EMANE
User Manual

0.7.3

labs.cengen.com/emane

EMANE
User Manual
0.7.3

Steven M. Galgano
Kaushik B. Patel
Eric Schreiber

DRS CenGen, LLC
1120 Route 22 East
Building 1, Suite 7
Bridgewater, NJ 08807

Rev. 4

EMANE is being developed by the Naval Research Laboratory (NRL) under the OSD Network Communication Capability Program (NCCP) and in cooperation with the Army Research Laboratory (ARL) High Performance Computing Mobile Network Modeling Institute (HPC MNMI) effort.

Copyright© 2012 - DRS CenGen, LLC, Bridgewater, New Jersey

This work is licensed under the Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

The copyright holder hereby grants to the U.S. Government a copyright license to use this work that is of the same scope as the rights set forth in the definition of "unlimited rights" found in DFARS 252.227-7014(a)(15)(June 1995).

Contents

1 Introduction **1**
1.1 Overview . . . 2
1.2 EMANE Demo Virtual Machine . . . 2
1.3 EMANE Deployment Diagrams . . . 4

I Emulation Infrastructure **5**

2 Infrastructure Basics **7**
2.1 NEM Platform Server . . . 7
2.1.1 Centralized Deployment Example . . . 9
2.1.2 Distributed Deployment Example . . . 10
2.1.3 NEM Platform Server Configuration Parameters . . . 11
2.1.3.1 `otamanagergroup` . . . 11
2.1.3.2 `otamanagerdevice` . . . 12
2.1.3.3 `otamanagerchannelenable` . . . 12
2.1.3.4 `eventservicegroup` . . . 12
2.1.3.5 `eventservicedevice` . . . 13
2.1.3.6 `debugport` . . . 13
2.1.3.7 `debugportenable` . . . 13
2.1.4 Shared Configuration Parameters . . . 13
2.1.4.1 `platformendpoint` . . . 14
2.1.4.2 `transportendpoint` . . . 14
2.2 Transport Daemon . . . 15
2.2.1 Centralized Deployment Example . . . 16
2.2.2 Distributed Deployment Example . . . 17
2.3 Demonstrations . . . 18
2.3.1 Demonstration 1 . . . 18
2.3.1.1 Demonstration Procedure . . . 19
2.3.1.2 Concept Review . . . 20
2.3.2 Demonstration 2 . . . 21
2.3.2.1 Demonstration Procedure . . . 21
2.3.2.2 Concept Review . . . 22

3 Network Emulation Modules **25**
3.1 Defining an NEM . . . 26
3.2 Physical Layer . . . 27
3.2.1 Supporting Heterogeneous Waveforms . . . 29
3.3 Medium Access Control Layer . . . 29
3.4 Shim Layer . . . 30

4 Events **33**

4.1 Event Service . . . 33
4.1.1 Event Service Configuration Parameters . . . 35
4.1.1.1 `eventservicegroup` . . . 35
4.1.1.2 `eventservicedevice` . . . 35
4.2 Event Daemon . . . 36
4.2.1 Event Daemon Configuration Parameters . . . 37
4.2.1.1 `eventservicegroup` . . . 37
4.2.1.2 `eventservicedevice` . . . 37
4.3 Event Types . . . 38
4.3.1 Pathloss Event . . . 38
4.3.2 Location Event . . . 38
4.3.3 Comm Effect Event . . . 39
4.3.4 Antenna Direction Event . . . 39
4.4 Demonstrations . . . 39
4.4.1 Demonstration 3 . . . 39
4.4.1.1 Demonstration Procedure . . . 39
4.4.1.2 Concept Review . . . 41

5 XML Configuration **43**
5.1 Layered Configuration . . . 43
5.2 Automatic XML Generation . . . 45
5.3 Transport Grouping . . . 46
5.4 Demonstrations . . . 47
5.4.1 Demonstration 4 . . . 48
5.4.1.1 Demonstration Procedure . . . 48
5.4.1.2 Concept Review . . . 49
5.4.2 Demonstration 5 . . . 49
5.4.2.1 Demonstration Procedure . . . 49
5.4.2.2 Concept Review . . . 50

6 Deployment Debugging **51**
6.1 NEM Platform Server Debug Port . . . 51
6.2 Logging . . . 51
6.3 Demonstrations . . . 52
6.3.1 Demonstration 6 . . . 52
6.3.1.1 Demonstration Procedure . . . 53
6.3.1.2 Concept Review . . . 54
6.3.2 Demonstration 7 . . . 54
6.3.2.1 Demonstration Procedure . . . 54
6.3.2.2 Concept Review . . . 55

II Models **57**

7 Universal PHY Layer **59**
7.1 Model Features . . . 59
7.1.1 Pathloss Calculation . . . 59
7.1.2 Receive Power Calculation . . . 59
7.1.3 Directional Sector Antenna Support . . . 60
7.1.4 Noise Processing . . . 60
7.1.5 MAC-PHY Control Messaging . . . 61
7.2 Configuration Parameters . . . 61
7.2.1 `bandwidth` . . . 61
7.2.2 `antennagain` . . . 62

7.2.3 systemnoisefigure . . . 62
7.2.4 frequencyofinterest . . . 62
7.2.5 pathlossmode . . . 62
7.2.6 noiseprocessingmode . . . 63
7.2.7 defaultconnectivitymode . . . 63
7.2.8 txpower . . . 63
7.2.9 frequency . . . 63
7.2.10 antennaazimuthbeamwidth . . . 64
7.2.11 antennaelevationbeamwidth . . . 64
7.2.12 antennaazimuth . . . 64
7.2.13 antennaelevation . . . 65
7.2.14 antennatype . . . 65
7.2.15 subid . . . 65
7.3 Packet Processing Flows . . . 65
7.4 Demonstrations . . . 67
7.4.1 Demonstration 8 . . . 68
7.4.1.1 Demonstration Procedure . . . 68
7.4.1.2 Concept Review . . . 69

8 RF Pipe MAC Layer 71
8.1 Model Features . . . 71
8.2 Configuration Parameters . . . 72
8.2.1 enablepromiscuousmode . . . 72
8.2.2 enabletighttiming . . . 72
8.2.3 transmissioncontrolmap . . . 72
8.2.4 datarate . . . 73
8.2.5 delay . . . 73
8.2.6 jitter . . . 73
8.2.7 pcrcurveuri . . . 74
8.2.8 flowcontrolenable . . . 74
8.2.9 flowcontroltokens . . . 74
8.3 Packet Completion Rate . . . 75
8.4 Packet Processing Flows . . . 76
8.5 Demonstrations . . . 79
8.5.1 Demonstration 9 . . . 79
8.5.1.1 Demonstration Procedure . . . 79
8.5.1.2 Concept Review . . . 81

9 IEEE 802.11abg MAC Layer 83
9.1 Model Features . . . 83
9.2 Configuration Parameters . . . 84
9.2.1 mode . . . 84
9.2.2 enablepromiscuousmode . . . 84
9.2.3 distance . . . 84
9.2.4 unicastrate . . . 85
9.2.5 multicastrate . . . 85
9.2.6 rtsthreshold . . . 86
9.2.7 wmmenable . . . 86
9.2.8 pcrcurveuri . . . 87
9.2.9 flowcontrolenable . . . 87
9.2.10 flowcontroltokens . . . 87
9.2.11 queuesize . . . 87
9.2.12 cwmin . . . 88
9.2.13 cwmax . . . 88

9.2.14 aifs . . . 89
9.2.15 txop . . . 89
9.2.16 retrylimit . . . 90
9.3 Packet Completion Rate . . . 90
9.4 Packet Processing Flows . . . 91
9.5 Demonstrations . . . 95
9.5.1 Demonstration 10 . . . 95
9.5.1.1 Demonstration Procedure . . . 96
9.5.1.2 Concept Review . . . 96

10 Comm Effect Shim Layer **97**
10.1 Model Features . . . 97
10.2 Configuration Parameters . . . 97
10.2.1 defaultconnectivity . . . 97
10.2.2 filterfile . . . 98
10.2.3 groupid . . . 98
10.2.4 enablepromiscuousmode . . . 98
10.2.5 enabletighttimingmode . . . 99
10.2.6 receivebufferperiod . . . 99
10.3 Static Filters . . . 99
10.4 Packet Processing Flows . . . 102
10.5 Demonstrations . . . 104
10.5.1 Demonstration 11 . . . 104
10.5.1.1 Demonstration Procedure . . . 104
10.5.1.2 Concept Review . . . 106

III Transports **107**

11 Virtual Transport **109**
11.1 Transport Features . . . 109
11.2 Configuration Parameters . . . 109
11.2.1 address . . . 109
11.2.2 arpcacheenable . . . 110
11.2.3 arpmode . . . 110
11.2.4 bitrate . . . 110
11.2.5 broadcastmode . . . 111
11.2.6 device . . . 111
11.2.7 devicepath . . . 111
11.2.8 flowcontrolenable . . . 112
11.2.9 mask . . . 112
11.3 Flow Control . . . 112
11.4 Packet Processing Flows . . . 113
11.5 Demonstrations . . . 115
11.5.1 Demonstration 12 . . . 115
11.5.1.1 Demonstration Procedure . . . 115
11.5.1.2 Concept Review . . . 116

12 Raw Transport **117**
12.1 Configuration Parameters . . . 117
12.1.1 bitrate . . . 117
12.1.2 broadcastmode . . . 117
12.1.3 arpcacheenable . . . 118
12.1.4 device . . . 118

12.2 Transport Interoperability . . . 118
12.3 Packet Processing Flows . . . 119
12.4 Demonstrations . . . 121
12.4.1 Demonstration 13 . . . 121
12.4.1.1 Demonstration Procedure . . . 121
12.4.1.2 Concept Review . . . 124

IV Events **125**

13 Mitre Mobility Model Event Generator **127**
13.1 Configuration Parameters . . . 127
13.1.1 `inputfileformat` . . . 127
13.1.2 `inputfilecount` . . . 127
13.1.3 `totalnodes` . . . 128
13.1.4 `maxnemidpresent` . . . 128
13.1.5 `repeatcount` . . . 128
13.1.6 `utmzone` . . . 128
13.1.7 `entryreplay` . . . 129
13.1.8 `publishpathlossevents` . . . 129
13.1.9 `publishlocationevents` . . . 129
13.2 Mitre Mobility Model Format . . . 129
13.3 Demonstrations . . . 130
13.3.1 Demonstration 14 . . . 130
13.3.1.1 Demonstration Procedure . . . 130
13.3.1.2 Concept Review . . . 131

14 Emulation Script Event Generator **133**
14.1 Configuration Parameters . . . 133
14.1.1 `inputfile` . . . 133
14.1.2 `totalnodes` . . . 133
14.1.3 `repeatcount` . . . 133
14.1.4 `schemalocation` . . . 134
14.1.5 Emulation Script Data Format . . . 134
14.2 Demonstrations . . . 135
14.2.1 Demonstration 15 . . . 135
14.2.1.1 Demonstration Procedure . . . 135

15 Emulation Event Log Generator **137**
15.1 Configuration Parameters . . . 138
15.1.1 `inputfile` . . . 138
15.1.2 `loader` . . . 138
15.2 Emulation Event Log Format . . . 139
15.3 Demonstrations . . . 139
15.3.1 Demonstration 16 . . . 139
15.3.1.1 Demonstration Procedure . . . 139
15.3.1.2 Concept Review . . . 140

16 Comm Effect Event Generator **141**
16.1 Configuration Parameters . . . 141
16.1.1 `inputfile` . . . 141
16.1.2 `totalnodes` . . . 141
16.1.3 `maxnemidpresent` . . . 141
16.1.4 `repeatcount` . . . 142
16.1.5 `entryreplay` . . . 142

16.2 Comm Effect Impairment Format . . . 142
16.3 Demonstrations . . . 144
16.3.1 Demonstration 17 . . . 144
16.3.1.1 Demonstration Procedure . . . 144

17 Antenna Direction Event Generator 147
17.1 Configuration Parameters . . . 147
17.1.1 `inputfileformat` . . . 147
17.1.2 `inputfilecount` . . . 147
17.1.3 `totalnodes` . . . 147
17.1.4 `repeatcount` . . . 148
17.2 Antenna Direction Format . . . 148
17.3 Demonstrations . . . 149
17.3.1 Demonstration 18 . . . 149
17.3.1.1 Demonstration Procedure . . . 149
17.3.1.2 Concept Review . . . 149

18 GPSd Location Agent 151
18.1 Configuration Parameters . . . 151
18.1.1 `gpsdcontrolsocket` . . . 151
18.1.2 `pseudoterminalfile` . . . 151
18.1.3 `gpsdconnectionenabled` . . . 151
18.2 Demonstrations . . . 152
18.2.1 Demonstration 19 . . . 152
18.2.1.1 Demonstration Procedure . . . 152

V Python Bindings 155

19 Event Service Python Bindings 157
19.1 Configuration . . . 157
19.2 EventService . . . 158
19.3 EventLocation . . . 158
19.4 EventPathloss . . . 159
19.5 EventCommEffect . . . 160
19.6 EventAntennaDirection . . . 161
19.7 Demonstrations . . . 162
19.7.1 Demonstration 20 . . . 162
19.7.1.1 Demonstration Procedure . . . 162
19.7.1.2 Concept Review . . . 163

20 EMANE Library Python Bindings 165
20.1 Configuration . . . 165
20.2 Logger . . . 166
20.3 Event Agent Manager . . . 166
20.4 Transport Manager . . . 167
20.5 Platform Server . . . 167
20.6 Putting It Together . . . 168
20.7 Demonstrations . . . 169
20.7.1 Demonstration 21 . . . 169
20.7.1.1 Demonstration Procedure . . . 169
20.7.1.2 Concept Review . . . 170

Chapter 1

Introduction

The Extendable Mobile Ad-hoc Network Emulator (EMANE) is an open source[1] framework which provides wireless network experimenters with a highly flexible modular environment for use during the design, development and testing of simple and complex network architectures. EMANE provides a set of well-defined APIs to allow independent development of network emulation modules, emulation/application boundary interfaces and emulation environmental data distribution mechanisms.

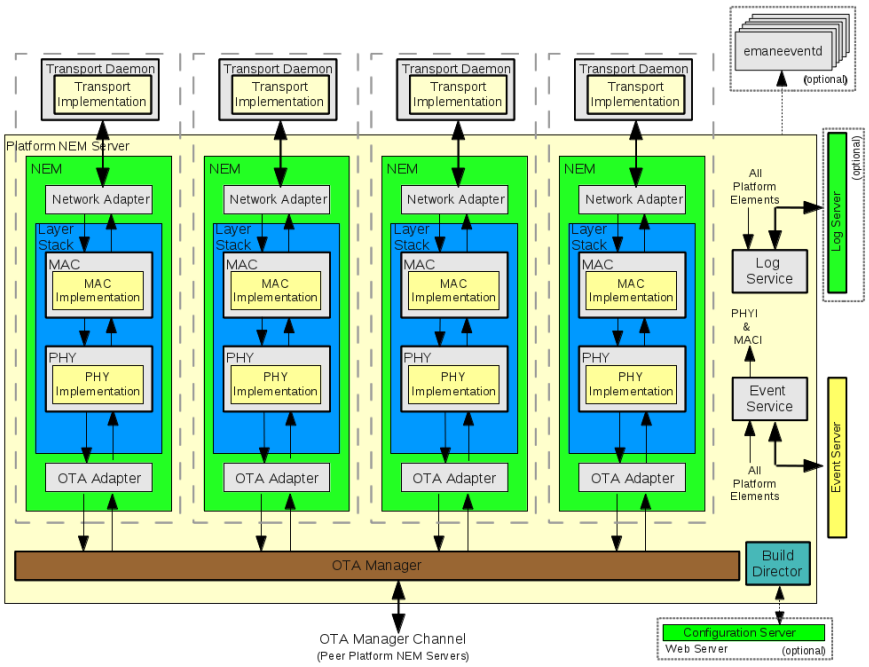


Figure 1.1: EMANE component diagram.

[1]EMANE is released under the BSD license.

1.1 Overview

A Network Emulation Module (NEM) is a logical component that encapsulates all of the functionality necessary to emulate a particular type of network technology. Each NEM is composed of plugin instances that are layered and interconnected to form an emulation stack. There are three types of NEM layers: Medium Access Control (MAC) Layer, Physical (PHY) Layer, and Shim Layer.

Emulation/application boundary interfaces provide the functionality responsible for transferring data between the emulation and application domain. The application domain refers to entities running during the experiment which are not part of EMANE. The application domain includes, but is not limited to, user space processes, kernel space processes, network appliances or any other device, system, or component that is not operating on behalf of or as part of EMANE. Emulation/application boundary interfaces are referred to as *transports*.

Emulation environmental data, such as pathloss information and GPS location information, are opaquely distributed in realtime to one or more targeted EMANE plugin components. These emulation data messages are referred to as *events*. Events can be generated using either an EMANE framework API or via a library interface. The EMANE framework provides an API that can be used to develop plugins which generate events based on experiment scenarios. These plugins are referred to as *event generators*. Additionally, a C language library API (libemaneeventservice) is provided to allow development of applications with embedded event generation capabilities.

Emulation environmental data can traverse the emulation/application boundary using agents that translate emulation domain data into application domain data. These agents are referred to as *event agents*. Event agents facilitate the reuse of any experiment scenario information propagated via an event that is of interest outside of the emulation domain. For example, position information contained in the EMANE Location Event which is used by some PHY layers to compute pathloss may also be of interest to application domain entities that require GPS location.

The key to the flexibility of EMANE is the use of application build factories to determine which emulation plugin components to instantiate and where they reside once deployed. Each instantiated plugin component belongs to a type specific component container. There are four types of component containers that can be configured to create and manage a variable number of plugin component instances:

- The *NEM Platform Server* creates and manages network emulation modules.
- The *Transport Daemon* creates and manages emulation/application boundary interfaces.
- The *Event Service* creates and manages emulation event generators.
- The *Event Daemon* creates and manages event agents that bridge emulation environmental data between the emulation and application space.

The number of component containers, the amount of plugin components contained in each container, and the location of those containers is referred to as the *EMANE deployment*. There are three types of EMANE deployments: centralized, distributed, and hybrid.

The goal of this manual is to introduce the EMANE framework and each of the components that comprise the standard EMANE distribution through a series of hands on demonstrations using the EMANE Demo Virtual Machine.

1.2 EMANE Demo Virtual Machine

The EMANE Demo Virtual Machine is a 32 bit RPM based Linux VM fully configured with the latest EMANE release and the latest EMANE User Manual Demonstrations. Any modern Linux installation with

lxc Linux Container[2] support running the latest EMANE release will be able to execute the EMANE User Manual demonstrations.

Each demonstration makes use of Linux Containers to create lightweight virtual nodes. Each container node is assigned their own Network and PID namespace to provide network stack and process isolation. Each demonstration configures the containers and the applications running in each container node respective to the features and mechanisms being demonstrated. Additionally, each container node is running an SSH server to allow hands on interaction and examination during the demonstration. Each container has a configured back-channel interface which is used to command and control the container. Figure 1.2 shows the demonstration node network diagram. In some demonstrations the back-channel interface is also used as the EMANE Over-The-Air interface and the EMANE Event interface. Table 1.1 lists the demonstration node back-channel addresses.

Table 1.1: Demonstration node back-channel interfaces.

	Back Channel	
NEM Id	Host Name	Address
1	node-1	10.99.0.1
2	node-2	10.99.0.2
3	node-3	10.99.0.3
4	node-4	10.99.0.4
5	node-5	10.99.0.5
6	node-6	10.99.0.6
7	node-7	10.99.0.7
8	node-8	10.99.0.8
9	node-9	10.99.0.9
10	node-10	10.99.0.10

Table 1.2: Demonstration node emulation boundary interface addresses.

	Wireless	
NEM Id	Host Name	Address
1	radio-1	10.100.0.1
2	radio-2	10.100.0.2
3	radio-3	10.100.0.3
4	radio-4	10.100.0.4
5	radio-5	10.100.0.5
6	radio-6	10.100.0.6
7	radio-7	10.100.0.7
8	radio-8	10.100.0.8
9	radio-9	10.100.0.9
10	radio-10	10.100.0.10

Depending on the transport used by a specific demonstration, the demonstration node interface used as the emulation/application boundary will differ but the address assigned will remain the same. Demonstrations using the Virtual Transport will assign addresses on the 10.100.0.0/24 network to `emane0`. Demonstrations using the Raw transport will assign the same addresses to `eth1`. Table 1.2 lists the emulation/application boundary interface addresses.

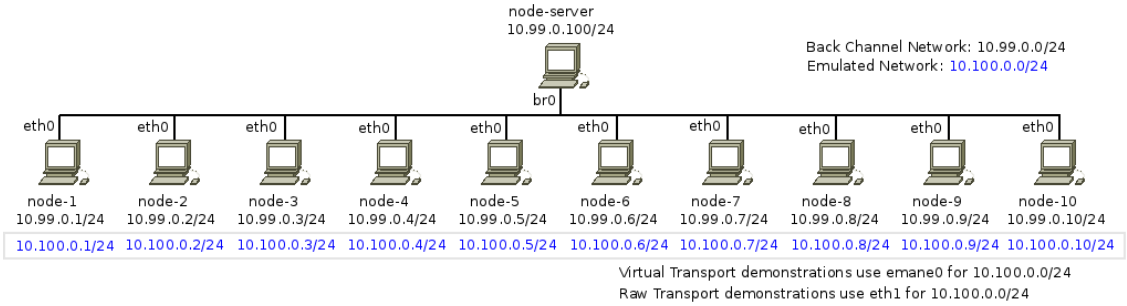


Figure 1.2: Demonstration node network diagram.

[2] http://lxc.sourceforge.net

1.3 EMANE Deployment Diagrams

Each of the EMANE demonstrations found throughout this manual contain EMANE Deployment Diagrams. These diagrams use a set of icons to convey certain aspects of EMANE deployments. Deployment diagrams depict NEM Platforms and their contents along with Transport Daemons and their contents. Deployment diagrams do not convey topology or routing information about the wireless network or networks being emulated. Figure 1.3 shows the deployment diagram key.

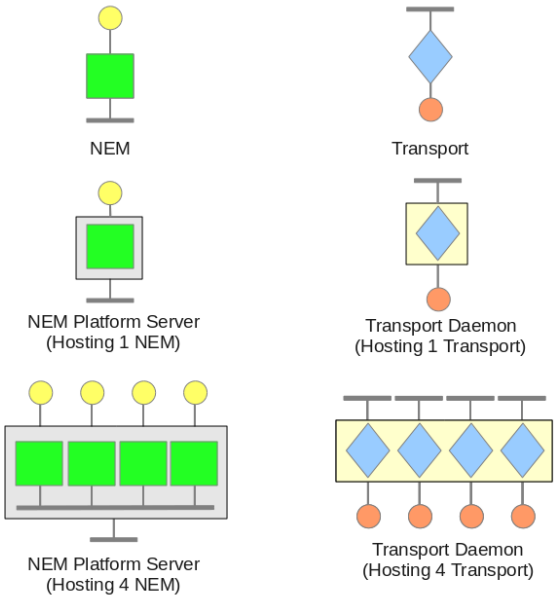


Figure 1.3: EMANE Deployment Diagram Key.

Part I

Emulation Infrastructure

Chapter 2

Infrastructure Basics

Designing an EMANE experiment first starts with selecting the appropriate NEM(s) necessary to accomplish the experiment objectives. For illustrative proposes, we will be examining a simple four node Bypass NEM experiment in order to introduce the various aspects of EMANE deployments. The deployment diagrams show in Figure 2.1 and Figure 2.2 depict a centralized and distributed EMANE deployment, respectively.

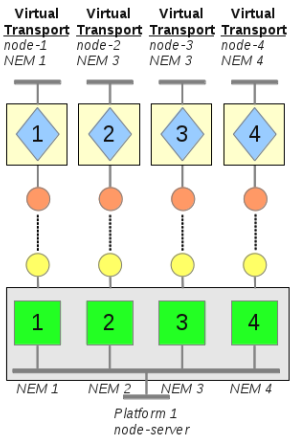


Figure 2.1: Four node centralized deployment.

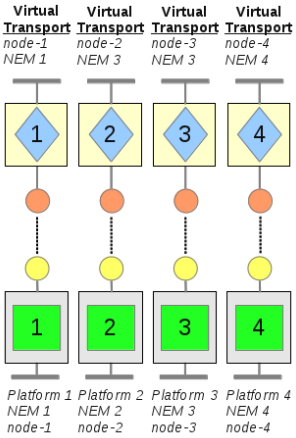


Figure 2.2: Four node distributed deployment.

2.1 NEM Platform Server

NEMs are created and managed by an NEM Platform Server. NEMs managed by the same NEM Platform Server are referred to as being part of the same platform. The determination as to how many NEM platforms are required to support a given emulation experiment is a function of the number of NEMs, the complexity of the emulation modules, and the processing and memory resources available.

Inter-NEM communication is manged by the NEM Platform Server. NEMs belonging to the same platform use thread shared memory message passing. NEMs belonging to different platforms communicate using a multicast channel referred to as the Over-The-Air (OTA) Channel. The EMANE infrastructure delivers all OTA messages to every NEM participating in the emulation. This provides the opportunity to

model more complex PHY phenomena such as RF interference.

NEM Platform Servers instantiate one or more NEMs based on an XML configuration file. The NEM Platform Server application is named `emane` and the NEM Platform Server configuration file is referred to as the *Platform XML*. Deploying a centralized, distributed, or hybrid EMANE emulation experiment is a function of the configuration contained within one or more platform XML files. Listing 2.1 shows the man page entry for the `emane` application.

A centralized deployment is one in which there is a single NEM Platform Server that instantiates all of the NEMs contained in the deployment.

A distributed deployment is one in which there are multiple NEM Platform Servers each containing a single NEM instance. In a distributed deployment the number of NEM Platform Servers equals the number of NEMs in the deployment.

A hybrid deployment is one in which there are multiple NEM Platform Servers, with at least one containing multiple NEM instances.

```
emane(1)                                                          emane(1)

NAME
       emane - EMANE Platform NEM Server

SYNOPSIS
       emane [OPTIONS]... CONFIG_URI

DESCRIPTON
       emane  is  the Platform NEM server application that creates and manages
       one or more NEMs. Each NEM is connected to a transport that facilitates
       the  opaque packet entry/exit point for the NEM network stack. Communi-
       cation between NEMs contained in the same emane platform is done inter-
       nal  to the platform. Communication between multiple emane platforms is
       done using the Over-The-Air (OTA) multicast channel.

       CONFIG_URI is the XML containing the Platform NEM Server configuration.

OPTIONS
       The following options are supported:

       --version
              Display version and exit

       --loglevel [0,4]
              Set the current application log level.
              0 - No Logging
              1 - Abort Level
              2 - Error Level
              3 - Stat Level
              4 - Debug Level

       --logserver DESTINATION:PORT
              Enable remote logging and direct all  logging  messages  to  the
              given endpoint.

       --realtime
              Run  with  realtime  priority and SCHED_RR.  Must have superuser
              privilege.

       --logfile FILE
              Log to a file

       --daemonize
              Run EMANE in the background

       --syslog
              Log to syslogd
```

Listing 2.1: `emane` man page entry.

2.1.1 Centralized Deployment Example

The centralized deployment depicted in Figure 2.1 can be instantiated using the platform XML shown in Listing 2.2. A single NEM Platform Server will instantiate four instances of the NEM defined via the *bypass-nem.xml* configuration file. All emulation functionality will be performed by a single server. In this example, the NEM Platform Server host name is `node-server`. The four NEMs it will instantiate are highlighted in yellow. The NEMs are assigned ids `1`, `2`, `3` and `4`, respectively. The platform XML also specifies that the emulation/application boundaries associated with the four NEMs will be residing on `node-1`, `node-2`, `node-3`, and `node-4`, respectively.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
3
4 <platform name="Platform 1" id="1">
5  <param name="otamanagerchannelenable" value="off"/>
6  <param name="eventservicegroup" value="224.1.2.8:45703"/>
7  <param name="eventservicedevice" value="lo"/>
8
9  <nem name="NEM-1" id="1" definition="bypassnem.xml">
10  <param name="platformendpoint" value="node-server:8201"/>
11  <param name="transportendpoint" value="node-1:8301"/>
12  <transport definition="transvirtual.xml">
13   <param name="address" value="10.100.0.1"/>
14   <param name="mask" value="255.255.255.0"/>
15  </transport>
16 </nem>
17
18 <nem name="NEM-2" id="2" definition="bypassnem.xml">
19  <param name="platformendpoint" value="node-server:8202"/>
20  <param name="transportendpoint" value="node-2:8302"/>
21  <transport definition="transvirtual.xml">
22   <param name="address" value="10.100.0.2"/>
23   <param name="mask" value="255.255.255.0"/>
24  </transport>
25 </nem>
26
27 <nem name="NEM-3" id="3" definition="bypassnem.xml">
28  <param name="platformendpoint" value="node-server:8203"/>
29  <param name="transportendpoint" value="node-3:8303"/>
30  <transport definition="transvirtual.xml">
31   <param name="address" value="10.100.0.3"/>
32   <param name="mask" value="255.255.255.0"/>
33  </transport>
34 </nem>
35
36 <nem name="NEM-4" id="4" definition="bypassnem.xml">
37  <param name="platformendpoint" value="node-server:8204"/>
38  <param name="transportendpoint" value="node-4:8304"/>
39  <transport definition="transvirtual.xml">
40   <param name="address" value="10.100.0.4"/>
41   <param name="mask" value="255.255.255.0"/>
42  </transport>
43 </nem>
44
45 </platform>
```

Listing 2.2: NEM Platform Server configuration for Figure 2.1.

Since all four NEMs contained in this experiment reside in a single NEM Platform Server there is no need to enable the OTA Manager Channel. Line 5 of Listing 2.2 disables the OTA Manager Channel by setting the `otamanagerchannelenable` parameter to `off`.

Additionally, for the purposes for this experiment, emulation events will be generated locally on the same server that is hosting the NEM Platform Server. Lines 6-7 of Listing 2.2 configure the Event Service Channel

and associate the channel with the loopback interface using the `eventservicegroup` and `eventservicedevice` parameters.

2.1.2 Distributed Deployment Example

The distributed deployment depicted in Figure 2.2 can be instantiated using the platform XML show in Listings 2.3, 2.4, 2.5, and 2.6. Four NEM Platform Servers will each instantiate a single instance of the NEM defined via the *bypassnem.xml* configuration file. In this example, `node-1`, `node-2`, `node-3`, and `node-4` each host an NEM Platform Server containing a single NEM. Each node also hosts an emulation/application boundary. Since both the NEM Platform Server and the emulation/application boundary reside on the same host they can communicate via their respective host's loopback interface.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
<platform name="Platform 1" id="1">
  <param name="otamanagerchannelenable" value="on"/>
  <param name="otamanagerdevice" value="eth0"/>
  <param name="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>

  <nem name="NEM-1" id="1" definition="bypassnem.xml">
    <param name="platformendpoint" value="localhost:8201"/>
    <param name="transportendpoint" value="localhost:8301"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.1"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
    </nem>
</platform>
```

Listing 2.3: NEM Platform Server 1 configuration for Figure 2.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
<platform name="Platform 2" id="2">
  <param name="otamanagerchannelenable" value="on"/>
  <param name="otamanagerdevice" value="eth0"/>
  <param name="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>

  <nem name="NEM-2" id="2" definition="bypassnem.xml">
    <param name="platformendpoint" value="localhost:8201"/>
    <param name="transportendpoint" value="localhost:8301"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.2"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
    </nem>
</platform>
```

Listing 2.4: NEM Platform Server 2 configuration for Figure 2.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
3  <platform name="Platform 3" id="3">
4    <param name="otamanagerchannelenable" value="on"/>
5    <param name="otamanagerdevice" value="eth0"/>
6    <param name="otamanagergroup" value="224.1.2.8:45702"/>
7    <param name="eventservicegroup" value="224.1.2.8:45703"/>
8    <param name="eventservicedevice" value="eth0"/>
9
10   <nem name="NEM-3" id="3" definition="bypassnem.xml">
11     <param name="platformendpoint" value="localhost:8201"/>
12     <param name="transportendpoint" value="localhost:8301"/>
13     <transport definition="transvirtual.xml">
14       <param name="address" value="10.100.0.3"/>
15       <param name="mask" value="255.255.255.0"/>
16     </transport>
17     </nem>
18 </platform>
```

Listing 2.5: NEM Platform Server 3 configuration for Figure 2.2.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
3  <platform name="Platform 4" id="4">
4    <param name="otamanagerchannelenable" value="on"/>
5    <param name="otamanagerdevice" value="eth0"/>
6    <param name="otamanagergroup" value="224.1.2.8:45702"/>
7    <param name="eventservicegroup" value="224.1.2.8:45703"/>
8    <param name="eventservicedevice" value="eth0"/>
9
10   <nem name="NEM-4" id="4" definition="bypassnem.xml">
11     <param name="platformendpoint" value="localhost:8201"/>
12     <param name="transportendpoint" value="localhost:8301"/>
13     <transport definition="transvirtual.xml">
14       <param name="address" value="10.100.0.4"/>
15       <param name="mask" value="255.255.255.0"/>
16     </transport>
17     </nem>
18 </platform>
```

Listing 2.6: NEM Platform Server 4 configuration for Figure 2.2.

The four NEM Platform Servers contained in this experiment use the OTA Manager Channel to communicate. Lines 4-6 of Listings 2.3, 2.4, 2.5, and 2.6 enable and configure the OTA Manager Channel using the `otamanagerchannelenable`, `otamanagerdevice`, and `otamanagergroup` parameters.

Additionally, since there is more than one NEM Platform Server the Event Service Channel must be associated with an interface that is routeable between them. Lines 7-8 of Listings 2.3, 2.4, 2.5, and 2.6 configure the Event Service Channel and associate the channel with the `eth0` interface using the `eventservicegroup` and `eventservicedevice` parameters.

2.1.3 NEM Platform Server Configuration Parameters

2.1.3.1 `otamanagergroup`

The Over-The-Air (OTA) Channel mulitcast endpoint used to communicate between multiple NEM Platform Servers in an EMANE deployment.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="otamanagergroup" value="224.1.2.8:45702">`

Parameter value format description:

`<IPv4 Multicast Group>:<Port> | <IPv6 Multicast Group>/<Port>`

Name	Description
IPv4 or IPv6 Multicast Address	Over-The-Air Channel multicast group
Port	Over-The-Air Channel UDP port

2.1.3.2 `otamanagerdevice`

The network device to associate with the OTA Manager Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="otamanagerdevice" value="eth0"/>`

2.1.3.3 `otamanagerchannelenable`

Enable or disable the OTA Manager Channel. When disabled, there is no inter-NEM Platform Server communication and only NEMs managed locally by a single NEM Platform Server will be able to communicate.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="otamanagerchannelenable" value="on"/>`

2.1.3.4 `eventservicegroup`

The Event Service Channel mulitcast endpoint used to communicate events between EMANE components in an EMANE deployment.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicegroup" value="224.1.2.8:45703">`

Parameter value format description:
`<IPv4 Multicast Group>:<Port> | <IPv6 Multicast Group>/<Port>`

Name	Description
IPv4 or IPv6 Multicast Address	Event Service Channel multicast group
Port	Event Service Channel UDP port

2.1.3.5 `eventservicedevice`

The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicedevice" value="eth0"/>`

2.1.3.6 `debugport`

NEM Platform Server telnet debug port UDP port.

Type: Unsigned 16 bit Integer
Range: [`0`, `65535`]
Default: `47000`
Count: 1
XML Format: `<param name="debugport" value="47000"/>`

2.1.3.7 `debugportenable`

Enable or disable the NEM Platform Server telnet debug port.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="debugportenable" value="off"/>`

2.1.4 Shared Configuration Parameters

The `platformendpoint` and `transportendpoint` configuration parameters are shared by both the NEM Platform Server and the emulation/application boundary. The `platformendpoint` configuration parameter names the socket address to which the NEM Platform Server's NEM binds to receive traffic from the emulation/application boundary instance. The boundary instance sends all traffic to this address. The `transportendpoint` configuration parameter names the socket address to which the emulation/application boundary binds and to which the NEM instance sends all of its traffic. Figure 2.3 illustrates the parameter

relationships. The NEM Platform Server and emulation/application boundary require both parameters and the values must be the same for both.

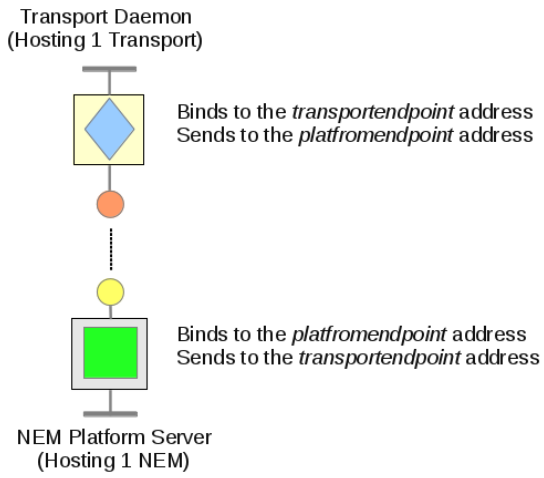


Figure 2.3: `platformendpoint` and `transportendpoint`.

2.1.4.1 `platformendpoint`

The endpoint that an NEM should bind to in order to receive messages from its respective emulation/application boundary. The address an emulation/application boundary instance sends to when communicating with its respective NEM.

Type:	String
Range:	N/A
Default:	N/A
Count:	1
XML Format:	`<param name="platformendpoint" value="node-server:8201"/>`

Parameter value format description:

`<IPv4 Unicast Address>:<Port> | <IPv6 Unicast Group>/<Port>`

Name	Description
IPv4 or IPv6 Unicast Address	NEM bind address and emulation/boundary send address
Port	UDP port

2.1.4.2 `transportendpoint`

The endpoint that an emulation/application boundary should bind to in order to receive messages from its respective NEM. The address an NEM instance sends to when communicating with its respective emulation/application boundary instance.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="transportendpoint" value="node-1:8301"/>`

Parameter value format description:

<IPv4 Unicast Address>:<Port> | <IPv6 Unicast Group>/<Port>

Name	Description
IPv4 or IPv6 Unicast Address	Emulation/Application boundary bind address and NEM send address
Port	UDP port

2.2 Transport Daemon

Emulation/Application boundary components are responsible for transferring data between the emulation and application domains[3]. The top of every emulation stack (NEM) is connected to one side of its associated emulation/application boundary instance. The other side of the emulation/application boundary instance interfaces with the application domain. Data received from the application domain is transmitted opaquely to the boundary's respective NEM for OTA message processing. The emulation boundary is the only EMANE component aware of the exact format of the application domain data. Along with the opaque data, the boundary component supplies the destination address, either a unicast or broadcast NEM identifier, to the NEM stack. This decoupling of application domain data knowledge from the emulation functionality allows NEMs to be used in conjunction with various types of networks, for example, IP and non-IP based networks. EMANE boundary components may or may not be designed to inter-operate with other dissimilar boundary components in the same experiment.

Emulation/Application boundary components are created and managed by a Transport Daemon. Boundaries managed by the same Transport Daemon are referred to as being part of that Transport Daemon. The Transport Daemon can instantiate one or more emulation/application boundaries based on an XML configuration file. The Transport Daemon application is named `emanetransportd` and the Transport Daemon configuration file is referred to as the *Transport Daemon XML*. There is nothing explicitly related to centralized, distributed, or hybrid EMANE deployments in the Transport Daemon XML. Listing 2.7 shows the man page entry for the `emanetransportd` application.

