素の挙動を失わないという条件で、仮想ノード実現技術を利用できる。仮想ノード実現技術の利用に関しては後述する。

実証環境構成要素の挙動 大規模実証環境では、実環境と同様のハードウェアおよびソフトウェアを利用できるため、その挙動の実環境との同一性は高い。しかし、さまざまな種類のハードウェアおよびソフトウェアを用意することは困難であるため、ある程度の制限はある。

2.5.1.2 実験遂行の容易性

実証環境構成要素の制御の粒度 実ノードの制御方法は基本的に通常の操作手順と同様に、コマンド実行によるプロセス単位の制御となる。

実験に必要となる時間 実ノードを用いた環境では、実験自体に必要な時間はシナリオに記述された時間である。

実験の準備に必要となる時間 実ノードの設定は、ソフトウェアシミュレータを利用した実験に比べてコストが大きい。OS の導入や各種設定を行うためにはある程度の時間が必要となる。実験トポロジが大規模になるほど必要な時間は長くなるため、支援システムにより容易に準備を行うためのしくみが必要である。

実験パラメータの変更 実験のパラメータ変更はソフトウェアシミュレータなどでは容易である。しかし、実ノードを利用した場合はパラメータの種類にもよるが、ある程度コストがかかる。特にノード数が大きくなった場合にはそのコストは爆発的に増大する。この点に関しても支援システムによる支援が必要である。

2.5.1.3 実験の再現性

実ノードを用いた実証環境では、実験の再現性を求めるのは難しい。イベントの発生などは基本的にコマンド入力で行い、それによる各要素の状態変移はその時点での状態の差違によって異なる可能性があるためである。再現性を確保するためには以下の性質に注意せねばならない。

実証環境構成要素の挙動 実験を繰り返した際に、実証環境を構成する構成要素の動作が、毎回、正確に同一であることは重要である。しかしノードや構成要素の状態の微少な違いにより、実証環境全体としては大きな差を生む可能性がある。しかし、これは実環境でも同様であり、ある程度許容すべきである。

イベント発生の正確性 前述の通り人手によるコマンド実行などは非常に精度に欠ける。支援システムにより、前もって指定されたタイミングで指定されたコマンドを実行する支援システムが必要である。

2.5.1.4 実験の検証容易性

ソフトウェアシミュレータでは事前に指定されたコマンドが指定されたタイミングで実行される。したがって実験後にどのタイミングでどのようなイベントが実行されたかは明確である。しかし、実ノードによる環境では通常そのような機構は存在せず、また、イベントが正しく実行されたかどうかを確認

表 2.1. 実証環境の特性と実験の適応性

	実環境との同一性			遂行の容易性				再現性		検証容易性	
	実時間性	規模追従性	構成要素の挙動	制御粒度	実験時間	準備時間	パラメータの変更	構成要素の挙動	イベント発生の正確性	操作履歴	実験要素の状態変移
ソフトウェアシミュレータ	×	△	×	ネットワーク、ノード、プロセス、パケットなど	△	○	○	×	○	○	○
小規模な実ノードによる実証環境	○	×	○	プロセス	○	×	×	○	×	×	×
大規模実証環境	○	△	○	プロセス	○	△	△	○	△	○	△

することもできない。操作履歴とそれによる実験要素の変移を実験終了後に把握できなければ、実験の正当性を確認できない可能性がある。これについても支援システムである程度サポートする必要がある。

2.5.2 行われた実験の分析

[178] では、階層型 IP トレースバック機構を提案、実装し、小規模な実ノードによる環境での動作検証が行われており、より実環境に近い大規模な実験トポロジ上でのデータの取得のため StarBED の利用を検討している。実環境用の実装を用いる必要があったため、ソフトウェアシミュレータの利用は困難である。また、多くの AS 間の通信を模倣した大規模な実験トポロジが必要であるため、研究室レベルでの実験トポロジ構築は困難である。運用環境を意識しての検証であるためこれは、構成要素の挙動の同一性と、環境の規模の問題から規模追従性が重要であった。

[95] では、Multi-player Online Games（MOGs）への peer-to-peer 型通信の導入を提案している。小規模な環境での動作検証後、実用化に向けさらに大規模な環境として StarBED での動作検証を行っている。実環境用のソフトウェア利用と約 300 ノードを利用した実験トポロジの構築は、ソフトウェアシミュレータ、研究室レベルの実ノードを用いた環境での実現は困難であり、測定する対象が実時間であったことから、構成要素の挙動の同一性と規模追従性と実時間性が重要であった。

[160] は、マルチキャストを用いたグループ通信が ISP のバックボーンに対しどのような影響を与えるのかを検証するため、StarBED においての仮想ノードを含め 1000 ノードを越える実験トポロジを構築し実験を行った際の工夫や知見をまとめたものである。この時、仮想ノードはグループ通信のユーザによるトラフィック生成部分に利用し、コアネットワーク部分は実ノードにより構成された。また、この際には、奈良先端科学技術大学院大学へ JGN を介して接続しトラフィックを迂回させ、実ネットワークの遅延などの特性を実証環境に取り入れた。この実験には、規模追従性および挙動の同一性が重要であった。

また、これ以外にも、さまざまなスイッチのマルチキャストトラフィックの転送能力の検証や、TCP の挙動の観察、大規模なトポロジ上でのトラフィック解析、有名ベンダのルータとオープンソースで開発されているソフトウェアルータの性能比較などが StarBED を用いて行われた。

これらの実験では、実環境に利用するための実装を、実ノードによる大規模かつ現実的な実験トポロジ上で検証したいという要求が大きい。これはソフトウェアシミュレータ、研究室レベルの実ノード環境では実現できない。また、実時間でのデータ測定が必要である場合も多い。当然、実験を容易に行えるためのしくみはすべての実験で重要である。しかし、これまで行われてきた実験では、実験を容易に行えるという点よりも実環境との同一性が重要視された。

また、VPN や VLAN などの技術を用い、StarBED の各ノードをインターネットのさまざまな位置に出現させ、その位置のトラフィック情報を計測することにより、さまざまなプロトコルの系としての動作の観察を行うような実験も計画されている。

これらの結果をふまえて、ソフトウェアシミュレータ、小規模な実ノードによる実証環境、大規模実証環境の性質を表 2.1 に示す。大規模実証環境 StarBED は、従来の研究室レベルの実ノードによる環境では不可能であった、数百台規模での実ノードを用いた実験トポロジの容易な構築とその上でのシナリオ遂行を可能とすることでこの要求に応え、その有効性が証明された。

2.6 関連研究

X-Born[275] や Planetlab[193] は、分散したサイトの実ノードを接続し、大規模な実証環境を構築するものである。我々が提案している環境は、一つのサイトに多くの計算機を用意することにより実現するものであるため、比較対象とはならない。以下では、Netbed と ModelNet について考察する。

2.6.1 Emulab/Netbed

Netbed[271] は、分散システムと実ノードによる環境およびシミュレータの統合環境であり、豊富な実験管理機能をもつ。変更可能なパッチパネルをソフトウェアにより制御し、物理的な結線を変更しトポロジを変更できる。この機能と VLAN をあわせて利用することで柔軟な実験トポロジを構築できる。ns-2 を実ネットワークに接続できるように拡張した nse[249] を利用し、ソフトウェアシミュレータと実ノードによる環境を接続しており、ns-2 を利用している部分では、前述の通り実環境と同一の実装は扱えない。

