CL-004

ns-3 シミュレーションを生成するグラフィカルツール
Graphical tool for generating ns-3 simulation

小野貴弘 †　平中幸雄 †　武田利浩 †
Takahiro Ono　Yukio Hiranaka　Toshihiro Taketa

1　はじめに

現実の世界では様々なネットワークシステムが動作している。これらネットワークシステムの性能評価にはネットワークシミュレータが用いられる。ネットワークシミュレータの中でも ns-3[1]というネットワークシミュレータはフリーソフトで（GNU GPLv2 license の下で）自由に使用することができ、学術的な研究機関で利用されることが多い。しかし、ns-3 はシミュレーションの準備に多くの時間と労力を必要とする。ユーザはシミュレーションを行うためだけに ns-3 の仕様・コーディング方法を理解する必要がある。単にシミュレーションを行うことが目的のユーザにとっては、ns-3 シミュレーションを自動的に生成するようなツールが期待される。

本研究では、容易にシミュレーションを生成可能とするためのグラフィカルツール(TGIM)を開発する。

2　シミュレーション生成

一般に ns-3 を用いるワークフローは、Figure 1 に示すようになる。ns-3 は、主にシミュレーション時のモデルの提供・実行・ログの出力を行うが、それ以外は手動か、外部の支援ツールを用いて行う。ns-3 は統合開発環境(IDE)を提供していないため、ユーザはシナリオを手動で管理しなければならない。また、出力データの解析は外部のツールを利用して行うことになる。

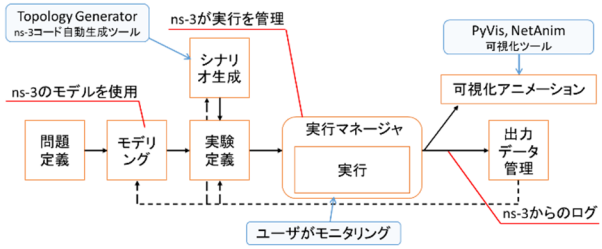

Figure 1. ns-3 を用いたシミュレーションのワークフロー

ユーザは ns-3 の指定する C++言語でシナリオを記述する。しかし、C++言語および ns-3 ライブラリの仕様を習得するのは容易ではない。そこで C++を用いずにシナリオを記述できる方法がないか調査した。

調査の結果、Topology Generator[3](T.G.)というツールを知った。これはストラスブール大学の Pierre Weiss 氏と Sebastien Vincent 氏によって作成された GUI で視覚的にトポロジ作成を支援するツールである。ns-3 には外部ツールと連携するための機能が不足しているため、シミュレーションを自動的に生成するには、ns-3 のコードを直接生成する方法がとられる。T.G.も例外ではなく、GUI の入力情報から C++で記述された ns-3 シナリオのコードを生成する。当初、このシステムを改良し、ユーザがシミュレーションをより容易に生成できるようにしようと試みた。しかし、その T.G.では、内部での機器・接続情報の管理方法が複雑、生成されたシナリオのコードをそのまま ns-3 で実行できず、コードを手動で修正する必要があるなどの、多くの課題があることがわかった。

特に、生成されたシナリオのコードをそのまま ns-3 で実行できないというのは大きな問題である。その解決には大きな困難があったため、新たなシミュレーション生成ツールを開発することにした。このシステムを TGIM (Topology Generator IMproved)と名付けた。

単にシミュレーションのためのコードを生成することが目標ではない。さらにその先に進んだ、シミュレーション全体のワークフローを支援することが最終的な目標である。これを念頭に置き TGIM を開発した。

また、本研究では有線系ネットワークシミュレーションを当面の対象とし、無線系ネットワークシミュレーションについては後に検討する。

3　システム設計の概要

シミュレーションを容易とするためには、シナリオの表現方法が非常に重要になる。はじめに、シナリオを表現するための方法として、ネットワークトポロジ記述に特化したドメイン固有言語(DSL)を設計した。この DSL を原則として TGIM を構成した。

Figure 2 に新たに開発したシステムの構成を示す。TGIM は以下のツールから成る。

(1) **ネットワークジェネレータ**：シミュレーション対象のネットワークの情報を管理するツールである。実際は、ユーザに最も近い GUI 内部で動作する。ネットワークの情報は、DSL で表現される。

(2) **コードジェネレータ**：ネットワークジェネレータで生成された DSL コードを、対応する ns-3 コードへ変換するツールである。このコードジェネレータで出力されたファイルを ns-3 コード断片ファイルと呼ぶ。コードジェネレータはパッケージャから呼び出される。