```
emanetransportd(1)                                    emanetransportd(1)

NAME
       emanetransportd - EMANE virtual interface transport

SYNOPSIS
       emanetransportd [OPTIONS]... CONFIG_URI

DESCRIPTON
       The  emanetransportd  application,  also  referred  to as the Transport
       Server, creates and manages one or more transports. Each  transport  is
       connected  to a respective NEM and is responsible for creating the net-
       work application entry/exit point into/out of its respective NEM stack.

       CONFIG_URI is the XML containing the transport daemon configuration.

OPTIONS
```

[3]The application domain refers to entities running during the experiment which are not part of EMANE. The application domain includes, but is not limited to, user space processes, kernel space processes, network appliances or any other device, system, or component that is not operating on behalf of or as part of EMANE.

```
       The following options are supported:

       --help Display usage and exit

       --version
              Display version and exit

      --loglevel [0,4]
              Set the current application log level.
              0 - No Logging
              1 - Abort Level
              2 - Error Level
              3 - Stat Level
              4 - Debug Level

       --realtime
              Run with realtime priority and SCHED_RR.   Must  have  superuser
              privledge.

       --logserver DESTINATION:PORT
              Enable  remote  logging  and  direct all logging messages to the
              given endpoint.

       --logfile FILE
              Log to a file

       --daemonize
              Run EMANE in the background

       --syslog
              Log to syslogd
```

Listing 2.7: `emanetransportd` man page entry.

Chapter 11 Virtual Transport on page 109 will describe the Virtual Transport in depth, however, a basic understanding of this emulation/application boundary is necessary in order to proceed. The Virtual Transport creates a virtual interface on the host machine that is used to route traffic into and out of the emulation domain. In both the centralized and distributed example, `node-1`, `node-2`, `node-3` and `node-4` will each have a virtual interface named `emane0` that will be assigned the addresses `10.100.0.1/24`, `10.100.0.2/24`, `10.100.0.3/24` and `10.100.0.4/24`, respectively.

The only difference between the Transport Daemon XML generated from Listing 2.2 and the XML generated from Listings 2.3, 2.4, 2.5, and 2.6, are the `platformendpoint` and `transportendpoint` parameter values.

2.2.1 Centralized Deployment Example

Listing 2.8, 2.9, 2.10, and 2.11 show the Transport Daemon XML generated automatically from the NEM Platform Server XML shown in Listing 2.2. Automatic XML configuration generation is described in Section 5.2 Automatic XML Generation on page 45.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
3  <transportdaemon>
4    <instance nemid="1">
5      <param name="transportendpoint" value="node-1:8301"/>
6      <param name="platformendpoint" value="node-server:8201"/>
7      <transport definition="transvirtual.xml">
8        <param name="address" value="10.100.0.1"/>
9        <param name="mask" value="255.255.255.0"/>
10     </transport>
11   </instance>
12 </transportdaemon>
```

Listing 2.8: Transport Daemon 1 XML for NEM 1 from centralized NEM Platform Server XML Listing 2.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="2">
    <param name="transportendpoint" value="node-2:8302"/>
    <param name="platformendpoint" value="node-server:8202"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.2"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.9: Transport Daemon 2 XML for NEM 2 from centralized NEM Platform Server XML Listing 2.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="3">
    <param name="transportendpoint" value="node-3:8303"/>
    <param name="platformendpoint" value="node-server:8203"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.3"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.10: Transport Daemon 3 XML for NEM 3 from centralized NEM Platform Server XML Listing 2.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="4">
    <param name="transportendpoint" value="node-4:8304"/>
    <param name="platformendpoint" value="node-server:8204"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.4"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.11: Transport Daemon 4 XML for NEM 4 from centralized NEM Platform Server XML Listing 2.2.

2.2.2 Distributed Deployment Example

Listing 2.12, 2.13, 2.14, and 2.15 show the Transport Daemon XML generated automatically from the NEM Platform Server XML shown in Listing 2.3, 2.4, 2.5, and 2.6, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="1">
    <param name="transportendpoint" value="localhost:8301"/>
    <param name="platformendpoint" value="localhost:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.1"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.12: Transport Daemon 1 XML for NEM 1 from distributed NEM Platform Server XML Listing 2.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="2">
    <param name="transportendpoint" value="localhost:8301"/>
    <param name="platformendpoint" value="localhost:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.2"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.13: Transport Daemon 2 XML for NEM 2 from distributed NEM Platform Server XML Listing 2.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="3">
    <param name="transportendpoint" value="localhost:8301"/>
    <param name="platformendpoint" value="localhost:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.3"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.14: Transport Daemon 3 XML for NEM 3 from distributed NEM Platform Server XML Listing 2.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="4">
    <param name="transportendpoint" value="localhost:8301"/>
    <param name="platformendpoint" value="localhost:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="10.100.0.4"/>
      <param name="mask" value="255.255.255.0"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 2.15: Transport Daemon 4 XML for NEM 4 from distributed NEM Platform Server XML Listing 2.6.

2.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

2.3.1 Demonstration 1

This demonstration deploys a four node centralized Bypass NEM emulation experiment illustrated in Figure 2.4. The goal of this demonstration is to become familiar with the basic EMANE components in a centralized deployment.

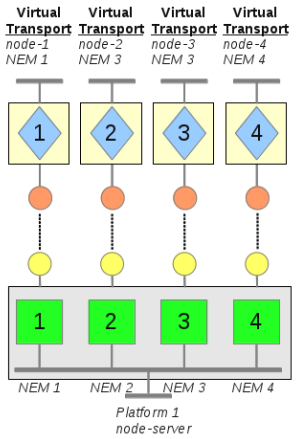


Figure 2.4: Demonstration 1 - Four node centralized Bypass NEM deployment.

2.3.1.1 Demonstration Procedure

1. Review the Demonstration 1 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/1
[emane@emanedemo 1] less platform.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 1]$ sudo ./lxc-demo-start.sh
```

3. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

4. Connect to virtual `node-1`.

```
[emane@emanedemo 1]$ ssh node-1
```

5. Review the running processes.

```
[emane@node-1 ~]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S+     0:00 /usr/lib/lxc/lxc-init -- /tmp/lxc-node/1/1/init.sh -s 11:29:
    6 ?        Sl     0:00 emanetransportd -r -d /home/emane/demonstration/1/transportdaem
   13 ?        Ssl    0:00 /usr/sbin/olsrd -f /home/emane/demonstration/1/routing1.conf
   16 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/tmp/lxc-node/1/1/run/sshd.pid
   17 ?        S      0:00 sshd: emane [priv]
   19 ?        S      0:00 sshd: emane@pts/0
   20 pts/0    Ss     0:00 -bash
   80 pts/0    R+     0:00 ps ax
```

6. Review `node-1`'s network interface configuration.

```
[emane@node-1 ~]$ ifconfig
bmf0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.100.0.1  P-t-P:10.100.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

emane0    Link encap:Ethernet  HWaddr 02:02:00:00:00:01
          inet addr:10.100.0.1  Bcast:10.100.0.255  Mask:255.255.255.0
          inet6 addr: fe80::2:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
```

```
          TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:5284 (5.1 KiB)  TX bytes:1794 (1.7 KiB)

eth0      Link encap:Ethernet  HWaddr 02:01:00:00:00:01
          inet addr:10.99.0.1  Bcast:10.99.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:186 errors:0 dropped:0 overruns:0 frame:0
          TX packets:129 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21057 (20.5 KiB)  TX bytes:18923 (18.4 KiB)
```

7. Review `node-1`'s routing table.

```
[emane@node-1 ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.99.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 emane0
10.100.0.2      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
10.100.0.3      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
10.100.0.4      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
224.0.0.0       0.0.0.0         240.0.0.0       U     0      0        0 bmf0
```

8. Ping another radio using the *radio-NEMID* host naming convention.

```
[emane@node-1 ~]$ ping -c 5 radio-2
PING radio-2 (10.100.0.2) 56(84) bytes of data.
64 bytes from radio-2 (10.100.0.2): icmp_req=1 ttl=64 time=0.745 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=2 ttl=64 time=2.64 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=3 ttl=64 time=3.94 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=4 ttl=64 time=1.69 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=5 ttl=64 time=3.32 ms

--- radio-2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 0.745/2.471/3.946/1.141 ms
```

9. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

10. Stop the demonstration.

```
[emane@emanedemo 1]$ sudo ./lxc-demo-stop.sh
```

2.3.1.2 Concept Review

1. What are the EMANE components associated with this deployment type?
2. Why was the OTA Manager Channel disabled for this demonstration?
3. What can be deduced from setting the Event Service device to `lo`?
4. What are the implications of not specifying the Event Service device?
5. Redeploy the demonstration and shutdown the NEM Platform Server. What happens when you ping another radio? Why?

```
[emane@emanedemo 1]$ sudo ./lxc-demo-start.sh
[emane@emanedemo 1]$ sudo killall -QUIT emane
[emane@emanedemo 1]$ ssh node-1
[emane@node-1 ~]$ ping -c 5 radio-2
```

2.3.2 Demonstration 2

This demonstration deploys a four node distributed Bypass NEM emulation experiment illustrated in Figure 2.5. The goal of this demonstration is to become familiar with the basic EMANE components in a distributed deployment.

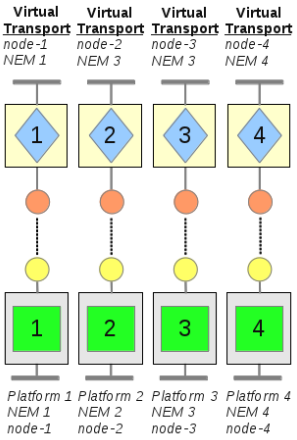


Figure 2.5: Demonstration 2 - Four node distributed Bypass NEM deployment.

2.3.2.1 Demonstration Procedure

1. Review the Demonstration 2 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/2
[emane@emanedemo 2] less platform1.xml
[emane@emanedemo 2] less platform2.xml
[emane@emanedemo 2] less platform3.xml
[emane@emanedemo 2] less platform4.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 2]$ sudo ./lxc-demo-start.sh
```

3. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

4. Connect to virtual `node-1`.

```
[emane@emanedemo 2]$ ssh node-1
```

5. Review the running processes.

```
[emane@node-1 ~]$ ps ax
 PID TTY      STAT   TIME COMMAND
    1 ?        S+     0:00 /usr/lib/lxc/lxc-init -- /tmp/lxc-node/2/1/init.sh -s 14:11:
    6 ?        Sl     0:00 emanetransportd -r -d /home/emane/demonstration/2/transportdaem
   11 ?        Sl     0:00 emane /home/emane/demonstration/2/platform1.xml -r -d -l 2 -f /
   24 ?        Ssl    0:00 /usr/sbin/olsrd -f /home/emane/demonstration/2/routing1.conf
   27 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/tmp/lxc-node/2/1/run/sshd.pid
   28 ?        S      0:00 sshd: emane [priv]
   30 ?        S      0:00 sshd: emane@pts/0
   31 pts/0    Ss     0:00 -bash
   91 pts/0    R+     0:00 ps ax
```

6. Review `node-1`'s network interface configuration.

```
[emane@node-1 ~]$ ifconfig
bmf0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.100.0.1  P-t-P:10.100.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

emane0    Link encap:Ethernet  HWaddr 02:02:00:00:00:01
          inet addr:10.100.0.1  Bcast:10.100.0.255  Mask:255.255.255.0
          inet6 addr: fe80::2:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:62 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:5284 (5.1 KiB)  TX bytes:1794 (1.7 KiB)

eth0      Link encap:Ethernet  HWaddr 02:01:00:00:00:01
          inet addr:10.99.0.1  Bcast:10.99.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:186 errors:0 dropped:0 overruns:0 frame:0
          TX packets:129 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:21057 (20.5 KiB)  TX bytes:18923 (18.4 KiB)
```

7. Review `node-1`'s routing table.

```
[emane@node-1 ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.99.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 emane0
10.100.0.2      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
10.100.0.3      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
10.100.0.4      0.0.0.0         255.255.255.255 UH    1      0        0 emane0
224.0.0.0       0.0.0.0         240.0.0.0       U     0      0        0 bmf0
```

8. Ping another radio using the *radio-NEMID* host naming convention.

```
[emane@node-1 ~]$ ping -c 5 radio-2
PING radio-2 (10.100.0.2) 56(84) bytes of data.
64 bytes from radio-2 (10.100.0.2): icmp_req=1 ttl=64 time=2.62 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=2 ttl=64 time=3.88 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=3 ttl=64 time=4.91 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=4 ttl=64 time=4.87 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=5 ttl=64 time=1.61 ms

--- radio-2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4020ms
rtt min/avg/max/mdev = 1.610/3.582/4.916/1.293 ms
```

9. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

10. Stop the demonstration.

```
[emane@emanedemo 2]$ sudo ./lxc-demo-stop.sh
```

2.3.2.2 Concept Review

1. What are the EMANE components associated with this deployment type?

2. Why was the OTA Manager Channel enabled for this demonstration?
3. Why was the Event Service device set to `eth0`?
4. What are the implications of not specifying the OTA Manager Channel device?
5. When is it possible to use `localhost` for the `platformendpoint` and `transportendpoint` parameters?

Chapter 3

Network Emulation Modules

A Network Emulation Module (NEM) is a logical component that encapsulates all the functionality necessary to emulate a particular type of network technology. EMANE supports two types of Network Emulation Modules: *structured* and *unstructured*.

A structured NEM is a component stack composed of a Physical (PHY) Layer implementation, a Medium Access Control (MAC) Layer implementation and zero or more Shim Layer implementations. Listing 3.1 shows the structured NEM definition of an IEEE 802.11abg NEM.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/local/share/emane/dtd/nem.dtd">
<nem name="IEEE 802.11 NEM">
  <mac definition="ieee80211abgmac.xml"/>
  <phy definition="universalphy.xml">
    <param name="subid" value="1"/>
  </phy>
  <transport definition="transvirtual.xml"/>
</nem>
```

Listing 3.1: Structured IEEE 802.11abg NEM definition.

An unstructured NEM is a component stack composed of zero or one PHY Layer implementation, zero or one MAC Layer implementation and zero or more Shim Layer implementations. Listing 3.2 shows the unstructured NEM definition of a Comm Effect NEM. From an instantiation point of view, the main difference between the two NEM Layer types, besides the number of layers, is that internal NEM definition verification checks are relaxed when unstructured NEMs are built.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="COMMEFFECT NEM" type="unstructured">
  <shim definition="commeffectshim.xml"/>
  <transport definition="transraw.xml"/>
</nem>
```

Listing 3.2: Unstructured CommEffect NEM definition.

Once instantiated, NEM component layers are configured and then connected to the layer immediately above and below. NEM layers communicate with each other using a generic message passing interface. Each layer is capable of communicating cross-layer control messages and OTA messages with their neighboring layers (See Figure 3.1). Examples of cross-layer messages include: per packet RSSI, carrier sense, and transmission control messages. Each layer has the capability to generate OTA messages for communication with their respective layer counterparts contained in different NEMs. Layers may also append and strip layer specific

headers to OTA messages.

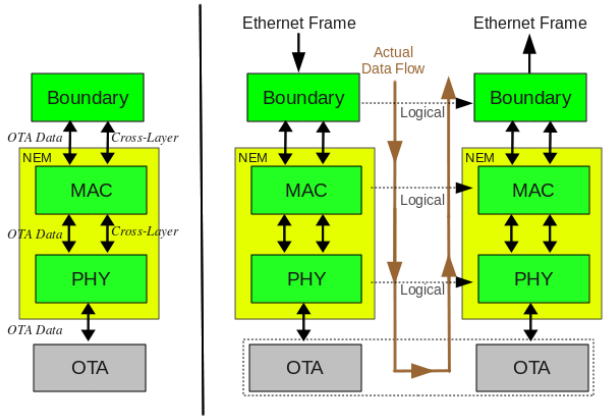


Figure 3.1: NEM Stack with Boundary and OTA connections *(Left)*. NEM Stack showing physical and logical communication paths *(Right)*.

The term *upstream* is used in EMANE to describe the path from OTA transmission up the NEM Layer stack to the application domain. The term *downstream* is used to describe the path from the application domain down the NEM Layer stack towards OTA transmission.

3.1 Defining an NEM

The Bypass NEM used in both the centralized and distributed emulation experiments presented in Chapter 2 Infrastructure Basics on page 7 is composed of two EMANE components: Bypass MAC Layer and Bypass PHY Layer.

An NEM component layer composition is defined using an XML configuration file. Listing 3.3 shows the Bypass NEM definition. An NEM layer stack is represented using `<mac>`, `<phy>`, `<shim>` and `<transport>` XML elements. Each element has a mandatory `definition` attribute which references the XML configuration associated with the respective component layer.

All NEM definitions are subject to the following rules which are enforced by the EMANE NEM DTD:

1. The order in which child elements are listed within the `<nem>` definition block corresponds to the order the plugin layers will be connected once instantiated, with the exception of the `<transport>` element.
2. The first child element in the `<nem>` definition block is the most upstream non-transport layer.
3. The `<transport>` element must be the last child element in the `<nem>` definition block.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="BYPASS NEM">
  <mac definition="bypassmac.xml"/>
  <phy definition="bypassphy.xml"/>
  <transport definition="transvirtual.xml"/>
</nem>
```

Listing 3.3: Bypass NEM XML configuration.

Line 4 of Listing 3.3 specifies the use of the MAC Layer defined in the *bypassmac.xml* file (See Listing 3.4). Line 5 specifies the use of the PHY Layer defined in the *bypassphy.xml* file (See Listing 3.5). Line 6 specifies the use of the transport defined in the *transvirtual.xml* file (See Listing 3.6).

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE mac SYSTEM "file:///usr/share/emane/dtd/mac.dtd">
3 <mac name="bypassmac" library="bypassmaclayer"/>
```

Listing 3.4: Bypass MAC XML configuration.

Line 3 of Listing 3.4 specifies the name of the MAC Layer, *bypassmac* and the name of the plugin to instantiate, *bypassmaclayer*. In Linux and OS X the plugin name corresponds to a dynamic-link library named *libbypassmac.so*. In Win32 the plugin name corresponds to a dynamic-link library named *libbypassmac.dll*.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE phy SYSTEM "file:///usr/share/emane/dtd/phy.dtd">
3 <phy name="bypassphy" library="bypassphylayer"/>
```

Listing 3.5: Bypass PHY XML configuration.

Line 3 of Listing 3.5 specifies the name of the PHY Layer, *bypassphy* and the name of the plugin to instantiate, *bypassphylayer*. In Linux and OS X the plugin name corresponds to a dynamic-link library named *libbypassphy.so*. In Win32 the plugin name corresponds to a dynamic-link library named *libbypassphy.dll*.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE transport SYSTEM "file:///usr/share/emane/dtd/transport.dtd">
3 <transport name="Tap transport" library="transvirtual">
4   <param name="bitrate" value="0.0"/>
5   <param name="devicepath" value="/dev/net/tun"/>
6   <param name="device" value="emane0"/>
7 </transport>
```

Listing 3.6: Virtual Transport XML configuration.

Line 3 of Listing 3.6 specifies the name of the emulation/application boundary, *Tap transport* and the name of the plugin to instantiate, *transvirtual*. In Linux and OS X the plugin name corresponds to a dynamic-link library named *libtransvirtual.so*. In Win32 the plugin name corresponds to a dynamic-link library named *libtransvirtual.dll*.

3.2 Physical Layer

The primary function of the Physical Layer component within a wireless network emulation environment is to accurately account for the key set of factors that impact the reception of data. In its simplest form, data reception within the PHY is based on the Signal to Interference plus Noise Ratio (SINR) at the receiving node. For most waveforms, slight variations in SINR (less than 1dB) can impact the receiver's ability to receive data as seen in Figure 3.2. These key factors for the PHY layer can be placed in two categories: 1) factors that impact the receive signal and 2) factors that impact the interference and noise.

Key factors impacting receive signal are associated with signal propagation and antenna modeling. Signal propagation models can be complex depending on many variables such as channel type (air-to-air, air-to-ground, ground-to-ground), environment (urban, rural, foliage, blockage), atmosphere, refraction, etc. As such, high fidelity modeling of such effects is typically performed offline or using dedicated computing resources and provided in realtime during emulation. Antenna effects such as pointing, gain patterns and platform motion (yaw/pitch/roll) can also drastically alter the received signal. The ability to account for

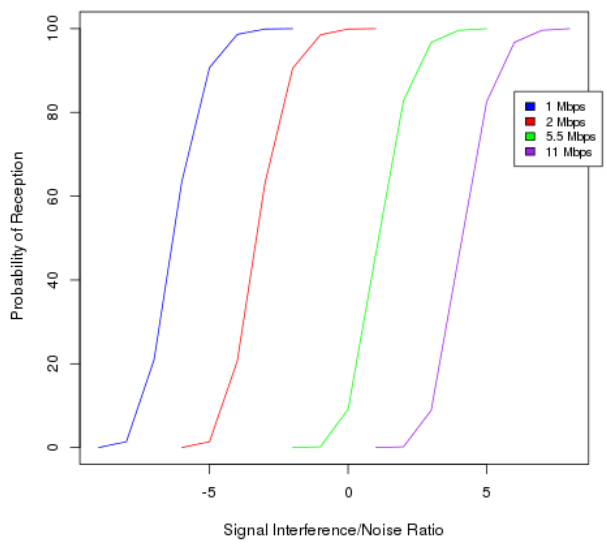


Figure 3.2: IEEE 802.11 B Mode Packet Completion Rate.

things such as inaccuracies in directional antenna pointing or an aircraft rolling during a turn is critical for accurate signal estimation.

The Noise and Interference piece of the SINR has two basic components: 1) receiver sensitivity and 2) interference from both intentional and unintentional RF emitters. Receiver sensitivity is defined as the minimum input power at the receiver for (possible) successful data reception or the noise floor of the receiver when there is no other interference. Any received signal with power less than the receiver sensitivity would not be detected at the receiver. Receiver sensitivity is based on thermal noise power (dBm) and the Noise Figure (dB) associated with the receiver. Thermal noise power, approximated as

$$-174 + 10\log(bandWidth)$$

is a function of the receiver bandwidth as indicated in Table 3.1. The Noise Figure defines any additional degradation of the signal caused by components within the RF signal chain of the receiver and as such will reduce the receiver sensitivity. Typical radio receivers will have a Noise Figure of 4 to 6 dB.

Table 3.1: Thermal Noise as a function of receiver bandwidth.

Bandwidth	Thermal Noise Power	Description
1 MHz	-114 dBm	Bluetooth channel
2 MHz	-111 dBm	Commercial GPS channel
6 MHz	-106 dBm	Analog television channel
20 MHz	-101 dBm	WLAN 802.11 channel
40 MHz	-98 dBm	WLAN 802.11 40 MHz channel
1 GHz	-84 dBm	UWB channel

Interference is defined as any additional RF energy within the RF spectrum of the receiver and can raise the over all Noise Floor. The impact of RF interference from either intentional or unintentional (fratricide) sources can be significant compared to effects on propagation from phenomena such as ducting, weather, foliage, and others. As networks/systems become more complex and RF resources remain scarce, RF interference is a common occurrence rather than an anomaly and as such needs to be accounted for within the emulation environment to properly assess overall network performance.

3.2.1 Supporting Heterogeneous Waveforms

In order to support experimentation using heterogeneous NEMs (waveforms), a common PHY Layer header is used to communicate the information necessary to model complex RF phenomena such as RF interference. The EMANE Common PHY Header is a mandatory PHY Layer model header. This header allows different physical layer models to process the potential spectrum impact of packets generated by other waveforms. A PHY Layer, just like all NEM Layers, is not limited on the number of headers it can add to a downstream packet, however the plugin API mandates the presence of the Common PHY Layer header for all OTA packets.

The Common PHY Layer Header contains the following information:

- The Registration Id of the PHY Layer Model
- The transmit power in dBm of the transmitter
- The antenna gain in dBi of the transmitter
- The timestamp of the transmitted packet
- The duration of the transmitted packet
- The center frequency in KHz
- The bandwidth in KHz
- The packet sequence number

The Common PHY Layer header also supports optional antenna pointing information:

- The transmitter antenna type
- The transmitter antenna azimuth beam width in degrees
- The transmitter antenna elevation beam width in degrees
- The transmitter antenna azimuth in degrees
- The transmitter antenna elevation in degrees

The standard EMANE distribution supplies a universal PHY Layer that is used by all the standard models. See Chapter 7 Universal PHY Layer on page 59 for more details.

3.3 Medium Access Control Layer

Packet mode channel access schemes, also commonly referred to as Medium Access Protocol (MAC), for wireless communications have significant impact on network performance associated with scale, throughput, and latency. The MAC protocol defines the mechanisms used to control access to a wireless medium shared by multiple nodes and can also include other controls (queuing, acknowledgments, retries, fragmentation, segmentation, etc.) in support of quality of service (QoS) requirements. Unlike the PHY, the MAC must be

specifically designed/tailored for a given wireless technology. For example, the channel access protocol (primary function of the MAC) can vary from very simple, such as static Time Division Multiple Access (TDMA) where nodes are statically assigned one or more time slots for transmissions, to very complex, such as a hybrid protocol that dynamically leverages the benefits of contention based access via Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) in conjunction with contention free dynamic TDMA reservations on top of advanced channelization techniques using Frequency Division Multiple Access (FDMA) or Code Division Multiple Access (CDMA).

There are significant challenges associated with accurately modeling certain channel access schemes within the framework of a realtime emulation environment. One of the major challenges associated with accurate realtime emulation of MAC protocols is timing. To highlight this, we examine CSMA/CA, a contention based access protocol which is utilized in both commercial and tactical products. CSMA/CA utilizes two basic principles: carrier sensing and collision avoidance. Here we look a little deeper into the timing associated with collision avoidance for 802.11. In 802.11, collisions are minimized by requiring all nodes with a packet for transmit to select a random slot within a given contention window once the channel becomes idle. The size of a slot is on the order of tens of microseconds, sufficiently long enough in real systems to ensure multiple nodes selecting different slots will not step on one another. Within an emulated system, however, the time it takes for a packet to get from one emulated node to another can be significantly larger than the contention delay implying that actual implementation of the protocol will not provide an accurate representation of the collision and back-off phenomena associated with CSMA/CA. Advanced statistical models or simplistic approximations can be utilized to account for such a phenomena, but the determination of which model is appropriate should be based on the research/test objective.

3.4 Shim Layer

Sometimes it is useful or necessary to monitor or modify the communication between contiguous layers of an NEM. This can be achieved by creating and inserting a Shim Layer between the layers of interest. A Shim Layer allows interaction with the OTA and cross-layer messaging exchanged between contiguous layers without requiring modifications to those layers. Shims may generate their own OTA messages for communication with their respective Shim Layer contained in a different NEM and can append and strip layer specific headers to OTA messages.

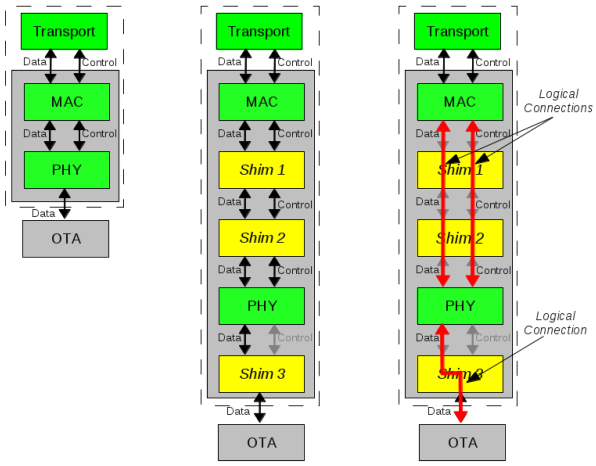


Figure 3.3: NEM Layer stack as it appears without Shims and with three Shims inserted. Two Shims reside between the MAC and PHY layers and one Shim resides between the PHY Layer and the OTA message interface.

It is possible to insert one or more Shims between the layers of a NEM stack or at the top and bottom of a layer stack. A Shim plugin implements the same generic interface as a MAC and PHY plugin. This

generic interface allows Shims to be inserted between components without requiring those components to have knowledge of their presence. Figure 3.3 shows what an NEM layer stack looks like after inserting Shim Layers and the resulting physical and logical connections. Listing 3.7 shows an IEEE 802.11 NEM definition with a Timing Analysis Shim inserted between the MAC and PHY Layers.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "http://configserver/dtd/nem.dtd">
<nem name="IEEE 802.11 NEM">
  <mac definition="ieee80211abgmac.xml"/>
  <shim definition="timinganalysisshim.xml"/>
  <phy definition="universalphy.xml"/>
  <transport definition="transvirtual.xml"/>
</nem>
```

Listing 3.7: IEEE 802.11abg NEM definition with a Timing Analysis Shim.

Chapter 4

Events

An emulation event is an opaquely distributed message which is delivered in realtime to one or more targeted EMANE components. Every EMANE component is capable of transmitting and receiving events.

4.1 Event Service

EMANE components which create events based on experiment scenarios are called Event Generators. Event Generators are instantiated and managed by the EMANE Event Service, which provides the interface used to publish events.

The Event Service instantiates one or more Event Generators based on an XML configuration file. The Event Service application is named `emaneeventservice` and the Event Service configuration file is referred to as the *Event Service XML*. The Event Service also requires a deployment configuration file. This file is referred to as the *Deployment XML* and details the contents of the NEM Platform Servers involved in the emulation experiment. Listing 4.1 shows a sample Event Service XML configuration. Listing 4.2 shows the deployment file resulting from the emulation experiment described in Section 2.1.2 on page 10. Listing 4.3 shows the manual entry for the `emaneeventservice` application.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventservice SYSTEM "file:///usr/share/emane/dtd/eventservice.dtd">
<eventservice name="Sample Event Service" deployment="deployment.xml">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>
  <generator name="Emulation Event Log Generator" definition="eelgenerator.xml"/>
  <generator name="Antenna Direction Generator" definition="antennadirectiongenerator.xml"/>
</eventservice>
```

Listing 4.1: Sample Event Service XML configuration loading two Event Generators.

No restriction is placed on how Event Generators create events. Some Event Generators may read precomputed state information from input files and publish events on time boundaries. Others may create event data algorithmically in realtime and publish the events based on update threshold logic. When new EMANE event types are introduced, no modifications are necessary to existing EMANE components, provided those components are not required to process the new events. Only the Event Generator and the destination EMANE components need to know the more specialized form of the generically transmitted event. Events are addressable using a three field tuple: NEM Platform Server identifier, NEM identifier and component identifier.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE deployment SYSTEM "file:///usr/share/emane/dtd/deployment.dtd">
<deployment>
  <platform id="1">
    <nem id="1"/>
  </platform>
  <platform id="2">
    <nem id="2"/>
  </platform>
  <platform id="3">
    <nem id="3"/>
  </platform>
  <platform id="4">
    <nem id="4"/>
  </platform>
</deployment>
```

Listing 4.2: Deployment XML resulting from the emulation experiment described in Section 2.1.2 on page 10.

```
emaneeventservice(1)                                    emaneeventservice(1)

NAME
       emaneeventservice - EMANE Event Service

SYNOPSIS
       emaneeventservice [OPTIONS]... CONFIG_URI

DESCRIPTION
       emaneeventservice  creates  and manages event generators. These genera-
       tors create events that are transmitted to targeted NEM components.

       CONFIG_URI is the XML containing the Event Service configuration.

OPTIONS
       The following options are supported:

       --version
              Display version and exit

       --loglevel [0,4]
              Set the current application log level.
              0 - No Logging
              1 - Abort Level
              2 - Error Level
              3 - Stat Level
              4 - Debug Level

       --realtime
              Run with realtime priority and SCHED_RR.   Must  have  superuser
              privilege.

       --starttime
              Set the start time HH:MM:SS (ex. 09:30:00)

       --nextday
              Set the start time to start the test on the next day, after mid-
              night

       --logserver DESTINATION:PORT
              Enable remote logging and direct all  logging  messages  to  the
              given endpoint.

       --logfile FILE
             Log to a file

       --daemonize
              Run EMANE in the background

       --syslog
              Log to syslogd
```

Listing 4.3: `emaneeventservice` man page entry.

Event Generator XML configuration mirrors the layout of the PHY Layer, MAC Layer, and Shim Layer XML described in Section 3.1 Defining an NEM on page 26 with two differences: Event Generators use the *eventgenerator.dtd* and the root XML element is `<eventgenerator>`. Listing 4.4 shows a sample Event Generator XML configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE eventgenerator SYSTEM "file:///usr/share/emane/dtd/eventgenerator.dtd">
<eventgenerator name="Emulation Event Log" library="eelgenerator">
  <param name="inputfile" value="/home/emane/demonstration/8/scenario.eel"/>
  <param name="loader" value="location:eelloaderlocation:full"/>
  <param name="loader" value="pathLoss:eelloaderpathloss:full"/>
  <param name="loader" value="antennadirection:eelloaderantennadirection:full"/>
</eventgenerator>
```

Listing 4.4: Sample Event Generator XML configuration.

4.1.1 Event Service Configuration Parameters

4.1.1.1 `eventservicegroup`

The Event Service Channel mulitcast endpoint used to communicate events between EMANE components in an EMANE deployment.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicegroup" value="224.1.2.8:45703">`

Parameter value format description:
<IPv4 Multicast Group>:<Port> | <IPv6 Multicast Group>/<Port>

Name	Description
IPv4 or IPv6 Multicast Address	Event Service Channel multicast group
Port	Event Service Channel UDP port

4.1.1.2 `eventservicedevice`

The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicedevice" value="eth0"/>`

4.2 Event Daemon

An Event Agent is an EMANE component that translates events from their emulation domain representation to a format usable by application domain entities. They facilitate the reuse of any experiment scenario information propagated via an event that is of interest outside of the emulation domain. For example, position information contained in the EMANE Location Event which is used by some PHY Layers to compute path loss may also be of interest to application domain entities that require GPS location. Event Agents are instantiated and managed by an Event Daemon.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventdaemon SYSTEM "file:///usr/share/emane/dtd/eventdaemon.dtd">
<eventdaemon name="EMANE Event Daemon 1" nemid="1">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>
  <agent definition="gpsdlocationagent.xml"/>
</eventdaemon>
```

Listing 4.5: Sample Event Daemon XML configuration loading one Event Agent.

Event Daemons instantiate one or more Event Agents based on an XML configuration file. The Event Daemon application is named `emaneeventd` and the Event Daemon configuration file is referred to as the *Event Daemon XML*. Listing 4.5 shows a sample Event Daemon XML configuration. Listing 4.6 shows the man page entry for the `emaneeventd` application.

```
emaneeventd(1)                                                    emaneeventd(1)

NAME
       emaneeventd - EMANE event daemon

SYNOPSIS
       emaneeventd [OPTIONS]... CONFIG_URI

DESCRIPTON
       emaneeventd  application  is  a  container application that creates and
       manages event agents. These agents register to receive events  and  act
       upon these events to allow external applications to use the event data.
       One example is the  gpsdlocationagent.  This  agent  receives  location
       events and translates those events to NMEA stings which it communicates
       to gpsd by creating a pseudo terminal and acting like a GPS receiver.

       CONFIG_URI is the XML containing the event daemon configuration.

OPTIONS
       The following options are supported:

       --version
              Display version and exit

      --loglevel [0,4]
              Set the current application log level.
              0 - No Logging
              1 - Abort Level
              2 - Error Level
              3 - Stat Level
              4 - Debug Level

       --logserver DESTINATION:PORT
              Enable remote logging and direct all  logging  messages  to  the
              given endpoint.

       --realtime
              Run  with  realtime  priority and SCHED_RR.  Must have superuser
              privilege.

       --logfile FILE
              Log to a file
```

```
--daemonize
       Run EMANE in the background

--syslog
       Log to syslogd
```

Listing 4.6: `emaneeventd` man page entry.

Event Agent XML configuration mirrors the layout of the PHY Layer, MAC Layer, and Shim Layer XML described in Section 3.1 Defining an NEM on page 26 with two differences: Event Agents use the *eventagent.dtd* and the root XML element is `<eventagent>`. Listing 4.7 shows a sample Event Agent XML configuration.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventagent SYSTEM "file:///usr/share/emane/dtd/eventagent.dtd">
<eventagent name="gpsdlocationagent" library="gpsdlocationagent">
  <param name="gpsdconnectionenabled" value="no"/>
  <param name="pseudoterminalfile" value="/var/tmp/gps.pty"/>
</eventagent>
```

Listing 4.7: Sample Event Agent XML configuration.

4.2.1 Event Daemon Configuration Parameters

4.2.1.1 `eventservicegroup`

The Event Service Channel mulitcast endpoint used to communicate events between EMANE components in an EMANE deployment.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicegroup" value="224.1.2.8:45703">`

Parameter value format description:

<IPv4 Multicast Group>:<Port> | <IPv6 Multicast Group>/<Port>

Name	Description
IPv4 or IPv6 Multicast Address	Event Service Channel multicast group
Port	Event Service Channel UDP port

4.2.1.2 `eventservicedevice`

The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="eventservicedevice" value="eth0"/>`

4.3 Event Types

The standard EMANE distribution contains four types of events:

- Pathloss Event
- Location Event
- Comm Effect Event
- Antenna Direction Event

4.3.1 Pathloss Event

A pathloss event contains a variable length list of pathloss entries that are unique to each targeted NEM. A pathloss entry consists of a transmitter NEM Id and the pathloss to/from the transmitter NEM with respect to the targeted NEM. EMANE components that are capable of processing pathloss events maintain a table of transmitter NEM Ids and pathloss entry data, updating their respective entries as events are received.

The standard EMANE distribution contains one component which processes pathloss events: the *Universal PHY Layer* (See Chapter 7 Universal PHY Layer on page 59). The Universal PHY Layer can be configured to use pathloss to calculate the *rxPower* associated with a received over-the-air packet.

The standard EMANE distribution contains two event generators that produce pathloss events: *Mitre Mobility Model Event Generator* and *Emulation Event Log Generator* (See Chapter 13 Mitre Mobility Model Event Generator on page 127 and Chapter 15 Emulation Event Log Generator on page 137). Pathloss events can also be generated using libemaneeventservice or its corresponding Python bindings. The number of pathloss entries contained in each Pathloss Event depends on the generator type and/or configuration.

The Loss Controller GUI application can be used to create and playback Mitre Mobility Model Event Generator based pathloss scenarios. The Loss Controller GUI is introduced in Section 12.4.1 Demonstration 13 on page 121.

4.3.2 Location Event

A location event contains a viable length list of location entries that update the GPS location of one or more NEMs. A location entry consists of an NEM Id and the latitude, longitude, and altitude associated with that NEM. EMANE components that are capable of processing location events maintain a table of NEM Ids and location entry data, updating their respective entries as events are received.

The standard EMANE distribution contains two components which process location events: *Universal PHY Layer* and *GPSd Location Agent* (See Chapter 7 Universal PHY Layer on page 59 and Chapter 18 GPSd Location Agent on page 151). The Universal PHY Layer can be configured to use location to calculate the pathloss between two NEMs base on either a 2-ray flat earth or free space propagation model. The Universal PHY Layer also requires location for directional antenna support. The GPSd Location Agent uses location events to emulate an attached GPS receiver.

The standard EMANE distribution contains three event generators that produce location events: *Mitre Mobility Event Generator*, *Emulation Event Log Generator*, and *Emulation Script Generator* (See Chapter 13 Mitre Mobility Model Event Generator on page 127, Chapter 14 Emulation Script Event Generator on page 133, and Chapter 15 Emulation Event Log Generator on page 137). Location events can also be generated using libemaneeventservice or its corresponding Python bindings. The number of location entries contained in each Location Event depends on the generator type and/or configuration.

4.3.3 Comm Effect Event

A Comm Effect event contains a variable length list of communication effect entries that are unique to each targeted NEM. A communication effect entry consists of a transmitter NEM Id and the latency, jitter, loss, duplication rate, unicast bitrate, and broadcast bitrate associated with packets received from the respective transmitter NEM to the target NEM. EMANE components that are capable of processing Comm Effect events maintain a table of transmitter NEM Ids and communication effect entry data, updating their respective entries as events are received.

