EMANE User Training

0.6.2

CenGen Inc
9250 Bendix Road North
Columbia, MD 21045

cengen.com
labs.cengen.com/emane

EMANE User Training Workbook

0.6.2

CenGen Labs

EMANE User Training Workbook: 0.6.2

CenGen Labs

Copyright © 2010, 2009 CenGen Inc., Columbia, Maryland

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of CenGen Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

Preface vii
How to use this workbook vii
Creating an EMANE training installation vii
Reading an EMANE Deployment Diagram viii
I. Overview 1
Introduction 2
Emulation Stack 4
Component Communication 6
Emulation Environmental Data 8
Applications and Utilities 9
XML Hierarchy 11
Workshop 00 - EMANE Applications and Documentation 13
Problem Statement 13
II. Deployments 14
Deployments 15
The Shim Layer 18
Structured and Unstructured NEM 19
Workshop 01 - Centralized Deployment 20
Deployment Description 20
Problem Statement 20
Platform Attributes 21
Platform Parameters 21
NEM Attributes 22
NEM Parameters 22
XML Listing 22
Concept Review 23
Workshop 02 - Distributed Deployment 24
Deployment Description 24
Problem Statement 24
Platform Attributes 25
Platform Parameters 25
NEM Attributes 26
NEM Parameters 26
XML Listing 26
Concept Review 28
Workshop 03 - Hybrid Deployment 29
Deployment Description 29
Problem Statement 29
Platform Attributes 30
Platform Parameters 30
NEM Attributes 31
NEM Parameters 31
XML Listing 31
Concept Review 34
III. Transports 35
Transports 36
Virtual Transport 37
Raw Transport 38
Interactive Transport 39
Transport Interoperability 40
Workshop 04 - Virtual Transport 41

Deployment Description 41
Problem Statement 41
NEM Attributes 41
NEM Parameters 42
Virtual Transport Parameters 42
XML Listing 43
Concept Review 45
Workshop 05 - Raw Transport 46
Deployment Description 46
Problem Statement 46
NEM Attributes 46
NEM Parameters 47
Raw Transport Parameters 47
XML Listing 47
Concept Review 49
Workshop 06 - Interactive Transport 50
Deployment Description 50
Problem Statement 50
NEM Attributes 51
NEM Parameters 51
Interactive Transport Parameters 51
Interactive Commands 51
XML Listing 52
Concept Review 53
IV. Network Emulation Models 54
Models 55
RF Pipe Model 56
MAC Details 56
PHY Details 56
IEEE 802.11 abg Model 58
MAC Details 59
PHY Details 59
Comm Effect Model 61
Shim Details 61
Workshop 07 - RF Pipe Model 62
Deployment Description 62
Problem Statement 62
RF Pipe MAC Parameters 63
RF Pipe PHY Parameters 63
Interactive Commands 66
XML Listing 66
Concept Review 67
Workshop 08 - IEEE 802.11 abg Model Model 69
Deployment Description 69
Problem Statement 69
IEEE 802.11 abg MAC Parameters 70
IEEE 802.11 abg PHY Parameters 75
Interactive Commands 77
XML Listing 78
Observations Explained 79
Concept Review 80
Workshop 09 - Comm Effect Model Model 81
Deployment Description 81
Problem Statement 81

Comm Effect Shim Parameters 82
Interactive Commands 83
XML Listing 84
Concept Review 85
V. Event Generators 86
Generators 87
Mitre Mobility Generator 88
Generator Details 88
Generator Data Format 88
Data Format Sample 89
Emulation Script Generator 90
Generator Details 90
Data Format Sample 90
Emulation Event Log Generator 91
Generator Details 91
Generator Data Format 91
Data Format Sample 92
Workshop 10 - Mitre Mobility Event Generator 93
Deployment Description 93
Problem Statement 93
Mitre Mobility Generator Parameters 94
XML Listing 95
Concept Review 95
Workshop 11 - Emulation Script Generator 97
Deployment Description 97
Problem Statement 97
Emulation Script Generator Parameters 98
RF Pipe PHY Parameters 98
XML Listing 101
Concept Review 102
Workshop 12 - Emulation Event Log Generator 103
Deployment Description 103
Problem Statement 103
Emulation Event Log Generator Parameters 104
XML Listing 104
Concept Review 104
VI. Event Agents 105
Agents 106
GPSd Location Agents 107
Workshop 13 - GPSd Location Agent 108
Deployment Description 108
Problem Statement 108
GPSd Location Agent Parameters 108
XML Listing 108
Concept Review 109
VII. Tips 110
Tips 111
VIII. Controllers 114
Controllers 115
Workshop 14 - EMANE Loss Controller 117
Deployment Description 117
Problem Statement 117
XML Listing 117
Concept Review 118

Workshop 15 - EMANE Comm Effect Controller 119
Deployment Description 119
Problem Statement 119
XML Listing 119
Concept Review 120
IX. More On Deployments 121
Workshop 16 - Surrogate Radio Models using RF Pipe 122
Deployment Description 122
Problem Statement 122
XML Listing 123
X. Advanced Event Capabilities 124
Event Service Language Bindings 125
Workshop 17 - libemaneeventservice 126
Description 126
Publisher Sample Application 126
Subscriber Sample Application 126
libemaneeventservice Configuration 126
Configuration Format 127
Workshop 18 - libemaneeventservice Perl language bindings 128
Description 128
Sample Scripts 128
Workshop 19 - libemaneeventservice Python language bindings 129
Description 129
Sample Scripts 129
XI. Putting It All Together 130
Workshop 20 - Putting it All together: Distributed RF Pipe MANET 131
Deployment Description 131
Workshop 21 - Putting it All together: Centralized RF Pipe MANET 132
Deployment Description 132
Workshop 22 - Putting it All together: Distributed Comm Effect MANET 133
Deployment Description 133
Workshop 23 - Putting it All together: Centralized Comm Effect MANET 134
Deployment Description 134
Workshop 24 - Putting it All together: Centralized IEEE 802.11 abg MANET 135
Deployment Description 135
XII. Manual Pages 136
C emaneEventService 137
C emaneEventLocation 141
C emaneEventPathloss 145
C emaneEventCommEffect 149
Perl EMANE::EventService 153
Perl EMANE::Event::Location 156
Perl EMANE::Event::Pathloss 158
Perl EMANE::Event::CommEffect 161
Python emaneeventservice 164
Python emaneeventlocation 166
Python emaneeventservice 168
Python emaneeventservice 170

Preface

How to use this workbook

This workbook is part of the EMANE User Training course. This workbook accompanies a set of workshop XML examples that are used to introduce the key concepts and features of EMANE components and EMANE add-on projects. Along with the workbook and XML workshops an EMANE Virtual Machine containing a complete build and execution environment for all the latest EMANE components is used as a training platform.

Creating an EMANE training installation

An EMANE training installation is a linux machine with all existing EMANE components installed. The goal is to provide a student with complete access to all the applications, component plugins, man pages, and documentation available. A training installation does not have to be a virtual machine.

