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

Title	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
Author(s)	宮地，利幸
Citation	
Issue Date	2007-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/3570
Rights	
Description	Supervisor:篠田　陽一，情報科学研究科，博士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

博 士 論 文

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

指導教員　篠田 陽一 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

宮地 利幸

2007 年 3 月

目 次

1 序 論 1

1.1 新たな技術の開発 1
1.2 インターネットの登場とインターネット向け技術の検証技術 2
1.3 本研究の目的 3
1.4 本論文の構成 4

第I部 ネットワーク実験の性質と実現手法 6

2 ネットワーク実験 8

2.1 ネットワーク実験の段階と種類 9
2.1.1 ネットワーク技術の開発手順 9
2.1.2 ネットワーク実験の段階とその目的 10
2.2 ネットワーク実験の実現 12
2.2.1 ソフトウェアシミュレータ 13
2.2.2 実証環境 13
2.2.3 大規模実証環境 15
2.3 実環境と実験駆動単位 18
2.3.1 実環境の性質 18
2.3.2 実験駆動単位の性質 19
2.3.3 実験駆動単位の構成要素 19
2.3.4 実環境と実験駆動単位の差異 20
2.4 各実験実行環境のアプローチ 22
2.4.1 ソフトウェアシミュレーション 22
2.4.2 実証環境 22

2.4.3 大規模実証環境 23

3 実験支援 25

3.1 ネットワーク実験実行の問題点 25
3.1.1 実験内容の検討・決定 25
3.1.2 実験駆動単位の構築・実験実行 26
3.1.3 実験データの解析 27
3.2 実験手順の入出力 27
3.2.1 実験内容の検討・決定 27
3.2.2 実験駆動単位の構築・実験実行 29
3.2.3 実験データの解析 29
3.3 実験支援の可能性 29
3.3.1 実験内容の検討・決定 29
3.3.2 実験駆動単位の構築・実験実行 30
3.3.3 実験データの解析 31

4 実験実行者が実験実行環境に求める性質と機能 33

4.1 実験事例 33
4.1.1 SICC の基本性能評価 33
4.1.2 マルチキャスト対応スイッチの性能検証 36
4.1.3 サーバ選択手法の新提案の検証 37
4.1.4 メガノード IP マルチキャストセンサネットに関する実験 . . 38
4.2 実験実行者が求める性質 38
4.2.1 実環境との同一性 40
4.2.2 実験実行の容易性 40
4.2.3 実験の再現性と検証容易性 41
4.2.4 要素のモデル化・抽象化の可否 42
4.2.5 身 近 さ 42
4.2.6 管理機構の存在 42
4.3 実験実行環境に求められる性質と各実験実行環境 42
4.4 実験実行者が実験実行環境に求める機能 45

4.4.1 設定記述にしたがった自動的な実験実行 45
4.4.2 実験トポロジの柔軟な設定 46
4.4.3 実験用要素の自動設定 46
4.4.4 シナリオの自動実行 46
4.4.5 ノードの時刻の同期 47
4.4.6 実験状態の把握 47
4.4.7 ログの自動収集 47
4.4.8 環境の初期化 47
4.4.9 他の実験実行者との資源分離 48
4.4.10 実験状態の保存 48
4.4.11 複数の実験実行環境の接続 48
4.4.12 実験要素の基本性能の検証 48

第 II 部 大規模実証環境の開発運用と評価 50

5 大規模実証環境の設計 52
5.1 実ノードを利用した環境での実験の実行手順 53
5.2 実験設備への要件 58
5.3 支援ソフトウェアへの要件 58

6 StarBED 60
6.1 StarBED のトポロジ 60
6.2 ノード構成 63
6.3 各要件へのアプローチ 63
6.3.1 管理用ネットワークの分離 68
6.3.2 リンク特性の模倣 68
6.3.3 ノードのコンソール操作 68
6.3.4 自動的な実験トポロジの構築 68
6.3.5 電源管理機能 69
6.3.6 そ の 他 69

7 SpringOS の設計と開発 71

7.1 実験資源の管理モデル 71

7.2 各種要件の実現アプローチ 72

7.2.1 実験実行者による設定の認識・解析 72

7.2.2 資源の状態・属性管理 72

7.2.3 実験用ノードの設定 73

7.2.4 実験トポロジの構築 74

7.2.5 シナリオにしたがった実験の実行 74

7.2.6 実験用ノードの状態管理/表示 78

7.2.7 実験ログ収集 78

7.2.8 リンク特性の模倣 78

7.2.9 電源管理機能 78

7.3 SpringOS を構成するモジュールとその動作 79

7.3.1 SpringOS のモジュール 79

7.3.2 SpringOS の処理の流れ 82

7.4 ディスクイメージの作成 85

7.5 設定記述例 85

7.6 StarBED での仮想機械の利用 90

7.6.1 SpringOS/VM の処理手順 90

7.6.2 仮想機械を利用した場合のネットワーク構成 92

7.6.3 資源管理 93

7.6.4 ノード割り当て 93

7.6.5 実ノード上での仮想機械のための設定 94

7.6.6 シナリオの実行 94

7.7 考 察 95

8 StarBED と SpringOS の評価 96

8.1 実験概要 96

8.2 実験の進行 98

8.3 ディスクイメージの作成 98

8.4 実験のシナリオ 100
8.5 評価実験で利用した設定記述 103
8.6 実験結果 111
8.6.1 ディスクイメージの作成 111
8.6.2 実験の実行 113
8.7 SpringOS/VM の動作確認 120
8.8 考 察 125

第 III 部 より高度な実験支援環境の構築にむけて 127

9 汎用的な実験支援ソフトウェアのアーキテクチャの提案 129
9.1 大規模実証環境に求められる機能の実現 129
9.1.1 資源管理および資源割り当て 130
9.1.2 資源利用の排他処理 131
9.1.3 設定読み込みおよび実験駆動単位の生成 132
9.1.4 ノード制御 132
9.1.5 実験設備および実験駆動単位の状態監視 134
9.1.6 ログ収集 135
9.1.7 実験状態の保存と復元 135
9.1.8 複数の大規模実証環境の協調 135
9.1.9 例外処理 136
9.2 提案アーキテクチャ 136
9.2.1 資源管理 137
9.2.2 設定読み込みおよび実験駆動単位の生成 137
9.2.3 ノード機能の指定および割り当て 138
9.2.4 ノードの電源管理 140
9.2.5 ノードへのソフトウェア導入および設定 142
9.2.6 ノードでのコマンド実行 143
9.2.7 トポロジおよびリンク特性の設定 143
9.2.8 大規模実証環境および実験駆動単位の状態監視 146

9.2.9 ログ収集 . . . 147
9.2.10 実験設備の初期化 . . . 147
9.2.11 実験実行者による手動実験制御 . . . 149
9.2.12 資源利用の排他処理 . . . 149
9.2.13 実験駆動単位の制御 . . . 149
9.2.14 実験状態の保存と復元 . . . 149
9.2.15 複数の大規模実証環境の協調 . . . 150
9.2.16 例外処理 . . . 150
9.3 考 察 . . . 152
9.3.1 汎 用 性 . . . 152
9.3.2 大規模実証環境の協調 . . . 153
9.3.3 大規模実証環境の評価基準 . . . 154

10 実験内容の決定支援手法の提案 155
10.1 実験実行環境の決定支援 . . . 155
10.1.1 実験の目的 . . . 155
10.1.2 モデル化・抽象化の可否に従属する性質 . . . 157
10.1.3 規模追従性 . . . 159
10.1.4 管理機構の存在 . . . 159
10.1.5 実験実行環境の決定 . . . 160
10.2 実験トポロジの決定支援 . . . 161
10.2.1 単純なトポロジ . . . 162
10.2.2 より現実的なトポロジ . . . 162
10.3 考 察 . . . 165

11 結 論 168
11.1 本研究の成果 . . . 168
11.2 本研究分野の課題と展望 . . . 170

参考文献 174

本研究に関する発表論文 178

図 目 次

2.1 実験実行環境と実験駆動単位 14

3.1 実験の実行手順と問題 26
3.2 各実験手順での入出力の一例 32

4.1 TCP Fairness の検証用トポロジ 35
4.2 高速輻輳回避の検証用トポロジ 36
4.3 Intra Session Fariness の検証用トポロジ 37
4.4 マルチキャスト対応スイッチの性能検証用トポロジ 38
4.5 サーバ選択手法検証用トポロジ 39

5.1 一般的なネットワーク技術の開発手順 54
5.2 提案するネットワーク技術の開発手順 55
5.3 実験の実行手順 56

6.1 情報通信研究機構 北陸リサーチセンター 61
6.2 StarBED のトポロジイメージ 62
6.3 クライアント装置 A 64
6.4 クライアント装置 F 65
6.5 実験用スイッチ A（Cisco Catalyst 6509） 65
6.6 実験用スイッチ B1（Cisco Catalyst 6009） 66
6.7 実験用スイッチ B3（Foundry BigIron MG8） 66
6.8 実験用スイッチ C3（Foundry BigIron RX-16） 67
6.9 状態表示用スクリーン 70

7.1 ノード自律モデル 75
7.2 コマンド送信モデル 76

7.3 SpringOS での採用モデル 77
7.4 NI,FNCP および ENCD の通信 81
7.5 SpringOS の各機能の関連 83
7.6 本実験例での設定ファイル 88
7.7 実験のタイムライン 89
7.8 SpringOS/VM の処理手順 91
7.9 仮想機械を用いた実験トポロジ 92

8.1 評価実験用トポロジ 97
8.2 クライアント装置 A の管理ネットワーク構成 99
8.3 評価実験のタイムライン 102
8.4 評価実験用の変数定義部 104
8.5 評価実験用設定記述 server（netserver）用クラス定義 105
8.6 評価実験用設定記述 client（netperf）用クラス定義 107
8.7 評価実験用設定記述 ネットワーククラス定義 108
8.8 評価実験用設定記述 インスタンス生成部 108
8.9 評価実験用設定記述 トポロジ設定部 108
8.10 評価実験用設定記述 グローバルシナリオ 109
8.11 評価実験用設定記述 StarBED 基本設定 110
8.12 ノード数とディスクコピーの所要時間の関係 116
8.13 ノード数と VLAN 設定の所要時間の関係 117
8.14 ノード数とシナリオ実行の所要時間の関係 117
8.15 ノード数と VLAN 設定削除の所要時間の関係 118
8.16 ノード数と実験全体の所要時間の関係 118
8.17 実験全体の所要時間の関係 119
8.18 netperf 実験用トポロジ 121
8.19 VMware 起動ノード用クラス 122
8.20 仮想機械用クラス 123

9.1 想定する実験設備のトポロジ 130
9.2 実験駆動単位の構築手順 139

9.3 資源割り当て手順 141
9.4 ノードのディスクへのソフトウェア導入手順 144
9.5 シナリオの実行手順 145
9.6 ログ収集の手順 148
9.7 複数の大規模実証環境の協調手順 151
9.8 大規模実証環境の協調の概念図 153

10.1 適切な実験実行環境およびトポロジ提案 156
10.2 実験実行環境と実験駆動単位の要素数の関係 160
10.3 ダンベル型トポロジ 163
10.4 スター型トポロジ 163
10.5 デイジーチェイン型トポロジ 163
10.6 ツリー型トポロジ 164

表 目 次

4.1 各実験実行環境の性質 44

6.1 StarBED のノード構成 64
6.2 StarBED の主要スイッチ 67

7.1 SpringOS の各機能と所属する管理レベル 84

8.1 クライアント装置 A のノード情報 97
8.2 FreeBSD インストール条件 112
8.3 FreeBSD のインストール所要時間 112
8.4 ディスクイメージ作成の所要時間 112
8.5 2 台（1 ペア）での実験実行の所要時間（秒） 113
8.6 6 台（3 ペア）での実験実行の所要時間（秒） 114
8.7 10 台（5 ペア）での実験実行の所要時間（秒） 114
8.8 50 台（25 ペア）での実験実行の所要時間（秒） 114
8.9 100 台（50 ペア）での実験実行の所要時間（秒） 115
8.10 150 台（75 ペア）での実験実行の所要時間（秒） 115
8.11 200 台（100 ペア）での実験実行の所要時間（秒） 115
8.12 それぞれの台数での所要時間の平均値（秒） 116

10.1 各実験実行環境の性質の関連性 159
10.2 実験の種類と規模から導かれる実験実行環境 161

第1章

序 論

1.1 新たな技術の開発

これまで人類はその生活をより快適にするために、さまざまな新しい技術を考案して来た。このような技術のすべてが成功したわけではなく、数多くの失敗があったことはいうまでもない。しかし、その失敗が有益な技術を創り出すための蓄積となり、技術の創成コストを下げ、さらに優秀な技術を産み出してきたのである。このようにして創成された有益な技術が集約し、現在の豊かな社会の礎となっている。したがって、失敗は新たな技術のための基盤知識として非常に重要であるといえる。

このように、失敗は貴重な資料であるが、実環境に技術を導入する前にできるだけ排除されるべきである。実環境にはすでに数多くの技術が導入されており、新たな技術は、既存の技術と相互に影響をおよぼしあう。また、実環境に導入されればその利用者も増大するため、想定しない利用のされ方や大規模な環境で動作することによる問題が生じることがある。そのため、実環境に技術や製品を導入する前には、技術・製品の質の向上と、実環境にすでに存在する要素に対する悪影響の排除、そして耐規模性の確保を目的とした実験が行われる。

自動車や航空機などの開発では、スーパーコンピュータを用いた詳細なシミュレーションによる論理的な検証と、プロトタイプ実装を用いた実践的な検証が行われる。シミュレーションでは、詳細なデータをまえもって入力しておき、何かイベントが起きた際に引き起こされる現象を論理的に計算する。これにより、さ

まざまな現状を観測でき、特に、ある現象が発生している際の内部の状況を確認できる。これは、プロトタイプ実装を実際に動作させ確認することは非常に困難である。一方、プロトタイプ実装を用いた実験では、実環境向けの実装そのものを利用し、実践的な結果を得ることができるが、そのコストは高いといえる。また、医療に関する開発では、実験室での化学反応等の検証の後に、動物実験、臨床実験など少しずつ対象環境に近づけながら段階的に検証が行われている。

このように技術・製品を実環境に導入するためには、まえもって実環境を模倣した検証専用の環境を何らかの方法で実現し、その環境内で検証が行われている。

1.2 インターネットの登場とインターネット向け技術の検証技術

現在の重要な社会基盤の一つとしてインターネットがある。インターネットは、1960 年代に誕生した新たな環境で、そもそもインターネット自体が実験的なネットワークであった。したがって、新たな技術はインターネットそのもので検証されており、検証専用の環境は必要ではなかった。しかし、現在インターネットは急速に社会に浸透し、インターネット上で数多くの重要なサービスが運用されている。

このような急激な発展にともない、インターネット向けのさまざまな技術が日々提案されているが、従来のように新たな技術の検証をインターネット上で行えば、既存のサービスに影響をおよぼす可能性がある。もはや一利用者の独断によるインターネット上での実験は一つのテロリズムになりかねないのである。したがって、新たな技術をインターネットへ導入する前に、インターネットから分離された擬似インターネットともいえる実験専用の環境で十分な検証する必要がある。

しかし、インターネットの発展が急速すぎたため、その検証技術は成熟に足りない。このため、インターネットに新たな技術が導入されてから問題が発覚し、インターネット上のさまざまなサービスなどに影響をおよぼす事例は後を絶たない。原因として、インターネットを模倣する技術の議論が不十分であることがあげられる。さまざまな検証手法が存在し、それぞれ適用できる実験は異なる。論理的な検証には数値検証や、ソフトウェアシミュレーションが、実践的な検証にはイ

ンターネットで実際に利用されるハードウェアおよびソフトウェアを用いた実証実験が行われている。

数値計算やソフトウェアシミュレーションでは、実験の実行者が想定した理想の環境上での検証が行える。しかし、その一方で、実際にインターネット上で動作させる実装そのものを利用できない場合が多く、その結果が実際の環境での挙動と同一のものであるかについて、深い考察が必要である。また、インターネット上で実際に利用されるハードウェアおよびソフトウェアを利用して構築した環境では、構成要素はインターネットと同一のものであるため、インターネットに近い挙動を期待できるが、インターネットのような大規模な環境の構築は不可能であり、数十台程度のある程度小規模な環境でも、その構築コストは大きい。

したがって、現在の多くの技術は、実行の容易さから数値検証やソフトウェアシミュレーションや、数台の計算機からなる実験用の環境で検証されることが多い。しかし、実験用の環境を実現する手法により、構築される環境の性質も異なり目的とする実験結果を得られないこともある。実験目的に見合った環境が選択されることが理想であることはいうまでもないが、現状では既存の検証手法の性質に関する議論が不十分であり、また、実験の実行コストが高いために、必要な手法を利用できない場合も多い。さらに、実験トポロジや実験項目も実験結果に大きな影響をおよぼすが、現状ではその決定は実験の実行者に委ねられている。このため、実験結果の正確性についてはそれぞれの実験について検討が必要である。同じ目的・機能を持つ技術であっても実験の実行者が異なれば、実験内容が異なり、単純に比較ができないため、比較が必要な場合には同じ実験内容での再実験が必要となる。

1.3 本研究の目的

前節で述べたように、ネットワーク実験に関する議論はいまだ不十分である。本研究ではさまざまな視点からネットワーク実験の実行の支援手法に関する議論を行う。

まず、ネットワーク実験の性質と、ネットワーク実験を実現する既存の手法の性質を整理する。その上で実験の手順の整理を行い、それぞれの手順での支援の

可能性を検討する。また、実験を実際に実行する手順に着目し、現時点で支援についての議論が不十分である大規模な実験を実環境で利用されている機器を利用して行うための環境について議論する。特に、このような環境を実現するための実験設備である StarBED と、StarBED での実験実行を支援するソフトウェアである SpringOS についての詳細を述べる。

StarBED のように多数のノードを持つ実験設備上での実験支援のために必要になる機能の議論および整理は現状でなされていない。本研究では、StarBED と SpringOS の開発・運用の延長として、実験支援ソフトウェアに一般的に必要となる機能を整理し、さらにその機能を実現するアーキテクチャを提案する。また、提案アーキテクチャを用い、実験支援ソフトウェアを協調させ、StarBED だけでなく複数の実験設備を利用した実験の実現を目指す。

以上のような実験の実行の支援だけではなく、実験実行前に決定される、検証対象の技術に応じた実験内容を決定するための手法についても議論する。

1.4 本論文の構成

本論文は、以下の 3 部と序論・結論から構成される。

- 第 I 部 ネットワーク実験の性質と実現手法
- 第 II 部 大規模実証環境の開発運用と評価
- 第 III 部 より高度な実験支援環境の構築にむけて

第 I 部では本研究で対象とするネットワーク実験の定義と、ネットワーク実験の一般的な進行方法、実現手法について整理し、実験目的と利用できる実験手法を述べる。また、実験支援の基礎検討として、実験を行う際の手順についてまとめ、それぞれの手順の入出力と支援内容について議論する。また、そのうち実験実行に着目し、実験実行者が実験実行環境に求める機能をまとめる。第 II 部では、これまで開発・運用してきた大規模な実験設備である StarBED および、StarBED での実験実行を支援するためのソフトウェア SpringOS の詳細について述べ、その評価を行う。第 III 部では、今後、より高度な実験を行うための実験設備および支

援ソフトウェアのアーキテクチャの提案と、実験内容の決定支援として、実験実行環境、実験トポロジの決定手法について議論する。

第I部

ネットワーク実験の性質と実現手法

第 I 部ではネットワーク実験とは何かを議論する。2 章でネットワーク実験の手順を整理し、それぞれの手順での実験の目的についてまとめる。また、ネットワーク実験を実現するための既存技術についてまとめ、インターネットのような実際に運用されているネットワーク環境と、検証用環境の性質や構成要素を整理する。

3 章では、一つの実験を行うための手順を整理し、それぞれの手順に必要な入力情報と出力情報を整理し、実験支援の可能性についての議論を行う。

4 章では、これまでに行われた実験例を紹介する。これは、これまで実験を行ってきた研究者および開発者にインタビューを行った結果とこれまでの開発運用の経験にもとづくものである。また、そこから得られた実験の実行者が求める性質および機能をまとめ、各既存技術の性質を整理する。

第 I 部で議論される内容は、現状のネットワーク実験の基本的な性質の整理であり、ネットワーク実験に関する研究の基礎となる部分である。

第2章

ネットワーク実験

ネットワーク実験の目的は大きく分けて2つに分類される。一つは新規技術の検証のためであり、もう一方は教育目的での既存技術の動作確認のためである。本研究ではこのうち主に前者の新規技術の検証に着目し議論を進める。

新規技術の検証のためのネットワーク実験は、新たに発案したアイディアが有効であるかや、ソフトウェア/ハードウェア実装の挙動が仕様を満たしているか、バグがないか、そして実環境もしくは既存のネットワークサービスへの影響がないかなどを確認するために行われる。

ある実験は、対象とするネットワーク環境に実際に対象技術を導入した際に観測される結果と、その実験結果が一致しなければ意味をなさない。したがって、対象とする運用中のネットワーク環境（実環境）を忠実に模倣することが必要である。ただし、求められる忠実さは実験により異なる。たとえば、アイデアやアルゴリズム、プロトコルが正しく動作するかといった検証などでは、実験の実行者が想定する理想的な環境を用意することが重要である。実環境では必ず別のネットワークサービスや計算機、その他のアプリケーションソフトウェアなどによる外乱が生じるが、このような目的の実験には外乱は必要ない。一方、実装の検証では、対象の実環境を外乱なども含めて模倣し、その上で想定外の状況などが起きないかを検証する必要がある。このように、検証の段階や目的により検証手法は異なる。

本章では、ネットワークサービスの検証手順について整理し、それぞれの段階で重要となる実験目的について述べる。また、ネットワーク実験の実現方法とし

て現在広く利用されている手法を紹介する。さらに、実環境と実験用環境のそれぞれの性質について整理し、既存の手法での模倣アプローチについてまとめる。

2.1 ネットワーク実験の段階と種類

ネットワーク実験の目的はさまざまである。しかしネットワークソフトウェアおよびハードウェアの開発段階に着目した場合、それぞれの段階で開発者が確認したいと考える点には共通点がある。本節では、ネットワーク技術の開発手順と、それぞれの段階での実験の目的を整理する。

2.1.1 ネットワーク技術の開発手順

まず、ネットワーク技術のおおまかな開発手順を以下に示す。

1. **新規アイデアの発案** 新たなアイデアを発案する。
2. **新規アイデアの効果確認** アイデアを論理的に解析し、その有効性を確認する。
3. **実環境向け実装の作成** 論理的な解析の結果、有効性があると認められれば、実環境向けの実装を構築する。
4. **実環境向け実装の挙動検証** 実環境向け実装が想定どおりに動作するか、他のネットワークサービスに影響をおよぼさないかを検証する。
5. **実環境への導入** 実環境向け実装を実環境へ導入する。

本章では、このうち実際に試験を行う手順2、4に関してその目的をまとめる。手順2のアイデアの効果確認は、実験の実行者が想定する状況において、そのアイデアが想定どおりに動作するかという点を検証するため、論理的検証である。一方、手順4は、実環境向けの実装が手順2で確認した結果と同様の挙動を示すかを確認するため、実践的検証である。論理的検証では、理論やアルゴリズムを検証するために、実験の実行者の想定する理想的な環境を作り出し、その上で実験を行う。これにより、理論やアルゴリズムの論理的な性能や、その手順の妥当性などを検証できる。一方、実践的検証では、実環境向けの実装をバグや実装の仕

方まで含めて、手順の妥当性や性能を検証するため、論理的検証では確認できない実装上の問題の発見や、実環境への導入に際する問題点、また、実験実行者が当初想定していなかった問題の発見が期待できる。

2.1.2　ネットワーク実験の段階とその目的

前節で述べたとおり、ネットワーク実験には、論理的検証と実践的検証がある。論理的・実践的検証ともに以下の段階で実験が行われる。

1. 実装単体での動作試験
2. 最低限の系での動作試験
3. 性能試験
4. 実環境を考慮した周辺要素が無い環境での動作試験
5. 実環境を模倣した環境での動作試験

次に論理的検証および実践的検証の各段階での目的をまとめる。

・論理的検証

実装単体での動作試験 論理検証を実現するソフトウェアを開発し、対象技術部分の動作を補助モジュールなどを利用して検証し、単純なバグなどを修正する。

最低限の環境での動作試験 実環境で対象技術が動作する最小の環境を構築し、挙動の正当性を検証する。この時点で考えられるさまざまな状況を作り出し、それぞれに対して想定する挙動を示すかなどを検証する。

性能試験 さまざまな手法で対象技術もしくはその技術で管理・制御するリンクなどに負荷をかけ、論理的にどの程度の性能を持つかを検証する。

実環境を考慮した周辺要素の無い環境での動作試験 導入対象の環境を考慮した環境での手順の検証を行う。これにより規模による問題点や想定外の状況が発生しないかを検証する。

実環境を模倣した環境での動作試験 さらに、対象技術に関係ないと思われる周辺要素やトラフィックまでも導入したより現実的な環境上での手順検証を行い、既存の環境へ与える影響や、既存の環境から受ける影響を検証する。

・実践的検証

実装単体での動作試験 実験用の環境を構築する前に、*telnet* などの汎用的なツールや単純な補助プログラムを利用して、対象の実装が単体で想定どおりにの動作するかを検証する。この時点では単純なバグの修正を目的とする。

最低限の環境での動作試験 実環境上で対象実装が動作すると考えられる、最低限の環境を実環境向けの実装のみで構築し、実装されている挙動が正しいかどうかの検証を行う。

性能試験 さまざまな手法で対象の実装もしくはその実装が管理・制御するノードもしくはリンク、ネットワークに対して負荷をかけ、対象環境で想定される負荷に耐えられるかどうか、またはどの程度までの負荷に耐えられるかを検証する。また、どの部分に問題点があるのかを洗いだし、性能の向上を図る。

実環境を考慮した周辺要素の無い環境での動作試験 導入対象の環境を考慮した、より現実的な環境で検証を行う。小規模な環境では発生しなかった問題の発見を目的とする。

実環境を模倣した環境での動作試験 さらに対象技術に直接は関係ないと思われる周辺要素やトラフィックも導入し、より現実的な環境をつくり、想定していなかった問題が発生しないかや、既存環境への影響を検証する。

ただし、ネットワーク実験は、新たな技術や実装の検証のみのために行われるのではない。既存のネットワークの挙動観察のため、論理的および実践的な分析を行ったり、教育を目的とした例も少なくない。このような場合は、開発段階は関係ないため、上記の分類には当てはまらない。このような実験の内容、目的は、上記でまとめたネットワーク技術の開発段階の一つと一致することが多いため、目的に応じた環境を用意することとなる。ただし本論文では対象としない。

2.2 ネットワーク実験の実現

前述のとおり、実環境でネットワーク実験を実行した場合には、さまざまな問題を引き起こす可能性がある。したがって、実環境を模倣した環境で実験を行う。このための基礎技術としてシミュレーションとエミュレーションがあり、ともに対象を模倣する技術である。ただし、その手法と得られる結果は異なる。

シミュレーションは、実環境を何らかの手法でモデル化し、その上で同様にモデル化した検証対象の技術を動作させ、その挙動や影響、効果を確認する手法である。シミュレーションには思考実験や数値実験、ソフトウェアシミュレーションなどがある。思考実験は、実際に実験器具を用いることなく、人間の思考により対象の挙動を論理的に演算・検証する手法であり、数値実験は、事象を数式で表し、計算をもとに現象を検証する手法である。ソフトウェアシミュレーションでは、対象技術をモデル化し、アプリケーションソフトウェアとして実装し、このアプリケーションソフトウェアを動作させることで、その挙動を検証する。環境、対象技術ともすべてを実装する方法もあるが、ある程度汎用的な環境を前提とする場合には、汎用的なソフトウェアシミュレータを利用できる。

エミュレーションは、実環境のさまざまな要素を模倣し、実環境で動作するアプリケーションソフトウェアやハードウェアを動作させることで検証を行う手法である。周辺環境の動作はシミュレーションなどで実現され、対象技術と周辺環境とのインターフェース部分が実環境と同一であるため、対象実装からは周辺環境が実環境と同等に見える。エミュレーションはさまざまな段階で実現され、たとえば、OS のためのエミュレータの場合は、計算機を模倣し、実際の計算機と同一のインターフェースを OS に提供する。一方で、ハードウェア・ソフトウェアともに実環境と同一のものを利用する手法もある。

シミュレーションやエミュレーションを利用した、さまざまなネットワーク実験の実現技術が提案、利用されている。本論文では、特に実装を動作させる事で実験を行うための環境を実験実行環境と呼ぶ。ソフトウェアシミュレーションや実ノードを利用したものがこれに含まれる。また、実験の実行者（実験実行者）は、実験実行環境上に対象の実験用ノードや管理用ノードからなる実験駆動単位を構築し、その上で実験を行う（図 2.1）。実験実行環境は実環境を模倣するための環

境であり、模倣の方法は各手法で異なる。本節では、シミュレーションおよびエミュレーションの発展形ともいえる、既存の実験実行環境について述べる。

2.2.1 ソフトウェアシミュレータ

ネットワーク実験に関するソフトウェアシミュレータは、パケットやノード、ネットワークなどの単位で各要素をモデル化し、ネットワーク実験を行うソフトウェアである。汎用的なイベント駆動型のソフトウェアシミュレータが広く利用されており、汎用的なソフトウェアシミュレータの代表的なものとして、ns-2[1] や OPNET[2]、SSFnet[3] がある。ソフトウェアシミュレータは、実験実行者による実験トポロジや実験の手順（シナリオ）の記述にしたがいトポロジを構築し、その上で実験を実行するため、外乱のない理想的な環境上で実験を行える。ソフトウェアシミュレータを用いて実験を行う際には、ソフトウェアシミュレータが動作する計算機が最低 1 台あればよく、設定記述を作成すれば自動的に実験が実行されるため、実験を行うためのコストは小さい。

しかし、多くのソフトウェアシミュレータでは、実環境向け実装は利用できず、ソフトウェアシミュレータ専用の実装が必要となる。したがって、インターネットへの導入までを視野にいれた開発活動を行っている場合は、ソフトウェアシミュレータとインターネット用の 2 つの実装を用意する必要があり、さらに、それらが同一の挙動を示すかどうか検証が必要であるため、開発コストは増大する。また、ソフトウェアシミュレータ自体の挙動の正当性の検証も不可欠である。

多くのソフトウェアシミュレータは、実時間で動作せず、独自の論理時間で動作する。簡単な実験はシナリオで設定された時間よりも短い時間で終了するが、シミュレーション対象の要素の数が非常に多い場合や、それぞれの要素の挙動を詳細に模倣した複雑な実験を行う場合は、実験終了までにシナリオで設定された時間の数百から数千倍の時間がかかることもある。

2.2.2 実証環境

ネットワークに関する研究組織では、数台から十数台程度の計算機を用いて実験用環境を構築することが多い。これはエミュレーションを利用した環境であり、

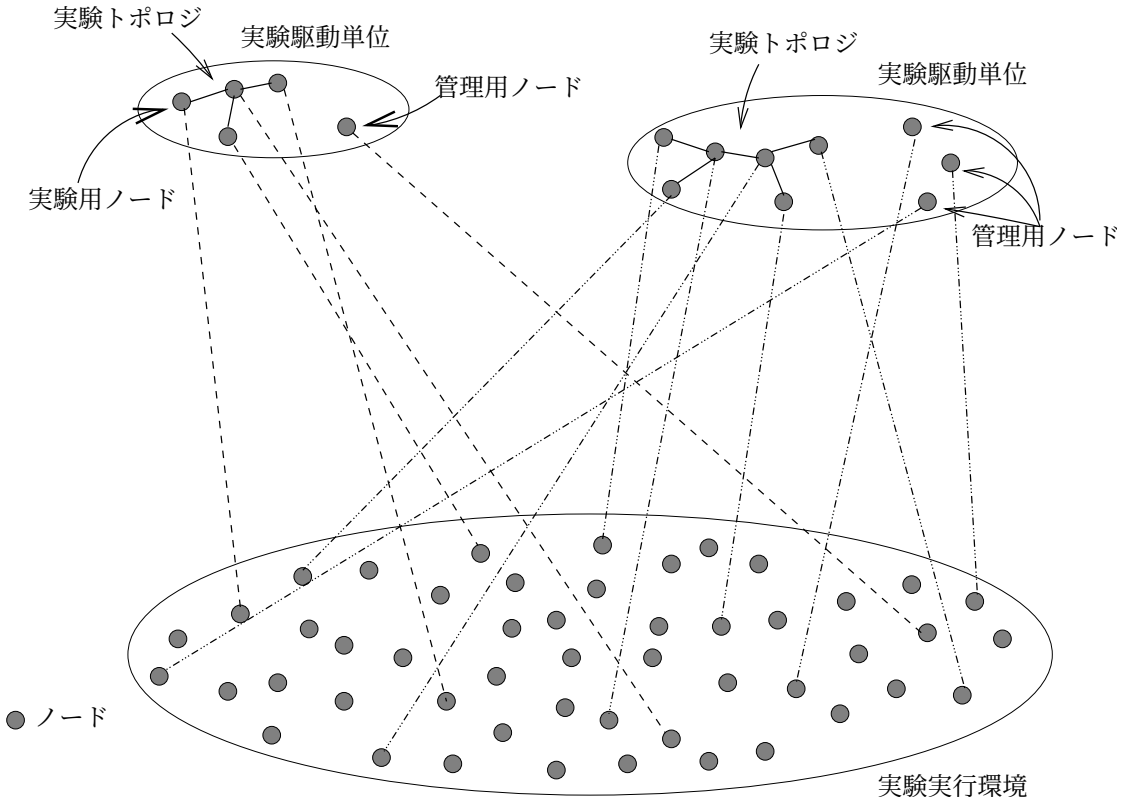

図 2.1: 実験実行環境と実験駆動単位

実験実行環境には実験に利用できるさまざまな種類のリソースが用意されており、実験実行者は必要なリソースの割り当てを受け、実験用のトポロジを構築する。また、実験を制御するために必要な管理用ノードも用意されている。実験トポロジを構成するリソース群および、管理用ノードを含む、ある実験の実行のために実験実行環境上に構築された要素群を実験駆動単位と呼ぶ。実験駆動単位は一つの実験実行環境上に複数用意されることもあり、その規模はまちまちである。図中には、リソースとしてノードのみを記述したが、リソースの種類は多岐にわたり、実験実行環境によってもその種類は異なる。たとえば、一台の計算機上で実験を実行できるソフトウェアシミュレータでは、実験実行環境として持つリソースは CPU の計算能力やメモリ量などであり、実環境と同様のハードウェアを利用した実験用環境では、ノードやリンクなどがそれに該当する。

実際に実環境で利用するものと同様のネットワーク機器および計算機を用いる。本論文では、数台から十数台規模の実ノードにより構築される小規模な実験実行環境を実証環境と呼ぶ。

実証環境では、実環境のための実装がそのまま利用できるため、新たな技術の仕様に記述されていない挙動やバグまでを含め、実環境に非常に近い観測結果が期待できる。また、各実装の操作方法も実環境と同一であるため、実環境の運用技術者の育成という目的にも利用できる。しかし、実証環境を構築するためには、実験トポロジを構築するために必要なノードを物理的に配置し、それぞれを接続し、必要であればスイッチなどの設定を行う必要がある。したがって、ノードを用意することによる経済的・空間的コストおよび、ノードの接続、ネットワーク設定のための人的・時間的コストが大きい。また、それぞれの実験用ノードに実験実行者が意図するソフトウェアを導入し、その後、実験手順どおりに制御する必要がある。これらの手順を手動で行った場合は、人的コストが大きくなるばかりではなく、シナリオ実行時には、コマンド入力のようなイベント発生のタイミングのずれや操作ミスにより、実験の精度が低下するおそれがある。

2.2.3 大規模実証環境

実ノードを利用した実験実行環境では、実環境向けの機器やソフトウェアを利用できるため、実環境に近い挙動をみせる実験駆動単位を構築できる。しかし、実験駆動単位を構築するコストが大きいため、数台から十数台程度の実ノードを用いた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験駆動単位で容易に実験が行えることが理想である。このような問題を解決するため、多数の計算機やネットワーク機器からなる実験設備が利用されている。しかし、このような実験設備を制御することは、その規模のために困難である。したがって、このような実験設備には専用の支援ソフトウェアが用意されていることが多い。支援ソフトウェアにより、実験トポロジの構築やノードへの OS・アプリケーションソフトウェアのインストールなどを補助し、実験を容易に進めることができる。本論文では、このような大規模な実験設備と支援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。

本節では、大規模実証環境の例として、Netbed[4]、PlanetLab[5]、VM Nebula[6]、SIOS[7]、GARIT および StarBED について述べる。

Netbed/Emulab

Netbed は、分散システムと実ノードによる環境およびソフトウェアシミュレータの統合環境であり、豊富な実験管理機能を持つ。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、VLAN 技術 [8] 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[9] を利用することで、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、大規模かつ柔軟な環境構築を可能とするが、その一方 ns-2 により実現している部分では、前述のとおり実環境と同一の実装は扱えない。

ある程度の規模の PC クラスタを持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、実環境のリンク特性を導入できるが、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。

Netbed では、実験実行者からのリクエストを受付け、設備が空き次第実験を実行する。実験は一括処理で実行されるため、実験を行いその状況を判断して次の実験を実行したい場合や、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、その後は実験実行者が手動により操作したい場合には不向きである。

PlanetLab

PlanetLab はさまざまな組織のネットワーク上に設置された計算機を接続し、構築された実験用のネットワークである。利用者は、実験トポロジをオーバーレイネットワークとして構築し実験を行う。Netbed と同様に実験ネットワーク中に実環境のネットワーク特性を導入できるが、同様の問題も存在する。また、Linux の Virutual Server を利用しているため利用できる OS に制限がある。

PlanetLab を用いて構築された実験駆動単位はインターネットと接続されているため、実験対象の技術が実環境であるインターネットに影響をおよぼすおそれがある。したがって PlanetLab を用いた実験は、別の実験実行環境を用いて対象技術の安全性がある程度確認された後に行われるべきである。

VM Nebula

VM Nebula は、米 VMware 社の仮想機械を実現するソフトウェア VMware[10] を用いて構築されたセキュリティ実験向けの大規模実証環境であり、NICT により開発、運用されている。ウィルスやワームなどにより攻撃されたノードは OS 自体に変更が加えられてしまう事が多い。VM Nebula では VMware のディスクイメージ保存機能を用いて、容易にディスクの状態をあるタイミングまで戻す事ができるだけでなく、汚染された OS の状態を保存できる。ただし、実ハードウェアを接続する機構を持たず、また、VMware の制限上 i386 PC にのみ対応する。

支援ソフトウェアは、トポロジの構築に対応しているが、シナリオ実行はサポートしていない。

SIOS

SIOS は、NICT により開発、運用されている、既存の攻撃のパターンにしたがい攻撃を再現するための大規模実証環境である。既存の攻撃パターンをデータベースとして保持し、攻撃トラフィックを生成する Attacker 部と攻撃を受ける Victim 部により成り立っている。

GARIT/AnyBed

GARIT は奈良先端科学技術院大学院大学に設置されている、種類や性能が均一でないノード群からなる実験設備で、このような混沌としたノード集合を扱うための実験支援ソフトウェア AnyBed[11] により制御される。ただし AnyBed は L2 スイッチを用いて実験トポロジを構築する実験設備に汎用化して設計された支援ソフトウェアであり、GARIT に特化したものではない。また、AnyBed は実験トポロジを構築できるが、シナリオの実行機能は備えていない。

StarBED/SpringOS

Netbed や PlanetLab を用いた場合には、遠隔地にある計算機の制御が困難である場合が多く、また、それぞれを接続するネットワークは実験実行者の管理下に無いため、その部分に関しての情報の取得や、トラブルが発生した場合の修正作業が困難である。そこで、一カ所に実験用の計算機を集め、その計算機を自由に設定することで、より柔軟な実験トポロジを構築できると考えた。また、Netbed のような環境では一括処理で実験をおこなえるため、容易かつ効率的に実験を行うことができるが、その一方、実際に実験を行い、その結果から、実験内容を修正して繰り返し実験を行うといった方法での実験は困難である。このような、実験を試行錯誤しながら行うといったスタイルのために、汎用的なネットワーク実験に利用できる StarBED を設計・実装した。StarBED は 680 台の PC ノードとそれらを接続するためのスイッチノードからなる。また支援ソフトウェアとして SpringOS が用意されており、自動的な実験トポロジの構築とシナリオ実行が実現されている。StarBED の詳細は 6 章で、SpringOS については 7 章で述べる。

