# 第 XXII 部

# 実ノードを用いた大規模なインターネットシミュレーション環境の構築

# 第 22 部
# 実ノードを用いた大規模なインターネットシミュレーション環境の構築

## 第 1 章　はじめに

Deep Space One ワーキンググループは実環境向けのハードウェアおよびソフトウェアを利用した大規模な実験用環境の構築・運用に関する研究に取り組んでいる。実ノードを用いた大規模な実験設備として StarBED や GARIT、仮想機械を利用した VM Nebula などの開発と運用を通して、実験設備のあり方や、実験設備の制御方法、さらに、実験の体系的分類などの議論を進めている。また、これに加え、実験を柔軟かつ容易に行うため、実験用環境への実トポロジの再現手法や、標準的なベンチマーク実験のテンプレート化などの研究を行っている。

本報告では Deep Space One ワーキンググループの本年度の主な活動について報告する。

- 実機ベース汎用大規模実験環境の状態の保存・復元機構
- 実験トポロジの構築手法
- 模倣インターネット環境の構築――AS 間ネットワーク構築の試行――
- 大規模ネットワーク実験設備への要件
- 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
- 実験報告
  - StarBED および SpringOS の性能評価
  - StarBED を用いた ICT ストレッサの実現

## 第 2 章　実機ベース汎用大規模実験環境の状態の保存・復元機構

本章では、ネットワーク実験をより容易にするため、テストベッド上のネットワークやソフトウェアの構成を保存・復元する機構について紹介する。文中の実装や動作確認は StarBED で行われたが、本機構の前提は他のテストベッドと共通する点も多く、StarBED 以外でも広く利用できるだろう。

### 2.1 実験保存・復元の必要性

ネットワーク実験テストベッドでは、様々な構成のネットワークを構築し実験が繰り返し実施される。さらに、一旦実験が終了しても、その後、実験が必要となる場面も多い。一旦終了から時間が経つと、別の実験向けにネットワーク構成が変更され、目標とする実験の構成に戻す作業が必要な場合もある。とくに、複数の利用者が多数の実験を同時に実施している大規模テストベッドでは、変更の規模が大きく頻度が高いため、この実験再開の準備に割かれる時間も長い。

そこで、本研究では実験の保存と復元を自動化し、実験再開の準備時間を軽減する。これにより実験再開が容易になり、テストベッドの稼働率、そして利用者の実験意欲を高める効果を期待できる。また、この自動化は人為的なミスの軽減ももたらす。

### 2.2 保存・復元機構の設計

#### 2.2.1 前提：期待する設備

実験を保存・復元するためには、テストベッド上で実施されている実験、そして対象となる資源を把握せねばならない、したがって、本研究では、テストベッドに資源を管理するなんらかの機構（資源管理機構）が用意されているものと仮定する。

また、実験ネットワークは VLAN などの論理（仮想）ネットワークで構築されていると仮定する。ケーブル敷設や機材移動（ラックへのマウント）などの物理ネットワーク変更の自動化は難しいため、本研究では除外する。特殊な装備として、L1 スイッチなる信号（フレーム）レベルでネットワークを論理的に構築可能な機材もあるが、これも除外する。

実験に用いられるノード（PC）は、管理にひとつ、実験向けに 1 つ以上、計 2 つ以上のネットワークインターフェイスを持つことを要件とする（図 2.1）。

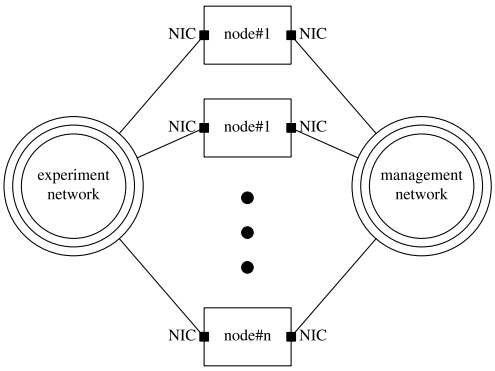


**図 2.1.**　ノードのネットワーク要件

### 2.2.2 プログラム構造

本研究では、実験をネットワークとソフトウェアの 2 側面で扱う。そこで、ネットワークとソフトウェアのそれぞれの構成を保存・復元する機構を設計する（図 2.2）。

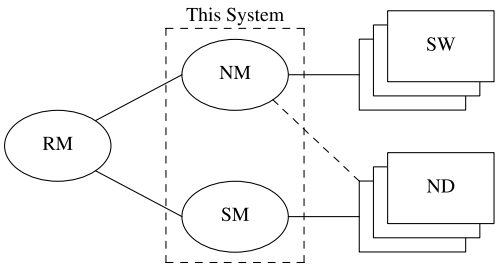


**図 2.2.**　プログラム構造

### 2.2.3 ネットワーク保存・復元

ネットワーク構成保存では、施設の資源管理機構との通信により、対象実験に所属しているノード群、そのノード群が接続しているスイッチやルータなどのネットワーク機器のポート集合を得る。次にネットワーク機器に接続し、それらのポート集合が形成している論理ネットワークを得る。その結果をファイルに保存する（図 2.3）。本研究ではノード外部にあるファイルサーバにこのファイルを保存する。

記録される内容は以下の通り。

- （論理）ネットワーク情報（VLAN）
- 所属ノード仕様
  CPU、メモリ、ディスクの種類や数量
- 所属ノード-NIC-ネットワーク機器ポート対応

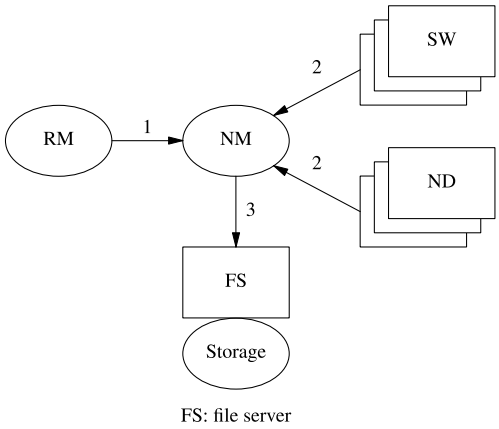


**図 2.3.**　ネットワーク保存・復元

- NIC 属性
  メディア種別（GigabitEthernet など）、IP アドレス

ネットワーク構成復元時は、記録されたネットワーク構成をもとに、ネットワーク機器上の対象ノード接続ポートで論理ネットワークを構築する。そして、各ノード上で IP アドレスなどを設定する。

### 2.2.4 ソフトウェア保存・復元

本研究では、補助記憶装置に保存された内容を**ソフトウェア**とよぶ。主記憶の内容は対象としない。

ソフトウェア構成保存では、施設の資源管理機構との通信により、対象実験に所属しているノード群を得る。次に各ノードのディスクイメージやファイルシステムなどの粒度でソフトウェアを保存する（図 2.4）。

ソフトウェアの復元は、保存された内容をその粒度に合わせて復元する。

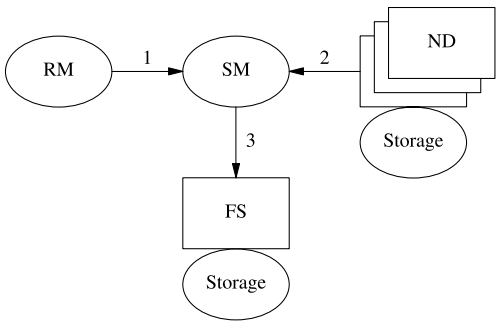


**図 2.4.**　ソフトウェア保存・復元

### 2.2.5 ソフトウェア保存粒度

**ディスクイメージ：**

ディスクやパーティション全体を保存すれば、

OS やファイルシステムに由来する制限を受けない。一方、保存機構はその内容の構造（ファイルシステムなど）を考慮していないため、未使用領域も含めて全体を保存・復元せねばならない。したがって所要時間は長い。

実験施設での実装例は [180, 200] などがある。

**ファイルシステム：**

ファイルシステムレベルの保存・復元は保存機構がその構造を理解しているため、未使用領域やファイルの空隙などの処理は不要となり、所要時間が短くなる。ただし、適用可能な OS やファイルシステムが限定される。また、シングルユーザモードでの実行が推奨されるなど制限をもつ。

**ファイルアーカイブ：**

第 3 のアプローチは、ファイルシステムの内包する i-node などの管理構造を省き、ディレクトリとファイルの構造のみを扱う。このレベルの保存は所要時間が最も短いが、ブートローダなどシステム依存の情報の復元は非常に困難となる。しかし、OS やファイルシステムに依存しないため、多くの実験に適用可能である。

#### 2.2.6 増分保存

実験の繰返しを指向するのなら、増分を保存して、より高速化が可能であろう。さらに、増分の考え方をノード間に適用すれば、共通部分と各ノード特有部分を分けて保存する着想に至る。また、共通部分をあらかじめ固定してインストールしておけば、毎回の再現時の復元に要する時間はさらに短くなるだろう。

#### 2.2.7 代替による柔軟な復元

実験再開時に、保存した実験で使用していた資源が使用できないこともある。これは、故障や他実験で利用中などの致命的・一時的な事象がある。いずれにせよ、保存した実験構成を、同一資源を用いた完全な復元を求めれば、その実現性は低くなる。

そこで、構成復元では保存中の資源と等価な資源を代わりに用いる（代替）ことで、その実現性を高める工夫を施す。たとえば、記録した実験構成のノードと同じ仕様であれば、任意のノードが代替資源として利用できるだろう。

### 2.3 実装

実装は大規模ネットワークテストベッドである StarBED[154] で行った。図 2.5 に実装構造を示す。資源管理機構は StarBED で稼働している SpringOS[115] の experiment resource manager（ERM）を用いた。