(3) **パッケージャ**：複数の断片ファイルを統合し、最終的に ns-3 で実行可能なファイル群を出力するツールである。パッケージャからの出力を元に ns-3 が動作することとなる。

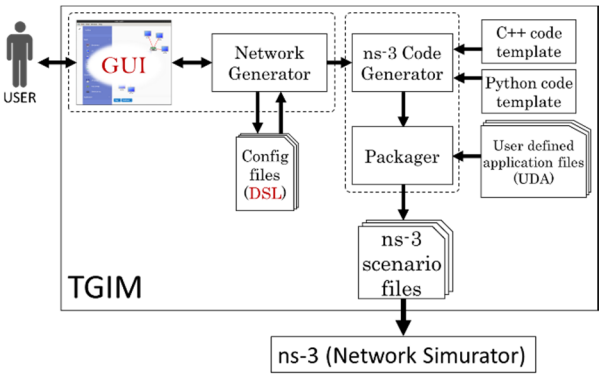

Figure 2. TGIM アーキテクチャ

† 山形大学　Yamagata University

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

4　DSL の設計

T.G.の問題点はトポロジ情報の管理方法だった。そもそも ns-3 でネットワークトポロジを構築する際にも、トポロジの管理はどうしても複雑になる。そのため、ns-3 ではユーザを補助する helper を用意している。helper は、ns-3 の奥の実装を隠蔽し、頻繁に行われる作業をまとめ、ユーザの負担を軽減する。例えば、ダンベルトポロジ、メッシュトポロジなどのトポロジは helper を利用し、生成できる。しかし、実際のネットワークは、helper で作成できるような単純なネットワークばかりではない。

実際のネットワークを対象とする場合、helper のみで対応するのは不可能であり、ユーザは ns-3 の奥の機能を用いて、コードを記述しなければならない。また、ns-3 のシナリオは、装置の選択、装置の結線、インターネットスタックの装備などを手続き的に記述するようにしている。そのため、必要な工程を前もって把握しておくことが重要となり、直感的にネットワークの構成を記述するのは容易ではない。T.G.の場合は、ネットワークの管理を C++のオブジェクトを用いて行っており、ns-3 の場合と同様にネットワーク構成の把握が容易でない。

この問題を解決するために、何らかの方法で ns-3 のコードを抽象化し、詳細な ns-3 の仕様と具体的な C++言語記述を意識しなくともよい方法を検討する必要があった。そこで、ネットワークシミュレーションの記述に特化した DSL を設計した。DSL ではネットワークトポロジの記述と、動作させるアプリケーションの指定が可能である。DSL での記述は、構造記述に特化した JSON をベースにし、構造的なアプローチをとった。これにより従来の手続き的なネットワーク構築と比べ、ネットワーク構成を構造的に把握できるようになった。また、JSON 形式であるため直接データとして扱える。

DSL では、ノードとチャネルの指定を可能としている。実用的な面で、複数サブネットの組み合わせや、ネットワーク記述をファイルへ分割することが必要になるため、これらの要件も考慮し DSL の仕様を定義した。

DSL では以下の 5 つの概念を使う。

(1) **ネットワークエントリ**：ネットワークのエントリポイントであり、ここから記述を始める。
(2) **チャネル**：ノード同士が通信を行うための媒体である。
(3) **ノード**：PC のようなネットワークのノードである。
(4) **サブネット**：他のネットワークをサブネットとして取り込む機能である。
(5) **アプリケーション**：ネットワーク上で動作するアプリケーションである。

ネットワークエントリ記述(List 1)では、始めにネットワーク名を指定する。次に、チャネルやノード、サブネット、アプリケーションの記述を行う。

チャネル記述(List 2)では、始めにチャネル名を指定する。次に、チャネルタイプとして Csma または PointToPoint を選択し、config に速度や遅延時間、経路特性などの指定が可能である。

ノード記述(List 3)では、始めにノード名を指定する。次に、netifs に接続先チャネル名を指定する。config にはノードに指定する IP アドレスの指定が可能である。

サブネット記述(List 4)では、始めにサブネット名を指定する。次に、サブネットを読み込むファイル名を指定する。netifs の設定は基本的にノードと同様である。up という特殊な項目があるが、ここには、そのサブネット内のノード名を指定する。そうすることでサブネット内のノードを、サブネット外のネットワークのノード（ゲートウェイ）として使用することが可能となる。これを TGIM ではサブネット・アップと呼んでいる。