The standard EMANE distribution contains one component that processes Comm Effect events: *Comm Effect Shim Layer* (See Chapter 10 Comm Effect Shim Layer on page 97).

The standard EMANE distribution contains one event generator that produces Comm Effect events: *Comm Effect Generator*. Comm Effect events can also be generated using libemaneeventservice or its corresponding Python bindings.

The Comm Effect Controller GUI application can be used to create and playback Comm Effect Generator based Comm Effect scenarios.

4.3.4 Antenna Direction Event

An antenna direction event contains a variable length list of antenna direction entries that update the antenna direction of one or more NEMs. An antenna direction entry consists of an NEM Id and the antenna elevation, azimuth, beam width elevation, and beam width azimuth associated with that NEM.

The standard EMANE distribution contains one component that processes antenna events: *Universal PHY Layer* (See Chapter 7 Universal PHY Layer on page 59). The Universal PHY Layer can be configured to use antenna pointing information as part of over-the-air packet processing.

The standard EMANE distribution contains two event generators that produces antenna direction events: *Antenna Direction Generator* and *Emulation Event Log Generator* (See Chapter 17 Antenna Direction Event Generator on page 147 and Chapter 15 Emulation Event Log Generator on page 137). Antenna direction events can also be generated using libemaneeventservice or its corresponding Python bindings.

4.4 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

4.4.1 Demonstration 3

This demonstration deploys a ten node distributed IEEE 802.11abg NEM emulation experiment illustrated in Figure 4.1. The goal of this demonstration is to become familiar with the basic EMANE components involved in producing and consuming events.

4.4.1.1 Demonstration Procedure

1. Review the Demonstration 3 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/3
[emane@emanedemo 3] less platform*.xml
```

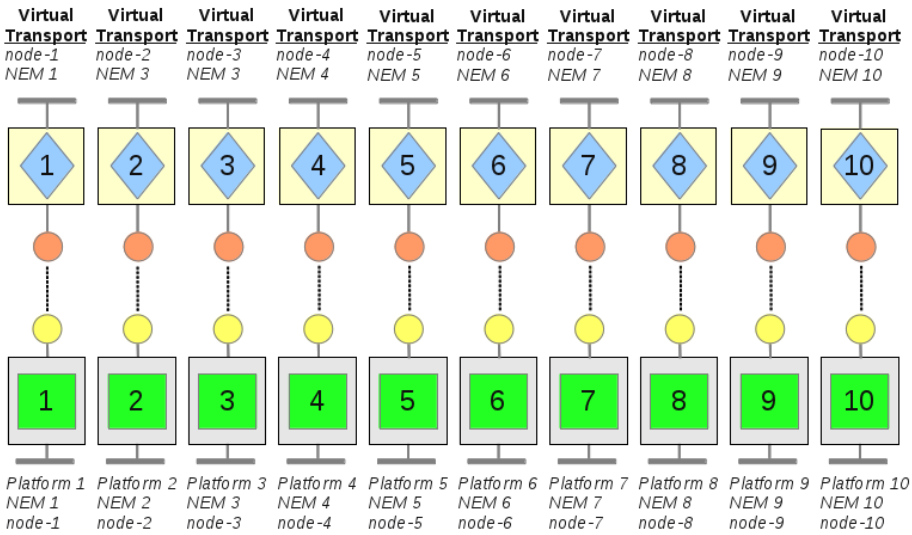
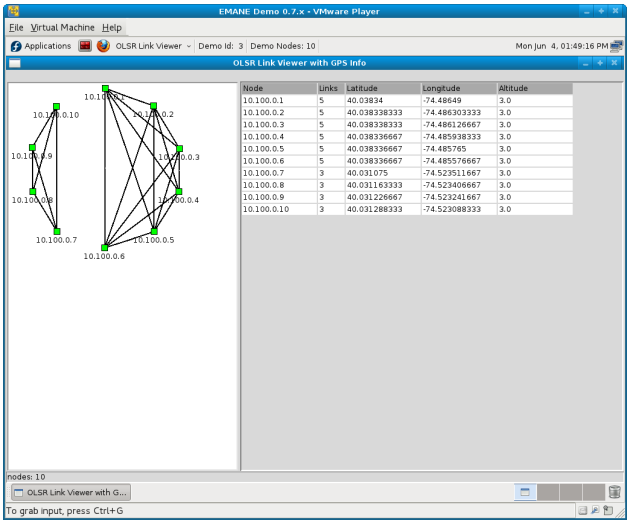


Figure 4.1: Demonstration 3 - Ten node distributed IEEE 802.11abg NEM deployment.

2. Deploy the demonstration.

```
[emane@emanedemo 3]$ sudo ./lxc-demo-start.sh
```

3. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher. The mobility scenario will begin in 30 seconds.



4. Connect to virtual `node-1`.

```
[emane@emanedemo 3]$ ssh node-1
```

5. Review the running processes.

```
[emane@node-1 ~]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S+     0:00 /usr/lib/lxc/lxc-init -- /tmp/lxc-node/3/1/init.sh -s
    6 ?        Sl     0:02 emanetransportd -r -d /home/emane/demonstration/3/transp
   11 ?        Sl     0:07 emane /home/emane/demonstration/3/platform1.xml -r -d -l
```

```
  24 ?        Sl     0:00 emaneeventd -d /home/emane/demonstration/3/eventdaemon1.
  32 ?        S<s    0:00 gpsd -n -b /dev/pts/0
  37 ?        Ssl    0:10 /usr/sbin/olsrd -f /home/emane/demonstration/3/routing1.
  40 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/tmp/lxc-node/3/1/run/sshd.
  41 ?        S+     0:00 /usr/local/bin/mgen input /home/emane/demonstration/3/mg
  43 ?        S      0:00 sshd: emane [priv]
  45 ?        S      0:00 sshd: emane@pts/1
  46 pts/1    Ss     0:00 -bash
 106 pts/1    R+     0:00 ps ax
```

6. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

7. Stop the demonstration.

```
[emane@emanedemo 3]$ sudo ./lxc-demo-stop.sh
```

4.4.1.2 Concept Review

1. What are the EMANE component event consumers in this demonstration and where are they hosted?
2. What is the difference between the Event Service and the Event Daemon?

Chapter 5

XML Configuration

5.1 Layered Configuration

EMANE uses a generic XML configuration design. All EMANE components are capable of specifying any number of configuration parameters using a generic syntax. These parameter value pairs are made accessible to their respective components via a configuration API that removes the need for component developers to process XML.

EMANE XML is layered to allow tailoring of lower levels of configuration to simplify deployment and promote reuse. Complex EMANE components are created by combining XML definitions. For example, an NEM is simply the combing of a MAC XML definition, PHY XML definition, and transport XML definition.

Configuration parameters do not need to be present in the XML if they are not required. Figure 5.1 depicts the XML configuration hierarchy for each of the EMANE container applications.

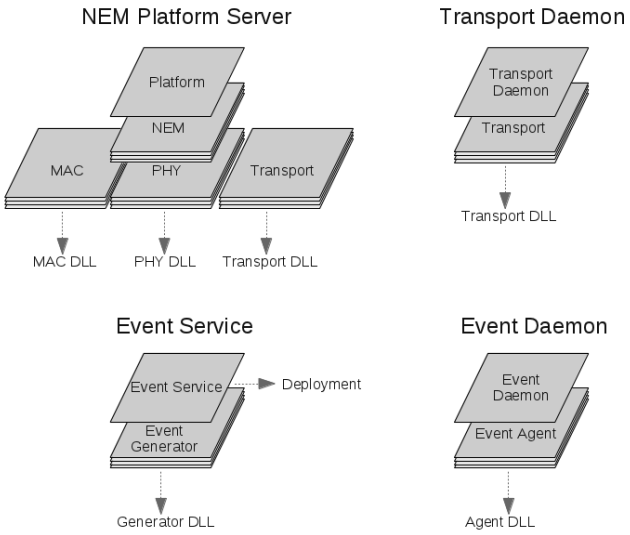


Figure 5.1: EMANE XML Hierarchy. Layers shaded in gray represent XML files.

Listings 5.1, 5.2 and 5.3 illustrate the EMANE layered XML concept. The NEM Platform Server XML shown in Listing 5.1 deploys two RF Pipe NEMs. In this example, the lowest level XML configuration in the hierarchy is the PHY Layer configuration contained in *universalphy.xml* (Listing 5.3). Here the `frequency` parameter is set to 2347 MHz. The NEM defined in *rfpipenem.xml* (Listing 5.2) uses the PHY Layer defined in *universalphy.xml* but overrides the `frequency` parameter, setting it to 3340 MHz. The NEM Platform Server definition contained in *platform.xml* instantiates two NEMs. NEM 1 overrides the `frequency` parameter value contained in *rfpipenem.xml* and sets NEM 1's `frequency` to 3000 MHz. NEM 2 uses the `frequency` value set in *rfpipenem.xml*, 3340 MHz. In this example the frequency value 2347 MHz contained in *universalphy.xml* is never used.

```
<platform name="Platform 1" id="1">
  <param name="otamanagerchannelenable" value="off"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="lo"/>

  <nem name="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8181"/>
    <param name="transportendpoint" value="localhost:8171"/>
    <phy definition="rfpipephy.xml">
      <param name="frequency" value="3000000"/>
    </phy>
    <transport definition="transraw.xml">
      <param name="device" value="eth1"/>
    </transport>
  </nem>

  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8182"/>
    <param name="transportendpoint" value="localhost:8172"/>
    <transport definition="transraw.xml">
      <param name="device" value="eth2"/>
    </transport>
  </nem>
</platform>
```

Listing 5.1: *platform.xml*: NEM Platform Server XML overriding NEM 1 `frequency` parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/local/emane/dtd/nem.dtd">
<nem name="RF-PIPE NEM">
  <mac definition="rfpipemac.xml"/>
  <phy definition="universalphy.xml">
    <param name="subid" value="2"/>
    <param name="frequency" value="3340000"/>
  </phy>
  <transport definition="transvirtual.xml"/>
</nem>
```

Listing 5.2: *rfpipenem.xml*: NEM XML overriding Universal PHY Layer `frequency` parameter.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE phy SYSTEM "file:///usr/local/emane/dtd/phy.dtd">
<phy name="universalphy" library="universalphylayer">
    <param name="bandwidth"                  value="1000"/>
    <param name="antennagain"                value="0.0"/>
    <param name="systemnoisefigure"          value="4.0"/>
    <param name="frequencyofinterest"        value="2347000"/>
    <param name="pathlossmode"               value="pathloss"/>
    <param name="noiseprocessingmode"        value="off"/>
    <param name="defaultconnectivitymode"    value="on"/>
    <param name="txpower"                    value="0.0"/>
    <param name="frequency"                  value="2347000"/>
    <param name="antennaazimuthbeamwidth"    value="360.0"/>
    <param name="antennaelevationbeamwidth"  value="180.0"/>
```

```
    <param name="antennaazimuth"            value="0.0"/>
    <param name="antennaelevation"          value="0.0"/>
    <param name="antennatype"               value="omnidirectional"/>
</phy>
```

Listing 5.3: *universalphy.xml*: Universal PHY Layer XML.

5.2 Automatic XML Generation

The standard EMANE distribution provides two tools that can be used to automatically generate Transport Daemon XML and deployment XML: `emanegentransportxml` and `emanegendeploymentxml`, respectively. Listing 5.4 shows the man page entry for `emanegentransportxml`. Listing 5.5 shows the man page entry for `emanegendeploymentxml`.

```
emanegentransportxml(1)                                emanegentransportxml(1)

NAME
       emanegentransportxml - EMANE transport daemon XML generator

SYNOPSIS
       emanegentransportxml [OPTIONS]... URI

DESCRIPTON
       Application  generates  XML configuration file(s) for virtual interface
       network entry point(s). Each EMANE platform contains one or  more  Net-
       work  Emulation  Modules (NEMs). In order to gain access to each of the
       NEMs, a virtual interface is configured to be brought up (either within
       the  platform  configuration  file or the nem configuration file). This
       application scans through the platform and nem configuration files  and
       extracts  interface-specific information into separate files. The files
       can then be used as configuration input into the emanetund  application
       to create an NEM-to-emanetund pair, establishing a 'tunnel' for network
       traffic.

OPTIONS
       The following options are supported:

       URI    URI of the XML containing the NEM configuration  for  the  plat-
              form.

       --outpath LOCATION
              Specifies output location for generated xml files
              Default: current working directory

       --dtdpath LOCATION
              Specifies the location of the directory containing DTDs
              Default: directory in the SYSTEM declaration in platform URI

       --dtd DTD
              Specifies the DTD file to validate against
              Default: transport.dtd

       --debug
              Enable debug output

       --help Print usage information and exit.
```

Listing 5.4: `emanegentransportxml` man page entry.

```
manegendeploymentxml(1)                                emanegendeploymentxml(1)

NAME
       emanegendeploymentxml - EMANE deployment XML generator

SYNOPSIS
       emanegendeploymentxml [OPTIONS]... URIs

DESCRIPTON
       Application  scans  through  the  supplied  platform  configuration XML
       file(s) to generate a single XML file which maps Network Emulation Mod-
       ules  (NEMs)  to their corresponding platform. The file is then used by
       emane for inter-platform network addressing.

OPTIONS
       The following options are supported:

       URIs   URI of one (or more) XML files containing the NEM  configuration
              for the platform.

       --inpath LOCATION
              Specifies the location containing XML with NEM configuration for
              the platform which uses the platformPID.xml  format  (where  PID
              corresponds  to a unique platform identifier). NOTE: This option
              assumes that NO files (URIs) were specified on the command line.
              The two are mutually exclusive.

       --outpath LOCATION
              Specifies output location for generated deployment XML file.
              Default: current working directory

       --dtdpath LOCATION
              Specifies the location of the directory containing DTDs
              Default: directory in the SYSTEM declaration in platform URI

       --dtd DTD
              Specifies the DTD file to validate against
              Default: deployment.dtd

       --debug
              Enable debug output

       --help Print usage information and exit.
```

Listing 5.5: `emanegendeploymentxml` man page entry.

5.3 Transport Grouping

The `emanegentransportxml` application will use the Platform XML `transport` element's optional `group` attribute, if present, to group transports with matching values into a single Transport Daemon XML configuration. The Platform XML in Listing 5.6 groups all the transport instances into the single Transport Daemon XML configuration shown in Listing 5.7.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
<platform name="Platform 1" id="1">
  <param name="otamanagerchannelenable" value="off"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="lo"/>

  <nem name="NEM-1" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8201"/>
    <param name="transportendpoint" value="localhost:8301"/>
    <transport definition="transraw.xml" group="alpha">
      <param name="device" value="veth1.1"/>
    </transport>
  </nem>
```

```
  <nem name="NEM-2" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8202"/>
    <param name="transportendpoint" value="localhost:8302"/>
    <transport definition="transraw.xml" group="alpha">
      <param name="device" value="veth2.1"/>
    </transport>
  </nem>

  <nem name="NEM-3" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8203"/>
    <param name="transportendpoint" value="localhost:8303"/>
    <transport definition="transraw.xml" group="alpha">
      <param name="device" value="veth3.1"/>
    </transport>
  </nem>

  <nem name="NEM-4" id="4" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8204"/>
    <param name="transportendpoint" value="localhost:8304"/>
    <transport definition="transraw.xml" group="alpha">
      <param name="device" value="veth4.1"/>
    </transport>
  </nem>
</platform>
```

Listing 5.6: Platform XML using Transport Grouping.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
<transportdaemon>
  <instance nemid="1">
    <param name="transportendpoint" value="localhost:8301"/>
    <param name="platformendpoint" value="localhost:8201"/>
    <transport definition="transraw.xml">
      <param name="device" value="veth1.1"/>
    </transport>
  </instance>
  <instance nemid="2">
    <param name="transportendpoint" value="localhost:8302"/>
    <param name="platformendpoint" value="localhost:8202"/>
    <transport definition="transraw.xml">
      <param name="device" value="veth2.1"/>
    </transport>
  </instance>
  <instance nemid="3">
    <param name="transportendpoint" value="localhost:8303"/>
    <param name="platformendpoint" value="localhost:8203"/>
    <transport definition="transraw.xml">
      <param name="device" value="veth3.1"/>
    </transport>
  </instance>
  <instance nemid="4">
    <param name="transportendpoint" value="localhost:8304"/>
    <param name="platformendpoint" value="localhost:8204"/>
    <transport definition="transraw.xml">
      <param name="device" value="veth4.1"/>
    </transport>
  </instance>
</transportdaemon>
```

Listing 5.7: Transport Daemon XML resulting from the Platform XML in Listing 5.6.

5.4 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

5.4.1 Demonstration 4

This demonstration deploys a ten node hybrid RF-Pipe NEM emulation experiment illustrated in Figure 5.3. The goal of this demonstration is to become familiar with the EMANE XML Hierarchy.

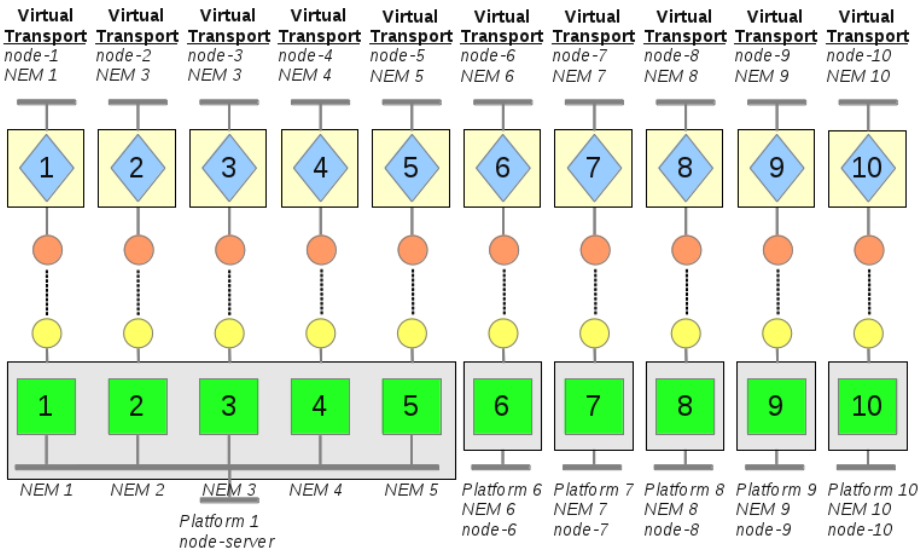


Figure 5.2: Demonstration 4 - Ten node hybrid RF Pipe NEM deployment.

5.4.1.1 Demonstration Procedure

1. Review the Demonstration 4 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/4
[emane@emanedemo 4] less platform{,[6-9],10}.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 4]$ sudo ./lxc-demo-start.sh
```

3. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

4. Visually verify that the network has formed *all-informed*. There should be 10 nodes with 9 routes each.

5. Stop the demonstration.

```
[emane@emanedemo 4]$ sudo ./lxc-demo-stop.sh
```

6. Modify the platform XML file *platform.xml* using your favorite editor. Change Lines 10, 19, 29, 38 and 47 to use *rfpipenem2.xml*. The below stream editor command can be used as a shortcut:

```
[emane@emanedemo 4]$ sed -i -e s/rfpipenem/rfpipenem2/ platform.xml
```

7. Rebuild the deployment files.

```
[emane@emanedemo 4]$ make
```

8. Compare *rfpipenem.xml* and *rfpipenem2.xml* using your favorite editor.

```
[emane@emanedemo 4]$ meld rfpipenem.xml rfpipenem2.xml
```

9. Redeploy the demonstration.

```
[emane@emanedemo 4]$ sudo ./lxc-demo-start.sh
```

10. Visually verify that 2 distinct networks have formed. There should be 10 nodes with 4 routes each.
11. Stop the demonstration.

```
[emane@emanedemo 4]$ sudo ./lxc-demo-stop.sh
```

5.4.1.2 Concept Review

1. If the same configuration parameter appears in an NEM definition in the platform XML and in that NEM's NEM XML configuration file which value is used?
2. Why did two distinct networks form after the *platform.xml* file was modified and redeployed?

5.4.2 Demonstration 5

This demonstration generates the XML configuration necessary to deploy a ten node hybrid RF-Pipe NEM emulation experiment illustrated in Figure 5.3. The goal of this demonstration is to become familiar with the EMANE automatic XML generation tools `emanegentransportxml` and `emanegendeploymentxml`.

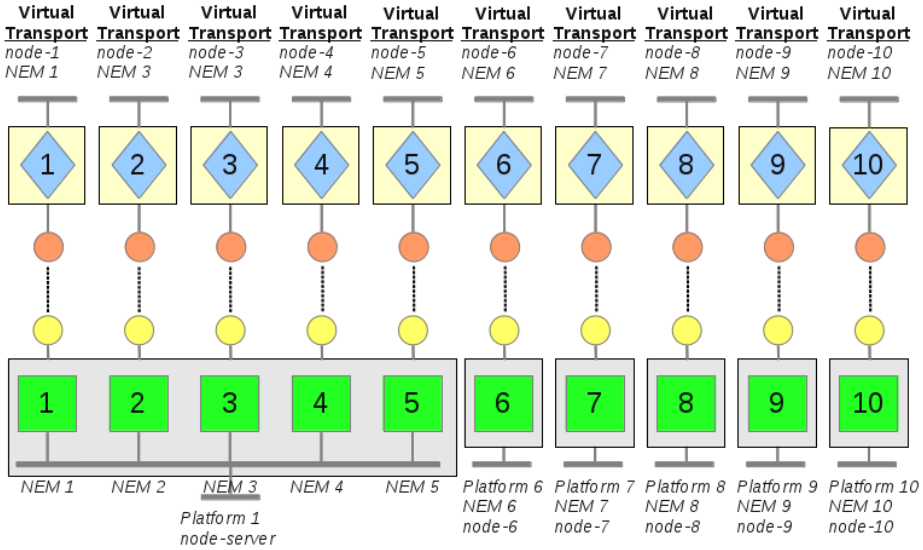


Figure 5.3: Demonstration 5 - Ten node hybrid RF Pipe NEM deployment.

5.4.2.1 Demonstration Procedure

1. Review the Demonstration 5 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/5
[emane@emanedemo 5] less platform{,[6-9],10}.xml
```

2. Generate the Transport Daemon XML configuration.

```
[emane@emanedemo 5]$ emanegentransportxml platform*.xml
```

3. Generate the Deployment XML configuration.

```
[emane@emanedemo 5]$ emanegendeploymentxml platform*.xml
```

5.4.2.2 Concept Review

1. Which EMANE application processes Platform XML?
2. Which EMANE application processes Deployment XML?

Chapter 6

Deployment Debugging

6.1 NEM Platform Server Debug Port

The NEM Platform Server can be configured to enable an external debug port using the `debugportenable` parameter. The debug port is accessible via any telent client and provides an interface to retrieve layer stack and statistic information for all NEMs contained in a platform. The default debug port TCP port is `47000` and can be modified using the `debugport` parameter. See Section 2.1.3 NEM Platform Server Configuration Parameters on page 11 for more information.

The NEM Platform Server currently supports eight commands:

`clear`	Clears the screen
`exit`	Terminates the debug port session
`help`	Lists supported command and displays command specifics
`stats`	Queries and displays the statistics of one or more of the NEMs contained in the platform
`clearstats`	Clears the statistics of one or more of the NEMs contained in the platform
`showstacks`	Displays the NEM Layer stack information of one or more the NEMs contained in the platform
`quit`	Terminates the debug port session
`version`	Displays the EMANE version

6.2 Logging

All four EMANE container applications support logging: `emane`, `emanetransportd`, `emaneservice` and `emaneeventd`. EMANE supports five log levels and logs can be directed to *stdout*, *file*, *syslog* or an ACE Log Server depending on the options used when invoking the container application. Table 6.1 lists the available log levels along with their `syslog` equivalent.

Table 6.1: EMANE Log Levels with `syslog` mapping.

Name	Description	`syslog` mapping	Option Value
`NOLOG_LEVEL`	No Logging		0
`ABORT_LEVEL`	Unrecoverable failure detected	`LOG_USER|LOG_CRIT`	1
`ERROR_LEVEL`	Recoverable failure notification	`LOG_USER|LOG_ERR`	2
`STATISTIC_LEVEL`	Statistic out message	`LOG_USER|LOG_INFO`	3
`DEBUG_LEVEL`	General verbose debugging	`LOG_USER|LOG_DEBUG`	4

To enable logging to *stdout* specify a log level greater than 0:

```
[emane@emanedemo 6] emane -l 4 platform.xml
```

To enable logging to a file specify a log level greater than 0 and use the `--logfile` option to specify the file:

```
[emane@emanedemo 6] emane -l 4 --logfile /tmp/emane.log platform.xml
```

To enable logging to `syslog` specify a log level greater than 0 and use the `--syslog` option:

```
[emane@emanedemo 6] emane -l 4 --syslog platform.xml
```

By default `syslog` does not output messages at `LOG_DEBUG`. In order to view these messages a rule must be added to the end of */etc/rsyslog.conf* and the service restarted:

```
if $programname == 'emane' and $syslogseverity == '7' then /var/log/messages
```

To enable logging to an ACE Log Server specify a log level greater than 0 and use the `--logserver` option to specify the endpoint:

```
[emane@emanedemo 6] emane -l 4 --logserver node-server:8089 platform.xml
```

6.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

6.3.1 Demonstration 6

This demonstration deploys a ten node centralized IEEE 802.11abg NEM emulation experiment illustrated in Figure 6.1. The goal of this demonstration is to become familiar with the NEM Platform Server Debug Port.

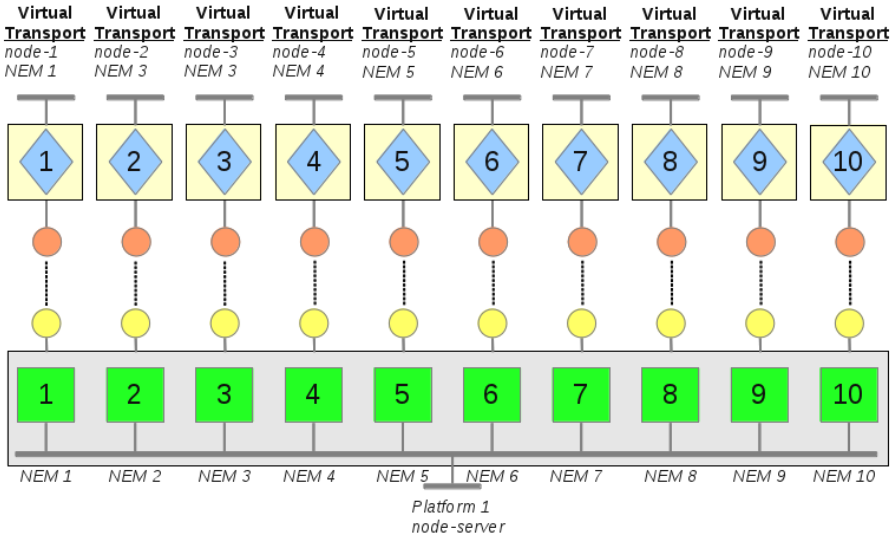


Figure 6.1: Demonstration 6 - Ten node centralized IEEE 802.11abg NEM deployment.

6.3.1.1 Demonstration Procedure

1. Review the Demonstration 6 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/6
[emane@emanedemo 6] less platform.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 6]$ sudo ./lxc-demo-start.sh
```

3. Connect to the NEM Platform Server Debug Port and display the command list.

```
[emane@emanedemo 6]$ telnet node-server 47000
Trying 10.99.0.100...
Connected to node-server.
Escape character is '^]'.
debugport %% help

Form more information: `help <command>`

command list:
 clear
 exit
 help
 stats
 clearstats
 showstacks
 quit
 version

debugport %%
```

4. View the detailed help for `showstacks` and `stats`.

```
debugport %% help showstacks

command    : showstacks
description: Show NEM stack
usage      : showstacks [nem]

debugport %% help stats

command    : stats
description: Show statistics
usage      : stats [all|nem|layer|index|stat]
```

5. Query NEM stack configurations and component statistics.

```
debugport %% showstacks 1
NEM             1
LAYER           INDEX
mac             0
phy             1

LAYER           INDEX
mac             0
phy             1

debugport %% stats index 1 1
Index                           1
Layer Type                      phy
numDownstreamPacketFromMAC      485
numDownstreamPacketToOTA        485
numUpstreamDiscardDueToFoi      0
numUpstreamDiscardDueToFreqBandwidth 0
numUpstreamDiscardDueToInvalidSubId 0
numUpstreamDiscardDueToNoPathLossInfo 0
numUpstreamDiscardDueToRegistrationId 0
```

```
numUpstreamDiscardDueToSinr    1590
numUpstreamDiscardDueToSubIdMismatch 0
numUpstreamPacketFromOTA       4317
numUpstreamPacketToMAC         2727
processedDownstreamControl     0
processedDownstreamPackets     485
processedEvents                924
processedUpstreamPackets       4317
```

6. Quit the telnet session and repeat the same query using netcat.

```
debugport %% quit
```

```
[emane@emanedemo 6]$ echo "stats index 1 1" | nc localhost 47000
```

7. Stop the demonstration.

```
[emane@emanedemo 6]$ sudo ./lxc-demo-stop.sh
```

6.3.1.2 Concept Review

1. What does the index refer to in respect to NEM Platform Server Debug Port commands?
2. Is it possible to script NEM Platform Server Debug Port commands?

6.3.2 Demonstration 7

This demonstration deploys a ten node centralized IEEE 802.11abg NEM emulation experiment illustrated in Figure 6.2. The goal of this demonstration is to become familiar with EMANE logging mechanisms.

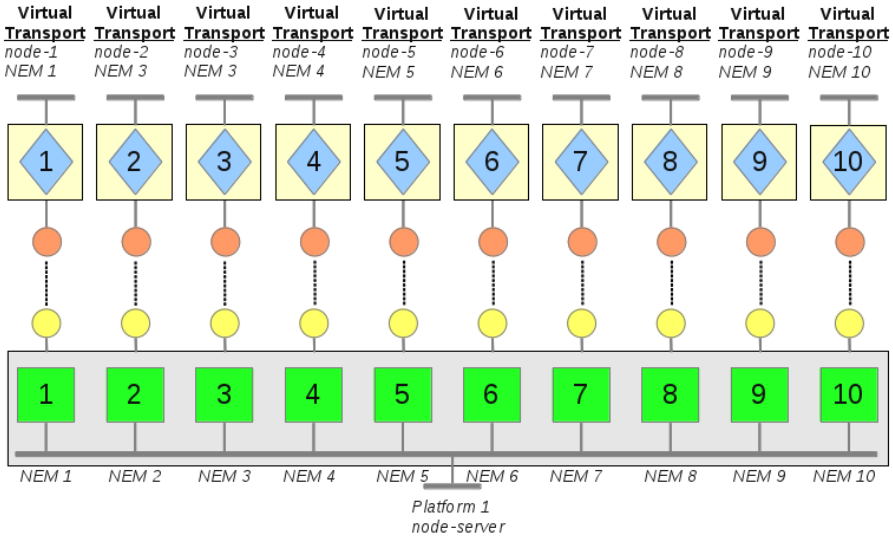


Figure 6.2: Demonstration 7 - Ten node centralized IEEE 802.11abg NEM deployment.

6.3.2.1 Demonstration Procedure

1. Review the Demonstration 7 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/7
[emane@emanedemo 7] less platform.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 7]$ sudo ./lxc-demo-start.sh
```

3. Shutdown the NEM Platform Server.

```
[emane@emanedemo 7]$ sudo killall -QUIT emane
```

4. Restart the NEM Platform Server with logging.

```
[emane@emanedemo 7]$ emane -l 4 platform.xml
```

5. Stop the demonstration.

