

# OLSRD HOWTO, Version 1.0

Written by Ron Lee and Joe Macker

Introduction

Installation

Configuration & Execution

Appendix: Topology Emulation via MAC layer configuration

## 1 Introduction

These technical notes and operational instructions relate to the INRIA Optimized Link State Routing (OLSR) mobile ad hoc routing daemon that has been modified by the US Naval Research Laboratory (NRL) for use in its research efforts. The codebase was based on the Version 3 of INRIA's Optimized Link State Routing (OLSR) protocol draft for the IETF. This present HOWTO document describes the installation and configuration of this NRL-modified OLSRD for RedHat Linux. Currently NRL-OLSRD supports only IPv4 addressing using UDP port 698. At present, the modified source code and related project information is available at <http://pf.itd.nrl.navy.mil/>

## 2 Installation

### 2.1 Building olsrd

Unarchive the nolsrd\_X.XXX.tgz file using the "tar" command:  
`tar -xzf nolsrd_X.XXX.tgz`

In the new directory execute the "make all" command, this creates olsrd and some additional tools.

### 2.2 Installing the olsrd program file

To install olsrd in the system directories, become a "root" user (either via the "su" command or logging-in as a "root" user. As a root user execute a "make install" command. This will install olsrd as "/usr/sbin/olsrd".

## 3 Configuration

### 3.1 Linux System Configuration:

At present this source code has only been ported and tested on Linux distributions. The following provides some additional notes related to kernel parameters, etc that need to be configured. In order to use your Linux computers as Mobile Ad Hoc (manet) nodes, it must be configured as a router and we recommend disabling ICMP redirects. To do this, we modify the `/etc/sysctl.conf` file as below:

```
# Enables packet forwarding
net.ipv4.ip_forward = 1

# Disable redirects
net.ipv4.conf.eth0.send_redirects = 0
net.ipv4.conf.eth0.secure_redirects = 0
net.ipv4.conf.eth0.accept_redirects = 0
net.ipv4.conf.lo.send_redirects = 0
net.ipv4.conf.lo.secure_redirects = 0
net.ipv4.conf.lo.accept_redirects = 0
net.ipv4.conf.default.send_redirects = 0
net.ipv4.conf.default.secure_redirects = 0
net.ipv4.conf.default.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
net.ipv4.conf.all.secure_redirects = 0
net.ipv4.conf.all.accept_redirects = 0
```

The last 3 lines above should be sufficient for disabling all interfaces.

Execute `"/sbin/sysctl -p"` command to apply the changes immediately; otherwise, a system reboot will cause the new variable entries to take effect.

### 3.2 OLSRD Command Line Options

The following provides some detailed instructions on running `olsrd` and related parameters that can be modified and recommended settings. Always the source code README files for the latest information.

The command line options for `nolsrd` release 1.0a7 are as follows:

```
olsrd [ -s ] [ -q ] [ -d <debug_level> ] [ -t ] [ -hint <hello interval value (secs)> ] [ -hjit  
<hello jitter value (secs)> ] [ -tcint <tc interval value (secs)> ] [ -tcjit <tc jitter value  
(secs)> ] [ -T <Polling Rate (secs)> ] [ -hmult <neighbor holdtime multiplier (int)> ]  
[tmult <topology holdtime multiplier (int)>] [ -i 'interface file' ] [ -g <default gw  
address > ] [ -tos value ] [ 'trace file' ]
```

### 3.2.1 Foreground & Background Process Operation

OLSRD can be operated in either the foreground or the background (i.e., daemon). By running with a "-d" (DEBUG) option OLSRD will run as a foreground process while displaying various debug information to the terminal. Without the "-d" option, OLSRD will execute as a background or daemon process.

### 3.2.2 Debug Levels

The OLSRD application has several different debug levels when used with the "-d" option. Debug level 1 prints neighbor table information as well as received hello interval status to stdout:

- 0 = pending
- 1 = asymmetric
- 2 = symmetric
- 3 = mpr

Debug level 2 prints the neighbor table and the topology table information.

### 3.2.3 OLSR Protocol Parameter Settings (jitter, intervals, etc)

In early testing of OLSR, it became apparent that interval timers and jittering of control packets are important parameters to experiment with to avoid the possibility of increased collisions due to protocol synchronization, this code adds the ability to modify those parameter values at run time. Here is a brief explanation of more salient parameters and their potential impact. The INRIA code also used a polled event timer so in order to make other parameters work properly the polling routine must be set to a reasonably small interval. An improved event handler would have been recommended, but at this time new version of olsrd are under development.

Hello Interval: the time (secs) between successive transmission of hello packets.

Hello Jitter: a jitter interval to randomize the hello transmission time, this helps further avoid protocol synchronization issues and packet collision.

