使ってみよう NS3

2017 年 6 月 28 日版
慶應義塾大学　大学院理工学研究科
岡本　聡

1. NS3 とは

1.1. NS3 の前には NS2 があった

NS (Network Simulator)は、オープンソースで開発が進められている、インターネット技術を主な対象とした離散事象ネットワークシミュレータ（NS1、NS2、NS3)の総称である。NS1 は、米国 DARPA の研究プロジェクトの研究成果でありローレンス・バークレー国立研究所において 1995 年頃に開発された。その後、UCB (カリフォルニア大学バークレイ校)において、NS2 の開発が継続して進められ、最終的に USC (南カルフォルニア大学)で管理が行われた。興味のある人は、
http://www.isi.edu/nsnam/ns/　　や
http://nsnam.sourceforge.net/wiki/index.php/User_Information に情報があるので覗いてみると良い。

NS3 は、NS2 の開発が収束したことを受けより進化したバージョンとして、フランスを中心として開発が進められている（https://www.nsnam.org/)。ただし、NS2 との互換性は無く、過去の資産が使えない。NS2 にある機能がまだ NS3 では実装されていないといった不都合もあるいが、OpenFlow、WiFi、WiMAX、LTE 等の最新のネットワーク技術に対応していることから、今後増々発展していくことが期待されている。また、NS3 は、実環境との統合が可能であり、シミュレータと実ネットワークとの間でパケットをやり取りすることも可能となっている。

NS3 は、シミュレータのコア、シナリオファイルは共に C++ で書かれている。そのため、シミュレーションの実行は、C++ ソースファイルをコンパイルしてシミュレータアプリケーションを構築して実行させることとなる。実際のコンパイルは waf という python ベースのビルドシステムを介して実行される。また、この授業では採り上げないが、python でスクリプトを書くことでも実行が可能となっている。

シミュレーション結果は、NS2 で好評だった、ネットワークアニメータ(nam: http://www.isi.edu/nsnam/nam/) を踏襲して付属の NetAnim を利用したアニメーションや、libpcap 形式のパケットトレースデータから、wireshark を利用して可視化することができる。

インターネットで検索すると色々な資料があります。この授業では、習うより慣れろということで、NS3 を体験してみましょう。

※ NS3 の最新版は、ns-3.26 ですが、授業においては利用するプログラムの仕様上 ns-3.24 を使います。

> 第 12 回から第 14 回の授業は、この資料に沿って自主的に行ってください。授業の最初に説明はありません。仲間と組んで進めても OK です。先生は、巡回していますので、随時質問してください。

2. NS3 がインストールされていることの確認

2.1. which コマンドにより ns を探索する

ワークステーションにログインして、以下のコマンドを打つ※”△”はスペースの意味※
$ which△ns
インストールされていれば、
$which△ns
/usr/local/bin/ns
$
のように、探索したコマンドの絶対パスが表示される。
ただし、ここで見つかった ns は、残念ながら NS2 である。

ITC のワークステーションにおいては、/usr/keio/ns-allinone-3.24/usr/local/ns-3-allinone/ に NS3 がインストールされている（はず）。自分専用の NS3 システムを構築する場合や、最新の NS3 システムを利用したい場合には、**2.3** の NS3 パッケージのダウンロードとインストールを行う。

※ /usr/local/ns-3-allinone/ が存在している可能性もある（実態は、/usr/keio/ns-allinone-3.24/usr/local/ns-3-allinone/ と同一）。その場合は、2.2 で指定するディレクトリを /usr/local/ns-3-allinone/ とします。

NS3 パッケージをビルドすると Disk 容量を 1.5 GB 程度使用してしまう。このため、ITC のワークステーションでは quota の関係から自分のホームディレクトリでビルドすることはお薦めしない。

現状、NS3 はインストールされている場所でプログラムを実行する必要があるため、上記 /usr/keio/ns-allinone-3.24/usr/local/ns-3-allinone/をどこかにコピーしてその場所に cd してから実行することが要求されている模様。 /work にコピーしても良いが、自動的に消されてしまうため、 lndir を利用して自分のホームディレクトリ配下に NS3 を配置する方法を紹介する。

※ mini 演習：quota が何なのか調べてみましょう。各自の上限は 2 GB に設定されているようです。
※ mini 演習：lndir がどのようなことを行うコマンドなのか、man コマンドや google 検索を利用して調べてみましょう。 重要キーワード symbolic link。

2.2. 自分のホームディレクトリに NS3 を配置

ITC が用意してくれた NS3 を自分のホームディレクトリで動くように修正を行います。

$ cd　　←このコマンドで自分のホームディレクトリに戻ります。
$ mkdir△ns-3-allinone
$ cd△ns-3-allinone
$ lndir△/usr/keio/ns-allinone-3.24/usr/local/ns-3-allinone△.　← ピリオドを忘れないこと。
$ ls　　← 中身を一応確認しましょう。

$ cd△ns-3.24

上書きされるファイルは、パーミッションの関係で書き込めないのでシンボリックリンクを削除してしまいます。

$ rm△build/ns3/*.h△build/ns3/private/*.h△build/c4che/*.py
$ rm△build/config.log△build/.lock-waf_linux2_build
$ rm△-r△build/.old_srcdir
$ rm△build/compile_commands.json△build/build-status.py
$ rm△build/src/*/*.pc

これで準備ができました。NS3 をビルドします。

$ ./waf△configure△--enable-examples△--enable-tests

赤字で表示されるメッセージは気にしなくても OK。最後にエラーで止まっていたら、きっと上書きしようとしていて Permission denied と言われているので、該当ファイルを削除してしまいましょう。

次にビルドです。

$ ./waf△build

時間が数分～数 10 分かかる可能性もあります。

build が終わったら、とりあえず動かしてみましょう。

$ ./waf△--run△hello-simulator

最後に Hello Simulator と表示されましたか？

2.3 を飛ばして **2.4** に行きますが、**2.3** にも目を通しましょう。

2.3. NS3 パッケージのダウンロードとインストール

2.3.1. NS3 ソースパッケージのダウンロード

https://www.nsnam.org/releases/ にソースやドキュメントが存在している。安定最新版は 3.26 (Oct. 3, 2016) であるが、授業では 3.24 を利用する。“Older” をクリックして、ns-3.24 (September 2015)を選択し、ns-allinone-3.24.1.tar.bz2 (23.7MB)をダウンロードする。

2.3.2. NS3 ソースパッケージの展開

各自のホームディレクトリに、ns-allinone-3.24.1.tar.bz2 をダウンロードしてきたら、以下の命令でパッケージを展開する。

※ホームディレクトリ以外（Desktop）にダウンロードしていることもあるので、ホームディレクトリ直下に mv しておきましょう。

NS3 用の作業ディレクトリを例えば ~/ns-3-allinone とします。

$cd　　←このコマンドで自分のホームディレクトリに戻ります。
$mkdir△ns-3-allinone
$ cd△ns-3-allinone
$ tar△jxvf△~/ns-allinone-3.24.1.tar.bz2

2.3.3. NS3 のビルドとテスト

$ cd△ns-allinone-3.24/ns-3.24
ビルドの仕方は README に書いているので目を通す。
README に書いてあるように以下の命令を実行して NS3 をコンパイルしてビルドする。
$ ./waf△configure△--enable-examples△--enable-tests
$ ./waf△build
テストを実行して確認する。
$ ./test.py
しばらくして、329 of 339 tests passed (329 passed, 10 skipped, 0 failed, 0 crashed, 0 valgrind errors) のように failed, crashed, errors が 0 となったら OK。
駄目な場合は、パッケージやモジュールが足りないので、
https://www.nsnam.org/wiki/Installation を参考にして追加する。

とりあえず動かしてみましょう。
$ ./waf△--run△hello-simulator

最後に Hello Simulator と表示されましたか？

次に、NS3 のアニメーション表示のための NetAnim を作成する。Qt の 4.8 以上が要求されているので、パッケージを追加することが必要かもしれない。
Mac の場合は、https://www.nsnam.org/wiki/NetAnim を参考にしてください。Linux では以下でできます。
$ cd△../netanim-3.106　← 今 ns-allinone/ns-3.24 にいる。ns-allinone/netanim-3.106 へ
$ make△clean　※Makefile が無い場合エラーとなるがそれで OK
$ qmake△NetAnim.pro
$ make
できあがった、NetAnim を ns-3.24 の配下にコピーしましょう。
$ cp△NetAnim△../ns-3.24/

2.4. PATH の設定

※細かなバージョンの数字は変わっているかもしれないので、それぞれの環境に合わせること。
自分の shell が /bin/sh や /bin/bash の場合 (多分こちらです)
~/.bashrc に以下を エディタ(emacs 等)を用いて追加記述します。

> ファイルを壊すとコマンドが使えなくなることもあるのでまずはファイルのバックアップをとる。
> $ cd
> $ cp△.bashrc△.bashrc.20170706

2.3 のビルド作業をしない人は、PATH に関する以下の記述を追加します。
/usr/local/ns-3-allinone が存在することをまず確認。存在していない場合は、先生へ連絡。
========== add from here ==========
export△NS_HOME=$HOME/ns-3-allinone/ns-3.24
export△NETANIM=/usr/local/ns-3-allinone/netanim

export△PATH=$NETANIM:$PATH
==========ここまで==========
追加したら **2.4.1** へ移行します。
2.3 のビルド作業をした人は、PATH に関する以下の記述を追加します。
========== add from here ==========
export△NS_HOME=$HOME/ns-3-allinone/ns-allinone-3.24/ns-3.24
export△NETANIM=$NS_HOME
export△PATH=$NETANIM:$PATH
==========ここまで==========

2.4.1. .bashrc の反映

~/.bashrc に追加した後で、以下の命令を実行して設定変更を有効にします。
$source△~/.bashrc

- source コマンドは、実行した Terminal Window でのみ有効。面倒な場合は Terminal Window を exit して、再度 Terminal Window を開くのが確実に反映させるコツ。

2.5. NetAnim のテスト

NetAnim コマンドのありかを確認します。
$ which△NetAnim
自分でビルドした場合は、~/ns-3-allinone/ns-allinone-3.24/ns-3.24/NetAnim と出ていれば OK。ITC ワークステーションでは、/usr/local/ns-3-allinone/netanim/NetAnim と出ることでしょう。

※ 違っている場合、先生へ連絡。

NetAnim のテストをします。まず、NS3 パッケージに附属している NetAnim 用のサンプルを実行してみます。
$ cd△$NS_HOME
$ cp△src/netanim/examples/star-animation.cc△scratch/
コピーした star-animation.cc を実行してみます。
$ ./waf△--run△star-animation
star-animation.xml が出来上がります。
以下のコマンドを実行して、NetAnim 画面が立ち上がれば OK です。
$ NetAnim
立ち上がったら、左上のフォルダアイコンをクリックして、star-animation.xml を選択して読み込ませます。

※ 三角形の Play Forward ボタンを押して、シミュレーション経過のアニメを楽しみましょう。
※ $NS_HOME/src/netanim/examples/ には、他にもサンプルがありますので、同じように scratch の下にコピーして実行してみましょう。

NS3 のサンプルは、$NS_HOME/examples/ 配下にあります。どんなファイルがあるのかを ls コマンドで確認し、気になったものを見てみましょう。また、実行することもできます。例えば、examples/tcp/star.cc を実行したければ

$ cd△$NS_HOME
$ ./waf△--run△star
で実行することができます。

3. 参考書及びインターネット上の情報

参考書：森北出版 ns3 によるネットワークシミュレーション（定価 4,200 円+税）
ISBN978-4-627-85201-3

http://www.nsnam.org　～ Web ページ
http://www.nsnam.org/documentation/　～ ドキュメント
http://www.nsnam.org/wiki　～ Wiki

4. NS3 で Hello World

NS3 の簡単な使い方を示します。画面に “Hello World”を出力させます。

以下の内容のファイル $NS_HOME/scratch/HelloWorld.cc を作成する。

```
$ cd△$NS_HOME/scratch
$ emacs△HelloWorld.cc
========================== add from here ================================
#include△”ns3/core-module.h”

using△namespace△ns3;

NS_LOG_COMPONENT_DEFINE△(”HelloWorld”);

int△main△(△int△argc, △char△*argv[]△)
{
  △△NS_LOG_UNCOND△(△”Hello△World!!”△);
}
==========================ここまで==================================
```

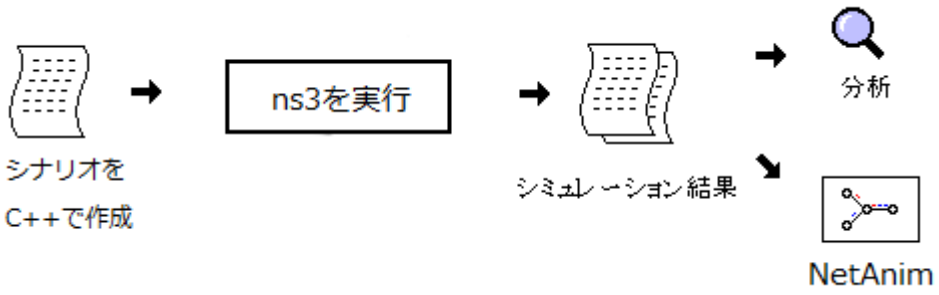
このプログラムを以下のように実行する。

```
$ cd△$NS_HOME
$ ./waf△--run△HelloWorld
```