```
[emane@emanedemo 6]$ sudo ./lxc-demo-stop.sh
```

6.3.2.2 Concept Review

1. What types of EMANE logging are permitted when daemonizing the container application?
2. What should be the maximum log level used during an emulation experiment? Why?

Part II

Models

Chapter 7

Universal PHY Layer

The Universal PHY Layer provides a common PHY implementation for the various MAC Layers supplied as part of the standard EMANE distribution. Its use is not mandatory but is encouraged for authors of other proprietary and non-proprietary MAC implementations as it provides a set of core functionality required by most wireless Network Emulation Modules. The key functionality includes the following:

- Pathloss Calculation
- Receive Power Calculation
- Directional Sector Antenna Support
- Noise Processing
- MAC-PHY Control Messaging

7.1 Model Features

7.1.1 Pathloss Calculation

Pathloss within the Universal PHY Layer is based on location or pathloss events. Pathloss is dynamically calculated based on location events when the `pathlossmode` configuration parameter is set to either `2ray` or `freespace`, which selects between the 2-ray flat earth or free space propagation models, respectively. Pathloss can be provided in realtime based on external propagation calculations using pathloss events. The `pathlossmode` configuration parameter should be set to `pathloss` in order to process inbound pathloss events.

7.1.2 Receive Power Calculation

For each received packet, the Universal PHY Layer computes the receiver power associated with that packet using the following calculation:

$$rxPower = txPower + txAntennaGain + rxAntennaGain - pathloss$$

Where,

txPower	Packet Common PHY Header transmitter power (See Section 3.2)
txAntennaGain	Packet Common PHY Header transmitter antenna gain (See Section 3.2)
rxAntennaGain	Configuration parameter `antennagain` (Section 7.2.2)
pathloss	Pathloss between transmitter and receiver determined based on `pathlossmode` configuration parameter (See Section 7.2.5)

If the *rxPower* is less than the *rxSensitivity*, the packet is silently discarded.

$$rxSensitivity = -174 + noiseFigure + 10\log(bandWidth)$$

Where,

bandWidth	Configuration parameter `bandwidth` (Section 7.2.1)
noiseFigure	Configuration parameter `noisefigure` (Section 7.2.3)

7.1.3 Directional Sector Antenna Support

The Universal PHY Layer provides support for directional antenna, if required. This support includes the ability to statically configure the directional antenna parameters (pointing and profile) as well as the ability to accept parameters from the MAC Layer via a control message on a per packet basis and via antenna pointing events. The Universal PHY Layer utilizes location events and Tx and Rx antenna information to determine if two nodes are visible. Current directional antenna support is based on sector antennas, where a sector is defined by antenna azimuth and elevation beam width. Any intersection between the transmitting and receiving antenna will apply full gain. Figure 7.1 illustrates a transmitter and receiver with and without beam overlap.

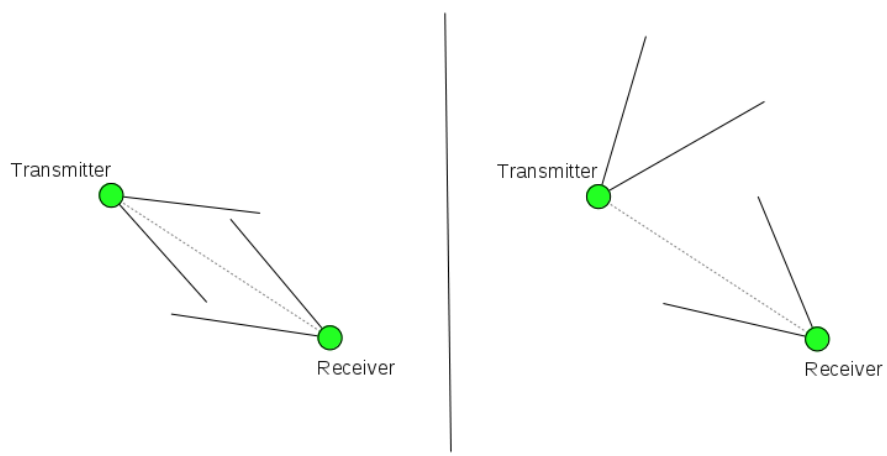


Figure 7.1: Transmitter and receiver antenna beam overlap *(Left)*. Transmitter and receiver no antenna beam overlap *(Right)*.

Figure 7.2 visualizes the four configuration items that are necessary to define the pointing and profile characteristics of a directional antenna: `antennaazimuth`, `antennaelevation`, `antennaazimuthbeamwidth` and `antennaelevation-beamwidth`.

7.1.4 Noise Processing

The Universal PHY Layer provides the ability to assess the impact of intentional and unintentional noise sources within the emulation by adjusting the noise floor. This is achieved by summing the energy of interferers within the appropriate frequency of interest over a given time interval and adjusting the noise floor accordingly when a valid packet is received. The Universal PHY Layer only computes interference for out-of-band packets. An out-of-band packet is one which is not from the same emulated waveform. The Universal PHY Layer determines waveform type by comparing the PHY Registration Id, center frequency,

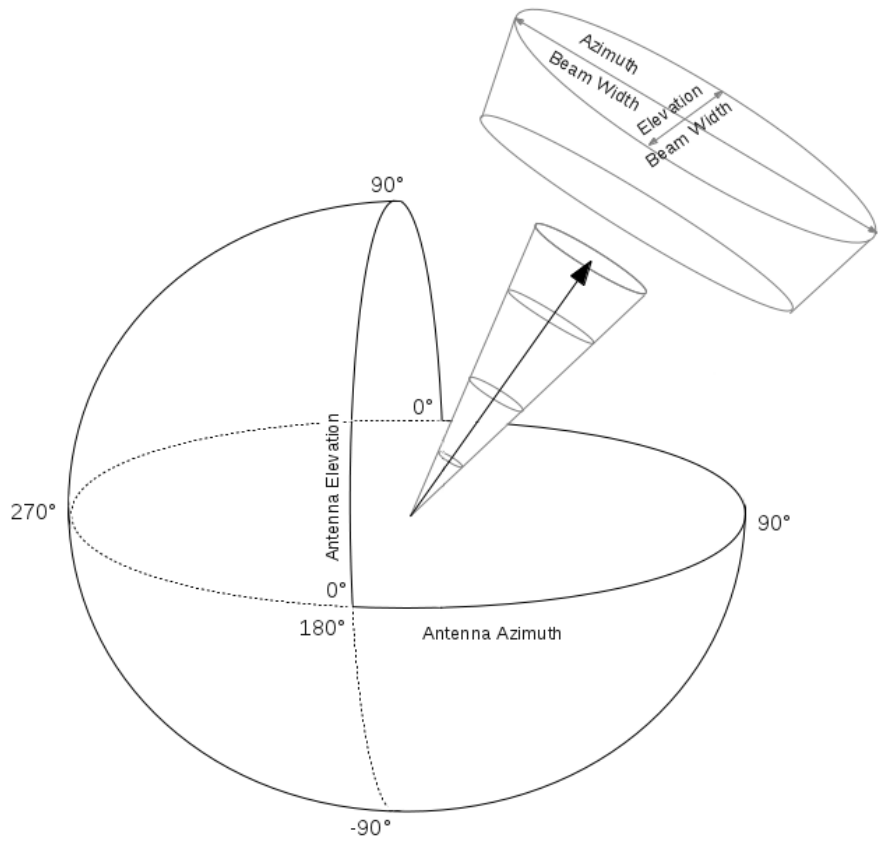


Figure 7.2: Universal PHY Layer Directional Antenna configuration.

and Universal PHY Layer subid of each packet. It is the responsibility of the MAC Layer implementation to account for in-band interference.

7.1.5 MAC-PHY Control Messaging

The Universal PHY Layer provides a control API on a per packet basis for every transmit (Tx) packet received from the MAC Layer for over-the-air transmission and every received (Rx) over-the-air packet sent to the MAC Layer for processing. The Tx Control API provides the MAC Layer with the ability to override default PHY Layer configuration for transmit power, message duration, transmit frequency and antenna pointing as required. The Universal PHY Layer utilizes the data from the Tx Control message to populate the Common PHY Header (See Section 3.2.1 Supporting Heterogeneous Waveforms on page 29). The Rx Control API provides the MAC Layer with the appropriate receive information (receive power, noise floor, message duration, propagation delay and receive frequency) to perform functions such as SINR based packet completion calculation, in-band collision detection and channel access protocol.

7.2 Configuration Parameters

7.2.1 `bandwidth`

Defines the center frequency bandwidth in KHz. This is used to compute the receiver sensitivity and is also included in all over-the-air transmissions to support noise processing calculations.

Type: Unsigned 16 bit Integer
Range: [1, 65535]
Default: 1000
Count: 1
XML Format: `<param name="bandwidth" value="1000">`

7.2.2 antennagain

Defines the antenna gain in dBi. This is used to compute the receive power associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets. This value can be overridden by a MAC Layer using the TX Control API.

Type: Float
Range: [-1024.0, 1024.0]
Default: 0.0
Count: 1
XML Format: `<param name="antennagain" value="0.0">`

7.2.3 systemnoisefigure

Defines the system noise figure in dB. The system noise figure is used along with the bandwith to compute the receiver sensitivity. See Section 3.2 Physical Layer on page 27 for more information.

Type: Float
Range: [0.0, 1024.0]
Default: 4.0
Count: 1
XML Format: `<param name="systemnoisefigure" value="4.0">`

7.2.4 frequencyofinterest

Defines a set of frequencies in KHz that the Universal PHY Layer will monitor. Multiple frequencies can be monitored to support MAC Layer implementations with frequency agility or hopping capability. Only packets received on a frequency of interest will be sent to the MAC Layer for processing provided waveform and receive power criteria are met. Separate noise floor calculations are maintained for each frequency of interest.

Type: Unsigned 32 bit Integer
Range: [1, 4294967295]
Default: 2347000
Count: Unlimited
XML Format: `<param name="frequencyofinterest" value="2347000">`

7.2.5 pathlossmode

Defines the pathloss mode of operation. The pathloss mode of operation determines whether pathloss or location events will be used as input to the Universal PHY Layer propagation functionality. See Section 7.1.1 Pathloss Calculation on page 59 for more information.

Type: String
Range: `pathloss`, `2ray`, `freespace`
Default: `pathloss`
Count: 1
XML Format: `<param name="pathlossmode" value="pathloss">`

7.2.6 `noiseprocessingmode`

Enables or disables noise processing. When `on`, out of band packets (not of this waveform) within a frequency of interest will raise the noise floor accordingly.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="noiseprocessingmode" value="off">`

7.2.7 `defaultconnectivitymode`

Defines the default connectivity mode for pathloss. When set to `on`, full connectivity will be engaged until a valid event (pathloss or location) is received based on the `pathlossmode` setting. Any valid event of the appropriate type, regardless if it contains information for the receiving NEM, will disengage default connectivity mode. Directional antenna functionality, if applicable, is bypassed when default connectivity mode is engaged. When set to `off`, no connectivity is in effect until a valid event is received.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="defaultconnectivitymode" value="on">`

7.2.8 `txpower`

Defines the transmit power in dBm. This is used to compute the receive power associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets. This value can be overridden by a MAC Layer using the TX Control API.

Type: Float
Range: [`-1024.0`, `1024.0`]
Default: `0.0`
Count: 1
XML Format: `<param name="txpower" value="0.0">`

7.2.9 `frequency`

Defines the transmit center frequency in KHz. This value is included in the Common PHY Header of all transmitted OTA packets. This value can be overridden by a MAC Layer using the TX Control API.

Type: Unsigned 32 bit Integer
Range: [1, 4294967295]
Default: 2347000
Count: 1
XML Format: `<param name="frequency" value="2347000">`

7.2.10 antennaazimuthbeamwidth

Defines the antenna azimuth beam width in degrees. This is used in the directional antenna pointing computation associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets when the `antennatype` is unidirectional. This value can be overridden by a MAC Layer using the TX Control API and/or via antenna pointing events.

Type: Float
Range: [0.0, 360.0]
Default: 360.0
Count: 1
XML Format: `<param name="antennaazimuthbeamwidth" value="360.0">`

7.2.11 antennaelevationbeamwidth

Defines the antenna elevation beam width in degrees. This is used in the directional antenna pointing computation associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets when the `antennatype` is unidirectional. This value can be overridden by a MAC Layer using the TX Control API and/or via antenna pointing events.

Type: Float
Range: [0.0, 180.0]
Default: 180.0
Count: 1
XML Format: `<param name="antennaelevationbeamwidth" value="180.0">`

7.2.12 antennaazimuth

Defines the antenna azimuth in degrees. This is used in the directional antenna pointing computation associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets when the `antennatype` is unidirectional. This value can be overridden by a MAC Layer using the TX Control API and/or via antenna pointing events.

Type: Float
Range: [0.0, 360.0]
Default: 360.0
Count: 1
XML Format: `<param name="antennaazimuth" value="0.0">`

7.2.13 antennaelevation

Defines the antenna elevation in degrees. This is used in the directional antenna pointing computation associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets when the `antennatype` is unidirectional. This value can be overridden by a MAC Layer using the TX Control API and/or via antenna pointing events.

Type:	Float
Range:	[`-90.0`, `90.0`]
Default:	`0.0`
Count:	1
XML Format:	`<param name="antennaelevation" value="0.0">`

7.2.14 antennatype

Defines the antenna type. If the antenna type is unidirectional, location event information for both the transmitter and receiver are required as inputs to the Universal PHY Layer for receive packet processing unless default connectivity mode is engaged. See Section 7.1.3 Directional Sector Antenna Support on page 60 for more information.

Type:	String
Range:	`omnidirectional`, `unidirectional`
Default:	`omnidirectional`
Count:	1
XML Format:	`<param name="antennatype" value="omnidirectional">`

7.2.15 subid

Defines the Universal PHY Layer subid. The Universal PHY Layer is used by multiple NEM definitions. Once instantiated, these NEMs may be using the same frequency. In order to differentiate between Universal PHY instances for different waveforms, the `subid` is used as part of the unique waveform identifying tuple: PHY Layer Registration Id, Universal PHY subid and packet center frequency. The subid may also be used to emulate instances of the same waveform operating with different TRANSEC keys.

Type:	Unsigned 16 bit Integer
Range:	[`1`, `65535`]
Default:	`1`
Count:	1
XML Format:	`<param name="subid" value="1">`

7.3 Packet Processing Flows

The Universal PHY Layer receive packet (upstream) processing flowchart is shown in Figure 7.3 and Figure 7.4. The Universal PHY Layer does not perform any relevant transmit packet (downstream) processing.

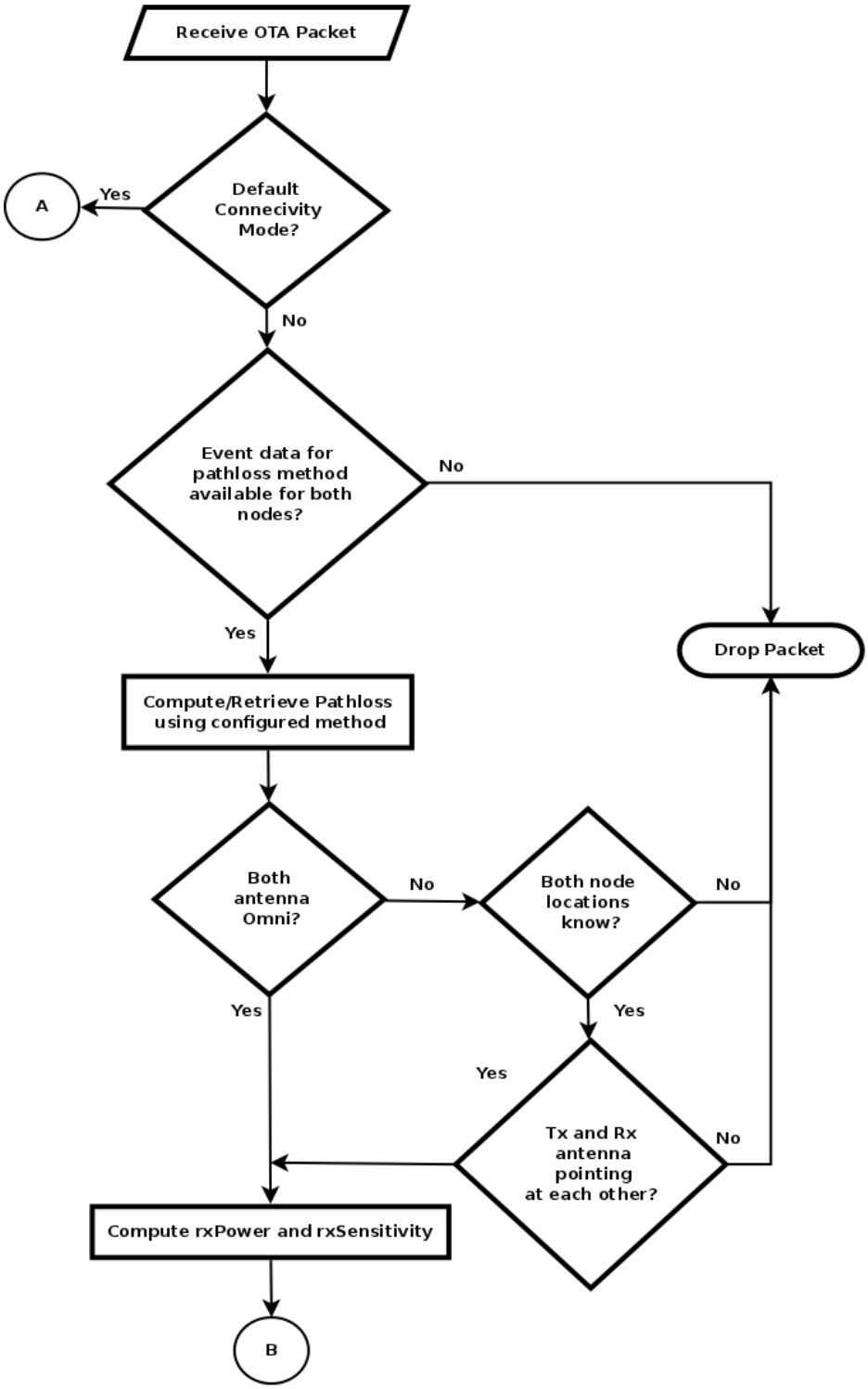


Figure 7.3: Part 1 - Universal PHY Layer receive packet (upstream) processing flow.

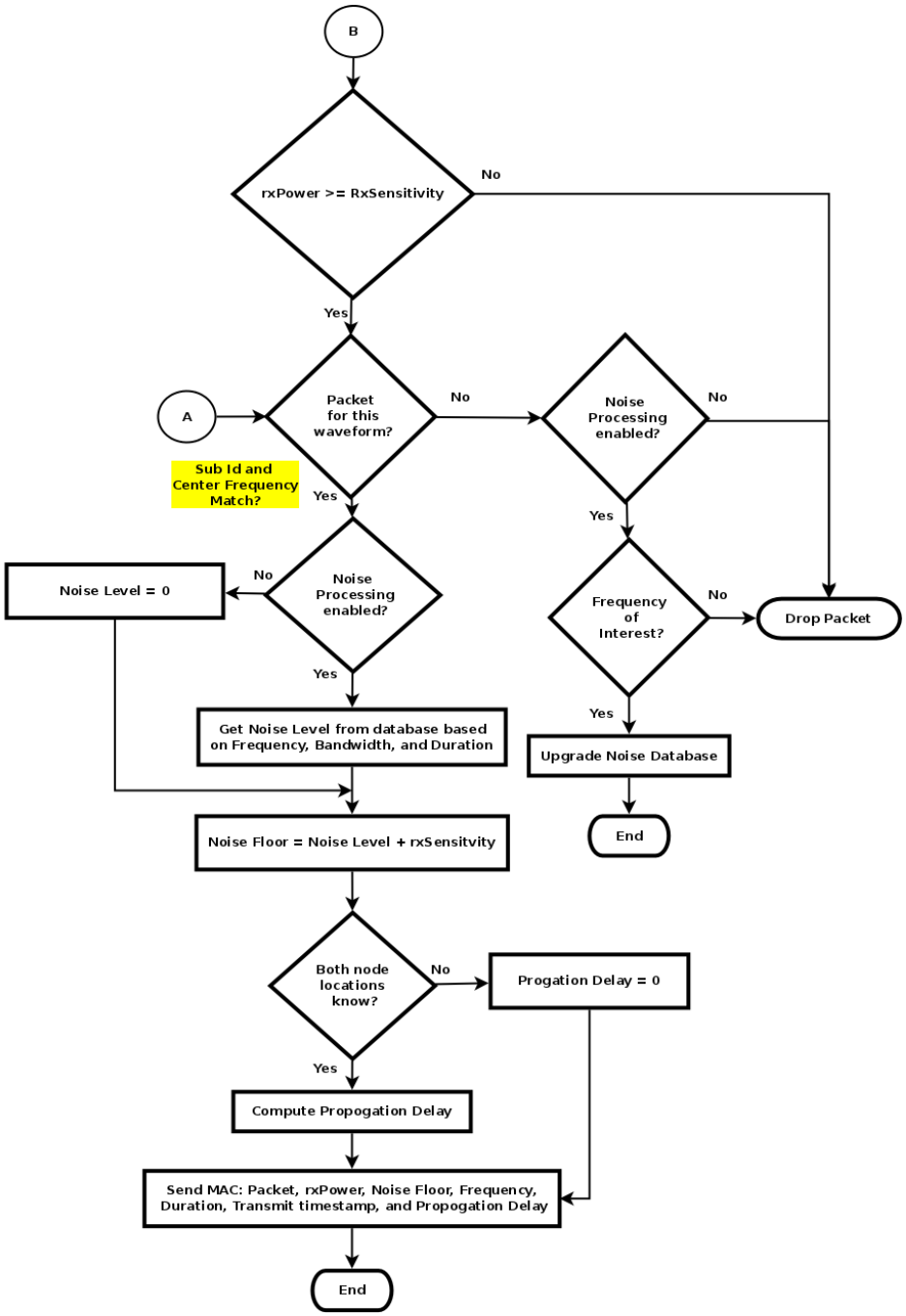


Figure 7.4: Part 2 - Universal PHY Layer receive packet (upstream) processing flow.

7.4 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

7.4.1 Demonstration 8

This demonstration deploys a four node centralized IEEE 802.11abg NEM emulation experiment illustrated in Figure 7.5. The goal of this demonstration is to become familiar with Universal PHY Layer directional antenna support.

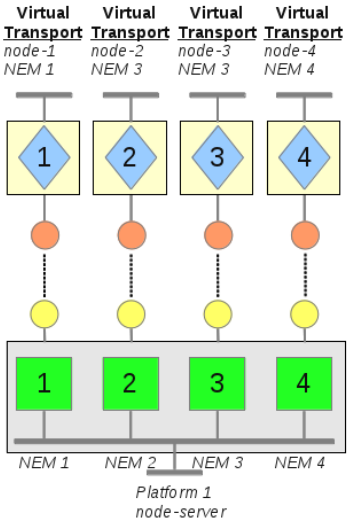


Figure 7.5: Demonstration 8 - Four node centralized IEEE 802.11abg (Universal PHY Layer) NEM deployment.

The four nodes in this demonstration are positioned in a rectangle located in Lakehurst, New Jersey.

Node:	Latitude	Longitude	Altitude
1	40.025495	-74.315441	3.0
2	40.025495	-74.312888	4.0
3	40.023235	-74.315441	5.0
4	40.023235	-74.312888	6.0

7.4.1.1 Demonstration Procedure

1. Review the Demonstration 8 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/8
[emane@emanedemo 8] less platform.xml
```

2. Review the antenna pointing scenario using your favorite editor. The Emulation Event Log Generator is used to send location and antenna point events. Figure 7.6 illustrates the antenna pointing used in this demonstration. See Chapter 15 Emulation Event Log Generator on page 137 for more information on using this generator.

```
[emane@emanedemo 8] less scenario.eel
```

3. Deploy the demonstration.

```
[emane@emanedemo 8]$ sudo ./lxc-demo-start.sh -d 10
```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Visually verify that the network link formations match the antenna pointing scenario.

6. Stop the demonstration.

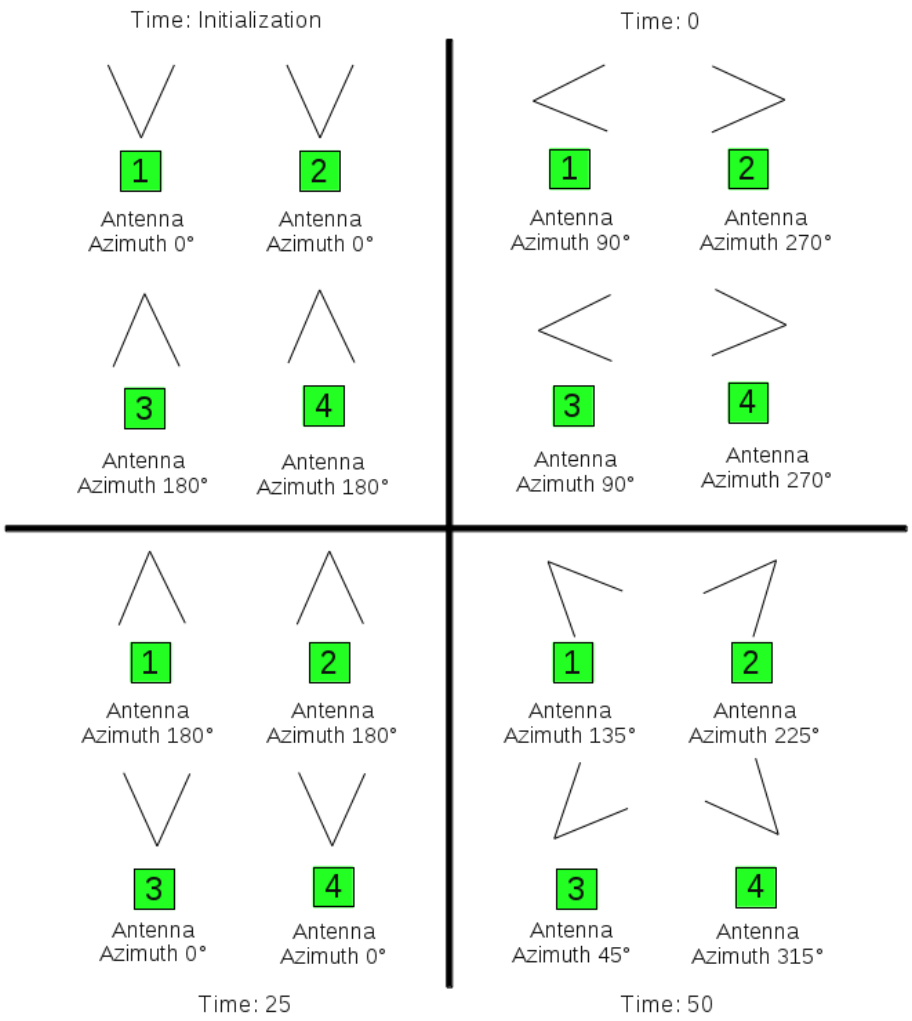


Figure 7.6: Demonstration 8 - Directional antenna pointing scenario.

```
[emane@emanedemo 8]$ sudo ./lxc-demo-stop.sh
```

7.4.1.2 Concept Review

1. How is the pathloss between nodes determined in Demonstration 8?
2. How are the initial (prior to event reception) antenna pointing and profile set for the NEMs in Demonstration 8?
3. What event information is required in order for the Universal PHY Layer to engage its directional antenna functionality?
4. What kind of profile would an antenna azimuth beam width of 360°and an elevation beam width of 180°produce?

Chapter 8

RF Pipe MAC Layer

The RF Pipe MAC Layer runs on top of the Universal PHY Layer and provides a generic MAC Layer that can be configured to emulate waveforms that do not have a specific Network Emulation Model available.

8.1 Model Features

The RF Pipe MAC Layer provides the following set of features to support wireless emulation of varying waveforms:

1. Data/Burst rate emulation of bandwidth (rate of data transfer): On the transmit side (downstream), the RF Pipe MAC Layer applies a delay between packets based on packet size and configured data rate to limit the data transfer rate as configured. Where,
   (a) Delay between packet transmissions is applied after packet transmission.
   (b) The computed delay is sent to the Universal PHY Layer and is included in the Common PHY Header as message duration.
   (c) Bandwidth is a per node limit and not an overall network limit.
2. Transmission delay emulation: On the receive side (upstream), the RF Pipe MAC Layer will compute and apply a transmission delay for each packet before sending it up the stack. The transmission delay is computed as follows:

$$transmissionDelay = messageDuration + delay + jitter + propagationDelay$$

   Where,

delay	Configuration parameter `delay` (Section 8.2.5)
jitter	Configuration parameter `jitter` (Section 8.2.6)
messageDuration	Provided by the transmitter via the Common PHY Header
propagationDelay	Provided by the Universal PHY Layer when node positions are available via location events

3. Use of user defined Packet Completion Rate (PCR) curves as a function of SINR as defined in Section 8.3 Packet Completion Rate on page 75. It should be noted that the RF Pipe MAC Layer does not apply any additional interference effects and as such, the use of negative SINR values within the PCR Curve file is valid only when noise processing is enabled within the Universal PHY Layer to raise the noise floor above the inherent receiver sensitivity (See Section 7.2.6 `noiseprocessingmode` on page 63).

8.2 Configuration Parameters

8.2.1 `enablepromiscuousmode`

Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="enablepromiscuousmode" value="off">`

Parameter value description:

Value	Description
`on`	Send all packets up the stack to the Transport Layer
`off`	Send only broadcast/multicast and locally addressed unicast up

8.2.2 `enabletighttiming`

Determines if the over-the-air time ($rxTime - txTime$) should be included in the overall packet delay time. Implies that the source and destination are in tight time sync.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="enabletighttiming" value="off">`

Parameter value description:

Value	Description
`on`	Do account for over-the-air time in delay calculation
`off`	Do not account for over-the-air time in delay calculation

8.2.3 `transmissioncontrolmap`

Defines the data rate, frequency, and power level to be used by the PHY Layer for all transmissions to a specified node. When a packet is transmitted to the destination NEM, the MAC Layer will send an accompanying control message to the PHY Layer that will cause the specified data rate, frequency and transmit power to be included in the Common PHY header.

Type: String
Range: N/A
Default: None
Count: Unlimited
XML Format:
```
<param name="transmissioncontrolmap" value="1:128:0:99">
<param name="transmissioncontrolmap" value="2:256:0:98">
<param name="transmissioncontrolmap" value="3:512:0:97">
```

Parameter value format description:
<Destination NEM>:<Data Rate>:<Frequency>:<Tx Power>

Name	Description	Type
Destination NEM	NEM Id of destination	Unsigned 16 bit Integer
Data Rate	Data Rate in Kbps	Unsigned 16 bit Integer
Frequency	Frequency in MHz	Unsigned 16 bit Integer
Tx Power	Transmit power level in dBm	Unsigned 8 bit Integer

8.2.4 `datarate`

Defines the data/burst rate in Kbps of the waveform being emulated. It is used on the transmit side (downstream) to compute transmission delay based on the packet size and data rate. The RF Pipe MAC Layer will wait for the message delay to expire before transmitting another packet.

Type: Unsigned 32 bit Integer
Range: [`1`, `4294967295`]
Default: `1000`
Count: 1
XML Format: `<param name="datarate" value="1000">`

8.2.5 `delay`

Defines the delay in microseconds that is to be included in the transmission delay. The delay is added to the delay introduced by the `datarate` parameter.

Type: Float
Range: [`1`, `MAX_FLOAT`]
Default: `0`
Count: 1
XML Format: `<param name="delay" value="0">`

8.2.6 `jitter`

Defines the jitter in microseconds to be included to the transmission delay. The jitter will be computed for each packet transmission based on uniform random distribution between +/- the configured jitter value.

Type: Float
Range: [`MIN_FLOAT`, `MAX_FLOAT`]
Default: `0`
Count: 1
XML Format: `<param name="jitter" value="0">`

8.2.7 `pcrcurveuri`

Defines the absolute file name that contains the SINR/PCR curve values. A minimum of one SINR/PCR pair is required, $POR = 0.0$ and $POR = 100.0$. Entries shall be in unique ascending order with up to two decimal places of precision for SINR. The PCR values shall represent the percentage with up to two decimal places of precision. See Section 8.3 Packet Completion Rate for more information.

Type: String
Range: N/A
Default: `rfpipepcr.xml`
Count: 1
XML Format: `<param="pcrcurveuri" value="rfpipepcr.xml">`

8.2.8 `flowcontrolenable`

Enables downstream traffic flow control with Virtual Transport. **flowcontrolenable** is only valid when using the Virtual Transport and the setting to either `on` or `off` must match the setting of **flowcontrolenable** within the Virtual Transport configuration. See Section 11.3 Flow Control on page 112 for more information.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="flowcontrolenable" value="off">`

Parameter value description:

Value	Description
`on`	Enable flow control with the Virtual Transport
`off`	Disable flow control with the Virtual Transport

8.2.9 `flowcontroltokens`

Defines the number of flow control tokens. This is an optional parameter used to override the default token setting when **flowcontrolenable** is `on`. See Section 11.3 Flow Control on page 112 for more information.

Type: Unsigned 16 bit Integer
Range: [`1`, `65535`]
Default: `10`
Count: 1
XML Format: `<param name="flowcontroltokens" value="10">`

8.3 Packet Completion Rate

The RF Pipe Packet Completion Rate is specified as a curve defined via XML. The curve definition is comprised of a series of SINR values along with their corresponding probability of reception. The curve definition must contain a minimum of two points with one SINR representing $POR = 0$ and one SINR representing $POR = 100$. Linear interpolation is preformed when an exact SINR match is not found. Listing 8.1 shows the default RF Pipe PCR curve file which results in the curve depicted in Figure 8.1.

Specifying a packet size (`<table>` attribute `pktsize`) in the curve file will adjust the POR based on received packet size. Specifying a `pktsize` of `0` disregards received packet size when computing the POR. The POR is obtained using the following calculation when a non-zero `pktsize` is specified:

$$POR = POR_o^{S_1/S_o}$$

Where,

POR_o	POR value determined from the PCR curve for the given SINR value
S_0	Packet size specified in the curve file (`pktsize`)
S_1	Received packet size

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pcr SYSTEM "file:///usr/share/emane/dtd/rfpipepcr.dtd">
<pcr>
    <table pktsize="0">
        <row sinr="0.0"   por="0"/>
        <row sinr="0.5"   por="2.5"/>
        <row sinr="1.0"   por="5"/>
        <row sinr="1.5"   por="7.5"/>
        <row sinr="2.0"   por="10"/>
        <row sinr="2.5"   por="12.5"/>
        <row sinr="3.0"   por="15"/>
        <row sinr="3.5"   por="17.5"/>
        <row sinr="4.0"   por="20"/>
        <row sinr="4.5"   por="22.5"/>
        <row sinr="5.0"   por="25"/>
        <row sinr="5.5"   por="27.5"/>
        <row sinr="6.0"   por="30"/>
        <row sinr="6.5"   por="32.5"/>
        <row sinr="7.0"   por="35"/>
        <row sinr="7.5"   por="37.5"/>
        <row sinr="8.0"   por="40"/>
        <row sinr="8.5"   por="42.5"/>
        <row sinr="9.0"   por="45"/>
        <row sinr="9.5"   por="47.5"/>
        <row sinr="10.0"  por="50"/>
        <row sinr="10.5"  por="52.5"/>
        <row sinr="11.0"  por="55"/>
        <row sinr="11.5"  por="57.5"/>
        <row sinr="12.0"  por="60"/>
        <row sinr="12.5"  por="62.5"/>
        <row sinr="13.0"  por="65"/>
        <row sinr="13.5"  por="67.5"/>
        <row sinr="14.0"  por="70"/>
        <row sinr="14.5"  por="72.5"/>
        <row sinr="15.0"  por="75"/>
        <row sinr="15.5"  por="77.5"/>
        <row sinr="16.0"  por="80"/>
        <row sinr="16.5"  por="82.5"/>
        <row sinr="17.0"  por="85"/>
        <row sinr="17.5"  por="87.5"/>
        <row sinr="18.0"  por="90"/>
        <row sinr="18.5"  por="92.5"/>
        <row sinr="19.0"  por="95"/>
        <row sinr="19.5"  por="97.5"/>
        <row sinr="20.0"  por="100"/>
    </table>
</pcr>
```

Listing 8.1: rfpipepcr.xml

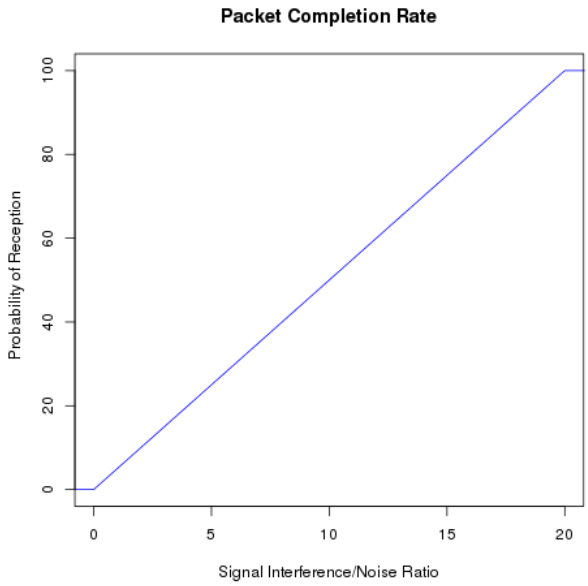


Figure 8.1: Graph of the RF Pipe Packet Completion Rate resulting from Listing 8.1.

8.4 Packet Processing Flows

The RF Pipe MAC Layer receive packet processing (upstream) flowchart is shown in Figure 8.2. The RF Pipe MAC Layer transmit packet processing (downstream) flowchart is shown in Figure 8.3.

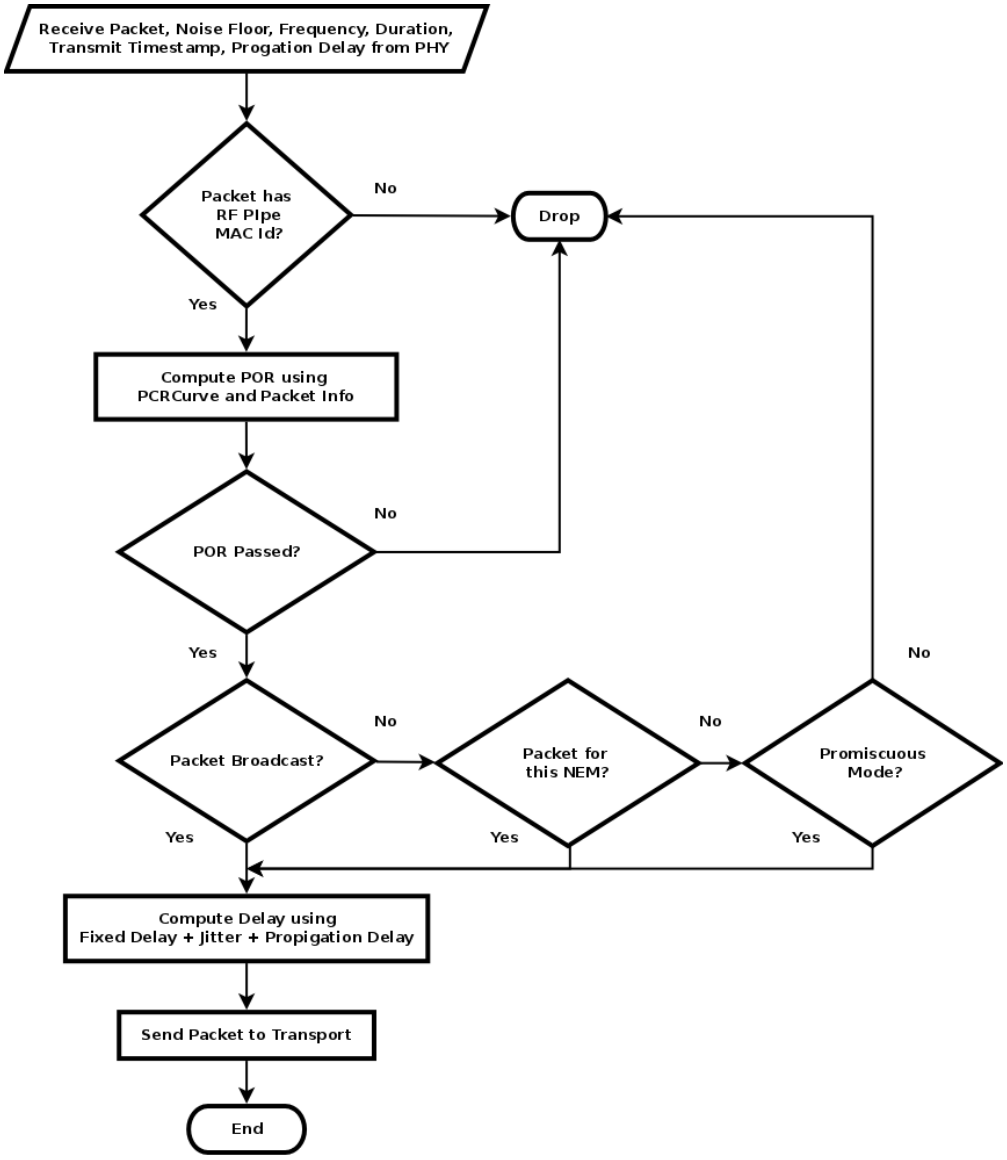


Figure 8.2: RF Pipe MAC Layer receive packet processing (upstream) flow.

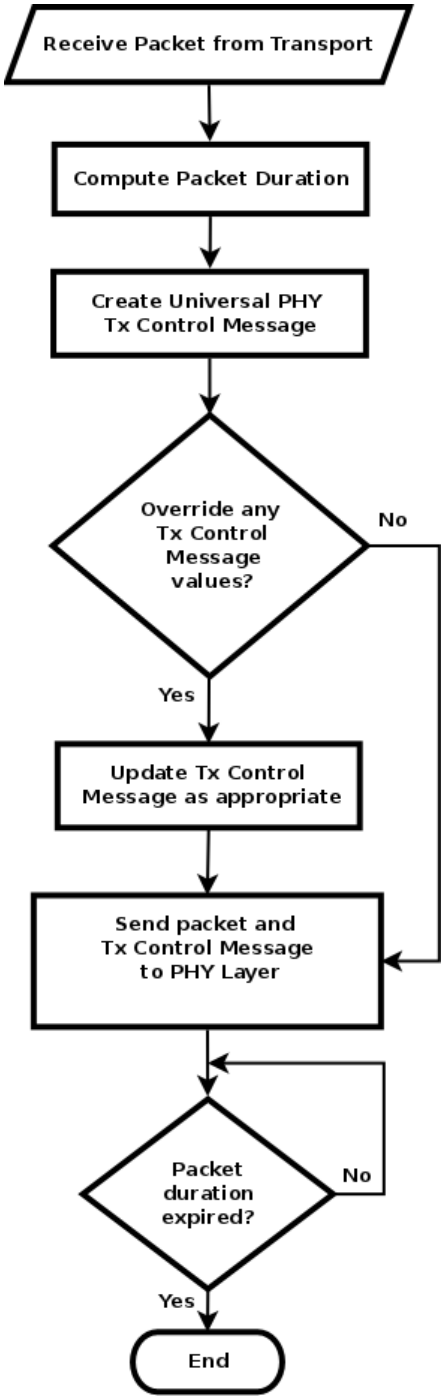


Figure 8.3: RF Pipe MAC Layer transmit packet processing (downstream) flow.

8.5 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

8.5.1 Demonstration 9

This demonstration deploys a seven node nine NEM centralized RF Pipe emulation experiment illustrated in Figure 8.4. The goal of this demonstration is to become familiar with using the RF Pipe MAC Layer to create surrogate waveform NEM definitions.

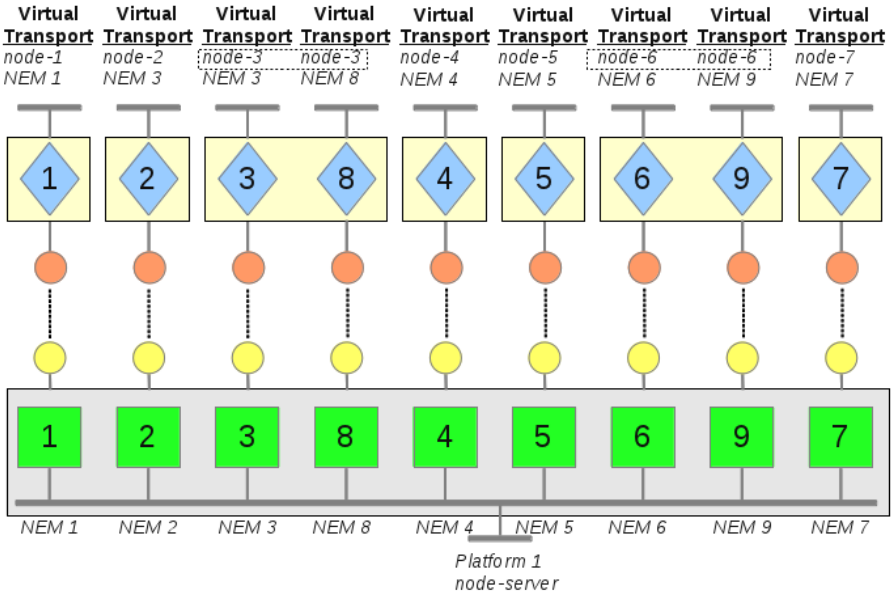


Figure 8.4: Demonstration 9 - Nine node centralized RF Pipe NEM Deployment with two 2-Channel gateways.

Figure 8.5 illustrates the resulting network formed once the mobility scenario begins. In this demonstration a SatCom Network comprised of NEMs 7, 8 and 9 is used to bridge two spectrally separated wireless networks. Two 2-Channel gateways are used to route between the three networks. One gateway is comprised of NEMs 3 and 8, and the other is comprised of NEMs 6 and 9.

8.5.1.1 Demonstration Procedure

1. Review the Demonstration 9 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/9
[emane@emanedemo 9] less platform.xml
```

2. Review the pathloss scenario using your favorite editor. The Emulation Event Log Generator is used to send pathloss events. See Chapter 15 Emulation Event Log Generator on page 137 for more information on using this generator.

```
[emane@emanedemo 9] less scenario.eel
```

3. Deploy the demonstration.

```
[emane@emanedemo 9]$ sudo ./lxc-demo-start.sh -d 10
```

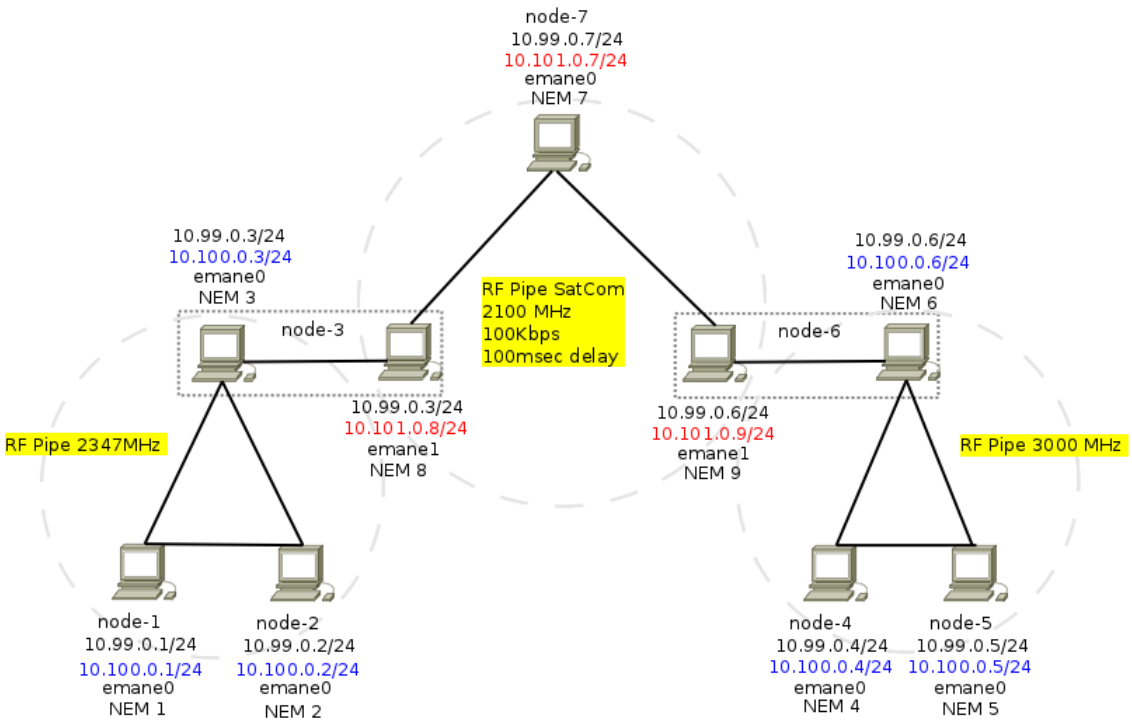


Figure 8.5: Demonstration 9 - Network Diagram. Networks in *blue* and *red* represent emulated wireless networks.