ある程度の規模の PC クラスタをもつサイトを接続することで、大規模な環境を構築しており、実験トポロジがさまざまなサイトに分割されることがある。これにより、サイト間の接続の問題が実験に影響をおよぼす可能性がある。また、何らかの問題が発生した場合に、実験対象とサイト間の接続部分のどちらに問題があったのかを切り分けるのは困難である。

StarBED は一般的な計算機を一カ所に集めることで、管理コストを低減し、また、別ネットワークから完全に分離された大規模な環境を提供することができる点が Netbed とは異なる。

Netbed の利点として実験ネットワーク中に実ネットワークの遅延を導入できることが挙げられるが、一カ所に集中した施設であっても、遅延などをエミュレートできる。むしろ、分散配置による制御の遅延や帯域の制限などが問題になると予想できるが、開発中の環境をインターネットに接続することになるため、その脆弱性も問題である。

Netbed のシステムは、実験遂行者からのリクエストを受け付け、施設が空き次第実験を遂行する。実験は一括処理で遂行されるため、実験を行いその状況を判断して次の実験を遂行したい場合や、教育を目的とした場合などの実験トポロジをある状態まで自動的に構築し、その後は実験遂行者の手動により操作したい場合には不向きである。

2.6.2 ModelNet

ModelNet[257] は実ノードと同様のソフトウェアが動作する Virtual Node を一台の実ノード上で多重化し大規模なトポロジを構築し、さらにコアノードでこれらの Virtual Node 間のリンク特性を変更するためのソフトウェアである。

Virtual Node を利用しているため各ノードの物理的な特性の検証や、物理的な特性に関連するソフトウェアなどは利用できず、性能測定用途にも利用は不可能である。また Virtual Node 自体が実ノードとどの程度の同一性をもっているかの検証がなされていない。

コアネットワークで、多数のパスに対しリンク特性をエミュレートするため、規模耐性の問題がある。

ModelNet はソフトウェアであり、ハードウェアとソフトウェアの両面から検討がなされている StarBED とは異なる。また、複数の実験遂行者による共有利用については考慮されていない。

2.7 まとめ

インターネットのような大規模な実環境に、新たな技術の実装を導入する前には、できるだけインターネットに近い環境で検証を行うことが望ましい。これまでの小規模な実ノードによる実証環境で行った実験結果は、場合によっては、インターネットのような大規模で多様的な実環境での挙動と大きく異なる場合がある。また、小規模な環境では観測し得ない計測結果を、大規模な環境では捕らえることが可能である。

我々は、このような問題を解決するため、大規模実証環境を提案し、実際に実装するにあたりさまざまな検討を行った。大規模実証環境を利用すれば、実ノードによる大規模な実験トポロジの構築および、シナリオの遂行を低コストかつ容易に実行できる。これにより、実ノードを利用することによる実環境の同一性を持ち、ある程度大規模な実証環境が実現可能となる。

本報告書では、大規模実証環境の一実装である StarBED の設計と、支援システムである SpringOS の動作について述べた。その上で、これまでに StarBED

で行われた実験について考察し、その他の方法では実現困難な実験例から、我々の提案した実証環境の有効性を示した。

第 3 章　SpringOS Ver. 1.1 の現状

DeepSpaceOne ワーキンググループでは、インターネットに関する技術や製品の評価実験に向けた研究開発に取り組んでいる。本報告書では、その一環として開発している、インターネットに関する実験を支援するプログラムパッケージ SpringOS Version 1.1 の現状を報告する。

新たなモデルやモジュール構成が議論されているため、将来変更される可能性があることをご留意願いたい。

3.1 SpringOS のねらい

SpringOS のねらいは以下のようになっている。

1) 設定と進行の自動化
   - 人為的ミスの軽減
   - 繰り返し実行を再現
2) 作業者の拘束時間の削減
3) 複数実験の同時進行
4) 汎用性
   - さまざまな施設で稼働する
   - P2P やセキュリティーなど、特殊な用途を想定しない

各項目を達成するため、我々はさまざまな実験支援モデルやモジュール、その間のプロトコルを設計し、プログラム群を開発した。

3.2 モデル

インターネットに関する技術や製品を評価する者（実験者）は、実験支援システムに実験に関するさまざまな指定（実験記述）を投入し、その結果を得る（図 3.1）。

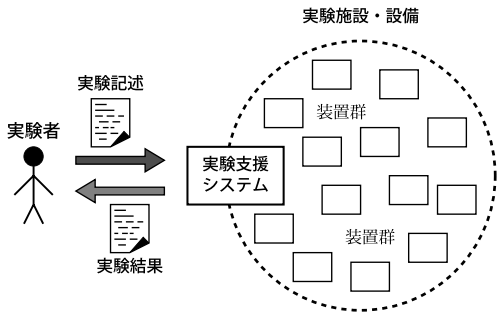

図 3.1. 実験支援システムを介したネットワーク実験

実験記述はノードやネットワークなどの構成要素の仕様、そして各ノードの挙動からなる。

実験記述 = ノード仕様
　　+ ネットワーク仕様
　　+ 各ノード挙動

支援システムは記述したノード仕様から、実験に用いるノード（実験ノード；今回は PC を想定）を定める。ノード仕様は所有ネットワークインターフェイスや起動ディスク種別が指定可能である。

ネットワーク仕様とは使用メディアや構成を指す。スイッチの設定手順を考えると、先に定まった利用ノードのネットワークインターフェイスからノードが接続されているスイッチ群が導かれ、ネットワーク構成（トポロジ）の設定が可能となる。ただし、ノードは複数のインターフェイスを持つため、支援システムがノードの所属するネットワークを自動的に導出することは困難である。したがって、それぞれのインターフェイスが所属するネットワークの指定、すなわち、ネットワーク構成に関する記述も必要である。ネットワーク構成を得ることで、スイッチへの具体的な指示が生成可能となる。

各ノードでは実験内容に応じて、さまざまな処理が必要である。このノードの挙動を、SpringOS ではシナリオと呼ぶ。

3.2.1 仕様要求にもとづいた資源割り当て

SpringOS ではノードやスイッチなどの装置、そして VLAN 番号などを資源と呼ぶ。物理的な実体だけでなく、論理的な資源もある点に注意願いたい。

複数の実験が同時に進行すると、資源利用の衝突が起きるおそれがあるため、その利用を排他的に制

限する必要がある。そこで、SpringOS では資源を管理するモジュール（後述）を設けた。

複数の実験が同時に進行するため、ノードをホスト名、IP アドレス、MAC アドレスなどで指定する形態を採用した場合、頻繁な資源の衝突が予想される。実験者間で何らかのルールを設けて利用対象を工夫することは可能だが、実験者から指定する以上は衝突の可能性を本質的に回避することはできない。また、資源が膨大になった際にはルールの周知や操作のミスの可能性も増す。

そこで、我々は実験支援システムから与えられた資源を用いて実験を遂行する形態を採用した。もちろん、実験支援システムが実験者に必要な資源を想像することは困難であるため、実験者が指定した仕様に合わせて、資源を割り当てる。すなわち、実験記述中のノードの数やネットワークインターフェイス数などの仕様から、資源管理モジュールへ資源を要求し、資源管理モジュールは利用可能な資源を割り当てる。

