

StarBEDにおける自動実験遂行機構

宮地 利幸[†] 知念 賢一[†] 篠田 陽一[§]

北陸先端科学技術大学院大学 情報科学研究科[†] / 情報科学センター[§]

Automatic Experiment Execution System on the StarBED

Toshiyuki Miyachi[†] Ken-ichi Chinen[†] Yoichi Shinoda[§]

SCHOOL OF INFORMATION SCIENCE[†] / CENTER FOR INFORMATION SCIENCE[§],
JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY

概要

インターネットに導入されるソフトウェアを開発するためには、その挙動を検証するための環境が必要となる。実験環境の構築のためソフトウェアシミュレータが広く利用されているが、多くのソフトウェアシミュレータではインターネットに導入される実装そのものを利用することはできず、実ノードによる実験環境が必要となる。

実ノードによる実験環境で、利用者が想定する実験手順を手動で実現するのは様々なコスト面から困難である。我々は、利用者が指定した実験手順にしたがって自動的に実験を駆動するためのシステムの設計を行った。またこのシステムを実装し StarBED 上で動作検証を行うことで、その挙動を確認した。

1 はじめに

新たな技術をインターネットに導入する際には、すでに運用されているサービスへの影響を防ぐため、最低限の検証を行う必要がある。現在のインターネット上では様々なサービスが行われており、未検証の技術の導入によるサービスの停止といった危険は回避されなければならない。新たな技術の検証用途にはソフトウェアシミュレータが広く利用されているが、多くの場合ソフトウェアシミュレータは、シミュレーション専用のプログラムコードを要求することが多い。したがってソフトウェアシミュレータでの検証は、実際にインターネットに導入されるソフトウェアの検証ではない。実際にインターネットへ導入されるのは、ソフトウェアシミュレーション用のプログラムコードではない。実際にインターネットに導入される実装のバグや仕様に記述されていない動作までを含めた検証が必要であり、このような実装の検証を行う場合は、実ノードを用いて実験環境を構築し、その上に被検証対象を組み込んで実験を行うことになる。

一般的に実験は、実験の手順(シナリオ)にそって進められる。ソフトウェアシミュレータを利用する場合には、利用者は前もってシナリオを記述しておくことで、実験中は別の作業を行うことができ、その拘束時間を回避できる。また、これにより、再度同様の実験を行うこともでき、実験の再現が可能であるというシミュレーションの長所の一つの要因にもなっている。

実ノードを用いた実験環境を利用する際にも、利用者の指定通りに実験ノード上でプログラムを起動する仕組みが必要である。手動で各プログラムを実行することもできるが、利用者は実験が実行されている間、常にシナリオにそってコマンドを入力せねばならないことになり、実験に拘束されることになる。また、実験ノードの数が多ければ多いほど、各実験ノードの制御は複雑になり、プログラムの実行ミスなどが当然発生する。また、実験の再現も困難である。

本論文では、ユーザにより指定された通りのタイミングで自動的にプログラムを起動するシステムの

一つとして、StarBED[1][2]で利用されている実験の自動遂行プログラムの設計について述べる。

2 実ノードによる実験環境

まず本章で、我々の提案する自動実験遂行機構が前提とする実ノードによる実験環境について述べる。

2.1 実ノードによる大規模な実験環境の必要性

ソフトウェアシミュレータを用いれば、大規模な実験環境を容易に模倣することができるが、ソフトウェアシミュレータは、ネットワークや各実験ノードの挙動を巨視的に模倣するものであり、ネットワーク上の様々な要素の詳細な動作の検証は行えない。また、実際にネットワーク上で動作している実装を利用できないことが多く、ソフトウェアシミュレータ上で観測された実験の結果と実ネットワーク上で挙動が異なる可能性がある。

実際にインターネットに導入するソフトウェアやハードウェアの実装の検証を行うためには、実ノードにより構築された実験用のネットワーク上で、対象となる実装を動作させる必要がある。

NSE[4]のように、ソフトウェアシミュレータと実ノードによるネットワークを接続し、実験環境を構築する手法も提案されている。しかし、検証対象のアルゴリズムや実装がどのような外部要因により影響を受けるかを予測することは困難であるため、両者を実験ネットワークのどの部分に割り当てるかを決定することも困難である。

一般的に、実験対象となる実装の挙動や、その実装による他のサービスへの影響を前もって知ることが困難なため、できるだけインターネットに近い環境で実験を行うことが重要である。対象となる環境としてインターネットを想定した場合は、非常に多くの実ノードにより構築された環境が必要となる。