2.3 実環境と実験駆動単位

実験実行環境は、インターネットなどの実環境を模倣するための環境である。本節では、模倣対象である実環境の性質と、実験実行環境上に構築された実験駆動単位の性質についてまとめる。

2.3.1 実環境の性質

本論文では、実環境を自律分散したノードとリンクの集合と定義する。実環境上に存在するノードのハードウェアやその上で動作するソフトウェアの種類、実装はまちまちである。これらのノードは、たとえ同じ仕様にそった実装であっても、実装の手法や手段によりその挙動や性能は異なる。また、実装からバグを排除することは不可能であり、挙動や性能を多様とする原因の一つとなる。

多種多様なノードおよびリンクが組み合わされて構築されたネットワークの挙動の予測は非常に困難である。また、インターネットは、さまざまなポリシを持つ

ネットワークが相互に接続され構築された環境であり、多数のノードやリンクによる複雑な挙動をみせるネットワークをさらに組み合わせた環境であるため、そのネットワークトポロジは多様性に富んだものとなっている。このようなトポロジ上で、多様なアプリケーションソフトウェアやネットワークサービスが動作し、それぞれのトラフィックが影響をおよぼしあっているため、トラフィックも非常に複雑なものとなっている。

2.3.2 実験駆動単位の性質

現実に即した実験結果を得るためには、対象の実環境とまったく同じ環境が求められる。しかし、これを実現するためには、対象の実環境と全く同様のハードウェアおよびアプリケーションソフトウェア構成のノードを、実環境と同様の位置、同様の接続方法で接続し、さらに同様の挙動を示す利用者を用意する必要があり、現実的ではない。したがって、実験実行者は、実験実行環境上に対象環境および対象技術を何らかの手法により、モデル化や抽象化し実験駆動単位を構築する。当然、このような実験駆動単位の挙動は、実環境と異なる。

2.3.3 実験駆動単位の構成要素

実験駆動単位は、トポロジを構成するノード群およびリンクとその上を流れるトラフィックから構成され、ノード群、トラフィック共に実験対象のものとそれ以外の部分に分けられる。以下でそれぞれを説明する。

対象要素 実験の対象のノード、OS、アプリケーションソフトウェアおよびリンクをさす。これらの挙動の観測が実験の目的であり、さまざまな視点から観測が行われる。

対象付属要素 実験対象そのものではないが、実験対象が動作するために必要になる要素。サーバクライアントモデルの技術であれば、サーバが実験対象であってもクライアントが必要である。

周辺要素 実験トポロジ中の実験対象外の部分をさす。対象要素に影響はないと考えられる要素であり、現実的なネットワークトポロジを構築するために用いられる。大規模化・複雑化するための実験対象を接続するネットワーク部分を構成するノードやアプリケーションソフトウェア、リンクなどがある。

対象トラフィック 実験対象および対象に関連する付属要素により生成されるトラフィックをさす。

バックグラウンドトラフィック 実験には直接関係のないトラフィックをさす。実環境では、必ず対象技術に関係のないトラフィックも流れており、より現実的な環境を模倣するためにはこのようなトラフィックも必要である。

実験駆動単位での対象技術およびその付属要素に着目した場合、周辺要素の多様さや規模は、そこに流れ込んでくるトラフィックの多様性や量として観測される。したがって、現実的なバックグラウンドトラフィックを実験駆動単位に導入することで、実験駆動単位の規模が小さく要素の多様性が低い場合でも、実験用としては十分な規模および多様性を持つ環境を構築できる。

以上であげた要素すべてを、すべての実験で用意する必要はなく、それぞれの目的により取捨選択が必要である。最低限の環境での動作試験では、実験対象と対象付属要素が必要であり、これらにより実験対象のトラフィックが生成される。実環境を考慮した周辺要素無し環境での動作試験では、導入環境を考慮し、対象要素と対象付属要素を導入先環境と同様に配置して実験を行う。実環境を考慮した環境での動作試験を行う場合には、対象技術に直接関係ないと思われる周辺要素を導入し環境を構築する。また、これらの要素によりバックグラウンドトラフィックが生成される。さらに現実的なバックグラウンドトラフィックを導入するため、実環境で採取されたデータを元に、バックグラウンドトラフィックを生成することもある。

2.3.4 実環境と実験駆動単位の差異

実験実行環境を用いて構築された実験駆動単位は実環境を模倣したものであり、完全に同一のものではない。本節ではこれらの差異について述べる。

前述のとおり、実環境は自律分散したノードとリンクの集合であり、それぞれの管理者は異なる。このような環境では、一口に計算機やOS、ネットワーク機器といっても、非常に多種多様なものが存在し、また、それらの上で動作するアプリケーションソフトウェアや、提供されているネットワークの運用方針等も多岐に渡る。また、同一の目的、ネットワークサービスを提供するためのものであっても、その実装および動作は異なる場合が多く、RFC[12] などの標準仕様にしたがっているものであってもその挙動は異なる。実験駆動単位には、実験実行環境が提供する範囲内の多様性を持たせることしかできない。また、インターネットなど巨大な実環境を対象と考えた場合、一般的には実験実行環境でその規模を模倣するのは不可能といえる。したがって、実環境と実験駆動単位の差異は、構成要素の多様性と規模の2点にある。

対象とする実環境と全く同じ実験駆動単位上で実験を行うことができれば、その結果は現実的な理想的なものであるが、前述のとおり実環境と同一の実験駆動単位の構築の実現はほぼ不可能である。したがって、既存の実験実行環境は、実環境をある程度モデル化し、必要な部分のみを実験駆動単位上に実現することで、実験の目的に十分応えられると考えられる実験駆動単位を構築する。対象技術に関連する問題を正しく検討し、モデル化・抽象化の手法を選択することで、実験実行者が目的とする部分に限定すれば、実環境と非常に近い挙動をみせる実験駆動単位を実現できる。ただし、必要な部分とそうでない部分を一意に判断することは困難であり、現在は実験実行者の判断に委ねられている。

実験実行環境では、すべての要素を実験実行者の管理下におけるため、さまざまな状況を意図的に作り出したり、各状況での各要素の状態を確認できる。実環境では、実験実行者の管理下に無い要素がほとんどであるため、各要素の設定状況や状態を確認することは難しく、そもそも設定の変更が許されない場合が多い。したがって、実験実行環境を用いた方が実環境よりも理想的な環境を構築することができる場合がある。たとえば、非常時のネットワークを実環境上で実現したい場合でも、実環境を破壊することは許されないため、実現は不可能であるが、実験実行環境上の実験駆動単位ではこのような環境を構築することが可能である。さらに、未来の実環境を現在の実環境上に構築することは不可能であるが、実験実行環境を利用すれば可能である。

2.4　各実験実行環境のアプローチ

本節では、前節で述べた実環境の性質について、既存の実験実行環境での実現方法についてまとめる。実験の実行には、必要な要素や規模の決定が必要であるが、これは現状では実験実行者に委ねられている。実験実行環境は、与えられた実験トポロジ、シナリオおよび測定内容などを忠実に実行する。

2.4.1　ソフトウェアシミュレーション

ソフトウェアシミュレーションでは、対象技術および周辺要素をモデル化・抽象化し実物よりも少ないリソースで実環境を模倣する。すべての要素をモデル化・抽象化することで、重要な挙動・要素のみを重点的に環境に導入しているといえる。不必要であると考えられた部分については省略されるため、実環境との同一性が低下する場合もあるが、実験実行者が慎重に検討を行うことでこれを回避できる場合もある。また、すべての要素がモデル化・抽象化された環境であるため、対象技術の実環境向け実装は利用できないことが多い。汎用的なソフトウェアシミュレータでは、一般的な要素はまえもって用意されていることが多いが、それぞれの要素の細かい部分の挙動はモデル化・抽象化により、丸められている場合が多く、多様性はそう高くないといえる。ソフトウェアシミュレーションでは、大きな要素をモデル化・抽象化し、小さなリソースで実現することも可能であるため、高度にモデル化・抽象化した要素を利用すれば、大規模な実験駆動単位を小さなリソースで構築することが可能である。しかし、高度なモデル化・抽象化が困難な要素が多い場合や、要素が複雑に関連する実験では、非常に大きな計算リソースが必要となり、現実的な時間で実験が終了しないことが多く、規模に対する耐性はそう高くない。また、汎用的なソフトウェアシミュレータでは、さまざまな手法によりトラフィックを生成するためのモジュールが利用できることが多い。

2.4.2　実証環境

実証環境では、実環境上で利用される実装そのものを利用することで実験駆動単位を構築する。実環境向けハードウェア上で実環境向けアプリケーションソフ

トウェアが動作し、アプリケーションソフトウェアが相互に通信を行いトラフィックを生成するモデルは、実環境と同一である。

実験駆動単位に多様性を持たせるためには、ハードウェアやアプリケーションソフトウェアの種類を増やすことが必要であるが、コストの問題から困難である。規模に関しても同様のことがいえる。したがって、ノード自体は実装そのものであるが、トポロジやサービスモデルなどはある程度、モデル化・抽象化することで、コストを削減することもできる。

2.4.3 大規模実証環境

大規模実証環境は、実環境と同様に実環境向けの多数のノードを実際に動作させることで、実環境と同様の複雑な実験駆動単位を実現することを目的としている。

実証環境よりも多数のノードを容易に利用でき、既存の多くの種類の実環境向け実装をそのまま利用できるため、多様性を持たせることはそう難しくない。ただし、ハードウェアに関しては用意されている実験設備の多様性に依存する。規模に関しては、実証環境よりも多数のノードを容易に利用できるが、対象とする環境によっては規模は不十分であるため、やはりトポロジなどについてはモデル化・抽象化が必要となる。

実証環境、大規模実証環境とも、バックグラウンドトラフィックを実ノードから生成することもできるが、1 ノードから大量のトラフィックを生成する手法やアルゴリズムについては、さまざまな研究がなされている。単純な方法として、iperf[13] や netperf[14] といった、帯域などのネットワーク特性のベンチマーク用アプリケーションソフトウェアや、FTP を利用したファイル転送を用いて、バックグラウンドトラフィックを生成する方法が頻繁に利用されている。しかし、このような手法では実環境上に流れる雑多なトラフィックを模倣することはできない。一方、Harpoon[15] のように、実環境上で採取されたトラフィック情報を元にトラフィックを生成するソフトウェアや、Spirent Communications[16] の Smartbits、Agilent Technologies の N2X[17][1] といった同様の手法をハードウェアで実現する製品も発表されている。このような手法を用いればある程度現実的なトラフィックの

[1] 旧称 Router Tester

生成が可能である。

大規模実証環境は、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。しかし、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実ノードを仮想的に多重化すれば、新たな実ノードを用意するためのコストおよび、保守費用も増加しないため、新たな実ノードを用意する場合よりもコストが小さい。

大規模実証環境の目的は実環境向け実装を大規模な実験トポロジ上で対象技術を検証することであり、仮想ノード上で、実験実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、実験実行者が意図するアプリケーションソフトウェアの動作に支障をきたす場合や、実験実行者が変更を加えた OS を利用できないといった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。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等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、実験実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。

ただし、仮想ノード技術を利用した場合は、仮想ノードの性質や性能に影響を受けるため、当該ノードで動作する技術の性能検証や、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。

第3章

実験支援

2 章で、ネットワーク技術の開発段階および実現方法についてまとめた。本章ではネットワーク実験の手順を整理し、支援の可能性について議論する。

3.1 ネットワーク実験実行の問題点

まず、図 3.1 に、ネットワーク実験の実行段階と、それぞれの段階の問題点を図示する。本節ではそれぞれの手順での問題点の詳細について述べる。

3.1.1 実験内容の検討・決定

ネットワーク実験を行うためには、まず実験の内容を決定する必要がある。現状では実験内容の決定は実験実行者に委ねられているため、同様の技術に対する検証実験であっても、その内容は異なる。したがって、実験結果の質は実験実行者によってまちまちであり、また、複数の技術を比較したい場合でも、既存の実験結果の利用が困難であり、新たに比較実験を行う必要がある。

この手順では、どのように適切な実験内容を決定するかと、決定された実験内容の正当性をどう保証するかという問題がある。

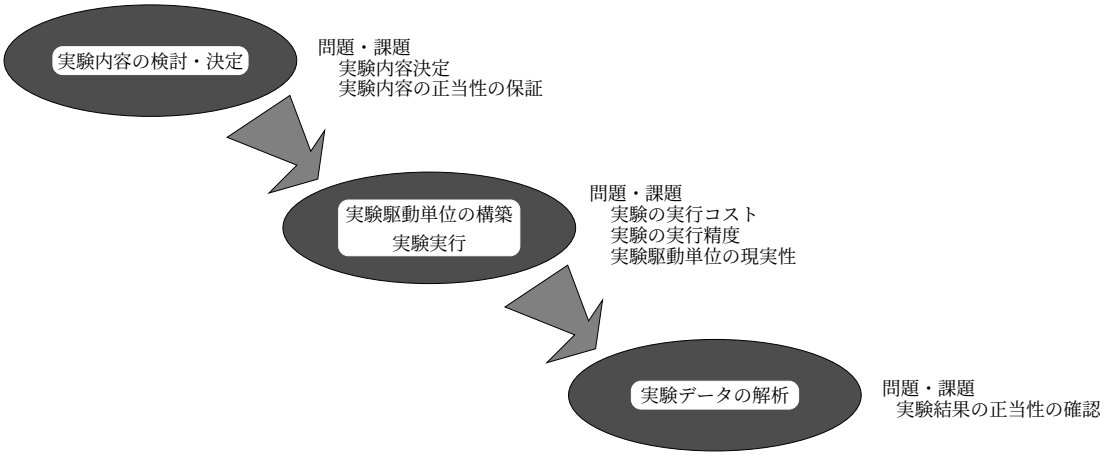

図 3.1: 実験の実行手順と問題

巨視的に見ると、ネットワーク実験の手順には、実験内容の検討・決定と実験駆動単位の構築と実験実行、そして実験データの解析がある。それぞれの手順に問題が存在し、これらを解決することで、より適切で精度が高い実験を行える。

3.1.2 実験駆動単位の構築・実験実行

実験内容の決定後、その内容にしたがい実験を実行する。このために実験実行環境上に実験駆動単位を構築し、実験シナリオを実行する。

利用する実験実行環境を利用可能にすることがまず必要である。ソフトウェアシミュレータでは、必要な機能の実装や基本部分の検証などが必要になる。実証環境や大規模実証環境では、実環境向け実装を用意する必要があるが、最終的に実環境に導入することを考慮すれば、実験のために必要になるわけではないため実験のコストとはいえない。また、実証環境では実験専用の資源を収集する必要がある。大規模実証環境では、実験設備の予約など利用までに処理が必要となる場合が多い。

次に、実験駆動単位を構築し、シナリオの実行を行う。この時の問題として、実験実行者が実験実行に拘束される時間の問題と、実験駆動単位の構築およびシナリオ実行の精度の問題がある。また、実証環境以外の実験実行環境では、実験の設定記述が必要となる。

ここでの課題は、実験実行のコスト削減と、実験実行の精度向上、そして構築

した実験駆動単位と実環境との乖離がないことをどのように保証するかなどが挙げられる。

3.1.3　実験データの解析

実験の実行後には実験データの解析が行われる。実験の結果自体は一つの客観的なデータであるが、このデータを実験実行者の視点から分析し、傾向などを分析する。したがって、実験の内容や実験実行の際の精度が十分であっても、分析方法が適切でない場合は、導かれる結論も適切でなくなる。また、他の実験実行者が行った同種の技術の実験結果を比較するためには、実験内容だけでなく、実験データの解析方法も同一でなければならない。さらに、実験自体が正しかったのか、構築した実験駆動単位が対象とする実環境と十分に近い挙動を示す環境であったのかといった部分も検証をする必要がある。

本論文では、実験により観測された各種数値などを実験データと呼ぶ、それらを解析し、最終的に実験の目的を満足する形に加工したものを実験結果と呼ぶ。

3.2　実験手順の入出力

基本的に各手順では、ある入力に対しての処理を行い、その手順での出力を生成する。これを繰り返して実験が終了する。本節では、各手順の入出力についてまとめる。

3.2.1　実験内容の検討・決定

実験内容の検討・決定手順で利用できる入力情報の一例を以下に示す。

- 設計情報
- 仕様
- ネットワーク実験の段階
- ネットワーク実験の目的

- 技術の種類
- 技術の目的
- 達成すべき目標
- 導入先環境

設計情報や仕様により、境界条件などの問題が発生しやすい条件を知ることができる。また、実験の目的により行われる実験は異なり、実験トポロジの種類や規模、測定すべき内容が異なる。さらに、すでに述べたとおり、実験内容はソフトウェア開発の段階に依存する場合がある。技術の種類や解決しようとしている目的そして、達成すべき目標は、実験のトポロジや、負荷の大きさなどを決定するためのヒントとなる。導入先環境は、実験トポロジの規模や性質、バックグラウンドトラフィックを決定する助けとなる。

これらの情報を何らかの方法で処理することにより、本段階での出力である以下の情報を生成する。

- 実験トポロジ
- 実験シナリオ
- 実験実行環境
- 監視方式
- 運用方針

実験トポロジは実験駆動単位を構成するノードに関する情報と、それらの接続情報とする。実験シナリオは、各ノードでのコマンド発行の順序や、タイミングである。また、利用する実験実行環境を決定する必要がある。これらは実験に直接関連する事項であるが、これ以外に実験の状況をどのように把握するかといった状態監視に関連する事項や、どのような方針で実験を実行するかといった実験の管理に関する事項も必要である。

3.2.2　実験駆動単位の構築・実験実行

本手順での入力は、基本的に実験内容の検討・決定手順での出力である。実験トポロジや実験シナリオ、実験実行環境は実験を行うために直接必要な情報であり、監視方式や運用方式は、実験実行後に実験の正しさや適切さを検証するために必要になる。また、実験実行中の問題検知にも必要となる。

これらの情報を入力とし、実験データと、実験駆動単位の管理データを出力とする。実験データは、基本的に実験対象の技術そのものに関するデータであり、実験駆動単位の管理データは、実験実行中の実験駆動単位全体もしくは、それを構成する要素の状況に関するデータである。

3.2.3　実験データの解析

実験データの解析手順の入力は、前手順である実験駆動単位の構築・実験実行手順での出力である実験データと実験駆動単位の管理データである。実験データをさまざまな方面から解析し、実験結果に加工するとともに、実験駆動単位の管理データとあわせて実験結果の正当性を保証することが必要である。

3.3　実験支援の可能性

前節で述べた各手順の入出力の例を述べた。与えられた入力を処理し、適切に出力する手法を開発することで、実験を支援することができる。本節では各手順での支援方法について検討する。

3.3.1　実験内容の検討・決定

前節で本手順に関する入力と出力について述べたが、直接入力から出力が生成されるのではなく、さまざまな中間情報がある。たとえば、実験トポロジは、実験駆動単位を構成するノードへ要求される機能や性質、トポロジの種類、トポロジの規模などから導かれる。実験項目は、どのような状態でどのような点についての観測を行うかである。技術・実装の検証では、問題点が多数発見されること

は技術・実装の品質が低いという意味で問題であるが、発見された問題が少なすぎることも問題である。新たな技術や実装には問題点が含まれるが、実験項目が適切でなかったり、項目数が少ないことにより、十分な実験が行われていない可能性がある。このため、実験項目が対象技術を評価するに十分対応できているかという点と、項目数が適切であるかは重要である。また、実験シナリオは、確認されるべき項目やその順序が中間情報として導き出され、それらを総合した結果である。このような中間情報を順序よく処理することにより、入力から出力を生成する可能性がある。

実験トポロジに関しては、実験によく使われるものが存在するため、これらの性質を検討し、実験の性質と照らし合わせることである程度決定できる。また、対象の実環境が明らかである場合は、実環境のトポロジそのものを利用できる。さらに、実験トポロジにあわせて、バックグラウンドトラフィックを導入する必要があるのかや、その種類を決定する。

利用できる実験実行環境についても、すでに検討した実験実行環境の性質と実験の性質の関連性を明らかにすることで、決定が可能である。

実験項目については、対象の技術の仕様や、これまで行われてきた実験の項目を検討することで、管理方法については実験実行環境の性質と、実験トポロジ、技術の仕様などから監視するポイントや内容を決定できる可能性がある。

3.3.2 実験駆動単位の構築・実験実行

実験駆動単位の構築および実験実行の支援は、実験実行環境によりその手順や問題は異なる。

専用のソフトウェアシミュレータを利用する場合は、実験内容にそった実験を行えるだけの機能を持つソフトウェアシミュレータを実装する必要がある。これを助ける形で ns-2 や OPNET、SSFnet といった、汎用的な実験に対応できるソフトウェアシミュレータが開発されている。汎用・専用どちらのソフトウェアシミュレータを利用する場合でも、ソフトウェアシミュレータの挙動が正確であるかを確認する手法が求められている。

実証環境では、実験用に用意できるハードウェアやソフトウェアがまちまちで

あるため、支援は困難である。

大規模実証環境は、基本的には実証環境の欠点を補い、大規模な実験駆動単位を容易に構築するためのものである。大規模実証環境では、支援ソフトウェアにより実験実行のコスト削減および、実験の精度の向上が図られている。

どの実験実行環境においても、実験駆動単位の管理データを生成するために、利用した実験実行環境を構成する各種要素の性質や性能などの基礎データを用意することで、実験実行の精度の保証などといった支援に利用できる。

3.3.3 実験データの解析

実験データを実験結果に加工するには、いくつかの段階が存在する。まず、一般的に実験が終了した時点の実験データは、実験のログファイル中に保存されている。これを、解析できる形に加工する。この集計データと、このデータを視覚化した情報から、実験の結論となる実験結果を導出するための支援を行う。したがって、実験データの収集と、データの視覚化で支援を行える可能性がある。

これまでに述べた各実験段階における入出力を図 3.2 に示す。

本研究では、まず、実験駆動単位の構築・実験実行手順に着目する。すでに述べた 3 手順では、実験を行うことがもっとも基本的な手順であるためである。また、実験内容の決定についても後述するが、実験データの解析については言及しない。

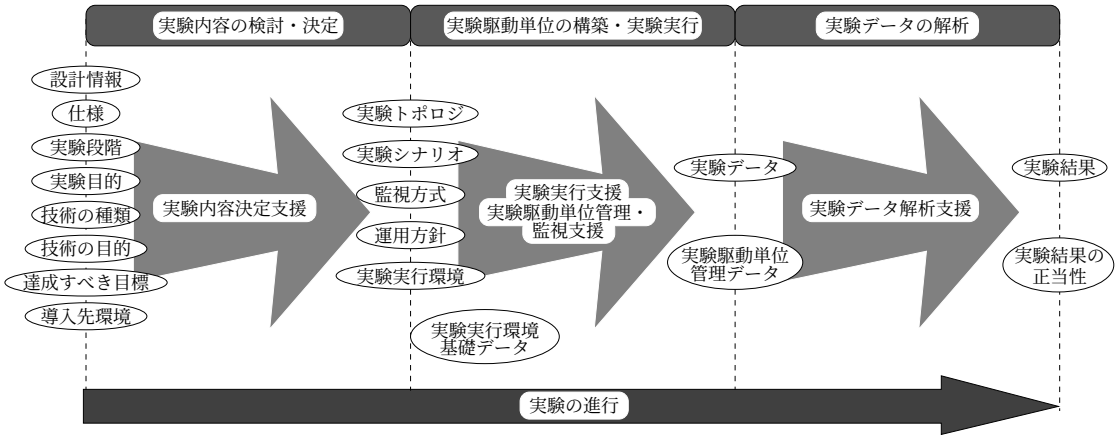

図 3.2: 各実験手順での入出力の一例

実験内容の検討・決定手順では、設計情報や仕様、すでに述べた実験の手順や目的などから実験実行のために必要な情報を決定する。実験駆動単位の構築・実験実行手順では、これらの情報にしたがった実験を行い、実験データの解析手順では、実験データを解析し実験の最終的な結果として出力する。それぞれの手順での処理についてさまざまな支援方法が考えられる。

第4章

実験実行者が実験実行環境に求める性質と機能

前章で述べたとおり、まず実験駆動単位の構築・実験実行手順の支援について議論を行う。このため、本章では、これまで StarBED で行なわれた実験および、さまざまな実験実行者にインタビューを行った結果を紹介・考察する。これにより、実験実行者が実験を行うにあたって、実験実行環境に求める機能と実験の性質の整理する。ここでは、特に実験実行環境に関わらず、複数の研究組織、企業に属する複数のグループを対象にインタビューを行った。

4.1 実験事例

本節では、主な実験事例を紹介する。

4.1.1 SICC の基本性能評価

SICC[18] は少人数のグループを対象にしたマルチキャスト方式 SGM 上で、TCP Fairness および高速輻輳回避、Intra Session Fairness を提供する輻輳制御技術である。

SICC を提案しているグループによる、TCP Fairness、高速輻輳回避、Intra Session Fairness に関する性能評価についての事例を紹介する。

まず、それぞれの性質について簡単に説明する。

TCP Fairness SICC はインターネット上の複数のフローの公平性を制御する。現状のインターネットでは TCP トラフィックが大半をしめるため、SICC ではセッション間のフローとの帯域の公平性のみ考慮している。この公平性を TCP Fairness と呼ぶ。

高速輻輳回避 SICC では TFRC に準じて送信レートを制御する。TFRC はインターネット上で他の TCP フローとの公平性を保ちながら送信レートを変更する方式である。これにより高速に輻輳を回避することが可能となる。

Intra Session Fairness マルチキャストを利用した場合に、ある受信者から見て、送信者から受信者間の帯域が十分に空いている状態でも、その他のある受信者の受信速度により、全体の受信速度が制限され、帯域を有効に活用できない事がある。SICC ではこの問題を解決する手法を提案している。

次に実験内容を紹介する。

TCP Fairness の検証

TCP Fairness の検証用トポロジを図 4.1 に示す。ボトルネックリンクの帯域および遅延、キューの種類、キュー長、TCP、SICC のセッション数を変更し、Router-Router 間で TCP と SICC のスループット比率を観測した。スループットは、SICC 受信側でのログを元に計算している。

実験実行環境には、StarBED と ns-2 と mini-StarBED を利用している。mini-StarBED は実験実行者が自組織内に構築した、8 台の PC からなる実験用の環境である。ここでは支援ソフトウェアとして SpringOS を利用している。ns-2 で論理的な性能検証を行い、その後、mini-StarBED で実践的検証を行っている。StarBED と mini-StarBED はノードの数による使い分けと、実験当時 StarBED では最大 100Mbps のネットワークインターフェースを持つ PC しか存在していなかったため、ボトルネックリンクに 1Gbps の帯域が必要な場合には、mini-StarBED を利用した。

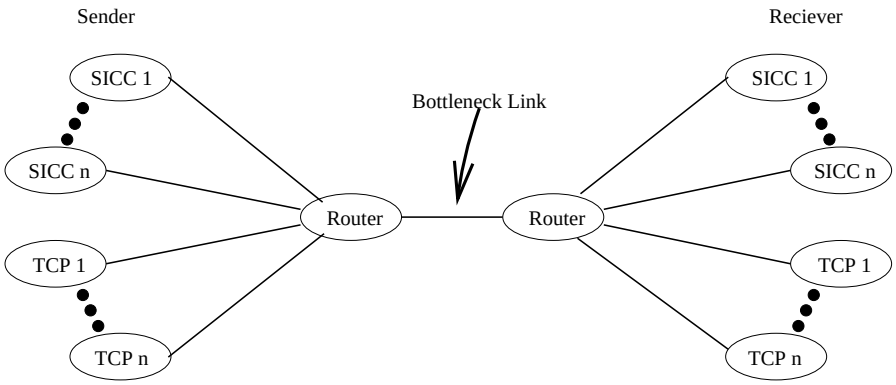

図 4.1: TCP Fairness の検証用トポロジ

観測対象として明示的にボトルネックとなるリンクを持つトポロジを利用し、ボトルネックリンクに TCP および、SICC のトラフィックを流す。ボトルネックリンクの性質を変更することで、さまざまなリンクの状態を対象リンク上に作り出し、それぞれの場合のスループットを観測した。

高速輻輳回避の検証

高速輻輳回避の検証用トポロジを図 4.2 に示す。SICC での通信中に、あるタイミングで UDP によるバースト転送を行い輻輳を発生させた。UDP トラフィックの生成後、どれだけの時間で輻輳回避アルゴリズムが働くのかと、輻輳回避アルゴリズム動作後、どの程度の時間で SICC のトラフィックが元に戻るのかを観測した。パラメータとして、ボトルネックリンクの遅延と UDP トラフィックの帯域を変更した。利用した実験実行環境は mini-StarBED と ns-2 である。対象となるトポロジが mini-StarBED で構築可能な規模だったため、StarBED ではなく mini-StarBED が選択された。

Intra Session Fairness

Intra Session Fairness の検証用トポロジを図 4.3 に示す。図右側の各 SICC reciever と図中央の Router 間の帯域および遅延をそれぞれ異なる値にし、Router と SICC reciever 間の帯域利用率を観測する。帯域利用率の計算は SICC reciever でのログを解析することにより行った。実験実行環境としては、StarBED と ns-2 を用いた。ns-2 で論理的な検証および、ns-2 のモジュール実装として公開されてい

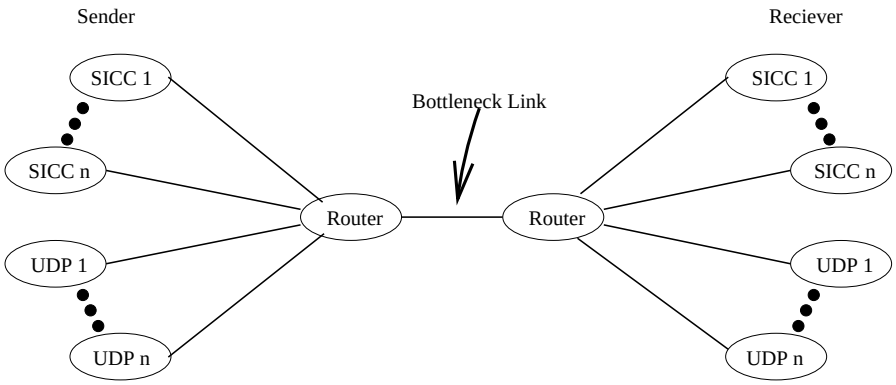

図 4.2: 高速輻輳回避の検証用トポロジ

TCP Fairness の検証用トポロジと同様、ボトルネックとなるリンクを持つトポロジを利用し、SICC のバースト通信を行う。SICC 通信中に、UDP 通信による輻輳を発生させ、観測を行った。

た競合技術との比較実験を行った。StarBED では ns-2 で利用したパラメータを用いて、実践的検証を行った。

これらの実験は、基本的に論理的な開発および検証を ns-2 で行い、その結果から実環境向け実装を構築し、StarBED や mini-StarBED での実験を行っている。期待した論理的性能がでるかソフトウェアシミュレータで検証し、実環境向け実装で論理的性能に近い値が出るかどうかを大規模実証環境で検証している。ソフトウェアシミュレータと大規模実証環境での実験結果を比較することで、実装の問題とアルゴリズム的な問題とを切り分け、対策を行い、次の実験での検証といったフェーズを繰り返すことで、よりよりアルゴリズムおよび実装を構築する狙いがあった。また、ソフトウェアシミュレータでは、パラメータの変更が容易なこと、帯域など物理的な制限を受けないといった点が重要であった。

4.1.2 マルチキャスト対応スイッチの性能検証

マルチキャストに対応するスイッチ実装の性能の検証を行った。図 4.4 に本実験で利用されたトポロジを示す。対象となるマルチキャスト対応スイッチに、直接受信ノードと送信ノードを接続し、受信ノードから join 要求を行うことで、マルチキャストトラフィックを対象スイッチに流す。join 要求を行うスイッチの数お

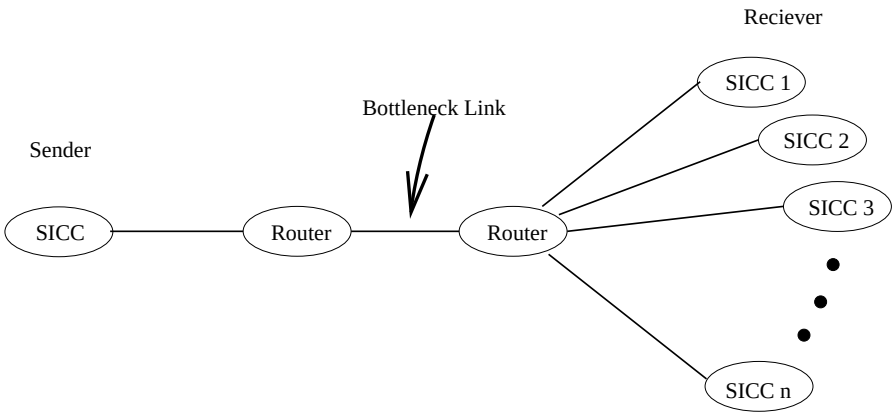

図 4.3: Intra Session Fariness の検証用トポロジ

その他の検証用トポロジと同様にボトルネックリンクを持つトポロジ。各 SICC reciever と Router 間の帯域および遅延を異なる値に設定し、帯域の利用率を観測した。

よび送信するコンテンツの転送レートを変更することで、スイッチの性能を検証する。この実験は StarBED の物理的接続を変更し、200 台程度の実ノードを利用して行った。本実験の実験対象は物理的なスイッチであり、多数の実ノードを接続し負荷をかける必要があった。したがって多数のノードと、管理用の別ネットワークが用意されていることなどが重要であった。

4.1.3 サーバ選択手法の新提案の検証

同じネットワークサービスを提供するサーバが複数存在する環境における、サーバ選択手法の提案 [19] についての検証を行った。利用したトポロジを図 4.5 に示す。トポロジはスケールフリーネットワーク [20] を元にした独自の生成ルールから、現実に即したトポロジを生成し利用している。図中の青いノードがクライアントノードで、その他の色のノードがサーバノードである。色の違いによりサーバノードの負荷を表している。このトポロジにおける特定のリンクのコストを変更したときの、サーバ負荷の変化を観測した。また既存のサーバ選択手法との比較実験も行った。実験実行環境は汎用のソフトウェアシミュレータは利用せず、実験実行者が専用に実装したソフトウェアシミュレータを用いている。本実験はアイデアの確認段階の論理的検証であり、理想的な実験駆動単位の生成とさまざまな性

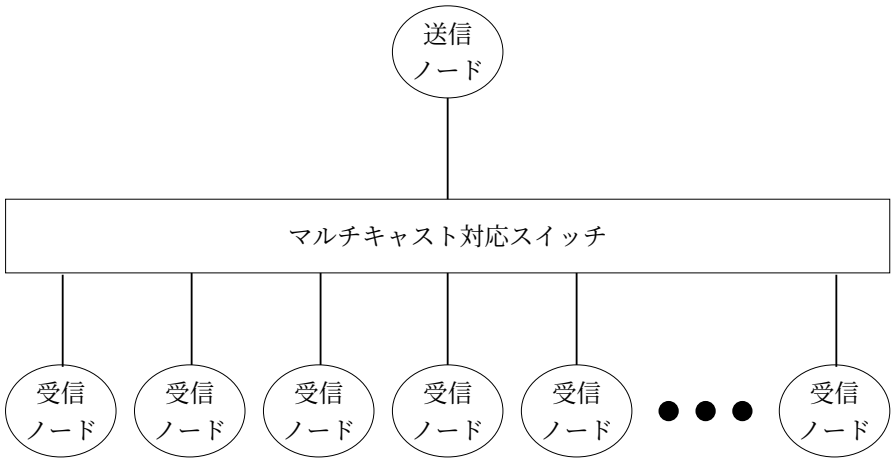

図 4.4: マルチキャスト対応スイッチの性能検証用トポロジ

受信ノードがマルチキャストグループへの join 要求を行うことで、マルチキャストトラフィックを対象スイッチに流す。このときのスイッチおよびその前後での状況を観測した。

質を持つリンク性質、また、リンク性質を容易に変更できることが重要であった。

4.1.4 メガノード IP マルチキャストセンサネットに関する実験

100 万台のノードが存在するセンサーネットワークにおける、マルチキャスト通信を行う場合の通信モデルについて、StarBED を用いて実験を行った [21]。1 台の実ノード上に 5000 程度のノードをプロセス多重により起動し、合計 100 万台のノードからなる実験トポロジを構築し、マルチキャスト通信モデルについて、その挙動を確認している。本実験では、多数のノードからなる実験駆動単位を構築することが必要であり、またそれらを容易に制御できる必要があった。

4.2 実験実行者が求める性質

以上の実験事例から、実験実行者が実験を行う際に重視した実験実行環境に求める性質を整理した。ここで述べる性質は、それぞれ実験の目的により取捨選択されるものである。各性質とも、実験によって求められる程度はさまざまであり、

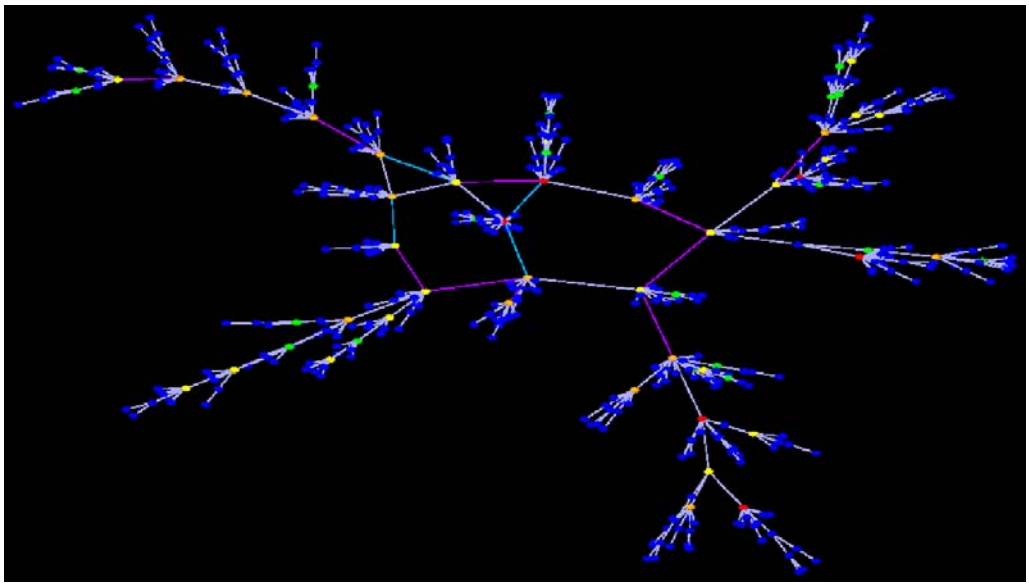

図 4.5: サーバ選択手法検証用トポロジ

DNS のルートサーバのように同一の機能を持つサーバが複数台存在する環境における、クライアント側のサーバ選択アルゴリズムの新提案を検証するためのトポロジ。スケールフリーネットワークの性質を応用し自動生成した。トポロジは 570 のノードにより構成されている。

それぞれの実験にとって十分な性質を持つ実験実行環境を検討することで、より小さなコストで実験を行うことが可能になる。

4.2.1 実環境との同一性

実環境での観測結果と実験実行環境を利用して構築された実験駆動単位での観測結果が大きく異なれば、実験自体の意義が問われる。対象とする実験により、必要とされる程度は異なるものの、実験駆動単位には実環境との同一性が必要となる。

実環境との同一性には少なくとも以下の性質が影響をおよぼす。

実時間性 実験駆動単位での時間の進み方。論理時間である場合と実時間である場合がある。実時間で実験が進む場合は模倣する実験が必要とする時間で実験が終わる。時間に関するデータの取得が必要な場合は実時間性が重要である。また、時間の精度が問題になる場合も多い。

規模追従性 どの程度の規模の実験駆動単位を構築できるかどうか。実験によっては大規模な環境が必要となる場合があるが、実験駆動単位の構築技術により、構築できる現実的な実験駆動単位の規模は異なる。