アプリケーション記述(List 5)では、始めにアプリケーション名を指定し、次にアプリケーションタイプとして、利用するアプリケーションの種類（例えば ping）を、予め定義された選択肢から選び指定する。args にはアプリケーションに指定するパラメータ（送信元、送信先ノードやレートなど）を指定する。

アプリケーションの開始と停止のタイミングは、アプリケーション引数として指定することとしている。

List 1. ネットワークエントリの記述

```
{
  "name"    : "<ネットワーク名>",
  "channel" : <チャネルの記述>,
  "node"    : <ノードの記述>,
  "subnet"  : <サブネットの記述>,
  "apps"    : <アプリケーションの記述>
}
```

List 2. チャネルの記述

```
"channel" : {
  "<チャネル名>" : {
  "type"   : "<チャネルタイプ>",
  "config" : {
    "<属性名>" : "<値>"
  }
}
```

List 3. ノードの記述

```
"node" : {
  "<ノード名>" : {
    "netifs" : [
      {
        "connect" : "<接続先チャネル名>",
        "config"  : { "<属性名>": "<値>" }
      }
    ]
  }
}
```

List 4. サブネットの記述

```
"subnet" : {
  "<サブネット名>" : {
    "load" : "<読み込む設定ファイル>",
    "netifs" : [
      {  "up" : "<サブネット内のノード名>",
         "connect" : "<接続先チャネル名>",
         "config" : { "<属性名>" : "<値>" }
      }
    ]
  }
}
```

List 5. アプリケーションの記述

```
"apps" : {
  "<アプリケーション名>" : {
    "type" : "<アプリケーションタイプ>",
    "args" : {
      "<引数名>" : <値>
    }
  }
}
```


Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

5　GUI

ユーザが視覚的にトポロジを構築できるよう、GUI を開発した。GUI では直感的なトポロジ構築が可能である。また、アプリケーションの設定も GUI で可能である。内部ではネットワークジェネレータが、DSL を用いて情報を管理している。画面の構成を Figure 3 に示す。

画面中央はキャンバスという領域である。ユーザはキャンバス上で視覚的にトポロジを構築することが可能である。キャンバス下方には、そのネットワーク上で動作させるアプリケーションのリストが表示される。ノードやチャネル、アプリケーションは、それぞれ名前や接続情報、経路特性などのプロパティを持っている。プロパティを変更したい場合は、アイコンをクリックすることで編集可能になる。

画面右側は、ノード、チャネル、アプリケーションのプロパティを編集するための領域である。

画面左側の領域に、ノード・チャネルのリストが表示されており、アイコンをクリックし、キャンバスに機器を追加できる。また、アプリケーションもここから追加が可能であり、追加したアプリケーションはキャンバスの下方に表示される。

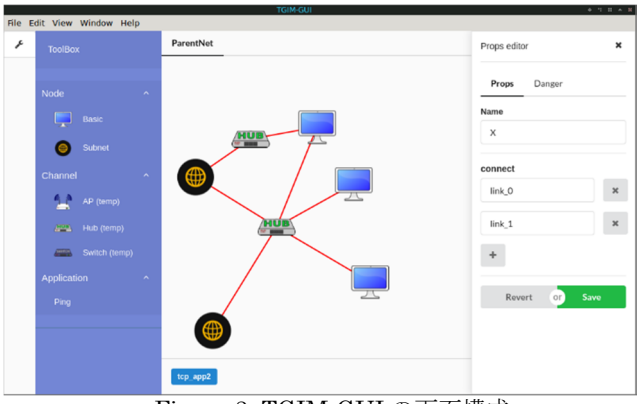

Figure 3. TGIM-GUI の画面構成

6　パッケージャ

パッケージャはコードジェネレータが出力したコードの断片やアプリケーション情報、その他必要な情報を集め、ns-3 のコード群に組み込むツールである。

具体的には、DSL ファイルを収集し、内部でコードジェネレータを呼び出し、ns-3 の C++コードへ変換を行う。また、依存関係の解決や不足しているファイルの生成なども行う。ユーザはパッケージャにいくつかのパラメータを指定することで、ns-3 で実行可能なコードを出力可能である。コードジェネレータの仕様は意識する必要がない。TGIM のパッケージャは、Web 系の開発で利用されるパッケージングシステム[5]を応用し作成した。

基本的には以下の 5 つの情報を指定することでパッケージングを行える。

(1) entry：ネットワークのエントリポイントとなるファイルを指定する。
(2) output：出力先のディレクトリを指定する。
(3) loader：DSL から ns-3 断片コードへの変換器（コードジェネレータ）を指定する。
(4) target： ns-3 のバージョンを指定する。
(5) simulator：シミュレーション停止の時間を指定する。