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Connect to virtual `node-1`.

```
[emane@emanedemo 1]$ ssh node-1
```

6. Ping a radio in NEM 1's network using the *radio-NEMID* host naming convention. Take note of the round-trip time.

```
[emane@node-1 ~]$  ping -c 5 radio-2
PING radio-2 (10.100.0.2) 56(84) bytes of data.
64 bytes from radio-2 (10.100.0.2): icmp_req=1 ttl=64 time=10.5 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=2 ttl=64 time=6.30 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=3 ttl=64 time=6.02 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=4 ttl=64 time=3.23 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=5 ttl=64 time=11.5 ms

--- radio-2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4012ms
rtt min/avg/max/mdev = 3.237/7.514/11.507/3.062 ms
```

7. Ping a radio in NEM 6's network using the *radio-NEMID* host naming convention. Take note of the round-trip time.

```
[emane@node-1 ~] ping -c 5 radio-6
PING radio-6 (10.100.0.6) 56(84) bytes of data.
64 bytes from radio-6 (10.100.0.6): icmp_req=1 ttl=62 time=450 ms
64 bytes from radio-6 (10.100.0.6): icmp_req=2 ttl=62 time=463 ms
64 bytes from radio-6 (10.100.0.6): icmp_req=3 ttl=62 time=448 ms
64 bytes from radio-6 (10.100.0.6): icmp_req=4 ttl=62 time=450 ms
64 bytes from radio-6 (10.100.0.6): icmp_req=5 ttl=62 time=459 ms

--- radio-6 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4004ms
```

```
rtt min/avg/max/mdev = 448.992/454.695/463.379/5.757 ms
```

8. Ping SatCom NEM-7 using 10.101.0.7. Take note of the round-trip time.

```
[emane@node-1 ~]$  ping -c 5 10.101.0.7
PING 10.101.0.7 (10.101.0.7) 56(84) bytes of data.
64 bytes from 10.101.0.7: icmp_req=1 ttl=63 time=223 ms
64 bytes from 10.101.0.7: icmp_req=2 ttl=63 time=226 ms
64 bytes from 10.101.0.7: icmp_req=3 ttl=63 time=229 ms
64 bytes from 10.101.0.7: icmp_req=4 ttl=63 time=226 ms
64 bytes from 10.101.0.7: icmp_req=5 ttl=63 time=225 ms

--- 10.101.0.7 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4013ms
rtt min/avg/max/mdev = 223.509/226.485/229.760/2.115 ms
```

9. Stop the demonstration.

```
[emane@emanedemo 9]$ sudo ./lxc-demo-stop.sh
```

8.5.1.2 Concept Review

1. Explain the difference in the observed round-trip times from Demonstration 9.
2. What are some alternative ways the configuration for the three emulated wireless networks could have been specified?
3. Take a look at the Transport Daemon XML used in this demonstration. What effect does the use of the `group` attribute in the `<transport>` element of the NEM definitions for NEMs 3, 6, 8 and 9 in the *platform.xml* have on the Transport Daemon XML?

Chapter 9

IEEE 802.11abg MAC Layer

The IEEE 802.11abg MAC Layer implementation runs on top of the Universal PHY Layer and has the objective of emulating the IEEE 802.11 MAC layer's Distributed Coordination Function (DCF) channel access scheme on top of the IEEE 802.11 Direct Spread Spectrum Sequence (DSS) and Orthogonal Frequency Division Multiplexing (OFDM) signals in space. The IEEE 802.11abg MAC Layer implementation of the Carrier Sense Multiple Access (CSMA) channel access protocol is based on applying collision effects from one and two hop (hidden) neighbors using realtime estimation of number of neighbors and channel activity. It is not an implementation of the protocol as defined in the IEEE 802.11 standard.

9.1 Model Features

The IEEE 802.11abg MAC Layer provides the following set of features:

1. Supports flow control as described in Section 11.3 Flow Control on page 112.
2. Supports the following waveform modes and data rates with the appropriate timing:
   (a) 802.11b (DSS rates: 1, 2, 5.5 and 11 Mbps)
   (b) 802.11a/g (OFDM rates: 6, 9, 12, 18, 24, 36, 48 and 54 Mbps)
   (c) 802.11b/g (DSS and OFDM rates)
3. Supports only the DCF channel access function. PCF and beacon transmissions are not supported.
4. Supports both unicast and broadcast transmissions. Unicast transmissions include the ability to emulate control message (RTS/CTS) behavior as well as retries without actually transmitting the control messages or the re-transmission of the data message. The emulation of unicast does not replicate exponential growth of the contention window as a result of detected failures.
5. Supports Wi-Fi multimedia (WMM) capabilities. The initial implementation supports the ability to classify four different traffic classes (Background, Best Effort, Video and Voice) where the higher priority classes (voice and video) are serviced first.
6. Supports user defined Packet Completion Rate (PCR) curves as a function of SINR as defined in Section 9.3 Packet Completion Rate on page 90.
7. Default curves are provided for each of the supported 802.11 modulation and data rate combinations. Default curves are based on theoretical equations for determining Bit Error Rate (BER) in an Additive White Gaussian Noise (AWGN) channel.
8. The IEEE 802.11abg MAC Layer does adjust the interference on a per packet basis based on detected collisions and as such supports negative SINR values as can be seen in the default curves.

9.2 Configuration Parameters

9.2.1 `mode`

Defines the 802.11 mode

Type:	Unsigned 8 bit Integer
Range:	[`0`, `3`]
Default:	`0`
Count:	1
XML Format:	`<param name="mode" value="0">`

Parameter value description:

Value	Description	
`0`	802.11b	DSSS only
`1`	802.11 a or g	OFDM
`2`	802.11b	DSSS only
`3`	802.11 b/g	DSSS and OFDM

9.2.2 `enablepromiscuousmode`

Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.

Type:	Boolean
Range:	[`off`, `on`]
Default:	`off`
Count:	1
XML Format:	`<param name="enablepromiscuousmode" value="off">`

Parameter value description:

Value	Description
`on`	Send all packets up the stack to the transport layer
`off`	Send only broadcast/multicast and locally addressed unicast up

9.2.3 `distance`

Defines the maximum distance in meters for supported point to point links within the network. This is used to adjust the slot timing to account for round trip propagation delays.

$$slotTime = fixedSlotTime + propagationTime$$

$$fixedSlotTime = 9$$

$$propagationTime = distance/300$$

Type: Unsigned 32 bit Integer
Range: [1, 4294967295]
Default: 1000
Count: 1
XML Format: `<param name="distance" value="1000">`

9.2.4 unicastrate

Defines the data rate index to be used for all unicast transmissions. The rate selection must be valid for the mode selected as defined in the Section 9.1 Model Features on page 83.

Type: Unsigned 8 bit Integer
Range: [1, 12]
Default: 4
Count: 1
XML Format: `<param name="unicastrate" value="4">`

Parameter value description:

Value	Description
1	1 Mbps
2	2 Mbps
3	5.5 Mbps
4	11 Mbps
5	6 Mbps
6	9 Mbps
7	12 Mbps
8	18 Mbps
9	24 Mbps
10	36 Mbps
11	48 Mbps
12	54 Mbps

9.2.5 multicastrate

Defines the data rate index to be used for all multicast/broadcast transmissions. The rate selection must be valid for the mode selected as defined in the Section 9.1 Model Features on page 83.

Type: Unsigned 8 bit Integer
Range: [1, 12]
Default: 1
Count: 1
XML Format: `<param name="multicastrate" value="1">`

Parameter value description:

Value	Description
1	1 Mbps
2	2 Mbps
3	5.5 Mbps
4	11 Mbps
5	6 Mbps
6	9 Mbps
7	12 Mbps
8	18 Mbps
9	24 Mbps
10	36 Mbps
11	48 Mbps
12	54 Mbps

9.2.6 rtsthreshold

Defines the minimum packet size in bytes required to trigger RTS/CTS for unicast transmissions. A value of 0 disables RTS/CTS. The effect of RTS/CTS for unicast transmissions is emulated using a statistical model.

Type: Unsigned 16 bit Integer
Range: [0, 65535]
Default: 0
Count: 1
XML Format: `<param name="rtsthreshold" value="0">`

9.2.7 wmmenable

Provides the ability to enable the WiFi Multimedia (WMM) type feature. Current capability supports the ability to service packets from a higher priority queue first and does not yet support the internal contention based logic as defined by 802.11e.

Type: Boolean
Range: [off, on]
Default: off
Count: 1
XML Format: `<param name="wmmenable" value="off">`

Parameter value description:

Value	Description
on	Enable WMM
off	Disable WMM

9.2.8 `pcrcurveuri`

Defines the absolute file name that contains the SINR/PCR curve values. A minimum of two SINR/PCR row entries per data rate are required, $POR = 0.0$ and $POR = 100.0$. Entries shall be in unique ascending order with up to two decimal places of precision for SINR. The PCR values shall represent the percentage with up to two decimal places of precision. See Section 9.3 Packet Completion Rate on page 90 for more information.

Type:	String
Range:	N/A
Default:	`ieee80211pcr.xml`
Count:	1
XML Format:	`<param="pcrcurveuri" value="ieee80211pcr.xml">`

9.2.9 `flowcontrolenable`

Enables downstream traffic flow control with Virtual Transport. **flowcontrolenable** is only valid when using the Virtual Transport and the setting to either `on` or `off` must match the setting of **flowcontrolenable** within the Virtual Transport configuration. See Section 11.3 Flow Control on page 112 for more information.

Type:	Boolean
Range:	[`off`, `on`]
Default:	`off`
Count:	1
XML Format:	`<param name="flowcontrolenable" value="off">`

Parameter value description:

Value	Description
`on`	Enable flow control with the Virtual Transport
`off`	Disable flow control with the Virtual Transport

9.2.10 `flowcontroltokens`

Defines the number of flow control tokens. This is an optional parameter used to override the default token setting when **flowcontrolenable** is `on`. See Section 11.3 Flow Control on page 112 for more information.

Type:	Unsigned 16 bit Integer
Range:	[`1`, `65535`]
Default:	`10`
Count:	1
XML Format:	`<param name="flowcontroltokens" value="10">`

9.2.11 `queuesize`

Defines the size of the queue for the given access category. When `wmmenable` is `off` only access category `0` is used.

Type: String
Range: N/A
Default: `0:255 1:255 2:255 3:255`
Count: 1
XML Format: `<param name="queuesize" value="0:255 1:255 2:255 3:255">`

Parameter value format description:
`<Access Category>:<Queue Size> [<Access Category>:<Queue Size>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
Queue Size	Queue Size	[1, 255]

9.2.12 `cwmin`

Defines the minimum contention window in slots for the appropriate access category. This value is used when calculating the overall packet duration. When `wmmenable` is `off` only access category `0` is used.

Type: String
Range: N/A
Default: `0:32 1:32 2:16 3:8`
Count: 1
XML Format: `<param name="cwmin" value="0:32 1:32 2:16 3:8">`

Parameter value format description:
`<Access Category>:<Contention Window> [<Access Category>:<Contention Window>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
Contention Window	Minimum contention window	[1, 65535]

9.2.13 `cwmax`

Defines the maximum contention window in slots for the appropriate access category. Not currently used for point-to-point failure exponential growth. When `wmmenable` is `off` only access category `0` is used.

Type: String
Range: N/A
Default: `0:1024 1:1024 2:64 3:16`
Count: 1
XML Format: `<param name="cwmax" value="0:1024 1:1024 2:64 3:16">`

Parameter value format description:

`<Access Category>:<Contention Window> [<Access Category>:<Contention Window>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
Contention Window	Maximum contention window	[1, 65535]

9.2.14 aifs

Defines the Arbitration Inter Frame Space (AIFS) time factor in slots for the appropriate access category. The inter frame space time in microseconds is computed as follows: $time = aifs * slotduration + sifs$, where *slotduration* is a function of distance and *sifs* is a function of the 802.11 mode. When `wmmenable` is `off` only access category `0` is used.

Type: String
Range: N/A
Default: `0:2 1:2 2:2 3:1`
Count: 1
XML Format: `<param name="aifs" value="0:2 1:2 2:2 3:1">`

Parameter value format description:

`<Access Category>:<AIFS Factor> [<Access Category>:<AIFS Factor>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
AIFS Factor	AIFS factor value in slots	[0, 255]

9.2.15 txop

Defines the maximum time in microseconds a packet can reside within the queue for a given access category. Once the packet enters the MAC queue and is not serviced for this time, it will be discarded and not transmitted. Setting the value to `0` disables the feature and will service all packets regardless of duration in the queue. When `wmmenable` is `off` only access category `0` is used.

Type: String
Range: N/A
Default: `0:0 1:0 2:0 3:0`
Count: 1
XML Format: `<param name="txop" value="0:0 1:0 2:0 3:0">`

Parameter value format description:

`<Access Category>:<Duration> [<Access Category>:<Duration>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
Duration	txop duration in microseconds	[0, 1000000]

9.2.16 `retrylimit`

Defines the number of retries permitted for the unicast messages for the appropriate access category. When `wmmenable` is `off` only access category `0` is used.

Type:	String
Range:	N/A
Default:	`0:2 1:2 2:2 3:2`
Count:	1
XML Format:	`<param name="retrylimit" value="0:2 1:2 2:2 3:2">`

Parameter value format description:

`<Access Category>:<Retry Limit> [<Access Category>:<Retry Limit>]...`

Name	Description	Range
Access Category	The access category	[0, 3]
Retry Limit	Number of retries	[0, 255]

9.3 Packet Completion Rate

The IEEE 802.11abg Packet Completion Rate is specified as curves defined via XML. The curve definitions are comprised of a series SINR values along with their corresponding probability of reception. A curve definition must contain a minimum of two points with one SINR representing $POR = 0$ and one SINR representing $POR = 100$. Linear interpolation is preformed when an exact SINR match is not found. Listing 9.1 shows the default IEEE 802.11abg PCR curve file which results in the curves depicted in Figure 9.1.

Specifying a packet size (`<table>` attribute `pktsize`) in the curve file will adjust the POR based on received packet size. Specifying a `pktsize` of `0` disregards received packet size when computing the POR. The POR is obtained using the following calculation when a non-zero `pktsize` is specified:

$$POR = POR_o^{S_1/S_o}$$

Where,

POR_o	POR value determined from the PCR curve for the give SINR value
S_0	Packet size specified in the curve file (`pktsize`)
S_1	Received packet size

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE pcr SYSTEM
"file:///usr/local/share/emane/dtd/ieee80211pcr.dtd">

<pcr>
  <table pktsize="128">
    <datarate index="1" rate="1Mbps">
      <row sinr="-9.0"  por="0.0"/>
      <row sinr="-8.0"  por="1.4"/>
      <row sinr="-7.0"  por="21.0"/>
      <row sinr="-6.0"  por="63.5"/>
      <row sinr="-5.0"  por="90.7"/>
      <row sinr="-4.0"  por="98.6"/>
      <row sinr="-3.0"  por="99.9"/>
      <row sinr="-2.0"  por="100.0"/>
    </datarate>

    <datarate index="2" rate="2Mbps">

    <datarate index="7" rate="12Mbps">
      <row sinr="3.0"   por="0.0"/>
      <row sinr="4.0"   por="14.3"/>
      <row sinr="5.0"   por="55.2"/>
      <row sinr="6.0"   por="87.5"/>
      <row sinr="7.0"   por="97.8"/>
      <row sinr="8.0"   por="99.8"/>
      <row sinr="9.0"   por="100.0"/>
    </datarate>

    <datarate index="8" rate="18Mbps">
      <row sinr="4.0"   por="0.0"/>
      <row sinr="5.0"   por="1.7"/>
      <row sinr="6.0"   por="21.5"/>
      <row sinr="7.0"   por="65.0"/>
      <row sinr="8.0"   por="91.2"/>
      <row sinr="9.0"   por="98.7"/>
```

```
19:       <row sinr="-6.0"  por="0"/>
20:       <row sinr="-5.0"  por="1.4"/>
21:       <row sinr="-4.0"  por="20.6"/>
22:       <row sinr="-3.0"  por="63.1"/>
23:       <row sinr="-2.0"  por="90.5"/>
24:       <row sinr="-1.0"  por="98.5"/>
25:       <row sinr="0.0"   por="99.9"/>
26:       <row sinr="1.0"   por="100.0"/>
27:     </datarate>
28:
29:     <datarate index="3" rate="5.5Mbps">
30:       <row sinr="-2.0"  por="0.0"/>
31:       <row sinr="-1.0"  por="0.2"/>
32:       <row sinr="0.0"   por="9.1"/>
33:       <row sinr="1.0"   por="46.2"/>
34:       <row sinr="2.0"   por="82.8"/>
35:       <row sinr="3.0"   por="96.7"/>
36:       <row sinr="4.0"   por="99.6"/>
37:       <row sinr="5.0"   por="100.0"/>
38:     </datarate>
39:
40:     <datarate index="4" rate="11Mbps">
41:       <row sinr="1.0"   por="0.0"/>
42:       <row sinr="2.0"   por="0.2"/>
43:       <row sinr="3.0"   por="8.9"/>
44:       <row sinr="4.0"   por="45.8"/>
45:       <row sinr="5.0"   por="82.5"/>
46:       <row sinr="6.0"   por="96.7"/>
47:       <row sinr="7.0"   por="99.6"/>
48:       <row sinr="8.0"   por="100.0"/>
49:     </datarate>
50:
51:     <datarate index="5" rate="6Mbps">
52:       <row sinr="-2.0"  por="0.0"/>
53:       <row sinr="-1.0"  por="5.5"/>
54:       <row sinr="0.0"   por="39.8"/>
55:       <row sinr="1.0"   por="79.0"/>
56:       <row sinr="2.0"   por="96.0"/>
57:       <row sinr="3.0"   por="99.5"/>
58:       <row sinr="4.0"   por="100.0"/>
59:     </datarate>
60:
61:     <datarate index="6" rate="9Mbps">
62:       <row sinr="-1.0"  por="0.0"/>
63:       <row sinr="0.0"   por="0.3"/>
64:       <row sinr="1.0"   por="10.5"/>
65:       <row sinr="2.0"   por="50.3"/>
66:       <row sinr="3.0"   por="84.9"/>
67:       <row sinr="4.0"   por="97.5"/>
68:       <row sinr="5.0"   por="99.7"/>
69:       <row sinr="6.0"   por="100.0"/>
70:     </datarate>
89:       <row sinr="10.0"  por="99.9"/>
90:       <row sinr="11.0"  por="100.0"/>
91:     </datarate>
92:
93:     <datarate index="9" rate="24Mbps">
94:       <row sinr="9.0"   por="0.0"/>
95:       <row sinr="10.0"  por="2.2"/>
96:       <row sinr="11.0"  por="23.8"/>
97:       <row sinr="12.0"  por="64.4"/>
98:       <row sinr="13.0"  por="90.4"/>
99:       <row sinr="14.0"  por="98.4"/>
100:      <row sinr="15.0"  por="99.8"/>
101:      <row sinr="16.0"  por="100.0"/>
102:    </datarate>
103:
104:    <datarate index="10" rate="36Mbps">
105:      <row sinr="10.0"  por="0.0"/>
106:      <row sinr="11.0"  por="0.1"/>
107:      <row sinr="12.0"  por="4.6"/>
108:      <row sinr="13.0"  por="32.4"/>
109:      <row sinr="14.0"  por="72.8"/>
110:      <row sinr="15.0"  por="93.4"/>
111:      <row sinr="16.0"  por="99.0"/>
112:      <row sinr="17.0"  por="99.9"/>
113:      <row sinr="18.0"  por="100.0"/>
114:    </datarate>
115:
116:    <datarate index="11" rate="48Mbps">
117:      <row sinr="16.0"  por="0.0"/>
118:      <row sinr="17.0"  por="1.3"/>
119:      <row sinr="18.0"  por="15.8"/>
120:      <row sinr="19.0"  por="53.5"/>
121:      <row sinr="20.0"  por="84.9"/>
122:      <row sinr="21.0"  por="96.8"/>
123:      <row sinr="22.0"  por="99.6"/>
124:      <row sinr="23.0"  por="100.0"/>
125:    </datarate>
126:
127:    <datarate index="12" rate="54Mbps">
128:      <row sinr="17.0"  por="0.0"/>
129:      <row sinr="18.0"  por="0.2"/>
130:      <row sinr="19.0"  por="5.7"/>
131:      <row sinr="20.0"  por="32.4"/>
132:      <row sinr="21.0"  por="71.3"/>
133:      <row sinr="22.0"  por="92.4"/>
134:      <row sinr="23.0"  por="99.9"/>
135:      <row sinr="24.0"  por="100.0"/>
136:    </datarate>
137:  </table>
138: </pcr>
```

Listing 9.1: ieee80211pcr.xml

9.4 Packet Processing Flows

The IEEE 802.11 MAC Layer receive packet processing (upstream) flowchart is shown in Figure 9.2 and Figure 9.3. The IEEE 802.11 MAC Layer transmit packet processing (downstream) flowchart is shown in Figure 9.4.

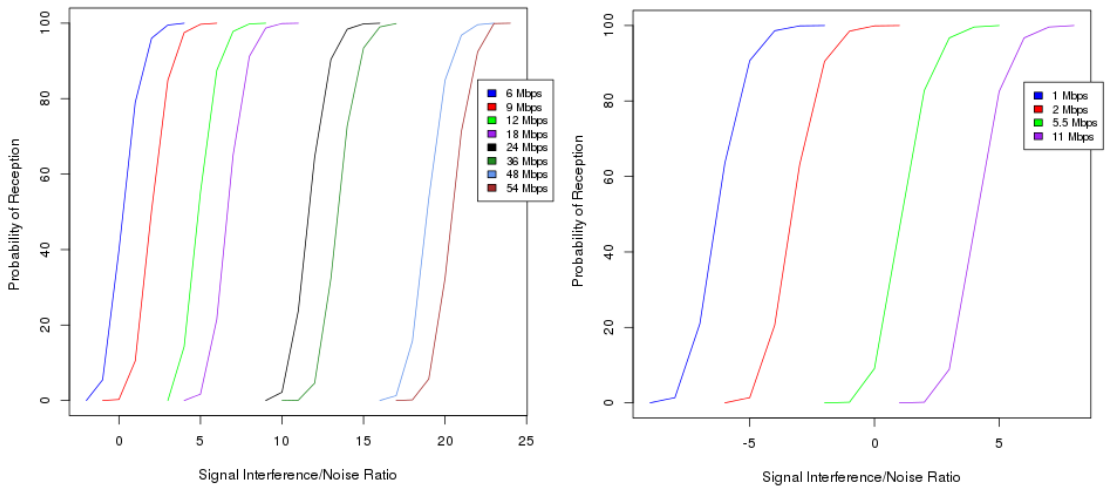


Figure 9.1: Graph of the IEEE 802.11abg Packet Completion Rate resulting from Listing 9.1.

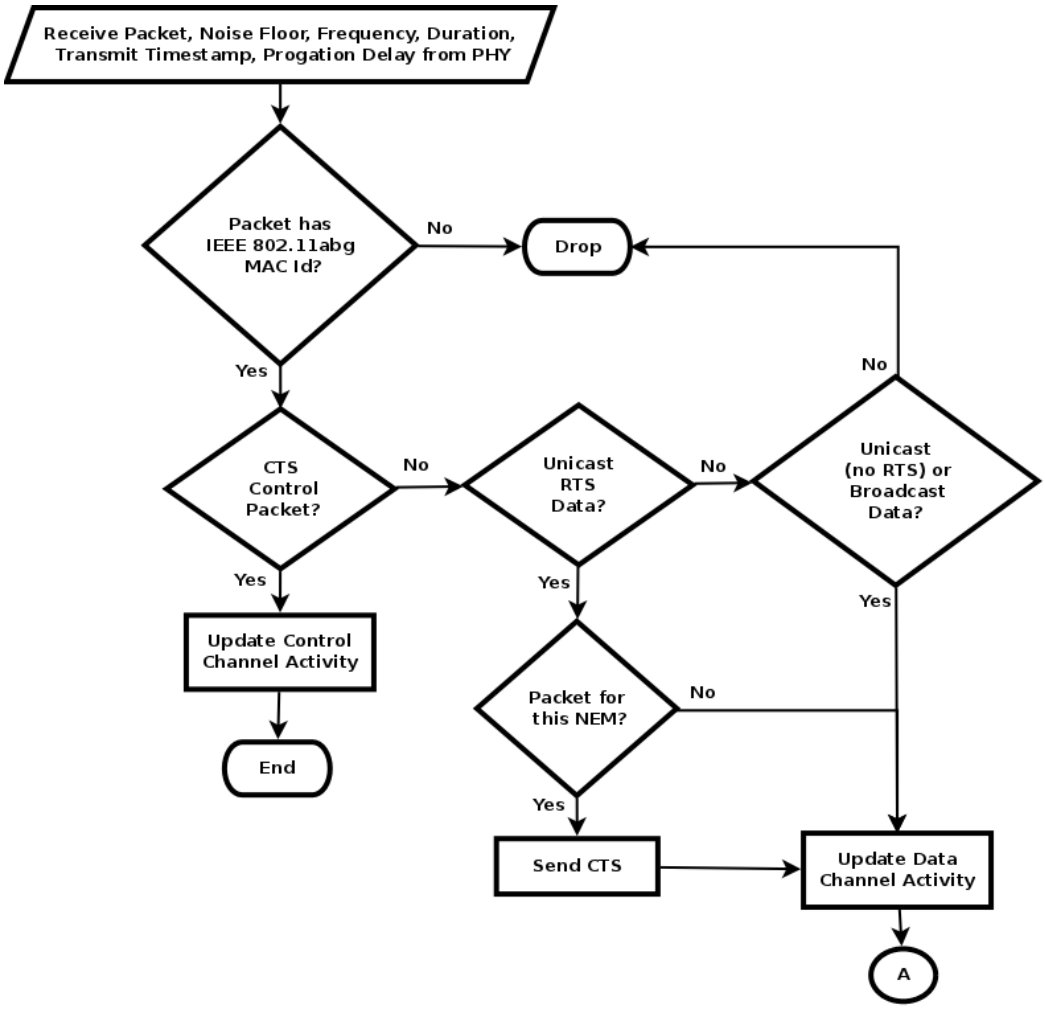


Figure 9.2: Part 1 - IEEE 802.11abg MAC Layer receive packet processing (upstream) flow.

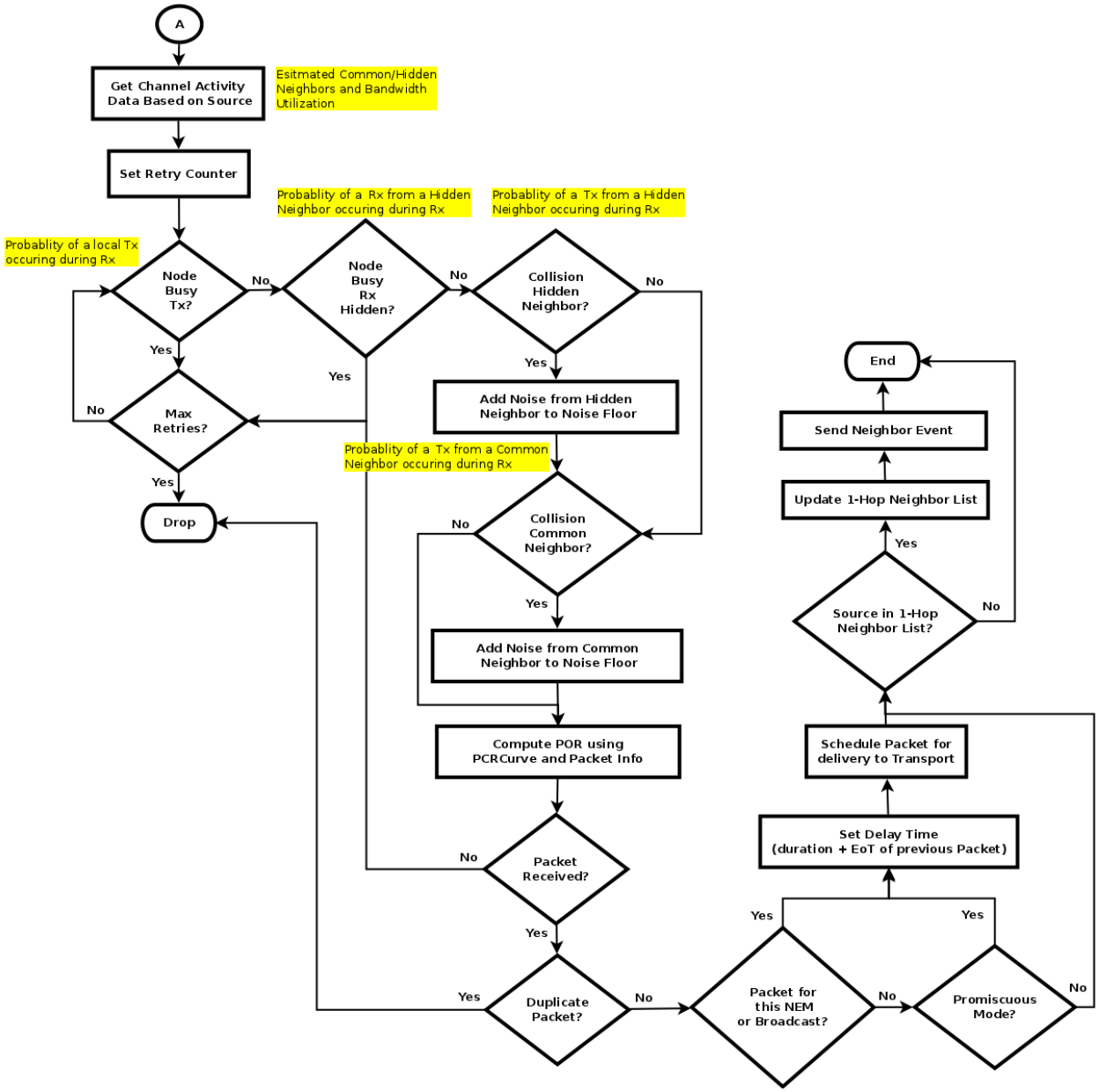


Figure 9.3: Part 2 - IEEE 802.11abg MAC Layer receive packet processing (upstream).

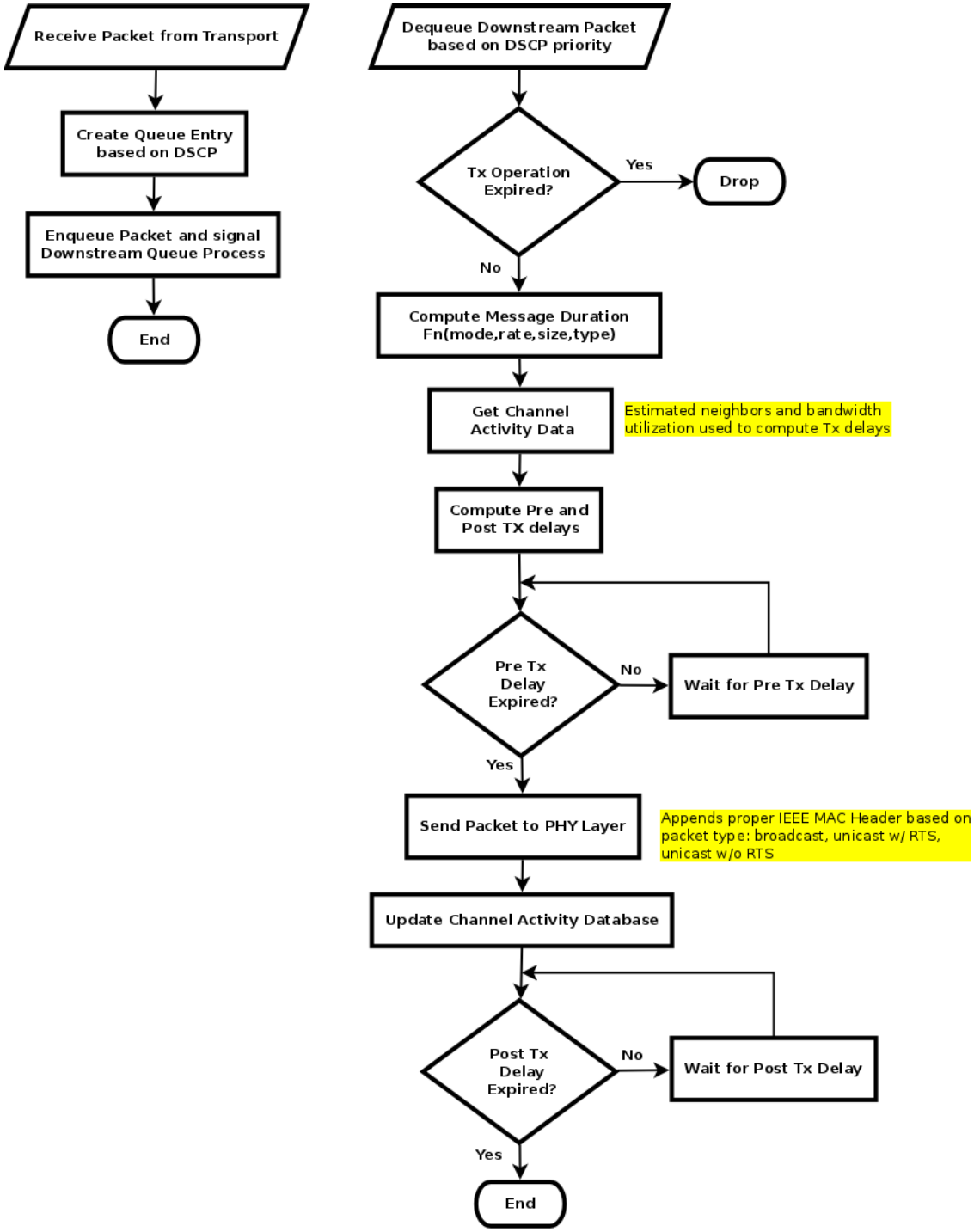


Figure 9.4: IEEE 802.11abg MAC Layer transmit packet processing (downstream) flow.

9.5 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

9.5.1 Demonstration 10

This demonstration deploys a four node centralized IEEE 802.11abg and RF Pipe emulation experiment illustrated in Figure 9.5. The goal of this demonstration is to become familiar with using the IEEE 802.11abg MAC Layer and to understand its noise processing capabilities.

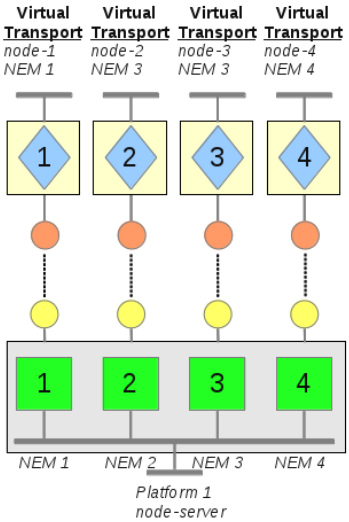


Figure 9.5: Demonstration 10 - Four node centralized IEEE802.11abg and RF Pipe NEM deployment.

In this demonstration NEMs 1 and 2 are IEEE 802.11abg instances and NEMs 3 and 4 are RF Pipe instances. All NEMs are using the same frequency in order to demonstrate the noise processing features of the IEEE 802.11abg MAC Layer.

Six MGEN flows are used to illustrate the noise processing functionality:

1. NEM 1 sends 10.24Kbps (128 byte packet x 10 per second) to NEM 2. With no noise 100% completion will result.

2. NEM 1 sends 81.92Kbps (1024 byte packet x 10 per second) to NEM 2. With no noise 100% completion will result.

3. NEM 1 sends 10.24Kbps (128 byte packet x 10 per second) to NEM 2 while NEM 3 sends 1.23Mbps (1024 byte packet x 150 per second) to NEM 4. NEM 3's transmissions will result in an increase in the Noise Floor and an SINR of -6db. This will result in approximately 63.5% completion.

4. NEM 1 sends 81.92Kbps (1024 byte packet x 10 per second) to NEM 2 while NEM 3 sends 1.23Mbps (1024 byte packet x 150 per second) to NEM 4. The change in packet size will result in a completion rate of approximately 2.6%.

5. NEM 1 sends 10.24Kbps (128 byte packet x 10 per second) to NEM 2 while NEM 4 sends 1.23Mbps (1024 byte packet x 150 per second) to NEM 3. Switching the transmitter to NEM 4 changes the SINR to -5dB. This will result in approximately 90% completion.

6. NEM 1 sends 81.92Kbps (1024 byte packet x 10 per second) to NEM 2 while NEM 4 sends 1.23Mbps (1024 byte packet x 150 per second) to NEM 3. This will result in approximately 45% completion.

9.5.1.1 Demonstration Procedure

1. Review the Demonstration 10 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/10
[emane@emanedemo 10] less platform.xml
```

2. Review the MGEN input files using your favorite editor.

```
[emane@emanedemo 12] less mgen*
```

3. Deploy the demonstration.

```
[emane@emanedemo 10]$ sudo ./lxc-demo-start.sh
```

4. This demonstration will run a short experiment and then display the results.

```
Waiting for experiment completion..........................done.

Flow   1      2      3      4      5      6
----   ------ ------ ------ ------ ------ ------
Tx 1   190    201    302    201    201    201
Rx 2   190    201    193    12     175    87
----   ------ ------ ------ ------ ------ ------
Comp%  100.00 100.00 63.91  5.97   87.06  43.28
```

5. Stop the demonstration.

```
[emane@emanedemo 10]$ sudo ./lxc-demo-stop.sh
```

9.5.1.2 Concept Review

1. How does the change in packet size between flow 3 and 4 and flow 5 and 6 affect the POR?
2. What completion rates would be expected if NEM 3 and NEM 4 switched frequencies?

Chapter 10

Comm Effect Shim Layer

The Comm Effect model provides the ability to control network impairments commonly provided via traditional commercial network emulators on a per link basis. The Comm Effect model is a Shim only unstructured NEM implementation.

