Boeing Quagga Software

Installation Guide and User's Manual

Authors:
Phillip A. Spagnolo
Thomas Goff
Thomas R. Henderson
Gary Pei

Copyright © 2006-08, The Boeing Company.

Table of Contents

1 Introduction 1
1.1 Licensing 1

2 Prerequisites 2
2.1 Patching the source code 2
2.2 OSPF-MANET 2

3 OSPFv3 Address Families 3
3.1 Overview 3
3.1.1 Draft compliance 3
3.1.2 License and Contributing Code 3
3.2 Enabling OSPF-AF 3
3.3 Configuring OSPF-AF 3
3.4 Running OSPF-AF 4
3.5 Open Issues 4

4 OSPF-MANET 5
4.1 Overview 5
4.1.1 Draft compliance 5
4.1.2 License and Contributing Code 5
4.1.3 Contributors 5
4.2 Protocol Operation 6
4.3 Enabling OSPF-MANET 6
4.4 Building OSPF-MANET 6
4.5 Configuring OSPF-MANET 6
4.6 Running OSPF-MANET 7
4.7 Use with Address Families 7
4.7.1 Redistribution between OSPFv2 and OSPFv3 MANET 8
4.8 OSPF-MANET Configuration Examples 8
4.9 OSPFv3 Simulation with GTNetS 9
4.9.1 Simulation 11
4.9.1.1 Patching the Quagga source code 11
4.9.1.2 Building the Simulator 11
4.9.1.3 Running the Simulator 12
4.9.1.4 Configuring the Simulator 12
4.9.1.5 Simulation Output 23
4.9.1.6 Validation 26
4.9.2 License and Contributing Code 28

1 Introduction

This manual describes installation and usage of Quagga software developed by Boeing.

There are four sets of software patches.

- Patch to enable OSPF extensions to quagga OSPFv3
- Patch to enable OSPF AF functionality in quagga OSPFv3
- Patch to enable OSPF MANET MDR functionality in quagga OSPFv3

1.1 Licensing

All software has been cleared for public release as open source, but it is important for users to understand the licensing associated with each piece of software.

Boeing's OSPF-MANET related software is a derivative work of the Quagga routing suite, which is licensed under the GNU General Public License (GPL) version 2. Therefore, the quagga-0.99.9 patches are provided under GPL version 2 (Copyright 2008 Boeing):


```
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

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., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301, USA.
```

2 Prerequisites

The system requires a relatively modern Linux distribution (with GNU development tools such as the gcc compiler and make) and related packages. We have tested this with Fedora Core 5 and 6 distributions and FreeBSD 7. Root privileges are needed.

As for host hardware requirements, either a Linux computer with an Ethernet connection, or a Linux virtual machine on some other (e.g., Windows) operating system, should work.

2.1 Patching the source code

The software is provided as a few patches because you may want to apply the patch to some other base release of the software (and it may still work).

2.2 OSPF-MANET

There are four patches provided. Each patch provides incrementally more functionality. **You should select and apply only one of the four patches!**

First, unpack quagga from quagga.net:

```
tar xvfz quagga-0.99.9.tar.gz
```

Next, select a patch to apply:

1. Boeing's OSPFv3 Extensions patch: quagga-0.99.9.ospfv3-extensions.patch
2. Boeing's OSPFv3 Address Families patch: quagga-0.99.9.ospfv3-addressfamilies.patch
3. Boeing's OSPFv3 MANET Designated Routers (MDR) patch: quagga-0.99.9.ospfv3-manetmdr.patch
4. Boeing's OSPFv3 MDR for GTNetS (simulator) patch: quagga-0.99.9.ospfv3-manetmdr-gtnets.patch

Patch 1 provides just some OSPFv3 extensions, intended for general OSPF enhancement. Patch 2 (Address Families) provides a patch corresponding to the Address Families extension for OSPFv3, but no MANET software. Patch 3 provides the code in patches 1 and 2, and adds the OSPF MANET MDR code. Patch 4 provides extensions to the MDR so that it funcionts in GTNetS. If in doubt, apply patch 3:

```
patch -p1 < quagga-0.99.9.ospfv3-manetmdr.patch
```

3 OSPFv3 Address Families

The following text describes the implementation of a mechanism for supporting multiple address families in OSPFv3 using multiple instances. It maps an address family (AF) to an OSPFv3 instance using the Instance ID field in the OSPFv3 packet header. This approach is fairly simple and minimizes extensions to OSPFv3 for supporting multiple AFs.

This implementation also enables OSPF MANET to support IPv4 routing (next chapter).

3.1 Overview

For now, please see Section 2 of draft-ietf-ospf-af-alt-05.

3.1.1 Draft compliance

Support of Address Families (AF) in OSPFv3 is supported according to draft-ietf-ospf-af-alt-05.

3.1.2 License and Contributing Code

This software builds on two projects (GTNetS and quagga) that are under GNU General Public License (GPL). Our modifications are also under GPL. Therefore, please consider this simulator as GPL'ed software.

We would like to encourage people to send in bug fixes, extensions, or to make their simulation scripts available (that produce their simulation results). By default, we will not include any contributed code, bug fixes, or patches unless you specify that you want to include said code in our future releases of this simulator (also under GPL).

3.2 Enabling OSPF-AF

Address families is enabled by compiling the flag `OSPF6_AF` into the source code. By default this flag is compiled into the code. It must be removed from configure.ac and config.h.in.

All modified code is found in the directories lib/, zebra/, and ospf6/. Each modification is flagged with the tag "OSPF6_AF".

3.3 Configuring OSPF-AF

Address Families can be configured in one of two ways.

1. Add the following lines to the ospf6d.conf file

```
interface <ifname>
ipv6 ospf6 instance-id <0-255>
```

2. From the vtysh or telnet terminal type:

```
> conf t
> interface <ifname>
> ipv6 ospf6 instance-id <0-255>
> exit
> exit
```

The value of the instance ID should one of the four ranges below. The most comman ranges are 0 to 31 for unicast IPv6 routing (standard OSPFv3) and 64 to 95 for IPv4 unicast routing.

```
Instance ID # 0    -  # 31     IPv6 unicast AF
Instance ID # 32   -  # 63     IPv6 multicast AF
Instance ID # 64   -  # 95     IPv4 unicast AF
Instance ID # 96   -  # 127    IPv4 multicast AF
Instance ID # 128  -  # 255    Unassigned
```

NOTE: The instance-id must be the same on all interfaces. Different Address Families cannot be used within the same ospf6d process. The router will fail if different ranges are used.

3.4 Running OSPF-AF

From a vtysh or telnet terminal type:

```
> show ipv6 ospf6 route
```

This should display the OSPFv3 routes. If IPv4 AFs are used then the route will appear as an IPv6 route with zeros before the IPv4 route. Next, type the following command for IPv4 or IPv6

```
> show ip route
> show ipv6 route
```

The entries with the `"*"` are going to be installed in the kernel routing table. If these tables are correct then the kernel routing table should be correct.

3.5 Open Issues

Enable different AFs to run in the same ospf6d instance. This would require a IETF draft changes and a sepeartion of LSAs within the database.

Known Issue: if instance IDs are not consistent on the interfaces then routing will fail.

4 OSPF-MANET

`OSPF-MANET` is a modification of OSPF version 3 (IPv6) for use in mobile ad hoc networks (MANETs). OSPF for IPv6 is described in RFC2740.

This chapter is a supplement to the main quagga (`http://www.quagga.net`) documentation. It describes the implementation, functionality, and usage of OSPF-MANET and related extensions.