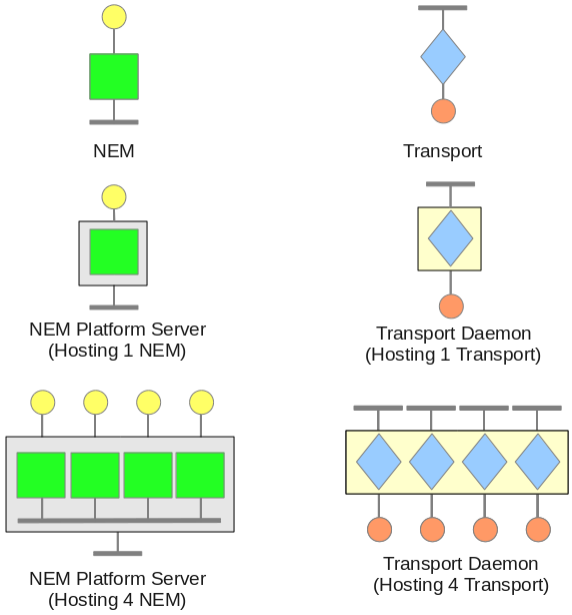
All the packages and training material used during EMANE User Training are available at the *EMANE Project Homepage* [http://labs.cengen.com/emane].

This training course covers the following EMANE components:

emane	0.6.2
emanecommeffect	0.4.1
emanecommeffectcontroller	0.4.1
EMANE-Event-CommEffect	0.4.2
EMANE-Event-Location	0.3.2
EMANE-Event-Pathloss	0.3.2
EMANE-EventService	0.3.2
emanelosscontroller	0.3.2
emanetransinteractive	0.1.2
emanetransplug	0.2.2
emanegeneel	0.0.1_pre1
libemaneeventservice	0.4.2
python-emaneeventcommeffect	0.4.2
python-emaneeventlocation	0.3.2
python-emaneeventpathloss	0.3.2
python-emaneeventservice	0.3.2

Reading an EMANE Deployment Diagram

Many of the workshop assignments found throughout the workbook 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.



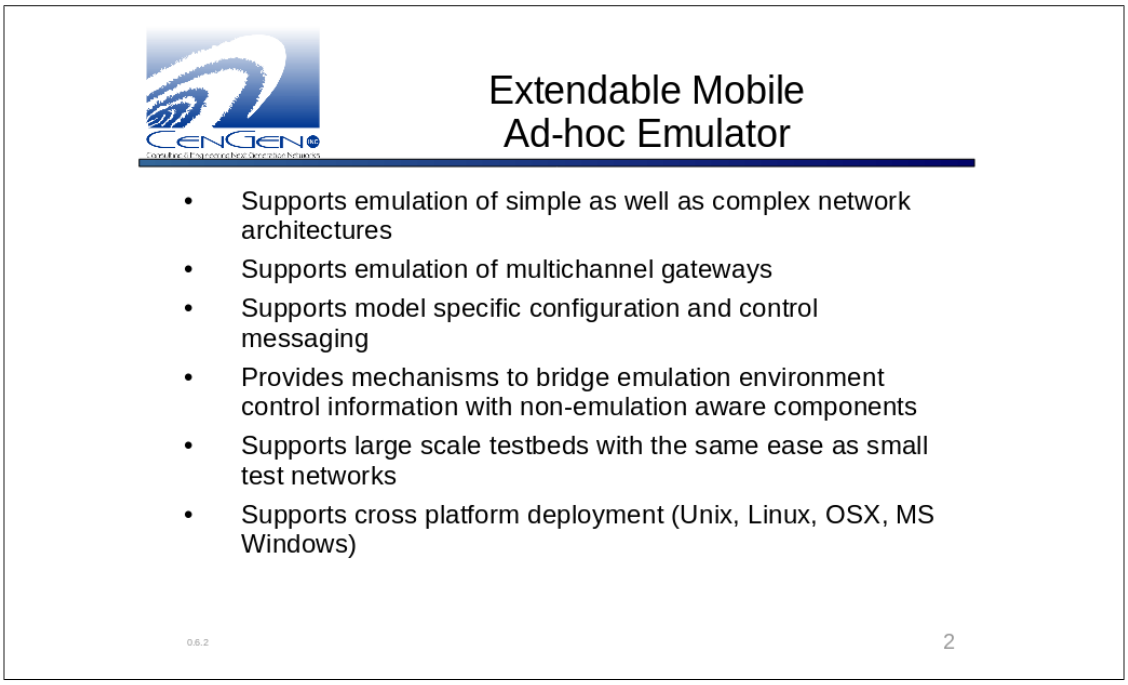
Part I. Overview

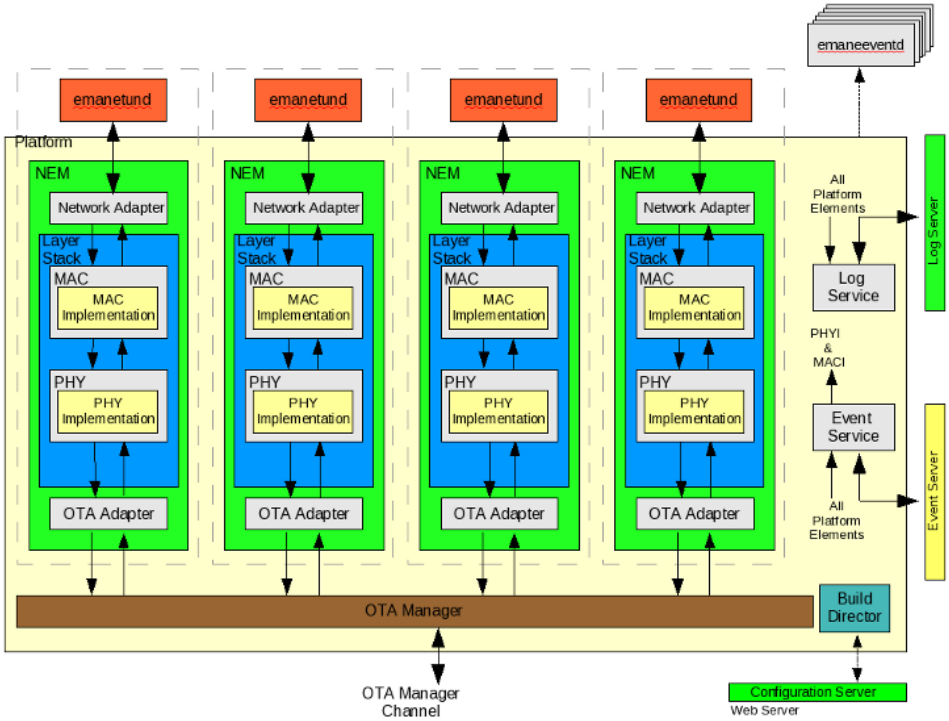
Contents

Introduction 2
Emulation Stack 4
Component Communication 6
Emulation Environmental Data 8
Applications and Utilities 9
XML Hierarchy 11
Workshop 00 - EMANE Applications and Documentation 13
    Problem Statement 13

Introduction

In its most basic form, EMANE is an infrastructure used for Mobile Ad-hoc Emulation. EMANE provides a set of APIs to allow independent development of network emulation modules (NEMs), emulation/application boundary interfaces (transports), and emulation environmental data distribution mechanisms (events).