実験駆動単位の構成要素の挙動 実験駆動単位を構成する要素の挙動の厳密さ。環境により要素を模倣できる度合いは異なる。

4.2.2 実験実行の容易性

実験を実行するために必要となるさまざまなコストは実験実行環境を選択する上で重要な要素となる。

実験実行の容易性には以下の性質が影響をおよぼす。

実験の所要時間 論理時間で実験が進む場合は、実験の進行がシナリオで指定された実験自体にかかる時間と異なる可能性が高く、シナリオで指定された時間よりも短い時間で実験が終了する場合や、逆に長い時間がかかる場合がある。短い時間で結果を得られる方が容易に実験を進められるといえる。

実験準備の所要時間 実験の実行以前に実験に必要なノードおよびトポロジ等の設定に必要となる時間。実験駆動単位の構築に必要になる時間と、設定記述などを用意する時間が含まれる。短い方が実験にかかるコストは小さいといえる。

実験パラメータの変更 実験を特徴付けるパラメータの変更の容易性。一般的に、さまざまなパラメータを変更しながら、実験が繰り返されることが多いため、実験パラメータ変更が容易であれば実験実行にかかるコストが小さくなる。

4.2.3 実験の再現性と検証容易性

実験の再現性は、実験を繰り返し行った際に同じ結果が得られるかどうかである。ネットワーク実験にとっては、常に実験結果が一定であることは重要である。対象技術の性質として結果が一定で無いという場合は問題無いが、それ以外の場合であれば、検証結果が変化してしまうという点で評価指標にできない。外乱がある環境では、外乱が実験結果にある程度の影響をおよぼす可能性はあるが、これは実環境でも同様である。

また、実験後に実験結果を分析、検証するが、どのようなコマンドが実行されたかや、各要素がどのように状態を遷移させたかを確認できなければ、十分な分析・検証が行えない。したがって、実験の検証容易性も重要な性質である。

実験の再現性および検証容易性には以下の性質が影響をおよぼす。

実験駆動単位の構成要素の挙動 実験駆動単位を構成する要素がそれぞれ毎回の実験で同じ動作を示すかどうか。また、同じ入力に対して同様の状態遷移をみせるか。

イベント発生の正確性 コマンド入力などのイベントの発生タイミングが毎回の実験で同一であるかどうか。

操作履歴 実験駆動のためのイベントの発生をどの時点で、どのように行ったかの記録が残されているか。

4.2.4　要素のモデル化・抽象化の可否

要素の集合や大きな要素をモデル化もしくは抽象化し、小さなリソースで実現することで、実験駆動単位の大規模化を行うことができる。また、実験対象の技術に関連が無いと思われる部分を排除しモデル化することにより、実験実行者が想定する理想的な実験駆動単位を構築できる。

4.2.5　身 近 さ

一般的な計算機で動作が可能であれば、実験実行者の判断のみで実験を行えるが、実験専用の設備などについては、他の実験実行者と共用される場合が多いため、利用までの手続きが必要な場合がある。これにより、実験を行うために、実験以外の部分に関するコストが発生する。実験実行者の立場などにより、利用することが困難な実験実行環境も存在する。

4.2.6　管理機構の存在

管理機構は実験実行環境を管理する機構であり、大規模実証環境の支援ソフトウェアなどがこれにあたる。管理機構が存在していれば実験実行が容易であるが、管理機構が想定していない実験を実行できない場合がある。

4.3　実験実行環境に求められる性質と各実験実行環境

各実験実行環境については 2 章でまとめた。4.2 節で述べた実験実行環境の性質を表 4.1 に示す。各性質について表中の○、×、△の意味を以下に示す。また、△は○と×の間とする。

- 実環境との同一性
  - 実時間性: 実時間で動作するものを○とし、実時間で動作しないものを×とした。

  - 規模追従性: 大規模な実験駆動単位を構築できるものを○とし、小規模な実験駆動単位しか構成できないものを×とした。
  - 実験駆動単位の構成要素の挙動（構成要素の挙動）: 挙動が実環境のものに近いものを○、そうでないもの×とした。

- 実験実行の容易性
  - 実験の所要時間（実験時間）: 実験が開始されてから終了するまでの時間が短いものを○、長いものを×とした。
  - 実験準備の所要時間（準備時間）: 実験を開始するまでに必要な時間について、短いものを○、長いものを×とした。
  - 実験パラメータの変更（パラメータの変更）: 変更が容易であるものを○、容易でないものを×とした。

- 実験の再現性と検証容易性（再現性検証容易性）
  - 実験駆動単位の構成要素の挙動（構成要素の挙動）: 構成要素の挙動が常に一定のものを○、変化するものを×とした。
  - イベント発生の正確性（イベント発生）: イベントの発生タイミングが一定のものを○、変化するものを×とした。
  - 操作履歴: 履歴を正確に保存できるものを○、そうでないものを×とした。

- 要素のモデル化・抽象化の可否（モデル抽象化）: モデル化・抽象化に関しては、モデル化・抽象化できるものを○、そうでないものを×とした。
- 身近さ: 実験実行者の身近に存在するものを○、そうでないものを×とした。
- 管理機構の存在: 存在するものに○、存在しないものを×とした。

2.2.1 節で述べたとおり、ソフトウェアシミュレータは、モデル化した各要素や要素集合を利用する。時間に関してもモデル化が行われているといえ、実時間で動作せず、ソフトウェアシミュレータ内で統一された論理時間で実験が進む。ま

表 4.1: 各実験実行環境の性質

	実環境との同一性			実行容易性			再現性検証容易性			モデル抽象化	身近さ	管理機構
	実時間性	規模追従性	構成要素の挙動	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	操作履歴			
ソフトウェアシミュレータ	×	△	×	△	△	○	○	○	○	○	○	○
実証環境	○	×	○	△	×	×	×	×	×	△	△	×
大規模実証環境	○	○	○	△	△	○	△	○	○	△	×	○

た、モデル化を行うことと、利用できる実装が実環境と異なることが多い点から、実環境との同一性は低いといえる。要素を集合としてモデル化すれば、大規模な環境も模倣できるが、実験駆動単位が大規模/複雑であれば、それらを模倣するために必要な計算リソースが大きくなり、現実的な時間で動作しない可能性がある。管理機構が存在するため、各要素のパラメータを容易に変更することができ、事前に想定された実験実行者のシナリオどおりに実験が進行するため、シナリオをコマンドの実行履歴としても利用できる。また、イベントが起った際のログの取得も比較的容易であるため、実験の再現性および、検証容易性に関しては優れているといえる。さらに 1 台の計算機を用意できれば最小限の実験を実行可能であるため、実験実行者に身近であるといえる。

2.2.2 節で述べたとおり、実証環境では、実環境と同一の実装を利用できるため、その挙動は実環境に近いといえる。実ノードの準備や、その設定などに必要なコストが大きいため、実験実行の容易性や、規模追従性は低いが、ネットワークなどの比較的大きな要素をモデル化・抽象化により、1 台のノードで表現することは可能である。また、実証環境には実験支援ソフトウェアが存在しないため、実験に必要なコマンドの実行が自動化されていないことと、実ノードの状況は常に変化し

ている点などから再現性は低いといえる。検証容易性に関しては、ログを収集する仕組みがないため優れているとはいえない。実証環境を構築するためには、実験実行者は身の回りから利用されていない実ノードを用意する必要がある。したがって身近さを考えると、ソフトウェアシミュレータほどは身近でないといえる。

2.2.3 節で述べたとおり、大規模実証環境には、多数の実ノードが存在し、実験支援ソフトウェアが存在するため、実環境との同一性を保ちつつ、規模追従性、実験容易性、再現性および検証容易性について優れているといえる。ただし、大規模実証環境の数は多くなく、利用のためには申請などが必要であったり、他の実験実行者が利用している期間は利用できないといった制限があるため、あまり身近ではないといえる。

4.4 実験実行者が実験実行環境に求める機能

本節では、実験実行環境に求める機能を整理する。これらの機能は基本的にすべての実験実行環境に統一的に要求される機能である。

4.4.1 設定記述にしたがった自動的な実験実行

実験実行者の目的は、実験対象の挙動や性能を知ることであり、実験の実行自体に労力をかけることは本質ではない。また、実験実行者が実験を行うために必要な手順をすべて手作業で行う場合、実験実行に必要な時間および手間は大きい。これは、実験駆動単位を構成することと、実験シナリオの実行のために、常にノード状態を監視し、状況にしたがったコマンドの実行が必要であるためである。また、これ以外にも実験の実行のためにはさまざまな制御が必要となる。まえもって実験実行者が記述した実験内容にしたがい、管理機構が実験を自動的に進めれば、実験の準備および、実験実行にかかる手間を軽減できる。また、手動での実験実行は、設定やコマンド実行のミスの発生や、シナリオ実行の際のタイミング調整などが困難であるため、機械的に実験の準備および実行を行うことで、実験結果の精度向上も期待できる。

4.4.2 実験トポロジの柔軟な設定

実験実行者が、実験の対象となるトポロジを構築するため、実験実行環境に存在するノードの配置などを検討することは対象のトポロジが大規模になるほど手間がかかる作業である。設定記述にしたがって自動的に必要な機能を持つノードを割り当て、目的のトポロジを構築できることが要求されている。トポロジの変更や手作業によるスイッチの設定なども自動的に行うことで、実験実行の手間の軽減および実験結果の精度の向上が期待できる。また、ベンチマークに関する実験などでは、測定対象のリンクの帯域やジッタなどを制限したいという要求が強い。このような要求を満たすため、設定に記述されたリンク性質の自動設定も求められている。

4.4.3 実験用要素の自動設定

実験実行者は、実験駆動単位を構成するために実験用要素の適切な設定を行う必要がある。

実ノードを利用する実験実行環境であれば、実験用ノードへ、必要なOSやアプリケーションソフトウェアなどのをインストールし、それらの設定を行う必要がある。このような作業は、多くのノードに対して同様の設定を行うことが必要であり、効率的とはいえない。この手間を低減するため、設定記述に記述された内容にしたがい、ノードへの必要なOSやアプリケーションソフトウェアの導入および設定の自動化が求められる。また、ソフトウェアシミュレータでも、必要なモジュールなどをそれぞれの要素に読み込ませるなどの作業が必要である。

4.4.4 シナリオの自動実行

実験実行者の意図する順序・タイミングで、ノード上でイベントを発生させることにより実験シナリオを実行できることが求められる。前述のとおり、これにより実験実行の手間の低減だけでなく、実験結果の精度と実験の再現性の向上も期待できる。

4.4.5　ノードの時刻の同期

実験のログを各ノードに残す場合は、そのタイムスタンプで実験の経過を確認することとなる。このような場合にノード時刻が正しく同期できていないと、実験実行後の実験データの解析に支障をきたすおそれがある。したがって何らかの方法でのノード時刻の同期が求められる。ただし、実際の時刻に合わせる必要はなく、実験駆動単位に存在するノードで時間が正確に同期できていればよい。

4.4.6　実験状態の把握

実験実行中には各ノードやトラフィック、トポロジの状況を把握することで、実験の進行状況や、トラブルの発生状況を確認できる。このような機能が無い場合は、想定外の状況が発生しても、実験実行後、結果を確認しなければその状況が把握できず、実験自体が無駄になる可能性もある。そのため、容易に実験駆動単位の情報を取得できることが求められる。

4.4.7　ログの自動収集

実験実行者は、実験実行後に実験用ノードのログを収集し、その結果から対象技術の分析を行う。多数のノードによる環境を利用した場合は、収集すべきログは多数のノードに分散していることも多く、ログを収集する手間が大きい。この手間を低減するための自動的なログの収集が要求される。

4.4.8　環境の初期化

実験駆動単位を構築するためには、実験実行環境の資源に対してさまざまな設定を行うことになる。このような設定が、次の実験を実行する際に残っていると障害の原因となりかねない。したがって、実験が終わったタイミングで環境を自動的に初期化することが必要である。

4.4.9　他の実験実行者との資源分離

複数の実験実行者で実験実行環境を共有し、それぞれの実験駆動単位を構築することがある。このような場合には、他の実験に影響をおよぼさないよう、資源を分離したいという要求がある。

4.4.10　実験状態の保存

複数の実験実行者で実験実行環境を共有していた場合に、利用できる期間が制限されることがあり、実験を中断・保存し、後日、その時点からの実験の続行をしたいという要求がある。また、ある状態での実験駆動単位や実験用ノードの状態を保存し、後で解析するために実験状態の保存についても実現が望まれる。

4.4.11　複数の実験実行環境の接続

各実験実行環境はさまざまな性質を持つことはすでに述べた。また、ソフトウェアシミュレータや大規模実証環境の実装は複数あり、それぞれが持っている機能にも特徴がある。たとえば、StarBED や Netbed、PlanetLab は特定のネットワーク実験に特化したものではなく、汎用的な実験設備であり、さまざまな用途に利用できる。一方、SIOS や VM Nebula といった大規模実証環境は、セキュリティ関連の実験に特化して設計されている。したがって、それぞれの実験実行環境を、実験トポロジの適切な部分に配置し利用することで、実験トポロジの規模の向上だけではなく、実験実行者の意図する実験を、より柔軟に実行できる [22][23]。このような実験を容易に実現するため、実験実行環境を相互に接続し、透過的に操作したいという要望がある。また、ここでは大規模実証環境の例のみ挙げたが、ソフトウェアシミュレータ同士や、ソフトウェアシミュレータと大規模実証環境を接続したいという要望がある。

4.4.12　実験要素の基本性能の検証

実験を行う際には実験駆動単位を構成する各要素の性質や性能がわかっていないと、最終的な実験結果の正当性を検討することができない。したがって、実験

実行環境に存在するさまざま要素の基礎データはまえもって収集されていることが望まれる。

以上が実験駆動単位の構築・実験実行手順を支援するために実験実行環境に求められる機能であるといえる。これらは基本的に実験を実行するために必要な機能であり、これらの機能が協調することで、実験内容の検討・決定手順で決定された内容を入力として受取り、実験を実行する。

第II部

大規模実証環境の開発運用と評価

第 II 部では、3 章で述べた実験駆動単位の構築・実験実行手順のうち、大規模実証環境に着目した支援手法について述べる。論理的な検証手法であるソフトウェアシミュレーションに関しては、汎用的なソフトウェアシミュレータが開発・利用されており、それらにより上記手順の支援がなされている。しかし、大規模な実験設備はまだ数が少なく、その議論は十分でない。そこで、大規模な実験設備である StarBED と、StarBED 用の実験支援ソフトウェアである SpringOS の開発と運用を続けてきた。5 章で大規模実証環境の設計として、実験設備および支援ソフトウェアに必要とされる要件をまとめ、それをふまえた上で 6 章で StarBED の詳細について、7 章で SpringOS の詳細について述べる。また、8 章で StarBED と SpringOS の評価をまとめる。

第5章

大規模実証環境の設計

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対象技術に影響するのか、また、新たな技術が既存の技術におよ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実行者がまえ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おそれがある。実証環境では、実環境と同様に、それぞれのノード上のプロセスが自律的に動作し、実験実行者が想定していなかった環境を産み出す可能性は大きく、実環境により近い実験結果が得られるが、一般的に実現できる実証環境の規模およびそれを構成する要素の多様性は、実環境とは大きく異なる場合が多い。このため、実証環境でのみ検証を行っても、実環境に対象技術を導入した際に、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る。一般的なネットワーク技術の開発手順を図 5.1 に示す。これに対し、大規模実証環境による、より現実に近い環境で対象技術の検証を行うことで、対象技術を実環境に導入した際に発生する問題をできるだけ、検証段階で発見することを目指す。本論文で提案する開発手順を図 5.2 に示す。

実証環境を利用した場合、実験駆動単位の構築および実験実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。

これらのコストを軽減するため、多数の実験用ノードを持つ施設を構築することで、実証環境を構築するたびに実ノードを調達するコストの低減を図った。この

施設を複数の実験実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。ただし、このような実験設備には、非常に多くの実ノードが存在し、手動での実験トポロジ構築は困難であるため、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実行にも、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、実験実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ソフトウェアを導入する支援ソフトウェアを用意し、実験実行の大部分を支援ソフトウェアにまかせることで、実験実行コストの削減と、人為的な実験精度の向上を図る。

本節では、このような環境を想定した実験の実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。

5.1 実ノードを利用した環境での実験の実行手順

一般的な実ノードを利用した環境での実験の実行手順について、支援ソフトウェア無しの場合を図 5.3 a）に、支援ソフトウェア有りの場合の手順を図 5.3 b）に示す。図中のアラビア数字で表した手順が実験支援ソフトウェア無しの場合、ローマ数字で表した手順が実験支援ソフトウェアが有る場合である。

まず、実験支援ソフトウェア無しの場合の手順の詳細を以下に示す。

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実験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。

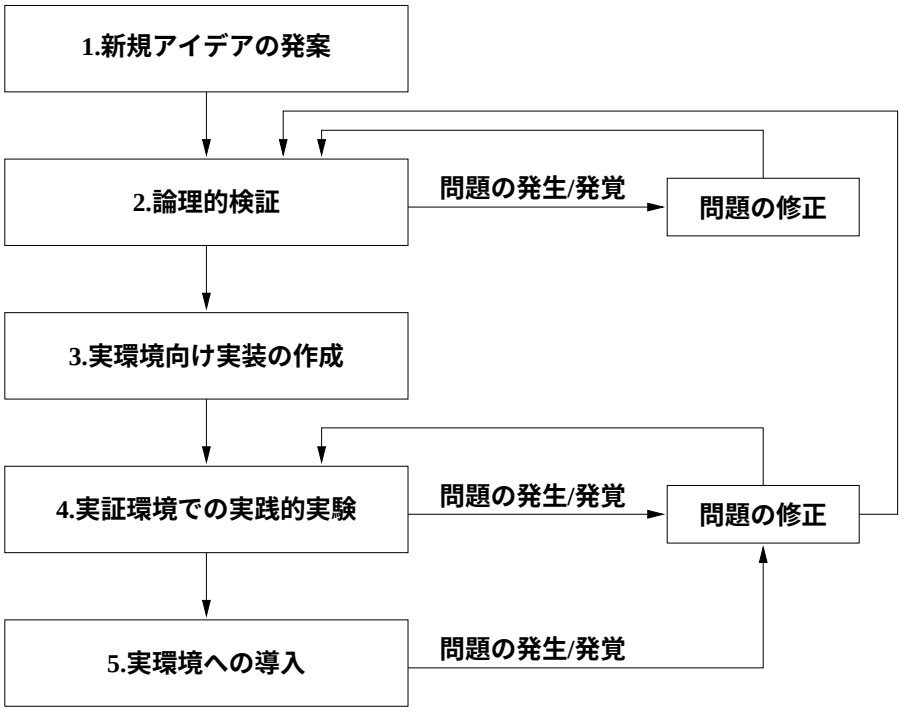

図 5.1: 一般的なネットワーク技術の開発手順

ネットワーク技術のアルゴリズムおよび性能の論理的検証と実証環境での実装の挙動および性能検証のみを行い実環境へ導入する。実証環境では顕在化しない問題も多く、実環境に技術を導入した際に、他のサービスとの関係や規模追従性に関する問題が発生することがある。

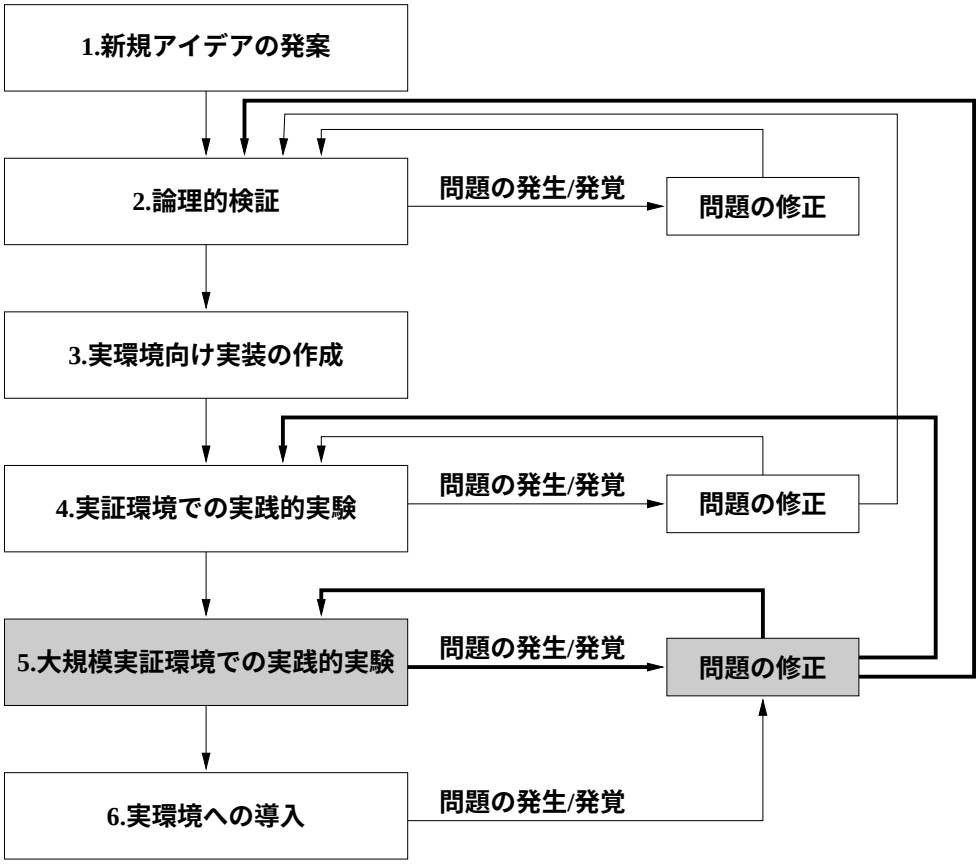

図 5.2: 提案するネットワーク技術の開発手順

実証環境で実装に関する最低限の検証後、実環境に導入する前に、実証環境よりも実環境に近い大規模実証環境においての実験を行う。これにより、規模や多様性などに起因する問題を洗い出し、実環境へ技術を導入する前に問題を修正する。

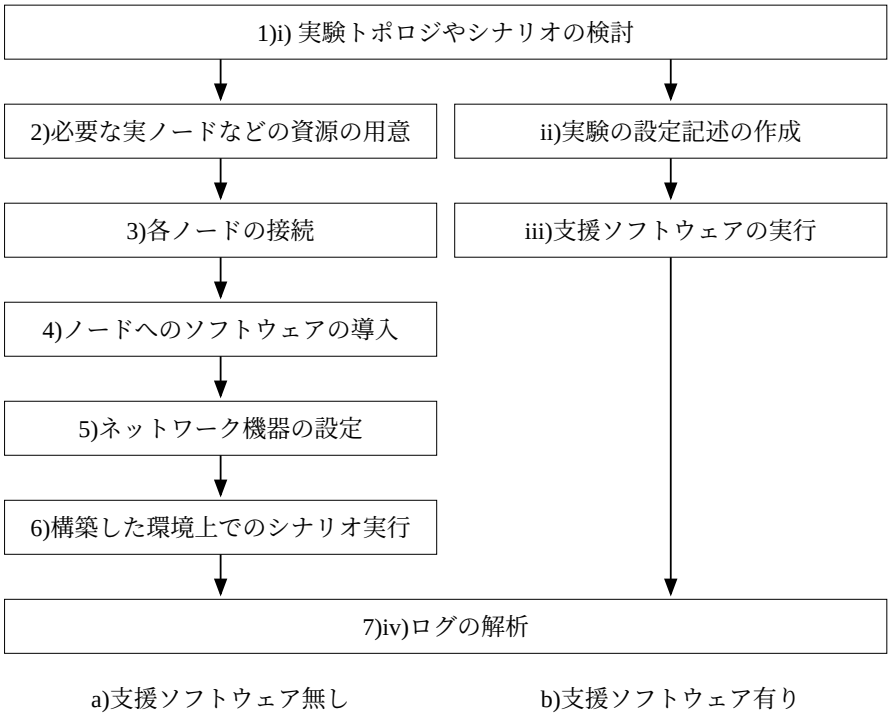

図 5.3: 実験の実行手順

実験支援ソフトウェアを利用できる場合とできない場合の、実験実行手順の比較。実験支援ソフトウェアが存在する環境であれば、実験実行者は設定記述を用意し、実験支援ソフトウェアを実行するのみで、実験結果を得ることができる。

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ソフトウェアを導入し、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実行し、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、まえ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設定記述を入力として受け取り、それにしたがい実験を実行する。この時の実験手順は、図 5.3 b）に示したとおりであり、実験実行者は実験の大部分を支援ソフトウェアにまかせることで、実験による拘束時間を短縮することができる。

以下に支援ソフトウェアを用いた場合の手順の詳細を示す。

i）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。

ii）実験の設定記述の作成 検討の結果にしたがって実験トポロジや実験シナリオなどを記述する。

iii）支援ソフトウェアの実行 設定記述を入力とし支援ソフトウェアを実行する。

iv）ログの解析 得られた実験データを解析する。

これにより、実験実行者は行う実験の検討および実験データの解析に注力できる。また、実験はまえもって記述された手順にしたがって機械的に実行されるため、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。

5.2 実験設備への要件

まずは大規模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 リンクの帯域やパケット損失、ジッタ、遅延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などがある。ただし、ソフトウェアで実現することも可能であるため、必ずしも実験設備で実現する必要はない。

ノードのコンソール操作 ネットワークが切断された際に、実験用ノードへ接続し、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構築 手動でケーブル接続を変更し実験トポロジを変更することはコスト、精度の問題から好ましくないため、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。

電源管理機能 実ノードの電源を制御し、電源の投入や切断、再起動を行える機構である。再起動や電源の切断はソフトウェアにより実現できるが、電源の起動は実験設備が WoL や IPMI、iLO といったハードウェアでの実現手法に対応していなければ実現できない。電源投入は非常に頻繁に行われる操作であるため、このような機構を備えていることが必要である。

5.3 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。

実験実行者による設定の認識・解析 実験実行者による実験用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、実験シナリオを実行する。

実験用資源の状態・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資源の状態およびそれらの資源の属性を管理しておき、その情報をもとに実験に必要な資源を実験実行者へ割り当てる。属性にはノードのアーキテクチャや、標準でインストールされているOS、ネットワークインターフェースの種類や数などがある。また、実験トポロジを構築するための資源として、VLAN番号などがある。複数の実験実行者による空間分割利用のため、排他処理が必要である。

実験用ノードの設定 実ノードにOSやアプリケーションソフトウェアを導入し、必要な設定を行う。

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。設定記述にしたがって、ネットワーク機器の設定を行う。

シナリオにしたがった実験の実行 設定記述にしたがって実験を実行する。シナリオ実行のための、各ノードのコマンド実行の人的・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や精度の向上も期待できる。

実験用ノードの状態管理/表示 各実験用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実行者に通知する。

実験ログ収集 必要なタイミングで、各実験用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 実験設備への要件としてすでに述べたが、Dummynet[24]やNIST Net[25]などのソフトウェアでの実現も可能である。また、実験設備でこの機能を提供している場合は、支援ソフトウェアでその制御を行える必要がある。

また、設備もしくは支援ソフトウェアを用いて、実ノードの電源管理を行える手法が必要である。Magic Packet Technology[26]によるWake on LANやIPMI[27]、iLO[28]などを利用し、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、SNMP[29][30][31]やコマンド実行により、ソフトウェア的に電源の管理を行うこともできる。

第6章

StarBED

StarBED は、実ノードを用いた大規模な実験を行うための実験設備として、通信・放送機構[1]により、石川県能美郡[2]の北陸 IT 研究開発支援センター[3]に設置された。図 6.1 に StarBED の外観を示す。

設置された 2002 年 4 月当初は 512 台の PC ノードとそれらを接続する実験用スイッチおよび、管理用スイッチ、各種管理用機器からなる施設であった。その後、2006 年 4 月から情報通信研究機構 北陸リサーチセンター[4]として運営されている。順次設備の増強がなされ、2007 年 3 月現在では 680 台の PC ノードと実験用・管理用スイッチ、管理用機器が用意されている。

本章では StarBED の詳細について述べる。

6.1 StarBED のトポロジ

図 6.2 に StarBED のトポロジのイメージを示す。図中央のクライアント装置群が実験用の PC ノード群であり、これらのノードはハードウェア構成により 6 つのグループに分けられている。実線で示した PC ノードの左側のネットワークが実験用ネットワークであり、破線で示した右側のネットワークが管理用ネットワークである。図中の実験用スイッチ群は ATM スイッチと LAN スイッチと記述した

[1]現情報通信研究機構
[2]現石川県能美市
[3]現北陸リサーチセンター
[4]http://starbed.nict.go.jp/

図 6.1: 情報通信研究機構 北陸リサーチセンター

StarBED が設置されている北陸リサーチセンターは石川サイエンスパーク内に建設された。本写真は石川サイエンスパーク内から撮影した。

が、実際には ATM スイッチ装置は 3 台、LAN スイッチ装置は A、B1、B2、B3、C2、C3、C4 の 7 台から構成されている。実験トポロジを設定する際には、これらのスイッチの VLAN もしくは ATM の設定を変更することで、仮想的にトポロジを構築する。また、StarBED は、WIDE[32] Internet の 10Gbps のネットワークおよび、Japan Gigabit Network（JGN） II[33] の 10Gbps 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。これにより、外部組織への接続や、実トラフィックの導入が可能である。

すべてのノードは管理用ネットワークに接続されたネットワークインターフェースをもち、管理用スイッチに接続されている。2002 年に導入されたクライアント装置 A から E のノード群は、24 台ずつ 1 台の LAN スイッチ装置 D に接続され、さらに上流の LAN スイッチ装置 F に集約される。LAN スイッチ装置 F には DHCP サーバや TFTP サーバ、FTP サーバといった制御用のノード群が接続されている。2006 年に導入されたクライアント装置 F は、LAN スイッチ装置 C1 に接続されており、各種制御用ノードは C1 に接続されている。

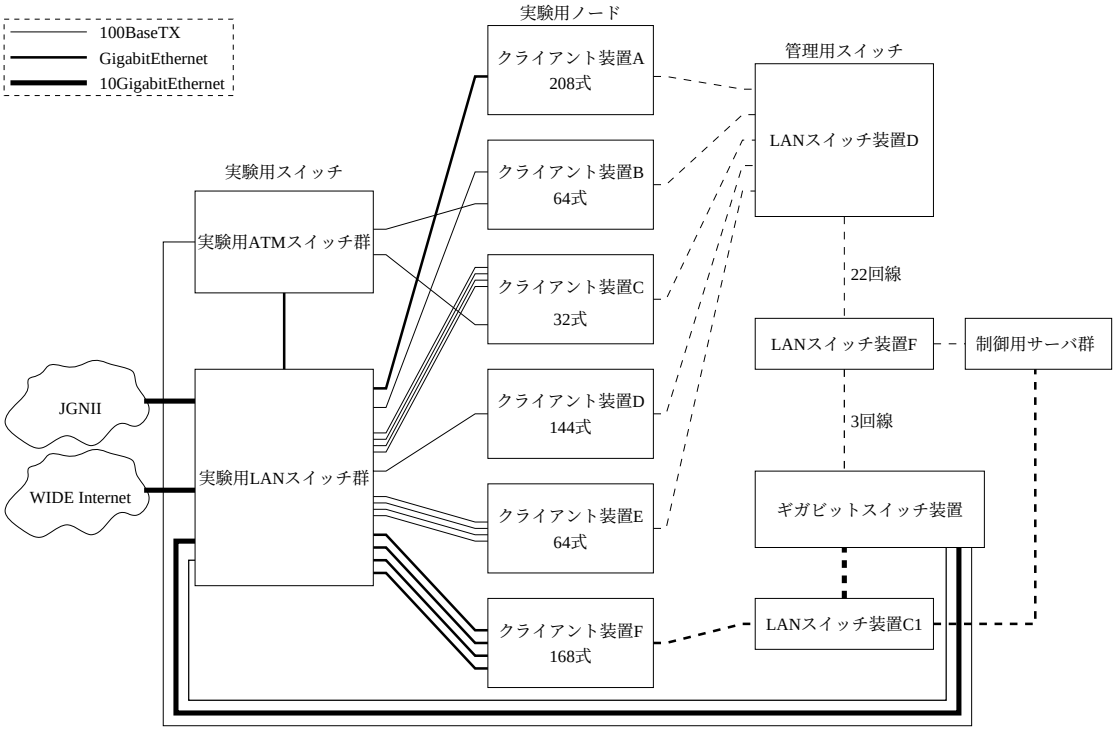

図 6.2: StarBED のトポロジイメージ

図中央のクライアント装置が実験用に用意された PC ノード群である。PC ノード群は必ず 2 つ以上のネットワークインターフェースを持ち、1 つのネットワークインターフェースは管理用ネットワークに接続されている。それ以外のネットワークインターフェースの数はクライアント装置により異なり、それらが実験用ネットワークに接続され、実験トポロジを構成するために利用される。図中では管理用ネットワークを破線で、実験用ネットワークを実線で示した。

実験トポロジは実験用スイッチの VLAN もしくは ATM の設定を変更することにより、仮想的に構築され、物理的な変更作業を必要としない。また、WIDE Internet および JGN II を利用することで外部への接続が可能である。

6.2 ノード構成

StarBED の PC ノードはその構成により A から F の 6 つのクライアント装置に分類されている。2007 年 2 月時点での PC ノードの構成を表 6.1 に示す。表中に示したネットワークインターフェースはすべて実験用のものであり、これとは別に管理用ネットワークに接続されたネットワークインターフェースを持つ。2002 年に導入されたクライアント装置 A-E のノード群は FastEthernet（FE）で、2006 年に導入されたクライアント装置 F のノード群は Gigabit Ethernet（GbE）で管理ネットワークに接続されている。図 6.3 にクライアント装置 A の写真を示す。2002 年の開所当時に設置された PC ノードのラック構成は基本的にすべて同一である。PC ノードが設置されているラックには無停電電源装置（UPS）は設置されておらず、別途 UPS 専用のラックを用意し各 PC ノードはそこに設置された UPS に接続されている。図 6.4 は 2006 年 4 月に新たに導入されたクライアント装置 F の PC ノードである。他のクライアント装置が設置されたラックの構成とは異なり、UPS も PC ノードと同一のラックに設置されている。StarBED の PC ノードは 1 ラックに 24 台が配置され、ラック下のスペースから冷気を通し、ラック上面から排気することでノードの発熱を抑制している。

実験用スイッチには Cisco 社 [34] の Catalyst 6509（図 6.5）、6009（図 6.6）、Foundry 社 [35] の BigIron MG8（図 6.7）、RX-16（図 6.8）が導入されている。表 6.2 に StarBED に設置されている主要なネットワークスイッチを示す。

これらの機器は前述の空調機構が備えられたシミュレータルームと呼ばれる専用の部屋に設置されている。StarBED の利用者は共用研究室と呼ばれる別室から、シミュレータルームのノード群に接続し実験を行える。

6.3 各要件へのアプローチ

5.2 節で実験設備の要件を挙げた。本節では、これらの要件への StarBED でのアプローチを述べる。

表 6.1: StarBED のノード構成

		A	B	C	D	E	F
モデル		NEC Express 5800					
		110Rc-1		120Ra-1	110Rc-1		110Rg-1
チップセット		ServerWorks LE					Intel E7230
CPU	タイプ	Pentium3					Pentium4
	クロック	1GHz					3.2GHz * 2
メモリ		512Mbyte					2Gbyte
HDD	タイプ	IDE		SCSI	IDE		SATA
	容量	30Gbyte		36Gbyte	30Gbyte		80Gbyte*2
ネットワーク	ATM	0	1	1	0	0	0
インター	FE	0	1	4	1	4	0
フェース	GbE	1	0	0	0	0	4
台数		208	64	32	144	64	168
導入時期		2002 年 4 月					2006 年 4 月

図 6.3: クライアント装置 A

2002 年の StarBED 設置当初に導入されたクライアント装置。PC ノード 24 台と、Raritan 社の KVM スイッチ、そして Cisco 社のスイッチ（スイッチ装置 D）が設置されている。

図 6.4: クライアント装置 F

2006 年に追加導入されたクライアント装置。他クライアント装置と異なり、同一のラックに UPS も設置されている。

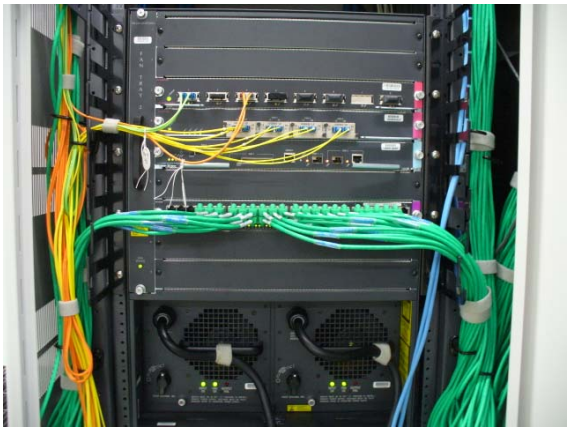

図 6.5: 実験用スイッチ A（Cisco Catalyst 6509）

2002 年の StarBED 設置当時に導入されたスイッチで、クライアント装置 B の 40 台の FastEthernet を収容している。

図 6.6: 実験用スイッチ B1（Cisco Catalyst 6009）

2002 年の StarBED 設置当時に導入されたスイッチで、クライアント装置 B の 24 台、クライアント装置 C の 32 台、クライアント装置 D の 124 台を収容している。実験用スイッチ B2 も同様の構成で、クライアント装置 D の 20 台、クライアント装置 E の 64 台を収容している。

図 6.7: 実験用スイッチ B3（Foundry BigIron MG8）

クライアント装置 A のネットワークインターフェースを ATM から GigabitEthernet に変更した際にそれを収容するスイッチとして導入された。1 モジュール 60 個の GigabitEthernet ポートを持つモジュールが接続されており、クライアント装置 A すべてのノードを収容している。

図 6.8: 実験用スイッチ C3（Foundry BigIron RX-16）

クライアント装置 F を接続するために導入されたスイッチ。図は C3 であるが、BigIron RX-16 が合計 3 台導入された。C2 はクライアント装置 F の PC ノードの 288 ポート、C3 が 192 ポート、C4 も 192 ポートを収容している。

表 6.2: StarBED の主要スイッチ

スイッチ名	利用目的	ベンダー	モデル
LAN スイッチ装置 A	実験	Cisco	Catalyst 6509
LAN スイッチ装置 B1	実験	Cisco	Catalyst 6009
LAN スイッチ装置 B2	実験	Cisco	Catalyst 6009
LAN スイッチ装置 B3	実験	Foundry	BigIron MG8
LAN スイッチ装置 C2	実験	Foundry	BigIron RX-16
LAN スイッチ装置 C3	実験	Foundry	BigIron RX-16
LAN スイッチ装置 C4	実験	Foundry	BigIron RX-16
LAN スイッチ装置 D	管理	Cisco	Catalyst 2924M-XL-E
LAN スイッチ装置 F	管理	Cisco	Catalyst 2948G-L3
LAN スイッチ装置 C1	管理	Foundry	BigIron RX-16
ギガビットスイッチ装置	管理	Cisco	OSR-7609

6.3.1 管理用ネットワークの分離

すでに述べたように、StarBED の各ノードは最低でも 2 つのネットワークインターフェースを持ち、1 つは管理用ネットワークに、それ以外のネットワークインターフェースは実験用ネットワークに接続されている。実験トポロジの設定や、ノードの設定は管理用ネットワークを利用して行われる。実験を支援するためのファイルサーバや DHCP サーバ、支援ソフトウェア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。

6.3.2 リンク特性の模倣

StarBED は、実験設備としてリンク特性の模倣には対応していない。ただし、すでに述べたとおり、StarBED は広帯域の対外線を持っているため、これを用いリンク特性を導入することは可能である。

6.3.3 ノードのコンソール操作

StarBED では Raritan 社 [36] の KVM 装置を導入し、共用研究室におかれた一組のキーボード、モニタ、マウスを利用して、すべての PC ノードのコンソールへ接続し、操作を行える。

6.3.4 自動的な実験トポロジの構築

各ノードの実験用ネットワークインターフェースは実験用ネットワークに接続され、それらは実験用スイッチに接続されている。実験用スイッチは VLAN もしくは ATM による仮想的なネットワーク構築機能を持っている。実験トポロジはこれら実験用スイッチの設定を変更することにより仮想的に構築する。したがって物理的に各ノードの接続変更を行うことは基本的にはない。

6.3.5 電源管理機能

設備としての電源管理機能として、StarBED の PC ノードでは Magic Packet Technology での Wake on LAN が利用できる。ただし Wake on LAN はノードの起動にしか利用できず、ノードの電源断や再起動には利用できない。また、何らかの不正処理などによりノードの OS が停止してしまった場合などに再起動を行うこともできない。クライアント F 装置は IPMI に対応しているため、これによる電源の投入や切断、再起動が行える。