また、ソフトウェアシミュレータではハードウェア実装の検証は当然行えないが、実ノードによる実験環境ではこれも可能となる。

2.2 StarBED

実ノードによる大規模な実験を行うための施設の一つとしてStarBEDがある。StarBEDには512台の実験用ノードが用意されており、それぞれのノードはスイッチにより接続されている。実験用のトラフィッ

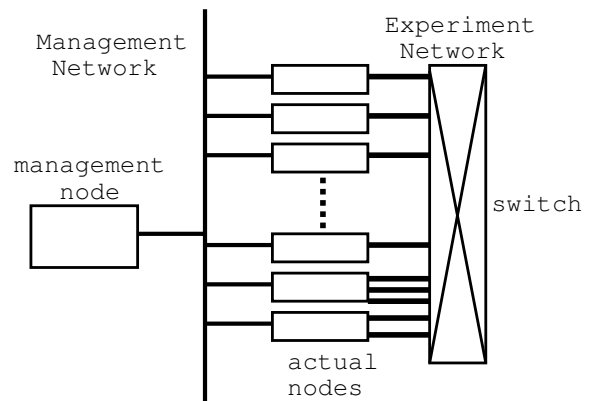


図 1: StarBED の概念図

クと管理用トラフィックを分離するため、各ノードには最低2つのネットワークインターフェースが用意されており、実験用ネットワークと管理用ネットワークに接続されている。StarBEDの概念図を図1に示す。管理側のネットワークには実験ノードを管理、観測するための管理用ノードが接続されており、我々の提案する実験遂行機構も管理用ネットワークに接続されているノードで動作することを想定する。

StarBEDでは、実験トポロジは、物理的なトポロジに変更を加えることなく、VLANなどを利用し仮想的に構築される。物理的なトポロジを変更しないことで、実験環境構築に必要な手順を節減し、施設を利用できる時間が長くなる。

2.3 実ノードによる実験環境での実験遂行手順

一般的な実ノードによる実験環境での実験の遂行手順を以下に示す。

- 1) **ノードの物理配線** 実ノードを用意しケーブルの配線など物理的な接続作業を行う。
- 2) **実験ノードとネットワークの設定** 実験に必要なOSやアプリケーションを実ノードにインストールし、IPアドレスの設定など、実験に必要な設定を行う。また、必要であればスイッチの設定なども行う。
- 3) **実験の遂行** 実験者が意図するシナリオにそって実験を遂行する。
- 4) **ログの収集および解析** 実験の結果ログを各実験

ノードやスイッチから収集し解析を行う。

本論文では、1)、2)についてはすでに完了していることを想定し、3)の実験の遂行部分に着目し、指定されたシナリオにそって自動的に実験を行うシステムの設計について述べる。

3 実験の自動駆動

本章では、実験を自動的に遂行するためのシステムについて議論を行う。

3.1 現状での問題点

多くのソフトウェアシミュレータでは、利用者によって計画されたシナリオの記述にしたがって実験が遂行される。しかし、実ノードを用いた実験環境には、このようなシステムが存在しない。実験をシナリオ通りに遂行するシステムが存在しない場合、利用者がそれぞれの実験ノードにログインしてプログラムを起動する方法や、リモートノードから `rsh`[6] や `ssh`[7] を利用してプログラムを起動する方法をとることになる。以下にその問題点を挙げる。

利用者の時間的コストの増大 利用者は実験実行中、常にコマンドの入力をする必要があるため、実験終了までの間、実験に拘束される。

再現性の低下 利用者の手によってコマンドを実行するため、全く同じ実験を行う場合でも、同じ時刻にコマンドを実行することは困難である。したがって2回の実験での差異が大きくなり、実験の結果へ影響を与える可能性がある。

実行誤り 利用者が一つずつコマンドを入力するため、予定していた手順を誤る可能性がある。この場合、誤りの存在自体を知ることが困難である。

ノード間同期 あるノードの挙動をトリガとして、別のノードで処理を開始したい場合に手動やスクリプトでの実行では、各ノード間でタイミングをあわせることは困難である。

我々は以上の問題点を解決するためのシステムを設計した。

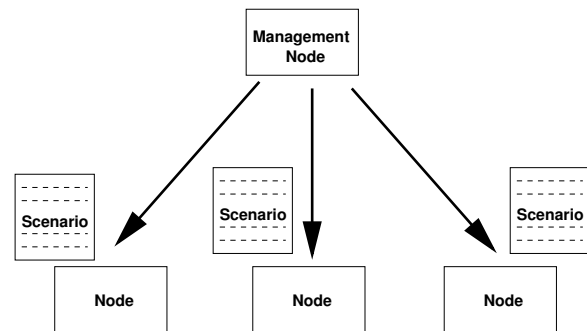


図 2: ノード自律モデル

3.2 モデル設計

実験のシナリオは前もって利用者により記述されることを前提とした。利用者は実験環境を構築およびシナリオの記述のみを行い、実験の遂行自体は我々のシステムにまかせることで、実験に拘束される時間を削減できる。