3.3 処理とモジュールの紹介

SpringOS は多数のモジュールを含むが、それらは実験駆動、ノード上の実験補助、ノードの初期化、施設側設定操作、監視の 5 つに大別される。表 3.1 に代表的なモジュールをあげる。

実験駆動：

他のモジュールと協調して実験を駆動する。実験記述の解釈、資源獲得、ノード起動、シナリオ実行が主な処理内容となる（後述）。

ノード上実験補助：

実験ノード上で実験を補助する。ネットワークインターフェイスの走査など、繁雑な操作を吸収する。

ノード初期化：

実験によっては実験に用いるソフトウェアが大規模となり、OS ごとインストールする場合もある。そのような場合に、ディスク内容のバックアップやレストアを行う。

施設側設定操作：

実験ノードへ与える各種設定の多くは施設側の管理サーバに蓄えられている。実験内容に応じてそれらの設定を変更するため、ノードだけでなく施設側でも設定変更のモジュールが必要である。

表 3.1.　SpringOS モジュール一覧

種別	名前	摘要
実験駆動	kuroyuri	実験駆動プログラム（実験記述言語評価器）
ノード上実行補助	coil	ブートローダ
	ifsetup	インターフェイス探査
	mine	ノード停止
ノード初期化	ni	ノードディスク操作（バックアップ・レストア）
施設側設定操作	dman	ディレクトリ操作（起動パラメータ操作）
	erm	資源管理
	fncp	ノード設定誘導
	wolagent	電源投入
（施設側）監視	sheepdog	死活・健康監視
	xant	トラフィック監視

監視：

多数の資源を用いている際にはその進行状況を把握するのは困難である。その進行状況把握を容易にするため、監視モジュールを設けた。

3.3.1 ノード起動と停止

SpringOS は PXE で実験ノードを起動する。PXE を用いて PC を起動すると、PC は TFTP を用いてディスクの Master Boot Record（MBR）に相当するブートローダをネットワークから取得して、そのブートローダを実行し起動を遂行する。したがって、このブートローダを置き換えることで、さまざまな OS を起動できる。内蔵ディスクのパーティションに格納されているブートローダを呼び出せば、そのパーティションに格納されている OS が起動される（階層的な起動）。あるいはネットワークブートのディスクレスな OS も起動可能である。

モジュール dman は tftpd が提供するファイル群を操作するモジュールで、このノードの OS の変更を実現する。モジュール coil は内蔵ディスク上の OS を起動するブートローダで、指定されたパーティションのブートローダを呼び出す。また、モジュール ni はディスクレスの FreeBSD 上で稼働するため、dman の支援が必要である。

PXE を起動するための電源投入は Wake on LAN（WoL）を用いる。モジュール wolagent は WoL の起動パケットを生成する。

モジュール mine はノードの停止を担当し、SNMP の要求に応じて、ノード上で shutdown プログラム

を実行する。また、reboot プログラムの実行も対応している。

3.3.2 ノードのソフトウェア・インストール

実験ノードが増えると、OS を含めたソフトウェアのインストールに多大な時間を消費する。SpringOS ではノードのソフトウェア・インストールに二つの機構を設けた。一つはディスクパーティション単位の操作で、パーティション内容をネットワーク上のサーバへ退避・復元する。あるノードの内容を他のノードへ繰り返し書き出すとソフトウェアを複製できる。OS のような大規模なソフトウェアはこの方法が適している。たとえば、約 4 GB のディスクイメージであれば、約 100 ノードに 6 時間程度で複製できる。モジュール ni はこのディスクパーティションの退避・復元を行う。

もう一つは、後述するシナリオ実行時に特定ファイルをネットワークから転送する。設定ファイルの変更・修正や特定の実験対象プログラムの更新には、こちらの方法が適しているだろう。

3.3.3 ネットワーク構築

物理的な配線は繁雑、かつ、人為的なミスを招きやすいため、我々は実験ネットワークの構築は VLAN で行う。実験記述中のネットワーク仕様記述の度に VLAN を作成する。

ネットワーク特性エミュレーション

ネットワーク実験ではインターネットの距離感を表現する典型的な手段として、構成要素間に遅延、ジッタや帯域の挿入が行われている。このような実験を支援するため、SpringOS ではネットワークの仕様記述に遅延やジッタなどのネットワーク特性を指定可能とした。そして、これらの特性の実現のため、dummynet や NetEm を制御する機能を持つ。ただし、現行では dummynet や NetEm を利用する際に 1 ノード消費するため、ネットワーク特性エミュレーションの箇所を多数指定すると、大量のノードを消費する。

3.3.4 資源管理

前述のように、SpringOS は実験者の記述した仕様にしたがって、支援システムが資源を割り当てる。そのため、資源管理モジュール erm は資源の仕様を格納している。加えて、利用者を記録し、複数の実験者の利用の衝突を回避（排他処理）を行う。

なお、多数資源を利用する際には、故障の発生を考慮せねばならない。実験駆動モジュールはノード獲得の際には予備を含めて、実験記述より多めのノードを要求する。

3.3.5 シナリオ実行

シナリオの実行は、実験駆動モジュール kuroyuri が担当している。kuroyuri は実験全体を駆動するプログラム master と各ノードで各ノードに割り当てられたシナリオを実行する slave で構成される。

ノードやネットワークの仕様記述と合わせて扱うため、シナリオの記述にはなんらかの言語が必要である。実験時にコンパイルやリンクなどの処理は繁雑になるために、その記述言語はスクリプト言語が適している。シナリオ実行を実現するもっとも単純な方法は、シェルを利用することであろう。ただし、シェルは実験者の学習が容易な点は重要だが、ネットワーク関連の機能をもたず、ネットワーク関連の操作を実現するためには多数の外部プログラムを必要とするため、SpringOS では採用していない。

一方、perl、python、ruby のようなネットワーク関連の機能が豊富な既存のスクリプト言語も存在するが、処理系の大きさと複雑さから統合が難しいと判断した。また、既存のスクリプト言語は構文が定まっているため、実験シナリオの記述を容易にする最適化した構文を設けるのが難しい。

このような考察から、SpringOS ではシェルに近い独自のスクリプト言語を設計した。この言語は以下のような機能を持つ。そして、これらは単純な構文で表現できる。

- 外部プログラム実行
- 外部プログラム実行の出力のリダイレクト
- 各種繰り返し
- 数字や文字列の演算
- 変数の宣言・代入
- ネットワークインターフェイスの走査
- 各ノードの属性（IP アドレスなど）の参照
- master 上の変数を slave へ継承
- ネットワークを介したメッセージの交換
- 特定メッセージの待機
- ネットワークを介したファイルの取得・送出