NS_LOG_UNCOND () は、無条件に LOG を書き出してくれるおまじない。

5. シミュレーションシナリオの記述

NS3 でシミュレーションを実行するためには、シナリオスクリプトを C++で記述します。NS3 は、シナリオに沿ってシミュレーションを実行してくれます。



シナリオでは、ノード、チャネル、ネットワークデバイス、アプリケーションを記述していきます。

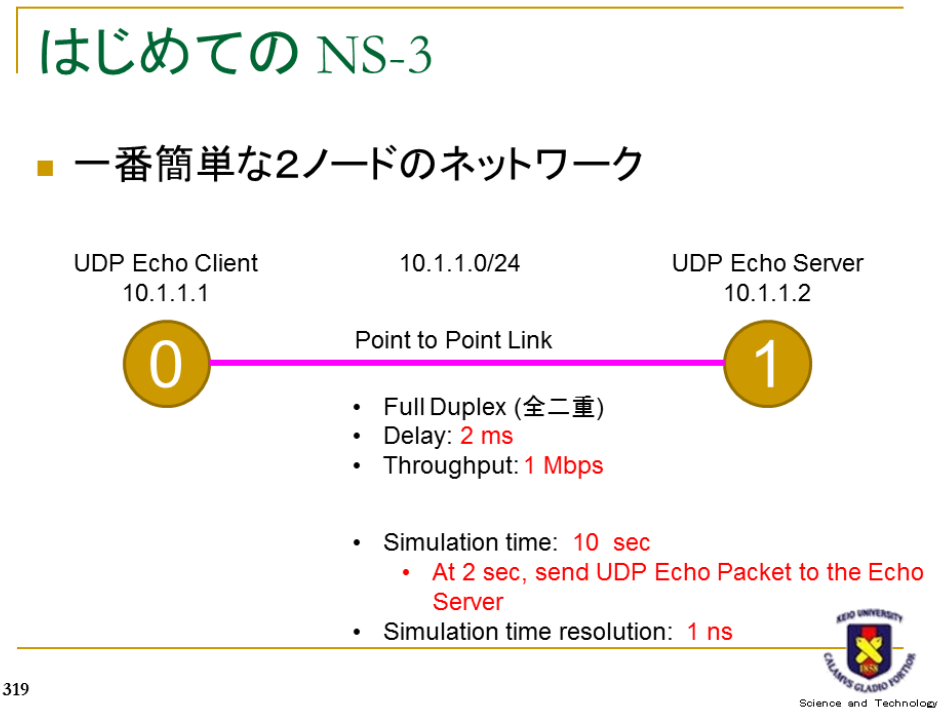
- ノード(Node)
  NS3 の基本的なコンピューティング・デバイス。ノードに、ネットワークデバイス、プロトコルスタック、アプリケーション等の機能を追加することで、ネットワーク上のルータやスイッチ等として表現できる。
- チャネル(Channel)
  ノード間の通信を定義する。有線系のネットワークでは、P2P チャネル、CSMA チャネル。無線系のネットワークでは、WiFi チャネル、WiMAX チャネル等を利用する。
- アプリケーション(Application)
  UDP アプリケーション、TCP アプリケーション等がある。具体的なアプリケーションはまだあまり開発されていないので、NS2 等では TCP で良く使われるアプリケーションである ftp なんかも無い。逆に言えば、自分で開発して世界に公開していけるチャンスでもある。

手順としては、①ノードを作る、②リンクを定義する、③ネットワークデバイスをノードに装着する、④プロトコルスタックをデバイスに載せる、⑤ネットワークアドレスをデバイスに割り当てる、⑥アプリケーション用のポート番号等を割り当てる、⑦アプリケーションをノードに装着し、それぞれの開始・終了時刻等を指定していく。

5.1. はじめての NS3

具体的なシナリオを、簡単な例を用いて説明します。題 11 回 NS3 入門で紹介した test1.cc は、$NS_HOME/examples/tutorial/first.cc を少しだけ書き換えたものです。まず、first.cc を scratch/test1.cc としてコピーします。

$ cd△$NS_HOME
$ cp△examples/tutorial/first.cc△scratch/test1.cc
$ cd△scratch
エディタで、test1.cc を書き換えます。



赤字の部分が test1.cc の変更箇所になります。

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
// 上の行は、Emacs 用のおまじない。C++のファイルには付けておくと便利。
// 下のコメント 14 行は削除しておきましょう。
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

// 本来は必要なヘッダだけ書けば良いのですが、とりあえず
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
```

```
#include "ns3/applications-module.h"

// C++のおなじない。NS3 用の機能を使うことを宣言
using namespace ns3;

// NS3 用の LOG のおまじない
NS_LOG_COMPONENT_DEFINE ("Test1");

int
main (int argc, char *argv[])
{
  Time::SetResolution (Time::NS);      // 時間の解像度を nano seconds にする
  // 下の 2 行は、アプリケーションで UDP を使う場合のおなじない
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

  // ノードをまずは作ります
  NodeContainer nodes;
  nodes.Create (2);    // ノード 0 と　ノード 1 の２個

  // Point-to-Point のリンクを作るための Helper を使います
  PointToPointHelper pointToPoint;
  // デバイスとチャネルのアトリビュートを設定してリンクを定義
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));

  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);
  // 全ノード間にリンクを作成するおなじない
  // 特定ノード間に設定する場合の例を下記に示す
  // device = pointToPoint.Install (nodes.Get (0), nodes.Get (1)) ;

  // 次に、プロトコルスタックをノードにインストール
  InternetStackHelper stack;
  stack.Install (nodes);

  // IPv4 を使います
  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0", "0.0.0.1");
  // 開始アドレスが、0.0.0.1 であることを指定しているので、ノード 0 が
  // 10.1.1.1/24、ノード 1 が 10.1.1.2/24 になります

  Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

```
  // ここからアプリケーションの設定をしています
  // UDP Echo Server の　待ち受けポート番号を 9 に設定しています
  UdpEchoServerHelper echoServer (9);

  // UDP Echo Server をノード 1 にインストールします
  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
  serverApps.Start (Seconds (1.0));     // シミュレーション開始後 1 秒で起動
  serverApps.Stop (Seconds (10.0));   // シミュレーション開始後 10 秒で終了

  // 次に UDP Echo Client を作成します
  // まず、宛先ノードのアドレスを指定しないといけません
  // ノード 1 のポート 9 が宛先 ですね
  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  // 毎秒 1 個、1024 バイトのパケットを送信します
  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

  // UDP Echo Client をノード 0 にインストールします
  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
  clientApps.Start (Seconds (2.0));   // シミュレーション開始後 2 秒で起動
  clientApps.Stop (Seconds (10.0));   // シミュレーション開始後 10 秒で終了

 // シミュレーションの実行と終了のおまじない
  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

書き換えが終わったら、セーブして一度実行してみましょう。

```
$ cd△..
$ ./waf△--run△test1
```

※ 実験 1 ：test1.cc を test1-1.cc にコピーし、UDP echo クライアントが 3.0 秒から、0.5 秒毎に合計 10 個パケットを送るように改造して実行してみましょう。

※ 実験 2 ： test1-1.cc を test1-2.cc にコピーし、LOG レベルを LOG_LEVEL_INFO から LOG_LEVEL_ALL に書き換えて実行してみましょう。 Clinet と Server のアプリケーション毎に LOG レベルは自由に変えることができます。詳細な情報が欲しい場合には LOG レベルを LOG_LEVEL_ALL にすると良いです。

※ LOG レベルはファイルを書き換えなくても環境変数から指定することもできます。

```
$ export△NS_LOG=UdpEchoClientApplication=level_all
$ ./waf△--run△test1
```

※ csh 系の場合 $ setenv△NS_LOG=UdpEchoClientApplication=level_all と指定

LOG レベルに指定できる値を以下に示します。

ログレベル	ログ情報	NS_LOG の値
LOG_LEVEL_ERRO	各種エラーメッセージ	level_error
LOG_LEVEL_WARN	警告メッセージ	level_warn
LOG_LEVEL_DEBUG	デバッグメッセージ	level_debug
LOG_LEVEL_INFO	通信イベント情報	level_info
LOG_LEVEL_FUNCTION	関数トレース情報	level_function
LOG_LEVEL_LOGIC	関数中の制御フロー情報	level_logic
LOG_LEVEL_ALL	全ての情報	level_all

※ LOG を使って、どこまで実行したか等を表示できます。値を出力したい場合は C++のプログラムで、String にして出力すれば良いわけです。

例えば、test1.cc に以下を追加して実行すると

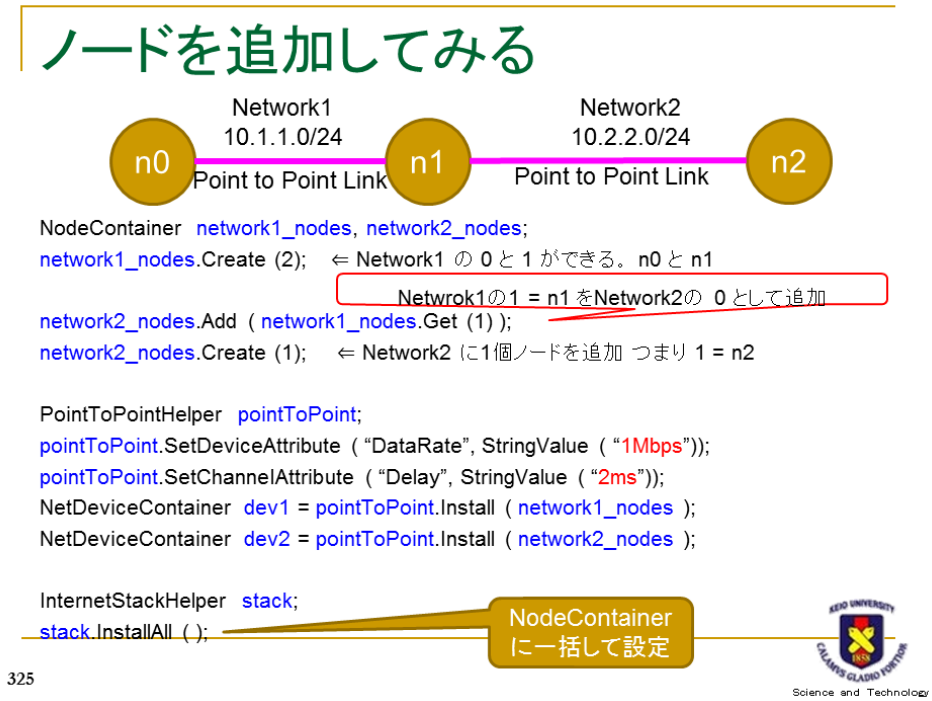
```
int
main (int argc, char *argv[])
{
  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);

// INFO レベルのログメッセージを出力
  NS_LOG_INFO△("Creating△Topology");
  NodeContainer nodes;
  nodes.Create (2);

$ export△NS_LOG=Test1=level_info
$ ./waf△--run△test1
Waf: Entering directory `**********************/build'
Waf: Leaving directory `**********************/build'
'build' finished successfully (*.***s)
Creating Topology
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

のように指定したメッセージが出力できます。

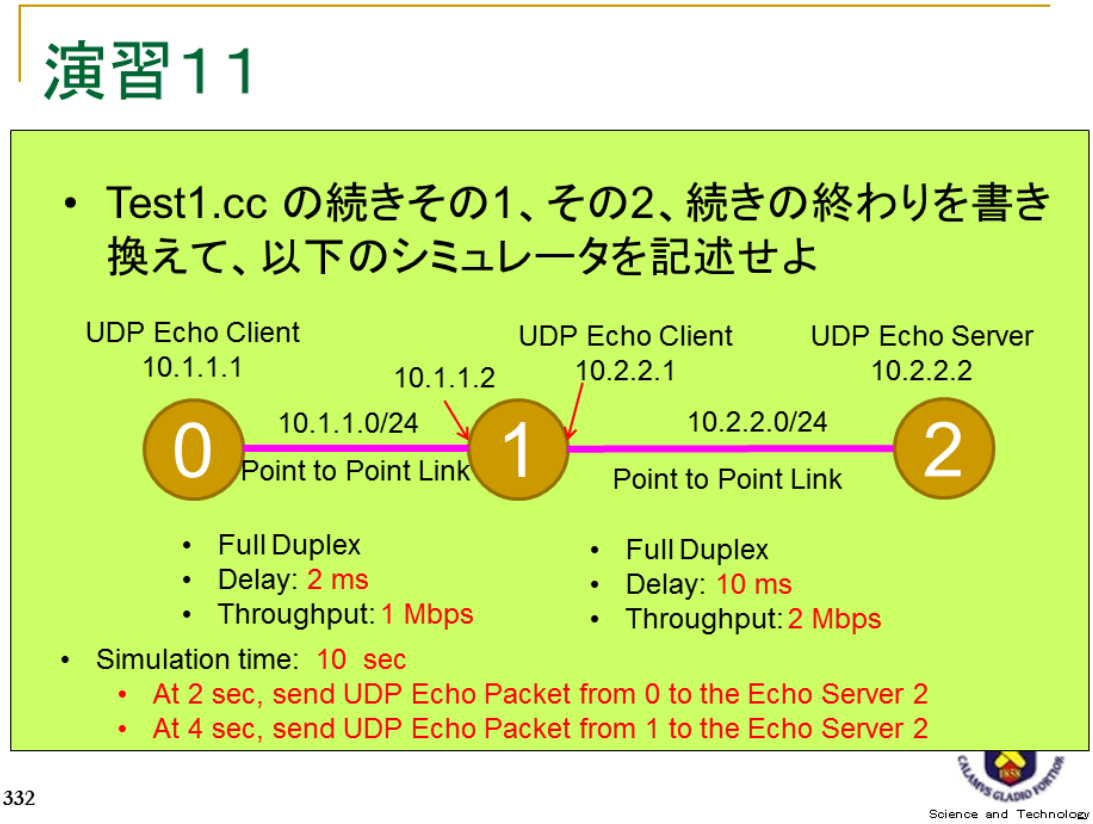
実行時に、メッセージが鬱陶しければ、NS_LOG の環境変数をクリア。

```
$ export△NS_LOG=
$ ./waf△--run△test1
```

5.2. ノードを追加してみる



※ 実験 3：test1.cc をコピーして、test2.cc を作ります。上のノードを追加してみるのスライドを参考にして、もう一度 演習 11 を test2.cc として作成し、実行してみましょう。



ヒント：Ipv4AddressHelper address1, address2 と二つ定義します。
ApplicationContainer で serverApps は 1 個だけ network2_nodes.Get(1) に配置。
宛先アドレスは、interface2.GetAddress(1) になります。
UdpEchoClientHelper は、宛先が共通なので、node 0 と node 1 から発出するパケット数、間隔、サイズが同じであれば、共用することもできます。
ルーティングしてあげないと、パケットが届かないので、ルーティングのおまじないを記述します。
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

5.3. コマンドラインからのパラメータ指定

C++なので、コマンドラインからパラメータを指定するのは容易です。

```
int
main (int argc, char *argv[])
{
// コマンドライン引数を使えるようにする
  CommandLine△cmd;
  cmd.Parse△(argc,△argv);
```

test1.cc を test1-4.cc にコピーして上記変更を加え、何が、できるのか色々試してみましょう。まずはコマンドラインヘルプから。

```
$ ./waf△--run△ “test1-4△--PrintHelp“
```

“ “ で括るのは、括らないと waf に --PrintHelp が渡ってしまって、

```
waf: error: no such option: --PrintHelp
```

と怒られるからです。

```
test1-4 [Program Arguments] [General Arguments]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintHelp:                 Print this help message.
```