10.1 Model Features

The Comm Effect model provides the ability to define the following network impairments:

- Loss: The percentage of packets that will be dropped utilizing a uniform loss distribution model.
- Latency: The average delay for a packet to traverse the network. The total delay is composed of a fixed and variable component. The fixed amount of the delay is defined via a `latency` configuration parameter and the variable amount via a `jitter` configuration parameter The jitter is determined randomly using a uniform random distribution model around +/- `jitter`. The randomly generated jitter value is then added to the fixed latency to determine the total delay.
- Duplicates: The percentage of packets that will be duplicated at the receiver.
- Unicast Bitrate: The bitrate for packets destined for the NEM or handled in promiscuous mode.
- Broadcast Bitrate: The bitrate for packets destined for the NEM broadcast address.

The network impairments defined above can be controlled via two mechanisms: Comm Effect Events and static filter based impairments. See Chapter 16 Comm Effect Event Generator on page 141 for more information on event based network impairments.

10.2 Configuration Parameters

10.2.1 `defaultconnectivity`

Defines the default communication effects of all NEMs at start up prior to receiving any Comm Effect events. All filter rules, if any, are still processed regardless of whether `defaultconnectivity` is in use.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="defaultconnectivity" value="on">`

Parameter value description:

Value	Description
`on`	All NEMS are allowed to communication without first receiving comm effect event data
`off`	No NEMS are allowed to communication without first receiving comm effect event data

10.2.2 `filterfile`

The name of the Comm Effect filter file to load. The filter file must be a fully qualified URI (absolute file name).

Type: String
Range: N/A
Default: None
Count: 1
XML Format: `<param name="filterfile" value="file:///etc/emane/filterfile.xml">`

10.2.3 `groupid`

Defines the NEM Group Id which will be used to group NEMs by the assigned Id value. When an NEM is assigned to a group it can only receive traffic from other members of the same group regardless of communication effects. If set to `0` the NEM is not associated with an NEM Group. If set to a value greater than `0` the NEM is associated with the NEM Group of the same value.

Type: Unsigned 32 bit Integer
Range: [`0`, `4294967295`]
Default: None
Count: 1
XML Format: `<param name="groupid" value="0">`

10.2.4 `enablepromiscuousmode`

Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="enablepromiscuousmode" value="off">`

Parameter value description:

Value	Description
`on`	Send all packets up the stack to the Transport Layer
`off`	Send only broadcast/multicast and locally addressed unicast up

10.2.5 `enabletighttimingmode`

Determines whether transmission time of the packet will be factored in when calculating delivery scheduling time. When enabled, tight time sync will be required for all platforms in the EMANE deployment containing Comm Effect NEM instances.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="enabletighttimingmode" value="off">`

Parameter value description:

Value	Description
`on`	Factor in transmission time of the packet when calculating delivery scheduling time
`off`	Ignore transmission time of the packet when calculating delivery scheduling time

10.2.6 `receivebufferperiod`

Specify the max sum of buffering time in seconds for packets received from an NEM. The buffering interval for each packet is determined by the bitrate for the source NEM and packet size. Packets are then placed in a timed queue based on this interval and any packets that would cause the receive buffer period to be exceeded are discarded. A value of `0.0` disables the limit and allows all received packets to stack up in the queue.

Type: Float
Range: [`0.0`, `MAX_FLOAT`]
Default: `1.0`
Count: 1
XML Format: `<param name="receivebufferperiod" value="1.0">`

10.3 Static Filters

In addition to the event based network impairments, the Comm Effect model provides the ability to define static filters to control network impairments. Filters are defined via an XML configuration file and have the following characteristics:

1. The filter file used by a given NEM within the emulation is identified at initialization time via the NEM's `filterfile` configuration parameter. Each filter defined in the `filterfile` is characterized by one or more `target` elements and a single `effect` element.

2. Currently only IPv4 Ethernet packet filter targets are supported. The `target` element has the following format within the filter XML file:

```
<target>
  <ipv4 src='0.0.0.0' dst='0.0.0.0' len='0' ttl='0' tos='0'>
    <udp sport='0' dport='0'/>
    <protocol type='0'/>
  </ipv4>
</target>
```

3. All of the IPv4 attributes are optional:

   (a) `src` - Source address in IPv4 header. Valid range 0.0.0.0 - 255.255.255.255, where 0.0.0.0 implies don't care.

   (b) `dst` - Destination address in IPv4 header. Valid range 0.0.0.0 - 255.255.255.255, where 0.0.0.0 implies don't care.

   (c) `len` - Total length in IPv4 header. Valid range 0 - 65535, where 0 implies don't care.

   (d) `ttl` - Time to live in IPv4 header. Valid range 0 - 255, where 0 implies don't care.

   (e) `tos` - Type of Service/Differentiated Services in IPv4 header. Valid range 0 - 255, where 0 implies don't care.

   In addition, a filter can be defined via the IPv4 protocol field in the header. The communication protocol can be defined by a name or numerical value. Currently, `udp` is the only protocol that can be defined by name. All other protocols must be identified via numerical value.

   (f) `udp` - Used to identify UDP protocol by name. When using this mechanism to define the udp protocol, `sport` and/or `dport` can also be identified for the udp protocol header. The valid range for `sport` and `dport` are 0 to 65535, where 0 implies don't care.

```
<target>
  <ipv4 dst='224.1.2.3'>
    <udp sport='12345' dport='12346'/>
  </ipv4>
</target>
```

   (g) `protocol` - Used when identifying the communication protocol based on numerical value. The `type` attribute identifies the numerical value for the IPv4 communication protocol with a valid range from 0 to 255.

```
<target>
  <ipv4 dst='224.1.2.3'/>
    <protocol type='89'/>
  </ipv4>
</target>
```

4. Each filter is assigned static network impairments (loss, latency, jitter, duplicates, unicastbitrate and broadcastbitrate).

```
<effect>
  <loss>20</loss>;
  <duplicate>0</duplicate>
  <latency sec='0' usec='200000'/>
  <jitter sec='0' usec='0'/>
  <broadcastbitrate>1024</broadcastbitrate>
  <unicastbitrate>8096</unicastbitrate>
</effect>
```

   The `effect` element has the following format:

   (a) `loss` - The loss 0 to 100 in percentage to be applied to the packets that match the associated `target`.

(b) `duplicate` - The duplicates 0 to 100 in percentage to be applied to the packets that match the associated `target`.

(c) `latency` - The fixed average delay to be applied to the packets that match the associated `target`. `sec` - Seconds have a valid range 0 to 65535. `usec` -Microseconds have a valid range 0 to 999999.

(d) `jitter` - The random variation applied to the packets that match the associated `target`. `sec` - Seconds have a valid range 0 to 65535. `usec` - Microseconds have a valid range 0 to 999999.

(e) `unicastbitrate` - The bitrate (bps) applied to packets addressed to the NEM or received in promiscuous mode matching the associated `target`. The bitrate has a valid range from 0 meaning unused to max unsigned 64 bit number.

(f) `broadcastbitrate` -The bitrate (bps) applied to packets addressed to the NEM broadcast address matching the associated `target`. The bitrate has a valid range from 0 meaning unused to max unsigned 64 bit number.

5. The filters and their associated impairments are defined at initialization and cannot be altered during emulation.

6. Filter ordering determines the network impairment and as such, more specific filters should be defined first. Each received packet is evaluated against the defined filters in order and the first match determines the impairment to be applied. For example, in the sample filter file shown in Listing 10.1, a packet as it is received by a node will be evaluated against each of the four filters (OSPF, TOS, UDP, DEFAULT) in order and the respective `effect` will be applied based on the first match.

   It should be noted that the inclusion of the `DEFAULT` filter should only be used when Comm Effect events are not being utilized since the filters take precedence. When filters are used in conjunction with Comm Effect events, the event driven impairments serve as the default effect for all packets that do not match a filter `target`.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE commeffect system "file:///usr/share/emane/commefect/dtd/commeffectfilters.dtd">
<commeffect>
<filter>
  <description>OSPF Packets</description>
  <target>
    <ipv4>
      <protocol type="89"/>
    </ipv4>
  </target>
  <effect>
    <loss>0</loss>
    <duplicate>0</duplicate>
    <latency sec='0' usec='0'/>
    <jitter sec='0' usec='0'/>
  </effect>
</filter>

<filter>
  <description>TOS (Type of Service) = 192</description>
  <target>
    <ipv4 tos='192'>
    </ipv4>
  </target>
  <effect>
    <loss>10</loss>
    <duplicate>150</duplicate>
    <latency sec='0' usec='100000'/>
    <jitter sec='0' usec='0'/>
  </effect>
</filter>

<filter>
  <description>UDP Multicast (destination address = 224.1.2.3 and destination port = 12345)</description>
  <target>
    <ipv4 dst='224.1.2.3'>
```

```
      <udp dport='12345'/>
    </ipv4>
  </target>
  <effect>
    <loss>20</loss>
    <duplicate>0</duplicate>
    <latency sec='0' usec='200000'/>
    <jitter sec='0' usec='0'/>
  </effect>
</filter>
<filter>
  <description>DEFAULT:  All Other Packets</description>
  <target/>
  <effect>
    <loss>40</loss>
    <duplicate>30</duplicate>
    <latency sec='0' usec='600000'/>
    <jitter sec='0' usec='100000'/>
    <broadcastcastbitrate>8096</broadcastcastbitrate>
    <unicastbitrate>8096</unicastbitrate>
  </effect>
</filter>
</commeffect>
```

Listing 10.1: Comm Effect Filter sample.

10.4 Packet Processing Flows

The Comm Effect Shim Layer receive packet (upstream) processing flowchart is shown in Figure 10.1 and Figure 10.2. The Comm Effect Shim Layer does not perform any relevant transmit packet (downstream) processing.

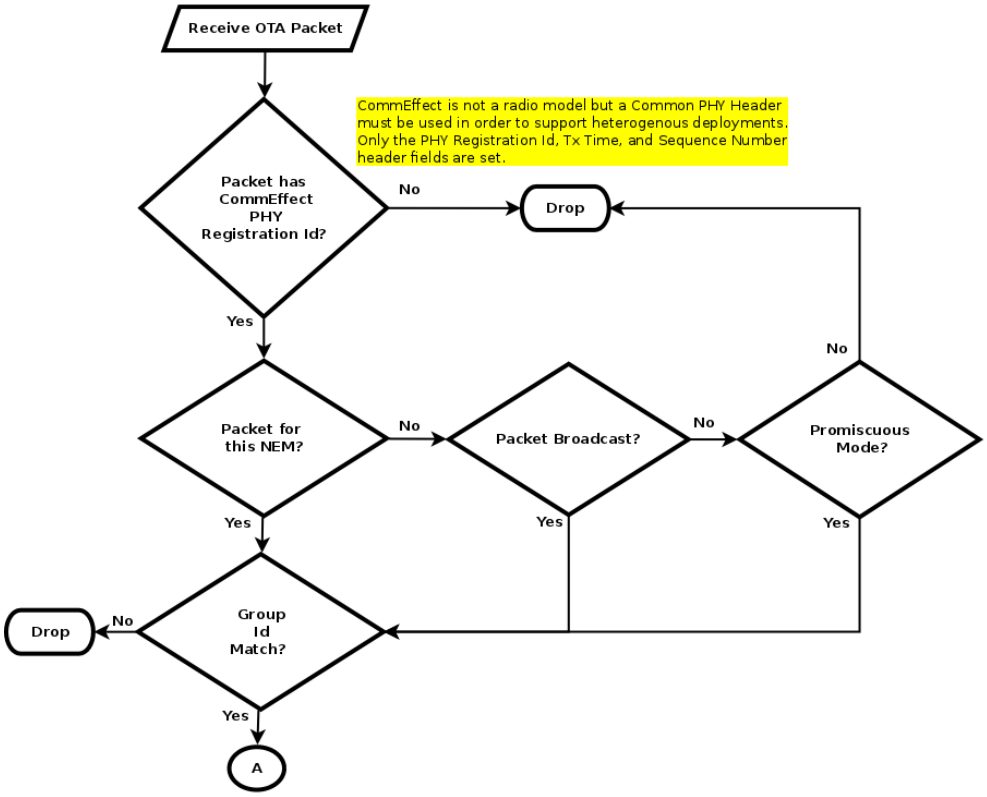


Figure 10.1: Part 1 - Comm Effect Shim Layer receive packet (upstream) processing flow.

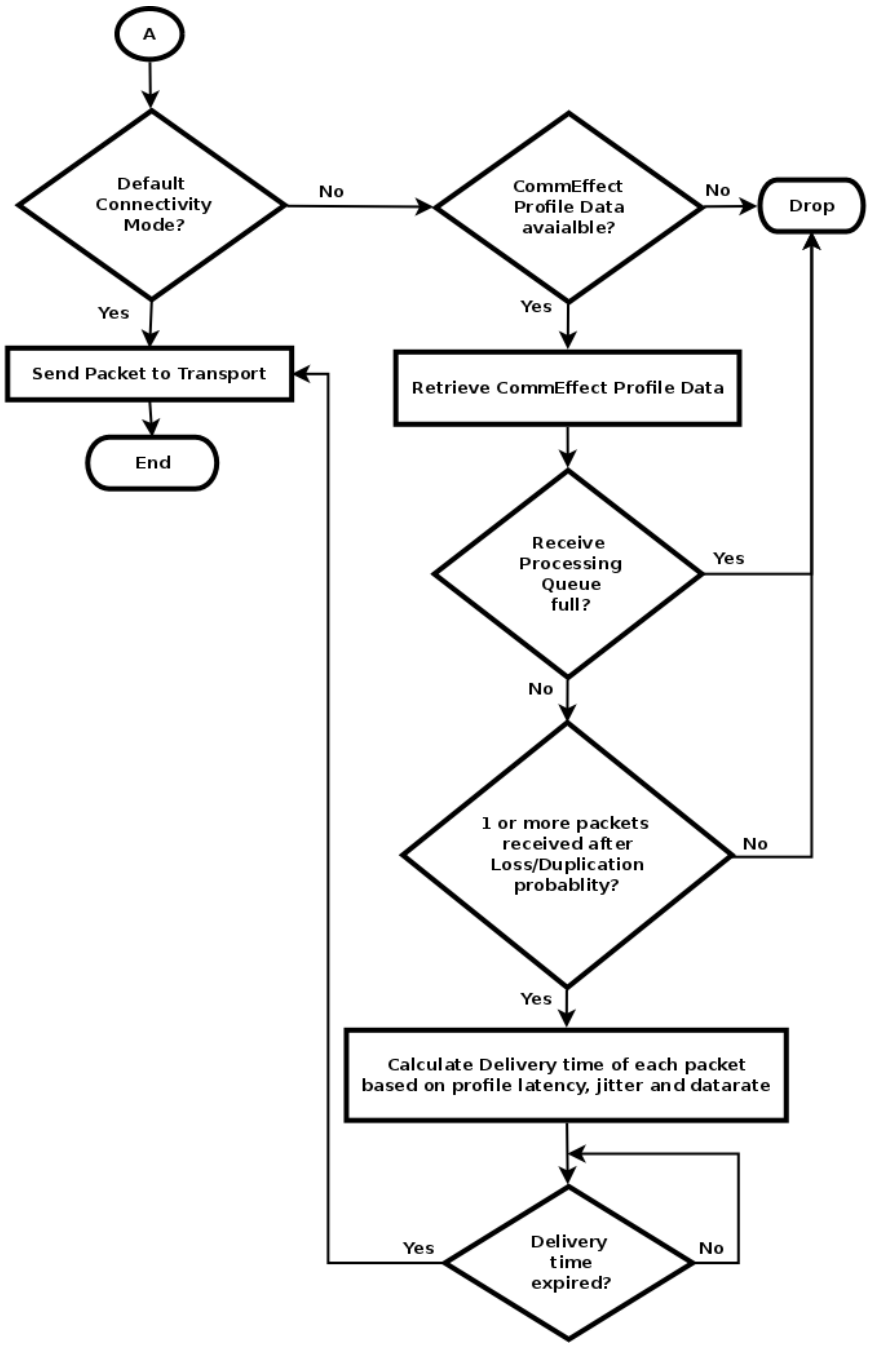


Figure 10.2: Part 2 - Comm Effect Shim Layer receive packet (upstream) processing flow.

10.5 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

10.5.1 Demonstration 11

This demonstration deploys a ten node distributed Comm Effect emulation experiment illustrated in Figure 10.3. The goal of this demonstration is to become familiar with using the Comm Effect Model and the Comm Effect Controller application.

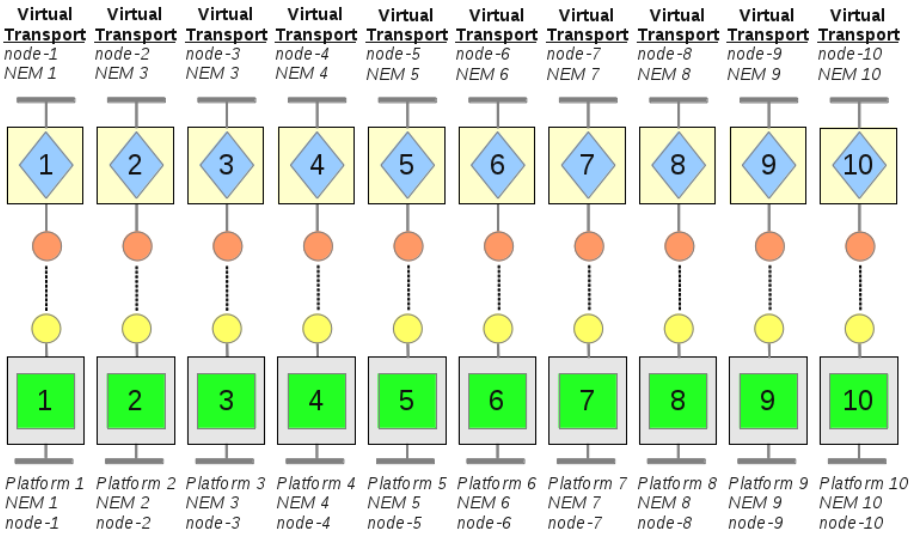


Figure 10.3: Demonstration 11 - Ten node distributed Comm Effect NEM deployment.

10.5.1.1 Demonstration Procedure

1. Review the Demonstration 11 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/11
[emane@emanedemo 11] less platform.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 11]$ sudo ./lxc-demo-start.sh
```

3. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

4. Open the Comm Effect Controller application. From the top panel select *Comm Effect Controller* from the same launcher used for *OLSR Link Viewer*.

5. Create a new project. *File* → *New*. See Figure 10.4.

   (a) Set the Project Name to `demo11`.

   (b) Set the number of nodes to `10`.

   (c) Verify Event Channel IP is `224.1.2.8`.

(d) Verify Event Channel Port is `45703`.

(e) Verify Event Channel NIC is `br0`.

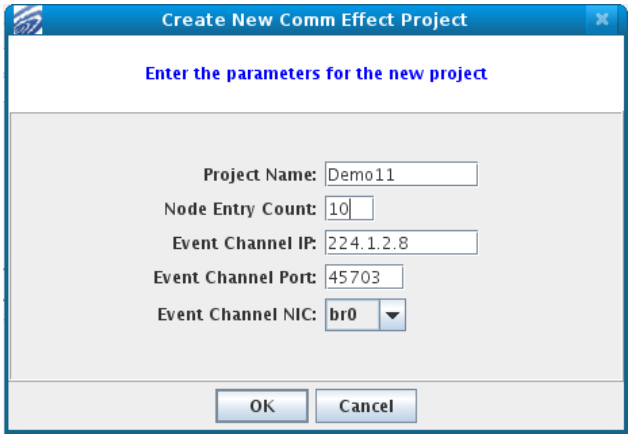


Figure 10.4: Demonstration 11 - Comm Effect Controller New Project dialog.

6. Experiment with modifying the Comm Effect scenario. Select an individual matrix entry to modify the effect between two nodes and try the *Quick Fill Panel* interface. See Figure 10.5 and Figure 10.6.

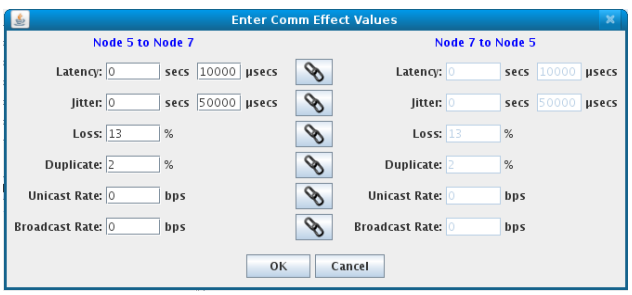


Figure 10.5: Demonstration 11 - Comm Effect Controller Effect Entry dialog.

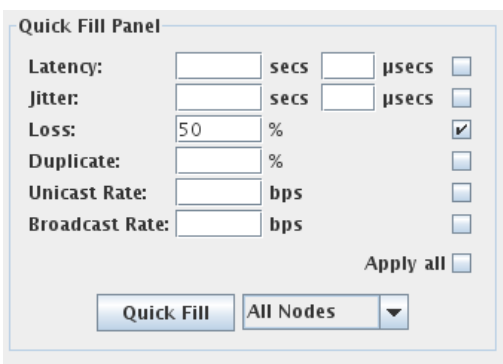


Figure 10.6: Demonstration 11 - Comm Effect Controller Quick Fill Panel.

7. Quick Fill all nodes to 50% loss and publish the scenario entry. *Quick Fill Panel: Loss 50 → Quick Fill → Enter Playback Mode → Play.*

8. Observe what happens to the network using the *OLSR Link Viewer*.

9. Stop the demonstration.

```
[emane@emanedemo 11]$ sudo ./lxc-demo-stop.sh
```

10.5.1.2 Concept Review

1. Why was `br0` used as the Event Channel device for this demonstration?
2. Is the Comm Effect Shim a radio model?
3. What are Comm Effect filters?

Part III

Transports

Chapter 11

Virtual Transport

The Virtual Transport creates a virtual interface for use as the emulation/application domain boundary. IP packets routed to the virtual device are encapsulated and transmitted to their respective NEM layer for downstream processing. Packets received over-the-air are processed up the NEM layer stack and transmitted to the NEM's respective virtual transport for injection back into the kernel IP stack. The newly created virtual interface is assigned an Ethernet address derived from the NEM Id associated with the transport using the following format: `02:02:00:00:XX:XX`, where `XX:XX` is the 16 bit NEM Id. This allows easy mapping of Ethernet MAC addresses to NEM Ids for unicast frames. Multicast and broadcast frames map to the NEM broadcast address `0xFFFF`.

11.1 Transport Features

Virtual Transport capabilities include the following:

- IPv4 and IPv6 Capable - Supports IPv4 and IPv6 virtual interface address assignments and packet processing.
- Flow Control - Supports flow control with a corresponding flow control capable NEM layer in order to provide feedback between the emulation stack and application domain socket queues.
- Virtual Interface Management - Supports configuring virtual interface addresses or can be configured to allow virtual interfaces to be managed externally, for example via DHCP.
- Raw Transport Interoperability - Supports interoperability with Raw Transport emulation/application domain boundaries using ARP caching to learn network/NEM Id associations.
- Bitrate Enforcement - Supports bitrate enforcement for use with models that do not limit bitrate based on emulation implementation.
- Broadcast Only Mode - Supports forced NEM broadcasting of all IP packet types: unicast, broadcast and multicast.

11.2 Configuration Parameters

11.2.1 `address`

Virtual device address. Supports IPv4 and IPv6.

Type: String
Range: N/A
Default: None
Count: 1
XML Format: `<param name="address" value="172.30.1.1">`

11.2.2 `arpcacheenable`

Enable ARP request/reply monitoring to map ethernet address to NEM.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="arpcacheenable" value="on">`

Parameter value description:

Value	Description
`on`	Enables ARP caching
`off`	Disabled APR caching

11.2.3 `arpmode`

Enable ARP on the virtual device.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="arpmode" value="on">`

Parameter value description:

Value	Description
`on`	Enables ARP on virtual device
`off`	Disables APR on virtual device

11.2.4 `bitrate`

Transport bitrate in Kbps. This is the total allowable throughput for the transport combined in both directions (upstream and downstream). A value of `0` disables the bitrate feature.

Type: Unsigned 64 bit Integer
Range: [`0`, `18446744073709551`]
Default: `0`
Count: 1
XML Format: `<param name="bitrate" value="0">`

11.2.5 `broadcastmode`

Broadcast all packets to all NEMs.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="broadcastmode" value="off">`

Parameter value description:

Value	Description
`on`	Enables broadcasting of all packets to all NEMs
`off`	Disables broadcasting of all packets to all NEMs

11.2.6 `device`

Virtual device name. Note: On OS X this ***must*** be `tap0`.

Type: String
Range: N/A
Default: `emane0`
Count: 1
XML Format: `<param name="device" value="emane0">`

11.2.7 `devicepath`

Path to the tap device.

Type: String
Range: N/A
Default (Linux): `/dev/net/tun`
Default (OS X): `/dev/tap0`
Default (Win32): `SYSTEM\CurrentControlSet\Control\Network\{4D36E972-E325-11CE-BFC1-08002BE10318}`
Count: 1
XML Format: `<param name="devicepath" value="/dev/net/tun">`

11.2.8 `flowcontrolenable`

Enables downstream traffic flow control with a corresponding flow control capable NEM layer. The **flowcontrolenable** parameter must match the setting of the corresponding NEM layer's **flowcontrolenable** parameter. See Section 11.3 Flow Control on page 112 for flow control details.

Type: Boolean
Range: [`off`, `on`]
Default: `off`
Count: 1
XML Format: `<param name="flowcontrolenable" value="off">`

Parameter value description:

Value	Description
`on`	Enable flow control with a flow control capable NEM layer
`off`	Disable flow control

11.2.9 `mask`

Virtual device network mask. Supports IPv4 and IPv6.

Type: String
Range: N/A
Default: None
Count: 1
XML Format: `<param name="mask" value="255.255.255.0"">`

11.3 Flow Control

The Virtual Transport supports flow control using a token based exchange mechanism performed in coordination with a corresponding NEM layer. A flow control token is a packet transmission unit where a single token represents permission for the transport to transmit a single packet downstream to a coordinating NEM layer. The standard EMANE distribution contains two NEM layers capable of performing flow control with the Virtual Transport: *RF Pipe MAC Layer* and *IEEE 802.11abg MAC Layer*. Flow control must be enabled on both the transport and the coordinating flow control capable NEM layer using the `flowcontrolenable` configuration parameter (See Sections 8.2.8 11.2.8 9.2.9).

The number of tokens available is specified using the `flowcontroltokens` configuration item available for both the RF Pipe Mac Layer (See Section 8.2.9) and the IEEE 802.11abg MAC Layer (See Section 9.2.10). Once started, a flow control enabled layer sends a control message to the Virtual Transport specifying the number of tokens available and then waits for the transport to acknowledge receipt of the token count. Any downstream packets received from the transport in the period between when the token count control message is sent and the acknowledgment is received are discarded.

When the Virtual Transport starts, it sends a control message to the flow control enabled layer requesting the current token count. No downstream packets are transmitted to the flow control enabled layer until the flow control token count control message is received. Once received, the transport will send an acknowledgment control message. This acknowledgment will satisfy the flow control enabled component in the

situation where it was started prior to the transport and was blocked waiting for a previous acknowledgment.

The Virtual Transport decrements its token count each time it sends a downstream packet. When the token count reaches zero no further packets are transmitted causing application socket queues to backup. The flow control enabled layer shadows the token count of the transport in order to detect when the transport has run out of tokens. Once available, the flow control enabled layer will send a flow control token count message restarting the process.

Using this method, either flow control component, the Virtual Transport or the coordinating layer, can restart any number of times and the token count will resync automatically.

11.4 Packet Processing Flows

The Virtual Transport receive packet (upstream) processing flowchart is shown in Figure 11.1. The Virtual Transport transmit packet (downstream) processing flowchart is shown in Figure 11.2.

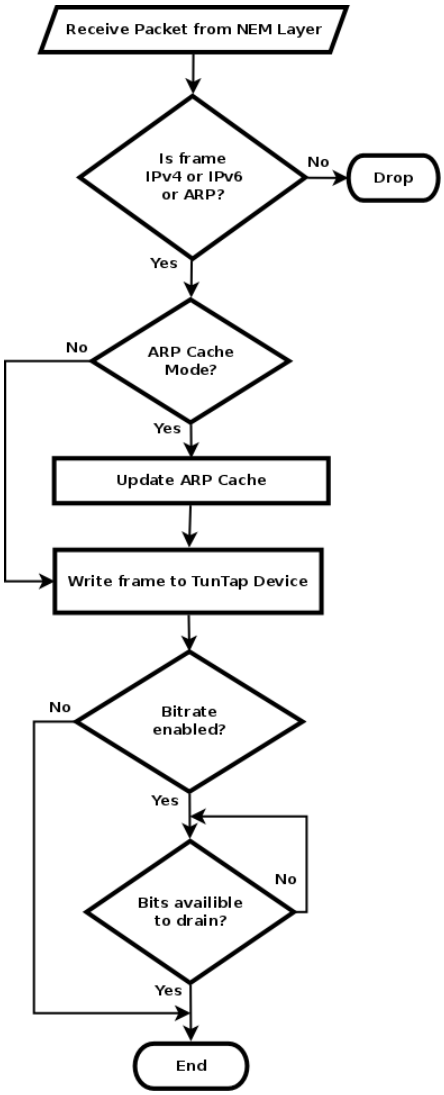


Figure 11.1: Virtual Transport receive packet (upstream) processing flow.

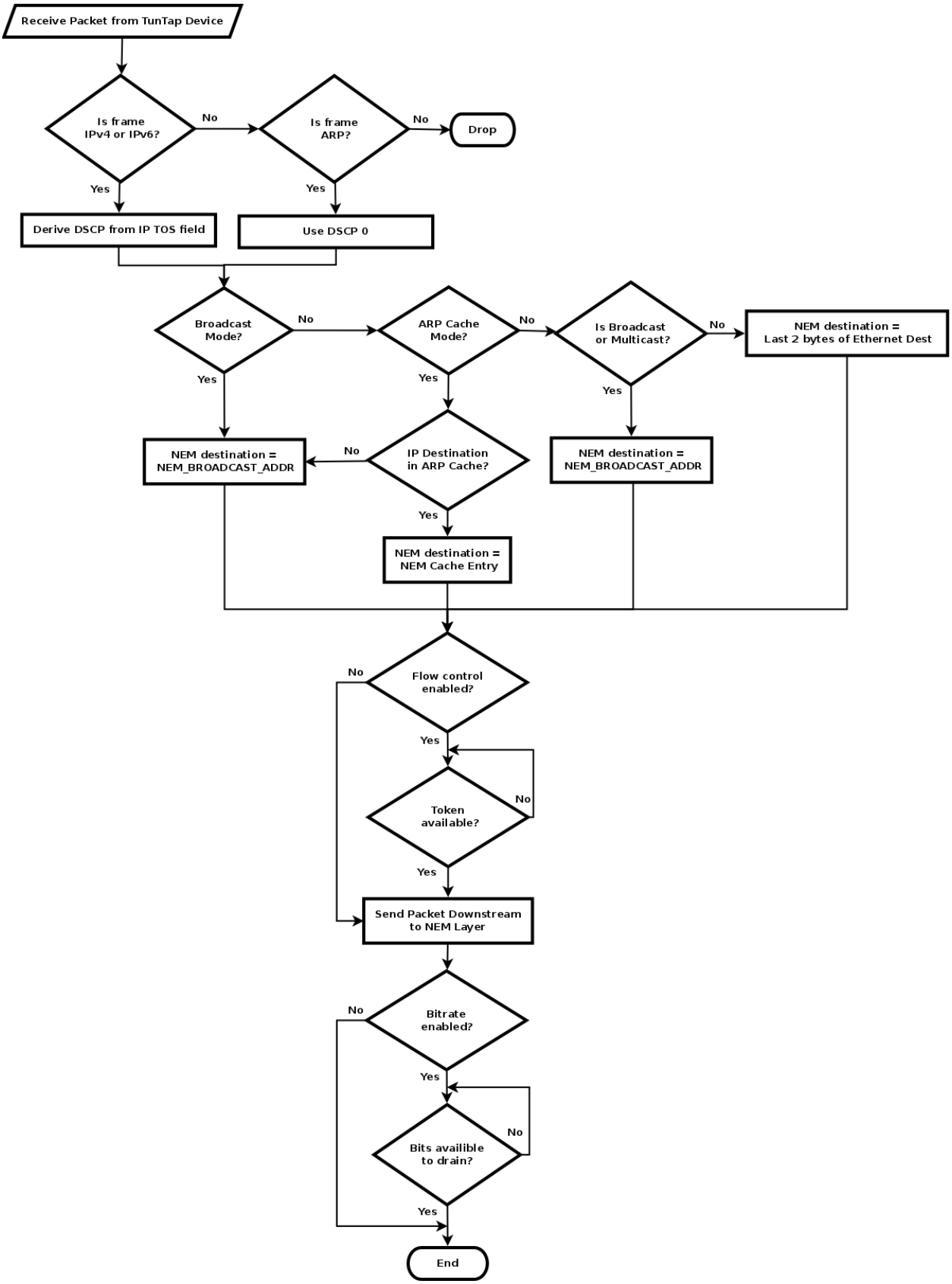


Figure 11.2: Virtual Transport transmit packet (downstream) processing flow.

11.5 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

11.5.1 Demonstration 12

This demonstration deploys a ten node centralized RF Pipe emulation experiment illustrated in Figure 11.3. The goal of this demonstration is to become familiar with using the Virtual Transport and its flow control capability.

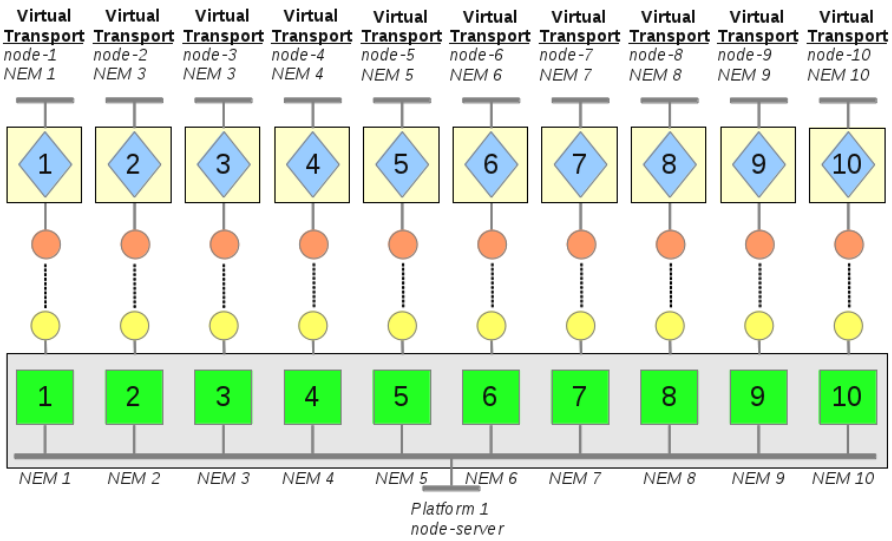


Figure 11.3: Demonstration 13 - Ten node centralized RF Pipe NEM deployment.

In this demonstration NEMs 1 - 5 have flow control enabled and NEMs 6 - 10 have flow control disabled. NEM 1 will send approximately 1.5Mbps worth of traffic to NEM 2 and NEM 6 will send at the same rate to NEM 7.

11.5.1.1 Demonstration Procedure

1. Review the Demonstration 12 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/12
[emane@emanedemo 12] less platform.xml
```

2. Review the MGEN input files using your favorite editor.

```
[emane@emanedemo 12] less mgen*
```

3. Deploy the demonstration.

```
[emane@emanedemo 12]$ sudo ./lxc-demo-start.sh
```

4. This demonstration will run a short experiment and then display the results.

```
Waiting for experiment completion.........................done.

Flow Control On  - Node 1 sent 2864 packets and Node 2 received 2864 packets

Flow Control Off - Node 6 sent 3516 packets and Node 7 received 2533 packets
```

5. Stop the demonstration.