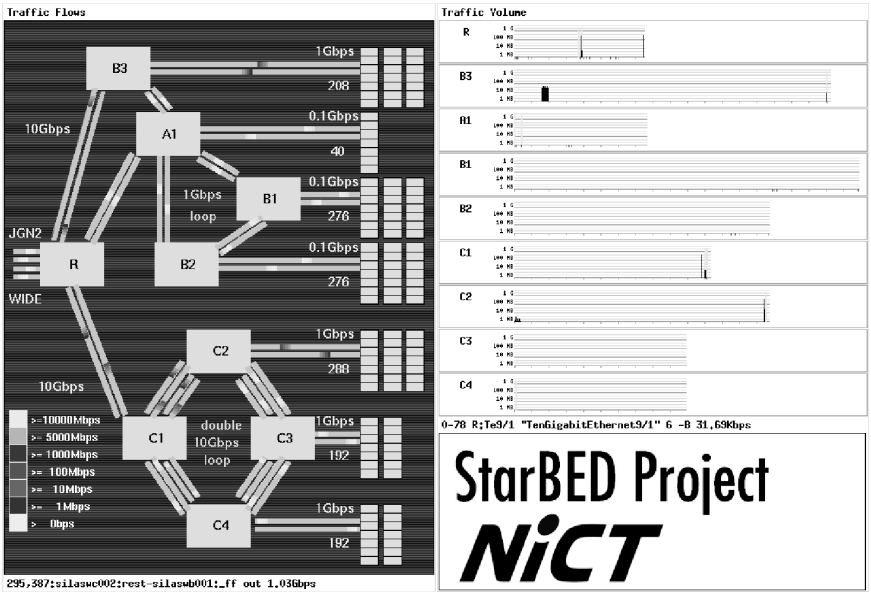

図 3.2.　xant 動作スクリーンショット

3.3.6 監視

モジュール xant はトラフィックの視覚化を行い、実験者の監視を補助する。トラフィック情報収集手段は SNMP で、定期的に指定された各スイッチの統計情報を収集し、その情報を指定された背景上に描画する（図 3.2）。現行では、各リンクのトラフィック量を色で表現しており、Mbps 未満を水色、Gbps 以上を赤色で着色する。視覚結果の出力は X Window System である。X を用いることで、マウスポインタ背後のリンクのトラフィック量をインタラクティブに表示する機能を実現できた。

モジュール sheepdog は、ICMP、SNMP、IPMI で各ノードの稼働状況を探査し、その結果を視覚化する（図 3.3）。ノードの死活監視では、状況に応じて要求される能力が異なる。起動や停止の処理の際には頻繁に状況を知りたいため、探査間隔を短く、処理結果を即座に反映させたいという要望がある。一方で、短い探査時間は大量の探査トラフィックを発行するため、起動や停止の処理を行っていない場合には、探査間隔を長くしたい。ただし、sheepdog が実験者の意向を知ることは難しい。そこで、sheepdog では、各ノードに関する探査結果を記録して、探査結果の傾向に応じて次回の探査時刻を変化させる工夫を施した。結果が頻繁に変わるノードの探査間隔は短く、結果の変化が乏しいノードの探査間隔は長くする。

また、大量トラフィックによるスイッチのパケット損失など、ノードが稼働していても探査が失敗する可能性がある。このような事態を想定して、1 度の探査失敗でノードを死亡状態と判定しないよう、前述のようなノード毎の探査結果の集計から状態を判定する。

3.4 実装

SpringOS の大部分が C 言語で、一部は perl、awk などのスクリプト言語やアセンブラで作成されている。プログラム長はのべ 12 万行余、3 MB である。

3.4.1 動作確認状況

SpringOS は以下の装置やソフトウェアで動作が確認されている。OS や CPU アーキテクチャ依存部分は少ないため、これ以外に多くの UNIX 系 OS で稼働すると思われる。

ノード OS：

- SunOS 5.9 [SPARC]（ni、coil、ifsetup を除く）
- Linux 2.4/2.6[i386]（ni を除く）
- FreeBSD 4.4/4.8/5.4 [i386]
- MacOS X（kuroyuri 中 sbpsh のみ）

スイッチ：

- Foundry IronWare
- CISCO IOS
- CISCO CatOS

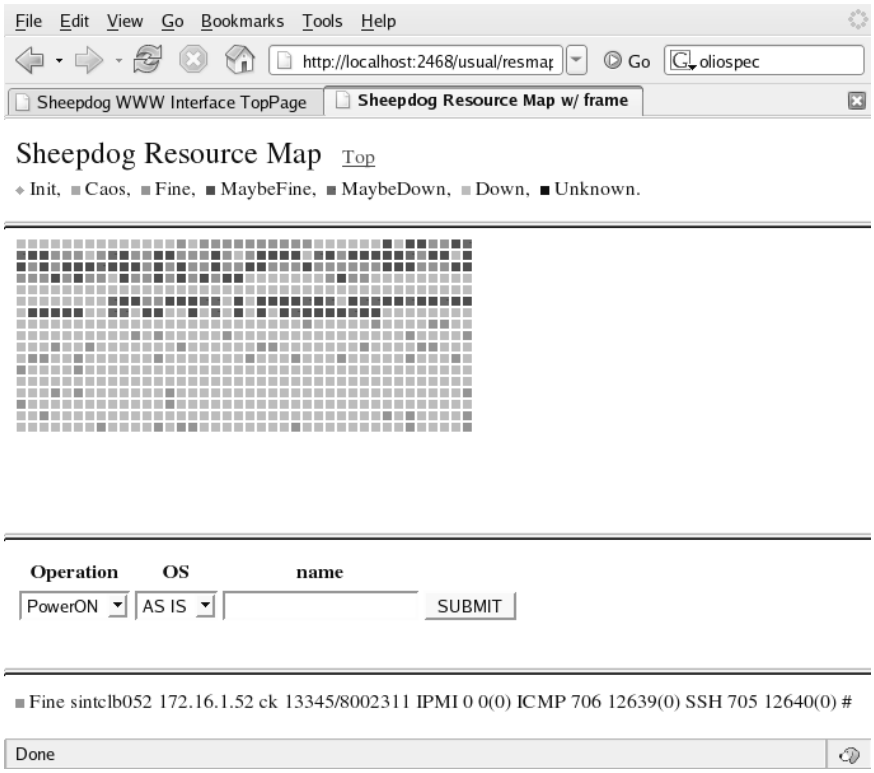

図 3.3. sheepdog 動作スクリーンショット

3.4.2 施設への制限

SpringOS は稼働施設を限定しないことを目指しているが、当初の開発環境である StarBED 由来の前提項目が見つかっている。以下にその項目をあげる。

- ノード起動時の電源投入を Wake on LAN で行う
- 起動手順を PXE で行う
- ノード停止を SNMP で行い、snmpd が常時稼働している
- ノードとスイッチはフラットに接続されている
- ネットワーク構成（トポロジ）を VLAN で構築する

電源関連はハードウェアの支援が必要なため、多少の前提は避けられない。WoL や PXE は広く普及しているため、さほど特殊な要求ではないだろう。

一方、ネットワークに関しては、物理層でネットワークを切り替える装置の導入、あるいは、実験の度に物理配線の工事を行うことは非常に困難であるため、VLAN の利用は他の方法に比べ有効である。

3.5 まとめ

SpringOS は必要なノードとネットワークの仕様と組み合わせ、そして、各ノードの挙動を記述すると、ノードへのソフトウェア・インストール、ネットワーク構築を含めた一連の実験を自動的に遂行する。その結果、操作ミスを軽減し、何度でも繰り返し実験を実施できる。そして、ネットワーク実験者の拘束時間の削減も達成できた。資源利用の排他処理の実現により、複数実験の進行を達成した。スクリプト言語を導入したことで、用途の限定事項も特にない。ただし、ノードの起動・停止、ネットワーク構成にいくつかの前提があるため、稼働施設の制限が残っている。