と出力されます。

コマンドラインからパラメータを指定するために、例えば、test1-4.cc の中で、デバイス属性を指定する行をコメントアウトしてみます。

```
  PointToPointHelper    pointToPoint;
  // pointToPoint.SetDeviceAttribute ( "DataRate", StringValue ( "1Mbps"));
  // pointToPoint.SetChannelAttribute ( "Delay", StringValue ( "2ms"));
```

指定しないとデフォルトの値で実行されます。まずコマンドラインから何も指定しないで書き換えた test1-4 を実行してみましょう。

※ 実験 4：test1 の実行結果と何が違っているのかわかりましたか？
遅延時間とリンク速度ががデフォルト値になったので、server received の時刻が変わってきます。

デフォルト値を確認してみましょう。

```
$ ./waf△--run△"test1-4△--PrintAttributes=ns3::PointToPointNetDevice"
```

コマンドラインからパラメータを指定してみます。

```
$ ./waf△--run△ “test1-4△\
        --ns3::PointToPointNetDevice::DataRate=1Mbps△\
        --ns3::PointToPointChannel::Delay=2ms”
```

サーバの受信時刻が元に戻っていることが確認できましたか？

次のステップとして、自分でコマンドラインから指定できるパラメータを追加してみましょう。test1.cc を test1-5.cc にコピーします。test1-5.cc に以下を追加します。

```
int
main (int argc, char *argv[])
{
  uint32_t△nPackets = 1;

  CommandLine△cmd;
  cmd.AddValue("nPackets",△"Number△of△packets△to△echo",△nPackets);
  cmd.Parse△(argc,△argv);

・・・

  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
  // echoClient.SetAttribute ("MaxPackets ", UintegerValue (1));
  echoClient.SetAttribute△("MaxPackets",△UintegerValue△(nPackets));
  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
```

Help を出すように実行するとコマンドライン引数が増えていることがわかります。

```
$ ./waf△--run△“test1-5△--PrintHelp“
```

※ 実験 5：パケットが 2 つ出るように指定して実行してみましょう。--nPackets=2 を与えます。

5.4. トレース機能の利用

NS3 では、シミュレーションのトレース情報がイベントトレース機構によって処理されて出力されます。トレースファイルとしては人間の目に優しい(?)ASCII トレースファイル形式と、tcpdump や wireshark で処理するための PCAP トレースファイルの二種類があります。

5.4.1. ASCII トレースファイルの利用

ASCII トレースファイルを利用するためには、Simulator ::Run を呼び出す前に以下の命令文を追加します。

```
//△ASCII△Tracing
AsciiTraceHelper△ascii;
pointToPoint.EnableAsciiAll△(ascii.CreateFileStream ("トレースファイル名"));
```

トレースファイル名は、一般的には ファイル.tr のようにサフィックスを付けます。
実行終了後、カレントディレクトリに ”ファイル.tr” が生成されます。
test1-6.cc を作ります。ファイル名は、test1-6.tr とします。

```
$ less△test1-6.tr
+ 2 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue
ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052
10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
- 2 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Dequeue
ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052
10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
r 2.01043 /NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacRx
ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052
10.1.1.1 > 10.1.1.2) ns3::UdpHeader (length: 1032 49153 > 9) Payload (size=1024)
+ 2.01043 /NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue
ns3::PppHeader (Point-to-Point Protocol: IP (0x0021)) ns3::Ipv4Header (tos 0x0 DSCP
Default ECN Not-ECT ttl 64 id 0 protocol 17 offset (bytes) 0 flags [none] length: 1052
10.1.1.2 > 10.1.1.1) ns3::UdpHeader (length: 1032 9 > 49153) Payload (size=1024)
```

1 つ目の詳細
00 + enqueue
01 2 時刻
02 /NodeList/0/DeviceList/0/$ns3::PointToPointNetDevice/TxQueue/Enqueue 出力された trace souce
03 ns3::PppHeader (point-to-point protocol のヘッダ、つまり Ethernet ではない
04 Point-to-Point Protocol: IP (0x0021))

```
05 ns3::Ipv4Header (   IP のヘッダ
06         tos 0x0 ttl 64 id 0 protocol 17 offset 0 flags [none]
07         length: 1052 10.1.1.1 > 10.1.1.2)
08 ns3::UdpHeader (   UDP のヘッダ
09         length: 1032 49153 > 9)
10         Payload (size=1024)
```

+ が enqueue、- が dequeue、r がパケット受信、ここでは出てきていませんが d がパケットの損失を意味します。

5.4.2. PCAP トレースファイルの利用

PCAP (Packet CAPture) トレースファイルは、パケットキャプチャで汎用的に使われているバイナリ形式のファイルである。PCAP トレースを行うためには、Simulator ::Run を呼び出す前に

```
//△pcap△tracing
pointToPoint.EnablePcapAll△("トレースファイル名");
```

を実行させる。もちろん ASCII トレースとの併用が可能である。PCAP トレースファイル名に .pcap を指定する必要は無く、自動的に .pcap として作成される。

test1-6.cc に上記を追加して実行すると、test1-6-0-0.pcap と test1-6-1-0.pcap の二つのトレースファイルが生成される。
test1-6-0-0.pcap は、ノード 0、NetDevice0 のインタフェースのトレース、test1-6-1-0.pcap は、ノード 1、NetDevice0 のインタフェースのトレースである。

5.5. ネットワークトポロジーの生成

5.5.1. スタートポロジーの自動生成

$NS_HOME/examples/tcp/star.cc を scratch/mystar.cc としてコピーします。

```
$ cd△$NS_HOME
$ cp△examples/tcp/star.cc△scratch/mystar.cc
```

mystar.cc を編集していきましょう。ポイントは、PointToPointStarHelper、OnOffHelper、Ipv4GlobalRoutingHelper の使い方です。

```
/* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
// 下のコメントは削除してしまいましょう。
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
```

```
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 */

#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/netanim-module.h"    // NetAnim 用だ！！！
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/point-to-point-layout-module.h" // Star Topology の定義がある

// Network topology (default)
//
//           n2 n3 n4                  . 全部で 9 ノード
//            \ | /                   .  n0 が hub
//             \|/                    . n1-n8 が spoke です。
//       n1--- n0---n5                .
//             /|\                     .
//            / | \                    .
//           n8 n7 n6                  .
//


using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("MyStar");  // ログの名称

int
main (int argc, char *argv[])
{

  //
  // Set up some default values for the simulation.
  // デフォルトの値をパケットサイズ 137 バイト、データレート 14 kbps に設定
  Config::SetDefault ("ns3::OnOffApplication::PacketSize", UintegerValue (137));
```

```
  // ??? try and stick 15kb/s into the data rate
  Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue ("14kb/s"));

  //
  // Default number of nodes in the star.  Overridable by command line argument.
  // コマンドラインで、nSpokes を指定可能にしている
  uint32_t nSpokes = 8;

  CommandLine cmd;
  cmd.AddValue ("nSpokes", "Number of nodes to place in the star", nSpokes);
  cmd.Parse (argc, argv);

  NS_LOG_INFO ("Build star topology.");  // メッセージ出力
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
  PointToPointStarHelper star (nSpokes, pointToPoint);  // スタートポロジー用
  NS_LOG_INFO ("Install internet stack on all nodes.");  // メッセージ出力
  InternetStackHelper internet;
  star.InstallStack (internet);  // star にプロトコルスタックを組込む
  // 一般的には、internet.InstallAll (); だが、Helper が用意した InstallStack () を使用する方が良いらしい

  NS_LOG_INFO ("Assign IP Addresses.");  // メッセージ出力
  star.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"));
  // star に IP アドレスを割り当てる

  NS_LOG_INFO ("Create applications.");  // メッセージ出力
  //
  // Create a packet sink on the star "hub" to receive packets.
  //
  uint16_t port = 50000;          //宛先ポート 50000
  // 以下の記述は PointToPointStarHelper が決めているので深くは考えない
  Address hubLocalAddress (InetSocketAddress (Ipv4Address::GetAny (), port));
  PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory", hubLocalAddress);
  ApplicationContainer hubApp = packetSinkHelper.Install (star.GetHub ());
  // 1 秒から 10 秒まで動かす
  hubApp.Start (Seconds (1.0));
  hubApp.Stop (Seconds (10.0));

  //
  // Create OnOff applications to send TCP to the hub, one on each spoke node.
  // 以下の記述は OnOffHelper が決めているので深くは考えない。ON 時間と OFF 時間の比が大切。以下は、1:0 なのでずっと ON
```

```
  OnOffHelper onOffHelper ("ns3::TcpSocketFactory", Address ());
  onOffHelper.SetAttribute ("OnTime", StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
  onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));

  ApplicationContainer spokeApps;

  for (uint32_t i = 0; i < star.SpokeCount (); ++i)
    {
      AddressValue remoteAddress (InetSocketAddress (star.GetHubIpv4Address (i),
port));
      onOffHelper.SetAttribute ("Remote", remoteAddress);
      spokeApps.Add (onOffHelper.Install (star.GetSpokeNode (i)));
    }
  spokeApps.Start (Seconds (1.0));
  spokeApps.Stop (Seconds (10.0));

  NS_LOG_INFO ("Enable static global routing.");
  //
  // Turn on global static routing so we can actually be routed across the star.
  // ルーティングは、この Ipv4GlobalRoutingHelper が全部面倒を見てくれます
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  NS_LOG_INFO ("Enable pcap tracing.");
  //
  // Do pcap tracing on all point-to-point devices on all nodes.
  // PCAP トレースを作成
  pointToPoint.EnablePcapAll ("mystar");

  NS_LOG_INFO ("Run Simulation.");
  Simulator::Run ();
  Simulator::Destroy ();
  NS_LOG_INFO ("Done.");

  return 0;
}
```

PointToPointStarHelper は以下の様に定義されている。

```
ns3::PointToPointStarHelper△{
  uint32_t△numSpokes,                    // スポークノード数
  PointToPointHelper△pointToPoint        // ノー接続用ヘルパーのインスタンス
}
```

mystar.cc では、NetAnim 用のヘッダファイルを読み込んでいるが、NetAnim 用の XML ファイルは生成されていない。生成するための改造例を以下に示す。

```
#include△"ns3/netanim-module.h"

・・・

uint32_t nSpokes = 8;
std::string△animFile = "mystar-animation.xml"; //  ファイル名指定

・・・

CommandLine cmd;
cmd.AddValue ("nSpokes", "Number of spoke nodes to place in the star", nSpokes);
cmd.AddValue△("animFile",△"File Name for Animation Output",△animFile);
//  コマンドラインからファイル名指定可能にする
cmd.Parse (argc, argv);

・・・
// Set the bounding box for animation
star.BoundingBox△(1, 1, 100, 100);
//  ここで呼ばれている  star  は、PointToPointStarHelper の star のこと

// Create the animation object and configure for specified output
AnimationInterface△anim△(animFile);
```

※ 実験 6: test2.cc を test2-animation.cc にコピーして NetAnim で見られるようにしてみましょう。もちろん star.BoundingBox (1, 1, 100, 100) は使えない。
NetAnim で使われるオプション指定を以下に示します。

```
anim.SetConstantPosition△(Ptr< Node > n,△double△x,△double△y);
```

で、ノード n の場所を指定します。

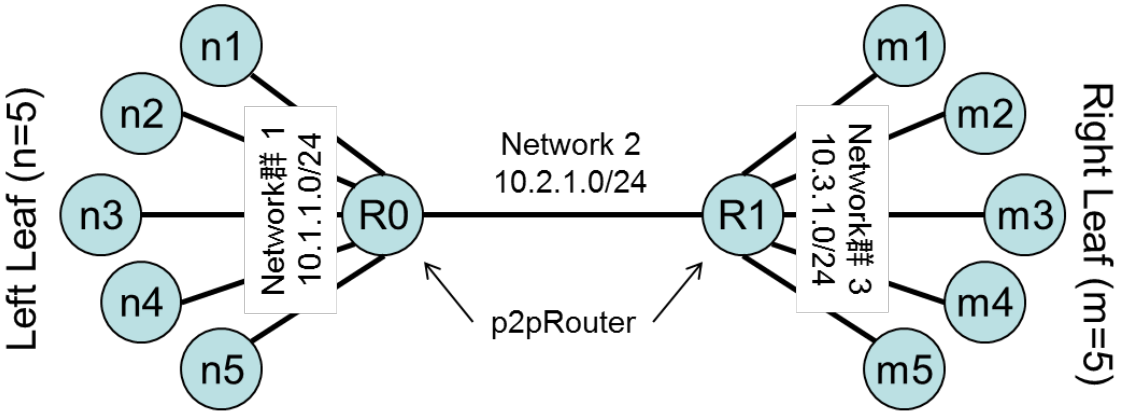
```
例：anim.SetConstantPosition (network1_nodes.Get(0), 1.0, 1.0);
anim.UpdateNodeDescription△(1,  △”node-1”);
```

ノード n に”node-1”というテキストを付与します。

```
anim.UpdateNodeSize△(1,△0.5,△0.5);
```

ノード 1 の大きさをデフォルトの x 方向 0.5 倍、y 方向 0.5 倍にします。

5.5.2. ダンベルトポロジーの自動生成



ダンベルトポロジーは、非常に良く使われるトポロジーであり、二つのルータで接続される三つのネットワーク群から構成される。ルータ間の接続リンクがボトルネックリンクとして、TCP/IP の制御プロトコル、パケットキューの管理方式等の特性解析に利用される。Network 群 1 は、10.1.1.0/24、10.1.2.0/24、10.1.3.0/24、10.1.4.0/24、10.1.5.0/24 の五つのネットワークであり、Network 群 2 は、10.3.1.0/24、10.3.2.0/24、10.3.3.0/24、10.3.4.0/24、10.3.5.0/24 の五つのネットワークとなる。
ダンベルトポロジーは、PointToPointDumbbellHelper を利用して生成する。利用のためには、#include “ns3/point-to-point-layout-module.h” が必要

```
ns3::PointToPointDumbbellHelper△{
  unit32_t          nLeftLeaf,              // 左辺のノード数
  PointToPointHelper△leftHelper,            // ノードを左辺のルータに接続するため
  unit32_t          nRightLeaf,             // 右辺のノード数
  PointToPointHelper△rightHelper,           // ノードを右辺のルータに接続するため

  PointToPointHelper△bottleneckHelper // 左右のルータを接続するため
}
```