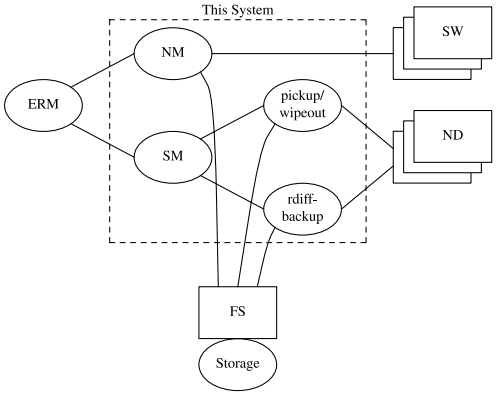


**図 2.5.** 実装構造

実験のネットワーク構造の保存・復元は、SpringOS の switch manager（SWMG）の一部に VLAN 走査などの機能を追加することで実現した。スイッチへの接続は TELNET プロトコルを用い、Foundry Networks 製 Ironware、Cisco Systems 製 IOS を制御可能である。なお、ERM が各ノード NIC が接続されたスイッチポートの対応情報を保持しているため、本実装では NIC-スイッチポート対応表は記録していない。

ディスクイメージの保存・復元は同じく SpringOS の pickup/wipeout を制御することで実現した。ファイルアーカイブの保存・復元は rdiff-backup[134] を制御することで実現した。

#### 2.3.1 資源等価性

StarBED のノードはほぼ同一で、名称で等価性を確認できるため、現在のところ実装していない。

### 2.4 動作確認

設計通り、以下の 4 つの復元の動作を確認した。

a) 同一資源を用いた完全復元
b) 代替ノードを用いた復元
c) 代替 VLAN を用いた復元
d) 他ベンダ製スイッチを用いた復元

この b)–d) が前述の柔軟な復元である。

### 2.5 将来への展望

実験中に発生するプロセスを保存・復元できれば、より便利になるだろう。そこで、プロセスの履歴の

保存を実装中である。

本研究の主題は実験をそのままの保存・復元であった。この方法はトライ・アンド・エラーの結果を再現するには有効な手段である。その一方で、手順をスクリプトで記述して実験を実施するアプローチもある。トライ・アンド・エラーで実験手順を検討した後は、手順が確立し、スクリプトへ移行することも考えられる。そこで、SpringOS Kuroyuri など、スクリプト実行機構との連携も興味深い開発対象である。

前述のように本システムでは等価性には対応できていない。これは今回採用した ERM の制限で、将来は対応が必要であろう。

本システムでは論理ネットワークとして VLAN を扱った。VLAN 以外（ネットワーク層など）への対応も望まれる。

### 2.6 総括

ネットワーク実験を繰り返し実施する際、実験再開の準備に多くの時間が割かれる。その時間を軽減するため、本研究では実験のネットワークやソフトウェアの構成を保存・復元する機構を設計した。単純な復元だけなく柔軟な復元も実現し、実験再開が容易となった。

実験にはネットワークやソフトウェア以外の側面もあるため、今後はそれらに逐次対応していく。

（`wide-paper-deepspace1-nonayou-ic2007-00.txt` 参照）

## 第 3 章　模倣インターネット環境の構築——AS 間ネットワーク構築の試行——

インターネットのような大規模な分散環境に新しい技術を導入する場合、その技術がインターネット上で適切に動作するのか、既存のインターネット環境に悪影響を及ぼさないのかなどを、バイナリ実装レベルで実践的に検証しておく必要がある。このような検証を行うためには、現実のインターネットに則した実験環境が必要である。そこで我々は、バイナリ実装の実証実験などに際し、インターネットの代わりとして利用できる模倣インターネット環境をテストベッド上に構築することを目指して、いくつかの取り組みを始めた。ここでは、模倣インターネット環境のうち、AS 間ネットワークの構築手法について述べ、テストベッド上への AS 間ネットワーク構築の試行とその評価、考察を行う。

### 3.1 模倣 AS 間ネットワークの要件

ここでは、インターネットの主要な中間要素である AS 間ネットワークに着目し、その模倣における要件について述べる。なお、インターネットの模倣を行うには、対象となるホストの模倣や AS 内ネットワークの模倣などについても検討を行う必要があるが、本稿では扱わない。

#### 3.1.1 模倣

本研究では、バイナリ実装の実証実験などを行う際に、インターネットの代替として利用可能な環境の構築を目指している。そのため、シミュレーションによる実現ではなく、バイナリ実装が動作する環境を用意し、通信では実パケットがやりとりされるような環境を構築することを目標とする。また、実際のインターネットに則した環境とするために、模倣 AS 間ネットワークのトポロジは実際のインターネットをもとに構成することとする。

インターネット上には、前述の通りおよそ 26,000 の活動中の AS（CAIDA の AS Ranking[26] による）があり、複雑な AS 間ネットワークが形成されている。AS 間の到達性情報は、EGP（Exterior Gateway Protocol）で交換され、すべての AS に必要な到達性情報が伝搬することで AS 間の通信を可能とする。いくつかの EGP があるが、ここでは単純化のため、代表的な EGP である BGP4（Border Gateway Protocol Version 4）のみを取り扱うこととし、AS 間ネットワークは BGP4 によって到達性情報を交換し、それに従って経路制御される網とする。

よって、BGP スピーカと BGP4 の到達性情報にもとづき経路制御するルータ、AS と他の AS をつなぐリンクが各 AS の最低限の構成要素であり、これらを実際のインターネットの AS 間ネットワークのトポロジに則して構成することで、AS 間ネットワークを模倣することとする（図 3.1 参照）。また、各リンクは帯域や遅延、パケットロス率などの性能が異なる。これらについても、実際のインターネットの各リンクの性能に則して模倣することとする。

なお、規模に関しては、対応できる規模が大きいほど良いと考えられるが、まずは現状のインターネッ

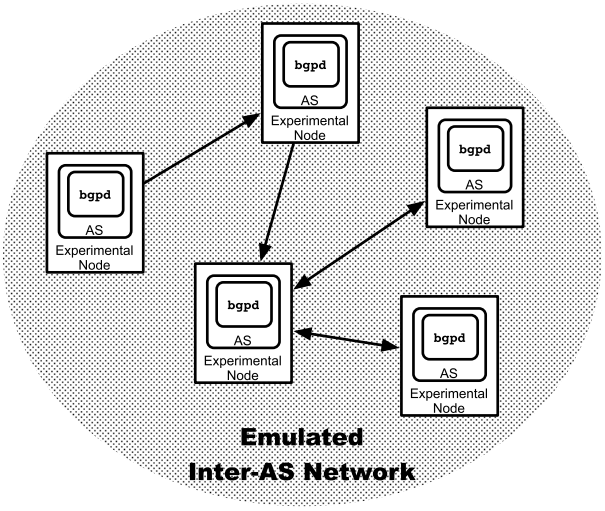


**図 3.1.** 模倣 AS 間ネットワーク

トの規模を最大目標値に設定し、AS 数では 26,000 程度、経路数では 250,000 程度 [66] を目標とする。

### 3.1.2 テストベッド

テストベッド上への AS 間ネットワークの構築手法を検討するにあたり、前提とするテストベッドについて確認しておく。

多くのホストやルータから成るインターネットを模倣するために、多数のノードから成るクラスタ環境であると想定する。ノード間は、何らかのネットワークリンクで接続され、必要とされるトポロジに合わせて柔軟にトポロジを変更できるものとする。また、バイナリ実装の実証実験などを目的とするため、各ノードは、OS やアプリケーションソフトウェアのバイナリ実装が動作する PC のノードを動作環境と想定するが、バイナリ実装が動作するなら、物理ノードであるか仮想ノードであるかは問わないこととする。

我々が研究開発してきた StarBED[114] は、830 台[1] の PC から成るネットワーク実証実験用クラスタで、すべてのノードは Ethernet もしくは ATM によって接続されており、VLAN や VC/VP によってネットワークトポロジを柔軟に変更可能であり、前提に則している。よって、本研究では、StarBED を用いて模倣インターネットをテストベッドに構築する試みを行うこととした。

## 3.2 構築手法

要求をもとに前提とする環境上に、模倣 AS 間ネットワークを自動的に構成する手法について述べる。

### 3.2.1 提案手法の概要

実際のインターネットの AS 間ネットワークのトポロジに則して構成するために、CAIDA の AS Relationship データをもとにして模倣 AS 間ネットワークは構成する。CAIDA の AS Relationship データは、AS 番号の組と AS 間の関係（customer、peer、provider、sibling）を示す数値からなる AS 間の関係性のリストである。具体的には、CAIDA の AS Relationship データを我々が研究開発してきた AnyBed[161] のトポロジ記述に変換し、そのトポロジ記述にもとづいて AS 間ネットワークの物理・論理トポロジを構成した。

本来 AS は、AS 内ネットワークと複数の AS 境界ルータから成っているが、ここでは単純化のために、各 AS を単一のノードで実現することとし、ノード上では BGP スピーカと BGP によるルーティングデーモンを動作させる。隣接 AS 数に応じてノードには仮想ネットワークインタフェースを用意し、すべての AS は隣接 AS と直接接続される。BGP スピーカとルーティングデーモンとしては、Quagga[132] の `bgpd`、`ospfd`、`zebra` を用いた[2]。

1　正確には 680 台の StarBED の PC 群と 150 台の JAIST から借用している PC 群。
2　今回の実験では、1 AS 1 ノード構成とし、`ospfd` は利用していないが、1 AS 多ノード構成も可能となっており、その場合には `ospfd` も利用する。

各 BGP スピーカの設定は、CAIDA の AS Relationship データをもとにした AnyBed のトポロジ記述から生成し、AS 間の関係にもとづいて設定するリンクの方向に従って経路フィルタを行うようにした。