6.3.6 そ の 他

その他にも StarBED では実験を柔軟に行うために以下の設備を備えている。

各種支援機器 Smartbits のような仮想トラフィック生成装置や、パケットスニファなど実験を支援するための機器が用意されている。

仮想機械の導入 StarBED はノード多重化のため、仮想機械の実装の一つである VMware を導入している。最小構成の UNIX 系の OS を仮想機械で動作させる場合には、StarBED の 1 台の PC ノードで、10 台程度の仮想機械を動作させることが可能である。これにより数千台規模の実験駆動単位の実現を可能としている。

ユーザラック 実証環境で利用したノードそのものの利用や、ハードウェア実装を大規模実証環境に接続して検証を行いたいという要望がある。これに対応するため、実験実行者の持ち込み機材を接続するための専用ラックが用意され、StarBED のネットワークへ接続することが可能である。

実験用ノードの状態管理/表示支援 5.3 章で支援ソフトウェアへの要件を述べた。そのなかの実験用ノードの状態管理/表示を補助するために、大規模なスクリーンを備えている（図 6.9）。

物理的配線の変更 実験トポロジを手動での変更はさまざまなコストが発生するため望ましくないことはすでに述べたが、実験によっては物理的配線を変更する必要がある。たとえば、L2 スイッチの性能試験をしたい場合などが挙げ

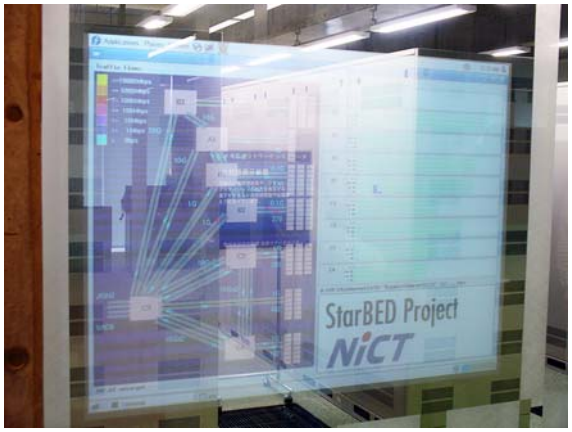

図 6.9: 状態表示用スクリーン

StarBED の共用研究室とシミュレーションルームの間の廊下に 2 面のスクリーンが設置されている。共用研究室とシミュレーションルームはガラス壁で区切られており、透過式のスクリーンがシミュレーションルームのガラス壁に設置されている。プロジェクタはシミュレーションルームの天井に設置されており、スクリーンの後方から投影する。シミュレータールーム前の廊下および共用研究室から映像を確認できる。

られる。各ノードの実験側ネットワークインターフェースの接続先を変更することで、このような実験が可能になる。StarBED では、接続状態や性能などを初期状態に戻しておくことを条件に、このような接続変更が許されている。ただし、SpringOS で実験を制御できない場合がある。

第7章

SpringOSの設計と開発

StarBED のような大規模な実験設備を運用するには、実験支援ソフトウェアが必要である。そこで StarBED の実験設備上で動作する実験支援ソフトウェア SpringOS を設計、実装した。SpringOS は実験支援を行うモジュールの群の総称であり、単一のソフトウェアをさす名称ではない。本章では 5 章で述べた支援ソフトウェアへの要件を充足するための SpringOS でのアプローチおよびその実現方法について述べる。

7.1 実験資源の管理モデル

まず、StarBED と SpringOS が採用している資源資源の管理モデルについて述べる。StarBED のような大規模実証環境では、同時に複数の実験実行者が、大規模実証環境に存在する資源の割り当てを受け、実験駆動単位を構築する。実験駆動単位を構成するために実験実行者に割り当てられた資源は基本的には実験実行者が自由に利用できる。たとえば、1 ノードを単一の実験実行者に割り当てているような大規模実証環境では、ノードの電源の投入や切断は実験実行者の判断で行える。しかし、1 ノードを複数の実験実行者で共有している場合は、ノードの電源を切断した場合に他の実験実行者の実験に影響をおよぼす可能性がある。このような、複数の実験駆動単位で共有される資源への操作に関する問題を排除するため、SpringOS では資源に関する操作を 2 つの管理レベルに分離し、他の実験実行者に影響をおよぼす可能性のない操作は実験実行者の判断で行い、そうでない操

作は大規模実証環境に用意された調停機構を通して操作する。大規模実証環境に用意された調停機構は、実験実行者からの要求を実行可能かどうかを判断し、適切に実行する。ここではそれぞれの管理レベルを、大規模実証環境レベルと実験駆動単位レベルと呼ぶ。

7.2 各種要件の実現アプローチ

5 章で支援ソフトウェアの要件を述べた。本節では、SpringOS の各要件へのアプローチを述べる。

7.2.1 実験実行者による設定の認識・解析

SpringOS では、専用の言語による実験の設定記述を入力として認識する。設定記述には、各実験用ノードの起動の方法や、ネットワークインターフェースの数と種類、そして実行されるシナリオなどが記述される。

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、本設定記述ではクラスの概念を導入した。これにより、一台分の詳細な設定を記述することで、同様の設定の多数のノードを宣言することが容易となる。

この設定記述にしたがい、実験が自動的に実行される。

7.2.2 資源の状態・属性管理

実験用資源として利用できるノードなどの情報を管理するデータベースサーバを用意した。実験資源の情報として、代表的なノード情報に、ネットワークインターフェースの情報がある。MAC アドレスや接続先、種類、数などである。ディスクの種類や状態監視に利用できる手法なども情報に含まれる。SpringOS では実験トポロジの構成に VLAN などを用いているため、VLAN を設定するために必要な VLAN 番号なども重要な資源の一つである。また、割り当て情報についても保持しており、実験実行者からの要求にしたがい、排他制御を行いながら、必要な資源を割り当てる。

7.2.3 実験用ノードの設定

実験用ノードの設定には、OS やアプリケーションソフトウェアの導入など、複数のノードで同一のものが利用できるものと、IP アドレスなどのネットワーク設定のように各ノードで固有のものが存在する。SpringOS では、OS やアプリケーションソフトウェアなど、複数のノードで共通して利用できるものを導入し、固有の設定については OS およびアプリケーションソフトウェアの導入後、シナリオの一部分としてコマンドを実行して設定を行う。

ノードへの OS やアプリケーションソフトウェアの導入には、1）まえもって実験実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、2）ディスクレスシステムとして各ノードを動作させる方法に対応している。

1）の手法では、実験実行者は StarBED の 1 台のノードの割り当てを受け、そのノードに必要な OS やアプリケーションソフトウェアのインストールと複数のノードで共通して利用できる設定を行う。その後、このノードのハードディスクの内容をバイナリイメージ（ディスクイメージ）としてファイルサーバに保存する。ここで作成したディスクイメージを各ノードのハードディスクに書き込むことで、実験実行者が最初に設定したノードと同様のソフトウェア構成、設定を持つノードを複製する。

2）の手法では、ディスクレスシステム用のディスクイメージとカーネルイメージを作成し、これを利用し起動する。

ディスクイメージの生成を支援する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。一方、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利点がある。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 を利用して取得した後、メモリ上に置く方法および NFS[37] を利用する方法が利用できる。NFS を利用した場合はネットワークを経由してディスクにアクセスを行うため、頻繁にディスクにアクセスする場合は、この負荷が実験結果に影響をおよぼすおそれがある。ただしログの記録程度であれば、ログ保存用スペースを実験用ノード上に確保することで回避できる。

7.2.4 実験トポロジの構築

前述のとおり、StarBED では実験トポロジを実験用ネットワークのスイッチの VLAN および ATM の VP/VC を用いて設定するため、SpringOS は StarBED のこの機能を利用し、実験トポロジを設定する。実験トポロジを構成するノードの接続情報はデータベースに保存されている。

7.2.5 シナリオにしたがった実験の実行

実験シナリオは、管理用ノードで動作する Scenario Master と、実験用ノードで動作する Scenario Slave が協調して実行する。シナリオの実行方法として 3 つのモデルを検討した。ここでは、3 つのモデルについて述べる。

ノード自律モデル 実験前にシナリオを Scenario Master が各実験用ノード上で動作する Scenario Slave に配布し、Scenario Slave は配布されたシナリオにそって自律的にコマンドの実行を行う。このモデルでは、各 Scenario Slave が自律的にコマンドを実行するため、各実験用ノードで実行されるシナリオ間の協調が困難である。図 7.1 にノード自律モデルの図を示す。

コマンド送信モデル Scenario Master からコマンド実行のタイミングごとに、実行すべきコマンドを実験用ノード上の Scenario Slave に送信する。各 Scenario Slave は受信したコマンドを単に実行する。このモデルでは、Scenario Master は実験用ノードの台数だけのセッションを制御しなければならず、実験駆動単位が大規模になるほど実装面の問題が発生する可能性がある。また大規模でなくてもあるタイミングに処理が集中した場合には、Scenario Master の負荷が大きくなり、設定されたタイミングどおりにシナリオを実行できない可能性がある。

Scenario Master と、実験用ノード上の Scenario Slave がメッセージを交換することで、実験用ノードで実行されているシナリオ間の協調を行うこともできるが、この処理を行った場合はさらに Scenario Master の負荷が高くなる。また実験用ノード間でメッセージ交換を行い、それぞれが協調するという方法も考えられるが、この方法では各 Scenario Slave が多くのメッセージ交換

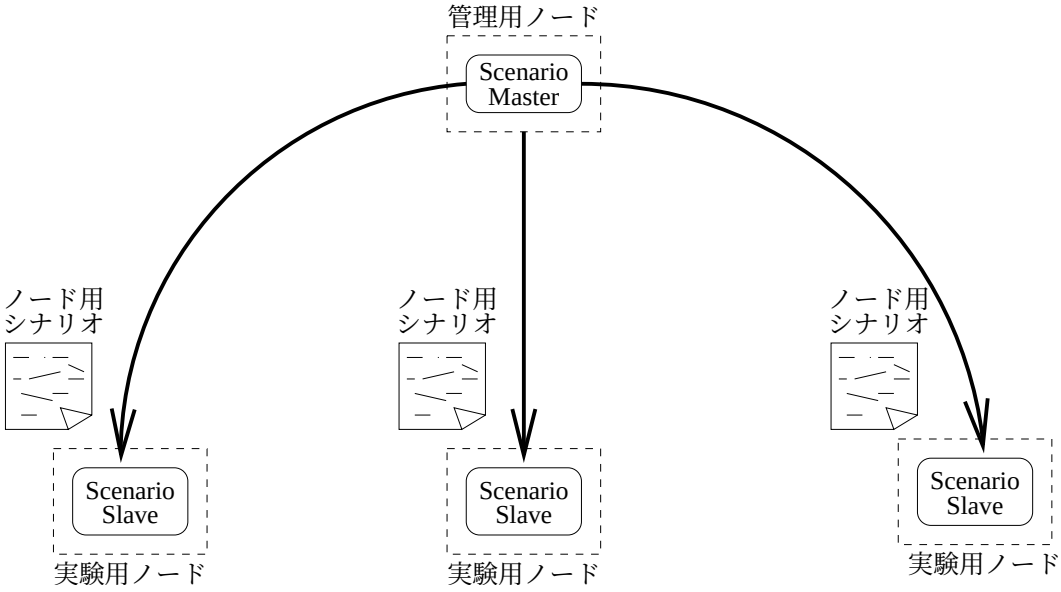

図 7.1: ノード自律モデル

実験前に各実験用ノードのシナリオを各実験用ノード上の Scenario Slave に配布し、Scenario Slave は受け取ったシナリオを自律的に実行する。各実験用ノードで実行されているシナリオ間の協調が困難である。

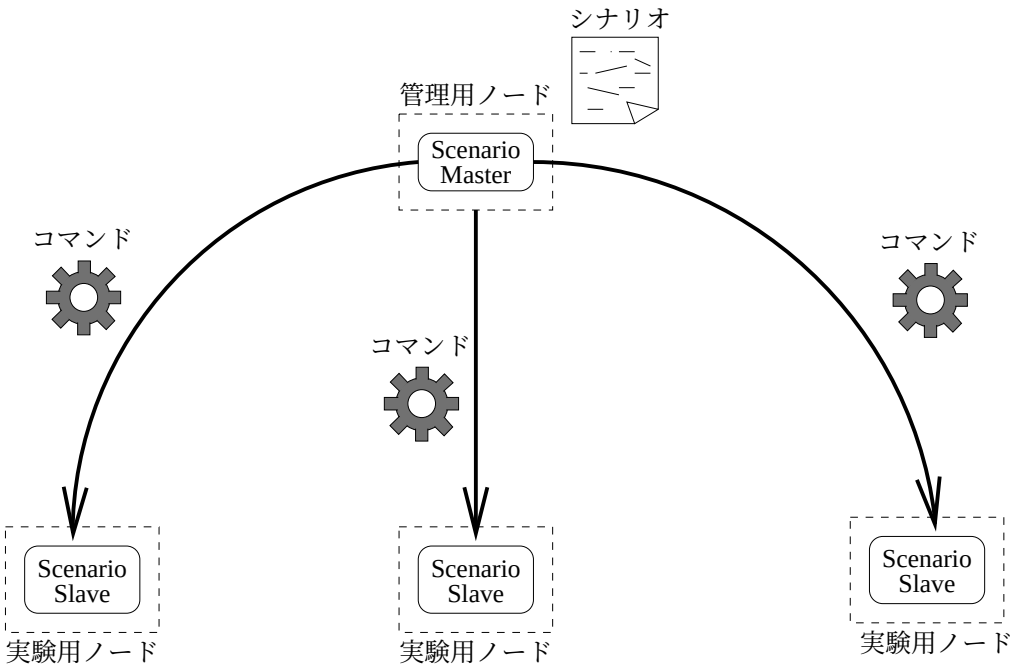

図 7.2: コマンド送信モデル

Scenario Master からそれぞれの実験用ノードに一つ一つ実行するべきコマンドを送信する。ノードの制御精度は高いと考えられるが、実験用ノードが多くなるほど Scenario Master の負荷が高くなる。

用のセッションを保持する必要があり、実験自体に影響をおよぼす可能性がある。このモデルを図 7.2 に示す。

これらをふまえて、まえもってノード用のシナリオを送付し、協調が必要な場合にのみメッセージパッシングを用いる手法を採用した。このモデルでは、Scenario Master は設定記述を読み込み、実験前に各実験用ノード用のシナリオを Scenario Slave に配布する。各実験用ノードが別のノードの挙動をトリガとしてイベントやコマンドを実行するといったノード間のシナリオの同期が必要な場合は、Scenario Master を通じた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ソフトウェア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 Scenario Slave が起動する。Scenario Master は定期的にノードへ接続を試み、Scenario Slave が起動後、接続が成功すると、Scenario Slave は Scenario Master から実行すべきシナリオを受け取りシナリオ実行を開始する。Scenario Slave は基本的にシナリオに記述されたコマンドを実行し、別のノードとの同期が必要な場合

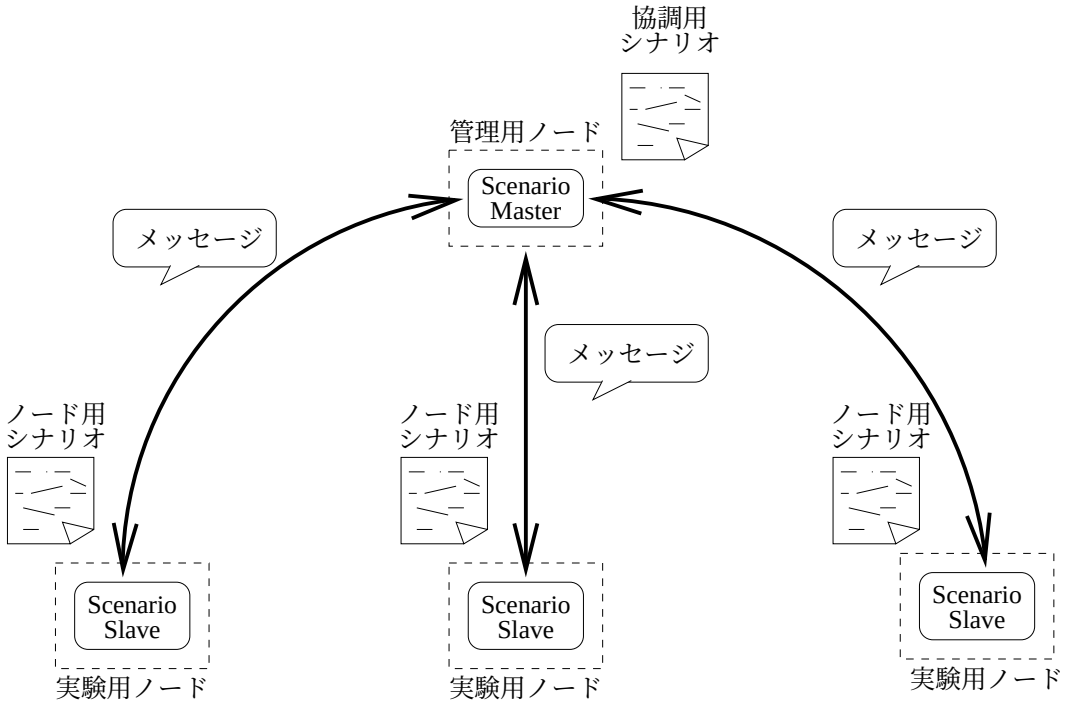

図 7.3: SpringOS での採用モデル

実験前に Scenario Master から各実験ノード用のシナリオを Scenario Slave に送付し、各実験ノード上の Scenario Slave は自律的にシナリオを実行する。他のノードシナリオとの協調が必要な場合にのみ Scenario Master とのメッセージ交換によりシナリオの停止と再開を制御する。

には、Scenario Master へのメッセージ送信や、Scenario Master からのメッセージを受信するまで、シナリオの実行を停止するといった処理を行う。また、Scenario Master は専用のシナリオを持っており、ある Scenario Slave からのメッセージを受信するまでのシナリオ実行の停止や、Scenario Slave へのメッセージの送信を行う。この Scenario Master のシナリオ実行により、各実験用ノードのシナリオ間の協調を行う。

このモデルでは、各実験用ノードのシナリオ間協調が可能となるばかりではなく、協調が必要な時のみメッセージ交換を行うことで、毎回コマンドを送信する場合よりも Scenario Master の負荷が軽減できる。このモデルを図 7.3 に示す。

7.2.6 実験用ノードの状態管理/表示

SpringOS では、各実験用ノードがどのような状態であるかをおおまかに把握している。ディスクイメージ導入中や、シナリオ実行中といった制御に必要な状態の管理であり、実験実行者の指定による状態管理などは行っていない。

7.2.7 実験ログ収集

ログは実験により出力形式や出力場所などが一定でない。したがって、SpringOS では、シナリオ実行によるログの収集を行う。これは特別な機構を用意しているわけではなく、実験実行者がシナリオとして FTP などを用いたファイルの転送を実行することとしている。

7.2.8 リンク特性の模倣

StarBED にはリンク特性の変更のためのハードウェア機器が存在しないため、ソフトウェアにより実現する。これらはシナリオ実行により各実験用ノード上で実現できるため、実験実行者が明示的にシナリオに記述することにより実現する。

7.2.9 電源管理機能

SpringOS では、StarBED の PC ノードがサポートする Wake on LAN によるノード起動と、各ノード上で動作させた SNMP デーモンと通信することによる電源断、再起動を実現する。

7.3 SpringOSを構成するモジュールとその動作

本節では、SpringOSを構成するモジュールについてまとめる。

7.3.1 SpringOSのモジュール

SpringOSは以下のモジュール群により形成されている。ここでは、SpringOS専用に開発したものだけでなく、FTPdなどの一般的なモジュールでもSpringOSの動作に必要なものであれば、その用途を説明する。

TFTPd/DHCPd StarBEDのすべてのPCノードは起動時にPXE[38]により、ブートローダーを取得し、取得したブートローダーを利用して起動する。これにより、DHCPdもしくはTFTPdで対象ノードのブートローダーの設定もしくはその実体を変更することで起動方法を変更する。このため、PXEで利用されるTFTPdやDHCPdが必要となる。

FTPd SpringOSではディスクイメージやカーネルイメージを保存するファイルサーバのインターフェースとしてFTP[39]を用いているため、FTPdが必要である。

Experiment Node Configuration Driver（ENCD）ENCDは実験実行の核となるモジュールであり、ENCDが実験実行者による設定ファイルを読み込み、他のモジュールに対して指示を送ることで実験を実行する。ENCDはある一連の動作をさせるための役割により別々に用意されている。たとえば、実験を実行するためのENCDはkuroyuri masterとして実装されている。また、ディスクイメージを作成する際にも、他のモジュールを指揮するENCDが必要となり、これはpickupとして、また、逆にシナリオの自動実行などをせずに、ディスクイメージを導入するためにwipeoutが実装されている。pickupとwipeoutは基本的には後述するNIと組として利用され、NIと通信することでその役割を果たす。

Experiment Resource Manager（ERM）大規模実証環境に存在する実験用の資源の管理・割り当てを行う。すでに述べたように実験用資源には実ノード

やVLAN IDなどがある。資源の割り当てを受けようとする場合、ネットワークインターフェースの種類や数、ノード性能などをERMに指定し、ERMはその要求を満たす資源を検索、見つかれば割り当てを行う。資源検索の方法は現状ではファーストマッチを利用しているが、さまざまな方式を利用できる。ERMは資源の割り当てを行うと、別の実験実行者に同じ資源を割り当てないような排他制御を行う。

Node Initiator（NI） NIは実験用ノード上で動作し、実験用ノードへのソフトウェア導入を行う。ハードディスクを扱うため、専用のディスクレスシステム上で動作し、ENCDから指示されたディスクイメージをファイルサーバから取得し、ハードディスクに書き込みを行う。

Facility Node Configuration Pilot（FNCP） StarBEDでは複数の実験駆動単位が同時に生成される。ある実験用ノードへディスクイメージを導入するためにNIが起動した際に、NIは通信すべきENCDと通信する必要があるが、ENCDのIPアドレスを固定することはできないため、何らかの方法でNIに対象のENCDのアドレスを指定する必要がある。これを実現する方法として、ENCDからNIに対して通信する方法と、NIが何らかの方法で通信すべきENCDのIPアドレスを調べる方法がある。ENCDからNIに通信する方法では、NIが起動したタイミングをはかりにくく、効率が悪い場合があるため、SpringOSでは後者の方法を採用している。FNCPはStarBED上で一台のみ動作しており、IPアドレスは固定とする。ENCDは自分のIPアドレスと、管理しているノードの管理側ネットワークの固定IPアドレスをFNCPに登録する。NIは起動した際にFNCPに通信を行い、通信すべきENCDを確認し通信を開始する。図 7.4にNIとFNCP、ENCDの関連を図示する。

Wake on LAN（WoL） Agent StarBEDでは実ノードの起動手法の一つとしてMagic Packet Technologyを利用したWake on LANを採用している。Wake on LANでは、対象のノードが存在するセグメントでMagic Packetをブロードキャストする。したがって、管理用ネットワークのセグメントが複数に分離されていた場合には、Magic Packetを中継するための機能が必要となる。WoL Agentは、すべての管理用セグメントに接続されたノードで動作して

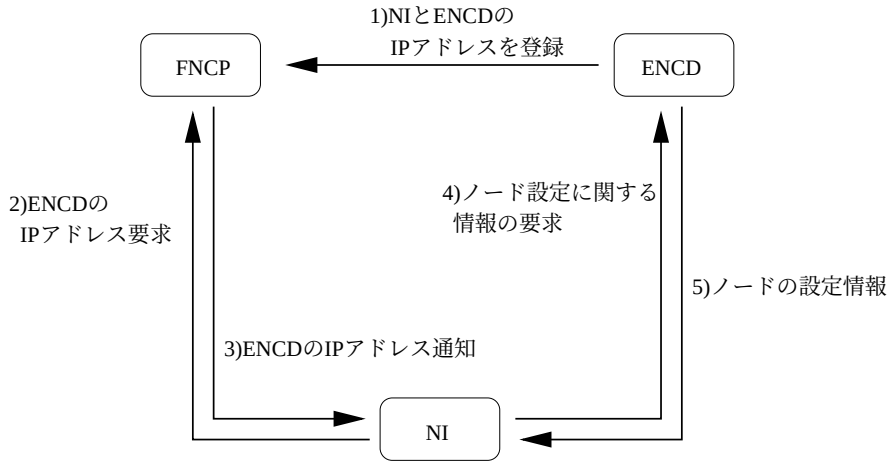

図 7.4: NI,FNCP および ENCD の通信

ENCD は起動時に ERM から割り当てられた実験用ノードの管理側 IP アドレスと、ENCD が動作している管理用ノードの IP アドレスを FNCP に通知する。これにより FNCP はある ENCD と制御される実験用ノードの対応を把握でき、NI 起動時に FNCP に問い合わせを行うことで、NI は対応する ENCD と通信を行える。

おり、ノードを起動する際には WoL Agent に起動要求を行い、WoL Agent が Magic Packet を必要なセグメントに送出する。

Directory Manipulator（DMAN）すでに述べたとおり、実験用ノードは PXE でブートローダーを取得し、その内容により、その起動方法を決定する。各ノードのブートローダーに関する設定は、DHCPd の設定として記述されているため、設定を変更した場合にすべてのノードに対して影響がでる可能性がある。そこで、SpringOS では DHCPd の設定に記述するブートローダーのファイル名は固定し、そのファイルをシンボリックリンクとして、シンボリックリンクがさすリンク先のファイルを変更することで、変更の影響範囲を単一のノードのみとしている。DMAN は、要求に応じてこのシンボリックリンクの設定を変更する。

SWConf 実験トポロジを構成するスイッチは複数の実験駆動単位により共有されるため、スイッチの設定をある実験実行者の判断で自由に変更されるべきでない。したがって、要求された設定内容が、その実験実行者が割り当てられ

ているノードのみに影響を与えるものであること、または要求する設定を行う権利があるかを確認するためのモジュールが必要となる。SWConf は実験実行者からの要求を受け、操作対象のスイッチポートや VLAN の設定に関して、その実験実行者が変更する権利を持つかどうかを ERM に確認し、問題がなければ要求された操作を実行する。

Scenario Master Scenario Master は、Scenario Slave と協調して実験シナリオを進行する。SpringOS では、実験駆動単位の構築を行う ENCD である kuroyuri master が Scenario Master の機能も備えている。

Scenario Slave Scenario Slave は、Scenario Master から実行すべきシナリオを受取り、設定にしたがい実験用ノード上でコマンドを実行する。実験シナリオを実行するものは、kuroyuri master と合わせて kuroyuri slave として実装されている。

図 7.5 に、各モジュールの関連を示す。また、それぞれのモジュールが所属する管理レベルを表 7.1 に示す。

7.3.2 SpringOS の処理の流れ

SpringOS のモジュール群の処理の流れは以下のとおりである。

1） **実験の設定記述の読み込み** ENCD が実験の設定記述を解析し、実験実行者の要求を認識する。

2） **実験資源の割り当て** ENCD は実験駆動単位を構築するために必要なノードおよび VLAN ID を ERM に要求する。ERM は要求された資源を検索し、必要な資源が用意できれば ENCD に対して割り当てる。

3） **ノードへのディスクイメージの導入** ENCD は FNCP に NI からの接続を行うための設定を要求する。また、DMAN に対して対象ノードが NI 用のディスクレスシステムで起動するよう、シンボリックリンクの切り替えを要求する。その後、WoL Agent にノードの起動を要求し、ノードが起動すると、FNCP

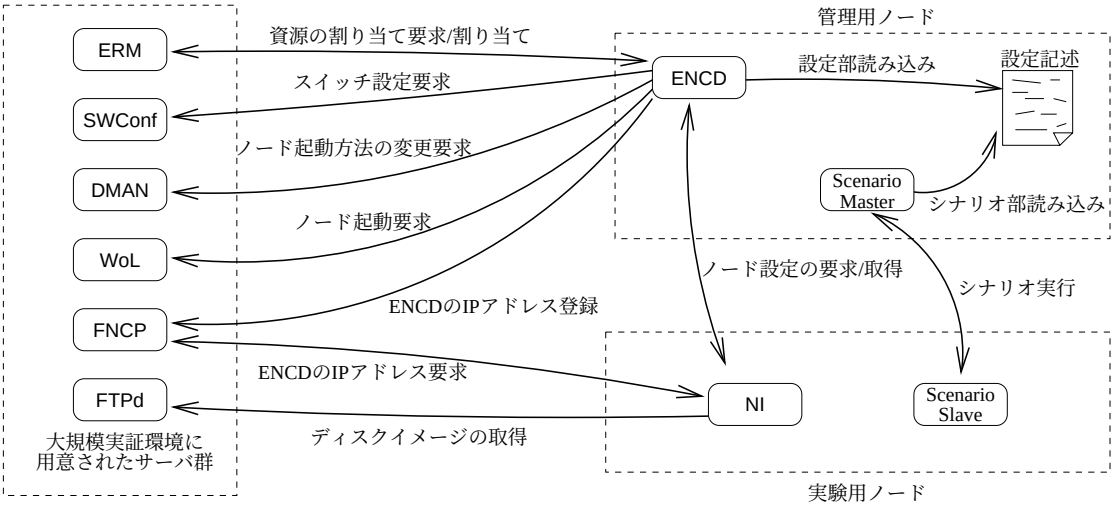

図 7.5: SpringOS の各機能の関連

ENCD が中核となり実験を実行する。設定ファイルを読み込んだ ENCD は設定記述にしたがい、その他の機能と連携して、実験を実行する。実験駆動単位の設定管理は ENCD が行うが、設定終了後 Scenario Master が実験用ノードで動作する Scenario Slave と協調して実験を行う。図中では ENCD と Scenario Master、NI と Scenario Slave が同時に描かれているが、実際に同時に起動することはない。

表 7.1: SpringOS の各機能と所属する管理レベル

管理レベル	モジュール
大規模実証環境	TFTPd
	DHCPd
	FTPd
	ERM
	FNCP
	WoL Agent
	DMAN
	SWConf
実験駆動単位	ENCD
	NI
	Scenario Master
	Scenario Slave

を介して、NI との通信を確立し、 必要な情報を NI に送る。NI は ENCD からの情報にしたがいディスクイメージをハードディスクに書き込む。

NI は ENCD からの情報にしたがい、ディスクイメージを取得し、ハードディスクに書き込みを行う。書き込みが終了すると、ENCD は DMAN を通じてノードが導入したディスクパーティションから起動するように設定を行い、NI はノードを再起動する。

なお、実験用ノードをディスクレスシステムとして起動する場合は、ハードディスクへのディスクイメージの書き込みなどは行わない。

4） **実験用スイッチの設定** ENCD は実験記述にしたがい、SWConf を通じて実験トポロジを構築する。

5） **ノードの初期設定および実験シナリオの実行** 実験用ノードでは必ず Scenario Slave が動作するよう設定されていることが必要である。

実験用ノード上で Scenario Slave が起動すると、Scenario Slave は Scenario

Master と通信を行い実験を開始する。実験の初期段階として IP アドレスなどの設定が必要となるが、これらの設定はシナリオの一部として実行される。他のノードとの協調が必要な場合のみ、Scenario Master を通じたメッセージ交換で他のノードと協調を行う。Scenario Slave は Scenario Master から MAC アドレスと IP アドレスの組を受取り、指定された MAC アドレスを持つネットワークインターフェースに、指定された IP アドレスを設定する。その他の必要な設定に関しても同様に Scenario Master との通信にしたがって実行される。

7.4 ディスクイメージの作成

ディスクイメージは、SpringOS で提供しているプログラム pickup を用いて作成できる。実験実行者は実験用ノードに必要な OS やアプリケーションソフトウェアの導入、設定などを行い、ハードディスクのパーティションをイメージとして pickup で取得し、ファイルサーバに保存する。pickup は ENCD として実装され、以下の順序でディスクイメージを保存する。

1. 対象ノードの起動方法を NI 用のディスクレスシステムに変更
2. WoL または SNMP を利用した実験用ノードの起動もしくは再起動
3. 起動した NI に対してディスクイメージ作成を指示
4. NI がディスクイメージを作成し指定されたファイルサーバに保存
5. 起動方法を変更前の状態に復帰
6. 再起動

7.5 設定記述例

本節では、*netperf* での帯域試験を行うという単純な実験を例にあげ、設定記述の例を説明する。本実験例では 3 台の実験用ノードからなる実験駆動単位が構成

される。一台は管理ノードであり、残りの 2 台はそれぞれ *netperf* のサーバプログラムとクライアントプログラムが動作するノードである。

実験記述はノード設定部とシナリオ部に分けられ、ノード設定部にはノードのネットワークインターフェースの種類や数などのハードウェアに関する条件と、起動方法や利用するディスクイメージなど導入するソフトウェアに関する情報が記述される。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 Scenario Master が実行するグローバルシナリオがある。

シナリオは以下の手順で実行されることとする。

1. server[0] が *netperf* のサーバプログラムである *netsever* を起動する。
2. Scenario Master が server[0] の IP アドレスを client[0] に通知する。
3. client[0] が *netperf* のクライアントプログラムである *netperf* を起動することで server[0] に対するトラフィックを発生させる。
4. *netperf* による性能試験が終了後、client[0] は Scenario Master に'cdone' というメッセージを発行する。
5. Scenario Master が'cdone' というメッセージを受信すると、server[0] に'quit' というメッセージを送信する。
6. 'quit' を受信した server[0] は *netserver* を終了する。

図 7.6 はこの時の設定ファイルの全容である。ただし、StarBED に用意されている FNCP や ERM などが動作する固定的なサーバに関する設定は別ファイルに記述されている。*netperf* のサーバとクライアント用の 2 つのノードクラスが 1 行目から 18 行目と 20 行目から 32 行目に定義されている。最初の定義がサーバ用で、それに続く定義がクライアント用であり、どちらのクラスに属するノードも 1 つのネットワークインターフェースを持つ。それぞれのノードのインスタンスは1 台ずつ生成されている。サーバノードとクライアントノードは直接接続されるため、一つの実験用ネットワークが 34 行目から 42 行目で定義され、ノードのインターフェースは"attach" 文により、このネットワーククラスのインスタンスに接続されている。

シナリオに関する定義は、“scenario” からはじまる 3 つのブロックで定義されている。最初の 2 つのシナリオは、ノードクラス中で定義されているが、これはノードシナリオであり、最後のシナリオ定義がグローバルシナリオである。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リストであり、グローバルシナリオはノード間の同期のために利用されるメッセージ制御のためのシナリオであるといえる。“recv” はメッセージ受信を待ち、指定された文字列に受け取ったメッセージを格納する。ノードは受け取ったメッセージにより挙動を変更することができ、これは 10 行目などの”msgswitch” で制御される。“wake” と”wakewait” はコマンド実行のために利用されるが、“wake” はコマンドをバックグラウンドプロセスとして起動し、“wakewait” はフォアグラウンドプロセスとして起動する。また、”sync” はグローバルシナリオ中でのみ利用され、指定されたノードからのメッセージを待つ。図 7.7 にメッセージ交換のタイムラインを示す。

```
1  nodeclass svclass {
2      partition 2
3      ostype "FreeBSD"
4      diskimage "ftp://anonymous:password@172.16.1.1/s_image.gz"
5      netif media fastethernet
6      scenario {
7          wake "/sim/netserver" "/sim/netserver"
8          loop {
9              recv x
10             msgswitch x {
11             "quit" {
12                 wakewait "/usr/bin/killall" "killall" "netserver"
13                 exit
14               } } } } }
15 nodeclass clclass {
16     partition 2
17     ostype "FreeBSD"
18     diskimage "ftp://anonymous:password@172.16.1.1/c_image.gz"
19     netif media fastethernet
20     scenario {
21         sleep 6
22         recv dst
23         sleep 24
24         wakewait "/sim/netperf" "/sim/netperf" "-H" dst
25         send "cdone"
26     } }
27 netclass ethclass {
28     media fastethernet
29     ipaddrrange "192.168.3.0/24"
30 }
31 nodeset client class clclass num 1
32 nodeset server class svclass num 1
33
34 netset ethnet class ethclass num 1
35
36 attach server.netif["lan0"] ethnet
37 attach client.netif["lan0"] ethnet
38
39 scenario {
40         send client[0] haddr(server[0].netif[0].ipaddr)
41         sync {
42                 msgmatch client[0] "cdone"
43         }
44         send server[0] "quit"
45     exit
46 }
```

図 7.6: 本実験例での設定ファイル

本実験例の設定ファイルの全容。これだけの設定記述を用意すれば、ノードへの OS 導入を含め、実験シナリオすべてが自動的に実行される。

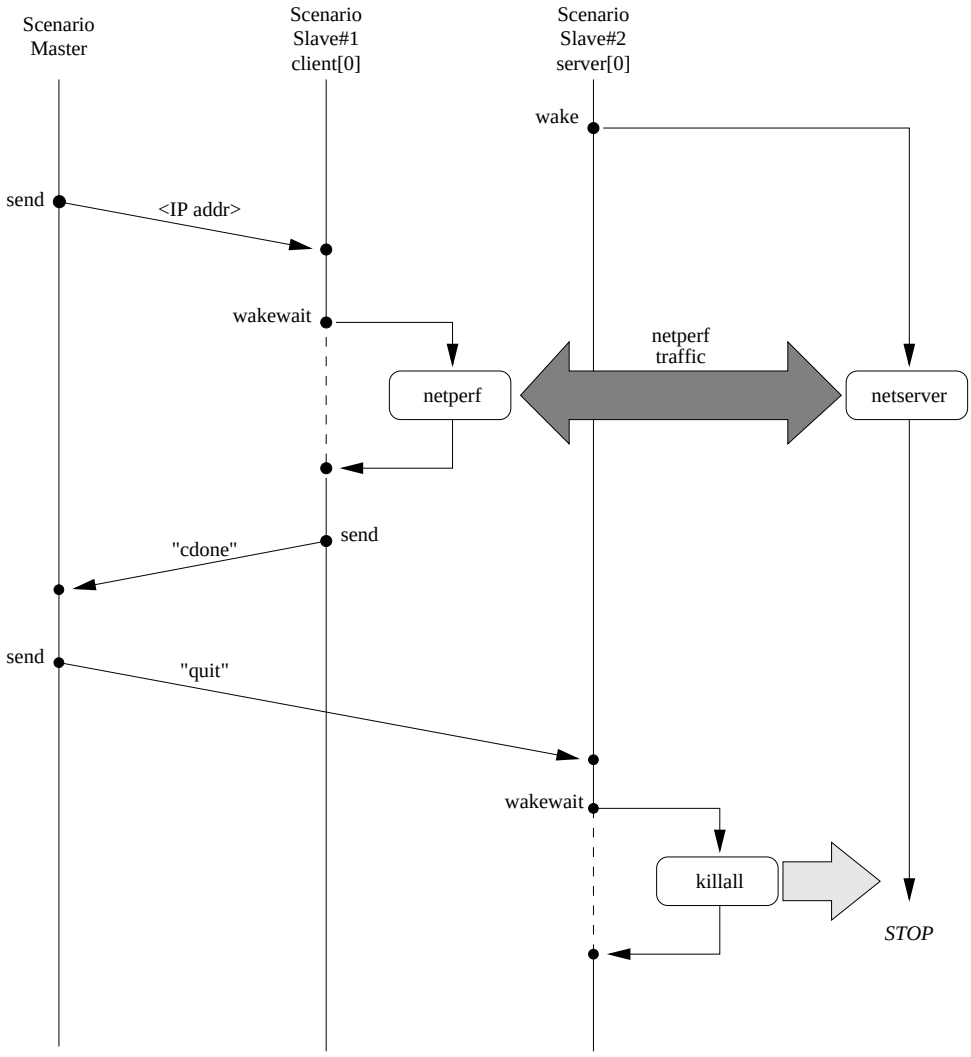

図 7.7: 実験のタイムライン

本実験でのタイムライン。一番左のラインが管理用ノード上の Scenario Master の動作を、中央のラインが *netperf* を実行する実験用ノード、右のラインが *netserver* を実行する実験用ノードの動作を表す。