図のダンベルトポロジーを使うための基本スクリプトを以下に示す。

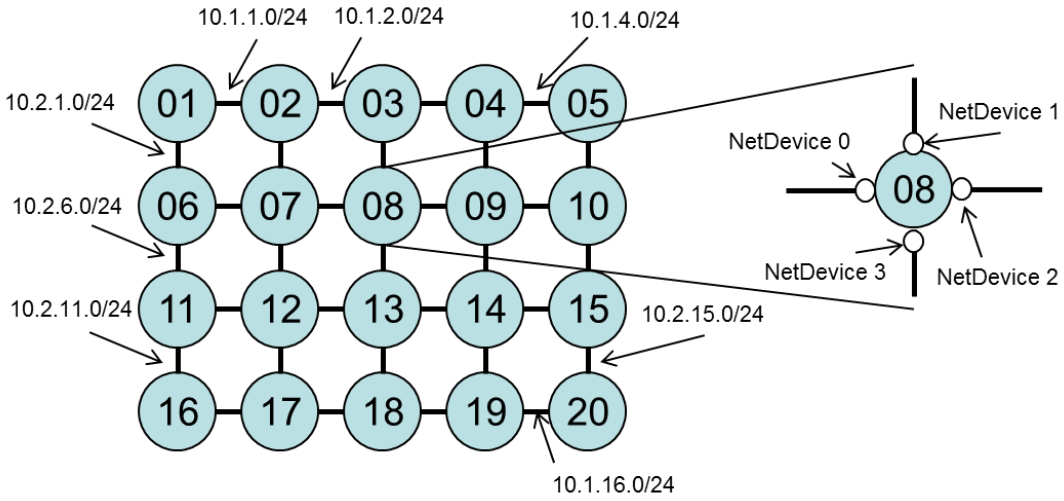
```
#include "ns3/point-to-point-layout-module.h"
using namespace ns3;

int main (int argc, char *argv[])
{
  uint32_t      nLeftLeaf = 5;
  uint32_t      nRightLeaf = 5;

  CommandLine cmd;
  cmd.AddValue ("nLeftLeaf", "Number of left side leaf nodes", nLeftLeaf);
  cmd.AddValue ("nRightLeaf","Number of right side leaf nodes", nRightLeaf);
  cmd.Parse (argc,argv);
```

```
  // Create the point-to-point link helpers
  PointToPointHelper     p2pRouter,     p2pLeaf ;
  // p2pRouter.SetDeviceAttribute    ("DataRate", StringValue ("10Mbps"));
  // p2pRouter.SetChannelAttribute ("Delay", StringValue ("1ms"));

  // p2pLeaf.SetDeviceAttribute       ("DataRate", StringValue ("10Mbps"));
  // p2pLeaf.SetChannelAttribute     ("Delay", StringValue ("1ms"));

  PointToPointDumbbellHelper    net (nLeftLeaf, p2pLeaf,
                                     nRightLeaf, p2pLeaf,
                                     p2pRouter);

  // Install Stack
  InternetStackHelper stack;
  net.InstallStack (stack);     // ここも InstallStack ()  が使われている。

  // Assign IP Addresses
  net.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"),
                           Ipv4AddressHelper ("10.2.1.0", "255.255.255.0"),
                           Ipv4AddressHelper ("10.3.1.0", "255.255.255.0"));

  // Set up the acutal simulation
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

具体的な TCP/IP アプリケーションのインストール例は、
$NS_HOME/src/netanim/examples/dumbbell-animation.cc を参考にすると良い。

※ 実験 7：dumbbell-animation.cc のソースを眺めてから実行して、パケット送信の様子を NetAnim で確認してみましょう。
※ 実験 8：Leaf ネットワークの数を 256 以上に設定すると何が起こるのでしょう？

5.5.3. グリッド型トポロジーの自動生成



グリッド（格子）型トポロジーは、メッシュネットワークやセンサネットワークの実験で良く利用される。グリッド型トポロジーは、PointToPointGridHelper を利用して生成する。利用のためには、#include “ns3/point-to-point-layout-module.h” が必要

```
ns3::PointToPointGridHelper△{
  unit32_t          nRows,          // グリッドの行数
  unit32_t          nCols,          // グリッドの列数
  PointToPointHelper△pointToPoint // ノード間のルータを接続するため
}
```

図のダンベルトポロジーを使うための基本スクリプトを以下に示す。
グリッド型トポロジーに IP アドレスを割り当てる際に、AssignIpv4Addresses メソッドに対して、行と列に別々のネットワークアドレスを指定することが必要となる。AssignIpv4Addresses メソッドに渡される IP アドレスとネットマスクは各行と列のリンクに割当てられるサブネットワークアドレスの初期値であり、初期番号から順次割当ていく。また、図の右側の拡大図のように、グリッドヘルパーでノードを生成する際に、各インタフェースポートのデバイス名を時計まわりの順で割り当てている。

```
#include "ns3/point-to-point-layout-module.h"
using namespace ns3;

int main (int argc, char *argv[])
{
  uint32_t      nRows = 4;
  uint32_t      nCols = 5;

  CommandLine cmd;
  cmd.AddValue ("nRows", "Number of Rows", nRows);
  cmd.AddValue ("nCols","Number of Columns", nCols);
```

```
  cmd.Parse (argc,argv);

  // Create the point-to-point link helpers
  PointToPointHelper     p2p ;
  // p2p.SetDeviceAttribute    ("DataRate", StringValue ("10Mbps"));
  // p2p.SetChannelAttribute ("Delay", StringValue ("1ms"));

  PointToPointGridHelper    grid_net (nRows, nCols, p2p);

  // Install Stack
  InternetStackHelper stack;
  grid_net.InstallStack (stack);   // ここも InstallStack () が使われている。

  // Assign IP Addresses
  grid_net.AssignIpv4Addresses (Ipv4AddressHelper ("10.1.1.0", "255.255.255.0"),
                                Ipv4AddressHelper ("10.2.1.0", "255.255.255.0"));

  // Set up the acutal simulation
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();

  Simulator::Run ();
  Simulator::Destroy ();
  return 0;
}
```

5.6. IPv6 の利用

IP アドレスの設定には、Ipv4AddressHelper と Ipv6AddressHelper が利用可能。
IPv4 の場合、

```
NetDeviceContainer△dev0=p2p.Install (network1_nodes);
Ipv4AddressHelper△ipv4;
ipv4.SetBase△(network_address,△netmask△[,△start_number]);
// SetBase の第三引数の start_number は 1 番から割り当てるなら省略可能
ipv4.Assign△(dev0);
```

IPv6 の場合その１（自動割当の場合）

```
NetDeviceContainer△dev0=p2p.Install (network1_nodes);
Ipv6AddressHelper△ipv6;
Ipv6InterfaceContainer△i=ipv6.Assign△(dev0);
```

IPv6 の場合その２（プレフィックス指定の場合）

NetDeviceContainer△dev0=p2p.Install (network1_nodes);
Ipv6AddressHelper△ipv6;
ipv6.SetBase△(Ipv6Address(“2001:1::”),△Ipv6Prefix(64));
Ipv6InterfaceContainer△i=ipv6.Assign△(dev0);

※ 実験 9：test2-animation.cc を test2-v6-animation.cc にコピーして、使用するアドレスを IPv6 にしてシミュレーションを実行してみましょう。

5.7. TCP、UDP の設定

TCP または UDP の受信端(Sink)は、PacketSinkHelper で生成される。設定例を以下に示す。

TCP sink の設定例
uint16_t△sinkPort = 8080;
Address△sinkAddress△(
　InetSocketAddress△(interfaces.GetAddress (1), △sinkPort));
PacketSinkHelper△packetSinkHelper△("ns3::TcpSocketFactory",△sinkAddress);

UDP sink の設定例
uint16_t△sinkPort = 8080;
Address△sinkAddress△(
　InetSocketAddress△(interfaces.GetAddress (1), △sinkPort));
PacketSinkHelper△packetSinkHelper△("ns3::UdpSocketFactory",△sinkAddress);

一方、送信側は、上位のアプリケーションとして、OnOffHelper、BulkSendHelper、UdpServer/ClinetHelper 、 UdpEchoServer/ClientHelper 、 Ipv4PingHelper 、 Ipv6PingHelper 等が用意されている。

自前のアプリケーションを開発する場合に、直にソケットを生成することもできる。
Ptr<Socket>△ns3TcpSocket△=△Socket::CreateSocket△(
　nodes.Get△(node_id),△TcpSocketFactory::GetTypeId△(△)△);
上記で、node_id はソケットを装着するノードの ID となる。

5.7.1. BulkSendHelper

BulkSendHelper は、大容量のデータ転送をシミュレーションするのに利用される。アプリケーション的には ftp を模擬していると考えれば良い。
BulkSendHelper のアプリケーションは、指定されたデータの最大転送量になるまでデータを生成する。ただし、シミュレーション時間が短いと転送終了前にシミュレーションが終了する。
利用の手順は以下の通り、
Step1: BulkSendHelper のインスタンスを取得(ftp とする)
　BulkSendHelper△ftp△(“ns3::TcpSocketFactory”,△Address(△));

Step2: インスタンスの属性を設定する
```
AddressValue△remoteAddress△(
        InetScoketAddress△(ifs.GetAddress(idx,△0),△PORT));
ftp.SetAttribute△(“Remoet”,△remoteAddress);
ftp.SetAddribute△(“MaxBytes”,△UintegerValue△(int△(500*1024*1024)));
```

remoteAddress は、接続先である Sink ノードのアドレス
ifs.GetAddress(idx, 0)は、Sink ノードの該当インタフェースポートに対応する IP アドレスを取得するためのメソッドであり、idx はノードコンテナで Sink ノードを生成する際に得られる番号
PORT は、使用される TCP ポート番号
MaxBytes では、例として 500 MB を指定している

Step3: インスタンスを Source ノードに装着する
```
ApplicationContainer△sourceApp1△=△ftp.Install△(net1_nodes.Get(0));
```

net1_nodes.Get(0) は、net1_nodes 中の最初のノードに装着することを意味する

5.7.2. OnOffHelper

トラヒック生成で一番使われるのは、OnOffHelper かもしれない。
OnOffHelper は、以下の属性を持つ

DataRate	On 状態でのデータ発生レート
PacketSize	On 状態で送出されるパケットのサイズ
Remote	宛先アドレス
OnTime	On の時間を表すランダム変数
OffTime	Off の時間を表すランダム変数
MaxBytes	生成されるデータのトータルバイト数。0 は無限大。
Protocol	使用されるプロトコルのタイプ。デフォルトは TcpNewReno

パケットサイズやデータ発生のレート指定は以下の通り
```
Config::SetDefault△(“ns3::OnOffApplication::PacketSize”,
                    UintegerValue△(512));
Config::SetDefault△(“ns3::OnOffApplication::DataRate”,
                    StringValue△(“500kb/s”));
```

使用するためには、まず OnOffHelper ソケットを生成する。
ネットワークデバイス ifs の最初のノード(net1_nodes の 1 番目のノード)に装着する例を示す。宛先は ifs の 3 番目のノードとする。
```
// ifs の最初のノードのアドレスを取得する。PORT は事前に定義済のポート番号
Address△srcAddress△(InetSocketAddress△(ifs.GetAddress△(0),△PORT));
// OnOffHelper のソケット(名称 tcp1)を生成
OnOffHelper△tcp1△(“ns3::TcpSocketFactory”,△srcAddress);
```

```
// 属性の指定
tcp1.SetAttribute△(“OnTime”,
        StringValue△(“ns3::ConstantRandomVariable△[Constant=1]”));
tcp1.SetAttribute△(“OffTime”,
        StringValue△(“ns3::ConstantRandomVariable△[Constant=0]”));
tcp1.SetAttribute△(“DataRate”,△StringValue△(“100kbps”));
tcp1.SetAttribute△(“PacketSize”,△UintegerValue△(1024));

// Sink ノードのアドレス指定
AddressValue△remoteAddress
        (InetSocketAddress△(ifs.GetAddress△(2),△PORT));
Tcp.SetAttribute△(“Remote”,△remoteAddress);

// Source ノードに装着
ApplicationContainer△sourceApp1△=△tcp.Install△(net1_nodes.Get(0));
```

OnOffHelper は、OnTime で指定された時間内では固定ビットレート(Constant Bit Rate: CBR)のトラヒックが生成される。デフォルトは 512 byte のパケットが 500 kb/s である。OffTime で 0 を指定すると、常時 On となる。

On 時間、Off 時間を表すランダム変数は様々なものが用意されている。

5.7.2.1. Constant OnOff フロー

```
OnTime と OffTime が固定。
tcp1.SetAttribute△(“OnTime”,
        StringValue△(“ns3::ConstantRandomVariable△[Constant=On 時間]”));
tcp1.SetAttribute△(“OffTime”,
        StringValue△(“ns3::ConstantRandomVariable△[Constant=Off 時間]”));
```

5.7.2.2. Exponential OnOff フロー

```
フローを生成する時間間隔が、指定された平均値を持つ指数分布の乱数で決定
tcp1.SetAttribute△(“OnTime”,△StringValue△(
        “ns3::ExponentialRandomVariable△[Mean=指数分布の平均値]”));
tcp1.SetAttribute△(“OffTime”,△StringValue△(
        “ns3::ExponentialRandomVariable△[Mean=指数分布の平均値]”));
```

もちろん、OnTime は固定、OffTime は Exponential を指定することも可能である。

5.7.2.3. Uniform OnOff フロー