ネットワークリンクは、当初各 AS 間リンクに Ethernet の IEEE802.1Q タグ付き VLAN を 1 つ割り当てることを想定した。しかし、タグの数が最大でも 4096 で不足だったため、すべての AS 間リンクを 1 つの VLAN 上で構成し、IP アドレスレベルでネットワークの分割を行うこととした。各ネットワークリンクの性能は、ネットワークエミュレータ `netem`[56] でエミュレートする。しかし、現状では各ネットワークリンクの性能を導出する手法が定まっていないため、今回の試行では、ネットワークリンクの性能については変更していない。

各 AS を単一のノードで実現したとしても、すべての AS を物理ノードで構築した場合、25,000 程度の物理ノードが必要となり、現実的ではない。しかし、シミュレーションではバイナリ実装に関する検証環境とはなりえない。そこで、仮想化技術を利用し、ハードウェア仮想化や OS 仮想化による仮想ノードを用いることとした。具体的には、XEN[28] によって Debian Linux を準仮想化し用いた。

物理ノードの OS としては、OS パッケージとして他種類の仮想化技術が導入済みで、Debian Linux をベースとした平易なパッケージである VMKnoppix[194] を採用した。VMKnoppix をネットワーク起動に対応するよう改良し、物理ノードはディスクレスでネットワーク起動する。

各物理ノードと各仮想ノードには、AS 間リンクとは別に、実験管理用のネットワークインタフェースを用意し、それを介して各種の操作を行う。

### 3.2.2 構築の手順

前述の概要にしたがい、入力されたノード記述と CAIDA AS Relationship のデータから、自動的に模倣 AS 間ネットワークが構成する提案方式の概略を以下（図 3.2）に示す。

0. 入力として、仮想ノードのノード設定の記述（以降、ノード記述）と物理ノードの資源記述、CAIDA AS Relationship のデータ（as-rel）を与える
1. CAIDA の AS Relationship データから指定したフィルタリング方式にしたがい **TF**（**Topology Filter**）が as-rel の一部を抜き出す
2. **TC**（**Topology Calculator**）がそれを AnyBed の論理トポロジ記述に変換
3. 論理トポロジ記述とノード記述から **CG**（**Configration Generator**）が各ノードの設定ファイル群（`bgpd.conf`、`zebra.conf` など）を生成
4. **WA**（**World Allocator**）が各仮想ノードをノードの設定と物理ノードの資源状況に合わせて割り当て、割り当てファイル（SWI）を作成
5. **WS**（**World Seeder**）は割り当てファイルに

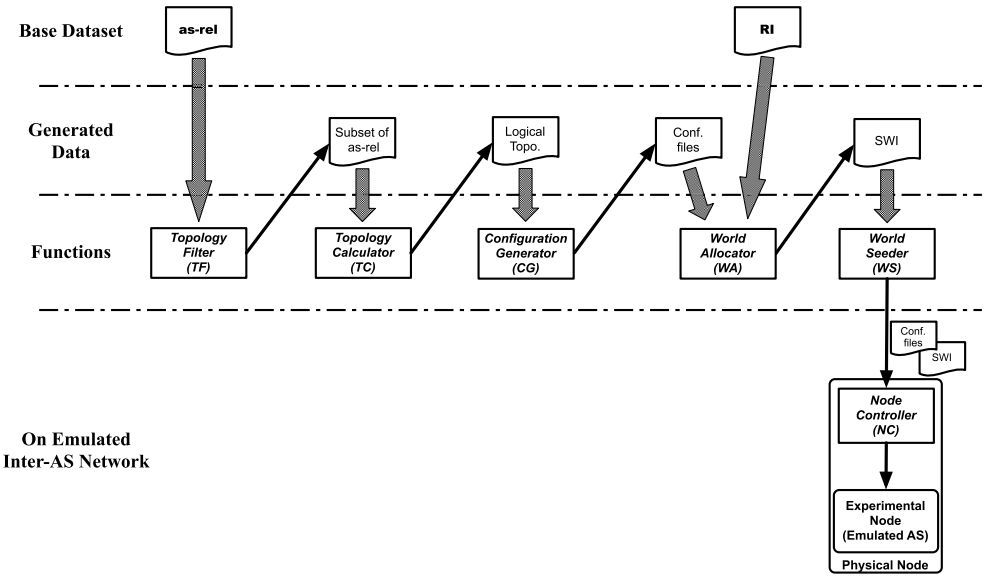


**図 3.2.**　提案方式の概要

したがい、物理ノードを起動し、XEN の設定・起動、物理ネットワークの設定と **NC**（**Node Controller**）の起動を行う

6. **NC** は設定ファイルを各仮想ノード用の XEN ディスクイメージに挿入し、各物理ノード上で割り当てられた仮想ノードを起動し、各仮想ノード起動時に仮想ネットワークインタフェースの設定と BGP スピーカおよびルーティングデーモンを起動する

**TF** と **TC**、**CG** については 4.1 節に詳細を述べる。また、XEN による仮想ノードで模倣環境を作る **WA** と **WS**、**NC** は、我々が XENebula と呼ぶツール群によって実現されている。

### 3.3 試行と評価

提案方式を用いて、実際に StarBED 上への模倣 AS 間ネットワークの構築について、いくつかの試行を行った。各試行には下記のルールで名称を付けてある。以降では、これに従って試行を区別する。

| 模倣対象. 試行範囲. もとにしたトポロジ |
|---|

ここでは、

- 簡単な性能評価を行った “InterAS.Top250.CAIDA070430”
- 日本のインターネットの模倣を目指した “InterAS.JPNIC0707.CAIDA070430”
- 現時点での最大 AS 模倣数を目指した “InterAS.Top5000.CAIDA070430”

について述べる。今回のすべての試行は 2007 年 4 月 30 日時点の CAIDA AS Relationship のデータにもとづいてトポロジを作成している。

また、すべての試行は、StarBED のグループ F を用いて行った。グループ F のノード構成については表 3.1 に示す。グループ F は 168 台あるが、今回はそのうち必要な台数のみを用いた。

**表 3.1.** 物理ノード　グループ F の構成

| 項目 | 構成 |
|---|---|
| CPU | Intel Pentium4 3.2 GHz × 2 |
| Memory | 2 GB |
| NIC | 1000Base-T × 6 |

#### 3.3.1 上位 250 AS — InterAS.Top250.CAIDA070430

CAIDA の AS Relationship の上位 250 AS を単純に抜き出し、AS 間ネットワークの模倣を試みた。5 物理ノードにそれぞれ 50 AS ずつを割り当て、模倣した。この模倣 AS 間ネットワーク環境を用いて、簡単な性能評価を行った。評価としては、遅延（RTT）と帯域について、物理ノードの性能との差異を計測した。

評価は、物理ノードと物理ノードを直結した場合（「物理ノード直結」）、物理ノード上で直接ルーティングデーモンを動作させて 5 ホップの経路を経由した場合（「物理ノード経由」）、模倣 AS 間ネットワークで 5 ホップの経路を経由した場合（「模倣 AS 間ネットワーク経由」）の 3 種について（図 3.3 参照）、`ping` コマンドと `iperf`[170] で計測した。計測結果を表 3.2 に示す。

物理ノード直結の場合の RTT は 0.1 程度（100 回試行平均値）で、物理ノード経由では 1.0 程度、模倣

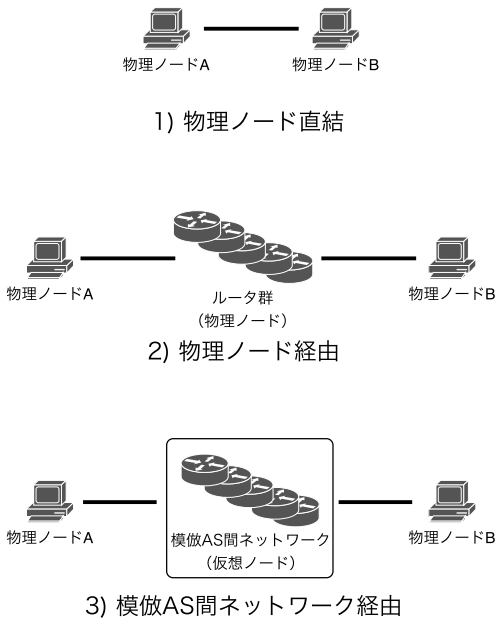


**図 3.3.** 評価環境

**表 3.2.** 計測結果

| 環境 | ping 平均 (ms) | iperf (Mbps) |
|---|---|---|
| 物理ノード直結 | 0.122 | 961 |
| 物理ノード経由 | 0.936 | 961 |
| 模倣 AS 間ネットワーク経由 | 1.489 | 160 |

AS 間ネットワークを経由した場合は 1.5 程度と、差異は認められるが大きな差異はない。帯域は、物理ノード直結の場合は 960 Mbps 程度、物理ノード経由の場合も同様となったが、模倣 AS 間ネットワークを経由した場合は 160 Mbps 程度であり、6 倍程度の差はあるが、100 Mbps 程度のネットワークを模倣する実験には実用上十分といえる。

#### 3.3.2 JPNIC — InterAS.JPNIC0707.CAIDA070430

CAIDA AS Relationship のデータから 2007 年 7 月時点の日本ネットワークインフォメーションセンター（JPNIC）の AS 番号登録情報に記載のある 394 の AS を抜き出し、日本の AS 間ネットワークの模倣を試みた。仮想ノードへのメモリ割り当ては前述の通りで、実行した結果、5 物理ノードで 394 AS を模倣することができた。

しかし、経路を評価した結果、複数の AS で経路の断絶が観測された。これは、単純に 394 AS を抜き出したため、394 AS に属さない AS を経由してしか BGP 経路情報を受け取れない AS が多くあったためと思われる。