シナリオにそって各実験ノードのプログラムを起動するための方式について検討した。以下では検討した各方式について述べる。また、シナリオの設定ファイルを読み込み、シナリオ全体を把握しているプログラムをシナリオマスタと呼ぶ。

ノード自律モデル 実験前にシナリオをシナリオマスタから各実験ノードに配布し、各実験ノードは配布されたシナリオにそって自律的にプログラムの起動を行うモデルである。このモデルでは、各実験ノードが自律的にプログラムを起動するため、各実験ノード間の同期が困難である。ノード間でメッセージを交換し同期をとる方法も考えられるが、それぞれのノードへのメッセージ送出手のタイミングや、受け取ったときの処理をシナリオにまとめて記述することになり混乱の元になりやすい。また、この方法では各実験ノードがメッセージ交換が必要となる別ノードへのセッションをその台数分保持する必要があり、実験自体に影響を及ぼす可能性がある。図2にノード自律モデルを示す。

コマンド送信モデル シナリオマスタからプログラム実行のタイミングごとに、実行するコマンドを各実験ノードに送信する。各実験ノードは受信

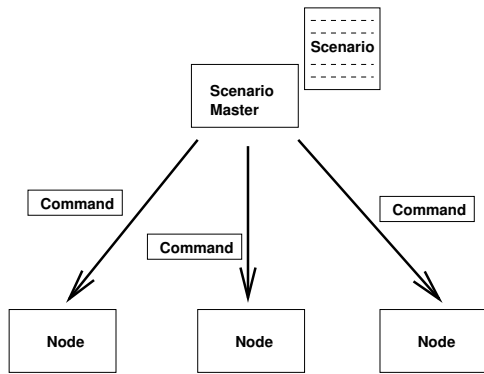


図 3: コマンド送信モデル

したコマンドを単に実行するというモデルである。このモデルでは、サーバは実験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、大規模な環境になるほどシナリオマスタのセッションの管理のための処理負担が大きくなり実験へ影響をおよぼす可能性がある。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、シナリオマスタの負荷が大きくなり、設定されたタイミングと実際にコマンドが実行されるタイミングに無視できない差異が生じる可能性がある。シナリオマスタと、実験ノードがメッセージを交換することで、実験ノード間の同期をとることもできるが、この処理を行った場合はさらにシナリオマスタの負荷が高くなる。各ノードが直接メッセージを送り同期をはかる方法もあるが、ノード自律モデルと同様の問題が発生する。このモデルを図3に示す。

これらを踏まえて、我々は前述した2つの手法の折衷案を採用した。我々のモデルでは、シナリオマスタは実験前に各実験ノード用のシナリオを配布し、実験ノード間の同期のみシナリオマスタを通して行う。OSの導入や設定が済んだ実験ノードは、シナリオを受け取るとそれを実行し、シナリオ記述にそってシナリオマスタへのメッセージの送信や、シナリオマスタからメッセージを受信するまで、シナリオ遂行を停止するといった処理を行う。また、シナリオマスタは専用のシナリオを持っており、実験ノードからのメッセージを受信するまでのシナリオ遂行

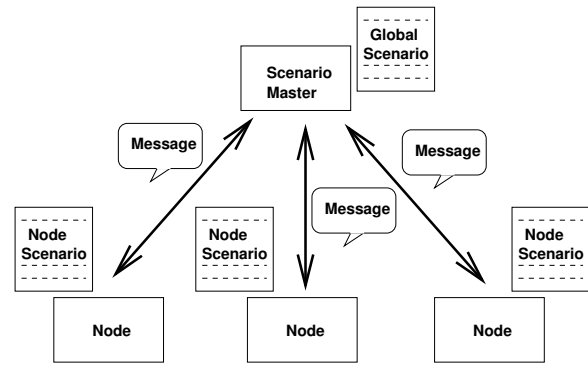


図 4: 提案モデル

の停止や、実験ノードへのメッセージの送信を行う。シナリオマスタのシナリオを実行することにより、各実験ノード間の同期をとる。

このモデルでは、実験ノード間同期が可能となるばかりではなく、実験ノード間の同期が必要な時のみ、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シナリオマスタの負荷が軽減できる。