```
フローを生成する時間間隔が、指定された最小値と最大値の間の一様分布乱数で決定
tcp1.SetAttribute△(“OnTime”,△StringValue△(
        “ns3::UniformRandomVariable△[Min=最小時間,△Max=最大時]”));
```

tcp1.SetAttribute△(“OffTime”,△StringValue△(
“ns3::UniformRandomVariable△[Min=最小時間,△Max=最大時]”));

5.7.2.4. その他の確率分布

NS3 では、以下の確率分布も使える。

分布名	NS3 変数名	属性パラメータ
パレート分布	ParetoRandomVariable	[Bound, Mean, Shape]
正規分布	NormalRandomVariable	[Mean, Variance, Bound]
対数正規分布	LogNormalRandomVariable	[Mu, Sigma]
ガンマ分布	GammaRandomVariable	[Alpha, Beta]
アーラン分布	ErlangRandomVariable	[K, Lambda]
三角分布	TriangularRandomVariable	[Mean, Min, Max]
ジッフ分布	ZipfRandomVariable	[N, Alpha]
ゼータ分布	ZetaRandomVariable	[Alpha]
ワイブル分布	WeibullRandomVariable	[Scale, Shape, Weibull Bound]
乱数系列	SequentialRandomVariable	[Min, Max, Increment, Consecutive]

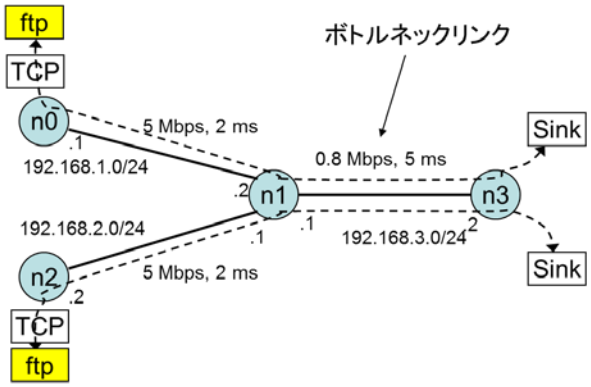
※ それぞれの分布を検索等で調べて、各パラメータの意味と分布の意味を確認しましょう。

6. ネットワーキング実験

実験は、参考書：森北出版 「ns3 によるネットワークシミュレーション」に書かれている実験スクリプトを使用して行う。
まず、公開されているスクリプトをダウンロードしてこよう。
http://www.morikita.co.jp/exclusive/download/1136
ダウンロードされるのは、085201src.tar.gz である。展開すると ns3-book というディレクトリが作成される。必要なのは、ns3-book/work/local/ 以下となるのでコピーする。
$ cd△$NS_HOME
$ cp△-r△[ns3-book の場所]ns3-book/work/local△.
$ ls△local

exp01 以下のコンテンツがありすか？

6.1. TCP ソケット通信の基本特性



実験シナリオ

- n0-n1 間、n1-n2 間のリンクは全二重、容量 5 Mbps、転送遅延時間 2 ms
- n2-n3 間のリンクは全二重、容量 800 kbps、転送遅延時間 5 ms
- 各ノードでは、DropTail 方式でキューの管理を行い、ボトルネックリンク(n1-n3)のキューの最大サイズを 5 pkt (パケット)とする
- n0-n1、n1-n2、n1-n3 間のネットワークを夫々、192.168.1.0/24、192.168.2.0/24、192.168.3.0/24 とする
- n0-n3 間、n2-n3 間に TCP NewReno の FTP フローを流す。n0 と n2 は source、n3 は sink である。使用するポート番号は 50000 とする
- セッション中に転送される最大データ量は 500 Mbyte
- FTP の開始時刻は 01 s、シミュレーション時間を 20 s とする

シナリオのスクリプトは、local/exp01/exp01-simpleTCP.cc である。FTP のアプリケーションは、BulkSendHelper を利用して実装している。
以下に解説する。

```
1 /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2 /* exp01-simpleTCP: simulate TCP behaviours with a simple 4-nodes topology.
3  * F.Qian, Nov. 2012
4  *
5  * n0(tcp source)
6  *   \.1
7  *     \ net1(25Mbps,2ms)192.168.1.0/24
8  *   .2\
9  *         n1 -----net3(10Mbps,5ms)----- n3(tcp sink)
10 *   .1/   .1         192.168.3.0/24          .2
11 *     / net2(25Mbps,2ms)192.168.2.0/24
12 *   /.2
13 * n2(tcp source)
14 */
15
```

```
16 #include <iostream>        // このあたり C++の普通のヘッダ
17 #include <fstream>
18 #include <string>
19
20 #include "ns3/core-module.h"
21 #include "ns3/network-module.h"
22 #include "ns3/internet-module.h"
23 #include "ns3/point-to-point-module.h"
24 #include "ns3/applications-module.h"
25 #include "ns3/tcp-header.h"
26 #include "ns3/udp-header.h"
27
28 #define NET_MASK          "255.255.255.0"
29 #define NET_ADD1          "192.168.1.0"       // n0-n1
30 #define NET_ADD2          "192.168.2.0"       // n1-n2
31 #define NET_ADD3          "192.168.3.0"       // n1-n3
32 #define FIRST_NO          "0.0.0.1"           // .1 から開始
33
34 #define SIM_START          00.10       // 0.1 s 開始
35 #define SIM_STOP           20.10       // 0.1 + 20 = 20.1
36 #define DATA_MBYTES          500       // 500 MB
37 #define PORT               50000       // ポート番号 50000
38
39 using namespace ns3;
40
41 #define PROG_DIR          "local/exp01/data/"          // 出力格納場所
42
43 NS_LOG_COMPONENT_DEFINE ("exp01-SimpleTCP");
44
45 int main (int argc, char *argv[], char *envp[])    // 環境変数も参照できる
46 {
47         CommandLine cmd;
48         std::string       trace_socket = "false";
49 // 実行時 traceSocket=True が指定された時、送受信者間のソケット通信イベントの表示を行う。指定されていない場合、TCP NewReno の Congestion Window の修正イベントの表示を行う。
50         cmd.AddValue("traceSocket", "True:trace TCP flow between source and sink.", trace_socket);
51         cmd.Parse(argc, argv);
52
53         if(trace_socket.compare("True") == 0)
54                 LogComponentEnable("TcpSocketBase", LOG_LEVEL_FUNCTION);
55         else
```

```
56                      LogComponentEnable("TcpNewReno",
LOG_LEVEL_INFO);
57          // DropTailQueue の大きさを 5 pkt に設定
58            Config::SetDefault ("ns3::DropTailQueue::MaxPackets",
UintegerValue (5));
59
60            NS_LOG_DEBUG("Creating Topology");
61            NodeContainer net1_nodes;
62            net1_nodes.Create (2);          // まず n0 と n1 を作成
63
64            NodeContainer net2_nodes;
65            net2_nodes.Add (net1_nodes.Get(1));  // n1 を Add
66            net2_nodes.Create (1);          // n2 を作成
67
68            NodeContainer net3_nodes;
69            net3_nodes.Add (net1_nodes.Get(1));  //n1 を Add
70            net3_nodes.Create (1);          // n3 を作成
71
72            PointToPointHelper p2p1;       // n0-n1 間、n1-n2 間用
73            p2p1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
74            p2p1.SetChannelAttribute ("Delay", StringValue ("2ms"));
75            p2p1.SetQueue ("ns3::DropTailQueue");
76
77            NetDeviceContainer devices1;
78            devices1 = p2p1.Install (net1_nodes);
79
80            NetDeviceContainer devices2;
81            devices2 = p2p1.Install (net2_nodes);
82
83            PointToPointHelper p2p2;       // n1-n3 間のボトルネックリンク
84            p2p2.SetDeviceAttribute ("DataRate", StringValue ("800Kbps"));
85            p2p2.SetChannelAttribute ("Delay", StringValue ("5ms"));
86            p2p2.SetQueue ("ns3::DropTailQueue");
87
88            NetDeviceContainer devices3;
89            devices3 = p2p2.Install (net3_nodes);
90
91            InternetStackHelper stack;
92            stack.InstallAll ();            // プロトコルスタックを組込む
93
94            Ipv4AddressHelper address;
95
96            address.SetBase (NET_ADD1, NET_MASK, FIRST_NO);
97            Ipv4InterfaceContainer ifs1 = address.Assign (devices1);
```

```
 98              NS_LOG_INFO ("Network 1: " << ifs1.GetAddress(0, 0) << " - " <<
ifs1.GetAddress(1, 0));
 99
100              address.SetBase (NET_ADD2, NET_MASK, FIRST_NO);
101              Ipv4InterfaceContainer ifs2 = address.Assign (devices2);
102              NS_LOG_INFO ("Network 2: " << ifs2.GetAddress(0, 0) << " - " <<
ifs2.GetAddress(1, 0));
103
104              address.SetBase (NET_ADD3, NET_MASK, FIRST_NO);
105              Ipv4InterfaceContainer ifs3 = address.Assign (devices3);
106              NS_LOG_INFO ("Network 3: " << ifs3.GetAddress(0, 0) << " - " <<
ifs3.GetAddress(1, 0));
107            // ルーティングの自動実行
108              NS_LOG_INFO ("Initialize Global Routing.");
109              Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
110            // TCP Sink のアドレスを n3 の IF0 に設定
111              AddressValue remoteAddress (InetSocketAddress
(ifs3.GetAddress(1, 0), PORT ));
112            // BulkSendHelper でアプリケーション ftp を作成
113              BulkSendHelper ftp ("ns3::TcpSocketFactory", Address());
114              ftp.SetAttribute ("Remote", remoteAddress);
115              ftp.SetAttribute ("MaxBytes", UintegerValue (int(DATA_MBYTES
* 1024 * 1024 )));
116            // n0 に ftp をインストール
117              ApplicationContainer sourceApp1 = ftp.Install (net1_nodes.Get(0));
118              sourceApp1.Start (Seconds (SIM_START+0.1));
119              sourceApp1.Stop    (Seconds (SIM_STOP -0.1));
120            // 1 個の sink で、全 source をまかなう時のおまじない
121              // create a sink to recieve packets @ network3's node 1(192.168.3.2).
122              Address sinkAddress (InetSocketAddress (Ipv4Address::GetAny (),
PORT));
123              PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",
sinkAddress);
124
125              //sinkHelper.SetAttribute ("Protocol", TypeIdValue
(TcpSocketFactory::GetTypeId ()));
126              ApplicationContainer sink_apps = sinkHelper.Install
(net3_nodes.Get(1));                    // n3 に sink をインストール
127
128              sink_apps.Start (Seconds (SIM_START+0.1));
129              sink_apps.Stop    (Seconds (SIM_STOP -0.1));
130
131              // Tracing stuff
132              AsciiTraceHelper ascii;
```

```
133         std::string afname = std::string(PROG_DIR) + "simple-tcp.tr";
134         p2p2.EnableAsciiAll (ascii.CreateFileStream (afname));
135
136         // for PCAP Tracing
137         //std::string pfname = std::string(PROG_DIR) + "simple-tcp";
138         //p2p2.EnablePcapAll (pfname);
139
140         Simulator::Stop (Seconds(SIM_STOP));
141         Simulator::Run ();
142         Simulator::Destroy ();
143         return 0;
144 }
```

実行するためには、scratch 以下にコピーしておきます。
$ cp△local/exp01/exp01-simpleTCP.cc△scratch/
$ ./waf△--run△"exp01-simpleTCP△--traceSocket=False"

ns-3.25 以降で、仕様が変更されました。ns-3.25, 3-26 でこのまま実行すると、
msg="Logging component "TcpNewReno" not found. See above for a list of available log components",
file=../src/core/model/log.cc, line=370
terminate called without an active exception
とエラーになる。これは、TcpNewReno が、ns-3.25 以降削除されてしまったためである。
しかたが無いので、56 行目の LogComponentEnable("TcpNewReno", LOG_LEVEL_INFO); を
LogComponentEnable("TcpCongestionOps", LOG_LEVEL_INFO); に変更する。
再度実行すると、今度は、
msg="Could not set default value for ns3::DropTailQueue::MaxPackets",
file=../src/core/model/config.cc, line=779
terminate called without an active exception
となる。これを回避するには、58 行目をコメントにして、個別に DropTailQueue の MaxPackets をセットする必要があるのだが、残念ながらうまくいかない。
これは、次節以降にも通じる問題である。このため、授業では ns-3.24 を利用する。
参考：http://qian775.blogspot.jp/2016/04/ns3ns-325.html

実行すると、