#### 3.3.3 上位 5000 AS — InterAS.Top5000.CAIDA070430

CAIDA の AS Relationship の上位 5000 AS を単純に抜き出し、AS 間ネットワークの模倣を試みた。各仮想ノードのメモリは、隣接 AS 数 25 個あたり 24 MB を割り当てることとし、実行した結果、100 物理ノードで 5000 AS を模倣することができた。

ただし、50–100 仮想ノードが正常に起動せず、再起動を必要とした。この現象は、1000 AS 以上の模倣から散見されるようになり、仮想ノード数の増加に従って増えている。

また、模倣 AS 間ネットワークでは、ほぼ同時期にすべての BGP スピーカが起動するが、BGP では多数の AS が同時に立ち上がる状況を想定していないためか、BGP 情報が大量に増減を繰り返す現象がしばらく続いた。

仮想ノードへのメモリ割当量は、`bgpd` のメモリ利用量が隣接 AS 数に応じて増加することと、およそ 25 隣接 AS あたり 24 MB 程度を割り当てた場合には、メモリ不足のエラーを生じなくなることから、経験的に割り出した。この値は、上位 5000 AS 程度までの実験では普遍的に利用できるが、これより大きな AS 数を模倣する場合にも適用できるのかについては、検証が必要であり、今後より多数の物理ノードを用いた実験を計画している。

（`wide-paper-deepspace1-danna-ic2007-00.txt` 参照）

## 第 4 章　実験トポロジの構築手法

我々は、インターネットを対象とした様々な研究開発において、実インターネット規模のトポロジを用いて検証や評価を行うことは有用であると考えている。その際、大規模なトポロジを一から設計・作成するのは非常に負荷が高い。また、IP トレースバックやセキュリティの研究者からは、より実インターネットに近いリアリティのあるトポロジを用いて実験を行いたいという要望を受けている。我々はそのような要望を受け、実インターネットの計測データをもとに実インターネットを模したトポロジをテストベッド上に構築する研究とツールの開発を行っている。

本節では、3.2 節で述べた模倣トポロジの構築手順のうち、TF の詳細および具体例について記述する。

また、構築した実験トポロジの整合性検証、動作検証及び状況監視を行うための手法とツールについて述べ、最後に今後の課題について議論する。

### 4.1 実験トポロジの元となるデータセットとその加工

我々は実インターネットを模したトポロジを作成するために、インターネットで CAIDA が公開している AS Relationship データセット [21]（以下 as-rel データ）を利用している。as-rel データは、CAIDA が観測した実インターネットの AS トポロジを 1 行 1 接続関係の単純な構造として記述したテキストファイルである。ファイルの内容は図 4.1 のようになっており、1 列目と 2 列目が接続関係にある AS 番号で 3 列目が AS 間接続の種類を示している。接続の種類は 4 つあり、値が −1 であれば 1 列目の AS 番号の AS が 2 列目の AS 番号の AS のカスタマ AS、値が 1 であればその逆であることを示している。値が 0 の場合はそれぞれの AS が対等のピア関係にあり、値が 2 であればそれぞれの AS が同じ組織であ

るということを示している。

また、as-rel データの他に AS 番号、AS 名、場所を記述した asn_expand.txt[3]、日本国内に存在する AS 番号、AS 名を記述した as-numbers.txt[197] を利用して、AS 名と AS 番号の対応付けを行いノードのホスト名や可視化の際に AS 番号ではなく AS 名で認識できるようにしている。

実験トポロジを作成するためには、これらのデータセットを組み合わせて内容を加工し、各ノードの設定を生成せねばならない。そのために我々が開発したツールの一覧は下記の通りである。

- uniq-as.awk
  uniq な AS 番号に何があるかをチェックする
- grep-as-by-hop.pl
  as-rel データから AnyBed 用リンクファイル形式用にある単一 AS を中心として、$N$ Hop 内に存在するすべての AS を抜き出す

```
8563 2914 -1
8563 21414 -1
8563 21482 1
8563 3277 0
8563 4323 -1
28801 702 -1
28801 8422 -1
28801 8220 -1
```

**図 4.1.** CAIDA AS Relationship ファイルの内容

- grep-jprs-from-asrel.pl
  CAIDA の as-rel データから日本国内の AS のみを抜き出す
- asrel2anybedtopo.rb
  隣接 AS 数をもとに AS のランキングを行い、ランキングの上位 AS を任意の数抜き出す

これらのツールを組み合わせることにより、元のデータセットから任意の数の上位 AS を抜き出したり、日本国内の AS のみを抜き出すといった加工を行う。

図 4.2 にその加工の流れを示す。まず、AS 番号、AS 名、場所等が含まれた asn_expand.txt から name_from_asn_expand.sh を用いて AS 番号と AS 名のみを抜き出し as-name.txt とする。その後、実験に必要な AS トポロジに応じて、日本国内の AS のみを抜き出す場合は grep-jprs-from-asrel.pl、任意の数の上位 AS のみ抜き出す場合は asrel2anybedtopo.rb、特定 AS から任意半径の AS のみを抜き出す場合は grep-as-by-hop.pl を組み合わせて用いる。加工された as-rel データは as-name.txt を共に asrel2anybedtopo.rb により AnyBed 論理トポロジ形式に変換され、その後 configgen.rb によりノードの設定ファイルが生成される。また、as-name.txt と加工された as-rel データをもとにして QT AS Viewer によりトポロジの可視化が行える。図 4.3 に QT AS Viewer の画面を示す。QT AS Viewer はもともと IP Traceback の実験用に開発されたため、トレースバックのリクエスト・リプライや DDoS の被害者・

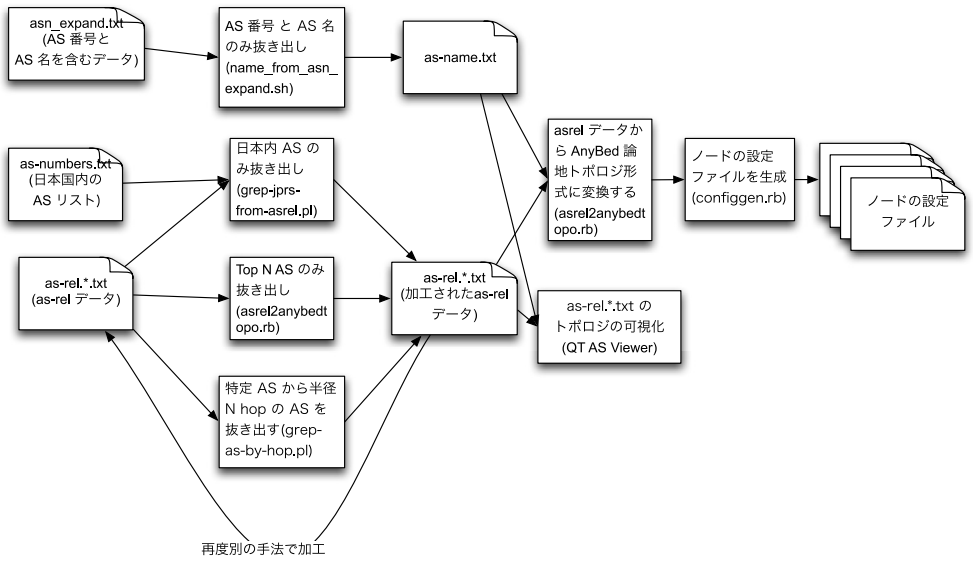


**図 4.2.** as-rel データの加工

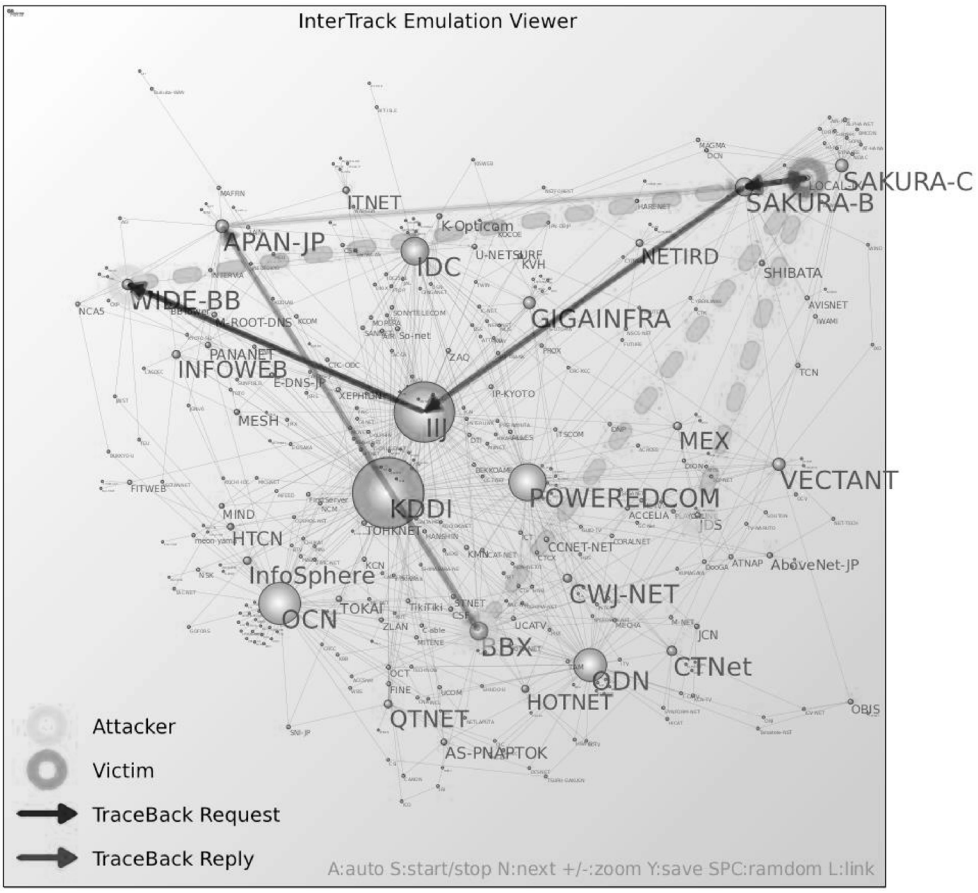


**図 4.3.**　QT AS Viewer で可視化した日本国内の AS トポロジ

攻撃者を表示する機能が備わっている（図 4.3 の左下凡例参照）。

### 4.2 ノードを用いた実験トポロジのエミュレート

実験トポロジを構築するためには、前節で述べた configgen.rb により生成された実ノードの設定を実ノードに反映させる必要がある。その反映機能、生成機能を含めたトポロジ構築システム全体を我々は AnyBed[161] と呼んでいる。AnyBed についての詳細は参考文献か 2006 年の WIDE 報告書を参照されたい。