今後はノードの仮想化などの機能充実に注力していきたい。

謝辞

SpringOS は多数の利用者の好意と努力に支えられて成長してきた。利用者の方々に深く感謝したい。

第 4 章　GARIT と AnyBed の概要

4.1 GARIT

GARIT は奈良先端科学技術大学院大学のインターネット工学講座において運用されている実装評価用のテストベッド環境である。GARIT は構築された時期が異なる 3 種類のクラスタから構成されている。最も古いクラスタは GARIT-DELL と呼ばれ、DELL 製のブレードサーバである PowerEdge 1655MC 12 台とブレードシャーシ内蔵のスイッチにより構成されている（図 4.1）。2 つめのクラスタは GARIT-Grande と呼ばれ、Sun のブレードサーバである Fire B100 40 台、ブレードシャーシ内蔵スイッチ、Foundry の FastIron Edge X448 3 台から構成されている（図 4.2）。最新のクラスタは Garit-10g と呼ばれ、PROSIDE の Opteron サーバ 18 台と Alaxala AX3630S-24T2X 13 台から構成されている（図 4.3）。これらのクラスタは、台数が必要な実験の場合は GARIT-Grande を利用し、10GbE を必要とする実験を行う場合は GARIT-10g を利用するというように、実験用途に応じた使い分けを行っている。

図 4.1.　GARIT-DELL

図 4.2.　GARIT-Grande

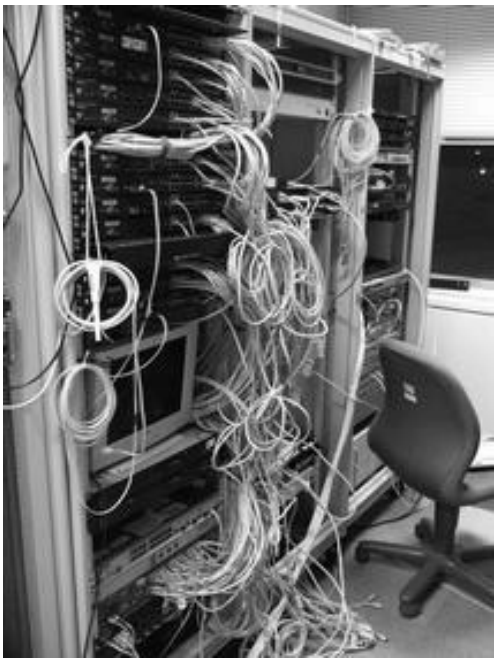

図 4.3.　GARIT-10g

4.2 AnyBed

GARIT での実験を支援するために開発されたツールが AnyBed[245, 302] である。AnyBed は様々なハードウェアで構成されるクラスタ環境において、同様の手順で実験トポロジを構築し、実験を行えるよう設計されたツール群である。

AnyBed 全体の設計を図 4.4 に示す。AnyBed は 3 つの層から構成される。1 つめは実験ネットワーク構築のための情報を集める層であるデータ収集層（design layer）、2 つめは資源割り当てを行う資源割り当て層（assignment layer）、3 つめは設定ファイルを配布し、実際の実験ネットワークを構築する設定反映層（injection layer）である。各層を構成する

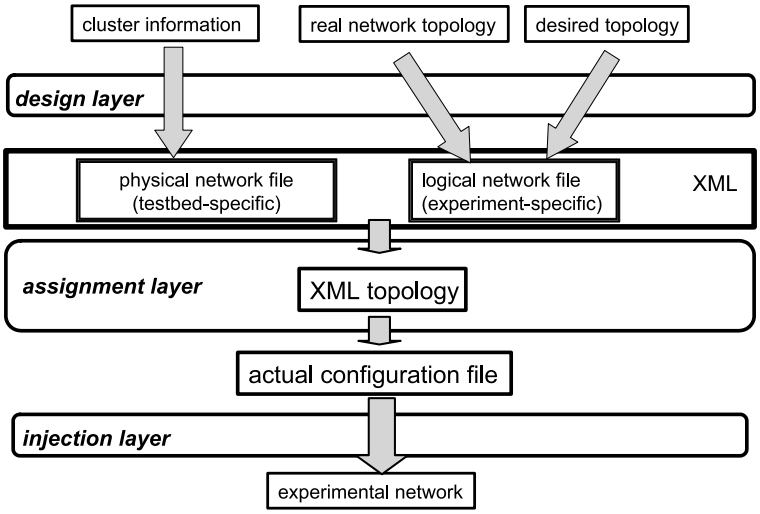

図 4.4. AnyBed 概念図

コンポーネントは容易に交換可能である。データ収集層と資源割り当て層のデータ交換には XML で記述されたファイルを用いる。

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

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

4.2.1 実験ネットワーク構築の流れ

実験者が AnyBed を用いて実験ネットワークを構築する際の流れを図 4.4 に示す。

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

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

4.2.2 物理トポロジと論理トポロジの記述

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

物理トポロジの例を図 4.5 に示す。この例では mc12 という名前のノードが bge0 という名前の管理用ネットワークインターフェイスと bge1、bge2 という名前の 2 つの実験用ネットワークインタフェイスを持ち、実験用ネットワークインターフェイスはそれぞれ mc1-sw2 というスイッチの ethernet 1/2、ethernet 1/7 という名前のポートに接続されている。

論理トポロジは実験者が構築しようとする実験ネットワークの論理的トポロジを記述する。論理トポロジの構造は次のとおりである。まずノードの集合を

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

図 4.5.　物理トポロジの例

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

図 4.6.　論理トポロジの例

表現する <nodes> 要素がある。<nodes> 要素はその子要素としてノードを表現する <node> 要素を持つ。<node> 要素は子要素としてノードが持つネットワークインターフェイスを表現する <interface> 要素を持つ。<interface> 要素はその子要素として、ネットワークインタフェイスが属するネットワークを表現する <network> 要素を持つ。

論理トポロジの例を図 4.6 に示す。この例では NodeA が NodeA-Int1、NodeA-Int2 というインターフェイスを持ち Net1、Net2 というネットワークに属している。

4.2.3 データ収集層

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

4.2.4 資源割り当て層

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

第一の機構は dispatcher である。本機構は、物理トポロジ、論理トポロジを入力として取り、XML トポロジを出力する。

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

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

4.2.5 設定反映層

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

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

第 5 章　次世代の大規模実証環境に必要な機能整理

実験の自動的な遂行のためには、実験前に実験トポロジおよび実験手順を記述する必要がある。利用者は、実験手順を記述することにより、実験環境および実験支援システムの機能を利用する。これらの手順は、実験環境が要求する文法にしたがって記述される。しかし、利用者が、実験のためだけに新た

な言語を習得するのは効率的ではない。一般的に普及しているプログラミング言語と同一の文法で実験の設定記述が可能なら、多くの利用者は新たな言語の習得なしに実験を行える。しかし、既存の言語の文法では、実験設定言語とは性質が異なるため、そのまま利用するのは困難である。そこで実験設定に適した言語を新たに提案する。

そこで、ネットワークの実験遂行に最低限必要な機能および、その機能を利用するための言語を定義する。一般的なプログラミング言語の文法にしたがって記述された設定記述を、我々の定義した言語に変換し、実験遂行効率の向上をはかる。単純な機能の集合として言語を構成することで、さまざまな文法からの変換に対応できる。複雑な機能は単純な記述の組合わせで実現する。