```
0.236902 [node 0] In SlowStart, ACK of seq 537; update cwnd to 1072; ssthresh 262140
0.264273 [node 0] In SlowStart, ACK of seq 1609; update cwnd to 1608; ssthresh 262140
0.291643 [node 0] In SlowStart, ACK of seq 2681; update cwnd to 2144; ssthresh 262140
0.313114 [node 0] In SlowStart, ACK of seq 3753; update cwnd to 2680; ssthresh 262140
0.324914 [node 0] In SlowStart, ACK of seq 4825; update cwnd to 3216; ssthresh 262140
0.340484 [node 0] In SlowStart, ACK of seq 5897; update cwnd to 3752; ssthresh 262140
0.352284 [node 0] In SlowStart, ACK of seq 6969; update cwnd to 4288; ssthresh 262140
0.364084 [node 0] In SlowStart, ACK of seq 8041; update cwnd to 4824; ssthresh 262140
0.375884 [node 0] In SlowStart, ACK of seq 9113; update cwnd to 5360; ssthresh 262140
0.387684 [node 0] In SlowStart, ACK of seq 10185; update cwnd to 5896; ssthresh 262140
```

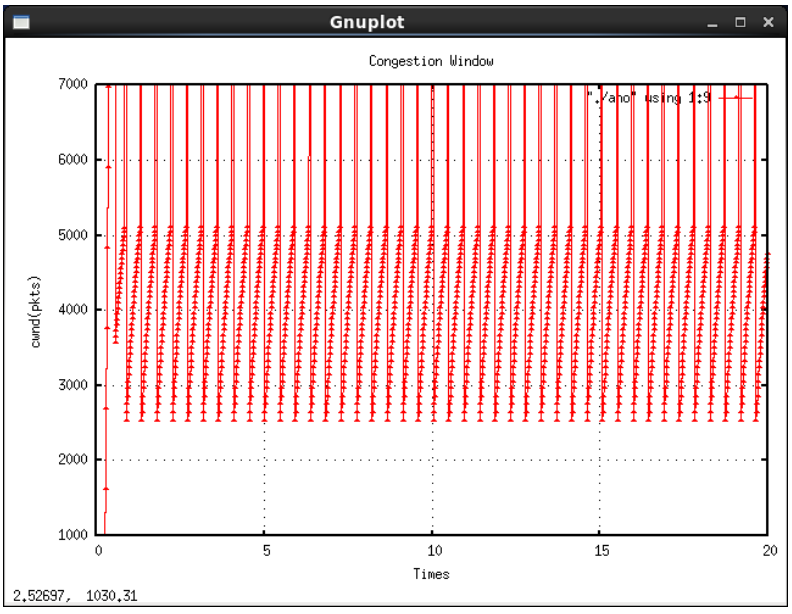
0.399484 [node 0] In SlowStart, ACK of seq 11257; update cwnd to 6432; ssthresh 262140
0.411284 [node 0] In SlowStart, ACK of seq 12329; update cwnd to 6968; ssthresh 262140
0.428984 [node 0] Triple dupack. Enter fast recovery mode. Reset cwnd to 5092, ssthresh to 3484 at fast recovery seqnum 19297
0.434884 [node 0] Dupack in fast recovery mode. Increase cwnd to 5628
0.440784 [node 0] Dupack in fast recovery mode. Increase cwnd to 6164
0.446684 [node 0] Dupack in fast recovery mode. Increase cwnd to 6700
0.452584 [node 0] Dupack in fast recovery mode. Increase cwnd to 7236
0.458484 [node 0] Dupack in fast recovery mode. Increase cwnd to 7772
・・・

のように、TCP の Slow Start と共に、Congestion Window (cwnd) が更新されていき、輻輳回避、リカバリーが行われていることがわかります。
cwnd が更新される様子を可視化してみましょう。

$ ./waf△--run△"exp01-simpleTCP△--traceSocket=Falese"△>&△exp01-log

ファイル exp01-log に出力が書きこまれました。これを gnuplot で表示してみます。

$ gnuplot
・・・
Terminal type set to 'x11'
gnuplot>△set△xrange△[0:20]
gnuplot>△set△yrange△[1000:7000]
gnuplot>△set△grid
gnuplot>△set△xlabel△'Times'
gnuplot>△set△ylabel△'cwnd△(pkts)'
gnuplot>△set△title△'Congestion△Window'
gnuplot>△plot△"./exp01-log"△using△1:9△with△linespoints△pt△9△ps△1.2
gnuplot>

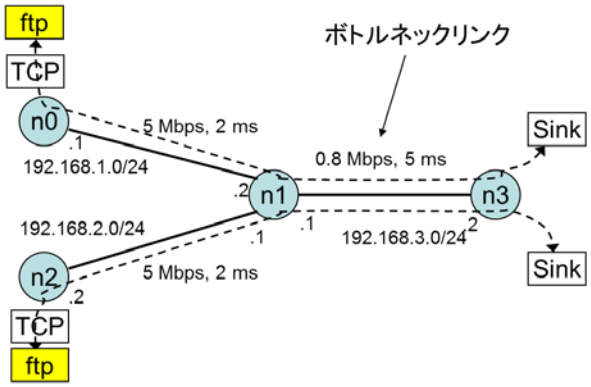


ファイル exp01-log を見てみると分かりますが、cwd の値が書かれている場所がスペースで区切られた 9 コラム目ではなく、12 カラム目や 13 カラム目になっている部分は正しい値が取得できなかったため、7000 を遥かに超えた値となって表示されています。

exp01-simpleTCP は、traceSocket=True にするとかなり詳細な動きが出力されます。

※ 実験 10：ここで提示したスクリプトは、PointToPointHelper でネットワークトポロジーを構築しましたが、PointToPointDumbbellHelper を使用した方が簡単です。書き換えましょう。
ただし、DumbbellHelper は右側のノード数 0 を指定できないようなので、1 ノード追加することが必要になるようだ。
※ 実験 11：PointToPointDumbbellHelper に書き換えた後に、NetAnim で表示できるようにしましょう。
※ 実験 12：ここで提示したスクリプトでは、n0 にだけ TCP の source が実装されています。n2 にも実装してみましょう。
※ 実験 13：ボトルネックリンクの容量を大きくすると cwd の更新の様子はどうなるのか、観察してみましょう。例えば、0.5 Mbp/s から 5 Mb/s まで 0.5 Mb/s 刻みで変化させてみます。

6.2. ネットワークスループットの観測



アプリケーションを BulkSend から OnOff アプリケーションに変更します。コマンドラインオプションとして、TCP new Reno (デフォルト)、TCP Reno、TCP Tahoe を使い別けられるように tcpType を、さらに発生させるフローのタイプを指定できるように flowType を導入します。
フロータイプとしては、Constant OnOff フロー、Exponential OnOff フロー、Uniform OnOff フローの三種類を使い別けます。
シナリオのスクリプトは、local/exp03/exp03-EstimateThroughput.cc である。scratch の下にコピーしておきましょう。

```
1 /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
2 /* exp03-EstimateThroughput.cc: simulate TCP behaviours with a simple 4-nodes topology.
3  * F.Qian, Nov. 2012
4  *
5  * n0(tcp source)
6  *  \.1
7  *   \ net1(25Mbps,2ms)192.168.1.0/24
8  *  .2\
9  *      n1 -----net3(10Mbps,5ms)----- n3(tcp sink)
10 *  .1/  .1       192.168.3.0/24        .2
11 *   / net2(25Mbps,2ms)192.168.2.0/24
12 *  /.2
13 * n2(tcp source)
14 */
15
16 #include <iostream>
17 #include <fstream>
18 #include <string>
19
20 #include "ns3/core-module.h"
21 #include "ns3/network-module.h"
22 #include "ns3/internet-module.h"
```

```
23 #include "ns3/point-to-point-module.h"
24 #include "ns3/applications-module.h"
25 #include "ns3/ipv4-global-routing-helper.h"
26 #include "ns3/tcp-header.h"
27 #include "ns3/udp-header.h"
28
29 #define NET_MASK    "255.255.255.0"
30 #define NET_ADD1    "192.168.1.0"
31 #define NET_ADD2    "192.168.2.0"
32 #define NET_ADD3    "192.168.3.0"
33 #define FIRST_NO    "0.0.0.1"
34
35 #define DATA_MBYTES        500
36 #define SIM_START          00.10
37 #define SIM_STOP           20.10
38 #define PORT               50000
39
40 #define PROG_DIR           "local/exp03/data/"
41
42 #define TH_INTERVAL        0.1        // The time interval in seconds for
measurement throughput
43 #define CW_INTERVAL        0.5        // The time interval in seconds for
measurement cwnd
44
45 using namespace ns3;
46
47 NS_LOG_COMPONENT_DEFINE ("exp03-OnOffTCPThroughput");
48   // congestion window  をトレースするためのモジュール
49 void
50 CwndTracer (Ptr<OutputStreamWrapper>stream, uint32_t oldcwnd, uint32_t
newcwnd)
51 {
52             //fprintf(stdout,"%10.4f %6d %6d\n",Simulator::Now ().GetSeconds
(),oldcwnd,newcwnd);
53             *stream->GetStream() << Simulator::Now ().GetSeconds ()
54                         << " \t " << newcwnd << std::endl;
55 }
56    //  スループットを計測するためのモジュールを作成していきます
57 uint32_t oldTotalBytes=0;          //スループット算出のための中間変数
58 uint32_t newTotalBytes;
59
60 void
61 TraceThroughput (Ptr<Application> app, Ptr<OutputStreamWrapper>
stream)
```

```
62 {
63          Ptr <PacketSink> pktSink = DynamicCast <PacketSink> (app);
64          newTotalBytes = pktSink->GetTotalRx ();
65          // messure throughput in Kbps
66          //fprintf(stdout,"%10.4f %f\n",Simulator::Now ().GetSeconds (),
67          //          (newTotalBytes - oldTotalBytes)*8/0.1/1024);
68          *stream->GetStream() << Simulator::Now ().GetSeconds ()
69                    << " \t " << (newTotalBytes - oldTotalBytes)*8.0/0.1/1024 <<
std::endl;   //   8.0/0.1/1024 の 0.1 も TH_INTERVAL とすべきです
70          oldTotalBytes = newTotalBytes;
71          Simulator::Schedule (Seconds (TH_INTERVAL), &TraceThroughput,
app, stream) ;       // TH_INTERVAL 毎に再帰的に呼ばれます
72 }
73
74 void
75 MyEventHandller (Ptr<Application> app, Ptr<OutputStreamWrapper>
stream)
76 {
77          Ptr<Socket> src_socket    = app-> GetObject<OnOffApplication>()->
GetSocket();
78          src_socket->TraceConnectWithoutContext("CongestionWindow",
79                    MakeBoundCallback(&CwndTracer, stream));
80 }
81
82 int
83 main (int argc, char *argv[])
84 {
85          std::string tcpType = "TcpNewReno";
86          std::string flowType = "Constant";
87          CommandLine cmd;
88
89          //LogComponentEnable("TcpSocketBase",
LOG_LEVEL_FUNCTION);
90
91          cmd.AddValue("tcpType",
92                    "TCP versions (TcpTahoe,TcpReno,TcpNewReno(default))",
tcpType);
93          cmd.AddValue("flowType",
94                    "Flow type (Constant(default),Exponential,Uniform)",
flowType);
95          cmd.Parse(argc, argv);
96
97          if(argc > 1) {
98                    if(tcpType.compare("TcpTahoe") == 0) {
```

```
 99                                  //LogComponentEnable("TcpTahoe",
LOG_LEVEL_INFO);
100                                 Config::SetDefault
("ns3::TcpL4Protocol::SocketType",
101                                         StringValue ("ns3::TcpTahoe"));
102                                 std::cout << "use " << tcpType << std::endl;
103                         } else if(tcpType.compare("TcpReno") == 0) {
104                                 //LogComponentEnable("TcpReno",
LOG_LEVEL_INFO);
105                                 Config::SetDefault
("ns3::TcpL4Protocol::SocketType",
106                                         StringValue ("ns3::TcpReno"));
107                                 std::cout << "use " << tcpType << std::endl;
108                         } else if(tcpType.compare("TcpNewReno") == 0) {
109                                 //LogComponentEnable("TcpNewReno",
LOG_LEVEL_INFO);
110                                 Config::SetDefault
("ns3::TcpL4Protocol::SocketType",
111                                         StringValue ("ns3::TcpNewReno"));
112                                 std::cout << "use " << tcpType << std::endl;
113                         } else {
114                                 std::cerr << "Invalid TCP version, use TCP
NewReno" << std::endl;
115                                 tcpType = "TcpNewReno";
116                         }
117             }
118
119             Config::SetDefault ("ns3::DropTailQueue::MaxPackets",
UintegerValue (10));    // この実験では バッファサイズを 10 pkts にしていますね
120             Config::SetDefault ("ns3::OnOffApplication::PacketSize",
UintegerValue (1024));
121             Config::SetDefault ("ns3::OnOffApplication::DataRate", StringValue
("100kb/s"));
122
123             NS_LOG_DEBUG("Creating Topology");
124              // PointToPointDumbbellHelper で書き換えた方が良いですね
125             NodeContainer net1_nodes;
126             net1_nodes.Create (2);
127
128             NodeContainer net2_nodes;
129             net2_nodes.Add (net1_nodes.Get(1));
130             net2_nodes.Create (1);
131
132             NodeContainer net3_nodes;
```

```
133             net3_nodes.Add (net1_nodes.Get(1));
134             net3_nodes.Create (1);
135
136             PointToPointHelper p2p1;
137             p2p1.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
138             p2p1.SetChannelAttribute ("Delay", StringValue ("2ms"));
139             p2p1.SetQueue ("ns3::DropTailQueue");
140
141             NetDeviceContainer devices1;
142             devices1 = p2p1.Install (net1_nodes);
143
144             NetDeviceContainer devices2;
145             devices2 = p2p1.Install (net2_nodes);
146
147             PointToPointHelper p2p2;
148             p2p2.SetDeviceAttribute ("DataRate", StringValue ("800Kbps"));
149             p2p2.SetChannelAttribute ("Delay", StringValue ("5ms"));
150             p2p2.SetQueue ("ns3::DropTailQueue");
151
152             NetDeviceContainer devices3;
153             devices3 = p2p2.Install (net3_nodes);
154
155             InternetStackHelper stack;
156             stack.InstallAll ();
157
158             Ipv4AddressHelper address;
159
160             address.SetBase (NET_ADD1, NET_MASK, FIRST_NO);
161             Ipv4InterfaceContainer ifs1 = address.Assign (devices1);
162             NS_LOG_INFO ("Network 1: " << ifs1.GetAddress(0, 0) << " - " <<
ifs1.GetAddress(1, 0));
163
164             address.SetBase (NET_ADD2, NET_MASK, FIRST_NO);
165             Ipv4InterfaceContainer ifs2 = address.Assign (devices2);
166             NS_LOG_INFO ("Network 2: " << ifs2.GetAddress(0, 0) << " - " <<
ifs2.GetAddress(1, 0));
167
168             address.SetBase (NET_ADD3, NET_MASK, FIRST_NO);
169             Ipv4InterfaceContainer ifs3 = address.Assign (devices3);
170             NS_LOG_INFO ("Network 3: " << ifs3.GetAddress(0, 0) << " - " <<
ifs3.GetAddress(1, 0));
171
172             NS_LOG_INFO ("Initialize Global Routing.");
173             Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