The key to the flexibility of EMANE is the XML based factories that are used to determine where emulation components and interfaces reside once deployed. There are four types of component containers that can be configured to create and manage a variable number of items:

- **NEM Platform Server** - Creates and manages network emulation modules
- **Transport Daemon** - Creates and manages emulation/application boundary interfaces
- **Event Service** - Creates and manages emulation environmental data generators
- **Event Daemon** - Creates and manages boundary agents that bridge emulation environmental data from the emulation space to the application space.

EMANE Internals: Required Toolkits

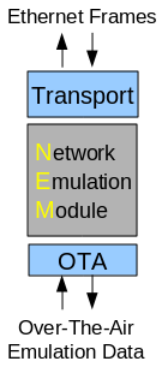
EMANE is built on top of the ADAPTIVE Communication Environment (ACE). For more information see: *The ADAPTIVE Communication Environment* [ttp://www.cs.wustl.edu/~schmidt/ACE.html/]

EMANE uses libxml to process all of its XML configuration. For more information see: *The XML C parser and toolkit of Gnome* [http://xmlsoft.org/]

Emulation Stack

Each node in the emulation is represented by an instance of an emulation stack. This stack encapsulates the functionality necessary to transmit, receive, and operate on data routed through the emulation space.

Emulation Stack Anatomy



- *Transport* – Mechanism responsible for transporting data to and from the emulation space. (Emulation/Application boundary interface)
- *Network Emulation Module* (NEM) - Emulation implementation functionality for a given radio model
- *Over-The-Air Manager* - Provides the mechanism NEMs use to communicate

0.6.2 4

Transport

- Realization of an emulation/application boundary interface. Provides the *entry* and *exit* point for all data routed through the emulation.

  For example the two ethernet frame transports: Virtual Transport and Raw Transport are responsible for interfacing with the underlying operating system

  - TunTap (linux, OS X)
  - libpcap (linux, OS X)
  - WinTap (win32)
  - winpcap (win32)

  Supports:

  - IPv4
  - Unicast
  - Multicast
  - IPv6
  - Broadcast
  - Throughput limitation
- Tunnels *transport data* as opaque payload to the corresponding NEM.

0.6.2 5

Network Emulation Module

- NEMs composed of two components

  MAC Layer – Medium Access Control Layer emulation functionality
  - CSMA
  - TDMA & Hybrid schemes
  - Queue Management, Discard, QOS
  - Adaptive Transmission Protocols (Power and Data Rate)

  PHY Layer – Physical Layer emulation functionality
  - Filter out of band packets
  - Waveform Timing
  - Half duplex operations
  - Transmit Power and antenna gain
  - Directional antenna support
  - Account for noise/collision
  - Probability of reception based on BER, RSSI, Packet Size