また、シナリオをそれぞれの実験ノードへ個別に入力するのは非常に時間がかかる作業であるが、設定ファイルをシナリオマスタで一括して読み込み、各実験ノードごとのシナリオに分割して実験ノードへ送ることでこの時間も短縮できる。このモデルを図4に示す。

シナリオの同期機構を考慮にいれ設定言語の設計を行った。基本的には設定ファイルに実行するプログラムと実行されるタイミングを列挙する形とする。タイミングは別ノードの挙動をトリガとする方式および、時刻をトリガとする2方式をサポートすることとする。これにより、より柔軟な実験遂行が期待できる。ns-2[3]は広く利用されているソフトウェアシミュレータであり、ns-2に採用されている設定記述を用いれば、これまでソフトウェアシミュレータにて行われていた実験をそのまま実ノード環境に移行できるが、この記述方法ではトリガにノードのイベントを利用できない。このため、独自の新たな言語の設計を行った。

3.3 言語設計

言語は利用者にわかりやすくシンプルであるということをお頭に置き設計した。シナリオ記述には2種類あり、1つは各実験ノードで実行するためのシナリオとシナリオマスタのシナリオがある。実験ノード用のシナリオをノードシナリオ、シナリオマスタ用のシナリオをグローバルシナリオと呼ぶ。基本的に、ノードシナリオは実験の遂行のためのものであり、グローバルシナリオは実験ノード間の協調のために利用する。

ある程度の規模のネットワーク実験を行う場合、各実験ノードはその役割により、グループに分けられると考えた。同じグループに属する実験ノードに同様のシナリオを個別に記述するのは冗長であるため、基本的にノードの属性はグループごとに定義することとする。ノードシナリオでは基本的に、プログラムの実行と別の実験ノードとの同期のためのシナリオマスタへのメッセージ送出、及びノードマスタからのメッセージを待ちシナリオ遂行の停止を行う。またグローバルシナリオでは、実験ノードへのメッセージの送信および、複数の実験ノードからのメッセージを待機してのシナリオ実行の停止を行う。

ここで、実験ノード2台を用意し一方からもう一方の実験ノードに対して ICMP echo request を送信するというだけの簡単な例を示す。この例の動作手順を以下に示す。

1. 実験ノードがシナリオを受け取るとシナリオマスタに対して、“setupdone”というメッセージを送る。
2. これを受信したシナリオマスタは、実験ノードに対して“start”というメッセージを送信する。
3. このメッセージを受信した実験ノードは、別の実験ノードに対し ping request を10回送信する。
4. 実験ノードは ping が終了するとシナリオマスタに対して“finished”というメッセージを送出する。

図5にこの例のノードシナリオを示す。ただし、前述の通り本システムは、あらかじめ実ノードで構築された環境上に、VLANなどで仮想的に実験環境

```

1 nodeclass clclass {
2   scenario {
3     send "setupdone"
4     recv msgfromsrv
5     msgswitch msgfromsrv {
6       "start" {
7         wakewait "ping" "ping" \
8           "-c" "10" "somehost"
9         send "finished"
10      }
11    }
12  }
13 }

```

図5: ノードシナリオの例

を構築し、この実験環境でシナリオを実行するという大きなシステムの一部として設計を行った。したがって、本来ノードクラスの設定にはシナリオ以外にも様々な設定が存在するが、本論文はシナリオについてのみ着目し、その他の部分については省略している。

実験ノードの集団は `nodeclass` を用いて宣言される。ブレースで囲まれた部分が `nodeclass` の属性の定義である。ノードシナリオ部分で、`send` を用いると自動的にシナリオマスタ向けにメッセージを送信する。また `recv` でシナリオマスタからのメッセージを待ち、メッセージを受け取るとその後指定した変数にメッセージを格納する。`msgswitch` で格納されたメッセージ内容により挙動を変更できる。`wake` および `wakewait` はプログラムを起動するために利用され、`wake` が子プロセスを起動しプログラムを実行させるのに対し、`wakewait` は自分自身でプログラムを起動しその終了を待つ。実験ノードは2台必要であるが、1台は ping に応答し ICMP echo reply を返すだけでいいため、特にシナリオは用意していない。この例のグローバルシナリオを図6に示す。