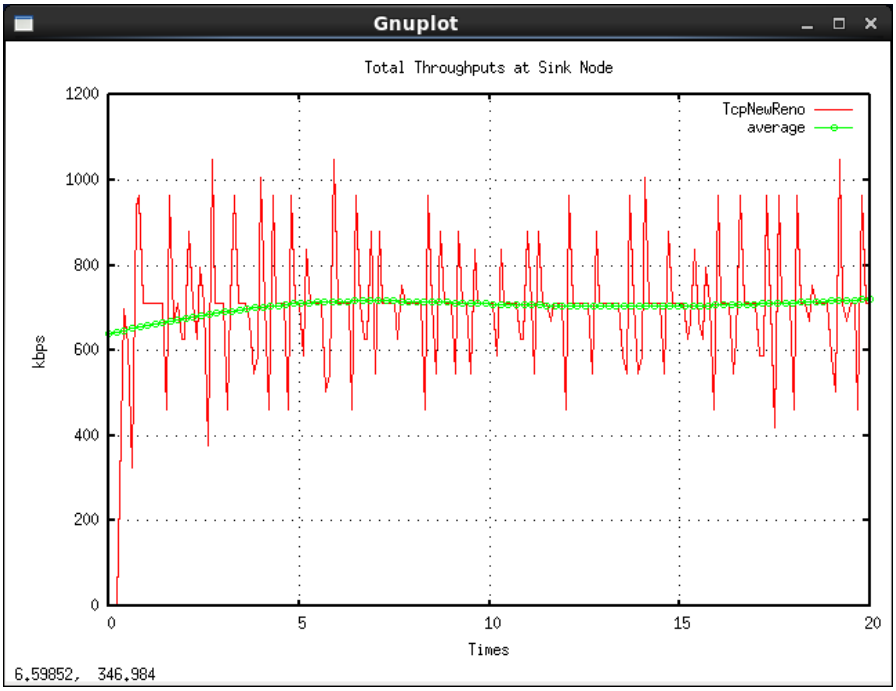
```
174             // OnOffHelper の設定を行います
175             Address srcAddress (InetSocketAddress (ifs1.GetAddress (0),
PORT));  // n0 の IP アドレスを取得して登録 n2 は、GetAny () でフォロー
176             OnOffHelper ftp ("ns3::TcpSocketFactory", srcAddress);
177             AddressValue remoteAddress (InetSocketAddress (ifs3.GetAddress(1,
0), PORT ));
178             ftp.SetAttribute ("Remote"      , remoteAddress);
179             ftp.SetAttribute ("DataRate"    , StringValue ("450kbps"));
180             ftp.SetAttribute ("PacketSize", UintegerValue (1024));
181             if(flowType.compare("Exponential") == 0) {
182                     // set on/off time randomly with exponential random
distribution
183                     ftp.SetAttribute ("OnTime" , StringValue
("ns3::ExponentialRandomVariable[Mean=0.352]"));        // 平均 0.352 秒
184                     ftp.SetAttribute ("OffTime", StringValue
("ns3::ExponentialRandomVariable[Mean=0.150]"));        // 平均 0.150 秒
185             } else if(flowType.compare("Uniform") == 0) {
186                     // set On/Off time randomly with uniform random
distribution(on/off in seconds)
187                     ftp.SetAttribute ("OnTime" , StringValue
("ns3::UniformRandomVariable[Min=1.0,max=5.00]"));    // [1 秒,5 秒]
188                     ftp.SetAttribute ("OffTime", StringValue
("ns3::UniformRandomVariable[Min=0.1,max=0.80]"));    // [0.1 秒,0.8 秒]
189             } else {
190                     ftp.SetAttribute ("OnTime" , StringValue
("ns3::ConstantRandomVariable[Constant=1]"));
191                     ftp.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
192             }
193         // n0 と n2 にインストールする
194             ApplicationContainer sourceApp1;
195             sourceApp1 = ftp.Install (net1_nodes.Get(0));
196             sourceApp1.Start (Seconds (SIM_START+0.1));
197             sourceApp1.Stop   (Seconds (SIM_STOP -0.1));
198
199             ApplicationContainer sourceApp2;
200             sourceApp2 = ftp.Install (net2_nodes.Get(1));
201             sourceApp2.Start (Seconds (SIM_START+0.1));
202             sourceApp2.Stop   (Seconds (SIM_STOP -0.1));
203
204             // create a sink to recieve packets @ network3's node 1(192.168.3.2).
205             Address sinkAddress (InetSocketAddress (Ipv4Address::GetAny (),
PORT));
206             PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",
```

```
sinkAddress);
    207
    208             //sinkHelper.SetAttribute ("Protocol", TypeIdValue
(TcpSocketFactory::GetTypeId ()));
    209             ApplicationContainer sink_apps = sinkHelper.Install
(net3_nodes.Get(1));
    210
    211             sink_apps.Start (Seconds (SIM_START+0.1));
    212             sink_apps.Stop   (Seconds (SIM_STOP -0.1));
    213
    214             AsciiTraceHelper ascii;
    215             std::string afname = std::string(PROG_DIR) + "simple-tcp.tr";
    216             p2p2.EnableAsciiAll (ascii.CreateFileStream (afname));
    217
    218             // make trace file's name
    219             std::string fname1 = std::string(PROG_DIR) + "exp03-" + tcpType + "-
" + flowType + ".cwnd";
    220             Ptr<OutputStreamWrapper> stream1 =
ascii.CreateFileStream(fname1);
    221             Simulator::Schedule (Seconds (CW_INTERVAL),
    222                     &MyEventHandller, net1_nodes.Get(0)->GetApplication(0),
stream1);
    223              // スループット時系列を記録するファイル名を作成する
    224             std::string fname2 = std::string(PROG_DIR) + "exp03-" + tcpType + "-
" + flowType + ".throughput";
    225             Ptr<OutputStreamWrapper> stream2 =
ascii.CreateFileStream(fname2);
    226             Simulator::Schedule (Seconds (TH_INTERVAL),
    227                     &TraceThroughput, net3_nodes.Get(1)->GetApplication(0),
stream2);    // こででスケジューラを呼び出している
    228
    229             // for PCAP Tracing
    230             //std::string pfname = std::string(PROG_DIR) + "simple-tcp";
    231             //p2p2.EnablePcapAll (pfname);
    232
    233             Simulator::Stop (Seconds(SIM_STOP));
    234             Simulator::Run ();
    235             Simulator::Destroy ();
    236
    237             return 0;
    238 }
```

スクリプトは、以下のように実行する。
$ ./waf△--run△"exp03-EstimateThroughput△--flowType=Constant"
$ ./waf△--run△"exp03-EstimateThroughput△--flowType=Exponential"
$ ./waf△--run△"exp03-EstimateThroughput△--flowType=Uniform"

実行結果として、cwnd の変化とスループットの変化が local/exp03/data/ に出力されている。
exp03-TcpNewReno-Constant.cwnd
exp03-TcpNewReno-Uniform.cwnd
exp03-TcpNewReno-Exponential.cwnd
exp03-TcpNewReno-Constant.throughput exp03-TcpNewReno-Uniform.throughput
exp03-TcpNewReno-Exponential.throughput

*.throughput には、計測されたスループット値が、0.1 秒毎に出力されている(単位は kb/s)。ファイルから値を読み出して、平均スループットを求めることができる。
このスループットは、n0 からのフローと n1 からのフローの両者の合計スループットである。local/exp03/ には、plot-exp03-throughput.pl が置いてある。
$ cd△local/exp03
$ gnuplot△plot-exp03-throughput.pl
で下記のようなグラフが得られる。.pl を読むと、グラフ中の平均スループットがどのように計算されているのかが分かる。



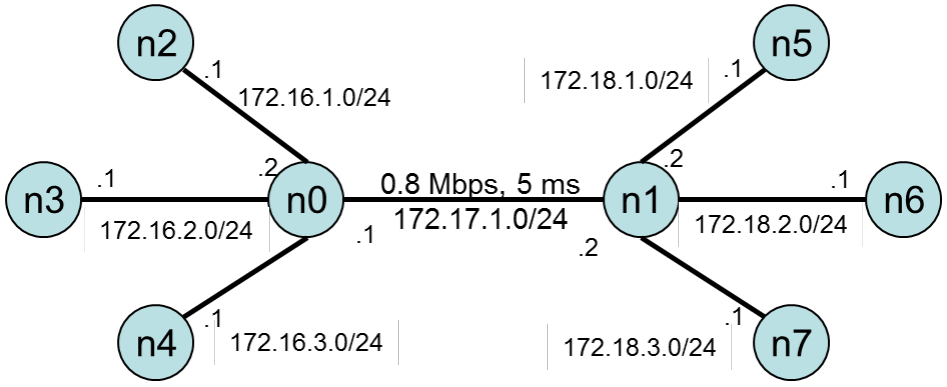
※ plot-exp03-throoughput.pl は、Constan OnOff 決め打ちである。Exponential 用、Uniform 用は自分で書き換えて作成する必要がある。
※ 実験 14：Constant OnOff は、On Time が連続な CBR であるため、Off Time がある

Exponential 及び Uniform と比較して高スループットとなることを確認してください。

※ 実験 15：スループットの観測周期 TH_INTERVAL を 0.1 から 0.8 まで 0.1 刻みで変化させ、平均スループットの曲線のみを１枚のグラフ(8 本の曲線が描かれる)にまとめてみましょう。
※ 実験 16：OnOff のパラメータを変更して、スループットがどのように変わるのかを見てみましょう。

6.3. フロー単位のスループット等の観測

ダンベルネットワークトポロジーを用いて、セッション毎の特性を評価する。ここでは FlowMonitorHelper の使い方を解説する。ソースファイルは local/exp04/ exp04-TcpFlowMonitoring.cc である。



実験用シナリオ

- n0-n1 間でボトルネックリンクを構成する。リンクは全二重、容量 800 kb/s、転送遅延時間 5ms
- n2、n3、n4 からルータ n0 間のリンクは、全二重、容量 10 Mb/s、転送遅延時間 1 ms
- 各ノードでは、DropTail 方式でキューの管理を行い、ボトルネックリンクのキューの最大サイズを 10 pkt とする
- 左側のネットワークは、172.16.1.0/24、ボトルネックネットワークは、172.17.1.0/24、右側のネットワークは、172.18.1.0/24 のアドレス空間を使用。アドレスの割当開始番号を 0.0.0.1 とする
- n2-n5 間、n3-n6 間、n4-n7 間に TCP NewReno の FTP フローを流す。左側のノードは source、右側のノードは sink である。使用するポート番号は 50000 とする
- OnOff 型 TCP フローを流し、On/Off Time は一様分布乱数で決定される。データの発生レートは 450 kb/s、パケットサイズは 1024 byte

実験方針

(1) トポロジーは、PointToPointDumbbellHelper を利用する

(2) フローモニターは、送受信したパケット数、転送遅延、ジッタ等、各フローに関する統計情報を収集するために利用する。フローモニターとしては、 FlowMonitorHelper を利用する。FlowMonitorHelper を使うためには、

FlowMonitorHelper△flowmon;

// 全てのノードに装着

Ptr<FlowMonitor>△monitor△=△flowmon.InstallAll(△);

// 個別ノードに装着

Ptr<FlowMonitor>△monitor△=△flowmon.Install(個別ノード);

のようにする。個別ノードにフローモニターを装着する際には、ノードポインタを指定する必要がある。ダンベルネットワークトポロジーでのノードポインタの取得は下記のように記述することで得られる。

PointToPointDumbbellHelper△dumbbell_net△(..., ..., ...);

dumbbell_net.GetLeft(i);　　// 左側ネットワークの i 番目のノードポンタの取得
dumbbell_net.GetRight(i);　// 右側ネットワークの i 番目のノードポインタの取得
dumbbell_net.GetLeft(△);　// 中央の左側ルータのノードポインタの取得
dumbbell_net.GetRight(△);　// 中央の右側ルータのノードポインタの取得

フローモニターは、ノードに装着後、シミュレーションを開始 Simulator::Run (); させる命令の後にフローの状態を観測する命令を追加する必要がある。
今までは
Simulator::Run ();
Simulator::Destroy ();
return 0;
であった。
フローの状態を取得するには、GetFlowStats メソッドを利用する。記述は以下、
std::map<FlowId,△FlowMonitor::FlowStats>△stats△=△monitor->GetFlowstats(△);
このメソッドでは、各フローID に対応した状態を取得するので、戻り値は以下の map クラスで表現される。
map<FlowId,△FlowMonitor::FLowStats>
実際に取得する情報は以下のような形となる。
flow_id△1,△flow1 のステータス構造体(FlowStats)
flow_id△2,△flow2 のステータス構造体(FlowStats)
flow_id△3,△flow3 のステータス構造体(FlowStats)
・・・
構造体 FlowStats は、src/flow-monitor/model/flow-monitor.h で定義されている

変数型	変数名	意味
Time	timeFirstTxPacket	フローの送信開始時刻
Time	timeFirstRxPacket	フローの受信開始時刻
Time	timeLastTxPacket	フローの送信終了時刻
Time	timeLastRxPacket	フローの受信終了時刻
Time	delaySum	フローの受信した全パケットの伝送遅延時間の合計
Time	jitterSum	フローの受信した全パケットの伝送遅延ジッタの合計
Time	lastDelay	フローの最後に計測したパケットの遅延時間
uint64_t	txBytes	フローの送信バイト総数
uint64_t	rxBytes	フローの受信バイト総数
uint32_t	txPackets	フローの送信パケット総数
uint32_t	rxPackets	フローの受信パケット総数
uint32_t	lostPackets	フローの損失パケット総数
uint32_t	timesForwarded	フローの転送パケット総数
Histogram	delayHistogram	フローのパケット遅延時間のヒストグラム
Histogram	jitterHistogram	フローの遅延ジッタのヒストグラム
Histogram	packetSizeHistogram	フローのパケットサイズのヒストグラム

(3) 各フローから IP アドレス情報の抽出を行う。具体的には、フローモニターの GetClassifier メソッドを利用して、該当フローの FlowClassifier オブジェクトを取得し、

そのポインタを Ipv4FlowClassifier に渡す。NS3 では、オブジェクトのインスタンス型を変換するために DynamicCast が用意されている。

```
Ptr<Ipv4FlowClassifier>△classifier△=
    DynamicCast<Ipv4FlowClassifier>△(flowmon.GetClassfier△(△));
```

さらに、Ipv4FlowClassifier の FindFlow メソッドを利用することで、各フローID に対応する FiveTuple (Source IP, Destination IP, Protocol, Source Port, Destination Port)の構造体を取得できる。