7.6 StarBED での仮想機械の利用

前述のとおり実ノードを利用した大規模実証環境では、実環境との同一性は重要であるため、仮想機械を利用した場合、実環境との同一性は低下する。しかし、実環境との同一性が高い必要性がある部分を実ノードで実現し、それ以外の部分を仮想機械で実現することにより、実験トポロジを大規模化できる。本節では、SpringOS を仮想機械を取り扱えるよう拡張した SpringOS/VM の詳細について述べる。

実験ソフトウェアを利用している実験実行者にとっては、できるだけ仮想機械と実ノードの区別なく扱えることが望ましい。StarBED では、多数のノードによる実験トポロジを容易に構築するために、VLAN などの仮想ネットワーク構築手法を用いて、物理接続を変更することなく実験トポロジを構築する。実ノードを利用する場合は、実ノードが接続されているノードのネットワークインターフェースが収容されているスイッチの設定を行えばよいが、仮想機械を利用する場合は、仮想機械が動作している実ノードのネットワークインターフェースを検索し、そのネットワークインターフェースが接続されているスイッチの設定を行う必要がある。

またシナリオ実行に関しては、仮想機械を利用した場合、仮想機械上で実行される実験駆動のためのシナリオと、仮想機械を動作させるための実ノード用のシナリオが必要となる。しかし、実験実行者が混乱することを避けるため、仮想機械を動作させるためのシナリオは実験実行者からは隠蔽することが望ましい。

以上のように、仮想機械を利用した場合は、仮想機械を動作させるノードの制御が必要となり、実験資源の管理および実験手順が複雑化する。

7.6.1 SpringOS/VM の処理手順

仮想機械のノード設定の際の処理手順を図 7.8 に示す。実ノードを実験用ノードとして利用する場合は、実ノードにシナリオが送信されれば実験が開始される。しかし、仮想機械を利用する場合は、実ノードに送信されるシナリオは仮想機械の設定および起動のためのものである。したがって、仮想機械設定、起動用のシ

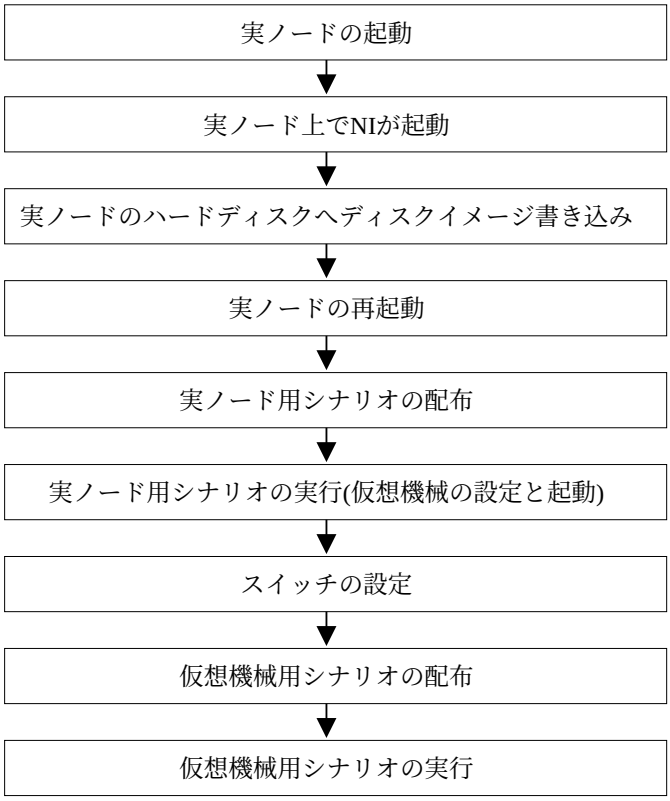

図 7.8: SpringOS/VM の処理手順

SpringOS/VM では SpringOS の手順に追加し、ホスト OS の制御が必要となるため、ノードの設定・制御も 2 重になる。

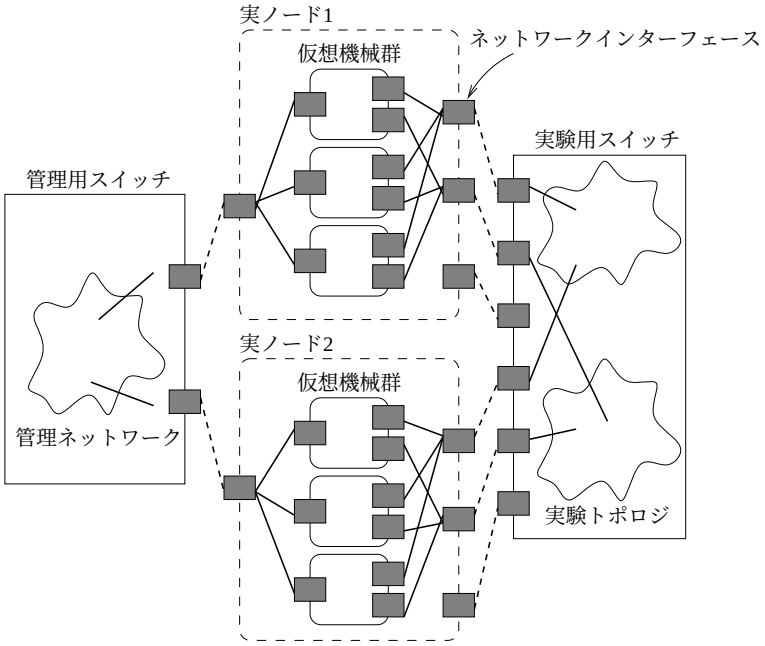

図 7.9: 仮想機械を用いた実験トポロジ

本実験では、2 台の実ノード上でそれぞれ 3 台、合計 6 台の仮想機械を動作させている。仮想機械のネットワークインターフェースは実ノードのネットワークインターフェースを通じて外部のネットワークに接続される。

ナリオを実ノードに送信し、仮想機械を起動し、仮想機械上で Scenario Slave が起動後に仮想機械用のシナリオを送信する。

7.6.2 仮想機械を利用した場合のネットワーク構成

前述のとおり SpringOS は大規模実証環境に管理用と実験用のネットワークが別々に用意されていることを想定している。同様に仮想機械も管理用および実験用のネットワークに接続することとする。仮想機械を利用した簡単な実験トポロジを図 7.9 に示す。図中の破線で示された長方形が実ノードであり、その中の実線の長方形が仮想機械を示す。実ノードと仮想機械にはそれぞれネットワークインターフェースが用意されている。ネットワークインターフェースの物理的な接続を破線で、仮想的な接続を実線で示した。VMware の *vmnet-bridge* を用いれば、

仮想機械のネットワークインターフェースを、実ノードのネットワークインターフェースが接続されているネットワークに接続されているようにみせることができる。この機能を用い、仮想機械の各ネットワークインターフェースを実ノードが接続されているネットワークに接続する。

7.6.3 資源管理

ERM のデータベースには実ノードの情報が 1 台ずつ記述されている。仮想機械の情報については専用の記述方法を用意し、同じデータベースで管理することとした。仮想機械を扱うため、これらの情報に加え実ノードで起動し得る仮想機械の識別子を付加した。この識別子は、実ノードの設計上動作させることができる最大台数分を記述した。

SpringOS は IP アドレスの設定などのためにネットワークインターフェースの MAC アドレスを必要とする。VMware は、MAC アドレスは指定された範囲内であれば自由に指定でき、VMware の設定ファイルに記述することで、その MAC アドレスを利用できる。この機能を利用し、まえもって MAC アドレスを決定し、ERM のデータベースに追加した。

管理ネットワークに接続されている実ノードのネットワークインターフェースには、DHCPd に MAC アドレスと IP アドレスが設定されており、起動時にその IP アドレスが自動設定される。仮想機械の管理ネットワークに接続されているネットワークインターフェースにも同様に常に同じ IP アドレスを設定するため、DHCPd に仮想機械で用いる MAC アドレスと IP アドレスの設定を行った。

7.6.4 ノード割り当て

動作させるアプリケーションソフトウェアの負荷を考慮することにより、1 台の実ノード上で動作させる仮想機械の台数を変更するため、実験実行者が実験の設定記述に、実ノード上で起動する仮想機械の数を指定することとした。ノード割り当て時には、実験実行者が指定した多重度と必要な仮想機械の総数から、必要な実ノード数を計算し、その値に予備ノード数を追加し ERM に実ノードの割り当てを要求する。ERM はこの情報にもとづき実ノードを検索し、実験実行者が指定

した多重度で仮想機械を起動できる実ノードが必要数存在すれば、その実ノードを割り当てる。

7.6.5 実ノード上での仮想機械のための設定

ERM により割り当てられた実ノードへのソフトウェアの導入の終了後、実ノード上では仮想機械のための設定を行うためのシナリオが実行される。このシナリオの主な役割は、*vmnet-bridge* の起動、VMware の設定ファイルの生成、VMware の起動である。

***vmnet-bridge* の起動** 設定記述にしたがい、仮想機械の各ネットワークインターフェースと、実ノードのネットワークインターフェースの対応を調査し、その結果をもとに *vmnet-bridge* を起動する。

VMware 設定ファイル生成 VMware の設定ファイルはテキスト形式であり、簡単に編集できる。すべての仮想機械に共通した設定部分をまえもって用意しておき、このファイルに各仮想機械独自の設定を追加する。各仮想機械に独立な設定には、各ネットワークインターフェースの MAC アドレスおよび接続形式、およびディスクイメージのファイルなどが指定される。

実験トポロジの構築 実ノードのネットワークインターフェースが収容されているスイッチの VLAN などを設定し、実験トポロジを生成する。

VMware の実行 設定ファイルで指定された数の VMware を起動する。

すべてのネットワークインターフェースは *vmnet-bridge* の機能を用いて、実ノードが接続されているネットワークに接続する。また、設定ファイルにしたがい、VMware のディスクイメージをダウンロードする。

7.6.6 シナリオの実行

以上の手順により実ノード上で仮想機械が起動し、実験トポロジが構築される。この仮想機械上で起動する kuroyuri slave と kuroyuri master が通信を行うことで、SpringOS と同様にシナリオを実行する。

7.7 考 察

SpringOS の実装により StarBED での実験実行が容易に行えるようになっただけでなく、実験シナリオの自動実行により実験の精度も向上できたといえる。また SpringOS/VM により StarBED に存在する実ノードの数以上の実験駆動単位の構築も可能となった。次章でこれらの点についての評価を行う。

第8章

StarBEDとSpringOSの評価

本章では、StarBED と SpringOS および SpringOS/VM の評価を行う。実験内容としては、*netperf* を利用した帯域評価により実験実行に必要な時間を計測し、それぞれについての検討を行った。

8.1 実験概要

netperf はサーバプログラムである *netserver* とクライアントプログラム *netperf* から構成され、*netserver* をまえもって起動しておき、*netperf* から *netserver* に対してトラフィックを生成することで、その間の帯域を測定する。評価実験用トポロジは図 8.1 のように構成した。評価実験では図中の *netserver* と *netperf* のペア数 n を変更し、実験に必要な時間を計測した。対応するペアは一つの VLAN に属し、複数のペアが存在する場合は、それぞれ別の VLAN に属するように設定を行った。

実験には StarBED のクライアント装置 A に所属するノード群を利用した。クライアント装置 A のノード情報を表 8.1 に再掲する。

実験管理用ノードには、StarBED のクライアント装置 A のノードを 1 台確保し、標準状態で導入されている FreeBSD 5.4 に SpringOS を導入し利用した。実験用ノードには FreeBSD 5.5 を新規にインストールし、*netperf* は FreeBSD の ports[40] を利用してインストールした。なお実験用ノードの数が増えても管理用ノードは常に一台である。

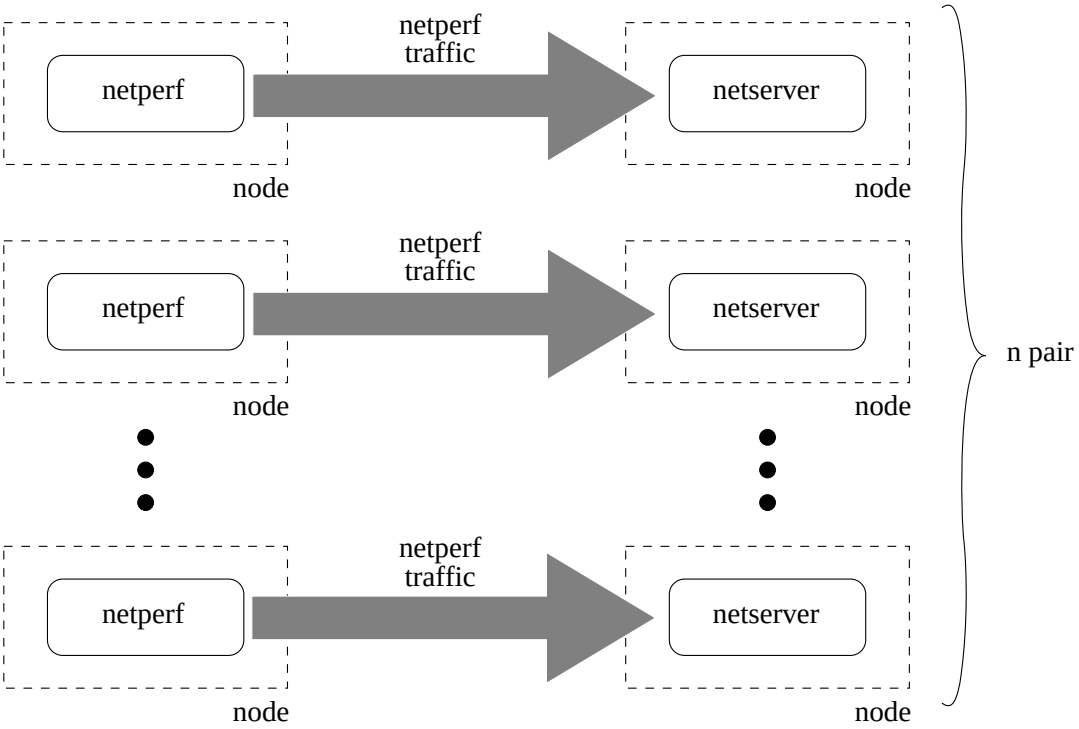

図 8.1: 評価実験用トポロジ

netperf が動作するノードと *netserver* が動作するノードのペアを増加させ実験を行う。この際にペアの 2 台だけが接続されている VLAN を用意する。したがって、n ペア存在する実験であれば n 個の VLAN を設定することになる。

表 8.1: クライアント装置 A のノード情報

モデル	NEC Express5800 110Rc-1
チップセット	ServerWorks LE
CPU	Pentium 3 1GHz
メモリ	512Mbyte
HDD	IDE 30GB
ネットワーク インターフェース	（on-broad）2 FastEthernet （extend） 1 GigabitEthernet

8.2 実験の進行

7.3.2 節で述べたとおり、SpringOS を利用した実験の進行は以下の順序で行われる。

1. 実験の設定記述の読み込み
2. 実験資源の割り当て
3. ノードへのディスクイメージの導入
4. 実験用スイッチの設定
5. 実験用ノードの初期設定および実験シナリオの実行

しかし、これ以前に、設定記述の作成および、ディスクイメージの作成が必要である。また、この後、資源の返却およびノードの初期化とスイッチの初期化が必要である。資源の返却には、リソースマネージャに対して資源の返却表明を行い、さらに、ノードの初期化およびスイッチの状態を初期化する。ノードは初期状態のディスクイメージを導入する方法と、初期状態に対して加えた変更を削除する方法がある。スイッチの初期化は実験に利用した VLAN 情報の削除が必要である。

8.3 ディスクイメージの作成

7.4 節で述べたとおり、ディスクイメージは、SpringOS で提供しているプログラム pickup を用いて作成できる。実験実行者は、まず実験用ノードに必要な OS やアプリケーションソフトウェアの導入およびそれらの設定などを行う。その後、設定を行ったハードディスクのパーティションをイメージとして pickup で取得し、ファイルサーバに保存する。

ディスクイメージの作成は管理ネットワークを利用して実行される。今回利用したクライアント装置 A が接続されている管理ネットワークの物理的構成を図 8.2 に示す。StarBED のノードは 1 ラックに 24 台設置され、各ラックにおかれたスイッチ D に収容されている。スイッチ D は上流のスイッチ F に接続され、ノードの管理

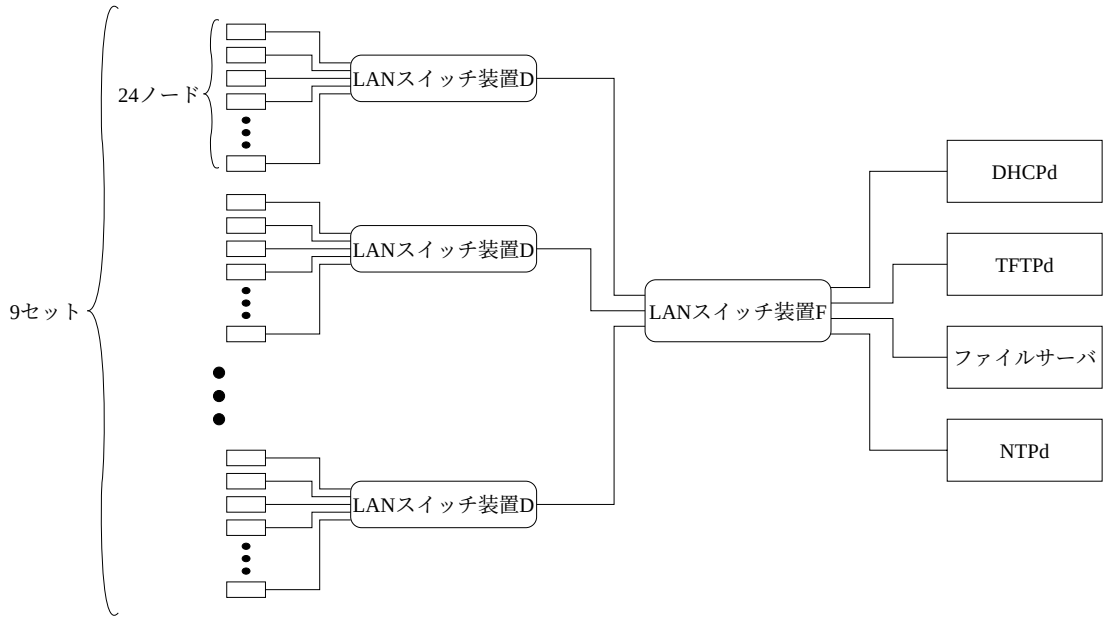

図 8.2: クライアント装置 A の管理ネットワーク構成

図中左の四角が実験用ノードを表し、24 台のノードが LAN スイッチ装置 D に接続され、すべての LAN スイッチ装置 D は LAN スイッチ装置 F に接続されている。各種サーバは LAN スイッチ装置 F に接続されている。図中すべてのリンクは FastEthernet で構成されている。

用サーバはこのスイッチ F に接続されている。図中のリンクはすべて FastEthernet であるため、帯域は 100Mbps である。

管理用ノードで pickup を動作させディスクイメージを作成した。pickup は TFTPd が動作しているノード上で動作している DMAN と通信し、対象のノードを NI が動作するディスクレスシステムで起動するよう変更し、Wake on LAN によりノードを起動するか、SNMP で再起動する。NI が起動すると、TFTPd と同じノードで動作している FNCP と通信し、ENCD である pickup が動作している管理ノードの IP アドレスを確認し、pickup と通信を行う。NI はこの通信により対象のパーティション番号やファイルサーバの情報を取得し、実際にディスクイメージを作成・保存する。

8.4 実験のシナリオ

本実験のタイムラインを図 8.3 に示す。図中では実験用ノード 2 台と管理用ノード 1 台の合計 3 台分のタイムラインを示した。実験部分は *netserver* が動作する server[0] と *netperf* を実行する client[0] で閉じている。台数を増やす場合は server と client のペアを増やして行った。

実験用ノードへの OS、アプリケーションソフトウェアの導入後、シナリオとして実行される処理を列挙する。ただし、SpringOS では、IP アドレスの設定などはシナリオの一部として実現しており、各実験用ノードで動作している kuroyuri slave（以下 slave）は、管理用ノードで動作する kuroyuri master（以下 master）と通信することにより、設定を行うべきネットワークインターフェースの MAC アドレスおよび IP アドレスを取得し、対象のネットワークインターフェースを MAC アドレスから判断し、IP アドレスを設定する。また、コマンドの実行には、すでに述べたとおり”wake” および”wakewait” があるが、本実験では”call” および”callw” を利用している。“wake” と”call”、”wakewait” と”callw” は対応しており、引数が違うことを除けば同じ動作をする。

また、以下 *netperf* を実行する実験用ノードを client ノード、*netserver* を実行する実験用ノードを server ノードと呼ぶ。

1. master は、各実験用ノードへシナリオを配布すると、client からの”csetupdone” と server からの”ssetupdone” メッセージを待つ。

2. 実験用ノードが起動し slave も起動する。

3. ログファイルに保存される時刻などを同期するため、*ntpdate* コマンドを実行し管理側ネットワークに接続された NTPd との時刻同期を行う。

4. *ifconfig* を実行し実験ネットワークに接続されたネットワークインターフェースの IP アドレスを設定する。

5. client ノードの slave は *netperf* を実行できる状況であることを master へ通知するため”csetupdone” メッセージを送信する。

6. server ノードの slave は、ネットワークインターフェースの設定後、*netserver* をバックグラウンドで実行する。

7. server ノードは *netperf* コマンドによる帯域測定の準備が整ったことを master へ通知するため、”ssetupdone” メッセージを送信する。

8. この時点で master が待っていたメッセージがそろう。

9. メッセージがそろったことをトリガとして、master は client ノード群へ、対応する server ノードの IP アドレスを送信する。

10. master は IP アドレスを送信した後、すべての client ノードからの”cdone” メッセージを受け取るまで待機する。

11. server ノードの IP アドレスを受信した client ノードの slave は、server ノードへの到達性を *ping* により確認する。

12. その後 *netperf* を実行し帯域を測定する。これらの実行ログは標準出力に出力されるため、リダイレクト機能を利用してファイルに保存する。

13. *netperf* 終了後、client ノードの slave は、FTP で管理用ノードに保存したログを送信する。

14. client ノードの slave は、実験が終了したことを master に知らせるため”cdone” メッセージを master へ送信する。

15. “cdone” メッセージ送信した client ノードの slave は *shutdown* コマンドを実行し、ノードの電源を切る。

16. master は、すべての client ノードから “cdone” メッセージを受け取ると、すべての server ノードへ”quit” メッセージを送信する。

17. “quit” メッセージを受け取った server ノードは *pkill* コマンドにより *netserver* を終了する。

18. server ノードはメッセージを送信後 *shutdown* コマンドの実行によりノードの電源を切る。

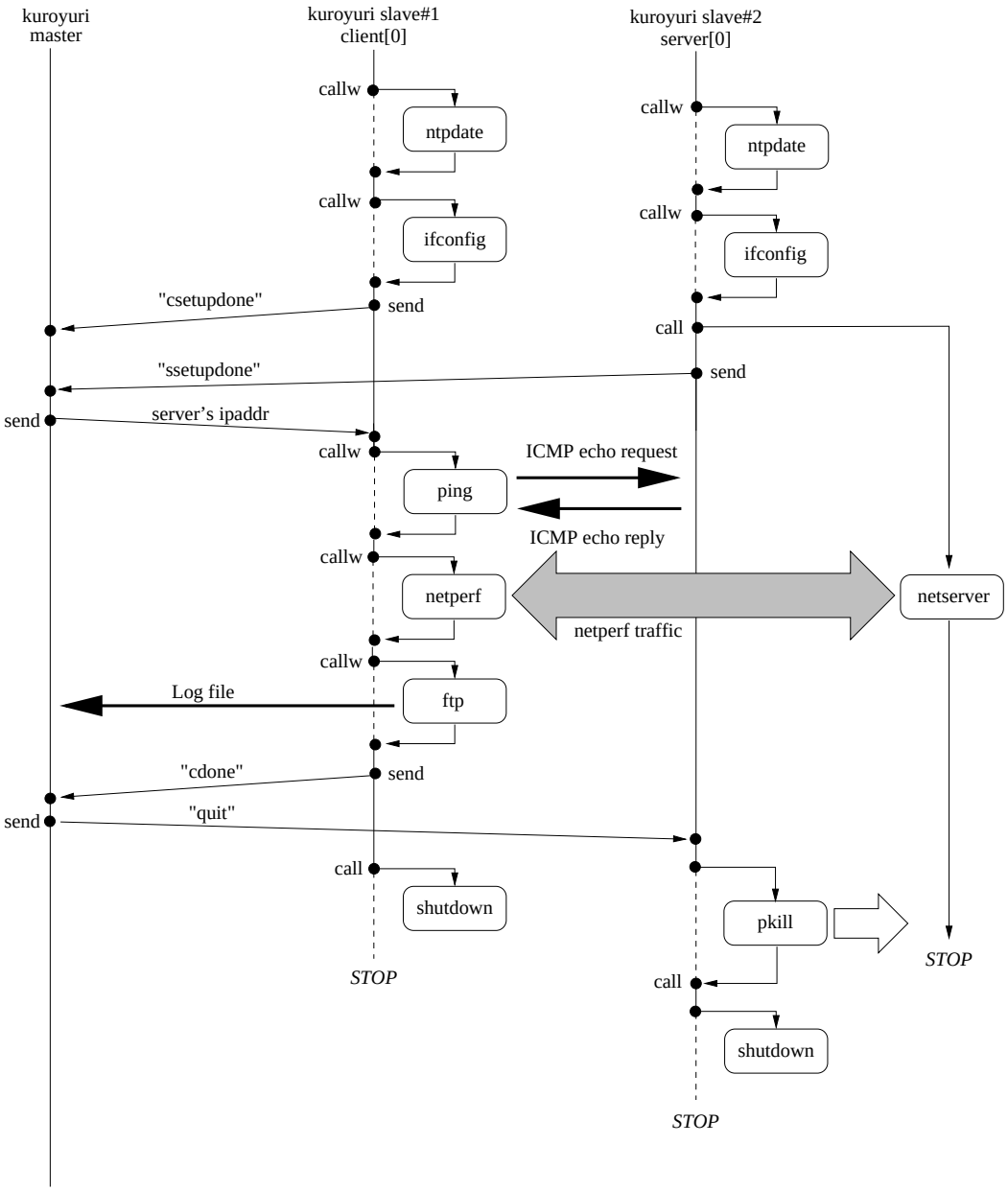

図 8.3: 評価実験のタイムライン

実験用ノードは *netperf* を実行する前に *ifconfig* によるネットワークインターフェースの設定を行い、ログ情報の同期のため *ntpdate* を実行する。この後 *netperf* を実行する client[0] は準備ができたということをメッセージ交換により kuroyuri master に通知する。一方、*netserver* が動作する server[0] では、*ntpdate* を実行した後、*netserver* を起動し、準備ができたことをシナリオマスタに通知する。その後、それぞれがメッセージ交換により同期を行いながら実験を進行する。

8.5 評価実験で利用した設定記述

8.4 節で述べたシナリオを実行するための設定記述を図 8.4〜8.10 に示す。実際にはこれらの設定は一つのファイルに上記順序で記述される。また、StarBED 施設として用意されているサーバなどの情報は変更される事が少ないため、別ファイルに用意することができる。今回も別ファイルとして用意し利用した。このファイルを図 8.11 に示す。

図 8.4 は、SpringOS の実行に必要とする変数および、本実験で利用する変数の定義部である。“user”、”project” はリソースマネージャの認証に用いられ、“encd ipaddr” は master を実行する管理用ノードの IP アドレスを指定する。多数のノードを利用した実験では、障害が発生しているノードが割り当てられるおそれがあるため、“sparenodemin” および”sparenoderatio” を指定することで、予備のノードの割り当てを受けることができる。“sparenodemin” で指定された台数もしくは、“sparenoderatio” で指定されたノード割合の大きな値の分だけ予備ノードが確保される。“sparenoderatio” が 100 であるが、これは、利用するノードは必要とされるノードの 100%という意味であり必要ノードと同数のノードを予備として確保するわけではない。ただし、本設定では予備ノードは割り当てない設定としている。“pairnum” 以降は実験で利用するグローバル変数の指定である。“export” することでノードシナリオなどでも利用できるようになる。“assure” で指定した変数は master 実行時のコマンドラインで指定することができる。

図 8.5 は、本実験で利用する *netserver* を実行するためのクラス定義である。“diskimage” で利用するディスクイメージを指定し、ハードディスクのパーティション 2 に保存し、そこから起動する設定となっている。ノードシナリオ部は基本的に実行されるコマンドが列挙されており、“netiffit” で MAC アドレスと IP アドレスの対応を調べ、*ifconfig* で指定された IP アドレスを指定する。図 8.3 では示さなかったが、実際に指定された IP アドレスなどは実験後の解析に必要になることがあるため、インターフェース情報などを表示させセログファイルに保存している。

図 8.6 は、*netperf* を実行するためのクラス定義である。物理的な要件と導入されるディスクイメージは *netserver* 用のものと同一であるが、実行されるシナリオのみ異なっている。*netserver* 用のクラスではノードの基本的な設定後 *netserver* を

```
user "starbed" "info@starbed.org"
project "starproj"
rbhfile "master.his"

encd ipaddr "172.16.0.208"
ipaddrrange "192.168.0.0/16"

sparenodemin 0
sparenoderatio 100

assure pairnum = 3
export pairnum
wolinterval 2000

nodenum = pairnum * 2
export nodenum

workdir="/var/tmp/"
bindir="/root/exp/"
pathifscan=bindir+"/ifscan"
pathnetperf="/usr/local/netperf/netperf"
pathnetserver="/usr/local/netperf/netserver"
epoch=time()
export workdir
export bindir
export pathifscan
export pathnetperf
export pathnetserver
export epoch
```

図 8.4: 評価実験用の変数定義部

SpringOS の実行に必要となる変数と、実験に利用する変数を定義している。

```
nodeclass svclass {
    method "HDD"
    disktype "IDE"
    partition 2
    ostype "FreeBSD"
    diskimage "ftp://usr:pass@172.16.220.222/sintcla206-rad0s2-200701161207.gz"
    netif media gigabitethernet
    scenario {
        callw "/usr/sbin/ntpdate" "172.16.254.165"
        chdir workdir
        callw "/sbin/ifconfig" "-a"

        netiffit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr

        callw "/sbin/ifconfig" "-a"
        callw "/bin/ls" "-l" pathnetserver

        callw "/usr/bin/pkill" "-9" "netserver"
        call pathnetserver

        send "ssetupdone"

        loop {
            recv x
            msgswitch x {
                "quit" {
                    callw "/usr/bin/pkill" "-9" "netserver"
                    call "/sbin/shutdown" "-p" "now"
                }
            }
        }
    }
}
```

図 8.5: 評価実験用設定記述 server（netserver）用クラス定義

SpringOS でノードおよびネットワークを定義するには、クラスの定義を行う。この部分は、*netserver* を実行するためのクラス定義。ノードに要求される条件と、導入されるべきディスクイメージおよび、ノードの設定とシナリオ実行のためのノードシナリオが記述されている。

起動するだけであったが、本クラスでは、*netserver* が動作するノードの IP アドレスを master から受取り、その IP アドレスへの疎通確認のための *ping* や、出力されたログファイルを管理用ノードにアップロードを行う。

図 8.7 はネットワーククラスの定義である。SpringOS ではネットワーククラスの定義としては基本的にメディアを指定する。利用する IP アドレスの範囲も指定できるが、これはインスタンスに指定した方がわかりやすいことが多いためあまり利用されない。

図 8.8 では、これまでに説明したクラスのインスタンスを生成している。“svclass” のインスタンス”server” と”clclass” のインスタンス”client”、“ethclass” のインスタンス”ethnet” をそれぞれ”pairnum” 個生成している。“pairnum” は図 8.4 で示したとおり”assure” で指定されているため、コマンドラインから指定できる。今回のようにノード台数を変更して繰り返し実験を行う際には有用である。

図 8.9 は、実験トポロジの定義部である。まず、それぞれのネットワークが利用する IP アドレスの範囲を “ethnet[i].ipaddrrange” に設定している。これにより、対象のネットワークに所属する各実験用ノードのネットワークインターフェースに指定された範囲から IP アドレスが自動的割り当てられる。また、各ノードのネットワークインターフェースを、対象のネットワークに”attach” することにより、所属するネットワークを指定している。どちらの設定とも”for” 文を利用することにより効率よく設定できる。

図 8.10 は、ノードの協調用に利用されるグローバルシナリオである。“sync” を利用すると、ブロック内に書かれたすべての条件がそろうまでグローバルシナリオの進行を停止できる。ここでは、すべての server ノードからの”ssetupdone” と、すべての client ノードからの”csetupdone” を受け取るまでシナリオを停止する。条件がそろうと、すべての client ノード群に、そのノードに対応する server の IP アドレスを送信する。この後、すべての client ノードから”cdone” メッセージが届くまでシナリオ進行を停止し、条件がそろえば、”quit” を server ノード群に送信し、シナリオ実行を終了する。

図 8.11 は StarBED 施設で提供している管理用サーバなどに関する設定記述である。基本的にあまり変更されないため、別ファイルに用意することが多い。今回、リソースマネージャは master が動作する管理用ノードで動作させた。その他には

```
nodeclass clclass {
    method "HDD"
    disktype "IDE"
    partition 2
    ostype "FreeBSD"
    diskimage "ftp://usr:pass@172.16.220.222/sintcla206-rad0s2-200701161207.gz"
    netif media gigabitethernet
    scenario {
        repfile=self.rname+"-n"+tostring(nodenum)+"-"+tostring(epoch)+".out"
        chdir workdir

        callw "/usr/sbin/ntpdate" "172.16.254.165"

        callw "/sbin/ifconfig" "-a"

        netiffit pathifscan
        callw "/sbin/ifconfig" self.netif[0].rname self.netif[0].ipaddr

        callw "/sbin/ifconfig" "-a"
        callw "/bin/ls" "-l" pathnetperf

        send "csetupdone"

        recv dst

        callw "/bin/ping" "-c" "10" dst
        callw "/bin/date" "+#%Y%m%d_%H%M%S" > repfile
        callw "/bin/date" "-u" "+#%Y%m%d_%H%M%S UTC" >> repfile
        callw pathnetperf "-H" dst >> repfile

        callw "/bin/cat" repfile

        netput repfile "ftp://install:install@172.16.0.208/"
        send "cdone"

        call "/sbin/shutdown" "-p" "now"
    }
}
```

図 8.6: 評価実験用設定記述 client（netperf）用クラス定義

netperf を実行するためのクラス定義。*netserver* 用のクラス定義と同様にログのため、日付の出力なども行っている。“netput” を利用することにより FTP でログファイルを管理用ノードにアップロードする。

```
netclass ethclass {
    media gigabitethernet
}
```

図 8.7: 評価実験用設定記述 ネットワーククラス定義

ネットワークのクラス定義。基本的にはネットワークのメディアの指定のみが必要である。

```
nodeset client class clclass num pairnum
nodeset server class svclass num pairnum

netset ethnet class ethclass num pairnum
```

図 8.8: 評価実験用設定記述 インスタンス生成部

すでに定義したクラスを利用して、ノード、ネットワークともインスタンスを生成している。

```
for(i=0;i<pairnum;i++){
    ethnet[i].ipaddrrange = "192.168."+tostring(i)+".0/24"
}

for(i=0;i<pairnum;i++){
    attach server[i].netif[0] ethnet[i]
    attach client[i].netif[0] ethnet[i]
}
```

図 8.9: 評価実験用設定記述 トポロジ設定部

実験トポロジの設定部。ネットワークインスタンスが利用する IP アドレスの範囲と、それぞれのインスタンスに所属する各ノードのネットワークインターフェースを指定している。

```
scenario {
    sync {
        multimsgmatch server "ssetupdone"
        multimsgmatch client "csetupdone"
    }

    for(i=0;i<pairnum;i++) {
        send client[i] haddr(server[i].netif[0].ipaddr)
    }

    sync {
        multimsgmatch client "cdone"
    }

    multisend server "quit"
}
```

図 8.10: 評価実験用設定記述 グローバルシナリオ

ノードの協調のために master が動作する管理ノードで実行されるグローバルシナリオ。基本的にはメッセージのやり取りを制御する。

FNCP や DMAN、WoL Agent の IP アドレスや、各スイッチのファームウェアの種類や IP アドレスなどが記述される他、ディスクイメージ導入時に利用する、NI が動作する専用のディスクイメージやカーネルの情報が記述されている。

```
rmanager ipaddr "127.0.0.1" port "1234"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.0.0/24"
wolagent ipaddr "172.16.1.101" port "5959" ipaddrrange "172.16.1.0/24"
fncp ipaddr "172.16.3.101"
tftpdman ipaddr "172.16.3.101"

nidiskimage "FreeBSD/ni04.fs"
nikernel "recover_system/kernel_recover"
pxeloader "recover_system/pxeboot-nohang"
setuptimeout total 86400 warm 5

swtype name "silaswa001" type "IOS"
swtype name "silaswb001" type "IOS"
swtype name "silaswb002" type "IOS"
swtype name "silaswb003" type "Ironware"
swtype name "silaswc001" type "Ironware"
swtype name "silaswc002" type "Ironware"
swtype name "silaswc003" type "Ironware"
swtype name "silaswc004" type "Ironware"
```

図 8.11: 評価実験用設定記述 StarBED 基本設定

StarBED の施設としてあまり変更が行われない設定は、このような別ファイルとして用意される。基本的には各サーバの IP アドレスやスイッチの種類、NI が起動するディスクイメージやカーネルの情報が記述されている。

8.6 実験結果

以下の項目について、計測された結果を示す。

1. ディスクイメージの作成
    - OS とアプリケーションのインストール
    - ディスクイメージの pickup による保存
2. 実験の実行
    - 実験用ノードへのディスクイメージの配布
    - 実験用スイッチの VLAN 設定
    - 実験用ノードへのシナリオ配布
    - シナリオの実行
    - VLAN 設定の削除
    - 全体

8.6.1 ディスクイメージの作成

ディスクイメージの作成のため、クライアント装置 A のノード 1 台に手動で FreeBSD 5.5 をインストールした。インストールの際の条件を表 8.2 に示し、この時のインストール所要時間を表 8.3 に示す。インストールした FreeBSD への ports からの *netperf* のインストールおよび、SpringOS の必要なモジュールとして kuroyuri slave と、ノードの再起動や終了をおこなうための SNMP のサーバプログラム（snmpmine）のソースコードのダウンロードおよびインストールには合計で 1 分ほどがかかった。

その後管理ノードで pickup を動作させ、FreeBSD をインストールしたパーティション 2 を保存した。表 8.4 に pickup の所要時間を示す。

表 8.2: FreeBSD インストール条件

インストールモード	Express
パーティション 番号	2
パーティション サイズ	4Gbyte
パーティション 設定	インストーラによる自動設定
package	すべて（ports 含む）
利用メディア	CD-ROM での起動 + ftp （ftp3.jp.freebsd.org）

表 8.3: FreeBSD のインストール所要時間

試行	所要時間（秒）
1	1158
2	1181
平均	1170

表 8.4: ディスクイメージ作成の所要時間

試行	所要時間（秒）
1	1281
2	1288
3	1282
4	1286
平均	1284

表 8.5: 2 台（1 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	459.6	21.5	18.0	11.0	524.7
2	426.7	21.7	18.0	11.0	492.0
3	440.8	21.5	17.9	11.2	506.9
平均	442.4	21.6	18.0	11.1	507.9

8.6.2 実験の実行

前節で解説した設定記述を利用して実験を行った。ノード数を 2、6、10、50、100、150、200 と変化させ、それぞれの台数での実験の所要時間を測定した。それぞれの計測値を表 8.5 〜 8.11 に示す。また、それぞれの台数での平均値を表 8.12 にまとめ、それぞれの段階でのノード数と所要時間の関係を図 8.12 〜 8.16 に、合計時間の関係を図 8.17 に示す。また図 8.17 中には合計時間だけでなく、ディスクコピーの時間に関してもプロットした。比例関係が認められるグラフには *gnuplot*[41] の fit 機能による最小自乗法を利用した近似直線を示した。合計は表に示した値の総和ではなく、master による設定記述の読み込みと解析や、slave へのノードシナリオの配布、ノードの予約解放にかかった時間も含めた master の実行時間を示す。

なお、今回の実験ではノードの初期化は行っていない。ただし、ハードディスクに初期状態のディスクイメージを書き込めばいいため、ディスクコピーと同じだけの時間がかかる。

表 8.6: 6 台（3 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	912.4	26.2	21.2	32.8	1007.3
2	894.9	26.6	21.1	34.2	992.4
3	882.1	26.2	21.2	34.8	980.9
平均	896.5	26.3	21.2	33.9	993.5