0.6.2 6

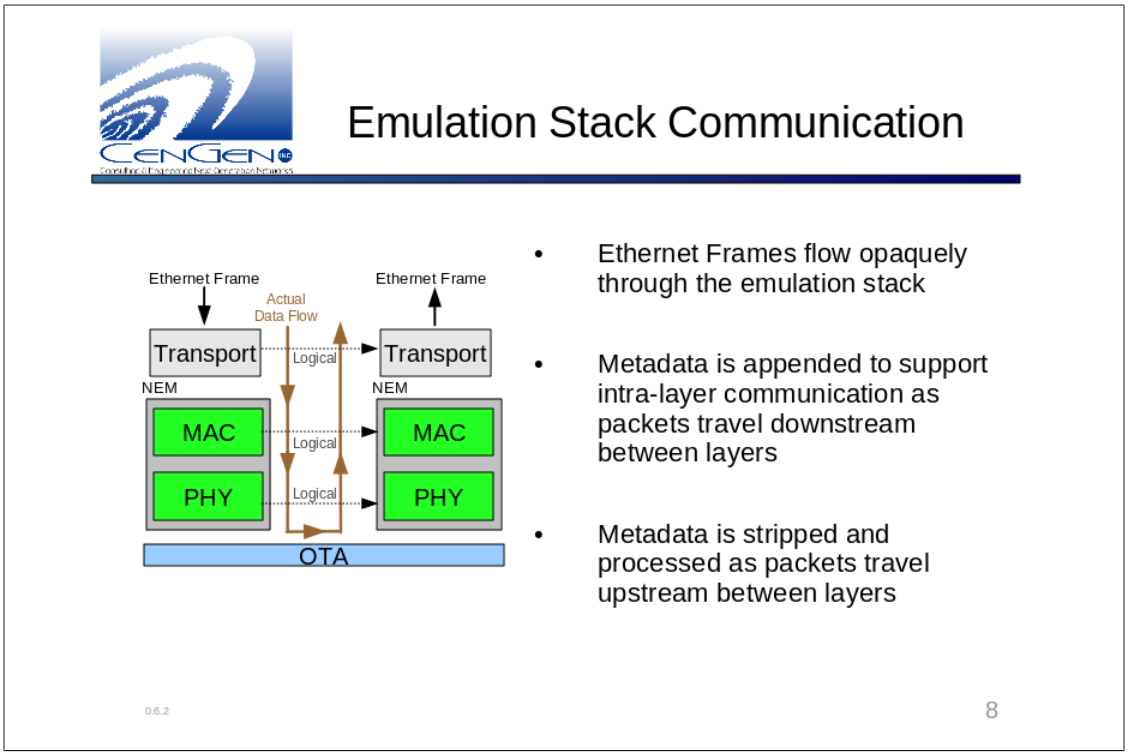
Over-The-Air Manager

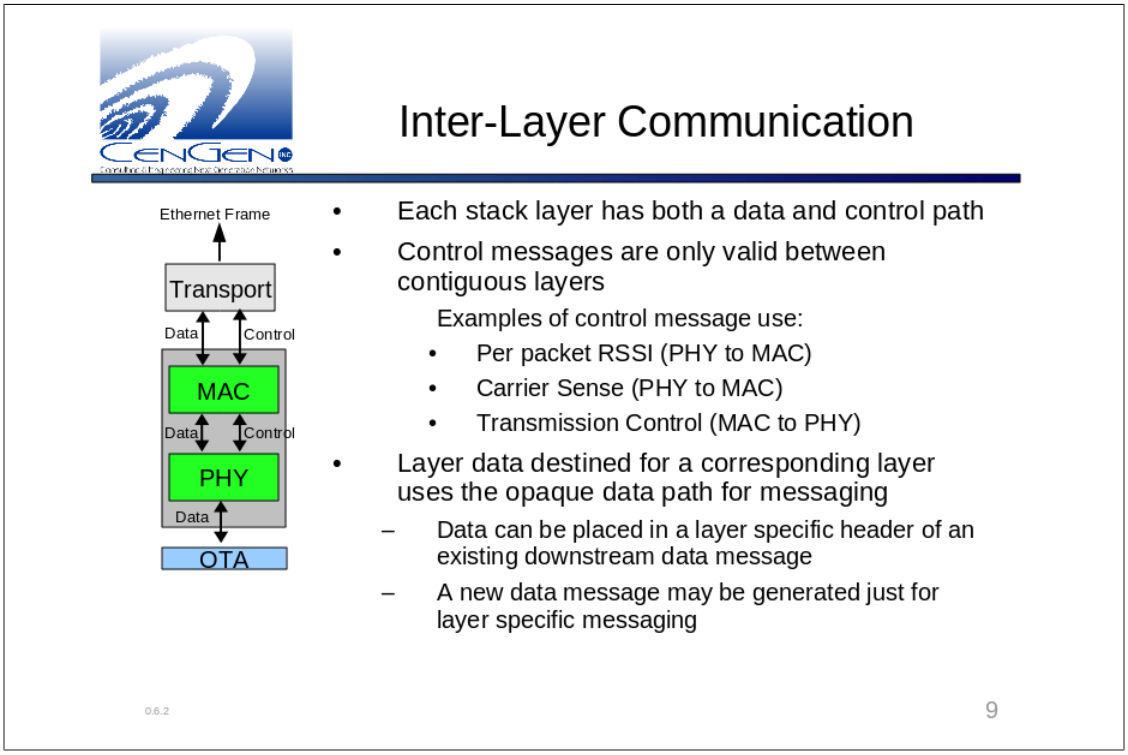
- Messaging infrastructure to deliver emulation radio model messages to all nodes in the deployment
  - Uses a multicast channel for message distribution for NEMs hosted by an NEM server different than that of the source NEM
  - Uses shared memory for message distribution for NEMs hosted by the same NEM server
- All messages are delivered to every NEM participating in the emulation (PHY Layer activity)
  - Provides the ability to model complex PHY phenomena such as RF interference
  - Multiple OTA mulitcast channels can be used to reduce overhead
  - OTA multicast channel can be disabled when utilizing a single NEM server

0.6.2 7

Component Communication

Emulation stack components communicate generically using an upstream/downstream message passing protocol. Mechanisms exist to transmit and receive control information between two contiguous component layers. Conceptually instances of the same layer communicate horizontally using their respective protocol.





PHY Layer Communication

- Radio PHY Models use a common header to allow cooperation between heterogeneous models in the same deployment
  - PHY Type
  - Source
  - Destination
  - Time
  - Message Duration
  - Center Frequency
  - Bandwidth
  - Data Rate
  - Antenna Gain
  - Antenna Beam Width
  - Antenna Elevation
  - Antenna Azimuth
  - Tx Power
- Data used to implement a higher fidelity RF model
  - Noise floor calculations
  - Frequency interference
  - Directional antenna support

0.6.2 10

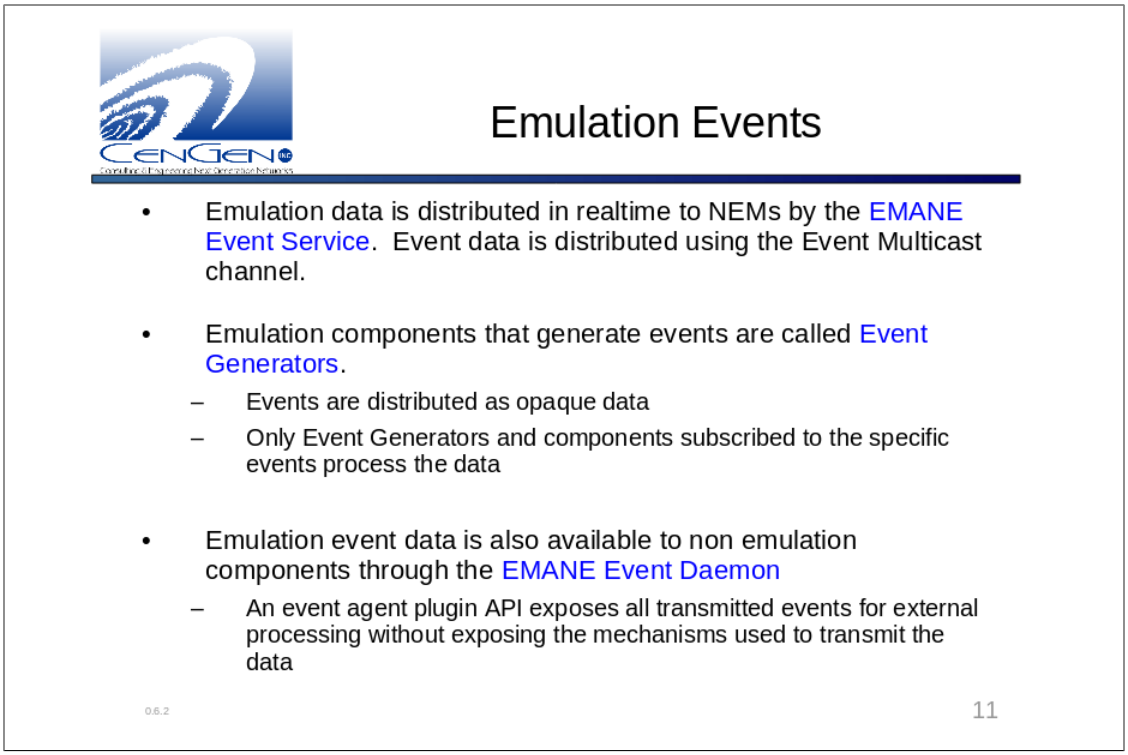
EMANE Internals: Generic Component Messaging API

A generic upstream/downstream API is used to pass data and control information between component layers. For more information see: *EMANE Developers Manual*.

```
namespace EMANE
{
  class UpstreamTransport
  {
  public:
    virtual
    void processUpstreamPacket(UpstreamPacket & pkt,
                               const ControlMessage & msg) = 0;
    virtual
    void processUpstreamControl(const ControlMessage & msg) = 0;

    ...
  };
}

namespace EMANE
{
  class DownstreamTransport
  {
  public:
    virtual
    void processDownstreamPacket(DownstreamPacket & pkt,
                                 const ControlMessage & msg) = 0;
    virtual
    void processDownstreamControl(const ControlMessage & msg) = 0;

    ...
  };
}
```

Emulation Environmental Data

Emulation Environmental Data such as location information, path loss, etc. can be transmitted to any interested component layer allowing realtime dissemination of information.