4.1 Overview

OSPF-MANET can be built for typical quagga usage as a standalone router, for support in virtual machines such as IMUNES, and within a discrete-event network simulator.

OSPF-MANET is defined for IPv6 (OSPFv3). With the addition of what is known as the *Address Families* patch, an instance of OSPF-MANET can also be run to build IPv4 routes. Note that to get both IPv4 and IPv6 routing, two instances of OSPFv3 must be running, as presently defined by the draft standard.

OSPF-MANET is distributed as a series of patches against a mainline quagga distribution. The Boeing server is located at `http://hipserver.mct.phantomworks.org/ietf/ospf/`.

4.1.1 Draft compliance

draft-ogier-manet-ospf-extension-09

4.1.2 License and Contributing Code

This software builds on two projects (GTNetS and quagga) that are under GNU General Public License (GPL). Our modifications are also under GPL. Therefore, please consider this simulator as GPL'ed software.

We would like to encourage people to send in bug fixes, extensions, or to make their simulation scripts available (that produce their simulation results). By default, we will not include any contributed code, bug fixes, or patches unless you specify that you want to include said code in our future releases of this simulator (also under GPL).

4.1.3 Contributors

This OSPF-MANET software is the product of a number of individuals, including:

- Jeff Ahrenholz
- Claudiu Danilov
- Tom Henderson
- Jeff Meegan
- Richard Ogier
- Gary Pei
- Phil Spagnolo

This research was funded by the Office of Naval Research under contract N00014-04-C-0003 and Boeing internal research. The simulator developed for this project is an extension of two open-source software projects: the Georgia Tech Network Simulator (GTNetS) and the Quagga Routing Suite.

4.2 Protocol Operation

For now, please see http://hipserver.mct.phantomworks.org/ietf/ospf/milcom06.pdf or Section 2 of draft-ietf-ospf-manet-mdr-00

4.3 Enabling OSPF-MANET

OSPF-MANET is enabled by compiling the flags into the source code.

- BUGFIX: Bugs found in the quagga source.
- OSPF6_CONFIG: Flag to add new configuration commands.
- OSPF6_DELAYED_FLOOD: Delay the flooding of an LSA for coalescing.
- OPF6_MANET_MDR: Enable the use of OSPF-MANET MDR is OSPFv3.
- OSPF6_DD_OPT: Enable an optimization to the database exchange algorithm.
- OSPF6_MANET_DIFF_HELLO: Enable differential Hellos in OSPF-MANET MDR.
- OSPF6_SMF: Hook to allow SMF to use MDR relay set.

By default these flags are compiled into the code. They must be removed from configure.ac to disable.

To support IPv4 routing using OSPF-MANET, one must also define the Address Families patch (described in another chapter).

4.4 Building OSPF-MANET

To build quagga as standalone router run:

```
autoreconf or ./update-autotools
./configure --enable-user=root --enable-group=root --enable-vtysh \
--with-cflags=-ggdb
make
make install
```

To build quagga in IMUNES (or Boeing's CORE), use the following configure line:

```
./configure --enable-user=root --enable-group=wheel \
     --sysconfdir=/usr/local/etc/quagga --enable-vtysh \
     --localstatedir=/var/run/quagga --with-cflags=-ggdb
```

4.5 Configuring OSPF-MANET

OSPF-MANET can be configured in one of two ways: command line interface (CLI) or config file (ospf6d.conf). In either case, you must install a zebra.conf and ospf6d.conf file in /usr/local/etc/.

- CLI: run configuration commands in vtysh or telnet
- put configuration commands in zebra.conf and ospf6d.conf

Here are the configuration commands added during the development of OSPF-MANET MDR.

```
* <code> router ospf6 </code>
** <code> router minls-interval <0-65535> </code>:  Minimum time between LSA origination.
** <code> router minls-arrival <0-65535> </code>:  Minimum time between LSA reception.
```

```
* <code> interface <ifname> </code>:  Select the interface to configure
** <code> ipv6 ospf6 network (broadcast|non-broadcast|point-to-multipoint|point-to-point|lo
*** broadcast:  Specify OSPF6 broadcast multi-access network
*** non-broadcast:  Specify OSPF6 NBMA network
*** point-to-multipoint:  Specify OSPF6 point-to-multipoint network
*** point-to-point:  Specify OSPF6 point-to-point network
*** loopback:  Specify OSPF6 loopback
*** manet-designated-router:  Specify OSPF6 manet-designated-router (MDR) network
** <code> ipv6 ospf6 flood-delay <1-65535> </code>:  Time in msec to coalesce LSAs before s
** <code> ipv6 ospf6 jitter <1-65535> </code>:  Time in msec to jitter sending of all ospf6
** <code> ipv6 ospf6 ackinterval <1-65535> </code>:  Interval of time to coalesce acks
** <code> ipv6 ospf6 backupwaitinterval <1-65535></code> :  Interval of time for MBDRs to w
** <code> ipv6 ospf6 diffhellos </code>:  Enable differential hellos
** <code> ipv6 ospf6 twohoprefresh <1-65535></code> :  When using differential Hellos, full
** <code> ipv6 ospf6 hellorepeatcount <1-65535> </code>:  Total hellos in succession that c
** <code> ipv6 ospf6 adjacencyconnectivity (uniconnected|biconnected|fully) </code>:  Level
*** uniconnected:  The set of adjacencies forms a (uni)connected graph.
*** biconnnected:  The set of adjacencies forms a biconnected graph.
*** fullyconnected: Adjacency reduction is not used, the router becomes adjacent with all o
** <code> ipv6 ospf6 lsafullness (minlsa|mincostlsa|mincost2lsa|mdrfulllsa|fulllsa) </code>
*** minlsa:  Specify min size LSAs (only adjacent neighbors)
*** mincostlsa:  Specify partial LSAs for min-hop routing
*** mincost2lsa:  Specify partial LSAs for two min-hop routing paths
*** mdrfulllsa:  Specify full LSAs from MDR/MBDRs
*** fulllsa:  Specify full LSAs (all routable neighbors)
** <code> ipv6 ospf6 smf_mdr_talk:  Allow SMF to use the MDR relay set.
```

4.6 Running OSPF-MANET

Run the following commands for the command prompt:

```
/usr/local/sbin/zebra -d
/usr/local/sbin/ospf6d -d
```

To verify OSPF-MANET is running, from a vtysh or telnet terminal type:

```
> show ipv6 ospf6 route
```

This should display the OSPFv3 routes. If IPv4 AFs are used then the route will appear as an IPv6 route with zeros before the IPv4 route. Next, type the following command for IPv4 or IPv6

```
> show ip route
> show ipv6 route
```

The entries with the "*" are going to be installed in the kernel routingtable. If these tables are correct then the kernel routing table should be correct.

4.7 Use with Address Families

To use OSPF MANET to carry IPv4 prefix information, one may enable it with the following configuration.