TC Interval: the time (secs) between successive transmission of TC messages (a TC minimum is also included in the code which I believe is set to 1 sec).

TC Jitter: a jitter interval to randomize the TC transmission time, this helps further avoid protocol synchronization issues and packet collision.

Polling Rate: an interval for the select routine used in controlling the event handler (not elegant code, but by setting this short one can improve the jitter operation,etc).

Holdtime multipliers: multiplication factor for the hello and tc intervals that determines the soft state timeouts for table entries (should at least be  $\geq 3$ ). There is a tradeoff here between deleting stale information and deleting valid entries too early due to missed control packets.

Example recommended settings for a typical 802.11 experiment are the following:

Polling interval = 0.05 sec  
Hello interval = 0.5 sec  
Hello jitter = 0.1 sec  
TC interval = 2.0 sec  
TC jitter = 0.5  
Neighbor timeout multiplier = 8  
TC timeout multiplier = 10  
Type of service setting = 16

These are the present default settings.

If more than one interface is available the interface for running olsrd must be specified with the interface option on the command line.

### 3.2.4 Default Gateway Support

Default gateway address: (this function and command line option was changed from the original INRIA code).

Using this option allows a static entry to a default gateway node (potentially multiple hops away) to be entered at all manet nodes. Typically this gateway node will be dual-homed, i.e., two network interfaces. One network interface is the wireless manet network interface and the other MAY be a wired network interface connected to an intranet/internet. To use this default gateway address one must configure the non-gateway manet nodes to point to this dual-homed node. When the non-gateway manet node's olsrd sees the gateway manet node's address in the topology a default routing entry to the gateway manet node will be added dynamically with the appropriate next hop information.

## 4 Appendix A: Tabletop testing configurations

Prior to field testing or to allow examining particular controlled failure scenarios it is often useful to have a method to control topology and routing neighborhoods

more directly outside of a simulation environment. NRL has used MAC filtering schemes to enable simplified tabletop testing of simple topologies and link failures in testing olsrd routing. While this does not capture particular MAC layer effects and radio propagation characteristics, it does allow for some structured preliminary testing under repeatable conditions. We include this information for those who might be interested in performing similar experiments. Other more sophisticated tools such as NISTnet and dummynet may also be used to add bandwidth limitations and errors on links, etc.

#### 4.1 Topology emulation using the Linux iptables firewall.

Without the ability to control physical layer parameters such as power levels it can be difficult to consistently and easily control the design of manet routing topologies in the laboratory. As a simple approach to overcome this barrier, NRL used the builtin Linux IP packet filter administration tool called iptables to create various topologies. This tool can be used with either a wireless or wired Ethernet interface. One can create asymmetric as well as symmetric links.

To do this we filter packets based on Ethernet MAC address. In order to simulate a link outage we insert an iptables entry to drop incoming packets from nodes we do not want to have a link to.

```
iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:XX -j DROP
```

Since iptables allows for interactive entries, and we can thus decide to accept packets from certain MAC addresses as well:

```
iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:XX -j ACCEPT
```

A useful example of iptables emulated manet routing would be 4 nodes arranged logically in a straight line.

Routing Topology:

A <---> B <---> C <---> D

Ethernet (MAC) Addresses:

Node A = XX:XX:XX:XX:XX:0A

Node B = XX:XX:XX:XX:XX:0B

Node C = XX:XX:XX:XX:XX:0C

Node D = XX:XX:XX:XX:XX:0D

Commands issued on each node:

Node A: iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0C -j  
DROP  
iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0D -j  
DROP

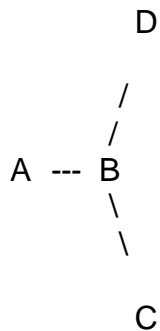
Node B: iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0D -j  
DROP

Node C: iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0A -j  
DROP

Node D: iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0A -j  
DROP  
iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0B -j  
DROP

Node A will filter out (i.e., reject or drop) all Ethernet (and thus, IP packets) coming from Node C and Node D. Node B will drop packets from Node D. Node C will reject packets from Node A. Node D will drop packets from Node A and B. Thus, this forces Node A to route data packets to Node D via nodes Node B and Node C.

With iptables we can change the topology while OLSRD continues to execute; thereby, simulating nodal movement. Unfortunately, sometimes we need to make appropriate changes on more than one node. A simplistic example would be for Node D to "move in range" of Node B.



We issue the appropriate entries to both Node B and Node D:

Node B: `iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0D -j ACCEPT`

Node D: `iptables -A INPUT -m mac --mac-source XX:XX:XX:XX:XX:0B -j ACCEPT`

To ease the burden of issue iptables commands, one is advised to use a shell scripting language to create the desired topology.