5.1 実験支援システムの役割概要

DeepSpaceOne ワーキンググループでは、これまで実ノードを用いた実験環境である StarBED 上での実験支援システム SpringOS を開発してきた。この経験から、以下の機能を実験支援システムに用意すれば、実験に必要な手順を自動的に実行できる。

ノード設定とシナリオの事前指定 実験トポロジを自動的に構築し、任意のシナリオで実験を進行させるためには、前もって支援システムに利用者の意図するトポロジおよびシナリオを入力する必要がある。このような要求を満たすためのしくみが必要となる。

ノードへの OS/ファームウェア導入 ノードへ利用者が指定した OS やファームウェアおよびアプリケーションを導入するためのしくみが必要である。

実ノードの設定 IP アドレスや経路情報など、実ノードを実験用のノードとして動作させるための設定を行うしくみが必要である。

実験手順の遂行 指定された順序/タイミングでコマンドを実行する必要がある。これにより実験が遂行される。タイミング調整の一つとして複数のノードの同期機構は必須である。

リソース管理機構 実験に必要なリソースを管理する。リソースには、実験に利用できるノードやネットワークを構築する際に利用する可能性がある VLAN 番号などが挙げられる。この機構は利用者から要求を満たすノードを割り当てる。また、複数の利用者により環境が共有される場合は、他の利用者に同じリソースを割り当てないような調停機構が必要である。

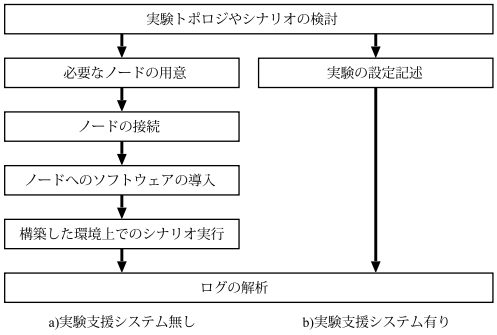

図 5.1. 実験手順の一例

ノードの起動・停止機構 ノードの起動や停止、再起動などの処理を自動的に行う機構。これにより実験開始時にノードを起動したり、終了時にノードを停止する。またノードがハングアップしてしまった場合に対応できるよう、電源レベルでの制御ができることが望ましい。

これらの手順を人手により行う場合と支援システムにより自動化された場合の比較を図 5.1 に示す。支援システムが存在しない場合、利用者は、各ノードのさまざまな制御は、コマンドの実行などにより手動で行うか、遠隔からスクリプトを実行することになる。すなわち利用者は実験実施中、常に作業を行わなければならない。また、実験の規模が大きくなった場合には、多数のノードを制御しなくてはならないため、実験の遂行は困難であり、さらに、人手による誤操作の問題は深刻である。一方、実験支援システムを利用すれば、実験の設定を事前に記述するのみで、実験は設定にしたがって忠実に行われ、新たな技術の開発および実験結果の検討という、本来の目的に注力できる。

5.2 実験の要素機能と設定記述

本節では、利用者が実験環境または実験支援システムに要求する最低限必要な機能とその記述方法について議論する。複雑な機能は単純な機能を組み合わせて実現する。多くの利用者にとって、perl や ruby、python などといった一般的な言語を利用できた方が実験を容易に行える。したがって、我々は一般的な言語で記述された設定を、図 5.2 のように、我々が定義した言語に変換する。これにより、利用者はよ

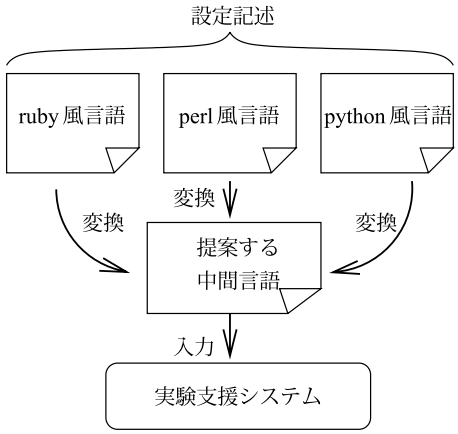

図 5.2.　設定記述の差異の吸収

り親しんだ言語で設定を記述でき、実験遂行の効率が向上する。

perl や ruby のような一般的な言語では、多くの場合、複数の要素の同期をとるための機能が用意されていない。しかし、ネットワーク実験ではノードの同期は必須であるため、新たな言語を定義する。

実験トポロジは基本的にノードとリンクにより成り立つため、ノードとリンクの設定ができなければならない。実験シナリオを遂行するためには、プロセスの実行・終了などのアクションを実行できる必要がある。また、前述の通り、アクションを起こすタイミングを、他ノードとの同期により決定するための機能も必要である。

ノード数が足りない場合などに、VMware[261] や Xen[255] のような仮想ノード実現技術を利用して、ノードの多重化を行い、一台の実ノードを複数台に見せる場合がある。この場合は、実ノードと仮想ノードの両方を制御する必要がある。また、実験ノード上で VLAN の設定を行う際には物理インターフェイスの上に論理インターフェイスが設定されるため、同様に物理インターフェイスと論理インターフェイスの設定が必要となる。この要求を満たすため、階層的に実験要素を定義するような機能が必要である。

5.2.1 ノード

実験の主体となるのはノードであるため、ノードに関する多くの記述が必要である。また、ノードの記述は、ノード選定の条件と、実験を行うためにノードに導入されるべき設定がある。

ノード選定の条件となりうる事項は以下の 2 点である。

ネットワークインターフェイスに関する設定　これは必要なネットワークインターフェイスの数だけ行われる。設定された数によりノードが選択される。また、ネットワークインターフェイスの識別子、種類および IP アドレスなど設定項目も含まれる。

ノードの性能に関する設定　実験対象の技術や実装により、一定値以上の性能をノードに要求する場合が多い。このような場合に、ノードの性能や種類を定義することが必要となる。具体的には、CPU の種類、CPU のクロック数、メモリ容量、HDD 容量などがある。

これらの条件にもとづいて、リソース管理機構により実験に利用できるノードが選定され、利用者に割り当てられる。実験によっては、リソース管理機構を介さず、意図的にノードを割り当てる場合があるため、それを満たす機能も必要である。

以下は、ノードを実験用の要素として動作させるために必要な設定事項である。

ノード名　ノードの識別子。実験中はここで指定した識別子で各ノードを判別する。

ノード種別　PC またはルータ、スイッチなど、ノードの種別を記述する。

ノード操作方法　ノードにより操作方法が異なる。特にスイッチなどはベンダーにより操作方法はさまざまである。どのような手順で設定および操作を行うかを指定する必要がある。

起動方法　ノードの起動方法には、PXE[98] や TFTP[229] を用いたディスクレスでの起動や、すでにノードの HDD にインストールされている OS を用いた起動などが考えられる。ここで記述された方法にしたがいノードが起動される。

OS の種類　指定した OS と実際に起動してきた OS が一致するかを確認することで、正しい OS が起動しているかどうかを確認できる。

ディスクイメージ/起動パーティション ID　ノードにソフトウェアを導入する方法の一つとして、前もって用意しておいたディスクのイメージを書き込むことにより、ディスクの内容を複製する方法がある。このとき、書き込むべきディスクイメージの URI が指定される必要がある。また、ディスクレス環境で利用する場合も、カーネルや、メモリイメージの URI が必要である。一方、ノードにすでに導入されている OS から