In the interface description section, define an instance-id greater than 64. Such as:

```
interface ath0
  ...
  ipv6 ospf6 instance-id 65
  ...
```

Then, in the router definition section, describe networks to be associated with OSPF MANET.

```
router ospf6
 router-id 10.1.0.1
 interface ath0 area 0.0.0.0
```

4.7.1 Redistribution between OSPFv2 and OSPFv3 MANET

(to be completed)

4.8 OSPF-MANET Configuration Examples

Here is an example of an interface declaration of an OSPF-MANET interface, from the `ospf6d.conf` file.

```
interface ath0
  ipv6 ospf6 priority 1
  ipv6 ospf6 transmit-delay 1
  ipv6 ospf6 instance-id 65
  ipv6 ospf6 ifmtu 1500
  ipv6 ospf6 cost 1
  ipv6 ospf6 hello-interval 2
  ipv6 ospf6 dead-interval 6
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 network manet-designated-router
  ipv6 ospf6 ackinterval 1800
  ipv6 ospf6 diffhellos
  ipv6 ospf6 backupwaitinterval 2000
  ipv6 ospf6 twohoprefresh 3
  ipv6 ospf6 hellorepeatcount 3
  ipv6 ospf6 adjacencyconnectivity biconnected
  ipv6 ospf6 lsafullness mdrfulllsa
  ipv6 ospf6 flood-delay 100
!
```

The below router declaration example tells quagga to run OSPF-MANET on interface ath0 and to redistribute OSPF and connected networks.

```
router ospf6
 router-id 10.1.0.1
 interface ath0 area 0.0.0.0
 redistribute ospf
 redistribute connected
!
```

4.9 OSPFv3 Simulation with GTNetS

This section describes the implementation of MANET extensions to OSPFv3 as an extension to the `quagga`[1] OSPFv3 routing daemon and the Georgia Tech Network Simulator (GTNetS).www.ece.gatech.edu/research/labs/MANIACS/GTNetS/ Figure 4.1 illustrates how the same quagga software is supported in an actual quagga implementation and also wrapped within the GTNetS simulation framework; this enables moving between implementation and simulation with the same code base, just by changing some compilation flags.

[1] http://www.quagga.net

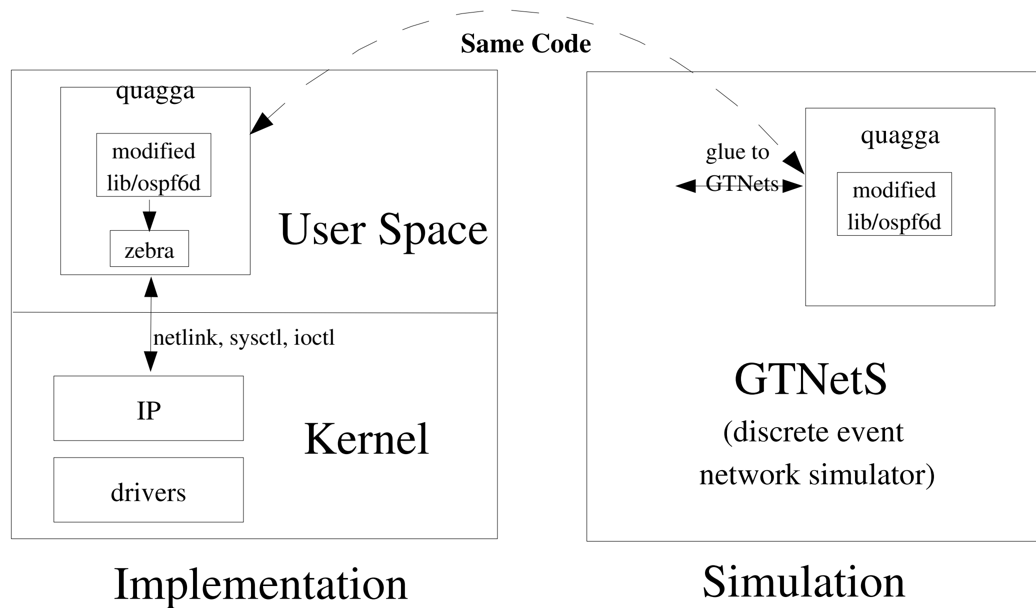

Figure 4.1: Architecture of Simulation and Implementation

This packet-level, discrete-event simulator has the following features:

- OSPFv3 for IPv6 based on recent `quagga` software (version 0.99.9).
- Support for enabling a proposal known as **MANET Designated Routers**, contributed by Richard Ogier and Phil Spagnolo draft-ietf-ospf-manet-mdr-00 .
- (slightly modified) 802.11 and wired channel models
- detailed logging and tracing of simulation events, including all quagga daemon logging

We have not removed any features of GTNetS, which are described elsewhere in the GTNetS documentation, or `quagga`, although we note that only OSPFv3 is supported in

the simulation (the full `quagga` routing daemon is available if run as implementation code). Here, we focus on the types of OSPF MANET simulations that we have done with this simulator.

The next section describes the details for using this software in *simulation* mode.

4.9.1 Simulation

4.9.1.1 Patching the Quagga source code

Before using the simulator you must hook GTNetS and Quagga together. Follow the following steps:

- Obtain the GTNetS and Quagga source code.
- Obtain the MANET-MDR GTNetS for Quagga: quagga-0.99.9.ospfv3-manetmdr-gtnets.patch
- Patch the source code: `patch -p1 < quagga-0.99.9.ospfv3-manetmdr-gtnets.patch`
- Run `./configure` in the quagga directory
- Comment out #define VERSION "0.99.9" in config.h
- In GTNetS create a smybolic link to quagga in the SRC directory. EX: ln -s ~/quagga-0.99.9/ ~/gtnets/SRC/quagga

4.9.1.2 Building the Simulator

These instructions assume that you are using a recent distribution of Linux. Other *nix variants probably work also, but haven't been tested.[2]

There are two ways to build the simulator: optimized, and with debugging symbols. Optimized runs slightly faster. To build, cd into the top level GTNetS directory, and follow the README. To build everything:

```
make clean
make depend
make all
```

Making "all" builds both an optimized and debugging (with debugging symbols) binary of each test file. The optimized runs a bit faster. There are other options in the Makefile to build subsets of the scripts. For example, to start compilation over and build only the optimized version of what is in the ietf directory:

```
make clean
make depend
```

`make~~` *target-type*

[Table: Targets and Types], page 12 displays how various targets (example scripts, ietf scripts, or validation scripts) can be combined with the two options for build type (optimized

[2] GTNetS has been compiled successfully on Linux with g++-2.96, g++-3.3, Microsoft Windows with MSVC-7.0, and Sun Solaris with SUNWS-CC version??. However, we have not tested anything except gcc-3.3.3 since we did the OSPF MANET extensions. We observed a compilation problem with the newer Fedora Core 3 distribution, which uses gcc-3.4– contact us if you need a patch to the file cpqtr.h to make that compiler work.

or with debugging symbols). For example, to compile only the scripts (programs) in the validation directory, and only for optimized binaries, use `make valid-opt`.

Possible *targets*	**Possible** *types*
examples	opt
ietf	opt or dbg
valid	opt

Various ways to combined Targets and Types

Qt animation option

A default requirement is the Qt libraries from Trolltech. These are typically included in Linux distributions if you use KDE. These are used if you want to look at animations (e.g. `testwirelessgrid1.cc` in the `EXAMPLES/` directory).

An alternative to configuring and installing a Qt distribution, if you do not care about animation, is to comment out the inclusion of the QT configuration script in the toplevel and sub directory makefiles.

```
include mk/qtcheck.mk
```

This will allow building without Qt.

Note: If you are using Fedora Core 2 or something very recent, you might have a multithreaded version of Qt installed. Even though Qt works, you don't have the right header files for GTNetS compilation. In this case, we suggest the following:

- install the source package `qt-x11-free-3.3.3` in the directory `/usr/local/src`
- make a symbolic link `ln -s /usr/local/src/qt-x11-free-3.3.3 /usr/local/qt3`
- when you `make examples-dbg`, you need to pass in the QTDIR environment variable as such: `make examples-dbg QTDIR=/usr/local/qt3/`

4.9.1.3 Running the Simulator

Example simulation executables are stored in the EXAMPLES directory. Versions that are built with debugging symbols contain the suffix "-dbg", and for optimized, the suffix "-opt".