パッケージャは始めに List 6 に示したような、設定ファイルを読み込む。entry に指定されたファイルを起点とし、DSL で記述されたファイルを再帰的に探索していく。パッケージャは見つけたファイルを順に C++コード断片ファイルへ変換していく。最後に、メインファイルの作成を行う。メインファイルとは ns-3 シミュレータがエントリとして読み込むファイルである。実際のメインファイルには ns-3 のライブラリ群、断片ファイル、アプリケーション記述ファイルがインクルードされる。

List 6. tgim-pack.config.json

```
{
  "entry": "entryNet.json",
  "output": "./output/",
  "loader": "tgim-generator",
  "target": "ns-3",

  "simulator": {
    "stop": 100
  }
}
```

7　システムの検証

実際に TGIM を使ってシミュレーションを生成する。手順は以下の通りである。

step 1 GUI でトポロジ構築・アプリケーション設定
　(1.1)　ノード・チャネルの配置・結線
　(1.2)　アプリケーションの配置・引数設定
　(1.3)　作業ディレクトリに DSL ファイル保存
step 2 作業ディレクトリ内でパッケージング処理
　(2.1)　設定ファイル初期化・編集
　(2.2)　パッケージング処理
step 3 ns-3 でシミュレーションを実行
　(3.1)　ns-3 作業ディレクトリへ、パッケージャから出力されたシナリオのファイルをコピー
　(3.2)　waf ビルドシステムでのビルド・実行

今回の実験ではダンベルトポロジの生成と ping アプリケーション実行を行う。ping アプリケーションは送信元を Node0、送信先を Node5 とする。ダンベルトポロジ(Figure 4)は 3 つのネットワーク（Network 0、Network 1、Network 2）によって構成される。

Figure 4 では説明のため、装置を Node と Router の 2 種類に書き分けているが ns-3 では通常の場合、すべてがルータとなり得る。そのため実際には Node と Router を区別する必要はない。TGIM では通信媒体をチャネルとして扱う。そのため Figure 4 のノード間を結ぶ線は TGIM 上ではチャネルとして扱われる。

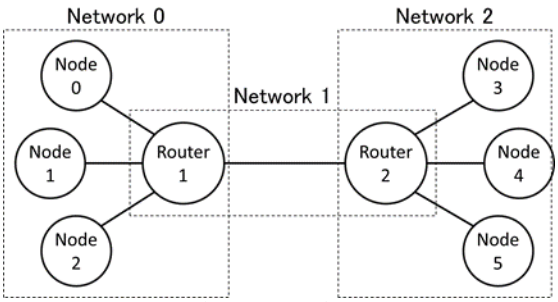

Figure 4. ダンベルトポロジの構成

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

8　シミュレーションの生成・実行

はじめに GUI でのトポロジ構築とアプリケーション設定を Figure 5 に示すように行い、作業ディレクトリに `DumbbellNet.json` という名前で DSL ファイルとして保存した。次にパッケージング処理を行う(List 7)。`tgim-pack init` でパッケージングに使用する設定ファイルの生成・初期化を行い、次にエディタで設定ファイルを編集する。今回エントリとなるファイルは”`DumbbellNet.json`”であるため、entry の項目にはこのファイル名を指定した。その他の設定は List 7 に示す通りである。最後に ns-3 のシミュレーション開始コマンドを入力し実行する(List 8)。実際の実行の様子は Figure 6 に示す通りである。この例では、サブネットの機能を利用していないが、GUI でも DSL と同様にサブネットとしてネットワークを分割し記述することが可能である。

9　考察

(1) 構造的で柔軟なシナリオ記述

ユーザにとってネットワークがどのような手順で作られたかは重要ではない。ネットワークがどのような形をしているかを、容易に把握できることが重要である。

そのことを念頭に置いた DSL と GUI により、従来の（ns-3 の C++記述のような）手続き的なシナリオ記述ではなく、（ネットワークのノード、チャネル、トポロジなどに重点を置いた）構造的な記述でシナリオ構築が可能とした。ユーザはネットワークの形に集中し、柔軟なシナリオ構築を行えるようになった。

(2) ns-3 に不足している機能の補完

DSL にはサブネットの概念を導入している。この概念は ns-3 では提供されていないが、DSL を用いることで、サブネットを利用したネットワークの分割を可能とした。また GUI は DSL を利用しているので、GUI 上でもサブネットが利用可能である。このように、ns-3 に不足している機能の補完を行った。