起動する場合は、パーティション・フラッシュメモリなどの起動元の指定が必要である。

割り当てる実ノード 実験用の実ノードを管理する機構が動作している場合は、必要な機能を持つノードを割り当てることもできるが、場合によっては意図したノードを割り当てたい場合もある。このような場合に、明示的にノードを指定できる必要がある。

実行するシナリオ このノードで実行されるシナリオを記述する。記述方法に関してはアクションの節で述べる。

ノードの階層化を実現するために、設定記述中に仮想ノードがどのノード上で動作するかを記述する。これにより、理論的には無限にノードの階層化を行うことができる。また、仮想ノードを起動するための各種設定が必要であるが、仮想ノード実現技術によりその設定は異なるため、ここでは言及をさける。

5.2.2 ネットワーク

我々の提案する方式では、ネットワークをノードと同様に一つの要素として扱うこととする。各ノードのネットワークインターフェイスを、ここで指定されたネットワークに接続することにより、ネットワークを形成する。

識別子 ネットワークの識別子。実験中はこの識別子で各ネットワークを判別する。

利用するアドレスレンジ アドレスレンジを設定し、接続したノードのネットワークインターフェイスの IP アドレスの自動設定を行う。ノードのネットワークインターフェイスに IP アドレスが静的に指定されていた場合は、ここで指定されたレンジ内のアドレスかどうかを判別することにより、設定ミスを検出するためにも利用できる。

種類 接続したノードのネットワークインターフェイスの種類を確認するために利用する。

5.2.3 アクション

実験トポロジ構築後の実験シナリオ実行では、プロセスの実行を中心とした処理が必要となる。ここでは、プロセスの実行などをアクションと呼ぶこととする。実験の遂行にはアクションの制御が必須となる。各ノードの設定にアクションが必要である場合も多いため、実験トポロジの構築にもこれらの記述は利用される。

我々は、メッセージ交換によりノードの同期を実現しているため、メッセージ交換のためのアクションや、最低限の制御構文も必要である。

プロセスの実行 各ノードでのプロセスの実行

プロセスの終了 指定されたプロセスを終了させる

メッセージの送受信 これによりノードの同期や、ステータスの報告を行う

条件分岐 アクションの実行を制御するために、条件分岐が必要である。

ジャンプ 設定記述中の指定した位置に強制的に処理を移す必要がある。

数値演算 条件評価のために数値計算ができる必要がある。

文字列演算 我々の手法ではメッセージ交換によりノードの同期を行うため、文字列の比較などの処理が必要になる。

第 6 章　既存のリンクエミュレータの評価

ネットワークリンクの遅延や帯域、パケットロス率などの特性をエミュレートする機構としてリンクエミュレータがある。リンクエミュレータの各特性の制御の精度や特性の設定を行ってから制御が安定するまでの期間などの過渡特性は、リンクエミュレータを利用したネットワークリンクの特性の動的制御に影響を及ぼす。そこで、本節では、リンクエミュレータとして `dummynet` を取り上げ、その過渡特性の解析について述べる。

6.1 はじめに

ネットワークに関する実験を行う上で、リンクの遅延や帯域、パケット損失率などを実験の内容に合わせて変化させる必要が生じる。このような場合に、実際に必要な特性を有する物理リンクを用意するのは困難もしくは高コストであるため、リンクの特性をエミュレートするリンクエミュレータが広く用いられている。

ここでは、特定のリンク特性を模倣することをリンクエミュレーション、それを実現する機器やソフトウェアを包括してリンクエミュレータ、特にソフ

トウェアの場合は、リンクエミュレーションソフトウェアと呼ぶこととする。

我々は、これまで実機ベースのネットワーク実験向け汎用実験設備である StarBED およびその実験支援ソフトウェア SpringOS[150] や仮想 PC によるセキュリティ向けの実証環境である VM Nebula[316, 317] などを研究開発してきた。これらにおいても、実験に応じたリンク特性の変更が求められており、現在開発中の次期実験支援ソフトウェアでは、リンクエミュレータによるリンク特性の制御機能の付加を予定している。

そこで、本節では、既存のリンクエミュレーションソフトウェアの制御の精度や設定変更時の過渡特性などの計測および解析について述べる。具体的には、FreeBSD の `dummynet`[201] を取り上げ、SmartBits[232] を用いた計測の結果とその解析について述べる。

この解析により、設定変更から実際に設定が反映されるまでの期間（設定移行期間）を明らかにし、リンクエミュレータの制御の仕方など、次期実験支援ソフトウェアの設計に反映させる予定である。

6.2 計測環境

まず、計測に用いた OS やソフトウェア、計測機器、その他の環境などの構成について述べる。

6.2.1 `dummynet`

`dummynet` は FreeBSD に標準で付属しているトラフィックシェイパーで、帯域、遅延、パケット損失率を制御することができる。kernel のオプションモジュールとして提供されており、IP ファイアウォールを提供する ipfw モジュールのインターフェイスである `ipfw` コマンドをユーザインターフェイスとして利用する。帯域制御には、WFQ2+（Worstcase Fair Weighted Fair Queueing; 最低限保証重み付き均等保証キュー）を用いている。

6.2.2 計測対象ノード・計測機器

計測は、StarBED2 上で米 SPIRENT 社の SmartBits を用いて行った。

計測対象のノードとして、StarBED 上のグループ F のノード 1 台（以降、ノード F1）を用いた。ノード F1 の構成を表 6.1 に示す。ノード F1 のカーネルに `dummynet` を用いるための下記の設定を追加して再構築し、利用した。

表 6.1.　計測対象ノード F1 の構成

項目	構成
CPU	Pentium4
Memory	2 GB
NIC	1000Base-T × 6
OS	FreeBSD 5.4 Release

表 6.2.　計測に利用した SmartBits の構成

項目	構成
本体	SmartBits 600B
モジュール	LAN-3310A
	GBIC 1000Base-T

```
1:  options IPFIREWALL
2:  options IPFIREWALL_VERBOSE
3:  options IPFIREWALL_DEFAULT_TO_ACCEPT
4:  options HZ=1000
5:  options DUMMYNET
```

1 行目は `dummynet` を利用する際のベースとなる kernel モジュールである IP ファイアウォールのモジュールを有効にする。2 行目では IP ファイアウォールのログ出力等を有効にする。3 行目は通常の IP ファイアウォールが初期状態はルールにマッチしないパケットはすべて deny（拒否）であるのに対し、accept（許可）に変更する。IP 以外のパケットも含めてすべて `dummynet` に流すために必須である。4 行目は kernel のタイマーの粒度を決める。`dummynet` を用いる際の推奨値が 1000 であり、今回はこの推奨値に設定している。5 行目は `dummynet` の kernel モジュールを有効にする。

SmartBits は、ネットワークの性能を計測する測定器で、送信ポートから発信され、受信ポートまで届いた通信の内容から、対象ネットワークのさまざまな性能を計測できる。今回 SmartBits は、表 6.2 に示す構成で、1000Base-T に対応している計測モジュールを導入して用いた。