Executables that we have been using are stored in the following directories:

- `ietf/random_waypoint_manet/` and
- `ietf/link_matrix_manet`.

Try running an example script such as `ietf/random_waypoint_manet/random_waypoint_manet-opt`. It should leave a `random_waypoint_manet.stat` file, and print runtime progress to standard output.

The files in the `SRC/` directory are built as a library `libGTNetS-debug.a,so` or `libGTNetS-opt.a,so`. This library is linked to whatever test program you are using. The typical GNU debugger `gdb` can be used to debug if debugging symbols are present.

4.9.1.4 Configuring the Simulator

This chapter describes common command-line interface configuration options, and then walks through a sample program in Section [Example Test Program], page 15.

Compile-Time Options

The default Makefile in the top level directory contains the following compile-time options (the defaults enabled are shown below):

```
# To run OSPF6D, enable:
CFLAGS      += -pg -fPIC -D$(OSNAME)
# The below defines provide general support for the MANET extensions
CFLAGS      += -DETRACE -DETRACE_DEBUG
CFLAGS    += -DGTNETS_REPLACE_SCHEDULER -DREPEATABLE_MOBILITY -DETHERNET_BOEING
CFLAGS    += -DBOEING_ARGS
CFLAGS    += -DMANET
# To add TDMA USAP, add the following:
CFLAGS    += -DUSAP_ADDON
# Needed by quagga
CFLAGS    += -DHAVE_CONFIG_H -DSYSCONFDIR="/usr/local/etc/quagga"

# Flags to enable modified basic quagga behavior
CFLAGS +=  -DBUGFIX
CFLAGS +=  -DOSPF6_CONFIG
#Address Families
CFLAGS +=  -DOSPF6_AF
#Redistribution improvements
CFLAGS +=  -DREDISTRIBUTION
# To run OSPF MDR, add the following:
CFLAGS +=  -DOSPF6_MANET_MDR
CFLAGS +=  -DOSPF6_MANET_DIFF_HELLO
CFLAGS +=  -DOSPF6_DELAYED_FLOOD
CFLAGS +=  -DOSPF6_DD_OPT
#GTNETS Specific
CFLAGS +=  -DGTNETS
CFLAGS +=  -DGTNETS_ETRACE_STAT
CFLAGS +=  -DGTNETS_REPLACE_SCHEDULER
CFLAGS +=  -DOSPF6_JITTER
```

For many of these options, the above flags just enable support for the extension but don't automatically turn them on. For example, to enable Differential Hellos in a particular script, you need to do two things:

- enable `-DOSPF6_MANET_DIFF_HELLO` in the top-level Makefile
- set the ospf6_inst interface variable "diff_hellos" to "true". For an example of how to do this, see the `random_waypoint_maney.cc` program; by appending "diff_hellos" at the command line, it will enable differential hellos with the follwoing code snippet:

  ```
  ospf6_inst->SetInterfaceDiffHellos(*it, ca.diff_hellos);
  ```

Below, we discuss how these options can be selectively enabled and disabled once they are compiled in.

Configuring the Basics

The number of nodes in the MANET can set at the command line. By default, 20 nodes are used. The size of the square topology can be set at the command line. By default, the topology is 500 meters by 500 meters. The location of the nodes is defined by a random generator of x and y coordinates with the max and min values being the size of the topology. GTNetS provides various random variable types, so the distribution of node positions can be changed by chaning the random variable in `WirelessGridRectangular()`.

GTNetS also has the ability to set the shape of the grid. Shapes are found in documentation. We have only experimented with rectangular and polar.

The time at which the simulator is stopped can be configured by changing the stop time at the command line. However, if this value is set below start time then Ospfv3 routers will never fully come up. A start time delay is useful to avoid synchronization of Ospfv3 routers and to give the routers time to initialize before staistics are collected and data is sent.

Configuring Wireless Behavior

The Wireless mode is defined by the WirelessGrid constructor. See the GTNetS documentation for further configuring the wireless model. Mobility is set to random waypoint. The random waypoint pause and velocity can be set at the command line.

onfiguring OSPFv3

Ospfv3 in GTNetS can be configured in any way that the Quagga implementation can be configured. We have passed a select few of these parameters from quagga into GTNetS. The GTNetS Ospfv3 configuration functions write the configuration to a standard quagga config file. A sample quagga config file is seen at gtnets/SRC/quagga/ospf6d/ospf6d.conf.

The GTNetS per Ospfv3 interface paramters are seen in Subsection [Ospf6 per Interface Configuration], page 20 and [Ospf6 per Interface Type Configuration], page 21. The most important parameter to set for wireless interface testing is the interface type. Currently, two wireless interface types are available, point-to-multipoint and manet-designated-router. The wireless interface type can be modified at the command line by selecting 0 for ptmp and 1 for manet. Differential hellos can be enabled on the manet interface at the command line.

The value of the hello interval can be changed from 2 seconds by changing `HelloInteval`. The neighbor dead interval can be changed from 6 seconds by changing `DeadInterval`. The retransmit interval can be modified from the command line. The per interface routing cost can be modified by changing `cost`. By default, the interace cost is set to 1.

Two additional parameters were added to GTNetS. We added the ability to jitter all the Ospfv3 packets. The amount of jitter is uniformally distributed between zero and `interface_jitter`. The default jitter is 100 msec. In addition, we added a paramter to delay the sending of LSAs. This enables us to coalesce LSAs in fewer LSUs. The delay is by default 100 msec, and it can be changed by modifying `interface_flood_delay`.

Configuring Data Traffic

All scenarios that we have configured use CBR data traffic. The level of traffic is configured on a network wide basis, so the same load of data traffic is produced for any number of nodes when the packet rate and packet size are fixed. The traffic load can be configure on

the command line by setting pktrate and pktsize. The default packet rate is 10 packets per second, and the packet size is 40 bytes. All data to wireless nodes must be sent to their loopback addresses. The loopback addresses on wireless nodes has been configured to be 0.0.RouterId.0/24.

Example Test Program

This sample program, `ietf/random_waypoint_manet/random_waypoint_manet.cc`, creates a scenario a simple randomly moving MANET. The scenario consists of N mobile nodes in a L meter square grid. Nodes move according to the random waypoint model where each node selects a destination within the grid and a velocity between 0 and V m/s. The node then moves to the destination, pauses for P seconds, selects a new destination and velocity, and repeats the process. Each node is a router and has a wireless interface with an 802.11b or TDMA radio at a carrier rate of 11Mbps. Each router also has a loopback interface. Each host generates constant bit rate (CBR) traffic to every other node by sending periodic B byte packets at a packet rate of R pkt/sec. Each simulation trial is M minutes long, of which the first D minutes of data is discarded; each OSPF router is started at a random time in the first D minutes. The wireless interface can be configured to use OSPFv3 routing with either a point-to-multipoint or manet interface.

Include Files

Necessary include files are listed below.