シナリオマスタは、実験ノードから“setupdone”というメッセージを受信するまでシナリオの遂行を停止し、このメッセージが届くと“start”というメッセージを実験ノードに対して送信する。その後、実験ノードから“finished”というメッセージが届くまで待機する。

```

1 scenario {
2   recv pingclient[0] msgfromcli
3   msgswitch msgfromcli {
4     "setupdone" {
5       send pingclient[0] "start"
6       sync {
7         msgmatch pingclient[0] "finished"
8       }
9     }
10  }
11  exit
12 }

```

図 6: グローバルシナリオの例

図 6 の 6 行目から 8 行目で宣言されている、`sync` と `msgmatch` を用いると、複数の条件がそろうまでシナリオの遂行を停止することができる。この 2 つのキーワードはグローバルシナリオに記述される。`sync` はそのあとにブレースで囲まれたブロック構造を持ち、ブロック内に記述された条件がすべて整うまでシナリオを停止する。`msgmatch` は `sync` のブロック構造中に記述されることが多く、実験ノードからのメッセージを待つ。

4 検証

我々は 3 章で提案したモデルにしたがった自動実験遂行機構を実装し、StarBED 上で評価実験を行った。結果が容易に想像できる実験を行い、その結果が正しいかを検証した。この実装は UNIX 系 OS 上で動作し、スイッチやルータなどで動作する専用ファームウェアは、開発環境が提供されていないことが多いため、現在は対象外としている。しかし本モデルの仕組み自体は簡単なものであるため、実装さえできればスイッチやルータ上でも利用できると考えている。

4.1 検証実験内容

帯域ベンチマークプログラムである `netperf[5]` を用いて 2 台の実験ノード間の帯域を測定した。この時利用した設定ファイルを図 7 に、また実験の流れを図 8 に示す。この例でも、図 7 中の実験ノードの IP アドレスやディスクイメージをインストールするための設定部分は省略した。

25、26 行目のキーワード `nodeset` で `nodeclass` で定

```

1 nodeclass svclass {
2   ...
3   scenario {
4     wake "/sim/netserver" "/sim/netserver"
5     send "serverstarted"
6     recv msg
7     msgswitch msg {
8       "quit" {
9         wakewait "/usr/bin/pkill" "\\
10          "pkill" "netserver"
11          exit
12        }
13      }
14    }
15  }
16 nodeclass clclass {
17   ...
18   scenario {
19     send "clisetupdone"
20     recv dst
21     wakewait "/sim/netperf" "/sim/netperf" "-H" dst
22     send "cdone"
23   }
24 }
25 nodeset client class clclass num 1
26 nodeset server class svclass num 1
27 ...
28 scenario {
29   sync {
30     msgmatch server[0] "serverstarted"
31     msgmatch client[0] "clisetupdone"
32   }
33 }
34 send client[0] haddr(server[0].netif[0].ipaddr)
35 sync {
36   msgmatch client[0] "cdone"
37 }
38 send server[0] "quit"
39 exit
40 }

```

図 7: 検証用設定ファイル例

義したノードを実際に何台用意するかを指定する。25 行目のクライアントの宣言文を例にとると、`clclass` のノードを `client` という名前で 1 台生成している。`nodeset` によって生成されたノードは、宣言された際に設定された名前とブラケットで囲まれた数字によって表現される。たとえば 31 行目のように `client[0]` と指定される。その他の、シナリオの記述は既に説明したため省略する。

この実験では、`netperf` のサーバ 1 台とクライアント 1 台の 2 台を用いる。サーバが起動すると `netperf` のサーバプログラム `netserver` を起動し、実験ノードが起動すると `netperf` のクライアントプログラムである `netperf` を実行する。この時 `netperf` を実行するタ