表 8.7: 10 台（5 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	1352.0	29.4	21.4	55.8	1473.4
2	1355.0	30.6	21.3	57.4	1479.8
3	1349.5	30.6	21.5	57.0	1474.0
平均	1352.2	30.2	21.4	56.7	1475.7

表 8.8: 50 台（25 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	5783.9	70.0	19.3	282.5	6170.7
2	5872.5	70.0	19.3	283.2	6260.7
3	5874.4	70.0	19.3	282.2	6261.1
平均	5843.6	70.0	19.3	282.6	6230.8

表 8.9: 100 台（50 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	11509.1	130.0	20.0	570.3	12245.3
2	11511.1	120.9	20.0	568.5	12235.9
3	11348.6	121.2	20.0	568.2	12074.2
平均	11456.2	124.0	20.0	569.0	12185.1

表 8.10: 150 台（75 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	17091.5	184.7	21.0	858.4	18172.0
2	17067.6	171.4	23.5	858.8	18136.1
3	17080.0	171.1	21.1	856.7	18144.9
平均	17079.7	175.7	21.9	858.0	18151.0

表 8.11: 200 台（100 ペア）での実験実行の所要時間（秒）

試行	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
1	22792.0	222.1	19.9	1144.2	24194.1
2	22828.0	221.8	21.2	1150.6	24238.0
3	22800.7	222.0	19.9	1149.1	24207.6
平均	22806.9	222.0	20.3	1148.0	24213.2

表 8.12: それぞれの台数での所要時間の平均値（秒）

ノード数	ディスクコピー	VLAN設定	シナリオ実行	VLAN削除	合計
2	442.4	21.6	18.0	11.1	507.9
6	896.5	26.3	21.2	33.9	993.5
10	1352.2	30.2	21.4	56.7	1475.7
50	5843.6	70.0	19.3	282.6	6230.8
100	11456.2	124.0	20.0	569.0	12185.1
150	17079.7	175.7	21.9	858.0	18151.0
200	22806.9	222.0	20.3	1148.0	24213.2

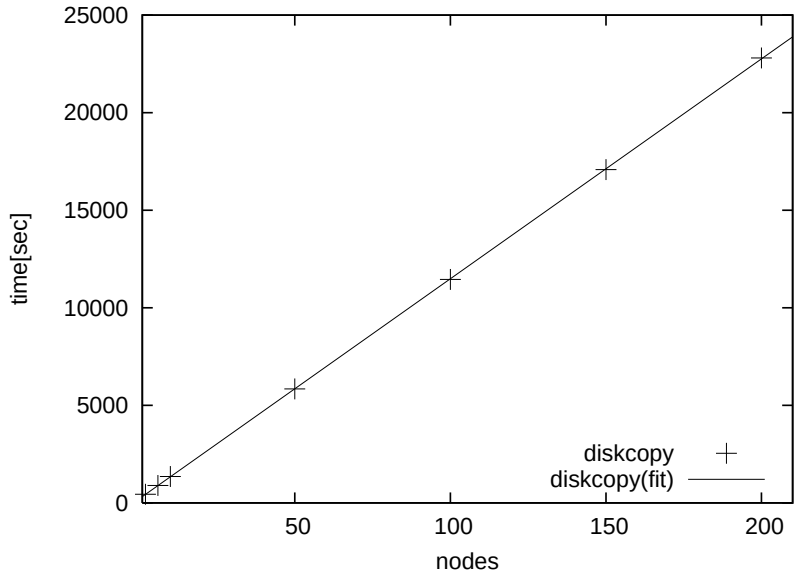

図 8.12: ノード数とディスクコピーの所要時間の関係

2 台から 200 台までのディスクコピーの所要時間は、最小自乗法で求められた直線にほぼ合致している。

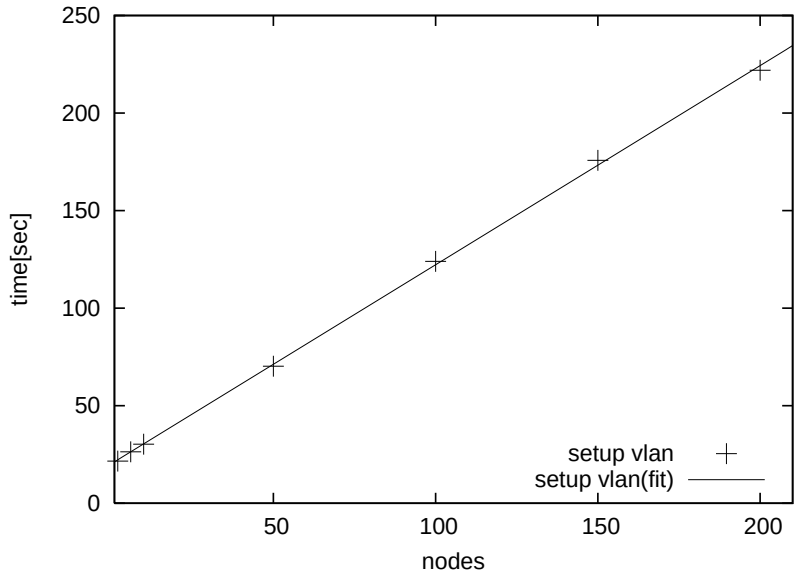

図 8.13: ノード数と VLAN 設定の所要時間の関係

2 個から 200 個までの VLAN 設定の所要時間は、最小自乗法で求められた直線にほぼ合致している。

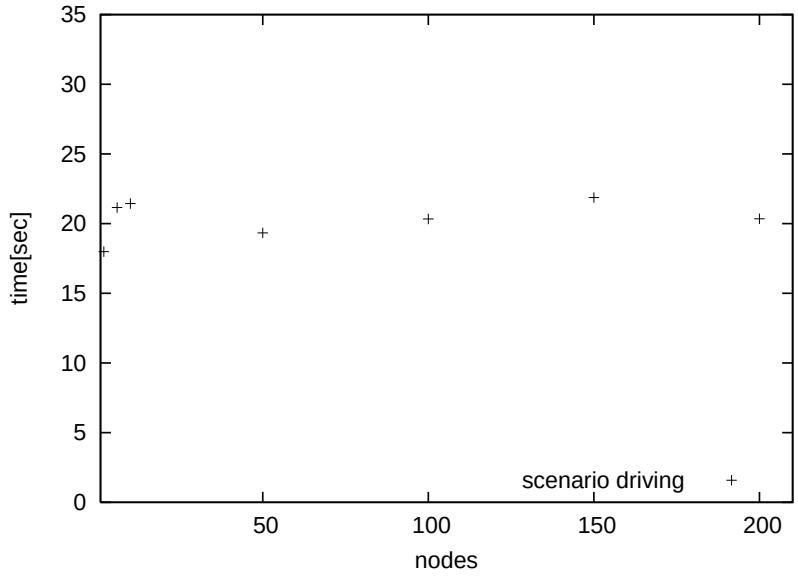

図 8.14: ノード数とシナリオ実行の所要時間の関係

シナリオの実行時間はノード数に比例しない。

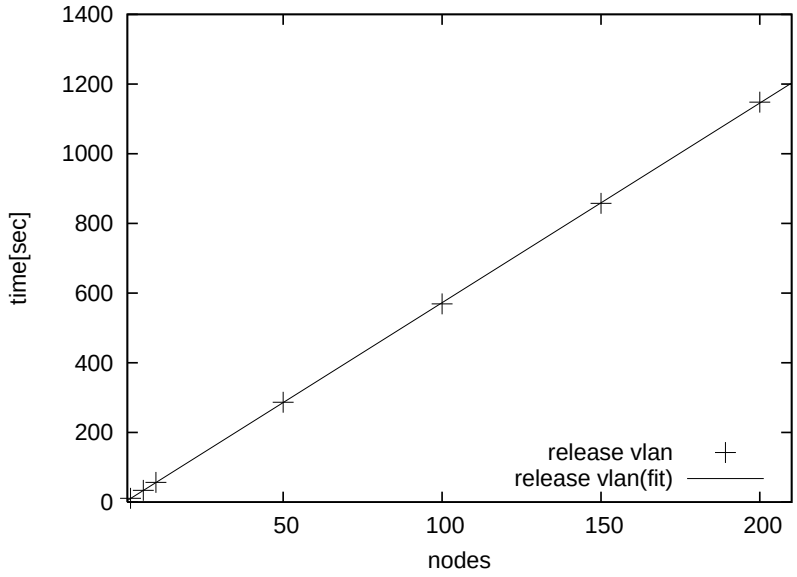

図 8.15: ノード数と VLAN 設定削除の所要時間の関係

2 個から 200 個までの VLAN 削除の所要時間は、最小自乗法で求められた直線にほぼ合致している。

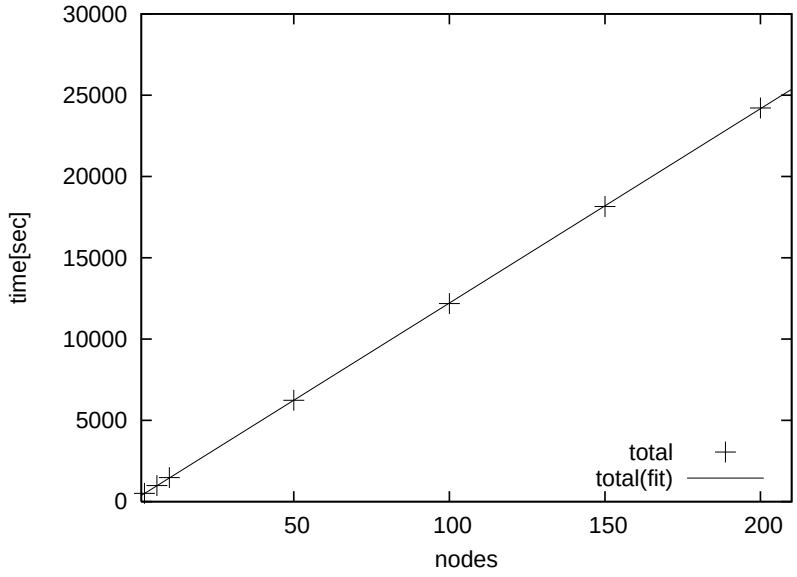

図 8.16: ノード数と実験全体の所要時間の関係

2 台から 200 台までの実験の所要時間の総和は、最小自乗法で求められた直線にほぼ合致している。

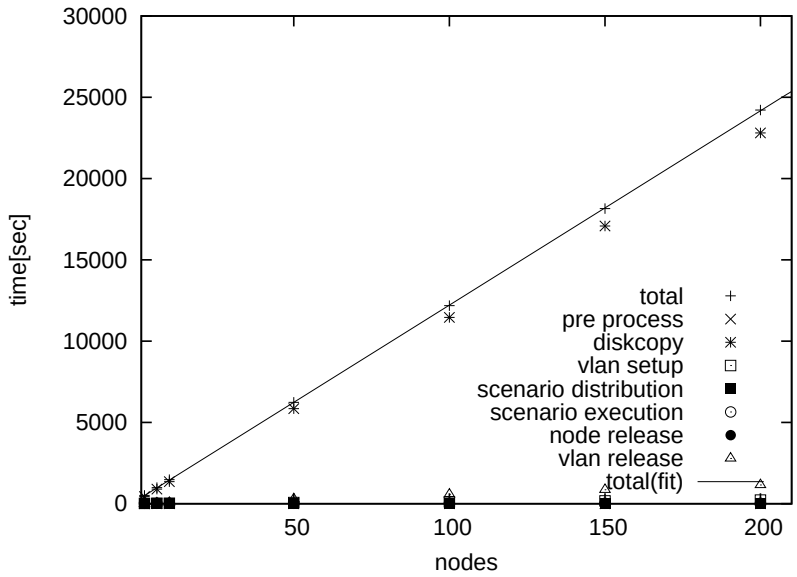

図 8.17: 実験全体の所要時間の関係

本実験の実行時間はほぼディスクコピーに必要な時間であることがわかる。ディスクコピーが必要ない実験では実験実行時間が大幅に削減できると考えられる。

8.7 SpringOS/VMの動作確認

SpringOS/VM の動作確認のため、StarBED 上で *netperf* を用いた実効帯域の計測を行った。実ノード上で動作する *netperf* のサーバプログラムである *netserver* へ、仮想機械上で動作する *netperf* のクライアントプログラム *netperf* からトラフィックを発生させるという簡単な実験である。*netperf* を動作させる仮想機械は、8 多重で動作させ合計 40 台用意したため、5 台の実ノードを利用した。

各仮想機械の実験用ネットワークインターフェースは 2 つ用意し、それぞれを実ノードの別のネットワークインターフェースを通じ実験ネットワークに接続した。各仮想機械では 2 つの *netperf* を起動し、それぞれ別のネットワークインターフェースからトラフィックを生成した。図 8.18 に実験トポロジを、図 8.19 と図 8.20 に設定記述の一部を示す。

図 8.19 は仮想機械の起動を行うための設定ファイルであり、実ノードの設定が記述されている。このクラスのインスタンスとして定義されるノードには 2 つ以上のネットワークインターフェースが用意されている。“scenario” からはじまるブロックに実行されるシナリオが記述されており、ネットワークインターフェースの設定や、VMware の設定ファイルを生成するためのスクリプトの取得と実行、そして VMware の起動を行う。

図 8.20 は仮想機械の設定である。8 多重で仮想機械を起動するよう指定されており、仮想機械を起動するための実ノード設定には図 8.19 で定義された”vmwareC” クラスを利用する。このクラスのインスタンスの仮想機械は、2 つのネットワークインターフェースを持ち、1 つは実ノードの 0 番目の物理ネットワークインターフェースを通じ “ethnet-0” というネットワークに接続され、もう 1 つは 1 番目の物理ネットワークインターフェースを通じ”ethnet-1” に接続される。シナリオにはネットワークインターフェースの設定後 kuroyuri master から *netserver* が起動しているノードの IP アドレスを取得し、そのアドレスに対して *netperf* を実行するよう記述されている。このクラスのインスタンスは”nodeset” ではじまる行で 40 台用意することが指定されている。また、これとは別にサーバ用のクラスも定義されており、サーバ用のインスタンスとして実ノードが 1 台用意される。

図 8.19 の設定部分は実験実行者が明示的に用意する必要はなく、別ファイルに用

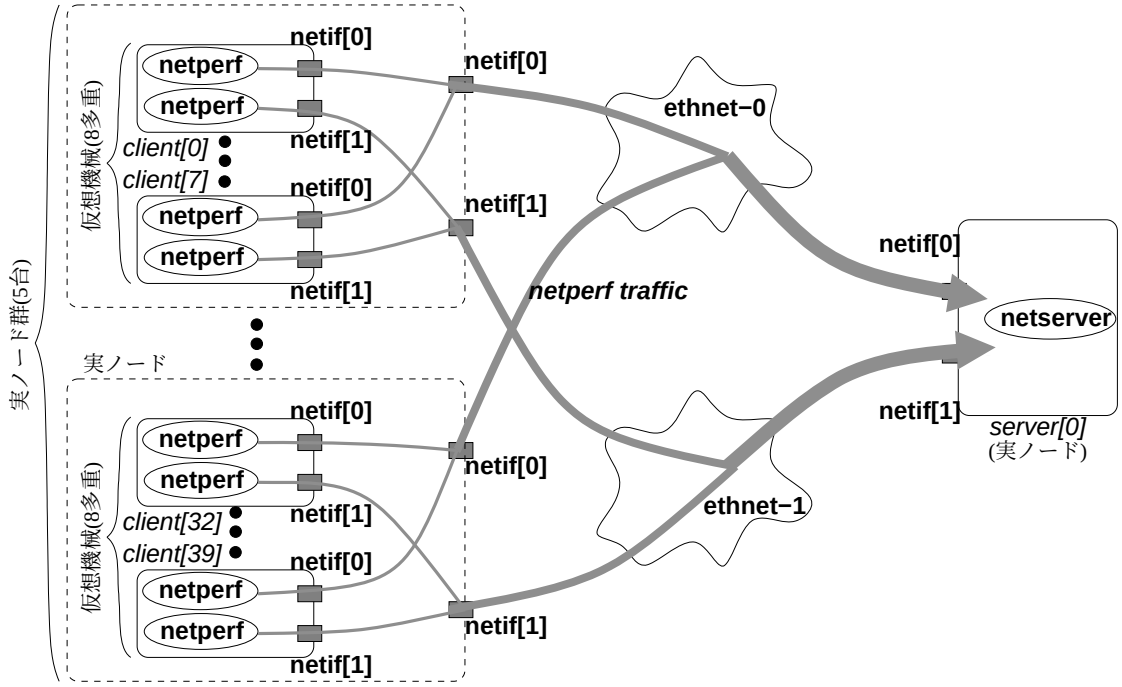

図 8.18: netperf 実験用トポロジ

5 台の実ノードを用意し、それぞれのノード上で 8 台の仮想機械を動作させた。また、別の 1 台の実ノードでは仮想ノードを動作させず *netserver* 用のノードとして動作させた。仮想機械上では 2 つの *netperf* が動作し、それぞれ異なるインターフェースを通じて別のネットワーク経由で *netserver* が動作するノードへトラフィックを流す。

```
nodeclass vmwareC {
  method "HDD"
  disktype "IDE"
  partition 4
  ostype "Linux"
  diskimage "ftp://172.16.3.253/vmhost.gz"
  netif media fastethernet
  netif media fastethernet
  scenario {
    netiffit "/usr/local/bin/ifscan"
    wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
      (haddr(self.netif[0].ipaddr))
    wakewait "/usr/bin/wget" "/usr/bin/wget" "-q" "ftp://172.16.3.253/vmsetup.pl"
    wakewait "/bin/mv" "/bin/mv" "vmsetup.pl" "/tmp/vmsetup.pl"
    wakewait "/usr/bin/perl" "/usr/bin/perl" "/tmp/vmsetup.pl" "/tmp/vm.hint"
    for(i=0;i<_launchvnodes;i++) {
      wake "/usr/bin/vmware" "/usr/bin/vmware" "-q" "-x"\\
        ("/root/vmware/linux0"+tostring(i)+"/linux.cfg")
    }
  }
}
```

図 8.19: VMware 起動ノード用クラス

VMware を起動するための実ノード用クラス設定。VMware 設定を実行する perl スクリプトを取得し、これを実行することで MAC アドレスの指定や、*vmnet-bridge* の設定を行う。

```
nodeclass clC {
  maxvnodes 8
  vntype vmwareC
  ostype "Linux"
  diskimage "ftp://172.16.3.253/linux.vmdk"
  netif media fastethernet via 0 net "ethnet-0"
  netif media fastethernet via 1 net "ethnet-1"
  scenario {
    netiffit "/usr/local/bin/ifsetup/src/ifscan"
    wakewait "/sbin/ifconfig" "/sbin/ifconfig" self.netif[0].rname\\
      (haddr(self.netif[0].ipaddr))
    recv dstA
    recv dstB
    wake "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstA
    wakewait "/usr/local/bin/netperf" "/usr/local/bin/netperf" "-H" dstB
    send "cdone"
  }
}
nodeset client class clC num 40
nodeset server class svC num 1
```

図 8.20: 仮想機械用クラス

仮想機械用のクラスでは、特に特別な指定は必要なく、*netperf* を実行するためのシナリオが用意されればよい。ただし、ネットワークインターフェースの指定として、実ノードのどの物理インターフェースに接続するかの設定が必要である。

意されているためVMwareを利用する場合は、図8.20のように”vntype” を”vmwareC” と記述すればよい。しかし、実験実行者が新たにクラスの定義を行えば、実験実行者が意図する方法でVMwareを起動することができる。また、新たなクラスを定義することで、別の仮想ノード実現手法にも対応することが可能である。

これらの設定記述を用いてSpringOS/VMを実行することにより、40台の仮想機械と1台の実ノードによる実験トポロジは問題なく構築され、*netperf* のトラフィックも発生した。これにより、SpringOS/VMが設定記述にしたがい、意図したとおりに実ノードの設定を行い、その上で仮想機械をに起動できていることが確認できた。

8.8 考 察

本節では8.6 節で示した実験結果の考察を行う。図 8.12、8.13、8.15 からわかるように、ディスクコピー、VLAN 設定、VLAN 削除はノード数と所要時間に比例関係がある。一方、図 8.14 からわかるように、シナリオの実行には特に比例関係は見られない。これは、SpringOS がシナリオを並列に実行するためである。ディスクコピーも同様に平行して行われるが、ディスクイメージが保存してあるファイルサーバの性能および、図 8.2 中のスイッチ F とファイルサーバ間、スイッチ D とスイッチ F 間の帯域がボトルネックとなり、ノード数に応じて所要時間が増加している。

手作業でノードへの OS のインストールを行った場合には、一台につき約 20 分（表 8.2）が必要となる。この場合、十数台の実験までであれば数時間でインストールが終わるが、SpringOS を用いればディスクイメージの導入には 22 分程度（表 8.7）で 10 台のインストールを完了できる。ただし、実際にはディスクイメージの元となる OS のインストールとして約 20 分（表 8.2）と、ディスクイメージの作成のための pickup の実行に 21 分（表 8.4）ほどかかるため、合計で 1 時間強の時間が必要である。したがって、人手で一台ずつインストールした場合の 3 台分の時間で、10 台のインストールが実現できるといえる。また、それ以上の台数では人手でインストールをすることは現実的ではないが、SpringOS を用いれば、図 8.11 に示したように、200 台で 6 時間強とディスクイメージの作成時間として 40 分強を足しても、7 時間程度で実現できる。台数が多くなればディスクイメージ作成にかかる時間が無視できるため、一台あたり約 2 分で OS とアプリケーションソフトウェアのインストールが可能である。

今回の実験では 1 ペアにつき 1 つの VLAN を用意したため、設定が必要なポート数だけでなく、VLAN の作成数もノード数に比例した。したがって、設定時間がノード数に比例することは自然である。VLAN の設定を人手で行う場合は、各ノードが接続されているスイッチのポートの確認など、非常に煩雑な作業が必要である。また、この理由から設定に間違いが生じやすい。SpringOS では、これらの情報をリソースマネージャが保存し、その情報により設定が自動化されるため、正確かつ迅速な設定を行える。

図 8.14 はシナリオ実行の所要時間を示している。シナリオ実行の所要時間はノード数に関係ないことが見て取れ、100 ペア程度の実験であれば、制御性能は十分であることがわかる。人手により実験を実行した場合には、各ノードでコマンドを入力することになり、これだけの短時間で実験を終了させることは困難である。

また、図 8.17 からは、ディスクコピーの所要時間が実験所要時間の大半をしめることが見て取れる。ディスクイメージの変更が必要なければ、シナリオでノードを *shutdown* せず、繰り返し実験に利用することができる。その場合は、利用する実ノードとシナリオ内の論理ノードの対応を記述し、master にディスクコピーをさせないように実行する。これにより、ディスクコピーが必要ない、小さなパラメータの変更のみによる再実験などを効率的に行える。

本節では、master が実行している時間を所要時間として表現してきた。しかし実際には master を実行するだけで、設定記述に記された実験はすべて自動で行われる。したがって、200 台の実験であっても、設定記述を用意し master を実行すれば実験実行者は実験が終わるまで、別の作業をすることができるため、実験実行者の実験への拘束時間は非常に小さい。

表 8.5 〜 8.11 に示したとおり、台数に関わらず実施した 3 回の計測結果の値の分散は小さかった。これにより、StarBED および SpringOS が安定していることがわかる。

これらから、StarBED および SpringOS では、以下の 2 点を達成したといえる。

- 人手で構成することが現実的でないような構成での実験を容易に実行できる
- 何度実行しても同様の結果が得られる

人手で実験を実行するには時間的に現実的でないという点だけでなく、多数の設定が必要になる実験では、すべての設定を間違いなく行うことは困難である。SpringOS を利用すれば、機械的に多数のノードの設定およびシナリオ実行が行われるため、ある程度の再現性を確保しているともいえる。

また、すでに 4.1 章で述べたように、StarBED および SpringOS を用いて行われた実験も数多く、SpringOS を StarBED よりも小規模な環境に導入している組織も存在する。このようなことからも、StarBED および SpringOS は有用であるといえる。

第III部

より高度な実験支援環境の構築にむけて

第 III 部では、より高度な実験支援のための議論を行う。

第 II 部では大規模実証環境の一実装として StarBED および SpringOS の詳細について述べた。このような経験から 9 章では、大規模実証環境に一般的に必要となる機能を整理する。これにより、実験実行者が、既存の大規模実証環境の機能を、利用できる大規模実証環境を選択できる。また、その機能を実現するためのアーキテクチャを整理し、複数の大規模実証環境の協調による、一つの実験駆動単位の構築の実現可能性を示す。

大規模実証環境とソフトウェアシミュレータでは、実験駆動単位の構築と実験実行の支援を行い、実験を容易かつ正確に実行できる。これらを利用するためには、事前に実験内容の決定が必要であるが、実験実行者が実験内容を決定した場合には、不適切な実験が行われる可能性がある。10 章では、実験実行環境に入力する実験内容の決定支援に関する議論を行う。

第9章

汎用的な実験支援ソフトウェアのアーキテクチャの提案

本章では、これまでの経験にもとづき、実験支援ソフトウェアに必要な機能を整理し、さらに、それらを実現するアーキテクチャを提案する。また、StarBED だけでなく、その他の実験設備との協調し、実験駆動単位に複数の実験設備を利用することで、大規模化が可能なだけでなく、より柔軟な実験を行うことができると考えた。これを実現することも、本アーキテクチャの目的のひとつである。

実ノードを利用した環境での実験の手順については、5.1 節で述べた。本章で対象にするのは、図 5.3 のうち、実験駆動単位の構築および実験シナリオの実行に関わる手順 2 から手順 6 とする。

9.1 大規模実証環境に求められる機能の実現

5 章で、大規模実証環境で実現する必要がある要件について述べ、StarBED および SpringOS の設計・開発とその評価について述べた。本節では、これらの経験をふまえた上で、より汎用的な支援ソフトウェアへの要件をまとめる。

ただし、本アーキテクチャでは以下の点を前提とする。

1. 実験実行者は必要な資源を大規模実証環境に要求し、割り当てを受けその資源を用いて実験駆動単位を構築する。

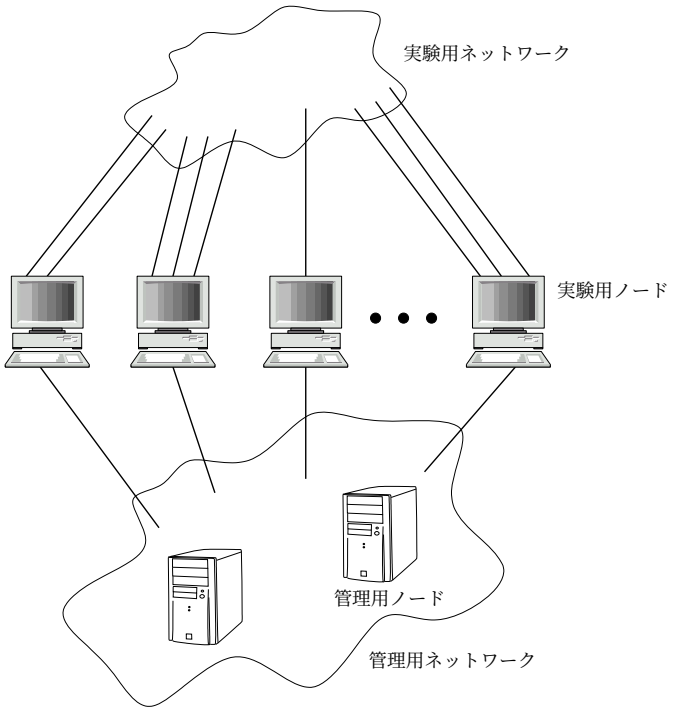

図 9.1: 想定する実験設備のトポロジ

StarBED と同様に管理用と実験用のネットワークが分離されていることを前提とする。これによりトラフィックの相互干渉を防ぐとともに、管理用ネットワークを通して実験用のネットワークを自由に再構成できる。

2. 同時に複数の実験実行者が大規模実証環境を同時に利用でき、大規模実証環境上に複数の実験駆動単位が生成される。

3. 実験駆動単位を構成するノードのネットワーク設定を実験実行者の意図どおりに行うため、実験用とは別に管理用のネットワークが用意され、すべてのノードが管理用ネットワークに接続されている。

図 9.1 に前提とするトポロジの概念を示す。

9.1.1 資源管理および資源割り当て

実験実行のためには、大規模実証環境に存在する資源の情報を管理し、実験実行者に割り当てる必要がある。この機能を持つモジュールをリソースマネージャ

と呼ぶ。

リソースマネージャが管理するべき資源は、大規模実証環境の実装にも依存するが、ノード、帯域などがあり、実験ネットワークを VLAN を利用して構築する場合は、VLAN 番号も資源情報として管理される。また、実験駆動単位の制御機構が動作するノードも資源として管理される。

リソースマネージャには、大規模実証環境の資源を管理するだけでなく、資源を実験駆動単位または実験実行者に割り当てる機能も必要である。実験実行者は設定記述として、それぞれのノードやリンクに必要とする機能を記述する。この設定記述にしたがい、実験駆動単位を制御するモジュールが、大規模実証環境のリソースマネージャに割り当て要求を行う。リソースマネージャはこの要求を満たすノードを何らかのアルゴリズムで選択し割り当てる。また、基本的には割り当てたノードを別の実験実行者に割り当てないよう排他制御を行う。

大規模実証環境での実験の実行方式として、利用する時点で資源が空いているかどうかを確認し、空いていれば実験実行するといったバッチ型の実行方式もあるが、ある期間に必要数のノードを確保しておき、実験を行う方式も考えられる。柔軟な実験制御のためには両方式に対応することが望ましい。資源をある期間、実験実行者に割り当てる場合は、予約状況も資源の情報として保持されなければならない。

9.1.2 資源利用の排他処理

大規模実証環境によっては、資源は単一の実験実行者に占有されるばかりではなく、複数の実験実行者に共有される可能性がある。これにより、大規模実証環境上の資源をより効率的に利用できる。このためには、実験実行者が必要としている資源に応じて共有できるかどうか、共有できるのであればその程度などが考慮されなければならない。ノードが共有される場合は、CPU やメモリの量などから共有が可能であるかが決定され、リンクについては必要な帯域や、許容できる遅延などの情報をもとに共有できるか判断される。大規模実証環境によっては、リソースマネージャが、資源共有のためのアルゴリズムを持っている必要があるが、実ノードを用いている場合は、指定した値でノード負荷やネットワーク特性を固

定することは不可能であるため、実験実行者による機能要求は、ある程度の範囲での指定を行うこととする。

ノードやリンクの割り当ての排他制御についてはすでに述べたとおり、リソースマネージャが管理する。これに加え、実際に資源に操作を行う際に、その実験実行者が対象の資源に対して操作が許されるのか、また、どの程度許されるのかを判断し、実際の操作を制限する必要がある。たとえば、大規模実証環境の中心となるリンクなどは、複数の実験実行者により共有される可能性は高い。このような場合にはその他の実験に対して影響をおよぼさないよう、管理される必要がある。共有されたリンクへの設定は、その実験実行者に割り当てられた範囲での設定のみに制限されるような機構が必要となる。リンク以外でも、共有される可能性のある資源の割り当てやそのような資源に対する操作を行う機構では、同様の排他制御を行う必要がある。

9.1.3 設定読み込みおよび実験駆動単位の生成

実験を実行するためには、実験駆動単位を生成する必要がある。実験駆動単位は制御モジュールと実験用ノードで構成され、実験用ノードは制御モジュールにより施設から割り当てられ、管理される。したがって、実験駆動単位を構成するためには、まず、制御モジュール群を起動しなければならない。

このためには、制御モジュール群が動作するノードの割り当てが必要である。割り当て要求は入力された設定記述から、実験の規模などを考慮して行われ、リソースマネージャはそれを満たす機能を持つノード群を割り当てる。このようにして起動された制御モジュール群は設定記述を詳細に読み込み、実験資源を確保し、実験駆動単位を生成する。

9.1.4 ノード制御

ノードを柔軟に制御することにより、ノードの初期設定だけでなく、リンクの特性の制御や、トポロジの構成変更などさまざまな機能が実現可能である。本節ではこれらの機構についてまとめる。

ノードの電源管理

実験を行うために各ノードの電源投入が必要であり、実験終了時には電源を切断する機能が必要である。容易な操作で多くのノードの電源を操作できることが望ましい。また、実験の準備段階や実験実行中にノードを再起動させる必要が生じることもあるため、再起動も行える必要がある。

ノードへのソフトウェア導入および設定

ノードのディスクに対して、あらかじめ用意されたディスクイメージの書き込みを行ったり、既存の OS に対してのアプリケーションソフトウェア導入および、それらの設定を行える必要がある。また、ハードディスクを利用してだけでなく、ディスクレスシステムとしてノードを動作させることにより、より時間的コストを軽減し、実験用ノードとして振る舞わせることができる。

トポロジおよびリンク特性の設定

指定されたトポロジを自動的に設定する仕組みと、各リンクの特性を変更できる仕組みが必要である。リンクの特性としては帯域やジッタなどが挙げられる。

ノードでのコマンド実行

基本的に実験シナリオは、各ノードで実行されるコマンドの集合といえる。各ノード上でコマンドを自動的に実行するための機構を用意し、設定記述で記述されたコマンドを各ノードで実行することで実験シナリオを実行する必要がある。この時、複数のノードで同期してコマンドを実行する必要が生じる場合があるため、あるノードでのイベントをトリガにして別のノードでコマンドを実行する仕組みが必要である。また、これ以外にもさまざまなトリガによるシナリオ制御ができることが望ましい。

ノード時刻の同期機構

実験によってはノードの時刻を同期させる必要がある。実験駆動単位内で統一されればよい場合と、実時刻と同期する必要がある場合がある。ネットワークを通して、ノードの時刻を同期するためにはNTPなどを利用する方法が一般的であり、各ノード上でのコマンド実行により実現できる。

実験設備の初期化

大規模実証環境によっては、その設備に、標準的なOSやアプリケーションソフトウェアが導入されており、利用後は初期状態に戻すことを要求される場合がある。これに対応するため、容易に実験設備を初期化できる枠組みが必要である。

実験設備の初期化は、初期化の対象となるノードを利用した実験とみなすことができ、ノードの初期状態のディスクイメージや設定ファイルの導入または、コマンド実行機能での初期化も可能であるため、専用の機構を用意する必要はない。

また、リソースマネージャと協調して動作することで、実験期間が終了した際に自動的に初期化を行うことも可能である。

実験実行者による手動実験制御

何らかのトラブルが発生した際に、実験実行者が、該当ノードに接続し、その操作を行いたいという要求は大きい。実験実行者が大規模実証環境の物理トポロジを意識せず、容易にノードのコンソールなどにアクセスできる機構が必要である。

9.1.5 実験設備および実験駆動単位の状態監視

大規模実証環境としては、各ノードの電源の状況や管理ネットワークでの到達性などを検知できればその健全性を確認できる。また、実験駆動単位ではより詳細な情報を管理したいという要求がある。各実験用ノードのOSやシナリオ実行の履歴、ノードの各インターフェースの情報からトポロジ情報も取得することで、実験が正しく動作しているのかを確認できる。したがって、大規模実証環境としてその健全性を確認する機構と、実験駆動単位で詳細に実験用ノードやリンクの

状況を確認する機構が必要である。さまざまな方法でノードの状況は確認できるが、これを実験実行者にわかりやすく提示する機能が必要となる。

9.1.6 ログ収集

ログは一般的には実験用ノード上のファイルに保存されるか、外部のノードに転送されまとめて保存される。したがって、ノードからのログファイルの転送もしくは、ノードからのログメッセージの転送ができればログの収集に対応できる。

9.1.7 実験状態の保存と復元

利用資源の制限などから、実験を中断する場合がある。このような場合に実験状態を保存しておき、後日保存された状態から実験を再開することができれば、初期設定などの工数を削減できる。また、ノードの状態を保存しておき、破壊的な作業を行ったあと元の状態に戻すということも実験によっては必要とされる。

ある時点でノードのディスクイメージを保存することで、ディスクの状態は保存することができるが、メモリや通信のためのコネクションの状態、さらには、リンク上のトラフィック状態を保存することは困難である。ただし、大規模実証環境の実装によりある程度は提供できる場合もあるため、実現方法は大規模実証環境の実装に依存する。

9.1.8 複数の大規模実証環境の協調

実験実行者にとっては、それぞれの大規模実証環境の性質や操作方法の詳細を学ぶことなく統一されたユーザーインターフェースで実験が行えることが望ましい。したがって、各大規模実証環境が連携し、実験実行者には複数の大規模実証環境を利用していることを意識させないインターフェースが必要である。

複数の大規模実証環境を利用した実験駆動単位を構築するためには、それぞれの大規模実証環境に存在する制御機構の協調が必要であり、このために各大規模実証環境に存在するモジュール間の一般的な通信方式が必要である。よほど特殊な機能を提供している大規模実証環境以外は、これらの機能の大規模実証環境内

部での実現方法を隠蔽し、外部からは一般的なインターフェースでアクセスでき、複数の大規模実証環境を利用した実験駆動単位が構築できる。

9.1.9 例外処理

実験対象の動作を事前に予測することは困難であり、実験中に、なんらかの例外が発生することは、想像に難しくない。したがって、それぞれの機能に例外処理機能が必要であり、それらを統一的に管理する例外処理機構が必要である。

実験設備および実験駆動単位の状態監視機構と連携することで、ノードの状況に応じて何らかのイベントを実行することが可能である。また、実験実行者はそれぞれの状況に応じた処理内容を指定できる必要がある。この時、考えられる処理としては以下が挙げられる。

- 何らかのコマンドを実行する
- 実験を終了する
- 無視して実験を継続する
- 実験を一時停止し実験実行者からの指示を待つ

9.2 提案アーキテクチャ

本節では前節で挙げた要件を実現する汎用アーキテクチャを提案する。

各機能の詳細な実現方法は、実験設備により異なるため、提案アーキテクチャでは、各機能を抽象化し、それぞれの実現手法を示す。また、7 章で述べたとおり、SpringOS は実験の準備段階と実験シナリオの実行段階を明確に区別していたため、シナリオの実行中のネットワークトポロジの変更や、ノード故障時に新たなノードを用意するといった機能を実現できなかった。提案アーキテクチャでは特にこのような段階をつくらず、実験の準備段階は、実験シナリオの一部としてのノード操作であると捉えて実験を進める。

以下、機能別に大別して提案するアーキテクチャを示す。

9.2.1 資源管理

資源情報を 2 種類に分類する。一つは資源の静的な情報である。ノードの情報であれば、ネットワークインターフェースの種類や数、CPU、メモリ、ハードディスクなどに関するハードウェア情報とノードの ID や初期状態でインストールされている OS の情報、管理用のネットワークインターフェースのネットワーク設定のような設定に関する情報が保持される。また、リンクの情報であれば、メディアや帯域などが保持される。これらの情報は故障発生時などの例外的な場合にのみ書き換えられる。これに加え、資源の割り当てなどに関する動的な情報がある。ある資源がその時点で、どの実験もしくは実験実行者に割り当てられているのかといった情報は頻繁に書き換えられる。また、ある実験実行者に割り当てることができる資源を管理する必要もある。これはいうなれば予約の管理である。

この 2 種類の資源情報を管理するモジュールを分離し、動的資源を管理するモジュールを Dynamic Resource Manager（DRM）、静的資源を管理するモジュールを Static Resource Manager（SRM）とする。SRM は資源の静的な情報を管理し、DRM は資源の動的な情報を管理する。また、資源の状態を保持するのみではなく、実験実行者への資源の割り当てを行う。さらに、実験実行者により持ち込まれた機材などを管理するため、DRM、SRM ともに、実験実行者が機材の情報を登録できるインターフェースも必要である。

DRM と SRM は大規模実証環境全体に存在する資源を管理し、実験実行者への資源割り当てを調停する必要があるため、大規模実証環境単位ごとに用意する必要がある。今後、リソースマネージャという表記は DRM と SRM 双方を指すこととする。

9.2.2 設定読み込みおよび実験駆動単位の生成

実験駆動単位を構築するためには、まず実験駆動単位の制御モジュールを起動しなければならない。このために、まず、設定記述を読み込み、その実験用の実験駆動単位の規模などに対応できる性能を持つノードを制御モジュール用に割り当てる必要がある。この機能を持つモジュールを Experiment Scheduler（ES）と呼び、大規模実証環境ごとに用意する。ES は大規模実証環境での実験実行のスケ