File application-cbr.h is for CBR traffic generation. File args.h is used for processing command line input. File ipaddr.h is a class for ipv4 addresses. File mobility-random-waypoint.h provides random waypoint mobility. File node.h is the base class for nodes. File rng.h is for random variable operations. File simulator.h is the base simulator class for GTNetS. File validation.h is the method to parse command line input used by GTNetS. File wireless-grid-rectangular.h is responsible for the number of nodes in the MANET, network topology, and wireless interface configuration. File wlan.h supports a wireless link layer. These last three files were developed by Boeing. File application-ospf6d.h is the main driver of quagga ospf6d. File etrace.h provides packet level tracing. File estat.h provides global statistics.

```
// Copyright 2004, The Boeing Company
#include <iostream>
#include <stdlib.h>

#include "application-cbr.h"
#include "application-ospf6d.h"
#include "args.h"
#include "estat.h"
#include "etrace.h"
#include "ipaddr.h"
#include "mobility-random-waypoint.h"
#include "node.h"
#include "rng.h"
#include "simulator.h"
#include "validation.h"
```

```
#include "wireless-grid-rectangular.h"
#include "wlan.h"
```

Set Seeds Function

The function `set_global_seeds()` is used to set GTNets global seeds from a single seed. When GTNets global seeds are set, results are repeatable when rerunning a scenario with the same seed. The seed should be set greater than or equal to one.

```
void set_global_seeds(int seed)
{
  // turn one seed into 6 repeatable seeds in the simulator using srand()
  unsigned long seeds[6];
  srand(seed);

  seeds[0] = rand();
  seeds[1] = rand();
  seeds[2] = rand();
  seeds[3] = rand();
  seeds[4] = rand();
  seeds[5] = rand();

  Random::GlobalSeed(seeds[0],seeds[1],seeds[2],seeds[3],seeds[4],seeds[5]);
}
```

Main Function Initialization

The start of the `main()` function begins in this subsection. Here the objects in the main function are initialized. `Validation::Init()` is used to set seeding, animation, and tracing in GTNetS. The function, `set_ospf6_config_arguments` sets all the default values used for ospf and stores them in `ca`. Execute `random_waypoint_manet-opt,dbg -h` at the command line to see the default values. Default values are set in `args.cc`. The function, `set_global_seeds()`, sets the global seed by calling the function in Section [Set Seeds Function], page 16.

```
int main(int argc, char** argv)
{
  struct ospf6_config_args ca;
  Validation::Init(argc, argv);
  Simulator s;
  Interface *inter;
  IFVec_t vInterface;
  std::string tracefile, statfile;
  char hostname[20], logfilename[20];
  IPAddr_t first_rtrid;

  // Default configuration arguments are located in SRC/args.cc file
  if (!set_ospf6_config_arguments(argc, argv, &ca))
    return 0;

  set_global_seeds(seed);
```

Output File Configuration

In this subsection, the tracefile and statfile names are set. If a tag was defined at the command line, the tag string will be appended to the file name. Stat files are ended with .stat and trace files are ended with .tr.

```
// If user specifies "tag=args", this means that the suffix appended
// to the .stat or .tr filename should be the concatenation of all command
// line arguments, e.g.:
// random-waypoint-mobility-opt-num_nodes=20.seed=72.stat
if (ca.tag == "args") {
  ca.tag.erase(0,4);
  for (int q = 1; q < argc; q++) {
    if (!strcmp(argv[q],"tag=args") ||
        !strcmp(argv[q],"trace") ||
        !strcmp(argv[q],"etrace") ||
        !strcmp(argv[q],"ospfv3.log"))
      continue;
    ca.tag.append(argv[q]);
    ca.tag.append(".");
  }
  ca.tag.erase(ca.tag.length()-1, 1);
}

// Prepare output file names
tracefile.append(argv[0]);
tracefile.erase(tracefile.length()-4,4); //delete "-opt" or "-dbg" from name
statfile.append(argv[0]);
statfile.erase(statfile.length()-4,4); // delete "-opt" or "-dbg" from name

// Make sure that output file names are less than 256 char limit
if ((statfile.length() + ca.tag.length() + 7) > 256) {
  int overrun =  statfile.length() + ca.tag.length() + 7 - 256;
  ca.tag.erase(ca.tag.length()- overrun, overrun);
}

if (ca.tag.size()) {
  tracefile.append(".");
  tracefile.append(ca.tag.c_str());
}
tracefile.append(".tr");

if (ca.tag.size()) {
  statfile.append(".");
  statfile.append(ca.tag.c_str());
}
statfile.append(".stat");
```

```
cout << statfile.c_str() << endl;

// Capture command line arguments for logging purposes
for (int q = 0; q < argc; q++) {
  argstring.append(argv[q]);
  argstring.append(" ");
}
```

Topology Configuration

The topology of the network is configured in this subsection. The ipv4 address of the wireless interface on node 0 is set to 10.0.0.1. Each subsequent node's wireless interface will receive an address incremented by 1 from the first. To use Ospfv3 with ipv4, the ip address must be less than 128.0.0.0 because the first bit in the ipv4 address is used to indicate a linklocal address when converting to an ipv6 addresses with ospfv3. A square topology with the bottom left corner at location (0,0) and sides `ca.grid_length` meters is created by `WirelessGridRectangular()`. The number of nodes is a constant defined by `ca.nNodes`. The nodes are uniformly distributed on the square grid. The wireless MAC layer is also specified in this function. Random waypoint mobility is defined by fuction, `AddMobility(RandomWaypoint())` if the velocity is greater that zero. The nodes travel to a random location with the topology at a speed uniformally distributed between 0 and `ca.velocity` m/s. After reaching the destination, the node pauses for `ca.pause_time` seconds and then moves again.

```
// ipv4 addresses must be less than 128.0.0.0 for ospf6 because we used
// the first bit in the ipv4 address to indicate a linklocal address
IPAddr_t firstIP = IPAddr("10.0.0.1");

Location l(0,0); // lower left corner of rectangular grid

// Randomize the distribution of nodes uniformly on square grid
WirelessGridRectangular g(ca.mac_protocol,
                          l,
                          Constant(ca.nNodes), //number of nodes
                          Uniform(0.0, ca.grid_length),
                          firstIP);

// Mobility:  Pause time is second argument, velocity is third argument
if (ca.velocity > 0)
  g.AddMobility(RandomWaypoint(g, Constant(ca.pause_time),
                               Uniform(0,ca.velocity)));
```

Tracing Configuration

Extended tracing is configured in this subsection. Exteneded tracing was developed by Boeing and more specifics can be found in Section [Tracing], page 23. By default, tracing is disabled. If tracing is enbabled at the command line then the tracefile is opened. Extended tracing is also disabled by default. If it is enabled at the command line then fuction,

SetExtended(), enables extended tracing. Extended tracing adds more detail to the basic packet level tracing.

The ability to turn off tracing of a specified layer is provided by function, LayerOff(). By default, all layers are enabled. Finally, additional events can be written to the tracefile. Events are used to display information or statistics not seen in packet level tracing. The function, SetTraceEventLevel(), sets the verbosity of events to display. A value of zero means no event tracing.

```
// Per-packet tracing configuration
if (ca.trace) {
  ETrace* egs = ETrace::Instance();
  egs->Open(tracefile.c_str());
  if (ca.etrace)
    egs->SetExtended(true);
  egs->LayerOff(1); //don't etrace this layer
  egs->LayerOff(2); //don't etrace this layer
  egs->SetTraceEventLevel(2);
}
```

Statistics Configuration

The collection of statistics is initialized in this subsection. By default, statistics are disabled. If statistics are enbaled at the command line then statistics begin to be collected at start_time seconds, and the results are output to the statistics file.

```
// Statistics collection
EStat* estat = EStat::Instance();
estat->CollectEStats(statfile.c_str(), ca.start_time);
estat->LayerOff(1);
estat->LayerOff(2);
```