6.2.3 計測環境の構成

計測は、図 6.1 のような構成で行った。SmartBits の送信ポートからノード F1 のネットワークインターフェイス em0 に送信されたパケットは、IP Forwarding され、ネットワークインターフェイス em1 から SmartBits の受信ポートに転送される。今回は IP Forwarding を利用しているが、`dummynet` はブリッ

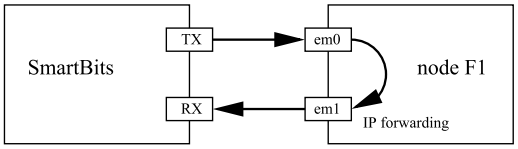

図 6.1. 計測環境の構成

ジでの利用も可能である。

ノード F1 上での外乱を軽減するために、いくつかのプロセス（init、login、tty とカーネルプロセス）を残して、他は停止して実験を行った。

6.3 遅延の測定

遅延の計測は、SmartBits の “Latency overtime” テストで実施した。“Latency overtime” テストは、指定した時間間隔ごとに受信したデータフレームの数および、遅延の最小値と最大値もしくは平均値を計測する。

今回は、計測間隔は 1 ms に設定し、1 ms ごとの受信フレーム数および遅延の平均値を計測した。フレームサイズは既定値の 124 バイトとした。

6.3.1 過渡計測

dummynet の遅延制御の性能および過渡特性を見るために、遅延の設定値を計測中に変更し、設定移行期間や設定変更の前後で遅延がどのように変化するかを計測した。具体的には、遅延の設定値を 0 ms から 100 ms もしくは 100 ms から 0 ms に 25 ms 刻みに増減させた。変更は 1 秒ごとに行い、1 ms あたりの遅延の平均値を計測した。

計測には下記の簡単なスクリプトを用い、SmartBits による計測の開始・停止とスクリプトの起動は手動で行った。

```
1:  ipfw -f flush
2:  ipfw pipe 10 delete
3:  ipfw <管理用の通信を許可>
4:  ipfw add 100 pipe 10 <em1から発信される通信>
5:  ipfw pipe config delay <設定する遅延>
    ※
6:  sleep 1
※遅延を変更して繰り返し
```

1 行目の ipfw flush は、ipfw が保持するルールセットを初期化する。2 行目の ipfw pipe 10 delete は今回 dummynet に用いるパイプを削除する。3 行目は、dummynet を通さない管理用の通信を通過させるルールセットを設定する。4 行目は、今回 dummynet を適用する通信をノード F1 の em1 から発信される通信に限定している。5 行目では実際に dummynet に遅延を設定する。6 行目では、次の遅延設定の変更まで 1 秒間、実行を停止する。

以降では、スクリプト中の 5 行目の ipfw pipe config ... を「遅延設定の変更」、6 行目の sleep 1 を「1 秒間の停止」と呼ぶこととする。

6.3.2 結果の予想

増加、減少いずれの場合においても、遅延は毎秒 25 ms ずつ変更する。そのため、遅延を 0 ms から 100 ms に上げていく場合には、グラフは 1 秒ごとに遅延が 25 ms ずつ上がっていく階段状になり、段が上がる直前に遅延の差分を吸収するために 25 ms の空白期間ができていると予想される。逆に、遅延を 100 ms から 0 ms に下げていく場合には、グラフは 1 秒 − 25 ms すなわち 975 ms ごとに遅延が 25 ms ずつ下がっていく階段状になると予想される。また、遅延の差分を吸収するために、段が落ちる前後に何らかの変動があると予想される。さらに、最初 100 ms は初期の遅延を発生するために空白期間になると予想される。

なお、今回はスクリプト上でタイマーとして sleep コマンドを用いているため、正確に 1000 ms ごとに遅延設定が変更されているとは限らないことも予想される。

これらの予想から、各段の長さや、段が変わる前後の挙動がどうなっているかを留意点として、解析する必要がある。

6.4 過渡解析

計測結果について、結果の予想に照らし合わせて過渡解析を試みる。

なお、スクリプトの起動や各コマンドの起動が計測データ上のどの時点で行われたのかは、計測データには直接記録されない。そのため、データに顕著な変化が現れた点を遅延設定の変更の効果が現れた時点と扱うこととする。

計測は 100 回行い、結果はその統計値である。

6.4.1 遅延の増加

遅延設定を 0 ms から 100 ms まで増加させていっ

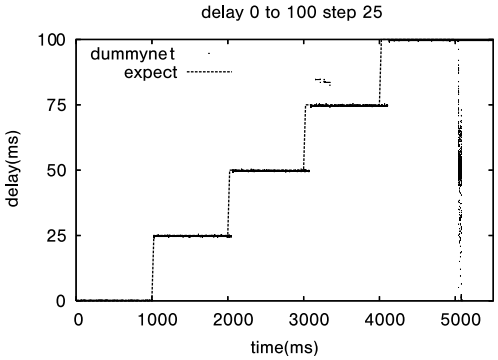

図 6.2.　遅延：0 ms から 100 ms

た場合の計測結果のグラフを予想と重ねて図 6.2 に示す。グラフの縦軸は遅延、横軸は時間である。

グラフから遅延を増加させていく場合、遅延の増加分と同程度の空白期間を考慮すると、

「遅延の増加分 + 7 ms」

の設定移行期間で設定した遅延に変更することができる。

6.4.2 遅延の減少

遅延設定を 100 ms から 0 ms まで減少させていった場合の計測結果のグラフを予想と重ねて図 6.3 に示す。グラフの縦軸は遅延、横軸は時間である。

グラフから遅延を減少させていく場合、設定遅延以下に収束するまでには、

「設定遅延 + 6〜7 ms + 4 ms」

の設定移行期間を必要とすると考えられる。

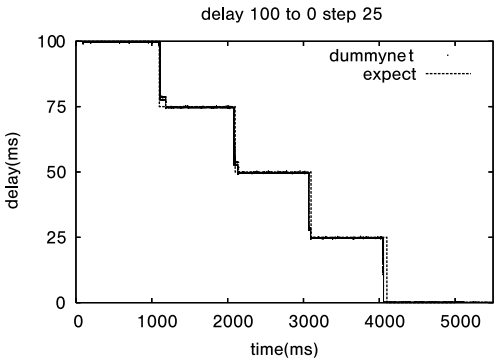

図 6.3.　遅延：100 ms から 0 ms

6.5 おわりに

本節では、FreeBSDに標準で付属している`dummynet`の遅延制御について、StarBED2 上で SmartBits を用いて計測し、過渡解析を行った。

解析によって、`dummynet` を利用してシェルスクリプトによって遅延を制御する際には、

- 遅延を増加させる場合、
  「遅延の増加分 + 7 ms」
- 遅延を減少させる場合、
  「設定遅延 + 10 ms〜11 ms」

の設定移行期間が必要との結果が得られた。

この結果をもとに、SpringOS や VM Nebula2 などの次期実験支援ソフトウェアにおけるリンクエミュレータの制御手法に反映させる予定である。

第 7 章　まとめ

今年度は次世代の支援ソフトウェアの開発のための基礎研究を主に行った。今後はこれらの研究をもとに、より柔軟な実験を行えるための支援ソフトウェアの開発を進める。また、実験の体系的分類やそれに適応する実験トポロジ・実験項目の検討も今後の重要な課題とする。