AnyBed が生成する設定は実ノード用であるが、XENebura を用いることで Xen の仮想マシンを実ノード上で動作させ、ノードの台数を大幅に増やすことができる（3 章参照）。

### 4.3 実験トポロジの動作検証と状況監視

過去に我々が行った実験の経験から、ハードウェアの故障、経路制御デーモン起動のタイミング、トラフィック過多による経路制御パケットの喪失、高い CPU 負荷による経路制御デーモンの異常動作などの原因により実験トポロジが正しく構築されない場合があることが分かっている。我々はこのような異常を検知するために実験トポロジの動作検証と状況監視が必要であると考え、そのためのツールである Oil を開発した。

Oil が持つ検証・監視の機能は、各ノードが持つ実験用 IP アドレスへの到達状況監視である NxM ping と経路の伝達状況監視である NxM netstat の 2 つに分けられる。NxM ping では、各ノードが持つ複数の実験用 IP アドレスに対して他のノードから到達性が有るかどうかを PING によってチェックする。NxM netstat では、各ノードが保持すべき経路をあらかじめ計算しておき、その経路と各ノードが実際に保持している経路を比較することにより、経路が正しく行き渡っているかどうかをチェックする。

これらの機能は図 4.4 に示されるようなアーキテクチャで実現されている。実験に用いられる各ノード

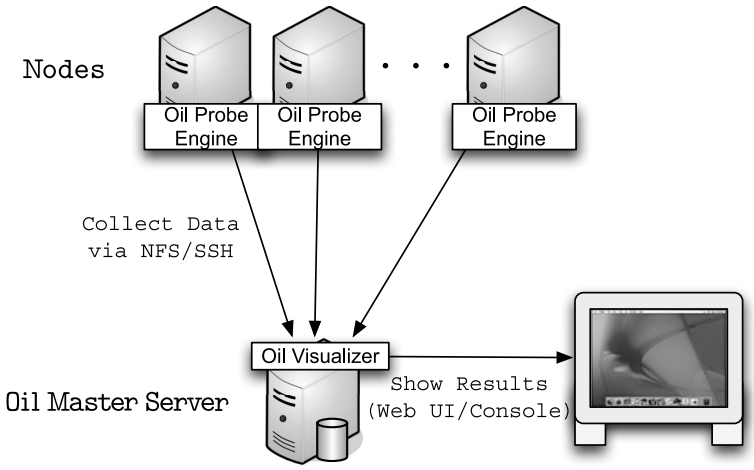


**図 4.4.** Oil アーキテクチャ


**図 4.5.** NxM ping と NxM netstat の結果の可視化

では Oil Probe Engine が動作し、PING の結果や経路表を NFS や SSH 経由でマスターノードに送信する。マスターノードでは Oil Visualizer がそれらの結果を実験者に分かりやすい形に可視化し表示する。

図 4.5 は NxM ping と NxM netstat の結果を Web から閲覧できるような形に可視化した表示例である。図の上側の NxM ping の結果表示部には、上行から下行に向けて各ノードが順に並んでおり、左列から右列に向けて各ノードの実験用インタフェイスに割り当てられている IP アドレスが順に並んでいる。この表を用いて各ノードから各ノードの実験用インタフェイスに向けて PING により到達性を確認し、到達性のある IP アドレスを緑色、到達性の無い IP アドレスを赤や灰色で表示する。図の下側の NxM netstat の結果表示部では、上行から下行に向けて各ノードが順に並んでおり、左列から右列に向けて各ノードが広報している経路が順に並んでいる。この表により、各ノードが保持すべき経路のうち、実際に保持している経路を緑色、保持していない経路を灰色で表示する。

また、ファイアウォール内に存在するテストベッ

```
root@deb-master:~/commander/bed# ./textoilroute.pl \
   -D g-sun32
 run dbgpcheck g-sun32
 hosts g-sun32 netstat and netstat -a \
   at 2007-12-2809571198803462

 check netstat log

 check at 192.168.255.101
     192.168.255.101 catches listen / total = 63 /63
 check at 192.168.255.102
     192.168.255.102 catches listen / total = 63 /63
(snip)
 check at 192.168.255.132
     192.168.255.132 catches listen / total = 63 /63
 start time     : 1198803462.34687
 dsh start time : 1198803462.38951
 dsh end time   : 1198803463.53383
 end time       : 1198803463.88291
```

**図 4.6.**　textoilroute.pl の実行例

```
root@deb-master:~/intertrack-scripts/commander/bed# \
   ./textoilbgp.pl -D g-sun32
 run dbgpcheck g-sun32
 hosts g-sun32 netstat and netstat -a \
   at 2007-12-2810291198805390

 check netstat log

 check at 192.168.255.101
     192.168.255.101 catches
    ESTA in v4 / OTHER in v4 / ESTA in v6 / OTHER in tcp6 \
      / TOTAL = 0 / 0 / 1 / 0 / 1
 check at 192.168.255.102
     192.168.255.102 catches
    ESTA in v4 / OTHER in v4 / ESTA in v6 / OTHER in tcp6 \
      / TOTAL = 2 / 0 / 0 / 0 / 2
(snip)
 check at 192.168.255.132
     192.168.255.132 catches
    ESTA in v4 / OTHER in v4 / ESTA in v6 / OTHER in tcp6 \
      / TOTAL = 0 / 0 / 1 / 0 / 1
 start time     : 1198805390.63949
 dsh start time : 1198805390.63955
 dsh end time   : 1198805391.29987
 end time       : 1198805391.31126
```

**図 4.7.**　textoilbgp.pl の実行例

ドを遠隔利用する場合などの Web ブラウザが利用できない場合を考慮して、CLI ベースの textoil も開発した。textoil は textoilroute.pl と textoilbgp.pl から構成され、textoilroute.pl では経路の伝達状況確認、textoilbgp.pl は BGPd の接続状況確認が行える。図 4.6 に textoilroute.pl の実行例を、図 4.7 に textoilbgp.pl の実行例を示す。これらのスクリプトは調査対象ノードを記述したグループファイルを引数に取り（例では g-sun32）、それらのノードの調査を行う。

### 4.4 今後の課題

実験トポロジの構築に関する今後の課題として、より正確なインターネットトポロジのエミュレーションを目指すために遅延の挿入機能や現実性のあるバックグラウンドトラフィックの挿入を考えている。また、現在のところ構築可能なトポロジは AS 間接続のもののみであるが、今後は AS 内トポロジと AS 間接続のトポロジを組み合わせたトポロジを構築できるようにしたい。

さらに、既存機能のうち改良すべき点として、スケーラビリティの向上とより実験の状態把握に使いやすい可視化手法の模索がある。現在の Web 閲覧向けの可視化手法では、多数のノードや多数の経路に適用した場合に表示に長時間を要したり視認性が悪化することが分かっている。また、データを収集する時点でも NFS サーバに高負荷がかかり、200 台程度でもリアルタイムな可視化が現状のツールでは行えないことがわかっている。このため、大規模な実験トポロジにも適用可能な新しい可視化手法を考案し実装していきたいと考えている。

## 第 5 章　大規模ネットワーク実験設備への要件

StarBED のような大規模な実験環境を制御するためには、実験設備とそれを制御する支援ソフトウェアにさまざまな機能が必要である。StarBED では、物理的な設備レベルで、実験用と管理用ネットワークの分離や、実験用ノードのコンソールへの容易なアクセス、ノードの死活管理、実験トポロジの柔軟な構築、ノードの起動方法の切り替えなどを提供している。本章ではこれらをふまえて、大規模実験設備への要件を整理する。

### 5.1 StarBED への要求

一般的に実ノードを利用した実験は以下の手順で行われる。

1. 実験トポロジやシナリオの検討
2. 各ノードの接続
3. ノードへのソフトウェアの導入
4. 構築した実験用環境でのシナリオ実行

5. ログの解析

StarBED ではこれらを支援するために実験設備として手順 (1) で決定された内容を満たすための、手順 (2)〜(4) までの手順を補助する。

これらの手順を補助するためには、StarBED に以下で挙げる機能を実現した。ここで「外部からの」はそのノードのコンソールを直接操作するのではなく、ネットワークなどを通し別のノードから一括操作できることを指す。

**外部からの実験トポロジの変更** 外部からスイッチノードなどを制御し、何らかの方法で実験トポロジを生成できる必要がある。

**外部からのソフトウェア導入** 外部から対象ノードへ必要とされる OS やアプリケーション、その設定などを導入する必要がある。

**外部からのノードの死活管理** ノードを利用するために外部からの起動や再起動、停止ができる必要がある。ただし、これはソフトウェアで対応できる部分もある。

また、以上の実験の実行自体に関わる部分だけでなく、高精度な実験を容易に行うために以下の機能が必要となる。

**外部からのノードのコンソール制御** ネットワーク実験中には何らかの障害により、実験に利用していたネットワークから実験ノードに接続できなくなることは珍しくない。また、OS のカーネルなどに関する障害が起きた場合は管理用ネットワークからの接続性も失われてしまう。このような場合に、ノードのコンソールに出力される情報の確認や、コンソールからのノード制御が一つの端末から行えれば復旧作業などが容易となる。