Ospf6 and CBR Intitialization

The intitilization of Ospfv3 routing and CBR traffic is performed in this subsection. The OSPF6DApplication must be created to run ospf6. The collection of Ospfv3 stats is configured by CollectStats(). The rate of monitoring the physical layer neighbors is set by function, LogNeighbors. These physical neighbor statistics are printed in the statistics file. The first router id indicates the id assigned to the first node to have an OSPF6DInstance. The rate of the cbr application is set so the whole MANET network generates pktrate packets per second, each of pktsize bytes. To avoid synchronization, the cbr traffic is started at a random time within the packet send interval.

```
// OSPF6 initialization
OSPF6DApplication ospf6d;
ospf6d.CollectStats(statfile.c_str(), argstring.c_str(), ca.start_time);
ospf6d.LogNeighbors(0.5);  // Log physical layer neighbors every 500 ms

OSPF6DInstance *ospf6_inst;
Uniform u(0,ca.start_time);
first_rtrid = IPAddr("0.0.0.1");
```

```
// CBR traffic initialization
IPAddr remoteIP;
char buf[32];
Rate_t rate_bps = ((Rate_t)ca.pktrate * ca.pktsize * 8) /
                  (ca.nNodes*(ca.nNodes-1));
// start traffic generators at a random time
Uniform cbrstart(ca.start_time-(double)(ca.pktsize*8)/rate_bps,ca.start_time);
```

Per Node Configuration

The start of a `for` loop that loops over all nodes is begun in this subsection. Each nodes radio range is set to the identical value, `radio_range`.

```
for (Count_t i = 0; i < g.Size(); ++i)
  {
    Node* n = g.GetNode(i);
    n->SetRadioRange(ca.radio_range);
```

Ospf6 per Node Configuration

In this subsection, each node is given an ospf6 instance. Also, the host name, log file name, and id of each router is set. The MinLSInterval and MinLSArrival are set here. The MinLSInterval is the minimum time between originating the same LSA. The MinLSArrival is the minimum time in which a router will accept the same LSA.

```
    // OSPF6 Router Configuration
    ospf6_inst = ospf6d.AddNode(n);

    sprintf(hostname, "ospf6_%d", n->Id()+1);
    sprintf(logfilename, "log_ospf6_%d.log", n->Id()+1);

    ospf6_inst->SetHostName(hostname);
    if (ca.ospfv3log)
      ospf6_inst->SetLogFileName(logfilename);
    ospf6_inst->SetRouterId(n->Id()+first_rtrid);
    ospf6_inst->SetMinLSInterval(ca.MinLSInterval);
    ospf6_inst->SetMinLSArrival(ca.MinLSArrival);
```

Ospf6 per Interface Configuration

In this subsection, the interfaces per node are configured with ospf6 interface specific parameters. The time between periodic hello sends on an interface is set with `SetInterfaceHelloInterval()`. The time interval when a neighbor dies due to inactivity of hello reception is set by `SetInterfaceDeadInterval()`. The interval to wait before retransmitting an LSA is set by `SetInterfaceRetransmitInterval()`. The time that a router waits to colesce LSAcks is set by `SetInterfaceAckInterval`. The ospf6 area this interface is on is set with `SetInterfaceArea()`. The ospf cost of sending on this interface is set with `SetInterfaceCost()`. The max jitter that is added to the sends of ospf6 packets is set with `SetInterfaceJitter()`. The packet jitter can be thought of as queing delay of `interface_jitter` msec before it is sending. Finally, the time to wait to accumulate LSAs before flooding is set with `SetInterfaceFloodDelay()`.

```
  vInterface = n->Interfaces();
  for (IFVec_t::iterator it=vInterface.begin(); it!=vInterface.end(); ++it)
  {
    ospf6_inst->SetInterfaceHelloInterval(*it, ca.HelloInterval);
    ospf6_inst->SetInterfaceDeadInterval(*it, ca.DeadInterval);
    ospf6_inst->SetInterfaceRetransmitInterval(*it, ca.RxmtInterval);
    ospf6_inst->SetInterfaceAckInterval(*it, ca.AckInterval);
    ospf6_inst->SetInterfaceArea(*it, 0);
    ospf6_inst->SetInterfaceCost(*it, ca.cost);
    // too much jitter can cause stale ospf6 state to be sent
    ospf6_inst->SetInterfaceJitter(*it, ca.interface_jitter); //msec
    ospf6_inst->SetInterfaceFloodDelay(*it, ca.interface_flood_delay); //msec
```

Ospf6 per Interface Type Configuration

The type of interface is set based on the characteristics of the interface. The interface type can be either point-to-multipoint, or MDR . The time which a Backup MANET Designated Router waits before flooding an LSA is set by `SetInterfaceBackupWaitInterval()`. MDRs have options for the level of adjacency connectivity and and the fullness of the router LSAs. The parameter alpha influences the probability of a packet being received within the radio range. When alpha = 1, all packets sent within radio range are received. The loopback interface is set to Ospfv3 type loopback. Then cbr traffic is sent from this node to the loopback address of every other router. The cbr traffic starts just prior to `start_time` seconds and ends at `stop_time - 5` seconds. The start time is jittered so synchronization among routers is avoided. Finally, all other interface are set to Ospfv3 type broadcast. In this scenario there are no other interfaces, so this event will not occur.

```
    if ((*it)->GetL2Proto()->IsWireless())
    {
      if(ca.w_int_type == 0)
        ospf6_inst->SetInterfaceType(*it, "point-to-multipoint");
      else if (ca.w_int_type == 1)
      {
        ospf6_inst->SetInterfaceType(*it, "manet-designated-router");
        ospf6_inst->SetInterfaceDiffHellos(*it, ca.diff_hellos);
        ospf6_inst->SetInterfaceBackupWaitInterval(*it,ca.BackupWaitInterval);
        ospf6_inst->SetInterfaceTwoHopRefresh(*it, ca.TwoHopRefresh);
        ospf6_inst->SetInterfaceHelloRepeatCount(*it, ca.HelloRepeatCount);

        if (ca.AdjConnectivity == 0)
          ospf6_inst->SetInterfaceAdjConnectivity(*it, "fully");
        else if (ca.AdjConnectivity == 1)
          ospf6_inst->SetInterfaceAdjConnectivity(*it, "uniconnected");
        else
          ospf6_inst->SetInterfaceAdjConnectivity(*it, "biconnected");

        if (ca.LSAFullness == 0)
          ospf6_inst->SetInterfaceLSAFullness(*it, "minlsa");
```

```
        else if (ca.LSAFullness == 1)
          ospf6_inst->SetInterfaceLSAFullness(*it, "mincostlsa");
        else if (ca.LSAFullness == 2)
          ospf6_inst->SetInterfaceLSAFullness(*it, "mincost2lsa");
        else if (ca.LSAFullness == 3)
          ospf6_inst->SetInterfaceLSAFullness(*it, "mdrfulllsa");
        else
          ospf6_inst->SetInterfaceLSAFullness(*it, "fulllsa");
      }
      if (i==0)
      {
        WirelessLink *l = (WirelessLink *) (*it)->GetLink();
        l->SetAlpha(ca.alpha);
      }
    }
    else if (strcmp((*it)->GetName(), "lo") == 0)
    {
      ospf6_inst->SetInterfaceType(*it, "loopback");
      if (rate_bps > 0 && ca.pktsize > 0)
      {
        // CBR traffic from this router to all other router's loopback addr
        for (Count_t j = 0; j < g.Size(); ++j)
        {
          if (i == j)
            continue;
          sprintf(buf, "0.0.%d.0", j+1); //loopback addr of routers
          remoteIP = IPAddr(buf);
          CBRApplication* cbr = (CBRApplication*) n->AddApplication(
                  CBRApplication(remoteIP,1000,NO_PORT,rate_bps,ca.pktsize));
          cbr->Start(cbrstart.Value());
          cbr->Stop(ca.stop_time - 5);
        }
      }
    }
    else
      ospf6_inst->SetInterfaceType(*it, "broadcast");
  }
```