(3) 外部ツールと ns-3 の違いの吸収

DSL は JSON をベースとしており、外部のプログラムで容易に編集できる。また、シナリオを抽象的に表現可能である。そのため、DSL は外部ツールと ns-3 間の新たなインタフェースとなり得る。実際、DSL は GUI と ns-3 の、シナリオの表現の違いを吸収することにも貢献している。DSL は、外部ツールと ns-3 の違いだけでなく、ns-3 と他のネットワークシミュレータとの違いをも吸収できると考えている。

10　おわりに

TGIM によって、容易なシミュレーション生成を実現した。具体的には、TGIM を用いることで ns-3 の仕様を意識せずとも、有線系ネットワークシミュレーションを可能とした。従来のように、ユーザはシミュレーションの為だけに、シミュレータの仕様を深くまで理解する必要はなく、少ない知識でシミュレーションが行えるようになった。

今後は無線系ネットワークのシミュレーションへの対応策を検討していく。さらに、現在のアプリケーションやシミュレーションの開始/停止などの指定は限定的であるため、より一般的なタイミング指定の方法を検討しなければならない。そのためには、DSL を拡張していかなければならないが、ns-3 と同じような問題（ライブラリの肥大化・複雑化）を抱え、シミュレータの扱いが困難とならないよう、追加の要素をうまく扱う方法を新たに考えていきたい。

参考文献

[1] ns-3 <https://www.nsnam.org/> (2018)
[2] PyViz <https://www.nsnam.org/wiki/PyViz> (2018)
[3] Topology Generator,
<https://github.com/idaholab/Topology_Generator> (2018),
<https://www.nsnam.org/wiki/Topology_Generator> (2018)
[4] 銭飛, “ns3 によるネットワークシミュレーション”, 森北出版, (2014)
[5] webpack, <https://webpack.js.org/> (2018)

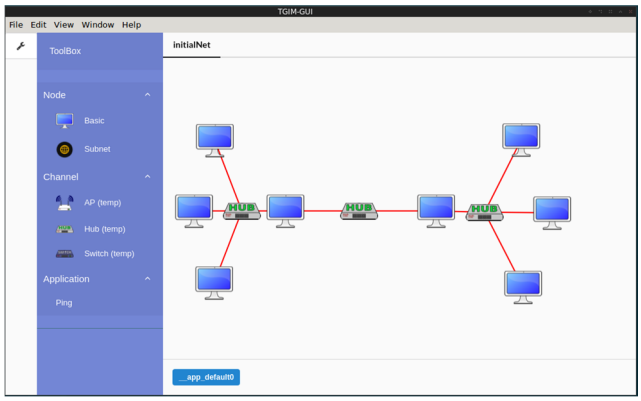

Figure 5. TGIM-GUI 上でのシナリオ構築

List 7. パッケージング処理の結果

```
% cd $HOME/Desktop/dumbbell-scenario
% tgim-pack init
Writing new package config file to ./tgim-
pack.config.json.
Copying module files to ./module.
Success!
% vi tgim-pack.config.json
% cat tgim-pack.config.json
{
  "entry": "DumbbellNet.json",
  "output": "./output",
  "loader": "tgim-generator",
  "target": "ns-3"
}
```

List 8. ns-3 でシナリオを実行

```
% cd $NS3LOCAL
% cp $HOME/Desktop/\
dumbbell-scenario/output/* .
% ./waf --run "pack-main" --vis
```

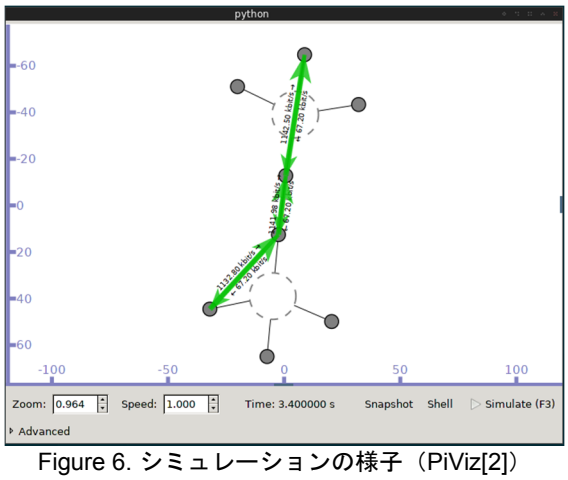

Figure 6. シミュレーションの様子（PiViz[2]）

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.