**実験トラフィックと管理トラフィックの分離** 実験の精度を高めるためには、管理トラフィックが実験トラフィックに影響をおよぼさないような構成が必要である。さらに、管理用の通信が実験トラフィックにより影響され、ノード管理を阻害しないよう、それぞれを分離する必要がある。

### 5.2 StarBED の問題点

StarBED には以下の問題点があった。

**ネットワーク分離** StarBED では実験用ネットワークと管理用ネットワークの分離は行っていたが、実験の実行者が日常的に利用するための生活ネットワークの定義がはっきり行われていなかった。また、管理用ネットワークがインターネットに接続可能な状態であった。これにより、インターネット上のリソースに容易にアクセスでき、実験の実行には効率的な場合も多いが、実験用のトラフィックが外部に流出してしまう危険性がある。したがって、標準的な状態では、管理用ネットワークをインターネットから切り離すことが望ましく、実験の実行者からの要求に従ってインターネットとの接続を管理する必要がある。これは実験用ネットワークも同様である。

**ネットワークの性能** StarBED の設置当初には、FastEthernet が主流であったが、近年、FastEthernet では十分に実験の要求を満たせない場合が多い。実験トポロジを構築するための OS の導入は、データ転送量が多く、StarBED の設置時に導入されたネットワーク機器の性能はいまや十分とはいえない。とくに管理用ネットワークでは、設置当初に高い性能が必要であると考えられていなかったため、その傾向は顕著である。しかし、実際には、OS イメージの導入などには管理用ネットワークを利用してきた。このため、実験用ノードとファイルサーバ間などの帯域がボトルネックとなり実験に必要な時間が長くなるといった問題が観測されている。

**ノードの死活管理機能の不足** StarBED のノードは WoL を利用して電源を投入することができる。しかし再起動や停止処理は SNMP やコマンド実行などソフトウェアに頼っている。実験中には何らかの問題により、ソフトウェアによる制御が不可能になってしまう場合が少なくない。このような場合に、遠隔地から再起動や停止を行うためには、ハードウェアレベルでの対応が必要である。

**実験の分離** StarBED の実験用ネットワークは VLAN を利用して構築するため、実験の実行者ごとに分離された環境が構築される。しかし、管理用ネットワークからはすべてのノードに常時アクセスできるよう、固定的に設定がなされており、実験の管理用トラフィックが分離されていない。このような環境では、ある実験で管理用ネットワークに多量のトラフィックを送出した場合に、他の実験の管理用トラフィックに影響をおよぼすだけでなく、実験に関するデー

タが他の実験の実行者に読み取られてしまうおそれや、セキュリティの実験などであれば、マルウェアなどの影響が他の実験に利用されているノードへ波及するおそれがある。

## 5.3 大規模な実験設備への要件整理

まずこれまでの StarBED 運用経験から得られた要件をまとめる。

- 目的別のネットワーク整備
- 外部からのノードのコンソール操作
- 外部からの実験トポロジ設定
- 外部からのノードの起動方法の変更
- 外部からのノードの死活管理

## 5.4 目的別のネットワーク整備

StarBED では管理用ネットワークと実験用ネットワークの 2 つのネットワークを用意していたが、これ以外にも実験の実行者がインターネットに接続するためのいわゆる生活用ネットワークが必要である。以下でそれぞれのネットワークについて整理する。

### 5.4.1 管理用ネットワーク

実験用ネットワークと管理用ネットワークのトラフィックの相互の影響を防ぐため、管理用ネットワークを実験用ネットワークから分離する。

これに加え、他の実験の実行者との管理用トラフィックの分離を行う必要がある。これは、実験間の管理用トラフィックの影響の排除のためである。実験の機密情報を他の実験の実行者から隠蔽するほか、セキュリティ関連の実験などで他の実験で利用されているノードへの攻撃が管理側ネットワークを通して行われないようにするためである。

### 5.4.2 実験用ネットワーク

実験用ネットワークでは、実験の実行者が指定した実験トポロジを構築する。このため、実験用ネットワークには自由にトポロジを変更できる装置が導入されている必要がある。このような機器には StarBED で利用しているような、仮想的にネットワークを構築できる VLAN や ATM に対応したスイッチのほか、物理的に接続を変更できるクロスバなどが利用できる。

### 5.4.3 生活用ネットワーク

実験の実行者は実験実行時のトラブルシューティングのための情報検索や、メールでの状況報告などのためにインターネットなどへの接続や、設備の情報など実験設備により提供されている情報へアクセスしたい場合が多い。このような用途のために、管理用、実験用のネットワークと分離した生活用ネットワークが必要である。図 5.1 に提案するトポロジのイメージを示す。3 つのネットワークは基本的には分離されるべきであるが、ファイルサーバを共有することで実験実行の容易さを向上させることができる。実験に必要となるファイルを、実験トポロジ構築前にファイルサーバに保存し、実験トポロジ構築後に実験用ネットワークを通じてファイルサーバからダウンロードするといった利用が可能となる。また、実験用ネットワークと管理用ネットワークは基本的には外部のネットワークとは切り離されているべきであるが、実験を複数の実験設備と接続して行う場合などのために、外部接続を行える機能が必要である。外部接続は標準的に提供するのではなく、実験の実行者からの要求を受けて適切に接続設定がなされるべきである。

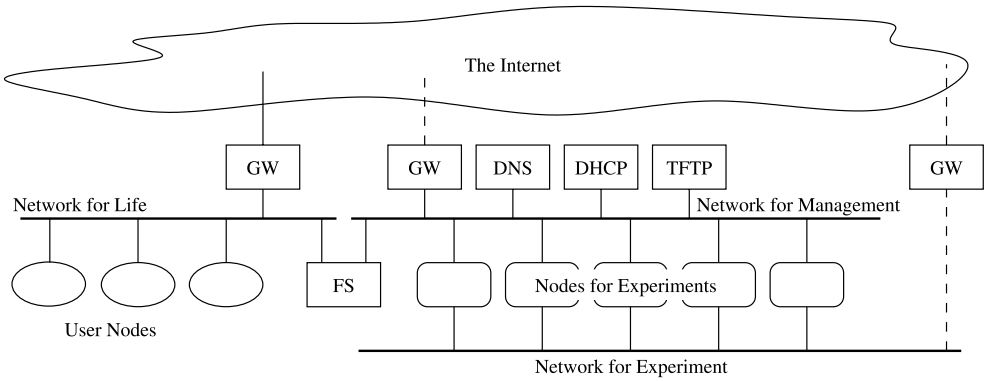


**図 5.1.**　設備に求められるトポロジイメージ

### 5.5 外部からのノードのコンソール操作

すでに述べた通り、なんらかの障害により実験用ノードを管理用ネットワーク経由で制御できなくなった場合などには、状況の確認や復旧作業のため実験ノードのコンソールを別の方法で管理したい場合が多い。このため、StarBED のような KVM による制御のほか、シリアル接続や PS2 や USB のエミュレーションなどが利用できる可能性がある。とくにセキュリティ関連の実験のためには、IP ネットワークを用いるとそこから攻撃が可能となってしまうため、シリアルや PS2 エミュレーションなどが有効である。

### 5.6 外部からのノードの起動方法の変更

大規模な実験設備を柔軟に利用するためには、実験の実行者が意図する OS で実験用ノードを動作させる必要がある。また、ノードの制御を行うための専用の OS で起動する場合もある。このような際に柔軟に目的の OS での起動が行える必要がある。これには、すでに OS が導入されているハードディスクのパーティションからの起動や、ディスクレスシステムとしての起動が行える必要がある。StarBED のように DHCP と TFTP、PXE を利用した起動方法の変更の他、ハードディスクに起動制御用の専用パーティションを用意し、常にここから起動した後に利用する OS を変更などの処理も可能である。

### 5.7 外部からのノードの死活管理

電源管理や再起動を含めたノードの死活管理が必要である。WoL での電源投入や、IPMI[67]、iLO[57] を利用した死活管理が利用できる。また前述の通り SNMP やコマンド実行を利用したソフトウェアでの対応も期待できる。

### 5.8 推奨事項

前節で必須事項を述べたが、その他にも実験をより柔軟に行うための推奨事項がある。本節では以下の 2 点について述べる。

- 時間同期
- リンク特性の模倣

各実験ノードのログ情報の同期のために、時間の同期が必要なことがある。また、実験によっては各ノードの時間が正確に同期している必要がある。時刻が実時刻と同期している必要がある場合と、実験用環境上のノードの時間が一致していれば良い場合がある。両者とも NTP を利用したソフトウェアレベルでの同期と、非常に時刻精度の高いノードを利用する方法がある。

実験用環境には、リンクの特性を導入することが一般的である。PlanetLab[130] や Netbed[180] では、実験用環境を分散させ、実環境のリンクを実験用環境中に配置することで現実的なリンク特性を導入している。一方 Modelnet[178] は、ソフトウェアでリンク特性を模倣する dummynet を利用している。また、ハードウェアでリンク特性を模倣するソフトウェアも存在する。このように、対外線を用意し実環境のリンク特性を導入する方法や、専用のハードウェアを導入しておくことでリンク特性の模倣の可能性が広がるといえる。（`wide-paper-deepspace1-mya-dicomo2007-01.txt` 参照）