Setting Start and Stop Times

The Ospfv3 routers are started at a random time between 0 and `start_time` seconds. This prevents Ospfv3 packets from being synchronized. The simulation starts at time 0 and ends at time `stop_time` seconds. The progress is displayed at the command line every 40 seconds. Simulations are commenced by function `Run()`.

```
    ospf6_inst->Start(u.Value());  // random start time of each ospf6 router
  }
  s.StopAt(ca.stop_time);
```

```
  s.Progress(40);
  s.Run();
}
```

Example Test Programs

- **ietf/initial/basic.cc**. This program, described above in the example, runs a basic wireless network with Ospfv3 routing.
- **ietf/random_waypoint_manet/random_waypoint_manet.cc**. This program is similar to the basic.cc program, but it contains many new updates. It includes support for using a TDMA MAC based on the Unifying Slot Assignment Protocol (USAP).
- **validation/sicds/basic/basic.cc** Performs validation of the MDR algorithm.
- **validation/sicds/pushback1/lsu_loss.cc** Performs validation of an LSA being delayed after lossing the initial flood.
- **validation/sicds/pushback2/lsack_loss.cc** Preforms validation of losing an LSAck and then having the backupwait timer expire.
- **validation/sicds/pushback3/lsu_loss_pushback_lsu_loss.cc** Performs validation of an LSA being lost, having a node send out a backupwait LSA and loose it, and then finally having another node send a backupwait LSA.
- **ietf/link_matrix_manet/link_matrix_manet.cc** Uses an extension to ethernet (known as link matrix) that allows the bringing up and down of specifed links at particular times.
- More scripts are included in the ietf and validation subdirectories. Please contribute any interesting scripts.

4.9.1.5 Simulation Output

There are two main ways to look at simulation output:

1. Explicit `"tcpdump"`-like tracing of packets to an output file that can be post-processed later
2. Compilation of summary statistics that can be printed out at the end of the simulation

GTNetS had a basic tracing capability, but we have extended it further to print out more detailed logging. This can be used to examine trace output for correct or errored behavior (much like a tcpdump output file), or can be fed into post-processing scripts to gather statistics. The file would typically be called `testname.tr`.

However, for large simulations, both the file size and the time to parse the traces can become very large, so we also provide a capability to dump summary statistics into a file at the end of the simulation run (into a file typically called `testname.stat`).

Tracing

How to Enable Tracing

GTNetS has a built-in tracing facility based on the *Trace* object. We cloned that object class to create the Extended Trace, or *ETrace* object. Either, neither, or both Trace and ETrace can be enabled on a particular script.

```
ETrace* egs = ETrace::Instance();
egs->Open("<file name>");
```

Figure 4.2: Sample code to control tracing

Figure 4.2 illustrates the basic C++ commands to add to your testfile to enable tracing. This snippet creates an ETrace object and associates it with a file. The filename can be anything you want. The semantics are to overwrite (not append) any existing file that may be at that location.

```
// Per-packet tracing configuration
if (ca.trace) {
  ETrace* egs = ETrace::Instance();
  egs->Open(tracefile.c_str());
  if (ca.etrace)
    egs->SetExtended(true);
  egs->LayerOff(1); //don't trace this layer
  egs->LayerOff(2); //don't trace this layer
  egs->TraceTopo(ca.tracetopo);  // trace topology every tracetopo seconds
  egs->SetTraceEventLevel(2);
}
```

Figure 4.3: Sample code to control tracing

Figure 4.3 shows how to control options on what is and isn't traced. By default, all protocol layers on all nodes are traced. There are a number of options to control this, however:

- SetExtended(true|false). This will enable extended tracing, which basically means to trace beyond the IP header and into the OSPF packet body. Default is false.
- SetEventTraceLevel(0|1|2). This controls the level of debugging output for protocol events (non-packet events) put into the tracefile (the higher the number, the more events that are logged). Level 1 adds the following events:
  - Count of OSPF packet drops and OTHER packet drops upon end of simulation
  - Information about the number of LSAs out of sync during database exchange
  - when a unicast LSU is sent
  - statistics on the number of neighbors of a node whenever a neighbor change event occurs

  Level 2 adds the following events:
  - Prints out when the router started operation
  - Dumps relay selector lists
  - More information traced regarding neighbors changing state
- LayerOff(n). This indicates that tracing at a particular protocol layer is disabled.
- NodeOff(n). This indicates that tracing at a particular node is disabled.
- TraceTopo(n). Enable the tracing of topology information every n seconds.

Sample Tracefile

```
s 3599.97947 N12 -int 1 -pkt_id 1329237  -l3proto IPv4 -src 10.0.0.13 -dst 0.0.1
4.0 -len 92 -ttl 64 -proto UDP
r 3599.97992 N8 -int 1 -pkt_id 1321800  -l3proto IPv4 -src 138.0.0.14 -dst 138.0
.0.9 -len 56 -ttl 0 -proto OSPFv3 -type LSAck
r 3599.98195 N13 -int 1 -pkt_id 1329237  -l3proto IPv4 -src 10.0.0.13 -dst 0.0.1
4.0 -len 92 -ttl 63 -proto UDP
```

Figure 4.4: Sample tracefile excerpt, with sample 80-column wrap

Figure 4.4 illustrates a sample tracefile excerpt from the testwirelessospf.cc script. In this example, extended tracing is off, so OSPF protocol parsing beyond the packet type is disabled. Each traceline starts with a single character event code. The following event codes are defined.

- **s**: Packet send event
- **r**: Packet receive event
- **f**: Packet forward event
- **d**: Packet drop event
- **c**: Packet collision event
- **e**: Protocol event (non-packet event)
- **n**: Node location and DR level(non-packet event)
- **a**: Adjacency involving a DR_OTHER router (non-packet event)
- **b**: Adjacency between (B)MDRs (non-packet event)

The next field is the node number, followed by a number of key-value pairs:

- **int**: Interface number that packet was seen on
- **pkt_id**: Packet unique id
- **l3proto**: Layer 3 protocol in use
- **...**: Other fields are probably self explanatory

These trace lines will contain more information (expanded) if extended tracing is enabled.

Animating MDR Topology

The topology of the OSPF-MDR interface can be displayed graphically in the form of an animated "gif" file. The animated gif is created by enabling topology tracing and by post processing the tracing to form the "gif" file. The appropriate trace lines are output by calling the function `ETrace::TraceTopo()`. If one is using `random_waypoint_manet.cc` then either `tracetopo=<>` or `tracetopoonly=<>` are called from the command line. The input parameter is the sampling rate in seconds. The final step is to run `gtnets/bin/tracetogif.pl` on the trace file. The usage of `tracetogif.pl` can be determined by running it with no input parameters. An example parameterization is shown here: `./tracetogif.pl -stop 2000 -delay 200 trace.tr`. The following would create an animated gif of the topology data between 1800 and 2000 secs, and it would shows snapshots of the topology every 2 seconds.

Statistics

To be able to turn off tracing for larger scenarios, we added some counter-based statistics gathering. At present, this is turned on by adding the following lines to a particular script:

```
//Extended Statistics Configuration
EStat* estat = EStat::Instance();
estat->CollectEStats(statfilename, delay);
estat->LayerOff(1);
estat->LayerOff(2);
//Ospfv3 Statitistics Configuration
OSPF6DApplication ospf6d;
ospf6d.CollectStats(statfilename, delay);
```

Figure 4.5: Sample code to control tracing

The output of the statistics will be deposited in the filename specified, with a `.stat` suffix.

Currently, EStat only generates statitics for UDP traffic, and all Ospfv3 stats are generated by `class OSPF6DApplication`. We did not use GTNetS built-in Statistics class because that code is not compatible with quagga.