exp04-TcpFlowMonitoring.cc

```
 1 /* -*- Mode:C++; c-file-style:"gnu"; indent-tabs-mode:nil; -*- */
 2 /* exp04-FlowMonitoring.cc: simulate TCP behaviours with flow monitor
 3  * F.Qian, Nov. 2012
 4  *
 5  *          Source              n2                                    n5    Sink
 6  *     (10Mbps,1ms)              \                                   / (10Mbps,1ms)
 7  *                                \ .1.0/24              .1.0/24   /
 8  *                                 \       bottleneck link       /
 9  * net1(172.16.0.0/16) n2 --n0 --net3(0.8Mbps,5ms)-- n1 -- n6
net2(172.17.0.0/16)
10  *                      .2.0/24    /       172.18.1.0/24          \     .2.0/24
11  *                               / .3.0/24              .3.0/24   \
12  *                              /                                    \
13  *        Left Leaf           n3                                     n7       Right
Leaf
14  */
15
16 #include <iostream>
17 #include <fstream>
18 #include <string>
19 #include <vector>
20
21 #include "ns3/core-module.h"
22 #include "ns3/network-module.h"
23 #include "ns3/internet-module.h"
24 #include "ns3/point-to-point-module.h"
25 #include "ns3/applications-module.h"
26 #include "ns3/ipv4-global-routing-helper.h"
27 #include "ns3/tcp-header.h"
28
29 #include "ns3/point-to-point-layout-module.h"
30 #include "ns3/flow-monitor-module.h"
```

```
31 #include "ns3/netanim-module.h"
32
33 #define NET_MASK          "255.255.255.0"
34 #define NET_ADDR1         "172.16.1.0"      // left leaf network
35 #define NET_ADDR2         "172.18.1.0"      // right leaf network
36 #define NET_ADDR3         "172.17.1.0"      // bottleneck link
37 #define FIRST_NO          "0.0.1.1"         //  この指定の仕方は初出ですね
38
39 #define SIM_START          00.0             // 0.00 秒開始
40 #define SIM_STOP           40.0             // 40.0 秒終了
41 #define DATA_MBYTES         500             // 500 MB 送信
42 #define PORT               50000            //  ポート番号
43
44 #define PROG_DIR           "local/exp04/data/"
45
46 using namespace ns3;
47
48 NS_LOG_COMPONENT_DEFINE ("exp04-SimTcpFlowMonitor");
49
50 int
51 main (int argc, char *argv[])
52 {
53         std::string animFile = "flow-monitoring.xml" ;   // Name of file for
animation output
54         CommandLine cmd;
55         cmd.Parse(argc, argv);
56
57         uint32_t      nLeftLeaf   = 3;
58         uint32_t      nRightLeaf = 3;
59
60         Config::SetDefault ("ns3::OnOffApplication::PacketSize",
UintegerValue (1024));
61         Config::SetDefault ("ns3::OnOffApplication::DataRate"    ,
StringValue ("500kb/s"));   //  後で 450 kb/s  に設定
62         Config::SetDefault ("ns3::DropTailQueue::MaxPackets"     ,
UintegerValue (100));    //  下でボトルネックリンクは  10  に設定
63          //  中央ネットワークの属性設定
64         // set bottkeneck link
65         PointToPointHelper p2pRouter;
66         p2pRouter.SetDeviceAttribute    ("DataRate", StringValue ("1Mbps"));
// 800kbps  ではないのか？
67         p2pRouter.SetChannelAttribute ("Delay", StringValue ("5ms"));
68         p2pRouter.SetQueue ("ns3::DropTailQueue", "MaxPackets",
UintegerValue (10) );
```

```
69              // 両側ネットワークの属性指定
70              PointToPointHelper p2pLeaf;
71              p2pLeaf.SetDeviceAttribute      ("DataRate", StringValue
("10Mbps"));
72              p2pLeaf.SetChannelAttribute     ("Delay", StringValue ("1ms"));
73              p2pLeaf.SetQueue ("ns3::DropTailQueue");
74
75              // make dumbble network topology
76              PointToPointDumbbellHelper dumbbell_net (nLeftLeaf, p2pLeaf,
nRightLeaf, p2pLeaf, p2pRouter);
77
78              // Install Stack
79              InternetStackHelper stack;
80              dumbbell_net.InstallStack (stack);
81
82              // Assign IP Addresses
83              dumbbell_net.AssignIpv4Addresses (
84                      Ipv4AddressHelper (NET_ADDR1, NET_MASK),
85                      Ipv4AddressHelper (NET_ADDR2, NET_MASK),
86                      Ipv4AddressHelper (NET_ADDR3, NET_MASK));
87
88              // Install on/off app on all right side nodes
89              OnOffHelper tcp ("ns3::TcpSocketFactory", Address ());
90              tcp.SetAttribute ("OnTime" , StringValue
("ns3::UniformRandomVariable[Min=3.0|Max=5.00]"));   // On 3 秒～5 秒
91              tcp.SetAttribute ("OffTime", StringValue
("ns3::UniformRandomVariable[Min=0.1|Max=0.50]"));   // Off 0.1 秒～0.5 秒
92              tcp.SetAttribute ("DataRate"    , StringValue ("450Kbps"));
93              tcp.SetAttribute ("PacketSize", UintegerValue (1024));
94
95              ApplicationContainer source;
96
97              for (uint32_t i = 0; i < dumbbell_net.LeftCount (); i++) {
98                      // Create an on/off app sending packets to the same leaf right
side
99                      AddressValue remoteAddress (InetSocketAddress
(dumbbell_net.GetRightIpv4Address (i), PORT));   // 宛先アドレスを設定
100                     tcp.SetAttribute ("Remote", remoteAddress);
101                     source.Add (tcp.Install (dumbbell_net.GetLeft (i))); // 送信元
アドレスを設定
102             }
103             source.Start (Seconds (SIM_START));
104             source.Stop    (Seconds (SIM_STOP));
105
```

```
106             ApplicationContainer sink;
107             for (uint32_t i = 0; i < dumbbell_net.RightCount (); i++) {
108                     Address sinkAddress (InetSocketAddress
(dumbbell_net.GetRightIpv4Address (i), PORT));
109                     PacketSinkHelper sinkHelper ("ns3::TcpSocketFactory",
sinkAddress) ;
110                     sinkHelper.SetAttribute ("Local", AddressValue
(sinkAddress));
111                     sink.Add (sinkHelper.Install (dumbbell_net.GetRight (i)));
112             }
113             sink.Start (Seconds (SIM_START));
114             sink.Stop    (Seconds (SIM_STOP ));
115
116             // Set the bounding box for animation
117             dumbbell_net.BoundingBox (1, 1, 100, 100);   // アニメーション用
118
119             // Create the animation object and configure for specified output
120             //AnimationInterface anim (animFile);   // 実際にはアニメーションし
ない
121
122             // Set up the acutal simulation       いつものおまかせルーティング
123             Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
124
125             // set Flowmonitor
126             FlowMonitorHelper flowmon;
127             flowmon.SetMonitorAttribute("JitterBinWidth",
ns3::DoubleValue(0.001));   // ヒストグラムの時間間隔
128             flowmon.SetMonitorAttribute("DelayBinWidth",
ns3::DoubleValue(0.001));   // ヒストグラムの観測時間間隔
129             flowmon.SetMonitorAttribute("PacketSizeBinWidth",
ns3::DoubleValue(20));   // ヒストグラムのパケットサイズ間隔
130             Ptr<FlowMonitor> monitor = flowmon.InstallAll();
131
132             Simulator::Stop (Seconds(SIM_STOP));
133             Simulator::Run ();
134
135             // the following lines must setup after Run()
136             monitor->CheckForLostPackets ();
137             Ptr<Ipv4FlowClassifier> classifier =
DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
138             std::map<FlowId, FlowMonitor::FlowStats> stats = monitor-
>GetFlowStats ();
139
140             std::cout << "-----------------------------------" << std::endl;
```

```
141             for(std::map<FlowId,FlowMonitor::FlowStats>::const_iterator
i=stats.begin(); i!=stats.end(); ++i) {
142                     Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i-
>first);
143     // 観測対象のフローを指定して計測させる
144                     if((t.sourceAddress=="172.16.1.1" && t.destinationAddress
== "172.18.1.1")
145                      ||(t.sourceAddress=="172.16.2.1" && t.destinationAddress
== "172.18.2.1")
146                      ||(t.sourceAddress=="172.16.3.1" && t.destinationAddress
== "172.18.3.1")) {
147
148                             std::cout << "Flow " << i->first  << " (" <<
t.sourceAddress
149                                     << " -> " << t.destinationAddress << ")\n";
150
151                             // The following flow stats can be find in src/flow-
monitor/model/flow-monitor.h
152
153                             // Contains the sum of all end-to-end delays for all
received packets of the flow.
154                             std::cout << "  Delay sum:  " << i-
>second.delaySum << "\n";
155
156                             // Contains the sum of all end-to-end delay jitter
(delayvariation) values for
157                             // all received packets of the flow.
158                             std::cout << "  Jitter sum:  " << i-
>second.jitterSum << "\n";
159
160                             // Total number of transmitted bytes for the flow.
161                             std::cout << "  Tx Bytes:  " << i-
>second.txBytes <<"\n";
162
163                             // Total number of received bytes for the flow.
164                             std::cout << "  Rx Bytes:  " << i-
>second.rxBytes <<"\n";
165
166                             // Total number of transmitted packets for the flow.
167                             std::cout << "  Tx Packets:  " << i-
>second.txPackets << "\n";
168
169                             // Total number of received packets for the flow.
170                             std::cout << "  Rx Packets:  " << i-
```

```
>second.rxPackets << "\n";
      171
      172                                 // Total number of packets that are assumed to be
lost, i.e. those that were
      173                                 // transmitted but have not been reportedly received
or forwarded for a long
      174                                 // time.
      175                                 std::cout << "     lost Packets:     " << i-
>second.lostPackets << "\n";
      176
      177                                 // Contains the number of times a packet has been
reportedly forwarded,
      178                                 // summed for all received packets in the flow
      179                                 std::cout << "Times Forwarded:     " << i-
>second.timesForwarded << "\n\n";
      180
      181                                 std::cout << "    Throughput: " << i->second.rxBytes
* 8.0
      182                                           / (i-
>second.timeLastRxPacket.GetSeconds()
      183                                           - i-
>second.timeFirstTxPacket.GetSeconds())/1024<< " Kbps\n";
      184                                 std::cout << "-------------------------------------" <<
std::endl;
      185                           }
      186             }
      187
      188             // make trace file's name
      189             AsciiTraceHelper ascii;
      190             std::string fname = std::string(PROG_DIR) + "exp04-TCP-
flowmon.xml";
      191             monitor->SerializeToXmlFile(fname, true, true);
      192
      193             //std::cout << "Animation Trace file created:" << animFile.c_str ()<<
std::endl;
      194             Simulator::Destroy ();
      195
      196             return 0;
      197 }
      198
```

※ 120 行のコメントをはずして、NetAnim が使えるようにしておきましょう
※ UDP の場合は、89 行を OnOffHelper△udp(”ns3::UdpScoketFactory”,△Address());
に変更し、関連する 90-93、100-101 行を udp に変更。106-112 行は不要。

※ 実験 17：UDP フローの平均スループットを求め、TCP の場合と比較して考察してください。
※ 実験 18：TCP フローと UDP フローを混在させ、各フローの平均スループットを考察してください

7. NS3 でのモジュール開発例

NS3 の各モジュールは$NS_HOME/src の下にそれぞれ独立したディレクトリが用意され、その中にモジュール、ヘルパー、テストスクリプト、サンプルスクリプト、ドキュメントというディレクトリで構成される。新しいモジュールを開発するということは、src の下にディレクトリを準備し、必要なファイルを用意しなければならない。ディレクトリ名がそのままモジュール名となる。
NS3 には、create-module.py という python スクリプトが用意されており、それを利用することで雛形を作成することができる。
ここでは、新規モジュールとして RoundRobin Scheduler を作成する。

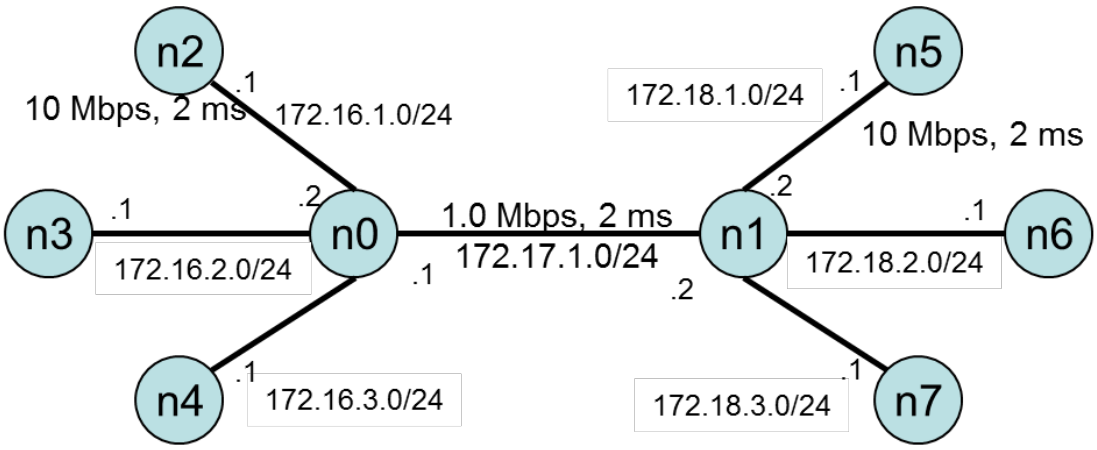
モジュール名称 RRQueue
$ cd△$NS_HOME/src
$ python△create-module.py△queue
Creating module 'queue', run './waf configure' to include it in the build
と出るので、後で ./waf configure してあげないといけない
次に、queue/wscript を下記のように書き換える。今回は、ヘルパーを作らない、またソースファイル名は、rr-queue.* とする。

```
1 # -*- Mode: python; py-indent-offset: 4; indent-tabs-mode: nil; coding: utf-8; -*-
2
3 # def options(opt):
4 #     pass
5
6 # def configure(conf):
7 #     conf.check_nonfatal(header_name='stdint.h', define_name='HAVE_STDINT_H')
8
9 def build(bld):
10     module = bld.create_ns3_module('queue', ['core'])
11     module.source = [
12         'model/rr-queue.cc',
13 ##         'helper/quue-helper.cc',
14         ]
15
16 ##     module_test = bld.create_ns3_module_test_library('queue')
17 ##     module_test.source = [
```

```
18 ##              'test/queue-test-suite.cc',
19 ##              ]
20
21       headers = bld(features='ns3header')
22       headers.module = 'queue'
23       headers.source = [
24           'model/rr-queue.h',
25 ##          'helper/queue-helper.h',
26           ]
27
28       if bld.env.ENABLE_EXAMPLES:
29           bld.recurse('examples')
30
31       # bld.ns3_python_bindings()
32
```

ソースファイルは、local/rr-queue の下に存在しているので、src/queue の中にコピーします。
$ cd△$NS_HOME/src/queue
$ cp△-r△../../local/rr-queue/src-queue/*△.

NS3 の再構築をします。
$ cd△$NS_HOME
$ ./waf△--enable-examples△--enable-tests△configure
$ ./waf△build



local/rr-queue/rr-queue-example.cc に、RRqueue と DropTailqueue を切替て利用可能なサンプルが置いてあるので、scratch/ にコピーします。
$ cp△local/rr-queue/rr-queue-example.cc△scratch/

ただし、rr-queue-example.cc は、 customized-app.h という src/queue/examples/ にあるヘッダを参照しているので、該当部分を

#include△”src/queue/examples/customized-app.h”
に書き換えることが必要である。

RRqueue の場合は、下記で実行
$ waf△--run△”rr-queue-example△--queueType=RR”

※ エラーになって動かないかもしれません。。。。。

DropTailqueue の場合は、下記で実行
$ waf△--run△”rr-queue-example△--queueType=DropTail”

※ 実験 19：実験結果の Fairness index を、自分で計算してみましょう。定義式は講義資料に出ています。
※ RRqueue の結果がエラーになった場合、以下の値を使ってください。
Flow 1 Throughput: 805.75 kbps
Flow 2 Throughput: 812.335 kbps
Flow 3 Throughput: 814.267 kbps
Flow 4 Throughput: 808.485 kbps
Flow 5 Throughput: 800.357 kbps
Flow 6 Throughput: 810.228 kbps

以上