## 第 6 章　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

StarBED や VM Nebula といった実証環境の開発・運用とその利用、また、さまざまな実験実行者へのインタビューにより実験支援ソフトウェアに必要とされる機能をまとめ、それを実現するためのアーキテクチャを提案した。このアーキテクチャに従った実装を用いることで、実証環境の連携がより容易となり、大規模かつ多機能な実験駆動単位を構築できる。

すべての実証環境がもつ機能は一致している必要はなく、実験により使い分けられるべきである。このような場合に我々がまとめた機能を軸とし、実証環境を比較、選択することが可能となる。（`wide-paper-deepspace1-mya-ipsjjournal200704-00.txt` 参照）

## 第 7 章　実験報告

本章では、Deep Space One ワーキンググループの研究の一環として行った実験内容および結果を報告する。

### 7.1 StarBED および SpringOS の性能評価

StarBED 上で SpringOS を利用して実験を行った場合の、ディスクコピーや実験トポロジ構築、シナリオ実行などの各段階に必要となる時間を計測した。

SpringOS は以下の手順で実験を実行する。

1. 実験の設定記述の読み込み
2. 実験資源の割り当て
3. ノードへのソフトウェアの導入
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シナリオの実行

FreeBSD 5.5 を一台の実験用ノードの 4 Gbyte のパーティションにインストールした。インストールしたパーティションを保存し、これを別のノード群に配布することでノードの複製を行う。このとき利用したノード情報を表 7.1 に示す。

ノードへディスクイメージを配布したのち、VLAN 設定による対象トポロジの構築、実験用シナリオの実行を行い、実験終了後に VLAN の初期化を行った。以上の手順に必要となる以下の項目について実行時間を計測した。

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
   - シナリオの実行
   - VLAN 設定の削除
   - 全体

シナリオ部分は、netperf を利用した帯域測定を行った。netperf はサーバとクライアントからなるアプリケーションで、2 台のノードの間の帯域を測定する。サーバとクライアントのペア数を増やすことで実験を大規模化した。台数を 2 台から 200 台まで変化させ、3 回ずつ各項目について測定を行った。表 7.2 にそれぞれの平均値を示す。

この結果から、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大規模な実験の容易な実行
- 設定記述にしたがった正確な実験駆動単位の構築とシナリオ実行

人手で実験を実行するには時間的に困難であるだけでなく、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、手動での実行に比べ高い精度と再現性が確保できる。

（`wide-paper-deepspace1-mya-wit2007-00.txt` 参照）

**表 7.1.**　実験に利用したノードの情報

| | |
|---|---|
| チップセット | ServerWorks LE |
| CPU | Pentium 3 1 GHz |
| メモリ | 512 Mbyte |
| HDD | IDE 30 GB |
| ネットワークインターフェース | FastEthernet ∗ 2<br>GigabitEthernet ∗ 1 |

### 7.2 StarBED を用いた ICT ストレッサの実現

インターネット上で提供されているサービスの信頼性を考察する上で、ユーザの視点からサービスの安定性を評価することは重要である。そこで、StarBED2 を用いて多数のユーザをシミュレートし、サービスの負荷耐性を評価することを目指し、システムの枠組み構築を進めた。

**表 7.2.**　台数と所要時間（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 |

### 7.2.1 負荷試験技術の方向性

インターネットサービスを提供するサーバの負荷試験を行う手法として、トラフィックジェネレータを用いて計測を行う手法が普及しつつある。トラフィックジェネレータは、ユーザの設定にもとづいて評価対象となるシステムに対して負荷をかけ、その挙動を計測する装置である。近年のトラフィックジェネレータは、その負荷傾向やアクセスパターンを実験者が作成するといった機能を備えるものも増えているが、実験者がトラフィックジェネレータ装置の提供している機能の中での自由度を与えられているに過ぎない。つまり、トラフィックジェネレータで生成できるトラフィックの汎用性と現実性には限界があり、ユーザが意図するトラフィックを完全に生成できているとは言えない。

そこで我々は、StarBED のノードをそれぞれのユーザ端末として動作させ、実験者の作成するユーザクライアント動作をさせることとした。この方式では、実験者は、StarBED ノードで実行可能なプログラムとしてユーザ挙動を記述すればよく、その自由度は非常に大きい。また、要望があれば、実際にユーザが使うオペレーティングシステムや端末プログラム（Web ブラウザなど）のそのものを動作させ、ユーザのキーボード入力やクリック動作を外部から与えるといった、現実環境に非常に近いエミュレーションも可能である。

本年は、とくに、現実的なサーバ負荷試験を実施するためのフレームワークの構築と、その利用例としてソフトウェア配布サーバ負荷試験への適用を行った。本章では、その概略を述べる。

### 7.2.2 提案フレームワーク

#### 7.2.2.1 現実的な負荷の再現

既存のトラフィックジェネレータには、Jakarta プロジェクトで開発されている JMeter[77] や SPIRENT 社 [153] 製の SmartBits や WebAvalanche、さらに Web Polygraph[179] 等がある。SmartBits はパケットを生成し帯域を測定することに特化した機器であり、プロトコルの細かな動作まではサポートしていない。JMeter や WebAvalanche、Web Polygraph は対象とするプロトコルの細かな動作を指定することができるが、それ以外のプロトコルには対応できず、またネットワーク上で実際に用いられているプログラムを用いることは困難である。

しかし、StarBED は実機の集合であり、OS やアプリケーションといったソフトウェアから、実機そのものの構成まで柔軟に変更を加えることができる。そのため、実際のネットワーク上で運用されているさまざまなプログラムや、これから運用することを意図している新たなプログラムをそのまま用いることができる。つまり、実際のネットワーク上の環境をより現実的に再現できるのである。

一方、実機を用いた実験では実験環境が大規模になるほど機器の管理が困難になるという問題があるが、それは SpringOS を用いることで解決できる。以上のような利点を持つ StarBED と SpringOS を利用することで、実機と実際に運用されているプログラムを用いた実環境に近い負荷を容易に実現できる。

サーバに対する現実的な負荷を考えた場合、前述した検証に利用するプログラムの現実性及び柔軟性を確保することに加えて、実際にサーバに与える負荷の特性を模倣することが必要である。負荷の特性として、転送されるデータのサイズやアクセスのタイミング、コネクションの持続時間等、さまざまなものがあげられる。これらの特性はサーバのアクセスログや、パケットモニタの結果から導き出すことが可能である。このようなさまざまな特性の中から必要とする特性を利用することによって、実験者が必要とするだけの現実性を容易に確保できる。

本研究で提案するフレームワークは、StarBED の実機と実際に用いられているプログラムを用い、それを SpringOS で管理し、実際のサーバへの負荷の特徴を再現させることで、サーバへの現実的な負荷を実現するものである。

#### 7.2.2.2 StarBED を用いたサーバ負荷試験の手順

前述の、サーバに対する負荷の現実性についての議論を踏まえ、以下の手順のサーバ負荷試験を提案する。

1. 負荷試験を行うサーバの現状でのログを解析し、サーバへの負荷の特性を調べる
2. 解析したサーバに対する負荷の特性の中から、必要な特性を模倣したアクセスを行う、SpringOS の実験記述ファイルを作成する
3. 作成した実験記述ファイルを用いて、StarBED 上の計算機群を動作させ、サーバ負荷試験を行い、サーバの振る舞いを確認する

### 7.2.3 提案するフレームワークを用いた実験例

#### 7.2.3.1 実験概要

提案するサーバ負荷試験の有用性を検証するための実験例を以下に記す。

実験は、北陸先端科学技術大学院大学で運用されているソフトウェア配布サーバ *ftp.jaist.ac.jp* に対するアクセスを解析し、その負荷の傾向を模倣した負荷を生成するものである。当該サーバは、保持しているソフトウェア等のコンテンツを FTP および HTTP で提供している。通常の HTTP サーバは HTML ファイルを中心とした、比較的ファイルサイズの小さなファイルの転送を目的としているが、当該サーバはアプリケーションのパッケージや OS のイメージファイルなどの大きなファイルを配布する。このため、一般的な HTTP サーバとは異なるアクセス傾向を持っていると考えられる。このようなサーバを実験対象として、インターネット上で新たなサービスを行うサーバに対する本フレームワークの有用性を検証する。

負荷生成用ソフトウェアとしては ccftp を用いた。ccftp は HTTP と FTP を用いてファイルを取得するソフトウェアであり、本実験のために製作した。ccftp は並行セッション数（同時に取得するファイル数）と、繰り返して取得する回数を指定することが可能である。

#### 7.2.3.2 実験トポロジ

実験に用いたトポロジの概念図を図 7.1 に示す。管理用ノードは SpringOS を用いて実験に用いる StarBED 上の各クライアントに処理を実行させるために存在する。管理用ノードから命令を受け、クライアントは HTTP、FTP の各サーバからファイルの取得を行う。

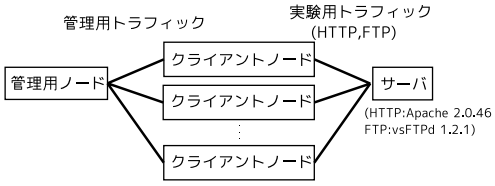


**図 7.1.**　実験トポロジ

#### 7.2.3.3 実験の手順

実験の手順を以下に示す。

1. サーバのログ解析
   現実的なサーバ負荷試験を実現するために、実験対象であるソフトウェア配布サーバの現時点でのアクセスログを取得し、アクセスの傾向を知るための解析を行った。
2. HTTP サーバ単体での負荷試験
   解析したサーバへのアクセス傾向をもとに、実環境で取得されるファイルのサイズの比率を模倣した HTTP サーバへのアクセスを行った。並行セッション数を変動させた実験を複数回行い、取得に成功する率を確認した。
3. FTP サーバ単体での負荷試験
   解析したサーバへのアクセス傾向をもとに、実環境で取得されるファイルのサイズの比率を模倣した FTP サーバへのアクセスを行った。並行セッション数を変動させた実験を複数回行い、取得に成功する割合を確認した。
4. HTTP、FTP 混在型の負荷試験
   HTTP 及び FTP サーバ単体での負荷試験での知見をもとに、SourceForge ダウンロードサーバの特徴である HTTP と FTP によるアクセスが混在した負荷を模倣した。HTTP と FTP のアクセスの比率は、前述のアクセスログを解析した結果を使った。