ジューリングを行う機能を持ち、設定記述にもとづき、指定されたタイミングで実験駆動単位を生成する。実験駆動単位には後述するさまざまな制御モジュールが必要であり、ES はこれらの制御モジュールを管理する Experiment Manager（EM）を起動する。具体的には ES が、EM の動作に必要な割り当てを DRM に要求し、DRM は管理する資源を検索し、要求を満たす EM のためのノードが割り当て可能であれば、ES にそれを割り当てる。ES はこのノード上で EM を起動する。起動された EM は ES からすべての設定ファイルを受け取り、より詳細に設定記述を解析、他の制御モジュールを起動するために、必要な性能を持つノードをそれぞれ要求する。EM 以外の制御モジュール群が起動すると、ES からノード設定や、シナリオ実行についての設定記述など、それぞれが必要とする部分の設定を受け取り、実験を実行する。

EM によって起動されるべきモジュールを、今後説明するものも含めて以下に挙げる。また、図 9.2 に動作手順を示す。

- Scenario Driver: シナリオの駆動
- Resource Information Manager: 実験駆動単位に割り当てられたノード情報の管理
- Topology Manager: 実験駆動単位のトポロジ制御
- Node Status Manager: ノードの状態監視
- Log Manager: ログ管理
- Node Initiator: ノードへのソフトウェア導入
- User Power Manager: ノードの電源管理

9.2.3 ノード機能の指定および割り当て

すでに制御用の資源割り当てについて簡単に述べたが、実験用ノードも基本的に同様の手法で割り当てられる。制御用資源は実験の規模などから必要性能を決

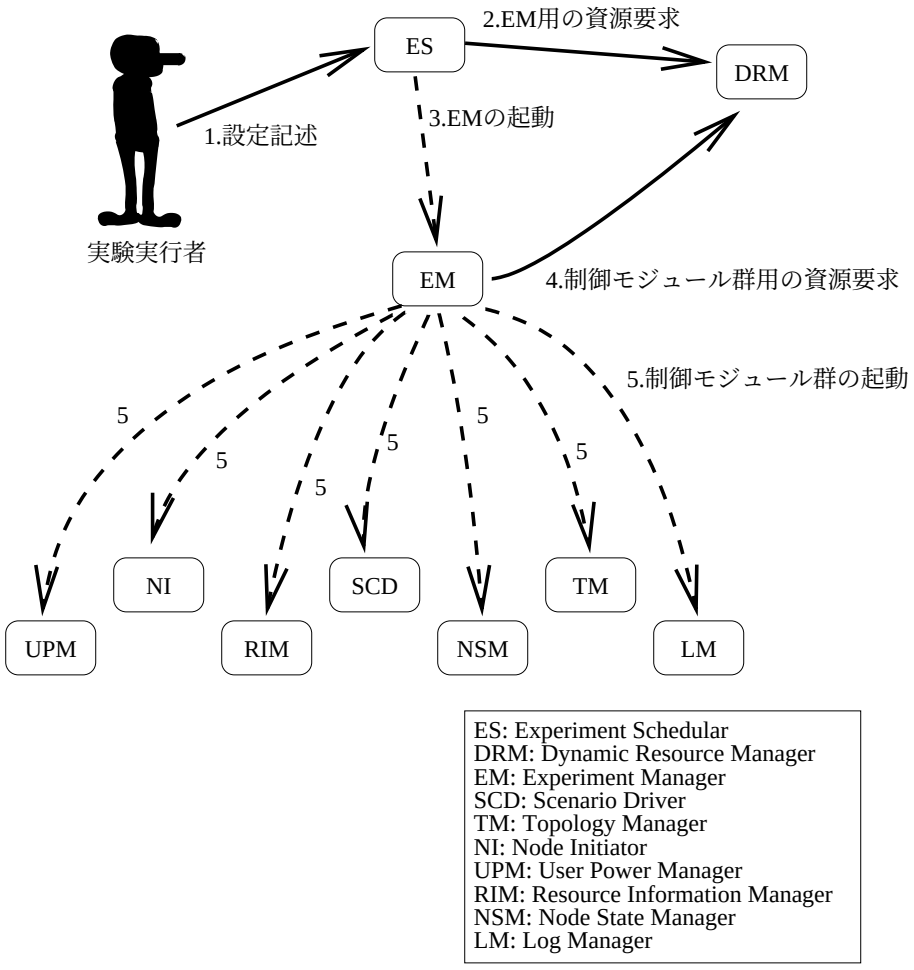

図 9.2: 実験駆動単位の構築手順

実験の設定記述を読み込んだ ES はその実験に割り当てる EM 用の資源を DRM に要求し、割り当てが成功すれば ES はその資源を用いて EM を起動する。EM は設定記述を ES から受け取り、その設定記述に必要なだけの管理用モジュールを動作させることができる資源を DRM に要求する。割り当てが成功すれば、EM はそれらの資源を利用してその他の管理用モジュールを起動する。

定したが、実験用ノードの機能は実験記述に具体的に記述される。また物理ノード名などによる静的な指定も受け付けられる必要がある。

EM が DRM へ資源要求を行い、DRM がその条件を満たす資源の検索を行い、割り当てを行う。割り当てに関する情報は DRM に保持されるが、より詳細な情報を保持するための制御モジュール Resource Information Manager（RIM）に登録される。SRM、DRM は、大規模実証環境の管理のためのモジュールであり、実験記述中で指定されたノードと物理ノードの対応や、動作すべき OS など、実験の詳細に関わる情報を持たないため、RIM がこのような情報を保持する。

図 9.3 に資源管理の手順を示す。図中の手順 0 の予約が必要であるかどうかは大規模実証環境の運用ポリシにより異なる。

9.2.4 ノードの電源管理

ノードの電源投入や電源断、再起動には、Magic Packet Technology や IPMI、iLO を利用したハードウェアレベルでの実現方法と、SNMP やコマンド実行による、ソフトウェアレベルでの実現方法がある。ハードウェアレベルでの制御を行うためには、大規模実証環境のノードのハードウェアにそのための機能が実装されている必要がある。また、ソフトウェアで実現する場合は、その機能を実現するプログラムがノード上で動作していればよいが、ノードの起動処理や、ハングアップ状態からの復帰処理を行うことはできない。

ノードの電源管理は、ノード上で動作する Power Agent（PA）と、実験駆動単位の制御モジュールである User Power Manager（UPM）で行われる。具体的には、PA は SNMP や IPMI などの機能を実現するハードウェアもしくはソフトウェアである。UPM に対してあるノードの電源管理要求が行われると、UPM が PA と通信し、要求された方式での電源管理を行う。電源管理を行う際に、UPM は RIM に各ノードの状態を問い合わせ、他の実験駆動単位と共有されているノードの場合は、大規模実証環境で動作する Power Manager（PM）に要求を行う。PM はリソースマネージャと通信を行い、要求された操作を行っていいのかどうかを確認した上で、可能であれば処理を行う。

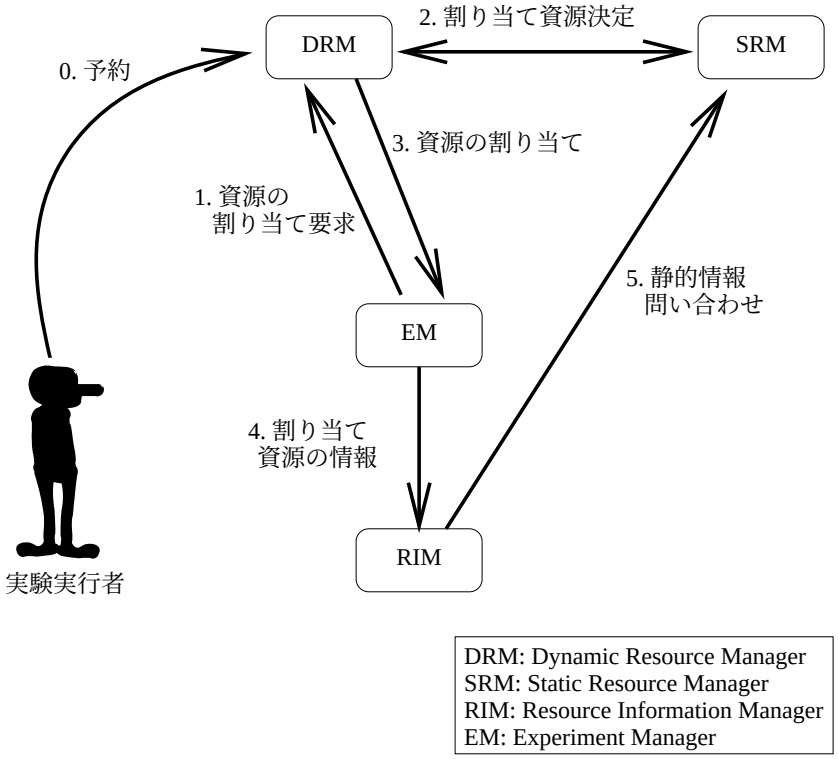

図 9.3: 資源割り当て手順

（0）予約が必要な環境では、実験実行者が DRM に予約情報を入力する。DRM は SRM と通信し、要求を満たすノードを検索し、割り当て可能な資源が発見できた場合は、資源を実験実行者に割り当て、その情報を保持する。実験実行時には、（1）EM から DRM に資源要求を行い、（2）DRM は SRM と通信し予約でその実験実行者に割り当てられた資源内で割り当てられるリソースが存在するかを検索し、（3）割り当てが可能であれば EM に資源を割り当て、その情報を保持する。（4）実験駆動単位では、割り当てられた資源の情報は RIM に保存され、詳細な情報が必要な場合には（5）RIM から SRM に問い合わせが行われる。

9.2.5 ノードへのソフトウェア導入および設定

ノードの起動方法は、ハードディスクに導入されているOSを利用する場合と、ディスクレスシステムとして起動する方法、そして、外部ノードから取得したディスクイメージをノードに導入し、導入したOSで起動する方法が考えられる。ノードのハードディスクに新たにOSを導入する場合は、ディスクレスシステムとして起動後、外部ノードからディスクイメージを取得し、ハードディスクに書き込み、ノードをそのハードディスクから起動するよう変更して再起動すればよい。ディスクイメージの書き込みなどの処理は、ディスクレスシステムで動作するコマンド実行機構を利用し、ディスクイメージのダウンロードや書き込みをコマンド実行により実現すればよい。したがって、ハードディスクに導入されているOSからの起動と、ディスクレスシステムとしての起動を切り替える機構および、コマンド実行機構により、前述の3手法に対応できる。また、アプリケーションソフトウェアの設定や、ノードのネットワーク設定など基本的な設定は、後述のコマンド実行機能を用いて行う。

StarBEDおよびSpringOSで採用している起動方法の切り替えの手法として、すべてのPCノードが起動時に必ずPXEでブートローダを取得し、それにしたがい起動する方法がある。この方法を用いると、各ノード用のブートローダを動的に変更することにより、PCノードの起動方法を切り替えられる。

それぞれのノード起動方法や、導入するディスクイメージなどはRIMに問い合わせる。ノードの起動方法はUser Boot Changer（UBC）が変更する。たとえば、PXEでノードが起動する際にブートローダに関する情報をDHCPで取得し、その情報をLDAP[42]やシンボリックリンクを利用することで変更する方法が考えられる。なお、他の実験とノードを共有している場合は、大規模実証環境で動作するBoot Changer（BC）が調停を行う。

ノードへのソフトウェアの導入の際には、Node Initiator（NI）が主体となって処理を進める。NIの指示によりコマンドを実行する、Node Agent（NA）がすべてのノード上で動作しており、ノードに対する操作を行う。NIはNAおよびUBC、PMと協調しノードへソフトウェアを導入する。BCおよびNIは場合によってはファイルストレージを持っており、ブートローダーやディスクイメージおよび必要なソフトウェアなどを保持する。

ノードのハードディスクへのOS導入手順を図 9.4 に示す。

ディスクイメージの生成方法は、ハードディスクもしくは1パーティションの情報を、すべてファイルとして保存してしまう方法がある。また、ディスクレスシステムとして起動する場合は、それぞれのOSで専用の手法が用意されている場合が多い。PCノードの場合はこういった手法が利用できるが、スイッチノードの場合は実装によりさまざまである。

9.2.6 ノードでのコマンド実行

シナリオの実行は各ノード上でコマンドを実行することで進められる。各ノードでのシナリオの実行状態や、ノード間同期を実現するScenario Driver（SCD）がEMから設定記述を受け取り、すべてのシナリオを管理する。各ノード上では、コマンドを実際に実行するScenario Agent（SCA）が動作する。

各ノードの同期はさまざまな方式により実現可能であり、大規模実証環境内で矛盾なくシナリオを実行できれば問題ない。

SCAの更新やバグに備え、SCAを起動するためのScenario Agent Loader（SCAL）を用意する。SCALにはOSで提供される起動スクリプトや、非常にシンプルな機能を持ったSCAが利用できる。シナリオ実行の手順を図 9.5 に示す。

9.2.7 トポロジおよびリンク特性の設定

自動的なトポロジ設定のため、実験設備には、外部からの設定変更によりトポロジを変更できるスイッチを導入する。VLANやATMのVP/VCによる仮想的なネットワーク構成の変更や、物理的に配線を変更できるネットワーク機器が利用できる。また、VLANを扱えるPCノードはこのようなスイッチノードと同様に利用することができる。

実際の設定はスイッチでのシナリオ実行が行えればよい。DRMからトポロジ設定に必要な資源の割り当てを受け、この資源を用いて各スイッチでコマンドを実行し実験トポロジに関する設定を行う。ただし、複数の実験駆動単位で共有されるスイッチでのVLANなどを用いたトポロジ変更や帯域制御に関する設定は、他

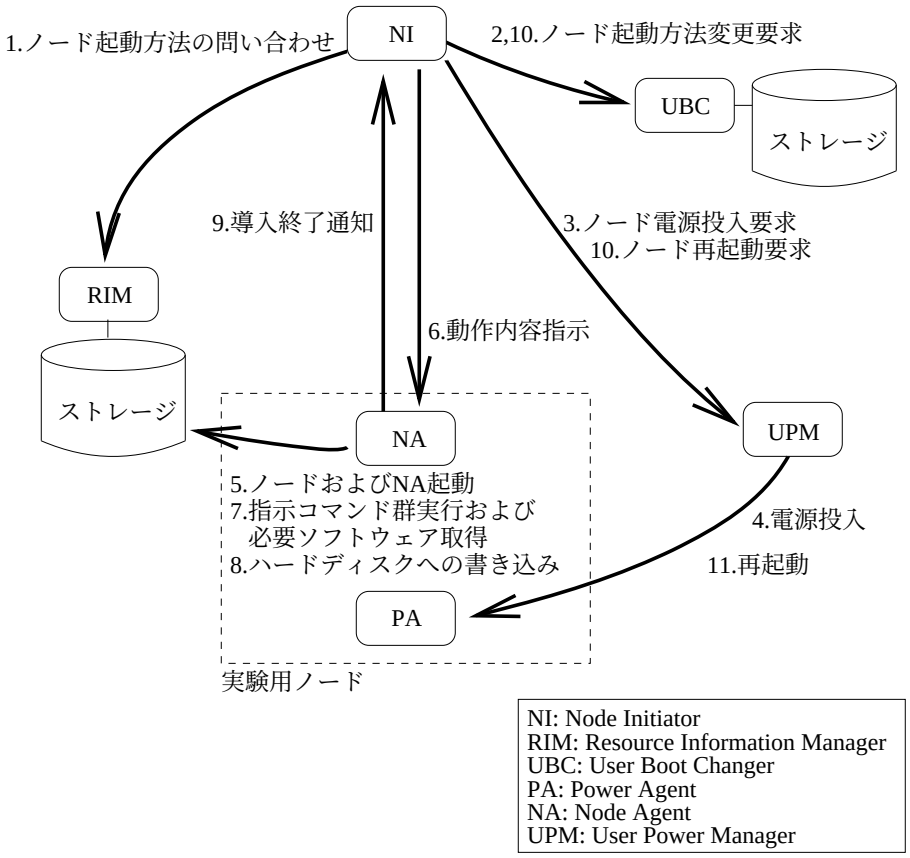

図 9.4: ノードのディスクへのソフトウェア導入手順

（1）NI は RIM に対してノードの起動方法の問い合わせを行う。ローカルディスクへ OS を導入する場合は、（2）NA が起動する専用の OS でノードを起動するよう、ノードの起動方法の変更要求を UBC に対して行う。UBC はこの要求にしたがって対象ノードの起動方法を変更する。その後、（3）（4）NI は UPM をとおして対象ノードの電源投入を行う。（5）対象ノードが起動し、NA が起動すると、（6）NI は NA に対して実行コマンド内容などの指示を行い、（7）NA はこれにしたがいコマンドを実行し、（8）ローカルハードディスクの内容を書き換える。終了後、（9）NA は NI に対して実行終了を通知し、NI は対象ノードを新たにインストールした OS で起動するため、UBC、UPM と協調して、（10）起動方法の変更と、（11）再起動を行う。

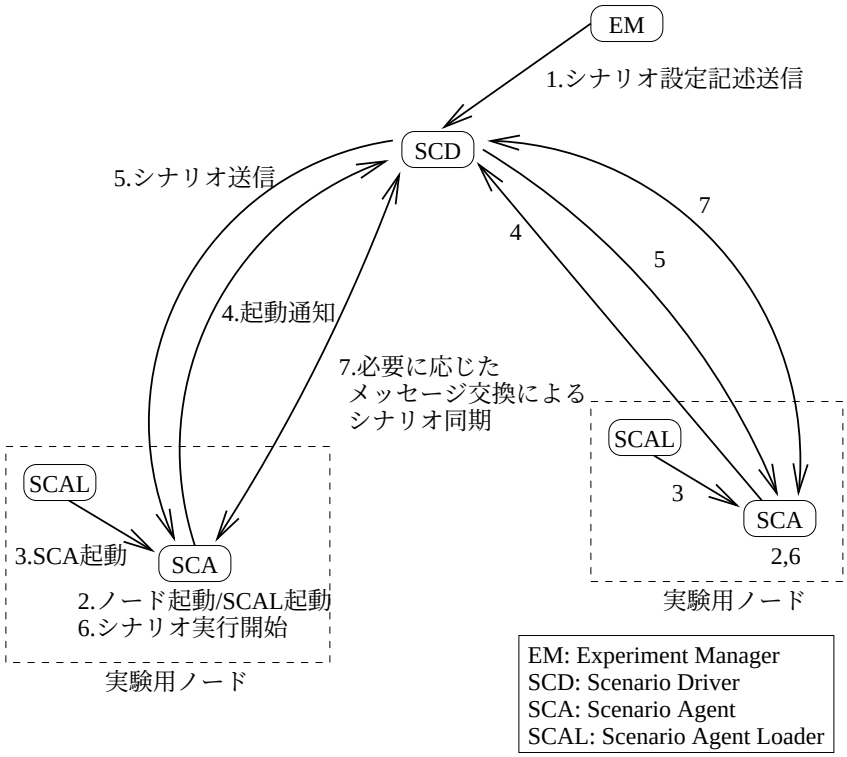

図 9.5: シナリオの実行手順

シナリオ実行は SCD が中心となり行う。(1) SCD は EM からシナリオ記述を受取る。(2) 実験用ノードでは SCAL が動作しており、(3) 指定された SCA を取得し起動する。(4) SCA が起動すると SCD に対し起動を通知する。これを受けた SCD はシナリオ実行が可能であると見なし、(5) 対応するノードシナリオを送信し、(6) SCA はシナリオの実行を開始する。シナリオ実行上、他のノードとの協調が必要な場合は、SCA と SCD 間の (7) メッセージ交換により実験進行の停止、再開を行う。

の実験駆動単位との調停が必要であるため、大規模実証環境に用意された Shared Link Manager（SLM）を通して行う。

9.2.8 大規模実証環境および実験駆動単位の状態監視

ノード状態の監視方法として、SNMP や ICMP[43] など一般的なプロトコルを利用する方法と、独自の方式を持ったアプリケーションソフトウェアを用いる方法がある。どちらの方法でも各ノード上で、専用のアプリケーションソフトウェアが動作している必要がある。このような情報の管理は、特に実験の詳細な情報を必要とせず、基本的な情報から状態を検知するため、大規模実証環境の機能として提供する。ノードが異常な状態に陥っていた場合は、DRM に通知され、DRM から実験駆動単位の RIM や EM に通知される。

実験駆動単位では実験用ノードやシナリオに関するより詳細な情報を保持しており、状態管理に関しても、これらの情報を用いて、より詳細に行いたいという要求がある。対象情報として、OS の種類やバージョン、実験シナリオの実行状態などがあげられる。これらの情報を確認することにより、実験の進行状態や、障害を検知することも可能である。これらは、実験駆動単位の制御モジュールである Status Manager（SM）と、実験用ノード上で動作する Status Agent（SA）により実現される。大規模実証環境で動作する Health Checker（HC）は基本的に電源の入切などおおまかな状態を監視する。しかし、実験駆動単位では、ノードの CPU 負荷やネットワークインターフェースの入出力も管理の対象となる。また、SA と協調することにより、実験の進行状況も管理する。取得したノードの情報は RIM に反映される。

ノードだけではなく、設定したリンク特性が正しく設定されているかを知るために、トラフィックを発生させ帯域などを検証したり、設定どおりにトポロジが構築されているのかなどを検証する必要もある。これらは、ICMP などを利用したノード間の疎通確認や、*netperf* や *iperf* を利用したノード間の帯域確認などが利用できる。この結果を Topology Manager（TM）に反映することで、実際のトポロジ情報を管理する。

ただし、実験によって取得されるべき情報は異なるため、実験実行者の設定記

述にしたがって監視対象を決定する。またHCやSMがノードの障害を発見した場合は、DRMに通知し例外処理を行う。

9.2.9 ログ収集

ログ収集は、Log Manager（LM）、Log Collector（LC）およびLog Agent（LA）によって実現される。LAは、ノード上で動作しログを残すアプリケーションソフトウェアであり、基本的には実験対象のアプリケーションソフトウェアがこれに該当する。

LCは、このようなアプリケーションソフトウェアが残したログを収集する。LCは、実験対象が動作するノード上で動作する場合と、管理側のノード上で動作する場合がある。実験対象のアプリケーションソフトウェアがログをファイルに直接書き出す場合が、LCが実験用ノード上で動作する場合の一例である。この場合、実験対象のアプリケーションソフトウェアは実験ログを出力するLAとそれを保持するLCの2つの役割を持つ。*syslogd*[44]などを用いてアプリケーションソフトウェアのログを別のノードに送信し、そのノードに保持する場合がLCが管理側ノードで動作する例である。

図9.6にログ収集の概念図を示す。LMはLCが保持した情報を最終的に回収し、実験実行者が複数のノードにアクセスする手間を低減する。また、ログの種類によっては、ログを解析しやすい形に整形するログ解析支援を行う。

このようにLCとLMの2段階の制御を行うのは、各ノード上のログがそのノードに保存される場合と、別のノードに送信されて保存される場合双方に対応するためである。それぞれ、実験により向き不向きがあるため、双方に対応できる必要がある。

9.2.10 実験設備の初期化

実験設備の初期化は、対象のノードを利用した実験駆動単位を生成し、各ノードに標準の設定を行うシナリオを実行することで実現できる。

Node Cleanup Manager（NCM）はDRMと連携し、予約割り当て後などに、その実験で利用されたノード群の初期化を行う。手法はすでに述べたとおり、対象

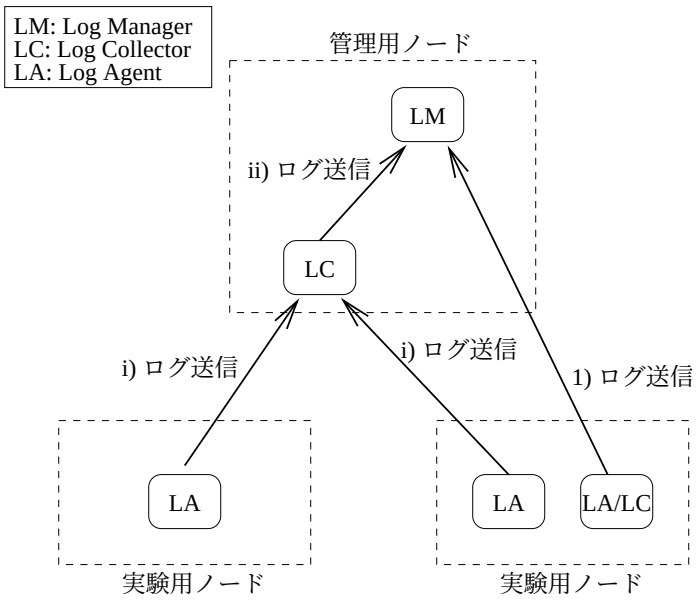

図 9.6: ログ収集の手順

LA が出力したログを LC が回収し、さらに LC から全体のログを LM に集める。LC は実験用ノードもしくは管理用ノードで動作する。LC が実験用ノードで動作する一例として、LA が実験用ノード上にログファイルを出力する場合がある。この時、LA が LC の機能を持っているといえる。一方、管理側ノードで LC が動作する場合があり、この時 LC は複数の実験用ノードのログを収集する。ログ情報は最終的には LM に送信される。

ノードを利用したノード初期化のためだけの実験を行う。

9.2.11 実験実行者による手動実験制御

ES はその時点で実行されている実験の EM が動作しているノードに関する情報を保持しており、実験実行者は EM に対して、実験の識別子や実験実行者名をキーに問い合わせを行うことで、EM へのアクセス方法を確認する。EM は、所属する実験駆動単位の制御モジュールの動作情報を保持しており、実験実行者から実験を実行するためのモジュール群を隠蔽する役割を持つ。EM は、実験を制御するためのさまざまなユーザーインターフェースを実験実行者に対して提供し、実験実行者からの要求にしたがい、そのほかのモジュールと協調することで実験を制御する。

9.2.12 資源利用の排他処理

資源利用の排他処理はさまざまなモジュールで対応が必要である。すでに述べたとおり、これまでに提案した各種モジュールでは、調停機構を用意している。

9.2.13 実験駆動単位の制御

実験駆動単位が生成されると、その後は SCD が中心となって実験を制御する。実験全体をシナリオ実行とみなし、ノードの割り当て要求や、ノードへのソフトウェア導入などもすべてシナリオの一部として実行するためである。これにより、実験中のノード割り当て要求や、ノードへのソフトウェア導入なども自由に行える。

9.2.14 実験状態の保存と復元

実ノードを利用した大規模実証環境では、ハードディスクのイメージをバイナリとして保存し、それを書き出すことで、ハードディスクの状態は保存できる。これらはノードへ導入するディスクイメージの作成と同様の手順で行える。しかし、ネットワーク状況やメモリ使用状態を保存することは困難である。一方 VM Nebula

ではVMwareの機能を利用し、仮想機械のディスクイメージを保存することにより、ディスクの内容、プロセスの起動状態、メモリの内容などを保存できる。しかし、仮想機械はノードを模倣する機構であるため、ネットワーク上の情報は管理できない。また保存したディスクイメージをハードディスクに書き込むことで、その復元も可能である。

実験状態の保存については、大規模実証環境の実現方法により、その可能性および実現方法が異なる。

9.2.15 複数の大規模実証環境の協調

ある大規模実証環境のESが設定記述を読み込み、他の大規模実証環境を利用するための記述があった場合には、他の大規模実証環境の制御モジュールと協調した動作を行い、実験を行う。

図9.7に協調の手順を示す。設定記述が入力された大規模実証環境のESは、ローカルのDRMに資源を要求する。DRMは、他の大規模実証環境のDRMと協調し、実験実行者の要求を満たす大規模実証環境の選択を行う。その後、ESは利用する大規模実証環境のESにEMの起動要求を行い、各大規模実証環境上にEMを起動する。その後は、それぞれのEMが協調することで、複数の大規模実証環境にまたがる実験駆動単位を形成する。各大規模実証環境上の制御モジュールはそれぞれの大規模実証環境上で動作するEMおよびSCDからの指示により動作する。各大規模実証環境のEMおよびSCD同士が協調することで、実験実行者からは複数の大規模実証環境に資源が分散していることを意識することなく実験が実行される。

9.2.16 例外処理

実験中の例外はすべての制御モジュールおよび、実験用ノードで発生する。このような例外はEMに通知されることとし、実験実行者は設定記述に例外の種類とそれに対しての処理を記述する。例外に対する基本的な処理としては、実験の停止と障害を無視した実験継続が考えられ、実験の停止の場合は、実験実行者が

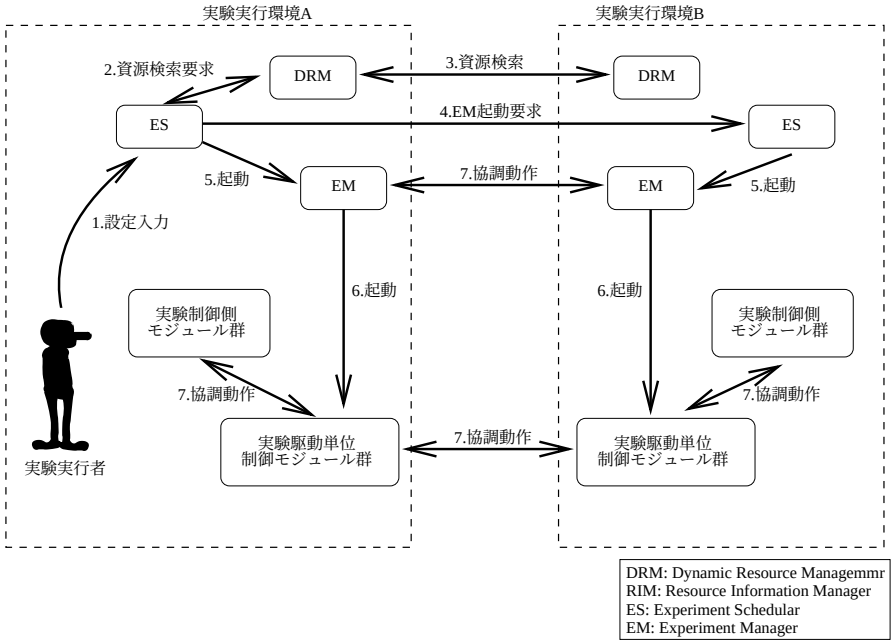

図 9.7: 複数の大規模実証環境の協調手順

複数の大規模実証環境にまたがった実験駆動単位を構築する際の動作手順。(1) 実験実行者がある大規模実証環境の ES に設定記述を入力すると、(2) ES はその大規模実証環境の DRM に資源の検索要求を行う。(3) DRM は、他の利用できる大規模実証環境の DRM と協調し要求を満たす資源を検索し、利用できる資源があれば割り当てる。(4) ES は割り当てを受けると、その資源が存在する大規模実証環境の ES に EM の起動要求を行う。(5) それぞれの大規模実証環境の ES は DRM から ES 用の資源の割り当てを受け、EM を起動する。(6) EM は DRM から実験駆動単位の制御モジュール用の資源の割り当てを受け、それらを起動する。(7) その後、それぞれの大規模実証環境の制御モジュールと ES が協調することで実験駆動単位を構築し実験を行う。

次の指示を行うまで実験駆動単位をその状態で保存する場合と、自動的に初期状態に戻すなどの処理も考えられる。

ノードやリンクの状態はSM、HCなどから、RIMおよびTMに通知されるため、これらの機構が不正状態を検知しEMに通知する。この通知を受けたEMはSCDと協調することで対応する例外処理を行い、実験実行者へ例外内容およびそれに対する処理を通知する。また、実験実行者によりノードの故障などが検出された場合は、大規模実証環境の管理者に障害情報が通知されなければならない。このような情報はDRMに送信され管理される。

実験駆動単位を構築する以前には、EM起動のための資源が確保できないなどの例外が考えられるが、このときはESから実験実行者に例外が通知される。

9.3 考 察

大規模実証環境で実験を実行するにあたり、実験実行者が必要とする機能を整理し、それを満たす汎用アーキテクチャを設計した。

本節では、このアーキテクチャについていくつかの面から考察を行う。

9.3.1 汎 用 性

これまでのStarBEDとSpringOSの開発・運用経験と、大規模実証環境やソフトウェアシミュレーションでの実験実行者へのインタビューから、大規模実証環境への必要機能を9.1節にまとめ、この要求を満たすことのできる支援ソフトウェアのアーキテクチャの一例を9.2節で提案した。インタビューの対象とした実験は複数の組織で行われた、全く異なる目的のものであるため、多くの種類の実験に対応できると考えられる。また、3章で述べたように、ある特別な目的を持つ大規模実証環境の機能を引き出すためには、専用の支援ソフトウェアが必要であるが、実験駆動単位を構築するための基本となる機能は大規模実証環境により大きく変わることはない。したがって、本アーキテクチャを用いることで、大規模実証環境の基本的な機能を引き出し、実験を行うことが可能である。

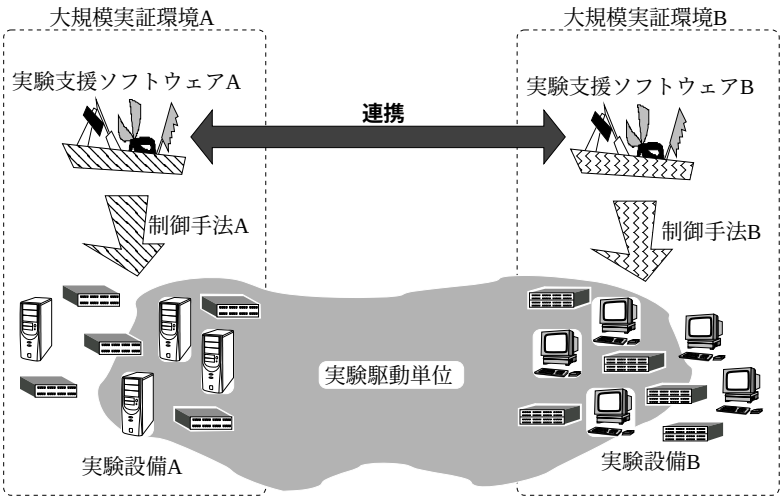

図 9.8: 大規模実証環境の協調の概念図

各実験設備の特別な機能を利用するためには、専用の支援ソフトウェアが必要であるため、各実験設備用の支援ソフトウェアの各種モジュールが協調して、一つの実験実行環境を構築する。

9.3.2 大規模実証環境の協調

現在、PlanetLab や Netbed では、さまざまな実験設備を独自の支援ソフトウェアで制御できるよう拡張が進められているが、実験設備および支援ソフトウェアの一般化の議論は進められていない。大規模実証環境に必要な機能を整理しモデル化することで、大規模実証環境の協調が可能になる。単一の支援ソフトウェアを構築すれば、すべての環境の機能を平均的に利用できるが、最大限活用することが難しい。たとえば、DETER[45][46] や SIOS、VM Nebula は、セキュリティ実験に特化した大規模実証環境であり、StarBED や Netbed、PlanetLab といった汎用的な大規模実証環境とは異なる操作が必要となる。このような実験設備上で動作する実験支援ソフトウェアは実験設備の性能を十分引き出すことができるように設計されるため、実験設備の機能を最大限に引き出すことができる。

本アーキテクチャでは実験実現のために基本的に必要となる機能を整理した。各大規模実証環境のこれらの機能を提供するモジュール群が協調することで、実験設備専用の操作を隠蔽し、複数の実験設備を利用した実験駆動単位を構築できる。図 9.8 に概念図を示す。

9.3.3 大規模実証環境の評価基準

本章で提案した一部の機能のみでも実験を実行することが可能であり、必ずしもすべての大規模実証環境が本章で述べた機能すべてに対応する必要はない。したがって、ある実験が実現可能である大規模実証環境とそうでない大規模実証環境がある。VM Nebula では前述のとおり実験シナリオの自動実行は実現しておらず、SpringOS も本章であげた機能のすべてを持っていないが、さまざまな実験に利用されてきた。このような環境でも大規模実証環境として十分な機能を持つといえる。

本章で整理した、大規模実証環境に必要な機能を、各大規模実証環境が持っているかを確認することで、ある実験もしくは実験実行者の要望に適応できるかという一つの評価基準となる。

第10章

実験内容の決定支援手法の提案

実験実行者は、実験の実行前に実験に利用するトポロジや検証項目を決定し、目的にそった環境を構築できる実験実行環境を選択する必要がある。これらの検討が十分で無い場合は、実験の正当性に問題が生じる場合がある。このような問題を解決するため、実験の性質などを再整理し、実験内容の決定支援のための議論を行う。適切な実験実行環境や実験駆動単位のトポロジは、実験内容を検討することである程度提案できる。本章では、実験の目的と、実験実行環境の性質、そしてトポロジの性質を分析・整理し、実験の目的に即した機能を提供できる実験実行環境および、適切なトポロジの決定支援を行うための議論を行う。図 10.1 に提案の概念図を示す。

10.1　実験実行環境の決定支援

本節では、実験の内容を検討することにより実験実行環境を一意に決定する手法について議論する。

10.1.1　実験の目的

実験を行う際にまず考慮すべき点は、実験の目的が対象技術の論理的検証であるか、その実装の実践的検証であるかという点である。論理的検証と実践的検証の大きな違いは、外乱がない理想的な環境で理想的な挙動を示す要素を対象とし

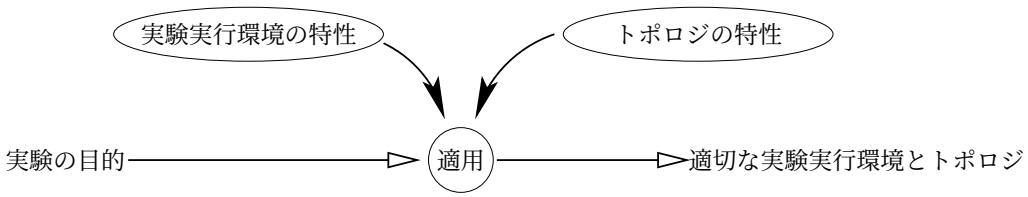

図 10.1: 適切な実験実行環境およびトポロジ提案

実験実行環境の性質とトポロジの性質を考察し、実験の目的から要求される事項に適用することで、適切な実験実行環境とトポロジを提案する。

て実験を行うのか、それとも、実環境に導入されるべき実装を利用し、バグや外乱も含めた実験を行うかである。外乱の無い理想的な実験駆動単位を構築するためには、実験駆動単位を構成する各要素をモデル化および抽象化し、実験に必要ないと思われる挙動を隠蔽する必要がある。

基本的にソフトウェアシミュレーションで実現される要素はモデル化・抽象化されたものであるため、論理的検証を行えるが、実践的検証には不向きである。また、シナリオの進行についてもまえもって用意した設定記述どおりに実験が実行される点と、要素の挙動を正確に制御できる点から、論理的検証を行う上で優位である。

一方、実証環境および大規模実証環境では、実環境向けの実装を利用できるため、実践的検証に向いているが、ノードのメモリ状況や他のプロセスの稼働状況などが、外乱として加わるため完全に外乱をなくすことは不可能であり、論理的検証は困難である。したがって、論理的検証を行う場合は、ソフトウェアシミュレータが適しているといえ、実践的検証には、実証環境もしくは大規模実証環境が適している。以上により、対象の実験が論理的検証であるか、実践的検証であるかにより、利用できる実験実行環境が絞り込める。

10.1.2　モデル化・抽象化の可否に従属する性質

前節で述べたとおり、ある実験実行環境が論理的検証に向いているか、実践的検証に向いているかは 4.2 節で述べたモデル化・抽象化の可否に関連する。

4.2 節でまとめた性質を、モデル化・抽象化の可否という視点から見直すと、多くの性質はモデル化・抽象化に付随する性質であるといえる。以下にモデル化・抽象化に付属する性質を上げる。

- 実環境との同一性

  実時間性 モデル化された環境では、かならずしも実時間でシナリオが進行しない。環境内で統一された時間軸上で矛盾無くイベントが起きればよい。また、時間の流れが均一でない場合も多く、イベントが発生しない期間は実行されないこともある。一方、モデル化されていない環境では実時間でシナリオが進行する。

  実験駆動単位の構成要素の挙動 モデル化・抽象化を行うことは、実環境で動作する要素の実験に不必要な挙動を模倣せず、必要な挙動だけを模倣することである。模倣対象である実環境で動作する要素と比較すると、モデル化・抽象化された要素と実環境で動作する要素の同一性はある程度失われている。