There are currently three types of EMANE events part of the EMANE distribution:

- **Pathloss** - Contains a variable list of path loss and reverse path loss values between the receiving NEM and one or more NEMs in the deployment.
- **Location** - Contains a variable list of one or more NEM locations: latitude, longitude, and altitude.
- **Comm Effect** - Contains a variable list of communication effects between the receiving NEM and one or more NEMs in the deployment. Communication effects are composed of the following: probability of loss, probability of duplication, latency, jitter, unicast, and broadcast bit rates. These effects can be assigned asymmetrically on a per like basis.

Component	Location	Pathloss	Comm Effect
RF Pipe	●	●	
IEEE 802.11 abg		●	
Comm Effect			●
GPSd Location Agent	●		

EMANE Component Event Usage

Applications and Utilities

EMANE Applications

- *emane* – NEM Platform Server application. Creates and manages one or more NEMs.
  - Input XML: platform file
- *emanetransportd* – Transport Daemon application. Creates and manages one or more transports.
  - Input XML: transportdaemon file
- *emaneeventd* – Event Daemon application. Creates and manages one or more event agents.
  - Input XML: eventdaemon file
- *emaneeventservice* – Event Service application. Creates and manages one or more event generators.
  - Input XML: eventservice file

0.6.2 14

EMANE Utilities

- *emaneloglevel* – Dynamical modify the log level of a running emane process
- *emanegentransportxml* – Generate corresponding transportdaemon XML from a platform XML file.
  - Input XML: platform file
- *emanegendeploymentxml* – Generate corresponding deployment XML from a platform XML file.
  - Input XML: platform file
- *emanegenxml* – Generate all XML required for an EMANE deployment from a properly formatted text input file
  - Input File: emanegenxml input file

0.6.2 15

EMANE Internals: Application Documentation

All EMANE applications and utilities have their own dedicated man page. Manual page for **emanegentransportxml**:

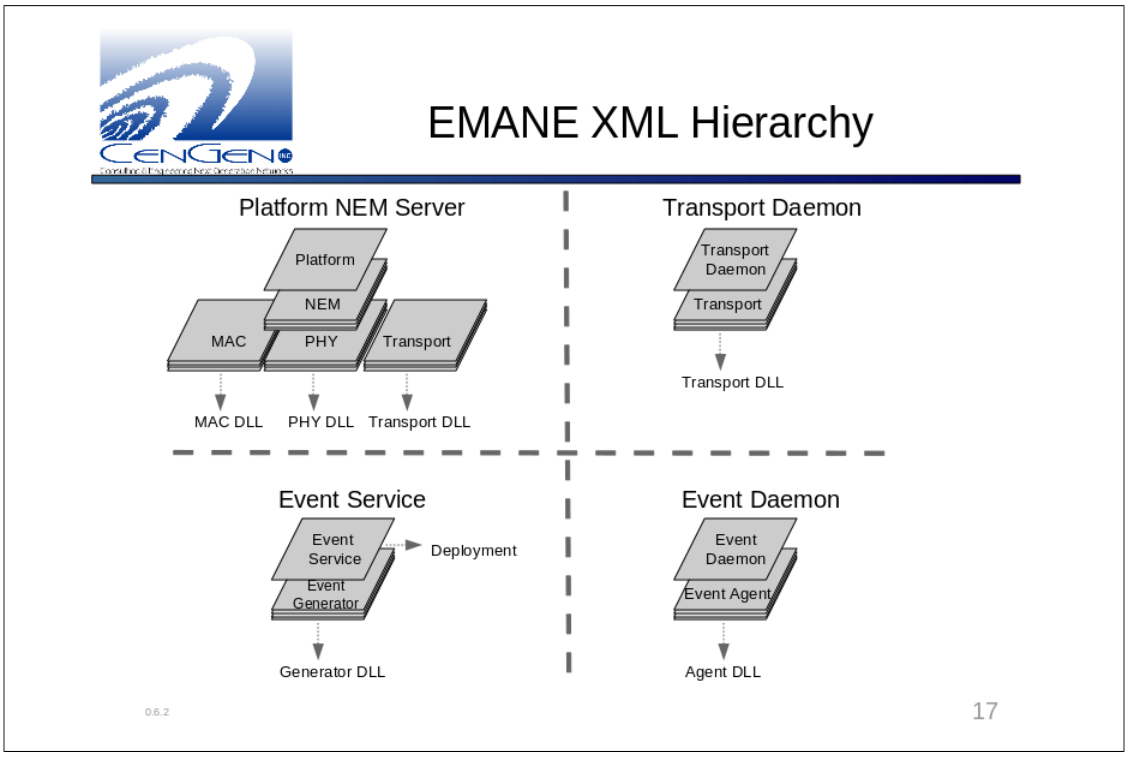
```
          emanegentransportxml(1)                                emanegentransportxml(1)

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.

Example
       Example application command line:
              emanegentransportxml --platform /var/www/html/xml/platform.xml

       Example transport XML output:

              <?xml version="1.0" encoding="UTF-8"?>
              <!DOCTYPE transportdaemon SYSTEM "file:///usr/share/emane/dtd/transportdaemon.dtd">
              <transportdaemon>
                <instance nemid="1">
                  <param name="platformendpoint" value="localhost:50001"/>
                  <param name="transportendpoint" value="localhost:40001"/>
                  <transport definition="transvirtual.xml">
                    <param name="bitrate" value="0"/>
                    <param name="address" value="172.30.1.1"/>
                    <param name="device" value="emane0"/>
                  </transport>
                </instance>
              </transportdaemon>


VERSION
       0.6.1

BUGS
       To report a bug, please send and email to <labs at cengen dot com>

SEE ALSO
       emane(1) emanetund(1)

AUTHORS
       CenGen Inc. Columbia, MD
       http://cengen.com

NOTES
       EMANE is released under the BSD License.

emanegentransportxml 0.6.1      Wed Jan 13 2010        emanegentransportxml(1)
```