```
[emane@emanedemo 12]$ sudo ./lxc-demo-stop.sh
```

11.5.1.2 Concept Review

1. What accounts for the difference in the number of packets sent and received by the two pairs of nodes?

Chapter 12

Raw Transport

The Raw Transport uses a specific network interface as the application/emulation boundary. IP packets read from the interface are encapsulated and transmitted to their respective NEM. Packets received OTA for upstream processing, either from another platform internal NEM or an NEM contained in another platform, are processed by the PHY Layer and MAC Layer and transmitted to the NEM's respective Raw Transport for transmission out the specified network interface.

12.1 Configuration Parameters

12.1.1 `bitrate`

Transport bitrate in Kbps. This is the total allowable throughput for the transport combined in both directions (upstream and downstream). A value of `0` disables the bitrate feature.

Type:	Unsigned 64 bit Integer
Range:	[`0`, `18446744073709551`]
Default:	`0`
Count:	1
XML Format:	`<param name="bitrate" value="0">`

12.1.2 `broadcastmode`

Broadcast all packets to all NEMs.

Type:	Boolean
Range:	[`off`, `on`]
Default:	`on`
Count:	1
XML Format:	`<param name="broadcastmode" value="off">`

Parameter value description:

Value	Description
`on`	Enables broadcasting of all packets to all NEMs
`off`	Disables broadcasting of all packets to all NEMs

12.1.3 `arpcacheenable`

Enable ARP request/reply monitoring to map Ethernet address to NEM.

Type: Boolean
Range: [`off`, `on`]
Default: `on`
Count: 1
XML Format: `<param name="arpcacheenable" value="on">`

Parameter value description:

Value	Description
`on`	Enables ARP caching
`off`	Disabled APR caching

12.1.4 `device`

Device to use as the raw packet entry point. Once a network device has been dedicated to a Raw Transport it should not be used for any other communication other than what should be routed into the emulation domain.

Type: String
Range: N/A
Default: N/A
Count: 1
XML Format: `<param name="device" value="eth0">`

12.2 Transport Interoperability

There is no guarantee that heterogeneous transports can be used in a given deployment. The Transport API is designed to allow transports to transmit opaque data to and from their respective NEM stacks. The format of the data is implementation dependent. It is up to individual transport implementations to take the appropriate steps necessary to allow for compatibility with transports of similar types.

Both the Virtual Transport and Raw Transport route Ethernet frames. There are two configuration options that will allow both transports to communicate with each other.

The first option is to use the ARP cache feature of both transports. When enabled, each transport will peek at all ARP response packets that are sent upstream from their respective NEM stacks. Using the information contained in the response and the NEM Id of the responder the transports will build a cache of destination addresses and NEM Ids. This will allow the transports to determine the NEM destination Id for all unicast data messages.

Parameter	Virtual Transport	Raw Transport
`arpcacheenable`	`on`	`on`

The second option is to use the broadcast only feature of the Virtual Transport. When ARP cache mode is disabled, the Raw Transport will send all unicast packets to the broadcast NEM Id address. No attempt is made to determine the NEM Id associated with the unicast destination address. Each inbound transport will attempt to deliver the unicast Ethernet frame and the kernel will drop all packets that do not match the host. The Virtual Transport must be configured to operate in the same manner.

Parameter	Virtual Transport	Raw Transport
`arpcacheenable`	`off`	`off`
`broadcastmode`	`on`	N/A

12.3 Packet Processing Flows

The Raw Transport receive packet (upstream) processing flowchart is shown in Figure 12.1. The Raw Transport transmit packet (downstream) processing flowchart is shown in Figure 12.2.

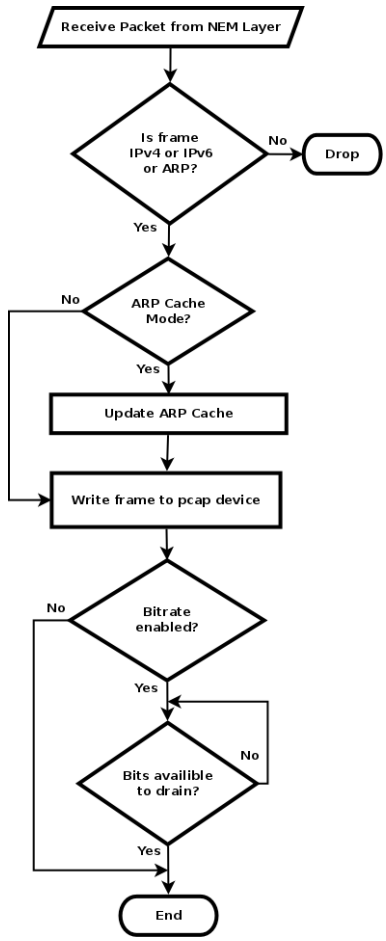


Figure 12.1: Raw Transport receive packet (upstream) processing flow.

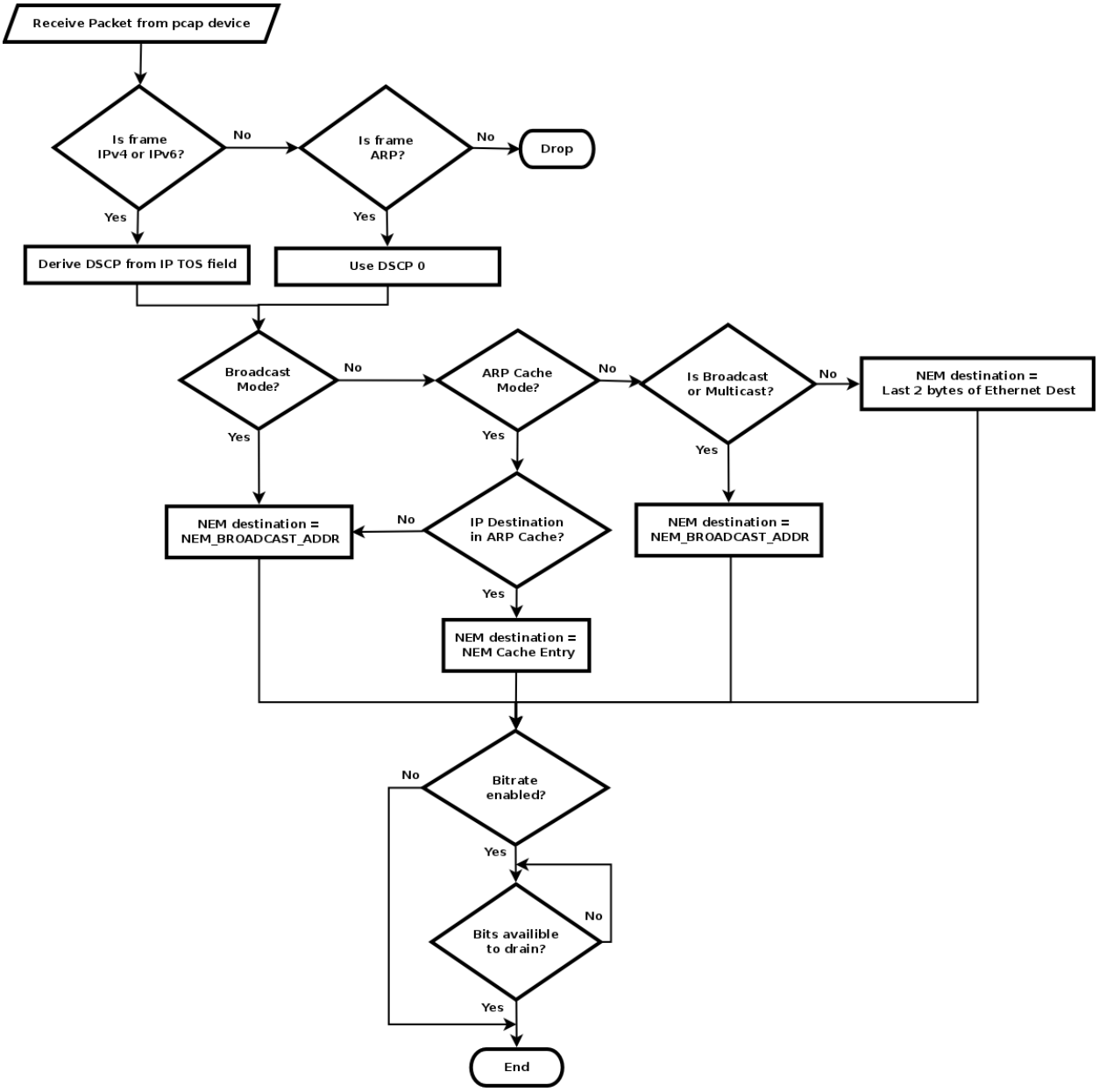


Figure 12.2: Raw Transport transmit packet (downstream) processing flow.

12.4 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

12.4.1 Demonstration 13

This demonstration deploys a ten node centralized RF Pipe emulation experiment illustrated in Figure 12.3. The goal of this demonstration is to become familiar with using the Raw Transport and the Pathloss Controller application.

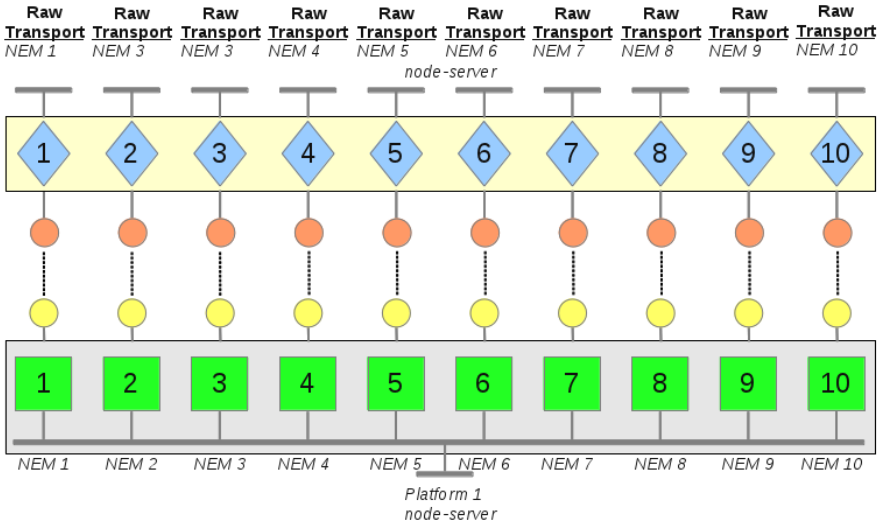


Figure 12.3: Demonstration 13 - Ten node centralized RF Pipe NEM deployment.

12.4.1.1 Demonstration Procedure

1. Review the Demonstration 13 platform XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/13
[emane@emanedemo 13] less platform.xml
```

2. Review the Demonstration 13 transport daemon XML using your favorite editor.

```
[emane@emanedemo 13] less transportdaemon1.xml
```

3. Deploy the demonstration.

```
[emane@emanedemo 13]$ sudo ./lxc-demo-start.sh -t transportdaemon1.xml
```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Connect to virtual `node-1`.

```
[emane@emanedemo 13]$ ssh node-1
```

6. Review the running processes.

```
[emane@node-1 ~]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S+     0:00 /usr/lib/lxc/lxc-init -- /tmp/lxc-node/13/1/init.sh
```

```
    5 ?        Ssl    1:06 /usr/local/bin/olsrd -f /home/emane/demonstration/13/ols
    8 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/tmp/lxc-node/13/1/run/sshd
    9 ?        Ss     0:00 sshd: emane [priv]
   11 ?        S      0:00 sshd: emane@pts/0
   12 pts/0    Ss     0:00 -bash
   72 pts/0    R+     0:00 ps ax
```

7. Review `node-1`'s network interface configuration.

```
[emane@node-1 ~]$ ifconfig
bmf0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:10.100.0.1  P-t-P:10.100.0.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)

eth0      Link encap:Ethernet  HWaddr 02:01:00:00:00:01
          inet addr:10.99.0.1  Bcast:10.99.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1:ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:266 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3164 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19089 (18.6 KiB)  TX bytes:1023885 (999.8 KiB)

eth1      Link encap:Ethernet  HWaddr 02:01:01:00:00:01
          inet addr:10.100.0.1  Bcast:10.100.0.255  Mask:255.255.255.0
          inet6 addr: fe80::1:1ff:fe00:1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:30160 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3999 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3801600 (3.6 MiB)  TX bytes:493342 (481.7 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:5956 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5956 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:297800 (290.8 KiB)  TX bytes:297800 (290.8 KiB)
```

8. Review `node-1`'s routing table.

```
[emane@node-1 ~]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
10.99.0.0       0.0.0.0         255.255.255.0   U     0      0        0 eth0
10.100.0.0      0.0.0.0         255.255.255.0   U     0      0        0 eth1
10.100.0.2      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.3      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.4      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.5      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.6      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.7      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.8      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.9      0.0.0.0         255.255.255.255 UH    1      0        0 eth1
10.100.0.10     0.0.0.0         255.255.255.255 UH    1      0        0 eth1
224.0.0.0       0.0.0.0         240.0.0.0       U     0      0        0 bmf0
```

9. Ping another radio using the *radio-NEMID* host naming convention.

```
[emane@node-1 ~]$  ping -c 5 radio-2
PING radio-2 (10.100.0.2) 56(84) bytes of data.
64 bytes from radio-2 (10.100.0.2): icmp_req=1 ttl=64 time=5.67 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=2 ttl=64 time=3.30 ms
```

```
64 bytes from radio-2 (10.100.0.2): icmp_req=3 ttl=64 time=6.87 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=4 ttl=64 time=3.08 ms
64 bytes from radio-2 (10.100.0.2): icmp_req=5 ttl=64 time=17.5 ms

--- radio-2 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4009ms
rtt min/avg/max/mdev = 3.086/7.287/17.506/5.306 ms
```

10. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

11. Open the Pathloss Controller application. From the top panel select *Pathloss Controller* from the same launcher used for *OLSR Link Viewer*.

12. Create a new project. *File* → *New*. See Figure 12.4.

    (a) Set the Project Name to `demo13`.

    (b) Set the Nodes Entry Count to `10`.

    (c) Set the No Loss Threshold to `90`.

    (d) Set the Full Loss Threshold to `110`.

    (e) Verify Event Channel IP is `224.1.2.8`.

    (f) Verify Event Channel Port is `45703`.

    (g) Verify Event Channel NIC is `lo`.

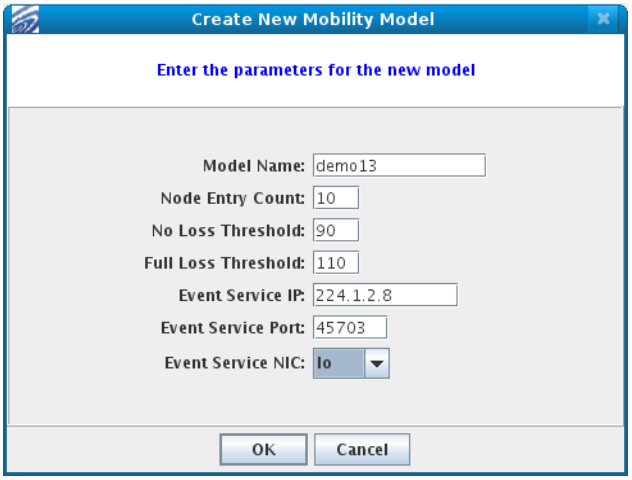


Figure 12.4: Demonstration 13 - Pathloss Controller New Project dialog.

13. Experiment with modifying the pathloss scenario. Select an individual matrix entry to modify the pathloss between two nodes and try the *Quick Fill Panel* interface. See Figure 12.5 and Figure 12.6.

14. Quick Fill all nodes to 100dB pathloss and publish the scenario entry. *Quick Fill Panel: 100* → *Quick Fill* → *Enter Playback Mode* → *Play*.

15. Observe what happens to the network using the *OLSR Link Viewer*.

16. Stop the demonstration.

```
[emane@emanedemo 13]$ sudo ./lxc-demo-stop.sh
```

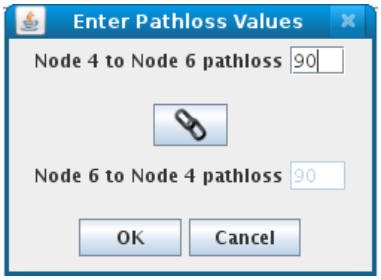


Figure 12.5: Demonstration 13 - Pathloss Controller Pathloss Entry dialog.

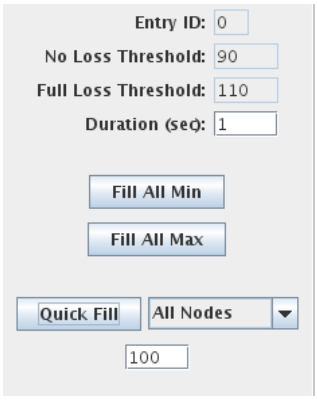


Figure 12.6: Demonstration 13 - Pathloss Controller Quick Fill Panel.

12.4.1.2 Concept Review

1. What are the EMANE components associated with this deployment type?
2. What are the possible usecases for using the Raw Transport?
3. Why must the interface assigned to a Raw Transport not be used for any other type of communication?

Part IV

Events

Chapter 13

Mitre Mobility Model Event Generator

The Mitre Mobility Model Event Generator creates pathloss and/or location events from input files in Mitre Mobility Format. The file contains pathloss and location information between nodes on one second boundaries.

13.1 Configuration Parameters

13.1.1 `inputfileformat`

Absolute file name of mobility file. One or more files may be specified by using a `printf()` style convention using the `inputfilecount` configuration item as an index.

Type:	String
Range:	N/A
Default:	None
Count:	1
XML Format:	`<param name="inputfileformat" value="path_loss_matrix_%02d.txt>"`

13.1.2 `inputfilecount`

Total number of input files. If the `inputfilename` contains a `printf()` style expression the count will be used as an index when creating the file names.

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="inputfilecount" value="1"`

13.1.3 `totalnodes`

Total number of nodes whose data is contained in the mobility files.

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="totalnodes" value="70"/>`

13.1.4 `maxnemidpresent`

Maximum NEM Id present in the emulation experiment. This value is used to reduce the number of events generated when a subset of nodes from the larger mobility data are used.

Type:	Unsigned 16 bit Integer
Range:	[`0`, `65534`]
Default:	None
Count:	1
XML Format:	`<param name="maxnemidpresent" value="12"/>`

13.1.5 `repeatcount`

The number of times the mobility data should be parsed and events generated. A `repeatcount` of `1` simply means to process the file once, generating events, and stop when the file is complete. A value greater than `1` allows you to process the data `repeatcount` times. A value of `0` will process the data repeatedly, restarting indefinitely.

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="repeatcount" value="1"/>`

13.1.6 `utmzone`

The UTM zone that corresponds to the UTM position information contained in the mobility data. This is required to convert the data when generating location events. A limitation of the Mitre Mobility Model format is that position data cannot cross UTM zones.

Type:	String
Range:	N/A
Default:	None
Count:	1
XML Format:	`<param name="utmzone" value="18T"/>`

13.1.7 `entryreplay`

Specify one or more space separated *time:count* pairs. `entryreplay` effectively allows you to hold at certain mobility entries for a specified amount of time. For example a value of "0:120 3600:1800" would use the data at mobility entry T0 for 2 minutes, sending out the T0 events 120 times and use the data at T3600 for 30 minutes.

Type: String
Range: N/A
Default: None
Count: 1
XML Format: `<param name="entryreplay" value="0:120 3600:1800"/>`

Parameter value format description:
`<Time>:<Count> [<Time>:<Count>]...`

Name	Description	Range
Time	Corresponds to the second of data contained in the mobility model	[0, 4294967295]
Count	Represents the number of times you want to replay that data	[1, 4294967295]

13.1.8 `publishpathlossevents`

Create/Publish pathloss events from mobility model input files.

Type: Boolean
Range: [off, on]
Default: on
Count: 1
XML Format: `<param name="publishpathlossevents" value="on"/>`

13.1.9 `publishlocationevents`

Create/Publish location events from mobility model input files.

Type: Boolean
Range: [off, on]
Default: on
Count: 1
XML Format: `<param name="publishlocationevents" value="on"/>`

13.2 Mitre Mobility Model Format

Mitre Mobility Model Format is an ASCII text file containing a single entry for every pair of nodes in the mobility scenario on one second boundaries. Listing 13.1 shows a sample mobility file. For each second, the number of entries required to completely describe the connectivity for N nodes can be represented by the

following equation:

$$\sum_{i=1}^{N-1} N - i$$

The Mitre Mobility Model text file contains eleven columns as defined below:

1. Time in Seconds
2. Node Target A
3. Node Target B
4. Pathloss between nodes. A single value denotes symmetric pathloss. Two values separated by a '/' denotes asymmetric pathloss. Where the first value is the pathloss from Node A to Node B and the second value is the pathloss from Node B to Node A.
5. Distance between nodes in meters
6. Node A UTM X position
7. Node A UTM Y position
8. Node A Antenna Height (altitude) in meters
9. Node B UTM X position
10. Node B UTM Y position
11. Node B Antenna Height (altitude) in meters

```
0   1   2 102      13  540654 4431315   3  540663 4431325   3
0   1   3 103/301  13  540654 4431315   3  540663 4431325   3
0   1   4 104/401  13  540654 4431315   3  540663 4431325   3
0   1   5 105/501  13  540654 4431315   3  540663 4431325   3
0   2   3 203/302  13  540654 4431315   3  540663 4431325   3
0   2   4 204/402  13  540654 4431315   3  540663 4431325   3
0   2   5 205/502  13  540654 4431315   3  540663 4431325   3
0   3   4 304/403  13  540654 4431315   3  540663 4431325   3
0   3   5 305/503  13  540654 4431315   3  540663 4431325   3
0   4   5 405/504  13  540654 4431315   3  540663 4431325   3
```

Listing 13.1: Mitre Mobility file sample.

13.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

13.3.1 Demonstration 14

This demonstration deploys a ten node distributed IEEE 802.11abg NEM emulation experiment illustrated in Figure 4.1. The goal of this demonstration is to become familiar with the Mitre Mobility Model Event Generator.

13.3.1.1 Demonstration Procedure

1. Review the Demonstration 14 Event Service XML using your favorite editor.

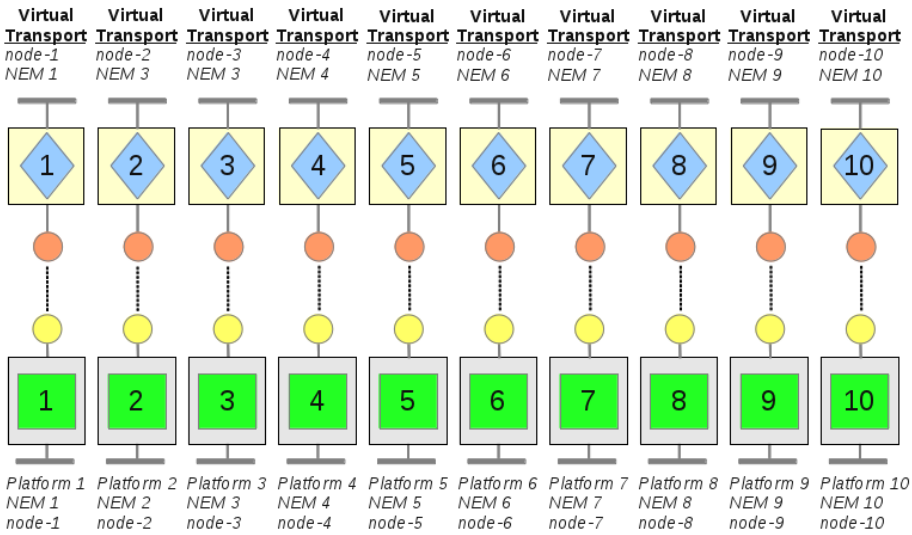


Figure 13.1: Demonstration 14 - Ten node distributed IEEE 802.11abg NEM deployment.

```
[emane@emanedemo ~] cd /home/emane/demonstration/14
[emane@emanedemo 14] less eventservice.xml
```

2. Review the Demonstration 14 Mitre Mobility Generator XML using your favorite editor.

```
[emane@emanedemo 14] less mitremobilitygenerator.xml
```

3. Deploy the demonstration.

```
[emane@emanedemo 14]$ sudo ./lxc-demo-start.sh
```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Stop the demonstration.

```
[emane@emanedemo 14]$ sudo ./lxc-demo-stop.sh
```

13.3.1.2 Concept Review

1. Why is the UTM zone a required configuration item?
2. What is the purpose of the `entryreplay` parameter?

Chapter 14

Emulation Script Event Generator

The Emulation Script Event Generator creates location events from input files in the Emulation Script Format. Emulation Script Format was developed by the Protean Research Group at Naval Research Laboratory [Protean Research Group, 2010]. The file contains location information between nodes on specific time boundaries.

14.1 Configuration Parameters

14.1.1 `inputfile`

Absolute file name of the emulation script mobility input file. One or more files may be specified by using this parameter multiple times and modifying the value attribute accordingly. Files are processed in the order they appear in the XML.

Type:	String
Range:	N/A
Default:	None
Count:	Unlimited
XML Format:	`<param name="inputfile" value="path_loss_matrix_00.xml/>"`

14.1.2 `totalnodes`

Total number of nodes whose data is contained in the mobility files.

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="totalnodes" value="70"/>`

14.1.3 `repeatcount`

The number of times the mobility data should be parsed and events generated. A `repeatcount` of `1` simply means to process the file once, generating events, and stop when the file is complete. A value greater than

`1` allows you to process the data `repeatcount` times. A value of `0` will process the data repeatedly, restarting indefinitely.

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="repeatcount" value="1"/>`

14.1.4 `schemalocation`

Specifies the location of the schema file used to validate files specified via `inputfile` parameters.

Type:	String
Range:	N/A
Default:	None
Count:	1
XML Format:	`<param name="schemalocation" value="http://configserver/schema/EmulationScriptSchema.xsd/>"`

14.1.5 Emulation Script Data Format

Emulation Script Format is an XML file containing `Event` elements. Each element contains two child elements:

1. `time` - Specifying the amount of time in seconds that has elapsed since the start (initial event).
2. `Node` - Specifying the Id (using an attribute) and the location (using a child element) of a given node in the network.

See [Protean Research Group, 2010] for a more detailed description of Emulation Script Format.

```
<?xml version="1.0" encoding="UTF-8"?>
<EmulationScript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="EmulationScriptSchema.xsd">
  <Event>
    <time>0</time>
    <Node id="1">
      <location>40.0310751857906,-74.5235179912516,3</location>
    </Node>
    <Node id="2">
      <location>40.0311648464297,-74.5234118838455,3</location>
    </Node>
  </Event>
  <Event>
    <time>1</time>
    <Node id="1">
      <location>40.0311648464297,-74.5234118838455,3</location>
    </Node>
    <Node id="2">
      <location>40.031227237558,-74.5232473639998,3</location>
    </Node>
  </Event>
</EmulationScript>
```

Listing 14.1: Emulation Script Data file sample.

14.2 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

14.2.1 Demonstration 15

This demonstration deploys a ten node distributed RP Pipe NEM emulation experiment illustrated in Figure 14.1. The goal of this demonstration is to become familiar with the Emulation Script Generator.

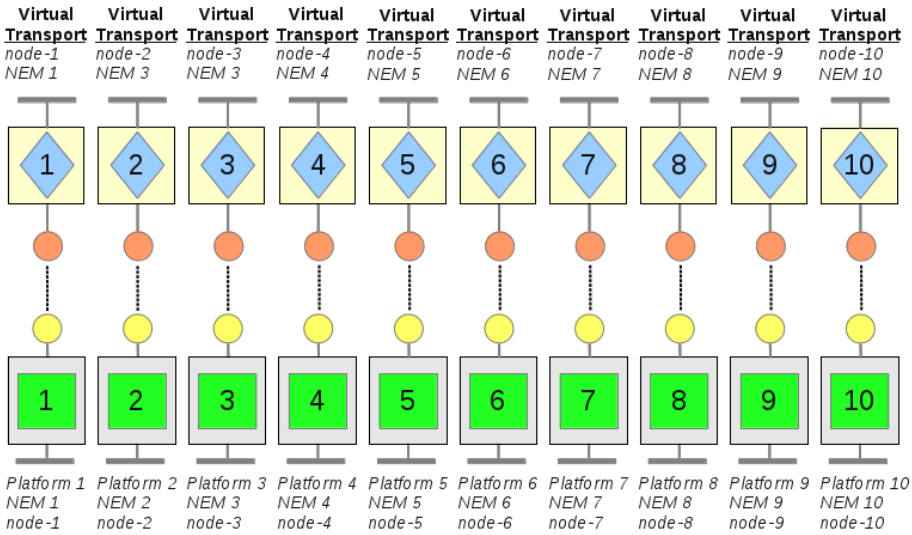


Figure 14.1: Demonstration 15 - Ten node distributed RF Pipe NEM deployment.

14.2.1.1 Demonstration Procedure

1. Review the Demonstration 15 Event Service XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/15
[emane@emanedemo 15] less eventservice.xml
```

2. Review the Demonstration 15 Emulation Script Generator XML using your favorite editor.

```
[emane@emanedemo 15] less emulationscriptgenerator.xml
```

3. Deploy the demonstration.

```
[emane@emanedemo 15]$ sudo ./lxc-demo-start.sh
```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Stop the demonstration.

```
[emane@emanedemo 15]$ sudo ./lxc-demo-stop.sh
```

Chapter 15

Emulation Event Log Generator

The Emulation Event Log (EEL) Generator creates EMANE events from input files in EEL Format. EEL format was developed by the Protean Research Group at Naval Research Laboratory.

> This (EEL) file format is a linear, text file format that can be used to convey the value of properties or parameters identified by a keyword. This file allows for "events" affecting modeling system components and/or their properties that occur over time to be expressed (e.g. as a file format to "drive" event generation over time) or to be logged (e.g. as a log file format for "capturing" run-time events for replay or post-processing analysis). The EEL file is a text format consisting of lines (a.k.a. "sentences") that each contain a timestamp, some "module identifier" and an event type "keyword" that implies the format and interpretation of the remainder of the line. The "keyword" approach allows a mixture of event types to be included within an EEL file and expanded over time as needed. Tools that process EEL file may choose to process a subset of event types as needed. The format also lends itself to simple filtering by event type, module identifier, etc using commonly-available tools (e.g., "grep", etc).
>
> The linear, time-ordered format also allows it to be incrementally processed such that even very bulky files can be handled as needed. Note that, in the interest of compactness, it is typically expected that the events included will represent "deltas" (i.e. changes) to any previously established state. However, one could choose to have each time epoch (or at some less granular interval such as once per minute) include the complete modeling system state (e.g. all current node locations, adjacencies, etc). This would result in a more bulky EEL file but could enable processing tools to "skip" to desired sections of the file without need to process the entire file from its beginning. This specification does not dictate or preclude such either usage.
>
> Thus, the skeleton format of lines within the EEL format is:
>
> ```
> <time> <moduleID> <eventType> <type-specific fields ...>
> ```
>
> [Protean Research Group, 2010]

The EEL Event Generator loads EEL sentence parsing plugins to parse and build EMANE events. Plugins are associated with event type keywords and are capable of producing either full or delta event updates. A delta event update contains EMANE events corresponding to EEL entries loaded since the last request for events made to the plugin. A full event update contains all the EMANE events necessary to convey the complete current state for all `moduleID` information loaded by the respective plugin.

Any EEL entries encountered that are not handled by a loaded parser are ignored.

There are three EEL sentence parsing plugins:

1. Pathloss Parser - Parses pathloss sentences and builds the resulting event.
   <time> nem:<Id> pathloss nem:<Id>,<pathloss>[,<reversePathloss>] [nem:<Id>,<pathloss>[,<reversePathloss>]]...

pathoss	Pathloss in dB
reversePathloss	Reverse Pathloss in dB

2. Location Parser - Parses location sentences and builds the resulting event.
   <time> nem:<Id> location <latitude>,<longitude>,<altitude>[,msl|agl]

latitude	Latitude in degrees.
longitude	Longitude in degrees.
altitude	Altitude in meters.

3. Antenna Direction Parser - Parses antenna direction sentences and builds the resulting event.
   <time> nem:<Id> antennadirection <elevation>,<azimuth>,<elevationBeamWidth>,<azimuthBeamWidth>

elevation	Antenna elevation in degrees.
azimuth	Antenna azimuth in degrees.
elevationBeamWidth	Antenna elevation beam width in degrees.
azimuthBeamWidth	Antenna azimuth beam width in degrees.

15.1 Configuration Parameters

15.1.1 `inputfile`

Absolute file name of the EEL input file. Additional EEL files may be specified using multiple `inputfile` parameters. Files are processed in the order they appear in the XML.

Type:	String
Range:	N/A
Default:	None
Count:	Unlimited
XML Format:	`<param name="inputfile" value="mobility.eel/>"`

15.1.2 `loader`

Map EEL event type keywords to EEL loader plugins. The optional `full` or `delta` determines whether events produced from the plugins represent only the new EEL entries processed since the last request for events or the complete current cached state. The default specification is `delta`.

Type:	String
Range:	N/A
Default:	None
Count:	Unlimited
XML Format:	`<param param name="loader" value="pathLoss:eelloaderpathloss:full"/>"`

Parameter value format description:

`<eventType>:<Plugin Name>:[full|delta]`

Name	Description
eventType	EEL Event Type
Plugin Name	Plugin corresponding to the EEL Event Type

15.2 Emulation Event Log Format

For more information on EEL Format see [Protean Research Group, 2010].

```
0.0  nem:70 pathLoss nem:22,96.3 nem:23,95.0 nem:24,95.1 nem:25,95.2 nem:26,95.3 nem:27,95.4 nem:28,95.5
     nem:29,95.0 nem:30,95.1 nem:31,95.2 nem:32,95
0.0  nem:70 pathLoss nem:42,95.3 nem:43,95.4 nem:44,95.5 nem:45,95.0 nem:46,95.1 nem:47,95.2 nem:48,95.3
     nem:49,95.4 nem:50,95.5 nem:51,95.5 nem:52,95.6
0.0  nem:70 pathLoss nem:62,95.2 nem:63,95.3 nem:64,95.4 nem:65,94.2 nem:66,94.2 nem:67,96.3 nem:68,96.3
     nem:69,123.3
0.0  nem:1 location gps 40.031075,-74.523518,3.000000
0.0  nem:2 location gps 40.031165,-74.523412,3.000000
0.0  nem:3 location gps 40.031227,-74.523247,3.000000
0.0  nem:4 location gps 40.031290,-74.523095,3.000000
0.0  nem:1 antennadirection 0,90,180,10
0.0  nem:2 antennadirection 0,270,180,10
0.0  nem:3 antennadirection 0,90,180,10
0.0  nem:4 antennadirection 0,270,180,10
```

Listing 15.1: Emulation Event Log sample.

15.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

15.3.1 Demonstration 16

This demonstration deploys a ten node distributed RP Pipe NEM emulation experiment illustrated in Figure 15.1. The goal of this demonstration is to become familiar with the Emulation Event Log Generator.

15.3.1.1 Demonstration Procedure

1. Review the Demonstration 16 Event Service XML using your favorite editor.

   ```
   [emane@emanedemo ~] cd /home/emane/demonstration/16
   [emane@emanedemo 16] less eventservice.xml
   ```

2. Review the Demonstration 16 Emulation Event Log Generator XML using your favorite editor.

   ```
   [emane@emanedemo 16] less eelgenerator.xml
   ```

3. Deploy the demonstration.

   ```
   [emane@emanedemo 16]$ sudo ./lxc-demo-start.sh
   ```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

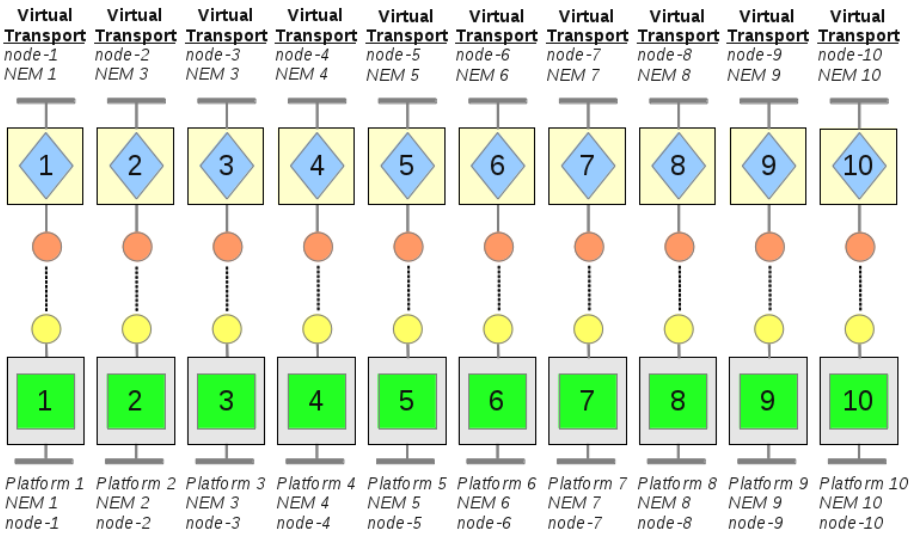


Figure 15.1: Demonstration 16 - Ten node distributed RF Pipe NEM deployment.

5. Stop the demonstration.

```
[emane@emanedemo 16]$ sudo ./lxc-demo-stop.sh
```

15.3.1.2 Concept Review

1. What happens when an Emulation Event Log sentence is encountered that does not match a sentence parser plugin definition?

Chapter 16

Comm Effect Event Generator

The Comm Effect Event Generator creates Comm Effect events from input files in the Comm Effect Impairment Format. The file contains impairment information between nodes on one second boundaries.

16.1 Configuration Parameters

16.1.1 `inputfile`

Absolute file name of the impairment file. Additional impairment files may be specified using multiple `inputfile` parameters. Files are processed in the order they appear in the XML.

Type:	String
Range:	N/A
Default:	None
Count:	Unlimited
XML Format:	`<param name="inputfile" value="commeffect.txt/>"`

16.1.2 `totalnodes`

Total number of nodes whose data is contained in the impairment file(s).

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="totalnodes" value="70"/>`

16.1.3 `maxnemidpresent`

Maximum NEM Id present in the emulation experiment. This value is used to reduce the number of events generated when a subset of nodes from the larger Comm Effect data are used.

Type: Unsigned 16 bit Integer
Range: [0, 65534]
Default: None
Count: 1
XML Format: `<param name="maxnemidpresent" value="12"/>`

16.1.4 repeatcount

The number of times the impairment data should be parsed and events generated. A `repeatcount` of `1` simply means to process the file once, generating events, and stop when the file is complete. A value greater than `1` allows you to process the data `repeatcount` times. A value of `0` will process the data repeatedly, restarting indefinitely.

Type: Unsigned 32 bit Integer
Range: [0, 4294967295]
Default: None
Count: 1
XML Format: `<param name="repeatcount" value="1"/>`

16.1.5 entryreplay

Specify one or more space separated *time:count* pairs. `entryreplay` effectively allows you to hold at certain impairment entries for a specified amount of time. For example a value of "0:120 3600:1800" would use the data at impairment entry T0 for 2 minutes, sending out the T0 events 120 times and use the data at T3600 for 30 minutes.

Type: String
Range: N/A
Default: None
Count: 1
XML Format: `<param name="entryreplay" value="0:120 3600:1800"/>`

Parameter value format description:
`<Time>:<Count> [<Time>:<Count>]...`

Name	Description	Range
Time	Corresponds to the second of data contained in the impairment data	[0, 4294967295]
Count	Represents the number of times you want to replay that data	[1, 4294967295]

16.2 Comm Effect Impairment Format

Comm Effect Impairment Format is an ASCII text file containing a single entry for every pair of nodes in the scenario on one second boundaries. Listing 16.1 shows a sample impairment file. For each second, the number of entries required to completely describe the impairments for N nodes can be represented by the

following equation:

$$\sum_{i=1}^{N-1} N - i$$

The Comm Effects Impairment text file contains eleven columns as defined below:

1. Time in seconds
2. NEM A Target
3. NEM B Target
4. Latency (seconds) - Second component of the average delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric latency between the NEM pairs.
5. Latency (microseconds) - Microsecond component of the average delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric latency between the NEM pairs.
6. Jitter (seconds) - Second component of the jitter on the delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric jitter between the NEM pairs.
7. Jitter (microseconds) - Microsecond component of the jitter on the delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric jitter between the NEM pairs.
8. Loss (percentage) - Loss percentage to be introduced between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric loss between the NEM pairs.
9. Duplicates (percentage) - The duplicate percentage to be introduced between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric duplication between the NEM pairs.
10. Unicastbitrate (bps) - The bitrate to be introduced between NEM A and NEM B for packets addressed to the NEM or handled in promiscuous mode. A single value denotes symmetry and two values separated by a '/' denotes asymmetric bitrate between the NEM pairs.
11. Broadcastbitrate (bps) - The bitrate to be introduced between NEM A and NEM B for packets sent the NEM Broadcast Address. A single value denotes symmetry and two values separated by a '/' denotes asymmetric bitrate between the NEM pairs.

```
0    1    2    0    0                0    0                0        0     10000  1000
0    1    3    0    0                0    0                0        0     10000  1000
0    1    4    0    0                0    0                0        0     10000  1000
0    2    3    0    0                0    0                0        0     10000  1000
0    2    4    0    0                0    0                0        0     10000  1000
0    3    4    0    0                0    0                0        0     10000  1000
1    1    2    0    500000           0    100000           50       0     10000  1000
1    1    3    0    500000           0    100000           50       0     10000  1000
1    1    4    0    500000           0    100000           50       0     10000  1000
1    2    3    0    500000           0    100000           50       0     10000  1000
1    2    4    0    500000           0    100000           50       0     10000  1000
1    3    4    0    500000           0    100000           50       0     10000  1000
2    1    2    0    0                0    0                100      0     10000  1000
2    1    3    0    0                0    0                100      0     10000  1000
2    1    4    0    0                0    0                100      0     10000  1000
2    2    3    0    0                0    0                100      0     10000  1000
2    2    4    0    0                0    0                100      0     10000  1000
2    3    4    0    0                0    0                100      0     10000  1000
3    1    2    0    300000/500000    0    100000/200000    25/40    0/30  10000  1000
```

```
3    1    3    0    300000/500000    0    100000/200000    25/35    0/100  10000  1000
3    1    4    0    300000/500000    0    100000/200000    25/50    0/50   10000  1000
3    2    3    0    500000           0    100000           25       0      10000  1000
3    2    4    0    500000           0    100000           25       0      10000  1000
3    3    4    0    500000           0    100000           25       0      10000  1000
```

Listing 16.1: Comm Effect Effect file sample.

16.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

16.3.1 Demonstration 17

This demonstration deploys a ten node distributed Comm Effect NEM emulation experiment illustrated in Figure 16.1. The goal of this demonstration is to become familiar with the Comm Effect Event Generator.

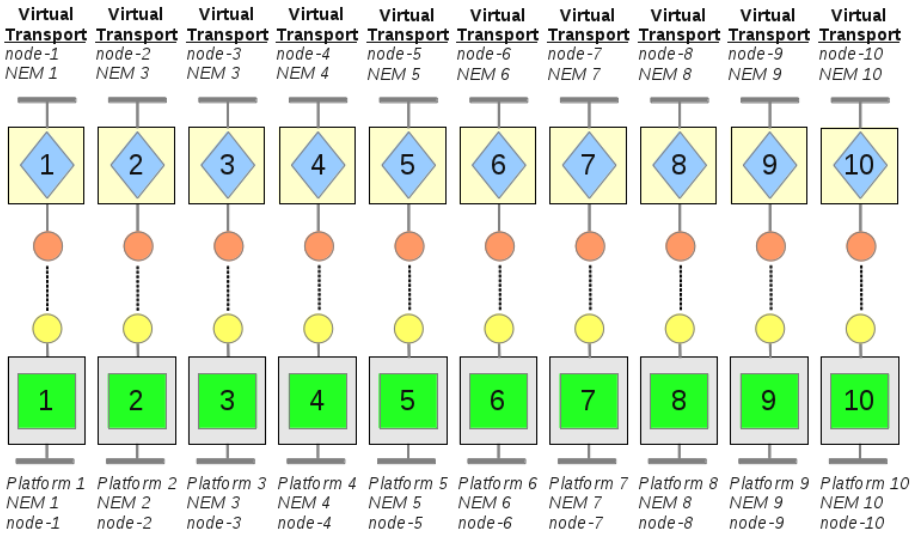


Figure 16.1: Demonstration 17 - Ten node distributed RF Pipe NEM deployment.

16.3.1.1 Demonstration Procedure

1. Review the Demonstration 17 Event Service XML using your favorite editor.

   ```
   [emane@emanedemo ~] cd /home/emane/demonstration/17
   [emane@emanedemo 17] less eventservice.xml
   ```

2. Review the Demonstration 17 Comm Effect Event Generator XML using your favorite editor.

   ```
   [emane@emanedemo 17] less commeffectgenerator.xml
   ```

3. Deploy the demonstration.

   ```
   [emane@emanedemo 17]$ sudo ./lxc-demo-start.sh
   ```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Stop the demonstration.