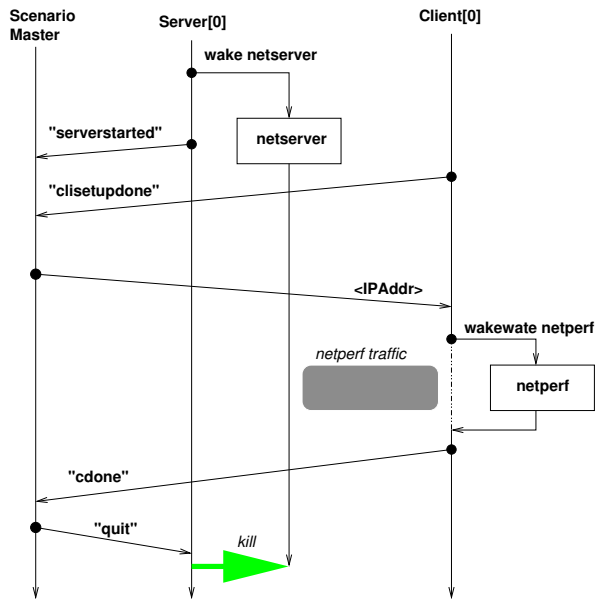


図 8: 検証実験の流れ

イメージなどを、シナリオマスタを通したグローバルシナリオを用いて調整している。このシナリオでの動作を以下に示す。

1. サーバが起動し `netserver` を起動するとシナリオマスタに対して "serverstarted" という文字列を送信する (4 行目)。
2. クライアントが起動するとシナリオマスタへ "clisetaupdone" という文字列を送信する (18 行目)。
3. シナリオマスタはこの 2 つのメッセージの受信を待ち (29 行目から 32 行目)、クライアントにサーバの IP アドレスを通知するメッセージを送信 (34 行目) する。
4. これを受信したクライアントは指定された IP アドレスのサーバへ向かって `netperf` を実行する (19 から 20 行目)。
5. `netperf` が終了するとクライアントはシナリオマスタに "cdone" という文字列を送信 (21 行目)。
6. この文字列を受信したシナリオマスタはサーバへ対して "quit" という文字列を送信 (38 行目) する。

7. このメッセージを受信したサーバは `netserver` プロセスを終了する。

4.2 検証結果

利用した実験ノードは 100BaseTX のリンクで接続されており、帯域が 90Mbps 程度という実験結果が得られた。このことから、各実験ノードでのプロセスの起動が成功しており、また実行結果に対する影響も小さいと考えられる。この実験 100 台規模の実験も行い、我々のシステムが正しく動作することを確認している。利用者は設定ファイルを記述するだけで良いため、拘束時間も大幅に削減できた。

5 今後の課題と制限事項

我々のシステムは基本的に実験環境上の PC を操作することを想定しており、一般的な OS が動作しないようなノードを操作することはできない。しかし、ノードの協調は簡単なメッセージパッシングによって行われているため、PC 以外のノード上でも簡単に実装できると考えられる。

現在、本機構により、一般的な実験の駆動は行えるが、シナリオマスタで一度に一つの同期のみしか扱えず、複数のコンテキストによる同期を同時に行えないため、複数のコンテキストを同時に扱う場合は、シナリオ記述が複雑になってしまう。また、実験実行中にノードが故障した場合などの、例外処理についても未検討である。今後は以上を扱えるような言語の拡張を進めていく。また、その他の実験支援システムとの協調についても、今後拡張を続けていく予定である。

6 まとめ

インターネットに投入される技術は、最終的に投入される実装自体の検証が行われなければならない。このような実験を行う際には実ノードによる実験環境に、実ノードの検証対象を組み込んで実験を行う必要がある。このような要求を満たすため、実ノードによる大規模なネットワーク実験環境および、その環境での実験を支援するためのシステムが求められている。

本論文では支援システムのうち、前もって用意された実験のシナリオを自動的に遂行するためのシステムのモデルについて議論を行った。その結果、我々

は各実験ノードのシナリオを前もって各実験ノードに配布、実験ノードは配布されたシナリオを実行し、シナリオマスタにクライアントがメッセージを送信することにより、クライアント間の協調をとるモデルを採用した。またシナリオ実行の提案モデルを充足するための、設定言語を提案した。最後に提案した手法を実装し、StarBED において実験を行いその有効性を確認した。

参考文献

- [1] StarBED プロジェクト. <http://www.starbed.org/>.
- [2] 独立行政法人 情報通信研究機構 北陸 IT 研究開発支援センター. <http://www.hokuriku-it.nict.go.jp/>.
- [3] The Network Simulator -ns-2. <http://www.isi.edu/nsnam/ns/>.
- [4] Kevin Fall, Network Emulation in the Vint NS Simulator, ISCC, Dec 1988
- [5] the Public Netperf Homepage.
- [6] "B.Kantor", RFC1282, BSD Rlogin, RFC1282, December 1991
- [7] T. Ylonen, SSH - Secure login connections over the internet, Proceedings of the 6th Security Symposium USENIX Association: Berkeley, CA.