- 実行容易性

  実験時間 モデル化された実験実行環境では、時間の流れが均一でない場合が多く、このような環境では複雑な処理が発生する時間帯や、多くのイベントが同時に発生する時間帯では、計算量が大きくなり時間の流れが遅くなるといえる。

- 再現性と検証容易性

  実験駆動単位の構成要素の挙動 モデル化された環境では、外乱が存在せず、それぞれの要素の挙動は常に同一であるため、同じ実験を複数回行ってもその結果は同一である。しかし、実ノードを利用した環境では、ノー

ドやリンク自体の状況によりその挙動が少なからず異なるため、実験を行うごとにその結果に差異がある。

モデル化・抽象化の可否に付随する属性を除くと、残りの属性は以下である。

- 実環境との同一性
  - 規模追従性
- 実験実行の容易性
  - 準備時間
  - パラメータの変更
- 再現性・検証容易性
  - 構成要素の挙動
- 身近さ

大規模なネットワークなどを単純化することで、見掛けの環境を大規模化できるため、規模追従性は、モデル化・抽象化の可否に影響を受ける性質といえる。しかし、大規模実証環境のように、モデル化・抽象化を行わず、多数のノードを実際に用意することで大規模化することも可能である。したがって、モデル化・抽象化にのみ影響される性質ではなく、用意できるノード数や実験支援ソフトウェアのような管理機構の存在に関連するといえる。

パラメータの変更容易性や構成要素の挙動の再現性や検証容易性は、実験駆動単位もしくは実験実行環境の管理機構が存在するかに関連する。

準備時間に関しては、それぞれの実験実行環境により準備の手順が異なり、それを一概に比較することはできないため、本論文では言及しない。

身近さについても、実験実行者により異なり、一概に論ずることはできない。

以上のようにモデル化・抽象化の可否と、管理機構の存在に関連する性質が多いため、4.2 節で述べた各実験実行環境の性質の関連を、これらの性質に着目し、表 10.1 に再整理する。以下の節では、モデル化・抽象化の可否、管理機構の存在、そして規模追従性に着目して議論する。

表 10.1: 各実験実行環境の性質の関連性

	モデル化・抽象化			管理機構の存在				
	実時間性	構成要素の挙動（実行容易性）	実験時間	パラメータの変更	操作履歴	イベント発生の正確性	構成要素の挙動（再現・検証）	規模追従性
ソフトウェアシミュレータ	×	×	△	○	○	○	○	△
実証環境	○	○	△	×	×	×	×	×
大規模実証環境	○	○	△	○	○	○	△	○

10.1.3 規模追従性

論理的検証であるか実践的検証であるかについで検証すべき点は、実験駆動単位の規模もしくは要素数である。すでに述べたようにソフトウェアシミュレータでは、制御対象の要素数が多い場合や、要素の関連が複雑に連携する場合は、計算に必要となるリソースが大きくなり、現実的な時間で実験が終了しないため、このような実験には向かない。実証環境は、用意できるノード数の問題と支援ソフトウェアが存在しない点から、要素数が多い実験には向かない。一方、大規模実証環境では、存在するノード数の規模までの実験を行える。図 10.2 に実験駆動単位を構成する要素数に対して、対応できる実験実行環境を示す。ソフトウェアシミュレータでは、高度なモデル化が行える実験であれば非常に巨大なネットワークを模倣できる。しかし、現在の大規模実証環境が提供できる程度のノード数で構成され、それらのノードがある程度協調する実験では、非常に長い時間がかかることが多く報告されている。したがって、本論文では、規模追従性に関して、大規模実証環境がソフトウェアシミュレータより優位であると結論づけた。

10.1.4 管理機構の存在

大規模実証環境およびソフトウェアシミュレータでは、管理機構が用意されており、実験実行環境および実験駆動単位が管理している。このような環境では、実

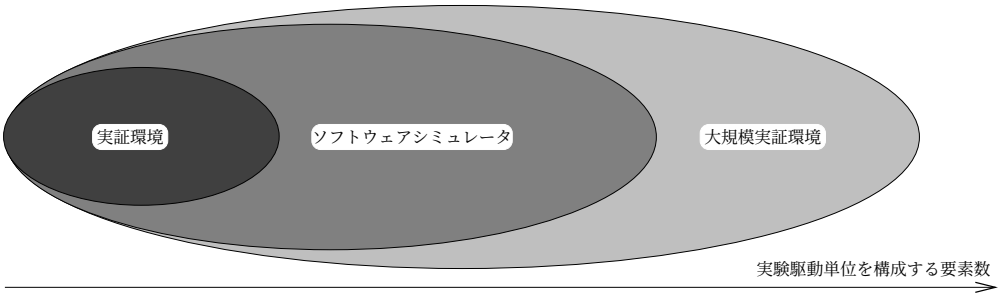

図 10.2: 実験実行環境と実験駆動単位の要素数の関係

実証環境は実ノードの用意やその設定のコストから大規模な実験には向かない。一方、大規模実証環境にはまえもって用意された実験用の実ノードと、それらを制御する実験支援ソフトウェアが用意されているため、用意されている実ノードの要素数もしくは、実ノード上でプロセスやスレッド、仮想ノードを動作させることで大規模な実験を実行できる。ソフトウェアシミュレータは設定コストは小さいが、現状の大規模実証環境で提供できる程度の規模の実験駆動単位での実験では、実験が終わるまでに非常に時間がかかってしまうため、大規模実証環境とソフトウェアシミュレータの中間に位置する。

験実行環境で想定されている範囲での実験しか行えないという欠点がある。たとえば SpringOS を用いて StarBED で実験を行う際には、VLAN を用いて実験ネットワークを構築するため、VLAN よりも下の層の技術の検証は困難である。また、VM Nebula では VMware で実現される仮想ノードを利用するため、ネットワークインターフェースや CPU などは、VMware が模倣できるもののみしか利用できない。Netbed や PlanetLab を利用した場合は利用できる OS に制限がある。このような制限を回避するためには、管理機構が存在しない実証環境を利用する必要がある。

10.1.5 実験実行環境の決定

以上により、実験実行環境を決定するためには、以下の点を考慮する必要がある。

1. 論理的検証であるか実践的検証であるか
2. 実験駆動単位を構成する要素の数もしくは複雑さ

表 10.2: 実験の種類と規模から導かれる実験実行環境

実験の種類	実験要素数	管理機構の実験への対応	対応する実験実行環境
論理的実験	-	-	ソフトウェアシミュレータ
実践的実験	少ない	できない	実証環境
		できる	大規模実証環境
	多い	できる	大規模実証環境

3. 管理機構による実験への制限

以上の項目を検討することで、利用できる実験実行環境が決定可能である。これらの項目と実験実行環境の関係を表 10.2 に示す。

論理的検証であるのか実践的検証であるのかは、開発の段階などから決定される。また、実験駆動単位の規模もしくは模倣対象の要素数についても、開発の段階が参考になる。最低限の系での動作試験および性能試験は、大規模な実験駆動単位が必要ない場合が多いため、すべての実験実行環境で対応が可能である。その後の実環境を考慮した環境では、実験駆動単位を構成する要素数が多数になることが多く、大規模実証環境を利用する必要がある。身近さは計画した実験を実行するための環境が用意できるかという点で重要な要素である。大規模実証環境は利用できる環境の数の問題などから、身近さが低いといえ、ソフトウェアシミュレータおよび実証環境を利用するための機材などは一般的なものであるため、身近さが高い。それぞれの実験実行環境でしか検証が不可能である技術や、現象も多く存在すると考えられるため、その時点でどの程度まで検証が必要なのか、また、どの実験実行環境を用いなければならないのか、そして、利用できなかった場合はどの点が検証できないのかなどについては熟慮が必要である。

10.2 実験トポロジの決定支援

実験に利用されるトポロジは、当然、その目的により使い分けられる。ネットワークソフトウェア開発の最終段階では、対象となる実環境を正確に模倣するこ

とが重要となる。しかし、開発の初期段階では、必要最低限の機能を実現できるトポロジで十分であることはすでに述べた。本節では、基本的なトポロジとより現実的なトポロジの決定支援について述べる。

10.2.1 単純なトポロジ

最低限の系での動作試験や性能試験では、対象の技術が動作するノードもしくは、対象とするリンクをシングルボトルネックとするトポロジが利用されることが多い。シングルボトルネックとなるトポロジとして、ダンベル（図 10.3）とスター（図 10.4）がある。リンク自体が観測対象であったり、リンクに流れるトラフィックに対して何らかの制御を行う技術の検証には、ダンベル型トポロジが利用できる。また、ノード自体もしくは、ノード上で動作するアプリケーションソフトウェアの検証を行うためには、スター型トポロジが利用される。

何らかの情報を複数のノードに伝搬していく様子などをの手順検証のためには、デイジーチェーン型トポロジ（図 10.5）が、また、階層構造を持った技術の検証にはツリー型トポロジ（図 10.6）が利用できる。

最低限の系での動作試験では、これらのトポロジを用いて、対象技術が動作する最低限のノード構成で実験を行う。性能試験では、これらのトポロジを用い、対象技術自体もしくは、対象技術が制御する要素に対して負荷を増加させるための要素を追加して実験を行うのが一般的である。

10.2.2 より現実的なトポロジ

対象の実環境への影響を調べる段階では、どのような要素が対象技術に影響をおよぼすのか、また、逆におよぼされるのかを確認するため、対象環境の再現性が重要になる。しかし、技術の導入対象のトポロジは非常に幅広いものとなり一般化は難しい。しかし、実環境のトポロジをある程度抽象化することで、利用することは可能である。

実際の実環境のトポロジを調べ、その環境を作り出す手法がある。インターネット上で ISP などの接続情報を調査するための技術は数多く発表されている [47][48]。AS 単位での実験を行う際には、実際の AS のトポロジを参考に、上位数十から

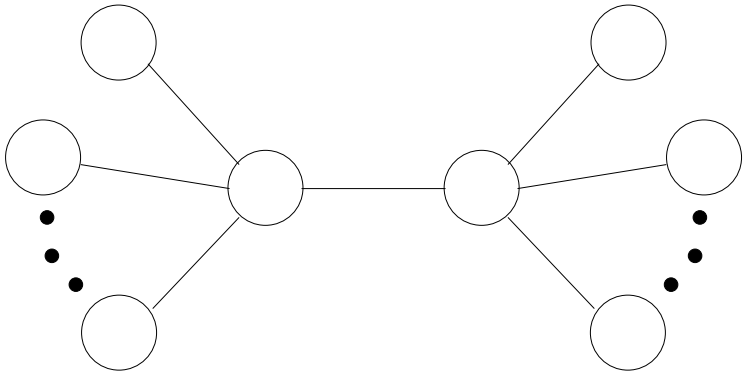

図 10.3: ダンベル型トポロジ

中央のリンクがシングルボトルネックとなる。このリンクに負荷をかけての実験や、このリンクの両側のノードで対象技術を動作させる場合などに利用できる。

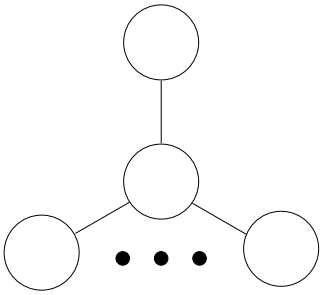

図 10.4: スター型トポロジ

中央のノードがシングルボトルネックとなる。このノードのハードウェアもしくはその上で動作するソフトウェアへ負荷をかけての試験などに利用できる。

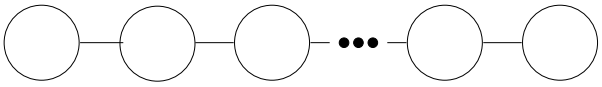

図 10.5: デイジーチェイン型トポロジ

片端のノードから逆端のノードへ、何台かのノードを経由して通信を行う技術の実験に利用できる。

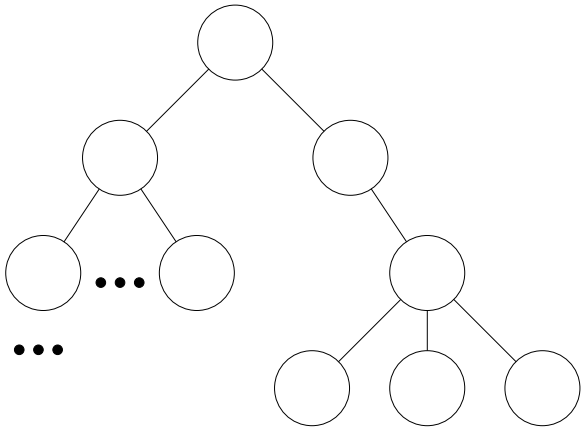

図 10.6: ツリー型トポロジ

階層構造を持つ技術の動作検証などの検証に利用できる。

数百といった AS で構成されるトポロジが利用できる。このような情報として、Cooperative Association for Internet Data Analysis （CAIDA）[49] が採取、分析している情報 [50] や、Rocketfuel[51] を利用して採取した ISP の接続情報 [52] などを利用できる。また、対象とする実環境のトポロジを確認できる場合は、その情報にもとづいた実験トポロジを構築できる。一方、モデル化・抽象化・成長モデルにより一般的なトポロジを構築する手法 [20][53][54][55] なども研究されている。文献 [19] では、成長モデルによるトポロジ生成の一例として、DNS のような複数の同じ機能を持つサーバが存在する際に、クライアントがサーバを選択するアルゴリズムの提案を行った。このアルゴリズムの動作検証のため、現実的な実験用トポロジをスケールフリーモデルを応用したトポロジの成長ルールを提案し、これにしたがってトポロジを構築した。以下に提案した成長ルールを示す。

- 1 台目のノードを設置する。
- 2 台目以降のノードは生成時に既存のノードを 1 台選択し、そのノードに接続する。新たなノードが接続されるノードは、ノードが持つリンク数に比例した確率で選ばれる。リンクを多く持つノードはより高い確率でさらに多くのリンクを得る。
- ノードを 10 台設置するごとにサーバを設置する。サーバはその時点でリン

クを最も多く持つノードにのみ接続される。選択したノードにすでにサーバが接続されていた場合は、次にリンクが多いノードに接続する。

- あるサーバを利用しているクライアント数が 20 を超えると、そのサーバが接続されていたノードとサーバを 2 つに分割し、ノードに接続されていたリンクは 2 つのサーバに接続しなおされる。

- ノードを 100 台設置するごとに、その時点でのリンク数に比例した確率で 2 台のノードを選択し、その 2 台を接続するリンクを生成することでループを作る。選択された 2 台がすでに接続されていた場合は、改めて選択しなおす。

以上のルールで 500 台のノードを設置してトポロジを生成した。最終的には、ノードの分割などによりノード数 510、サーバ数 60 のトポロジが構築された。これにより構築されたトポロジはすでに紹介したとおりである（図 4.5）。このように、既存のトポロジ成長モデルや実環境自体のトポロジの性質と、実験の目的や対象とする実験トポロジと対応づけることで、実験に利用できるトポロジを決定できる。

10.3 考 察

本章では、適切な実験実行環境および、初期段階の論理的・実践的検証の単純なトポロジ提案を行う手法を述べた。

実験実行環境は、対象とする実験が、論理的検証であるか実践的検証であるか、実験駆動単位の要素数とその複雑さ、そして管理機構による制限を考慮することにより決定できた。

実験トポロジに関しては、最低限の系での動作や性能試験に関しては、単純なトポロジを利用することで実現できる。しかし、実環境を考慮した実験で利用できるトポロジに関しては、対象とする技術によりさまざまである。典型的な ISP や企業内ネットワーク、P2P ネットワークなどのモデル化や、トポロジ生成アルゴリズムの提案、また、実環境のトポロジの調査などネットワークトポロジに関するさまざまな視点からの研究がなされている。このような研究の成果により得

られるトポロジと、実験の目的や対象とする実環境との関連性を議論することで、適切な実験トポロジを提案できる可能性がある。

また、実験駆動単位をより実環境に近づけるためには、実験駆動単位に導入すべきバックグラウンドトラフィックや、対象技術の利用者の挙動に起因するイベントやトラフィックの生成なども必要となる。さらに、実験データを収集するため、実験の項目や実験データの観測場所などについても決定の支援ができれば、より精度が高い実験が実現できる。

第11章

結 論

本研究では、まず、ネットワーク実験の性質と現在利用されている実験の実現手法および、実験の支援の可能性について整理し、実験を実際に実行する手順に着目した実験実行者の要望をまとめた（第 I 部)。さらに、実験実行環境のひとつである大規模実証環境に着目し、その実現方法およびその上での実験支援のための要件の整理を行い、その一実装である、StarBED および SpringOS の詳細について述べた（第 II 部)。これらの議論および経験をもとに、大規模実証環境に必要な機能群を整理し、実験支援ソフトウェアの汎用アーキテクチャの提案と、実験内容の決定支援手法について述べた（第 III 部)。

11.1 本研究の成果

第 I 部では、ネットワーク実験の性質および既存の実験実行環境の性質の整理（2 章）と、実験の手順の定義とそれぞれの手順に必要となる入力と出力についてまとめ（3 章)、実験実行者が実験に求める性能と機能を整理（4 章）した。

ネットワーク実験に関する研究のためには、ネットワーク実験自体およびネットワーク実験の実現手法についての知識は不可欠である。はじめに、ネットワーク実験の性質と、その実現方法について検討し、実環境に近い環境を構築するため、実環境とそれを模倣した実験駆動単位の挙動の差異を明らかした。また、容易かつ精度の高い実験を行うためには、人の手により実験を制御するのではなく、機械的に実現することが望ましい。このために、実験の手順を整理し、それらの

手順での入力および出力と、出力を生成するための手法について検討を行った。

第 II 部においては、大規模実証環境に要求される要件をまとめた（5 章）上で、これまで運用・開発してきた大規模実証環境を構成する StarBED と SpringOS について述べ（6 章、7 章）、これらの評価を行った（8 章）。

多数の実ノードを用いた実験の実行は 2 章で述べた手法のうちもっとも実行コストが大きい。現在の StarBED は 680 台の PC ノードとそれらを接続するスイッチノードで構成されており大規模な実験駆動単位を構築できるが、実験駆動単位を構成するためのノードの検索、ノードへのソフトウェアの導入、実験トポロジの構築、シナリオの実行を手動で行うことは非常にコストが高い作業である。StarBED で実験を行うための作業を整理し、前述の作業を自動化する SpringOS を設計・実装することで、実験実行者のコストを低減するだけでなく、StarBED を効率的に利用できるようになった。また、支援ソフトウェアの充実により、正確な実験結果が得られるようになった。これらの議論により、大規模実証環境の一実装での実験実行コストを大幅に低減できたことを確認した。

第 III 部では、第 II 部での経験をふまえ、大規模実証環境に一般的に必要とされる機能の整理と、それを実現する実験支援ソフトウェアのアーキテクチャ設計を行い（9 章）、複数の実験設備の協調の可能性を示した。また、適切な実験内容の決定支援について議論した（10 章）。

VM Nebula や SIOS、Netbed、PlanetLab、StarBED などの大規模な実験設備はそれぞれ異なる特性を持つため、複数の大規模な実験設備を統合的に利用し、一つの実験駆動単位中の適切な位置に適切な実験設備を割り当てることで、実験実行者が求める環境をより柔軟に構築できる可能性がある。しかし、これらの実験設備の特性を引き出すためには、専用の支援ソフトウェアが必要である。このような専用の支援ソフトウェアを協調させることで、複数の実験設備を利用した実験駆動単位を構築できる。実験実行のために大規模実証環境に必要となる機能を整理するとともに、これらの機能を実現する支援ソフトウェアのアーキテクチャを提案した。

実験の目的はさまざまであり、実験の目的により、対応できる実験実行環境は異なる。実証環境や大規模実証環境では、実環境向けの実装を利用しての実践的検証を行える。一方、アルゴリズム自体やプロトコルの挙動の論理的検証を行い

たい場合は、ソフトウェアシミュレーションを利用できる。ただし、大規模実証環境では、実験支援ソフトウェアや運用方針からの制限により、実験実行者が求める実験駆動単位を構築できない場合がある。このような場合や、身近に存在しない、予約がとれないといった理由から、大規模実証環境を利用できない場合は実証環境を利用する。実験トポロジに関しては、基礎的かつ単純なトポロジの性質の整理および実験の目的との関係を整理し、実環境のトポロジの利用や、既存のトポロジのモデリング技術および、トポロジ成長アルゴリズムの利用可能性を示唆した。

実験支援は実験内容の検討・決定、実験駆動単位の構築・実験実行そして、実験データの解析の各手順で必要だとした（3 章)。本論文では実験内容の検討・決定手順を支援として、実験実行環境の提案手法および、単純な実験に関するトポロジの提案と、現実的なトポロジを決定する際に利用できる可能性のある既存技術や情報について述べた。実験駆動単位の構築・実験実行手順に関しては、ソフトウェアシミュレーションにおいては、汎用ソフトウェアシミュレータがこの部分の支援を行っているといえるが、実ノードを用いた環境での実験支援の議論が不十分である。これを補うため、大規模実証環境の一実装である StarBED および SpringOS を実現し、その知見から実験支援ソフトウェアの汎用アーキテクチャを提案している。

11.2　本研究分野の課題と展望

3 章で述べたとおり、実験の手順は 3 つにわかれており、それぞれの出力が次の手順の入力となる。したがって、実験実行者が対象技術に関する情報と、検証対象の実装そのものを提示すれば、前述の 3 手順が自動的に実行され実験実行者は、妥当な実験結果のみを容易に得ることができる。

このためには、実験内容の検討・決定支援手順では、本研究で提案した実験実行環境の決定と単純なトポロジだけでなく、実験駆動単位の規模や導入されるべきバックグラウンドトラフィック、実験項目、各データの検証位置などを一意に決定する必要がある。これらは、3 章で挙げた設計情報や仕様、実験の段階や目的などから決定できる可能性がある。

また、実験駆動単位の構築および実験実行手順では、より精度の高い実験を可能とするため、大規模実証環境での精度を保証するための仕組みが必要である。そのためには、実験実行環境の各要素の性質や性能の評価方法や、性能の保証に関する問題を解決する必要がある。

そして、柔軟な実験駆動単位を構築するためには、本論文で提案した支援ソフトウェアのアーキテクチャの各機能を、それぞれの実験設備に対して具体化し実装する必要がある。実験データの解析手順については本論文では触れていないが、汎用的なデータの可視化手法や、集計方法が求められる。これらに加えて、各手順が正当なものであったかどうかを検証する機構が必須となる。

これらの手順を高度化していくことにより、実験の自動実行だけでなく、統一的な実験結果にしたがった技術の比較が可能となる。さらに、ある技術の品質を保証する機構などの実現も期待できる。これらは、ネットワークサービスの品質向上だけでなく、製品の導入支援に対する貢献も期待できる。

謝辞

本研究を行うにあたり、終始御指導を賜りました篠田 陽一 教授に深謝致します。本論文の審査委員をお引き受けいただきました、本学 丹 康雄 准教授、敷田 幹文 准教授、および、IIJ 技術研究所/本学 客員教授 長 健二朗 氏、奈良先端科学技術大学院大学 門林 雄基 助教授には、本論文を完成させるにあたりさまざまな御助言を頂きました。深く感謝致します。

本学 情報科学研究科 知念 賢一 助手、情報科学センター 宇多 仁 助手、独立行政法人 情報通信研究機構 情報通信セキュリティ研究センター 三輪 信介 研究員には、さまざまな場面での議論・示唆・協力を頂いたことを感謝致します。

WIDE プロジェクトのみなさまには、本研究を進める上でさまざまな有益なご意見をいただきましたことを感謝します。特に DeepSpaceOne ワーキンググループにおきまして、さまざまな議論をさせていただきましたことを感謝致します。

StarBED は 2002 年に通信・放送機構（TAO） 北陸 IT 研究開発支援センターに設置され、現在は、通信総合研究所（CRL）と通信・放送機構が統合し発足した独立行政法人情報通信研究機構（NICT）により、北陸リサーチセンターとして運営されています。本研究の一部である、大規模実証環境の運用・開発については、StarBED の設置無くしては成し得ませんでした。関係者各位に感謝します。特に、北陸 IT 研究開発支援センター/北陸リサーチセンターのスタッフとしてさまざまな方面から支援してくださいました、梅田 章 氏、佐野 正行 氏、小嶋 正博 氏、新舛 浩基 氏、竹中 ゆかり 氏には記して感謝の意を表します。

StarBED および SpringOS は実験実行者のための道具として開発・運用してきたものであり、このような道具は実際に利用されてはじめて価値を発揮するものです。StarBED および SpringOS を利用して実行された実験の結果や、道具としての使い勝手などについてのご感想、ご意見をフィードバックとしていただいたことにより、StarBED および SpringOS が道具としての成長できただけではなく、

研究としての面からもさまざまな考察を行うことができました。StarBED および SpringOS を利用してくださったみなさまなくしては、本研究は成り立ちませんでした。利用者のみなさまに心から感謝致します。特に、松下電器産業株式会社 ネットワーク開発センター IP アクセス方式グループアクセス第一チーム 村本 衛一 主任技師には、利用者の視点からさまざまなアドバイスをいただいただけでなく、本論文で紹介した実験事例に関する多くの情報をいただきました。ここに記して感謝します。

学業のみならず私生活の面からもサポートして頂いた本学 篠田研究室ならびにインターネット研究センターの諸氏に感謝します。

私生活の面においては、空想居酒屋いつもや 塚田 郁生・麗子 夫妻、ならびに常連客のみなさまに、常に励ましていただきましたことを感謝致します。

本論文の題字は父 宮地 柊石（繁）によるものです。多忙な中、私のわがままを聞き入れ、執筆してくださいましたことにつきましても深く感謝致します。

最後に、すべての面においてあたたかくサポートして頂いた両親には、本論文の完成をもって謝意をあらわすとともに、最大限の感謝をもちまして厚く御礼申し上げます。

平成 19 年 3 月

宮地 利幸

参考文献

[1] The VINT Project. *The ns Manual*. http://www.isi.edu/nsnam/ns/ns-documentation.html.

[2] OPNET Technologies, Inc. *OPNET User Forums*. http://forums.opnet.com/.

[3] *SSFnet network simulator*. http://www.ssfnet.org/.

[4] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. pp. 255–270, Boston, MA, December 2002. USENIXASSOC.

[5] Planetlab website. http://www.planet-lab.org/.

[6] 三輪 信介, 滝澤 修, 大野 浩之. 仮想 PC インターネットセキュリティ実験環境『VM Nebula』の設計と構築. 暗号と情報セキュリティシンポジウム（SCIS）. 電子情報通信学会, 2003 年 1 月.

[7] 大野 浩之, 武智 洋, 永島 秀己. インターネットの脅威に対抗しうる脆弱性データベースと検証システムの構築. 分散システム／インターネット運用技術 （DSM） シンポジウム. 情報処理学会, 2001 年 2 月.

[8] IEEE standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. December 1998.

[9] The VINT Project. *Network Emulation with the NS Simulator*. http://www.isi.edu/nsnam/ns/ns-emulation.html.

[10] VMware. http://www.vmware.com/.

[11] Mio Suzuki, Hiroaki Hazeyama and Youki Kadobayashi. Expediting experiments across testbeds with AnyBed: a testbed-independent topology configuration tool. Proceedings of 2nd International IEEE/Create-Net Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities （TridentCom）, 2006 年 3 月.

[12] S. Crocker. Host Software, RFC1. April 1969.

[13] The iperf TCP/UDP Bandwidth Measurement Tool. http://dast.nlanr.net/Projects/Iperf.

[14] The Public Netperf Homepage. http://www.netperf.org/.

[15] Joel Sommers and Paul Barford. Self-configuring network traffic generation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, October 2004.

[16] Spirent Communications. http://www.spirentcom.com/.

[17] Agilent N2X. http://advanced.comms.agilent.com/n2x/.

[18] 村本衛一, 米田孝宏, 鈴木史章, 鈴木良宏, 中村敦司. 送信者起動マルチキャストにおける輻輳制御方法の提案. インターネットコンファレンス 2003, 2003 年 10 月.

[19] Toshiyuki Miyachi, Kenjiro Cho, Yoichi Shinoda. On the Stability of Server Selection Algorithm against Network Fluctuations. LNCS 3837: Technologies for Advanced Heterogeneous Network（Proceeding of First Asian Internet Engineering Conference （AINTEC）, 2005 年 12 月.

[20] Albert-Laszlo Barabasi, Reka Albert and Hawoong Jeong. Mean-field theory for scale-free random networks. In *Physica A: Statistical Mechanics and its Applications*, Vol. 272, pp. 173–187, October 1999.

[21] 三角真, 中川晋一, 我如古津世史, 知念賢一, 篠田陽一. メガノード IP マルチキャストセンサネットに関する実験的検討. 信学技報 CQ2006016（2006-4）, pp. 75–81, 2006 年 4 月.

[22] 三輪信介, 宮地利幸, 大野浩之, 篠田陽一. 不正アクセス等再現実験環境の統合手法に関する研究. 暗号と情報セキュリティシンポジウム（SCIS2005）, pp. 1183–1188. 電子情報通信学会, 2005 年 1 月.

[23] 三輪信介, 宮地利幸, 大野浩之. 不正アクセス等再現実験環境の統合実験. マルチメディア，分散，協調とモバイル（DICOMO2005）シンポジウム論文集, pp. 393–396. 情報処理学会, 2005 年 7 月.

[24] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, Vol. 27, No. 1, pp. 31–41, 1997.

[25] NIST Internetworking Technology Group. *NIST Net network emulation package*. http://www-x.antd.nist.gov/nistnet/.

[26] Advanced Micro Devices, Inc. *Magic Packet Technology*, November 1995.

[27] Intel Corporation. *IPMI v2.0 specifications Document Revision 1.0*, February 2004.

[28] Hewlett-Packard Development Company. HP Integrated Lights-Out （iLO） Standard. http://h18000.www1.hp.com/products/servers/management/ilo/index.html.

[29] J.D. Case, M. Fedor, M.L. Schoffstall, and J. Davin. Simple Network Management Protocol (SNMP), RFC1157. May 1990.

[30] K. McCloghrie and F. Kastenholz. The Interfaces Group MIB, RFC2863. June 2000.

[31] D. Harrington, R. Presuhn, and B. Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC3411. December 2002.

[32] *WIDE PROJECT Home Page*. http://www.wide.ad.jp/.

[33] 情報通信研究機構. *JGN2 HomePage*. http://www.jgn.nict.go.jp/.

[34] Cisco Systems, Inc. http://www.cisco.com/.

[35] Foundry Networks. http://www.foundrynet.com/.

[36] Raritan, Inc. http://www.raritan.com/.

[37] Sun Microsystems. NFS: Network File System Protocol specification, RFC1094. March 1989.

[38] Intel Corporation. Preboot Execution Environment (PXE) Specifiction Version 2.1, 1990年sep月.

[39] J. Postel and J.K. Reynolds. File Transfer Protocol, RFC959. October 1985.

[40] The FreeBSD Documentation Project.
FreeBSD Handbook Chapter 4 Installing Applications: Packages and Ports.
http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/ports.html.

[41] gnuplot homepage. http://www.gnuplot.info/.

[42] W. Yeong, T. Howes, and S. Kille. X.500 Lightweight Directory Access Protocol, RFC1487. July 1993.

[43] J. Postel. Internet Control Message Protocol, RFC792. September 1981.

[44] C. Lonvick. The BSD Syslog Protocol, RFC3164. August 2001.

[45] DETER, "DETER Laboratory for Security Research".
http://www.isi.edu/deter/.

[46] Members of the DETER and EMIST Projects. CYBER DEFENCE TECHNOLOGY NETWORK AND EVALUATION – Creating an experiment infrastructure for develping next-generation information security technologies. In *COMMUNICATIONS OF THE ACM*, Vol. Vol.47 No.3., pp. 58–61, March 2004.

[47] k. claffy, T. E. Monk and D. McRobb. Internet tomography. In *Nature*, January 1999.

[48] Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, pp. 1371–1380, Tel Aviv, Israel, March 2000. IEEE.

[49] Cooperative Association for Internet Data Analysis （CAIDA）. http://www.caida.org/home/.

[50] CAIDA. *AS Relationships*. http://www.caida.org/data/active/as-relationships/.

[51] Neil Spring, Ratul Mahajan, David Wetherall. Measuring ISP Topologies with Rocketfuel. In *In Proceedings of ACM/SIGCOMM*, August 2002.

[52] *Rocketfuel maps and data*. http://cs.washington.edu/research/networking/rocketfuel/.

[53] S. Milgram. The small world problem. In *Psychology Today*, Vol. 2, pp. 60–67, 1967.

[54] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, Vol. 2, pp. 594–602, San Francisco, CA, March 1996. IEEE.

[55] BRITE: An Approach to Universal Topology Generation. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS 2001*, August 2001.

本研究に関する発表論文

[1] 宮地利幸:
“大規模ネットワーク実験環境におけるソフトウェアシミュレータの利用に関する研究”, 北陸先端科学技術大学院大学, 修士論文, 2002 年 3 月.

[2] 宮地利幸, 三輪信介, 知念賢一, 篠田陽一:
“ネットワーク実験支援ソフトウェアの汎用アーキテクチャの提案”, 情報処理学会論文誌, 48 巻 4 号, 特集号「ユーザ指向の分散システム／インターネットの運用・管理」. （掲載予定）

[3] Toshiyuki Miyachi, Junya Nakata, Razvan Beuran, Ken-ichi Chinen, Kenji Masui, Satoshi Uda, Yasuo Tan and Yoichi Shinoda:
“Realistic Simulation of Internet”, Systems Modeling and Simulation Theory and Applications, Asian Simulation Conference （ASC） 2006, Springer, ISBN 4-431-49021-3, pp. 386-390, Tokyo, Japan, October. 2006.

[4] 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, ACM Press, ISBN 1-59593-504-5, Pisa, Italy, October.2006.

[5] Toshiyuki Miyachi, Kenjiro Cho and Yoichi Shinoda:
“On the Stability of Server Selection Algorithm against Network Fluctuations”, LNCS 3837: Technologies for Advanced Heterogeneous Network （Proceeding of First Asian Internet Engineering Conference （AINTEC） 2005, pp. 210-224,Bangkok, Thailand, December. 2005.

[6] Toshiyuki Miyachi, Ken-ichi Chinen and Yoichi Shinoda:
"Automatic Configuration and Execution of Internet Experiments on an Actual Node-based Testbed", International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom) 2005, IEEE, ISBN 0-7695-2219-X, pp. 274-282, Trento, Italy, February. 2005.

[7] Razvan Beuran, Ken-ichi Chinen, Khin Thida Latt, Toshiyuki Miyachi, Junya Nakata, Lan Tguyen Nguyen, Yoichi Shinoda and Yasuo Tan:
"Application Performance Assessment on Wireless Ad Hoc Networks", Asian Internet Engineering Conference (AINTEC) 2006, Springer-Verlag LNCS 4311, pp. 128-138, Bangkok, Thailand, November. 2006.

[8] Razvan Beuran, Ken-ichi Chinen, Khin Thida Latt, Toshiyuki Miyachi, Junya Nakata, Lan Tguyen Nguyen, Yoichi Shinoda, Yasuo Tan, Satoshi Uda and Saber Zrelli:
"WLAN Emulation on StarBED", IET International Conference on Wireless, Mobile & Multimedia Networks (ICWMMN) 2006, pp. 856-859, Hangzhou, China, November. 2006.

[9] Ken-ichi Chinen, Toshiyuki Miyachi and Yoichi Shinoda:
"A Rendezvous in Network Experiment — Case Study of Kuroyuri", International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom) 2006, IEEE, ISBN 1-4244-0106-2, Barcelona, Spain, March. 2006.

[10] Eiichi Muramoto, Takahiro Yoneda, Atsushi Nakamura, Makoto Misumi, Toshiyuki Miyachi and Yoichi Shinoda:
"Report on a Method of Simulating Multicast Group Communication on the Internet", International Symposium on Towards Peta-Bit Ultra Networks (PBit) 2003, ISBN 4-9900330-3-5, pp. 182-188, Ishikawa, Japan, September. 2003.

[11] 宮地利幸, 知念賢一, 篠田陽一:
“ネットワーク支援システムに必要な機能と設定記述”, マルチメディア，分散，協調とモバイル シンポジウム （DICOMO） 2006, 論文集, 情報処理学会, pp. 769-772, 香川, 2006 年 7 月.（ヤングリサーチャ賞受賞）

[12] 宮地利幸, 知念賢一, 篠田陽一:
“StarBED における自動実験駆動機構”, 第 6 回インターネットテクノロジーワークショップ （WIT）2004 論文集, 日本ソフトウェア科学会 研究会資料シリーズ No.36 ISSN1341-870X, 日本ソフトウェア科学会 インターネットテクノロジ研究会, pp. 81-88, 石川, 2004 年 12 月.

[13] Razvan Beuran, Yoichi Shinoda, Shin-ichi Nakagawa, Junya Nakata, Toshiyuki Miyachi, Ken-ichi Chinen and Yasuo Tan:
“Performance Analysis of VoIP over WLAN”, Multimedia, Distributed, Cooperative and Mobile Symposium （DICOMO） 2006, IPSJ, pp. 849-852, Kagawa, Japan, July. 2006.

[14] 三輪信介, 宮地利幸, 大野浩之:
“不正アクセス等再現実験環境の統合実験”, マルチメディア，分散，協調とモバイル（DICOMO）2005 シンポジウム論文集,ISSN 1344-0640, 情報処理学会, pp393-396, 岩手, 2005 年 7 月.

[15] 高島大裕, 礒崎直樹, 宮地利幸, 知念賢一, 篠田陽一:
“有線ネットワークにおける無線ネットワークのデータ到達性エミュレーション技法”, 第 6 回インターネットテクノロジーワークショップ（WIT）2004 論文集, 日本ソフトウェア科学会 研究会資料シリーズ No.36 ISSN1341-870X, 日本ソフトウェア科学会 インターネットテクノロジ研究会, pp. 49-55, 石川, 2004 年 12 月.

[16] 宮地利幸, 知念賢一, 篠田陽一:
“SpringOS/VM: 大規模ネットワークテストベッドにおける仮想機械運用技術”, 情報処理学会研究報告書 2005-OS-99, ISSN 0919-6072, pp. 105-112, 沖縄,2005 年 5 月.

[17] 宮地利幸, 宇夫陽次朗, 森島直人, 篠田陽一:
“N*（NStar）: ns-2 の real external interface の構想”, 情報処理学会, マルチメディア通信と分散処理（DPS）研究会 103-20,pp. 113-118, 札幌, 2001 年 6 月.

[18] 三輪信介, 宮地利幸, 大野浩之, 篠田陽一:
“不正アクセス等再現実験環境の統合手法に関する研究”, 電子情報通信学会,2005 年 暗号と情報セキュリティシンポジウム（SCIS）2005, pp. 1183-1188, 兵庫, 2005 年 1 月.

[19] 三角真, 宮地利幸, 知念賢一, 篠田陽一:
“実ノードを利用したネットワークシミュレーションにおけるノードへの OS の導入及びパラメータ設定機構の開発”, 情報処理学会研究報告書 DPS-116, ISSN 0919-6072, pp. 95-100, 鳥取,2004 年 1 月.

[20] 宮地 利幸:
“誰でも使える StarBED 支援 OS の開発”, 北陸 IT 研究開発支援センター 研究成果発表会 新世代ネットワーク社会に向けて, 石川, 2005 年 11 月.

[21] Toshiyuki Miyachi and Kenjiro Cho:
“Comparison of server selection algorithms”, 5th CAIDA-WIDE Workshop, CA, USA, March. 2005.

[22] 宮地利幸, 知念賢一, 篠田陽一:
“大規模な汎用実験環境における資源管理”, インターネットコンファレンス 2004（Work In Progress）, 日本ソフトウェア科学会 研究資料シリーズ No.33, ISSN 1341-870X, pp. 131, 茨城, 2004 年 10 月.

[23] 宮地利幸, 知念賢一, 篠田陽一:
“StarBED における実験支援システム”, インターネットコンファレンス 2004（デモンストレーション）, 日本ソフトウェア科学会 研究資料シリーズ No.33, ISSN 1341-870X, pp. 141, 茨城,2004 年 10 月.

[24] Junya NAKATA, Toshiyuki Miyachi, Ken-ichi Chinen, Yasuo Tan and Yoichi Shinoda:
"StarBED2: Real-time Testbed for Ubiquitous Networks", Proceedings of INSS2006（Poster）, ISBN: 0-9743611-3-5, p.145, Illinois, USA, June. 2006.