```
[emane@emanedemo 17]$ sudo ./lxc-demo-stop.sh
```

Chapter 17

Antenna Direction Event Generator

The Antenna Direction Event Generator creates antenna direction events from input files in the Antenna Direction format. The file contains node based antenna profile and pointing information on one second boundaries.

17.1 Configuration Parameters

17.1.1 `inputfileformat`

Absolute file name of the antenna direction file. One or more files may be specified by using a `printf()` style convention using the `inputfilecount` configuration item as an index.

Type:	String
Range:	N/A
Default:	None
Count:	1
XML Format:	`<param name="inputfileformat" value="antenna_direction%02d.txt>"`

17.1.2 `inputfilecount`

Total number of input files. If the `inputfilename` contains a `printf()` style expression the count will be used as an index when creating the file names

Type:	Unsigned 32 bit Integer
Range:	[`0`, `4294967295`]
Default:	None
Count:	1
XML Format:	`<param name="inputfilecount" value="1"`

17.1.3 `totalnodes`

Total number of nodes whose data is contained in the antenna direction file.

Type: Unsigned 32 bit Integer
Range: [0, 4294967295]
Default: None
Count: 1
XML Format: `<param name="totalnodes" value="70"/>`

17.1.4 repeatcount

The number of times the antenna direction data should be parsed and events generated. A `repeatcount` of `1` simply means to process the file once, generating events, and stop when the file is complete. A value greater than `1` allows you to process the data `repeatcount` times. A value of `0` will process the data repeatedly, restarting indefinitely.

Type: Unsigned 32 bit Integer
Range: [0, 4294967295]
Default: None
Count: 1
XML Format: `<param name="repeatcount" value="1"/>`

17.2 Antenna Direction Format

Antenna Direction Format is an ASCII text file containing a single entry for every node using directional antenna on one second boundaries.

The Antenna Direction text file contains eleven columns as defined below:

1. Time in Seconds
2. Node Target
3. Antenna Elevation in degrees
4. Antenna Azimuth in degrees
5. Antenna Elevation Beam Width in degrees
6. Antenna Azimuth Beam Width in degrees

```
0  1 0 90  180 10
0  2 0 270 180 10
0  3 0 90  180 10
0  4 0 270 180 10
1  1 0 180 180 10
1  2 0 180 180 10
1  3 0 0   180 10
1  4 0 0   180 10
2  1 0 135 180 10
2  2 0 225 180 10
2  3 0 45  180 10
2  4 0 315 180 10
```

Listing 17.1: Antenna Direction file sample.

17.3 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

17.3.1 Demonstration 18

This demonstration deploys a four node centralized IEEE 802.11abg NEM emulation experiment illustrated in Figure 17.1. The goal of this demonstration is to become familiar with the Antenna Direction Event Generator.

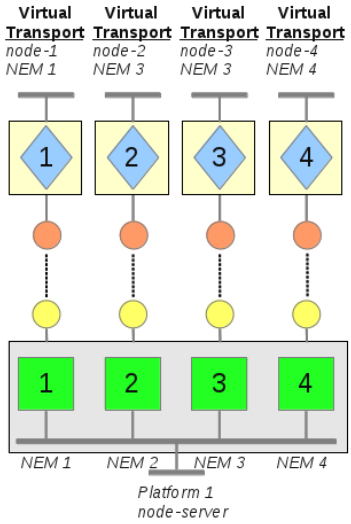


Figure 17.1: Demonstration 18 - Four node centralized IEEE 802.11abg NEM deployment.

17.3.1.1 Demonstration Procedure

1. Review the Demonstration 18 Event Service XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/18
[emane@emanedemo 18] less eventservice.xml
```

2. Review the Demonstration 18 Antenna Direction Event Generator XML using your favorite editor.

```
[emane@emanedemo 18] less antennadirectiongenerator.xml
```

3. Deploy the demonstration.

```
[emane@emanedemo 18]$ sudo ./lxc-demo-start.sh
```

4. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

5. Stop the demonstration.

```
[emane@emanedemo 18]$ sudo ./lxc-demo-stop.sh
```

17.3.1.2 Concept Review

1. Why are two event generators required for this demonstration?

Chapter 18

GPSd Location Agent

The GPSd Location Agent uses a pseudo terminal to emulate a GPS Receiver. Received location events are used to generate NMEA strings which are written to a pseudo terminal in order to make position information available to any application capable of parsing NMEA strings.

18.1 Configuration Parameters

18.1.1 `gpsdcontrolsocket`

The name of the GPSd control socket for adding the pseudo terminal to the device list. The control socket is used when the GPSd Location Agent instance should attempt to connect to GPSd. Only used when `gpsdconnectionenabled` is set to `on`.

Type:	String
Range:	N/A
Default:	`/tmp/gpsd.control`
Count:	1
XML Format:	`<param name="gpsdcontrolsocket" value="/tmp/gpsd.control"/>"`

18.1.2 `pseudoterminalfile`

The name of the file to create which will contain the name of the pseudo terminal in use by the GPSd Location Agent. Only created when `gpsdconnectionenabled` set to `off`.

Type:	String
Range:	N/A
Default:	`/tmp/gpsdlocation.pty`
Count:	1
XML Format:	`<param name="pseudoterminalfile" value="/tmp/gpsdlocation.pty"/>"`

18.1.3 `gpsdconnectionenabled`

Switch to set GPSd Location Agent to either actively connect to GPSd (`on`) or instead create a file containing the name of the pseudo terminal currently in use (`off`).

Type:	Boolean
Range:	[`off`, `on`]
Default:	`off`
Count:	1
XML Format:	`<param param name="gpsdconnectionenabled" value="off"/>"`

18.2 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

18.2.1 Demonstration 19

This demonstration deploys a ten node distributed IEEE 802.11abg NEM emulation experiment illustrated in Figure 18.1. The goal of this demonstration is to become familiar with the GPSd Location Agent.

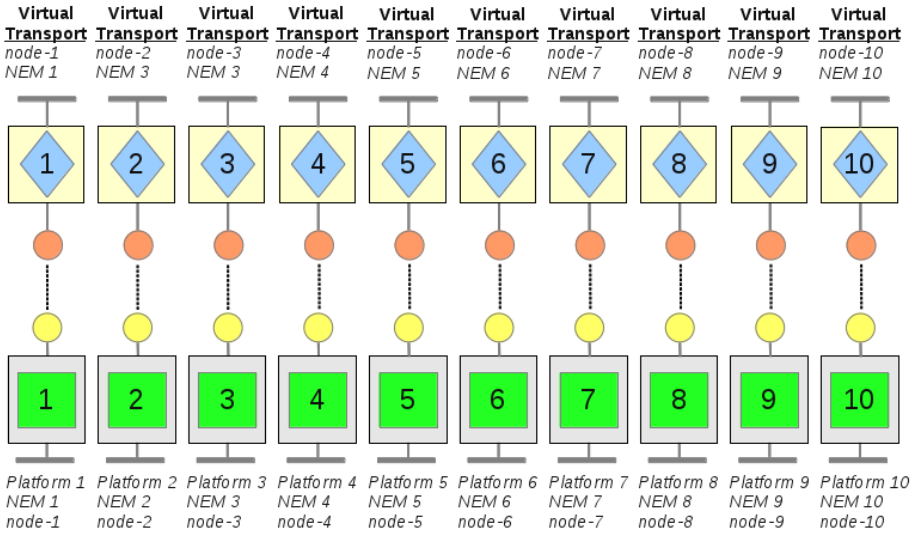


Figure 18.1: Demonstration 19 - Ten node distributed IEEE 802.11abg NEM deployment.

18.2.1.1 Demonstration Procedure

1. Review the Demonstration 19 Event Damon XML using your favorite editor.

```
[emane@emanedemo ~] cd /home/emane/demonstration/19
[emane@emanedemo 19] less eventdaemon{[1-9],10}.xml
```

2. Deploy the demonstration.

```
[emane@emanedemo 19]$ sudo ./lxc-demo-start.sh -d 0
```

3. Connect to virtual `node-1`.

```
[emane@emanedemo 19]$ ssh node-1
```

4. Review `node-1`'s emulated GPS.

```
[emane@node-1 ~]$ cgps
    Time:            2012-02-08T18:05:14.0Z PRN:   Elev:  Azim:  SNR:  Used:
    Latitude:        40.037071 N               1    41    104    41      Y
    Longitude:       74.482736 W               3    09    084    51      Y
    Altitude:        9.8 ft                    5    70    030    39      Y
    Speed:           0.0 mph                   7    35    185    25      Y
    Heading:         0.0 deg (true)            5    10    297    25      Y
    Climb:           0.0 ft/min                7    53    311    21      Y
    Status:          3D FIX (46 secs)          9    02    029    29      N
    GPS Type:                                 11    48    064    32      N
    Longitude Err:   +/- 7 ft
    Latitude Err:    +/- 9 ft
    Altitude Err:    +/- 22 ft
    Course Err:      n/a
    Speed Err:       +/- 13 mph
```

5. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

6. Stop the demonstration.

```
[emane@emanedemo 19]$ sudo ./lxc-demo-stop.sh
```

Part V

Python Bindings

Chapter 19

Event Service Python Bindings

The Python Event Service bindings allow for the creation of custom Python scripts which can interact with the EMANE Event Service using libemaneventservice, a C language library for developing embedded event processing applications.

For more information on the libemaneeventservice API refer to the EMANE Developer Manual and the manual entries: `emaneeventservice`, `emaneeventlocation`, `emaneeventpathloss`, `emaneeventcommeffect` and `emaneeventantennadirection`.

The Event Service Python bindings are comprised of bindings for the Event Service and the four standard EMANE events: Location, Pathloss, Comm Effect and Antenna Direction.

19.1 Configuration

In order to use the Python Event Service bindings, libemaneeventservice must be configured appropriately. libemaneventservice will search for its configuration in three locations before falling back and using application defaults. The search order is as follows:

1. If the environment variable `LIBEMANEEVENTSERVICECONFIG` exists, it will be used. This variable must be set to a configuration file name.
2. If `$HOME/.libemaneeventservice.xml` exists, it will be used.
3. If `/etc/libemaneeventservice.xml` exists, it will be used.
4. The default values are: group 224.1.2.8, port 45703, multicast loop enabled (1), TTL 32. No default multicast device is specified. The kernel routing table is used.

```
<?xml version='1.0' standalone='yes'?>
<emaneeventmsgsvc>
  <group>224.1.2.8</group>
  <port>45703</port>
  <device>lo</device>
  <mcloop>1</mcloop>
  <ttl>32</ttl>
</emaneeventmsgsvc>
```

Listing 19.1: `libemaneeventservice` configuration file.

19.2 EventService

The Python EventService module provides an interface for subscribing and publishing to the Event Service. Listing 19.2 shows a sample script which publishes a single Location Event.

```
#!/usr/bin/env python

import emaneeventservice
import emaneeventlocation

service = emaneeventservice.EventService()

# Location Event contains a variable list of NEM locations

location =  emaneeventlocation.EventLocation(10)

location.set(0,1,40.031075,-74.523518,3)
location.set(1,2,40.031165,-74.523412,3)
location.set(2,3,40.031227,-74.523247,3)
location.set(3,4,40.031290,-74.523095,3)
location.set(4,5,40.031361,-74.522942,3)
location.set(5,6,40.031433,-74.522836,3)
location.set(6,7,40.031075,-74.523518,3)
location.set(7,8,40.031165,-74.523412,3)
location.set(8,9,40.031227,-74.523247,3)
location.set(9,10,40.031290,-74.523095,3)

service.publish(emaneeventlocation.EVENT_ID,
                emaneeventservice.PLATFORMID_ANY,
                emaneeventservice.NEMID_ANY,
                emaneeventservice.COMPONENTID_ANY,
                location.export())
```

Listing 19.2: Python EventLocation publish example.

19.3 EventLocation

The Python EventLocation module provides an interface for creating and accessing Location Events. Location Event entries are returned as a dictionary of tuples:

```
{1: (1, 40.031075, -74.523517, 3),
 2: (2, 40.031165, -74.523411, 3),
 3: (3, 40.031227, -74.523246, 3),
 4: (4, 40.031289, -74.523094, 3),
 5: (5, 40.03136, -74.522942, 3),
 6: (6, 40.031432, -74.522835, 3),
 7: (7, 40.031075, -74.523517, 3),
 8: (8, 40.031165, -74.523411, 3),
 9: (9, 40.031227, -74.523246, 3),
 10: (10, 40.031289, -74.523094, 3)}
```

Where, the dictionary key is the NEM Id associated with the tuple value and the tuple contains the respective NEM Id, latitude in degrees, longitude in degrees and altitude in meters. Listing 19.3 contains sample code showing the creation and publication of a Location Event. Listing 19.4 contains a sample Location Event handler.

```
# Location Event contains a variable list of NEM locations

location =  emaneeventlocation.EventLocation(10)

location.set(0,1,40.031075,-74.523518,3)
location.set(1,2,40.031165,-74.523412,3)
location.set(2,3,40.031227,-74.523247,3)
location.set(3,4,40.031290,-74.523095,3)
location.set(4,5,40.031361,-74.522942,3)
location.set(5,6,40.031433,-74.522836,3)
location.set(6,7,40.031075,-74.523518,3)
location.set(7,8,40.031165,-74.523412,3)
location.set(8,9,40.031227,-74.523247,3)
location.set(9,10,40.031290,-74.523095,3)

service.publish(emaneeventlocation.EVENT_ID,
                emaneeventservice.PLATFORMID_ANY,
                emaneeventservice.NEMID_ANY,
                emaneeventservice.COMPONENTID_ANY,
                location.export())
```

Listing 19.3: Python EventLocation publish example.

```
def handleLocationEvent(event, platform, nem, component, data):
    global location

    location +=1

    print "received location event ", event, " platform ", platform,\
        " nem ", nem, " component ", component, " length ",len(data),\
        " bytes"

    event = emaneeventlocation.EventLocation(data)

    entries  = event.entries()

    for e in entries.values():
        print e
```

Listing 19.4: Python EventLocation handler example.

19.4 EventPathloss

The Python EventPathloss module provides an interface for creating and accessing Pathloss Events. Pathloss Event entries are returned as a dictionary of tuples:

```
{1: (1, 100.0, 100.0),
 2: (2, 100.0, 100.0),
 3: (3, 100.0, 100.0),
 4: (4, 100.0, 100.0),
 5: (5, 100.0, 100.0),
 6: (6, 100.0, 100.0),
 7: (7, 100.0, 100.0),
 8: (8, 100.0, 100.0),
 9: (9, 100.0, 100.0),
 10: (10, 100.0, 100.0)}
```

Where, the dictionary key is the transmitter NEM Id associated with the tuple value and the tuple contains the respective transmitter NEM Id, pathloss from the transmitter to the receiver in dB and the pathloss from the receiver to the transmitter in dB. Listing 19.5 contains sample code showing the creation and publication of a Pathloss Event. Listing 19.6 contains a sample Pathloss Event handler.

```
# Pathloss Event contains a variable list of transmittting NEM pathloss and
# reverse pathloss enties.

pathloss  = emaneeventpathloss.EventPathloss(10)

for index in range(0,10):
    pathloss.set(index,
                 index+1,
                 100,
                 100)

service.publish(emaneeventpathloss.EVENT_ID,
                emaneeventservice.PLATFORMID_ANY,
                emaneeventservice.NEMID_ANY,
                emaneeventservice.COMPONENTID_PHYI,
                pathloss.export())
```

Listing 19.5: Python EventPathloss publish example. This listing is atypical. Pathloss Events are usually tailored to each NEM target since they represent the pathloss from one ore more transmitters to a receiver.

```
def handlePathlossEvent(event, platform, nem, component, data):
    global pathloss

    pathloss +=1

    print "received pathloss event ", event, " platform ", platform,\
        " nem ", nem, " component ", component, " length ",len(data),\
        " bytes"

    event = emaneeventpathloss.EventPathloss(data)

    entries  = event.entries()

    for e in entries.values():
        print e
```

Listing 19.6: Python EventPathloss handler example.

19.5 EventCommEffect

The Python EventCommEffect module provides an interface for creating and accessing Comm Effect Events. Comm Effect Event entries are returned as a dictionary of tuples:

```
{1: (1, 0, 0, 0, 0, 65, 0, 0L, 0L),
 2: (2, 0, 0, 0, 0, 65, 0, 0L, 0L),
 3: (3, 0, 0, 0, 0, 65, 0, 0L, 0L),
 4: (4, 0, 0, 0, 0, 65, 0, 0L, 0L),
 5: (5, 0, 0, 0, 0, 65, 0, 0L, 0L),
 6: (6, 0, 0, 0, 0, 65, 0, 0L, 0L),
 7: (7, 0, 0, 0, 0, 65, 0, 0L, 0L),
 8: (8, 0, 0, 0, 0, 65, 0, 0L, 0L),
 9: (9, 0, 0, 0, 0, 65, 0, 0L, 0L),
 10: (10, 0, 0, 0, 0, 65, 0, 0L, 0L)}
```

Where, the dictionary key is the transmitter NEM Id associated with the tuple value and the tuple contains the respective transmitter NEM Id, latency seconds, latency microseconds, jitter seconds, jitter microseconds, percentage loss, percentage duplication, unicast bits per seconds and broadcast bits per second. Listing 19.7 contains sample code showing the creation and publication of a Comm Effect Event. Listing 19.8 contains a sample Comm Effect Event handler.

```
# Comm Effect Event contains a variable list of transmittting NEM Comm
# Effect impairment enties.

commeffect  = emaneeventcommeffect.EventCommEffect(10)

for index in range(0,10):
    commeffect.set(index,
                   index+1,
                   0,
                   0,
                   0,
                   0,
                   65,
                   0,
                   0L,
                   0L)

service.publish(emaneeventcommeffect.EVENT_ID,
                emaneeventservice.PLATFORMID_ANY,
                emaneeventservice.NEMID_ANY,
                emaneeventservice.COMPONENTID_ANY,
                commeffect.export())
```

Listing 19.7: Python EventCommEffect publish example. This listing is atypical. Comm Effect Events are usually tailored to each NEM target since they represent the impairments from one ore more transmitters to a receiver.

```
def handleCommEffectEvent(event, platform, nem, component, data):
    global commeffect

    commeffect +=1

    print "received commeffect event ", event, " platform ", platform,\
        " nem ", nem, " component ", component, " length ",len(data),\
        " bytes"

    event = emaneeventcommeffect.EventCommEffect(data)

    entries  = event.entries()

    for e in entries.values():
        print e
```

Listing 19.8: Python EventCommEffect handler example.

19.6 EventAntennaDirection

The Python EventAntennaDirection module provides an interface for creating and accessing Antenna Direction Events. Antenna Direction Event entries are returned as a dictionary of tuples:

```
{1: (1, 0.0, 90.0, 180.0, 10.0),
 2: (2, 0.0, 270.0, 180.0, 10.0),
 3: (3, 0.0, 90.0, 180.0, 10.0),
 4: (4, 0.0, 270.0, 180.0, 10.0),
 5: (5, 0.0, 90.0, 180.0, 10.0),
 6: (6, 0.0, 270.0, 180.0, 10.0),
 7: (7, 0.0, 90.0, 180.0, 10.0),
 8: (8, 0.0, 270.0, 180.0, 10.0),
 9: (9, 0.0, 90.0, 180.0, 10.0),
 10: (10, 0.0, 270.0, 180.0, 10.0)}
```

Where, the dictionary key is the NEM Id associated with the tuple value and the tuple contains the respective NEM Id, antenna elevation in degrees, antenna azimuth in degrees, antenna elevation beam width in degrees and antenna azimuth beam width in degrees. Listing 19.9 contains sample code showing the creation and publication of an Antenna Direction Event. Listing 19.10 contains a sample Antenna Direction Event handler.

```
# Antenna Direction Event contains a variable list of NEM directional antenna
# pointing and profile information
antennadirection =  emaneeventantennadirection.EventAntennaDirection(10)

antennadirection.set(0,1,0,90,180,10)
antennadirection.set(1,2,0,270,180,10)
antennadirection.set(2,3,0,90,180,10)
antennadirection.set(3,4,0,270,180,10)
antennadirection.set(4,5,0,90,180,10)
antennadirection.set(5,6,0,270,180,10)
antennadirection.set(6,7,0,90,180,10)
antennadirection.set(7,8,0,270,180,10)
antennadirection.set(8,9,0,90,180,10)
antennadirection.set(9,10,0,270,180,10)

service.publish(emaneeventantennadirection.EVENT_ID,
                emaneeventservice.PLATFORMID_ANY,
                emaneeventservice.NEMID_ANY,
                emaneeventservice.COMPONENTID_PHYI,
                antennadirection.export())
```

Listing 19.9: Python EventAntennaDirection publish example.

```
def handleAntennaDirectionEvent(event, platform, nem, component, data):
    global antenna

    antenna +=1

    print "received antenna direction event ", event, " platform ", platform,\
        " nem ", nem, " component ", component, " length ",len(data), " bytes"

    event = emaneeventantennadirection.EventAntennaDirection(data)

    entries  = event.entries()

    for e in entries.values():
        print e
```

Listing 19.10: Python EventAntennaDirection handler example.

19.7 Demonstrations

The following demonstrations were designed to re-enforce the material covered in this chapter. Deploy and review each demonstration.

19.7.1 Demonstration 20

This demonstration uses two sample Python scripts to illustrate how to use the Python Event Service bindings.

19.7.1.1 Demonstration Procedure

1. Review the Demonstration 20 sample Python scripts using your favorite editor.

   ```
   [emane@emanedemo ~] cd /home/emane/demonstration/20
   [emane@emanedemo 20] less eventpublisher.py eventsubscriber.py
   ```

2. Start the *eventsubscriber.py* script.

   ```
   [emane@emanedemo 20]$ ./eventsubscriber.py
   ```

3. In a separate terminal start the *eventpublisher.py* script.

```
[emane@emanedemo ~] cd /home/emane/demonstration/20
[emane@emanedemo 20]$ ./eventpublisher.py
```

4. Observer the output from the event subscriber script.

```
received commeffect event  207  platform  0  nem  0  component  0  length  382  bytes
(1, 0, 0, 0, 0, 65, 0, 0L, 0L)
(2, 0, 0, 0, 0, 65, 0, 0L, 0L)
(3, 0, 0, 0, 0, 65, 0, 0L, 0L)
(4, 0, 0, 0, 0, 65, 0, 0L, 0L)
(5, 0, 0, 0, 0, 65, 0, 0L, 0L)
(6, 0, 0, 0, 0, 65, 0, 0L, 0L)
(7, 0, 0, 0, 0, 65, 0, 0L, 0L)
(8, 0, 0, 0, 0, 65, 0, 0L, 0L)
(9, 0, 0, 0, 0, 65, 0, 0L, 0L)
(10, 0, 0, 0, 0, 65, 0, 0L, 0L)
received pathloss event  203  platform  0  nem  0  component  2  length  102  bytes
(1, 100.0, 100.0)
(2, 100.0, 100.0)
(3, 100.0, 100.0)
(4, 100.0, 100.0)
(5, 100.0, 100.0)
(6, 100.0, 100.0)
(7, 100.0, 100.0)
(8, 100.0, 100.0)
(9, 100.0, 100.0)
(10, 100.0, 100.0)
received location event  204  platform  0  nem  0  component  0  length  142  bytes
(1, 40.031075, -74.523517, 3)
(2, 40.031165, -74.523411, 3)
(3, 40.031227, -74.523246, 3)
(4, 40.031289, -74.523094, 3)
(5, 40.03136, -74.522942, 3)
(6, 40.031432, -74.522835, 3)
(7, 40.031075, -74.523517, 3)
(8, 40.031165, -74.523411, 3)
(9, 40.031227, -74.523246, 3)
(10, 40.031289, -74.523094, 3)
received antenna direction event  209  platform  0  nem  0  component  2  length  182  bytes
(1, 0.0, 90.0, 180.0, 10.0)
(2, 0.0, 270.0, 180.0, 10.0)
(3, 0.0, 90.0, 180.0, 10.0)
(4, 0.0, 270.0, 180.0, 10.0)
(5, 0.0, 90.0, 180.0, 10.0)
(6, 0.0, 270.0, 180.0, 10.0)
(7, 0.0, 90.0, 180.0, 10.0)
(8, 0.0, 270.0, 180.0, 10.0)
(9, 0.0, 90.0, 180.0, 10.0)
(10, 0.0, 270.0, 180.0, 10.0)
```

5. Stop the *eventsubscriber.py* script.

```
^C
Location Event          : 1
Comm Effect Event       : 1
Pathloss Event          : 1
Antenna Direction Event : 1
```

19.7.1.2 Concept Review

1. Which libemaneeventservice configuration file was used during this demonstration?

Chapter 20

EMANE Library Python Bindings

The EMANE Library Python bindings are contained in the Python `emane` module. By importing the `emane` module, a Python script can configure and run the equivalent of any of the EMANE applications: `emane`, `emanetransportd`, `emaneeventservice` and `emaneeventd`, alone or together. Configuration is performed via Python tuples and does not require XML configuration. Access to the EMANE logger allows log level and log destination configuration, similar to the application logger command line arguments.

The `emane` Python module wraps the C++ `libemane` library. Refer to the EMANE Developer Manual for more information on the `libemane` API. The `emane` module API documentation is available via `pydoc`.

This chapter develops an example Python script that runs an NEM Platform Server, Transport container and Event Agent container together. The resulting script is featured in Section 20.7.1 Demonstration 21 on page 169.

20.1 Configuration

The EMANE Library bindings enable the execution of EMANE experiments from self-contained Python scripts, without EMANE XML configuration. Listing 20.1 shows a code snippet that creates a configured RF Pipe MAC Layer. Configuration is passed into the MAC Layer constructor as a Python tuple of (name,value) tuples. The names and permissible values of the configuration items depend on the underlying implementation and correspond to the parameters traditionally set by XML configuration.

The configuration tuple shown here is a canned example, constructed from literal values. In practice, the user can construct these tuples using any useful method.

```
    # build a configured mac layer
    macconfig = ( ('enabletighttiming', 'off'),
                  ('flowcontrolenable', 'off'),
                  ('jitter', '0'),
                  ('pcrcurveuri',
                   'file:///usr/share/emane/models/rfpipe/xml/rfpipepcr.xml'),
                  ('datarate', '1000'),
                  ('flowcontroltokens', '10'),
                  ('delay', '0'),
                  ('enablepromiscuousmode', 'off') )

    nemlayers.append(emane.MACLayer(nemid, 'rfpipemaclayer', macconfig))
```

Listing 20.1: Python EMANE configuration excerpt.

20.2 Logger

An `emane.Logger` instance can be used to set the verbosity and destination of the log statements generated by constructed EMANE components. The default log destination is `stdout`. Output can be redirected to a file, to the system logger or to an ACE Log Server running at a remote destination. Listing 20.2, shows log output redirected to the file *mylogfile.txt* at level 4. The numeric level corresponds to the option values shown in Table 6.1.

```
    # Configure the EMANE logger to write to a file,
    logger = emane.Logger()
    logger.redirectLogsToFile('mylogfile.txt')

    # set the log level. legal values are 0 to 4, 4 is most verbose
    logger.setLogLevel(4)
```

Listing 20.2: Python Logger.

20.3 Event Agent Manager

Four different top level managers can be built by the `emane` module. Each manager corresponds to one of the EMANE applications:

- `Platform` is used by the `emane` application.
- `TransportManager` is used by the `emanetransportd` application.
- `EventGeneratorManager` is used by the `emaneeventservice` application.
- `EventAgentManager` is used by the `emaneeventd` application.

Listing 20.3 shows a function that creates and returns a configured `EventAgentManager`. The script follows a pattern that will become familiar in subsequent examples:

1. Create the fundamental EMANE component (NEM Layer, Transport, Event Generator or Event Agent).
2. Insert the component into the appropriate manager.
3. Use the manager to execute the contained components.

In this example, the manager contains one `EventAgent`, created first. It's constructor takes the library name of the agent implementation, the configuration tuple and an NEM Id. Some agents use the NEM Id to discard all events except those destined for a specified NEM. The Event Service configuration for the agent is passed in as a parameter because it is required by other builders in the top level example.

```
def buildEventAgentManager(nemid, eventserviceconfig):
    # build the agent manager
    agentconfig = \
        ( ('pseudoterminalfile', '/tmp/lxc-node/21/%d/var/lib/gps.pty' % nemid),
          ('gpsdconnectionenabled', 'no') )

    # first the agents
    agents = [ emane.EventAgent(nemid, 'gpsdlocationagent', agentconfig) ]

    # then the manager
    return emane.EventAgentManager(agents, tuple(eventserviceconfig))
```

Listing 20.3: Creating an `EventAgentManager`. This function is used in Listing 20.6.

20.4 Transport Manager

Listing 20.4 shows a function that creates and returns a configured `TransportManager` containing one `Transport`. The code pattern is similar to the previous example. Create a `Transport` first. The transport constructor takes the name of its implementing library, the configuration tuple and the NEM Id of the associated NEM. In an intermediate step, each transport must be inserted into a `TransportAdapter` that handles the socket communication to the NEM. Finally, a `TransportManager` is created from the list of adapters.

```
def buildTransportManager(nemid, endpointconfig):
    # build the transport manager

    # first the transport(s)
    transportconfig = ( ('devicepath', '/dev/net/tun'),
                        ('bitrate', '0.0'),
                        ('mask', '255.255.255.0'),
                        ('address', '10.100.0.%d' % nemid) )
    transport = emane.Transport(nemid, 'transvirtual', transportconfig)

    # then an adapter for each transport, handles nem<->transport connection
    adapters = []
    adapters.append(emane.TransportAdapter(transport, tuple(endpointconfig)))

    # then the manager
    return emane.TransportManager(adapters, ())
```

Listing 20.4: Creating a `TransportManager`. This function is used in Listing 20.6.

20.5 Platform Server

Listing 20.5 shows a function that creates and returns a configured Platform Server, an instance of the `Platform` class. Platform creation requires more steps than the other managers, but the usual pattern holds. To create an NEM, create a list of NEM Layers with the same NEM Id. The NEM constructor constructs its stack from the list by placing the first element at the top of the stack, nearest the Transport, and each subsequent layer in order beneath. A `Platform` is created from a list of NEMs, its own configuration tuple and a user defined Platform Id. EMANE events can be addressed to individual platforms based on Platform Id (Section 4.1 Event Service on page 33). Assign Platform Ids uniquely in a multi-platform emulation to differentiate events by platform.

The `emane` module will not create more than one Platform in a given script.

```
def buildPlatform(nemid, endpointconfig, eventserviceconfig):
    # build the layers for our nem stack from top to bottom
    nemlayers = []

    # first mac
    macconfig = ( ('enabletighttiming', 'off'),
                  ('flowcontrolenable', 'off'),
                  ('jitter', '0'),
                  ('pcrcurveuri',
                   'file:///usr/share/emane/models/rfpipe/xml/rfpipepcr.xml'),
                  ('datarate', '1000'),
                  ('flowcontroltokens', '10'),
                  ('delay', '0'),
                  ('enablepromiscuousmode', 'off') )
    nemlayers.append(emane.MACLayer(nemid, 'rfpipemaclayer', macconfig))


    # then phy
    phyconfig = ( ('noiseprocessingmode', 'off'),
                  ('antennatype', 'omnidirectional'),
                  ('frequencyofinterest', '2347000'),
                  ('antennaazimuth', '0.0'),
                  ('antennaelevation', '0.0'),
```

```
                    ('antennagain', '0.0'),
                    ('txpower', '0.0'),
                    ('bandwidth', '1000'),
                    ('frequency', '2347000'),
                    ('antennaelevationbeamwidth', '180.0'),
                    ('pathlossmode', 'freespace'),
                    ('subid', '2'),
                    ('systemnoisefigure', '4.0'),
                    ('defaultconnectivitymode', 'on'),
                    ('antennaazimuthbeamwidth', '360.0') )
    nemlayers.append(emane.PHYLayer(nemid, 'universalphylayer', phyconfig))

    # then an NEM from mac and phy
    nems = []
    nems.append(emane.NEM(nemid, nemlayers, tuple(endpointconfig)))

    # then the platform server from the nem(s)
    platformid = 1
    platformconfig = tuple( eventserviceconfig +
                            [ ('otamanagerdevice','eth0'),
                              ('otamanagergroup','224.1.2.8:45702'),
                              ('otamanagerchannelenable','on') ] )

    return emane.Platform(platformid, nems, platformconfig)
```

Listing 20.5: Creating a Platform Server. This function is used in Listing 20.6.

20.6 Putting It Together

Listing 20.6 shows a top level main function that builds three managers using the functions from previous examples. The log level is set to maximum output. `transportendpoint`, `platformendpoint` and Event Service configuration are defined here because they are needed by more than one build function.

The script calls the `start` method of the returned managers to run them. It pauses for an external signal while they execute and calls their `stop` method to clean up on exit.

The important thing to note here is that the developed Python script essentially configures and executes an instance of `emane`, `emanetransportd` and `emaneeventd` within one OS process and without external configuration.

```
def main():

    # set the log level
    logger = emane.Logger()
    logger.setLogLevel(4)

    # pass in the NEM Id
    nemid = sys.argv[1]

    # platform<-->transport config common to platform and transport
    endpointconfig = [ ('platformendpoint', 'localhost:%d' % (8200 + nemid)),
                       ('transportendpoint', 'localhost:%d' % (8300 + nemid)) ]

    # eventservice config, common to platform and agents
    eventserviceconfig = [ ('eventservicegroup', '224.1.2.8:45703'),
                           ('eventservicedevice', 'eth0') ]

    # build the application containers
    transportmanager = buildTransportManager(nemid, endpointconfig)
    platform = buildPlatform(nemid, endpointconfig, eventserviceconfig)
    eventagentmanager = buildEventAgentManager(nemid, eventserviceconfig)

    # start containers
    platform.start()
    transportmanager.start()
    eventagentmanager.start()

```

```
    # wait while they work
    s = SignalHandler(signal.SIGINT)
    s.wait()

    # stop containers
    eventagentmanager.stop()
    platform.stop()
    transportmanager.stop()
```

Listing 20.6: A main function running three managers.

20.7 Demonstrations

The following demonstration is designed to re-enforce the material covered in this chapter. Deploy and review the demonstration.

20.7.1 Demonstration 21

This demonstration deploys a ten node distributed RF Pipe NEM emulation experiment illustrated in Figure 20.1. Each node executes a Python script similar to the one developed above.

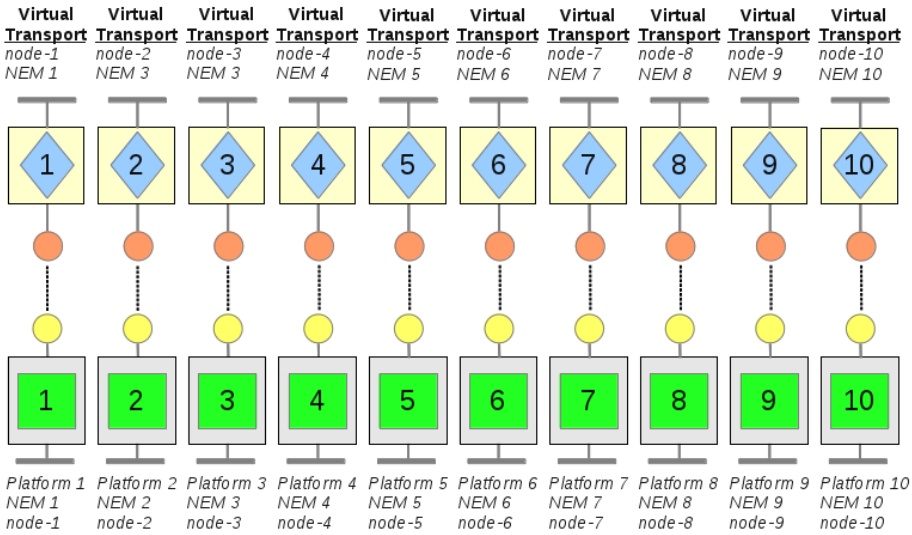


Figure 20.1: Demonstration 21 - Ten node distributed RF Pipe NEM deployment.

20.7.1.1 Demonstration Procedure

1. Review the *runemane.py* sample Python script using your favorite editor.

   ```
   [emane@emanedemo ~] cd /home/emane/demonstration/21
   [emane@emanedemo 21] less runemane.py
   ```

2. Deploy the demonstration.

   ```
   [emane@emanedemo 21] sudo ./lxc-demo-start.sh
   ```

3. Connect to virtual `node-1`

   ```
   [emane@emanedemo 21] ssh node-1
   ```

4. Review the running processes

```
[emane@node-1 ~]$ ps ax
  PID TTY      STAT   TIME COMMAND
    1 ?        S+     0:00 /usr/lib/lxc/lxc-init -- /tmp/lxc-node/21/1/init.sh -s 17:45:27
    5 ?        Ss     0:00 /usr/sbin/sshd -o PidFile=/tmp/lxc-node/21/1/run/sshd.pid
    8 ?        Sl     1:46 /usr/bin/python /home/emane/demonstration/21/runemane.py -d 1
                           --logfile /tmp/lxc-node/21/1/var/log/emane.log
   24 ?        S<s    0:13 gpsd -n -b /dev/pts/0
   29 ?        Ssl    1:35 /usr/sbin/olsrd -f /home/emane/demonstration/21/routing1.conf
   97 ?        Ss     0:00 sshd: emane [priv]
   99 ?        S      0:00 sshd: emane@pts/1
  100 pts/1    Ss     0:00 -bash
  161 pts/1    R+     0:00 ps ax
```

5. Open the OLSR Link Viewer application to monitor the emulated network. From the top panel select *OLSR Link Viewer* from the launcher to the right of the Firefox launcher.

6. Disconnect from `node-1`.

```
[emane@node-1 ~]$ exit
logout
Connection to node-1 closed.
```

7. Stop the demonstration.

```
[emane@emanedemo 21] sudo ./lxc-demo-stop.sh
```

20.7.1.2 Concept Review

1. How many Platform Servers can one Python script create?
2. When creating a structured NEM, should the MAC or PHY occur first in the layer list of the NEM constructor?
3. How should the script be modified to create a centralized deployment?

Bibliography

Protean Research Group. *EmulationScript Schema Description*. United States Naval Research Laboratory Code 5522, 2010.