To add additional statistics to Ospfv3, the statistic name must be added to the enumeration `ospf6_stats` in `ospf6_top.h`. Each of these names are elements in the statistics array found in struct ospf6. Next, the appropriate place to count the statistic must be found, and the statisitic is counted by incrementing the element in the array statistic. Statistics in struct ospf6 are per node, so in the `application-ospf6d.cc` function `OSPF6DApplication::ExtractStats()` the node stats are summed to create global network stats. The final step in creating a new stat is to add the printout of the value in the `application-ospf6d.cc` function `OSPF6DApplication::PrintStats()`.

Logging

Logging build into quagga Ospfv3 has been integrated with GTNetS, so the same logging produced by quagga can be produced in GTNetS. Quagga logging creates a seperate log for each router. To use the logging in GTNetS, the log file for each node must be named by calling the function `OSPF6Instance::SetLogFileName()`. When this function is called, logging will be generated for LSAs, Hellos, LSUs, LSAcks, LSRs, and Database Description packets. In addition, logging will be generated for neighbor and interface changes. The type of logging generated can be changed in `application-ospf6d.cc` function `OSPF6DInstance::CreateConfFile()`.

%% Note: Use two labels below for cross-referencing in two different reports

4.9.1.6 Validation

This section explains the validation of the OSPF modifications. Our validation scripts serve two purposes:

1. Detection of code modifications that inadvertently break the correct behavior of a protocol.
2. Isolation of particular protocol elements to ensure that they are working correctly.

In this section, we first describe the general script that can be used to detect changes to the behavior of protocols. We then follow with some examples of how we validated the specific OSPF modifications.

The "validate" Script

There is a script called `validate` in the top level directory. This script finds all executables in any subdirectories in the `validation/` directory and executes them. Each directory in `validation` contains the validation driver, an expected trace file, and expected stat file. The `validate` script compares the expected output with the current output. Any differences are output to a diff file, and they are flagged as a `FAILURE` at the command line. To run `validate` the environment variable `GTNETS_HOME` must be set to GTNetS base directory. This is very similar to how the ns-2 simulator validation works.

The idea behind this is that if there are changes to the behavior of the protocols covered by example scripts, then the validation output will start to diverge from the known good output. The researcher can then run the `validate` script to find out what scripts are broken, examine the `diff` file produced, and determine whether the source code is broken or whether the expected known good output needs to be updated (for `validate` to pass in the future).

Basic Validation

Let's first look at a very simple network– four OSPFv3 nodes fully meshed on a broadcast-based subnet (e.g., Ethernet) but running the multicast-capable point-to-multipoint (i.e., non-MANET) interface type. We will use the "Matrix" channel, a prototypical script for which can be found in `ietf/link_matrix_manet/link_matrix_manet.cc`. "Matrix" channel is intended to allow the packet delivery ratio between any pairwise set of nodes to be altered, but the default is full mesh connectivity.

This executable is built if the `ietf/` directory was enabled; this can be done with, for example, `make ietf-opt`, `make ietf-dbg`, etc. from the top level directory (see [Building the Simulator], page 11).

Let's assume that the "optimized" version of the binary has been built. Type `./link_matrix_manet-opt -help` to see the list of options and default values.

We will run with most of the defaults. However, let's enable all output tracing and logging, and set the number of nodes from its default of 20 to just 4, and turn off the minimal user data traffic by setting the packet rate to zero: `./link_matrix_manet-opt num_nodes=4 pktrate=0 trace etrace ospfv3log`

Next, type `more link_matrix_manet-opt.stat` to look at the statistics file generated. This simulation ran for 3600 seconds, and statistics were collected over the last 1800 seconds (total of 1800 seconds simulation time). There is not much of interest to see; there were 720 Hello messages (one Hello per node every 10 seconds for four nodes and 1800 seconds yields 720 Hellos). There were 32 LSU packets sent, and (further down in the statistics output) 48 "relay_flooded_LSAs". We will next sanity check these results by looking at the output trace file.

The detailed per-packet and per-event tracing can be found in the `link_matrix_manet-opt.tr` file. Even for this small quiet network, there are over 4500 lines of trace output generated. Grep for "Started" and you should see something like:

```
e 31.41292 N1 Ospf6 Router Started
e 57.94125 N2 Ospf6 Router Started
e 1274.60552 N0 Ospf6 Router Started
e 1634.38450 N3 Ospf6 Router Started
```

illustrating the starting times of each router for nodes 0 through 3 (the event code "e" signifies that this is an implementation event and not a packet).

For convenience, let's remove all Hello messages from the trace output: `grep -v HELLO link_matrix_manet-opt.tr > output.tr`. Now, let's look at the resulting `output.tr` file. We know that the first router started up at time 31.4, so at around time 1831.4, there should be some re-originated LSAs. This can be seen in the `output.tr` file:

```
e 1831.41492 N1 Re-orig LSA Link -id 0.0.0.1 -advrt 0.0.0.2 -age 0 -seq 21474836
50 -len 64
e 1831.41492 N1 Schedule Flood LSA Link -id 0.0.0.1 -advrt 0.0.0.2 -age 0 -seq 2
147483650 -len 64 from (null)
e 1831.41492 N1 Re-orig LSA Intra-Prefix -id 0.0.0.0 -advrt 0.0.0.2 -age 0 -seq2
147483650 -len 72
e 1831.41492 N1 Schedule Flood LSA Intra-Prefix -id 0.0.0.0 -advrt 0.0.0.2 -age0
 -seq 2147483650 -len 72 from (null)
s 1831.53392 N1 -int 1 -pkt_id 519  -l3proto IPv4 -src 138.0.0.2 -dst 224.0.0.5-
len 176 -ttl 1 -proto OSPFv3 -type LSU -len 156 -rid 0.0.0.2 -aid 0.0.0.0 -Num2
-Lsa Link -id 0.0.0.1 -advrt 0.0.0.2 -age 1 -seq 2147483650 -len 64 -Lsa Intra-P
refix -id 0.0.0.0 -advrt 0.0.0.2 -age 1 -seq 2147483650 -len 72
```

The two LSAs (Intra-Prefix and Link) are generated, and sent in a single LSU at time 1831.53392. Scrolling down in the tracefile, we can see that nodes N0, N3, and N4 also receive and reflood this LSU. This accounts for a total of 4 LSU transmissions, and there are 4 nodes that will perform this reorigination, so this accounts for 16 of the LSUs, and 32 of the LSAs in the above cited statistics counts. Now, consider that the Router LSAs are expiring at a different time, since they were all changed after the fourth router originally came up. Therefore, each node will generate an additional LSU, causing four LSU transmissions per node (an additional 16 LSU transmissions and LSA floods) starting around time 3435. This yields our counts of 32 LSU transmissions and 48 LSA floods.

Note that if there were a quagga implementation, output would be stored in output logs; these should be now available in the `log` directory.

Radio Range

The directory `validation/radio_range` contains some scripts used to determine the radio range of the selected 802.11-based radios, based on varying the distance between nodes. These scripts are not well-documented at this time, but were used to check the radio behavior for different parameterization.

4.9.2 License and Contributing Code

This software builds on two projects (GTNetS and quagga) that are under GNU General Public License (GPL). Our modifications are also under GPL. Therefore, please consider this simulator as GPL'ed software.

We would like to encourage people to send in bug fixes, extensions, or to make their simulation scripts available (that produce their simulation results). By default, we will not

include any contributed code, bug fixes, or patches unless you specify that you want to include said code in our future releases of this simulator (also under GPL).