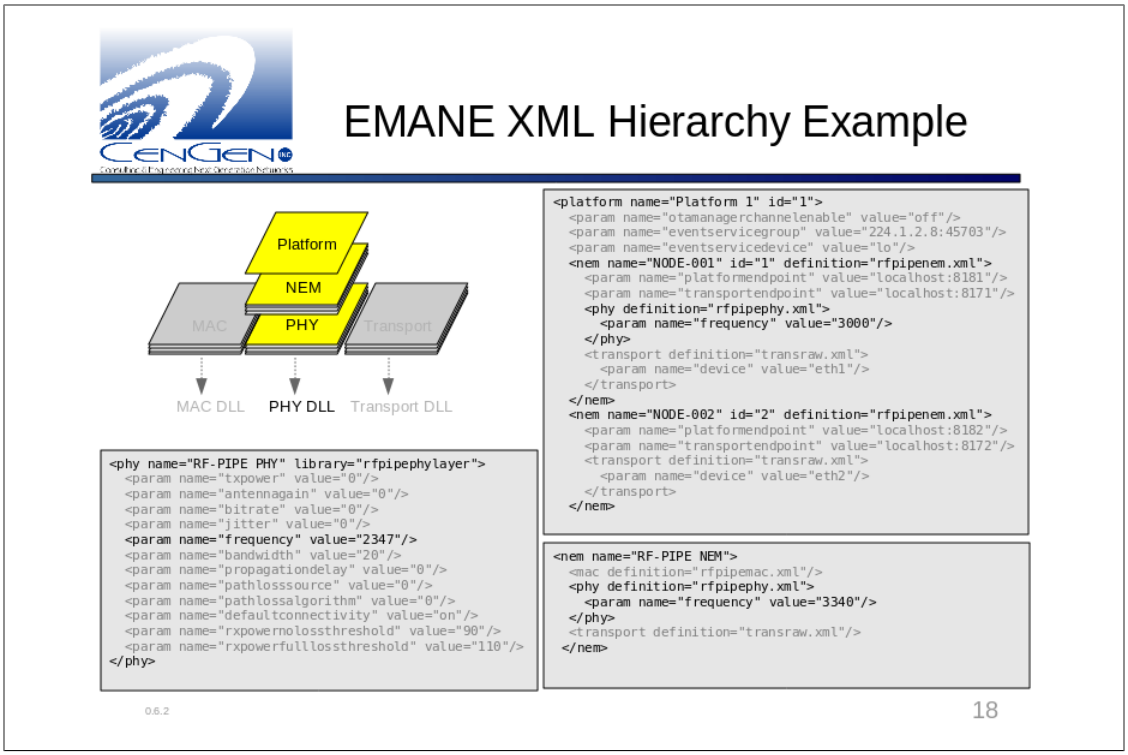
XML Hierarchy

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 any 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 combing 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. Refer to individual EMANE component documentation for details on required and optional configuration parameters.





In this example the lowest level PHY Layer configuration contained in *rfpipephy.xml* has the *frequency* parameter set to 2347 MHz. The NEM defined in *rfpipenem.xml* uses the PHY Layer defined in *rfpipephy.xml* but overrides the *frequency* parameter 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 *rfpipephy.xml* is never used.

Workshop 00 - EMANE Applications and Documentation

The goal of this workshop is to become familiar with the EMANE applications and documentation.

Problem Statement

Examine the man pages of each of the EMANE applications and utilities. Locate and examine the available documentation and sample XML.

Application man pages:

- **emane** - NEM Platform Server

```
> man emane
```

- **emanetransportd** - Transport Daemon

```
> man emanetransportd
```

- **emaneeventservice** - Event Service

```
> man emaneeventservice
```

- **emaneeventd** - Event Daemon

```
> man emaneeventd
```

Utility man pages:

- **emaneloglevel** - Dynamically modify log level of running **emane** process

```
> man emaneloglevel
```

- **emanegentransportxml** - Generate transport XML from platform XML

```
> man emanegentransportxml
```

- **emanegendeploymentxml** - Generate deployment XML from platform XML

```
> man emanegendeploymentxml
```

- **emanegenxml** - Generate all XML required for an EMANE deployment

```
> man emanegenxml
```

Part II. Deployments

Contents

Deployments 15
- The Shim Layer 18
- Structured and Unstructured NEM 19

Workshop 01 - Centralized Deployment 20
- Deployment Description 20
- Problem Statement 20
- Platform Attributes 21
- Platform Parameters 21
- NEM Attributes 22
- NEM Parameters 22
- XML Listing 22
- Concept Review 23

Workshop 02 - Distributed Deployment 24
- Deployment Description 24
- Problem Statement 24
- Platform Attributes 25
- Platform Parameters 25
- NEM Attributes 26
- NEM Parameters 26
- XML Listing 26
- Concept Review 28

Workshop 03 - Hybrid Deployment 29
- Deployment Description 29
- Problem Statement 29
- Platform Attributes 30
- Platform Parameters 30
- NEM Attributes 31
- NEM Parameters 31
- XML Listing 31
- Concept Review 34

Deployments

Emulation Deployment

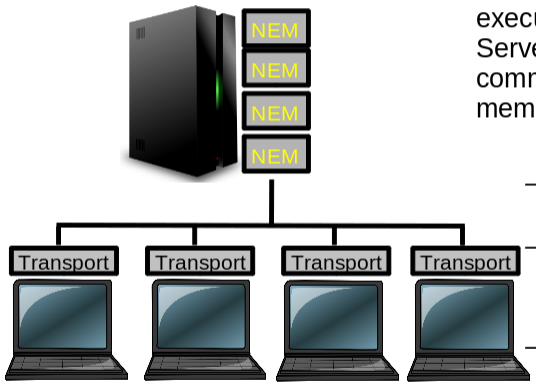
There are three different configurations for EMANE deployments:

- Centralized Mode – All emulation is performed on a central server
- Distributed Mode – Emulation is distributed across test nodes
- Hybrid Mode – A mixture of Centralized Mode and Distributed Mode test nodes

Modes only define the relationship between NEMs and NEM Platform Servers in the deployment.

0.6.2

20



All emulation functionality (NEM) is executed on a single NEM Platfrom Server. All OTA Manager communication is done using shared memory.

Pros:

- Allows dedicated emulation processing
- Simplifies emulator control and configuration deployment

Cons:

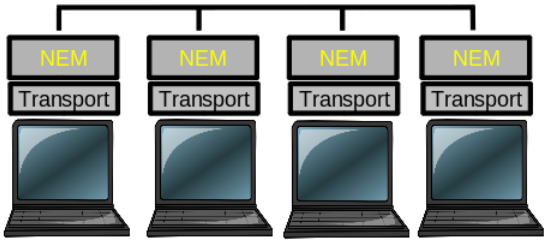
- Larger centralized computing power required to emulate large networks

0.6.2

21

Distributed Deployment

Emulation functionality (NEM) distributed across test nodes. OTA Manager communication is done using a multicast channel.