#### 7.2.3.4 ftp.jaist.ac.jp のログ解析

解析したログファイルは、HTTP と FTP の各プロトコルに対する 1 日分のアクセス情報を記録したものである。以下にそのログファイルを実際に解析した結果を示す。

まず、各プロトコルによるアクセスの回数は、HTTP によるアクセスが 582918 回であり、FTP によるアクセスが 336371 回である。つまり HTTP が全体のアクセスの約 63%であり、FTP によるアクセスが全体の 37%である。HTTP と FTP が混在する負荷を再現するために、この HTTP と FTP で取得したファイルの比率をスケーリングして負荷を生成する。

さらに、各プロトコルでダウンロードされるファイルのサイズを解析した。図 7.2 では HTTP と FTP の各プロトコルで取得されたファイルサイズを、常用対数で階級化した確率密度とその累積分布で示したグラフである。たとえば、横軸が 3 のとき、取得したファイルサイズは 100 から 999 バイトとなる。このグラフの横軸はファイルサイズ、縦軸は各サイ

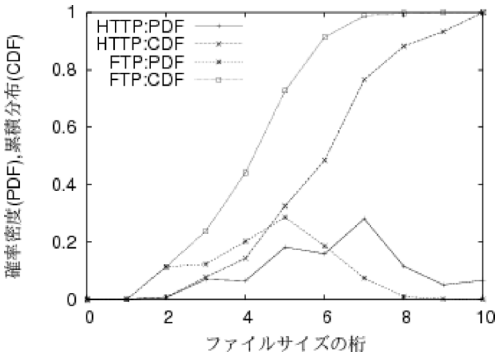


**図 7.2.** 取得されたファイルサイズの分布

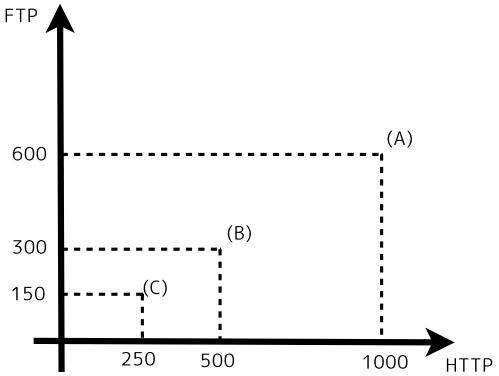


**図 7.3.** HTTP・FTP 混在：並行セッション数

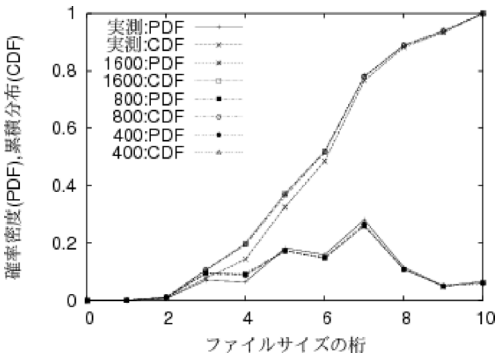


**図 7.4.** HTTP・FTP 混在：HTTP のファイルサイズ分布

ズのファイルが全体に占める割合を示している。これから、FTP よりも HTTP の方がサイズの大きなファイルを取得するために用いられていることがわかる。これは、当該サーバは CD や DVD のディスクイメージを提供していることと、近年 HTTP で配布されるパッケージが増加しているからだと考えられる。

実験対象であるサーバのアクセスログの解析結果より、同時にサーバに対して行われるアクセス要求は 1 秒あたり平均 HTTP で約 6.7 回、FTP で約 3.8 回であり、それほど大きな数ではない。それに対し、取得されるファイルサイズが大きいため、ファイルの取得に時間がかかり、サーバに対する並行セッション数が大きいと考えられる。そのため、サーバに対する並行セッション数に注目したサーバ負荷試験を行うこととし、サーバへの並行セッション数を増減させ挙動を確認した。

#### 7.2.3.5 HTTP と FTP の混在した負荷試験

HTTP、FTP サーバの単体での負荷試験の結果より、HTTP サーバへの並行セッション数は約 1000 が限界であることがわかった。このことと、前述した実際のサーバに対する HTTP と FTP のアクセスの比率を利用して、HTTP・FTP の両方を提供するサーバに対して HTTP と FTP の 2 つのプロトコルによるアクセスが混在する負荷試験を行った。

前述した通り、HTTP と FTP それぞれのプロトコルによるアクセスの比率は約 63 対 37 であった。そのため、試験に用いる計算機は 100 台とし、そのうち 63 台が HTTP によるアクセスを担当し、37 台が FTP によるアクセスを担当することとした。

図 7.3 はこの負荷試験において各プロトコルの並行セッション数を示したものである。並行セッション数は、HTTP によるアクセスの限界である 1000 を基準とした。各計算機で同時に 16 個のファイルを取得することとし、全体での並行セッション数を 1600 から始め（図中 A）、以降サーバの振る舞いにしたがって各計算機で同時に 8 個（図中 B）、4 個（図中 C）と並行セッション数を減少させた。また、1 回の実験でのセッション数の合計は 16000 とした。

図 7.4、図 7.5 は並行セッション数を変化させた各実験で取得したファイルサイズの分布を HTTP と FTP に分けて示したものである。図 7.4 は HTTP で取得したファイルサイズの比率であり、図 7.5 は FTP で取得したファイルサイズの比率を示している。

表 7.3 は、並行セッション数に対する、HTTP、FTP の各プロトコルの成功率と、全体に占める各プロトコルの比率である。並行セッション数が増加するほど、FTP による取得が失敗していることがわかる。そのため、並行セッション数が増加するほど HTTP で取得されるファイルの比率が上がっている。

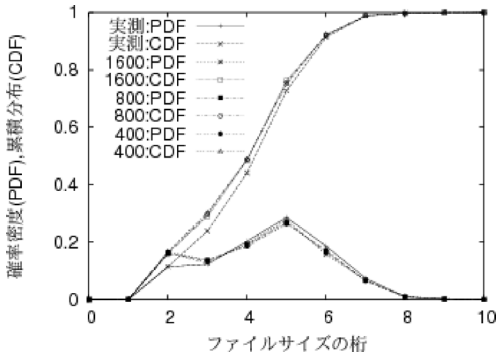


**図 7.5.** HTTP・FTP 混在：FTP のファイルサイズ分布

**表 7.3.** HTTP・FTP 混在：並行セッション数と成功率

| 並行セッション数（個） | | 400 | 600 | 1500 |
|---|---|---|---|---|
| 成功率（%） | HTTP | 100 | 100 | 96.0 |
| | FTP | 98.2 | 78.5 | 62.5 |
| | 全体 | 99.3 | 92.1 | 83.6 |
| 比率（%） | HTTP | 63.4 | 68.5 | 72.4 |
| | FTP | 36.6 | 31.5 | 27.6 |

### 7.2.4 考察

StarBED を用いて北陸先端科学技術大学院大学のソフトウェア配布サーバに対するアクセスの傾向を模倣したアクセスをし、本章で提案したフレームワークの検証を行った。その結果、HTTP、FTP それぞれのサーバと、HTTP、FTP が同時に提供されているサーバに対する取得サイズの分布を再現することができた。また、HTTP、FTP を混合させたサーバへの負荷試験の結果、各プロトコルの比率を模倣したアクセスも行えることが確認できた。

さらに、サーバ負荷試験としての結果として、HTTP 及び FTP サーバが提供できる並行セッション数の限界を確認することができた。また、HTTP と FTP が混在するアクセスを模倣した結果、HTTP による並行セッション数が増加することで、FTP の並行セッション数の限界が小さくなるという FTP 単体での試験では見出せないサーバの振る舞いを観測することができた。

また、通常の実機を用いてサーバへの負荷試験を実現する場合、さまざまな処理を手作業で行わなければならないが、StarBED/SpringOS を用いることで、多数のクライアントへの命令をほぼ自動で行えるようになっている。これによって、多数のクライアントからのサーバへの一斉アクセスなどの手動では困難な処理も実現している。

これらのことから、StarBED を用いたサーバ負荷試験の現実性と有用性を確認できた。

（`wide-paper-deepspace1-nonayou-dicomo2007-00.txt` 参照）

## 第 8 章　おわりに

本報告書では、今年度の Deep Space One ワーキンググループの活動報告を行った。本年度は、StarBED や GARIT のような実験設備や、SpringOS や AnyBed といった実験設備を制御するための支援ソフトウェア自体の研究だけでなく、複数の実験設備を協調させるための実験設備やソフトウェアのアーキテクチャについての議論を行った。また、これまで実験の実行段階に主眼をおいての研究活動をすすめてきたが、本年度からは、実行段階の前にあるべき計画段階を支援するためのトポロジ生成や、実験の信頼性を向上させるための実環境のトポロジの再現などの研究にも取り組んできた。

今後は、実験の実行自体の支援手法の研究を引き続き行うとともに、実験の計画段階および、実験後の実験結果の解析支援についての議論や、実験結果の現実性をより向上させる技術についての研究を進める。また、それに平行し、さまざまな実験を我々の環境で実行することにより、その結果を実験設備・支援ソフトウェアの開発にフィードバックするだけでなく、さまざまな実験例をテンプレートとして収集していく。