Pros:

- Distributes emulation functionality across all test nodes
- Easier to grow emulation deployment

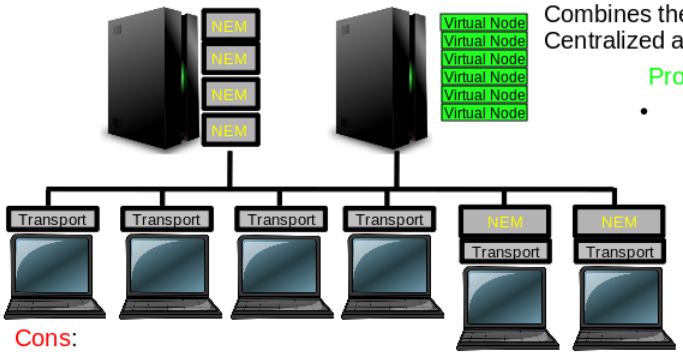
Cons:

- Slightly more complex to control
- Timing accuracy must be addressed since emulation spans multiple RTCs

0.6.2

22

Hybrid Deployment



Combines the best features of Centralized and distributed mode

Pros:

- Flexibility of growing and shrinking emulation deployment as asset become available

Cons:

- Complex to control
- Timing accuracy must be addressed

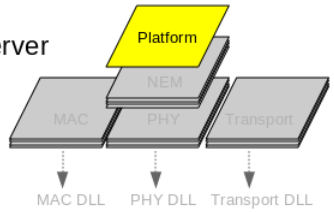
0.6.2

23

Platform XML Explained

Platform NEM Server XML is the root of all EMANE configuration

- Transport daemon XML and deployment XML automatically generated from platform XML
- Specify NEMs created by the platform server
  - NEM types (can be heterogeneous)
  - Associate transports
    - Transport Types
    - Endpoint connection information
- Specify configuration parameter values
  - OTA Manager multicast channel (group, port, and device)
  - Event multicast channel (group, port, and device)
  - Override values specified in the PHY, MAC, and transport XML



0.6.2

24

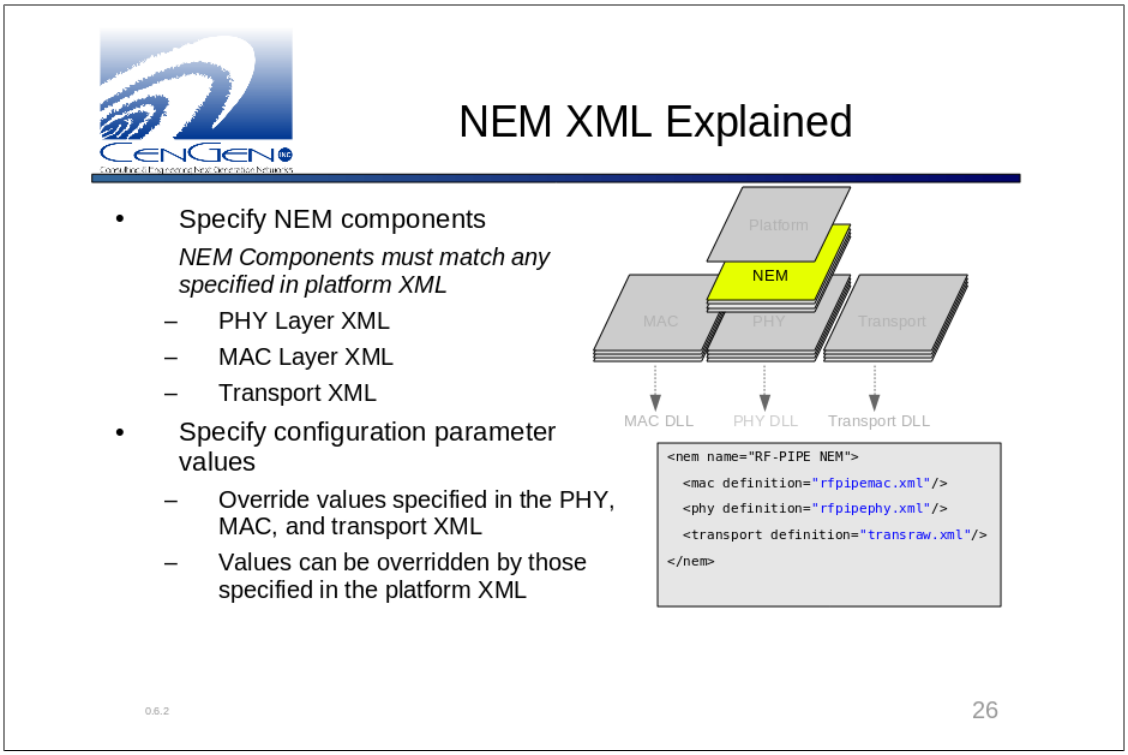
Platform XML Sample

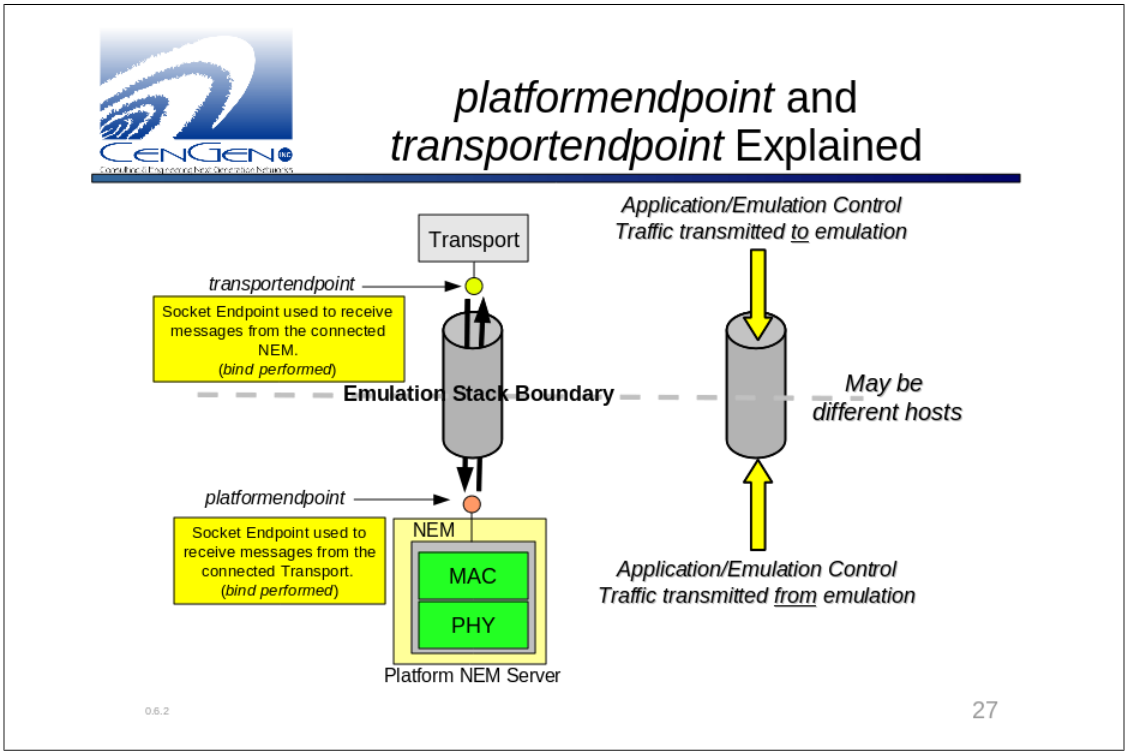
- Platform NEM Server (`emane`), transport daemon (`emanetransportd`), and event service (`emaneeventservice`) reside on the same machine
  - *platformendpoint* and *transportendpoint* using localhost
  - *eventservicedevice* using loopback
- NEMs using same definition: *rfpipenem.xml*
- NEMs using *transraw.xml*
  - Overriding transport *device* configuration parameter to associate different devices
  - Using a transport *group* tag to combine into a single transport daemon

```
<platform name="Workstation 1" id="1">
  <param name="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="otamanagerdevice" value="eth0"/>
  <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"/>
    <transport definition="transraw.xml" group="single">
      <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" group="single">
      <param name="device" value="eth2" />
    </transport>
  </nem>
</platform>
```

0.6.2

25

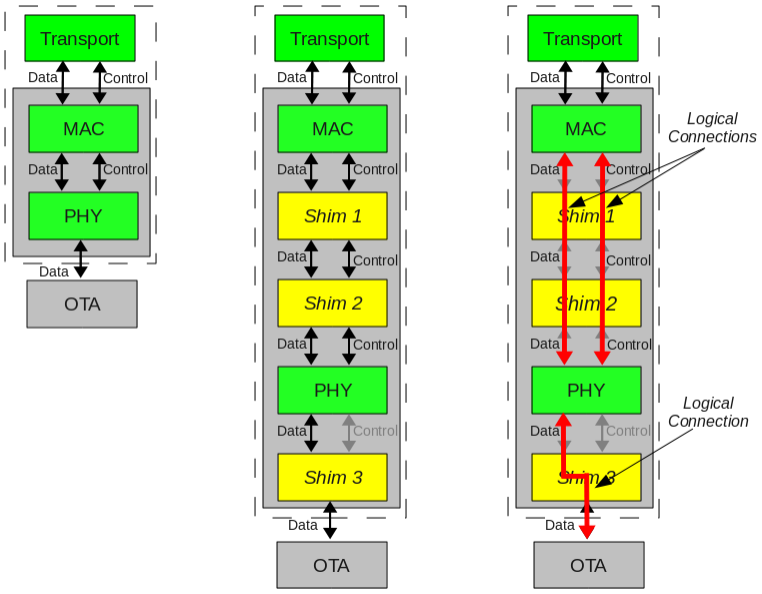




The Shim Layer

In addition to the three required NEM components, MAC, PHY, and Transport, NEMs can contain one or more Shims. From an API point of view a Shim layer has the exact same interface, upstream and downstream, as the three required NEM components. But unlike the three required component parts of an NEM a Shim layers can reside at any position in the NEM component stack.

A Shim provides a mechanism to insert functionality between layers to gain access to data and control messages traveling up and down the stack



NEM with three Shim Layers

Structured and Unstructured NEM

There area actually two modes of NEM creation, *structured* and *unstructured*. An NEM is *structured* by default.

A *structured* NEM must be composed of one MAC layer, one PHY layer, and one Transport layer. It may also contain one or more Shim layers. The positions of the three mandatory layers is fixed within the NEM stack. From top to bottom the order of components in a *structured* NEM is MAC, PHY, and then Transport. One or more Shims may appear anywhere in the NEM layer stack.

An *unstructured* NEM may contain any number of layers in any order. Unstructured NEMs provide EMANE component developers the ability to circumvent the internal builder logic used when creating an NEM.

The Comm Effect model is an example of a Shim only *unstructured* NEM:

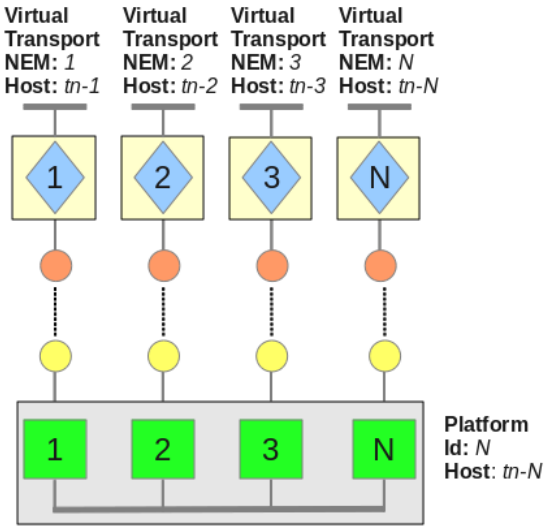
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="COMMEFFECT NEM"  type="unstructured">
  <shim definition="commeffectshim.xml"/>
  <transport definition="transraw.xml"/>
</nem>
```

Workshop 01 - Centralized Deployment

The goal of this workshop is to become familiar with configuring a centralized EMANE deployment.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Virtual Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 4 NEMs with incomplete configuration. Modify the NEM Platform Server configuration to achieve the following:

- Use Platform Id N, where N is your Training Id.
- Disable OTA Manager channel communication.
- The event service multicast group is *224.1.2.8* and port *45703*.
- The event service device is *lo*.
- All NEMs should use the RF Pipe Model.
- Use NEM Ids 1, 2, 3, and N, where N is your Training Id.
- Use NEM Platform Server *tn-N* or *10.99.0.N*, where N is your Training Id.
- Use test node addresses *tn-1*, *tn-2*, *tn-3*, and *tn-N* or *10.99.0.1*, *10.99.0.2*, *10.99.0.3*, and *10.99.0.N*, where N is your Training Id.
- For NEM N use the virtual interface address *radio-N* or *10.100.0.N*, where N is your Training Id.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Platform Attributes

Name:	**name**
Description:	Descriptive name of the platform.
Type:	string

Name:	**id**
Description:	Platform identifier. **Must be unique for all platforms in the deployment.**
Type:	UINT16

Platform Parameters

Name:	**otamanagergroup**
Description:	The OTA Manager mulitcast endpoint that is used to communicate between all platforms in a test deployment. GROUP:PORT format. **Required only for multi-platform EMANE deployments.**
Note:	See *otamanagerdevice*.
Type:	string
XML Format:	<param name="otamanagergroup" value="224.1.2.8:45702"/>

Name:	**otamanagerdevice**
Description:	The network device to associate with OTA Manager mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *otamanagergroup*.
Type:	string
XML Format:	<param name="otamanagerdevice" value="eth1"/>

Name:	**otamanagerchannelenable**
Description:	The OTA Manager Channel Enable parameter enables or disables the OTA Manager channel. When disabled there is no platform to platform communication. Only locally managed NEMs will be able to communicate with each other.
Type:	bool
XML Format:	<param name="otamanagerchannelenable" value="on"/>

Name:	**eventservicegroup**
Description:	The Event Service multicast endpoint that is used to transmit events to and from the platform. GROUP:PORT format.
Note:	See *eventservicedevice*.
Type:	string
XML Format:	<param name="eventservicegroup" value="224.1.2.8:45703"/>

Name:	**eventservicedevice**
Description:	The network device to associate with Event Service mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *eventservicegroup*.
Type:	string
XML Format:	<param name="eventservicedevice" value="eth1"/>

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

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

<!--
   * Set Platform name and id
   * Set otamanagerchannelenable value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform XXX" id="XXX">
  <param name="otamanagerchannelenable" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>

  <!--
     * Set NEM 1 definition
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-001" id="1" definition="XXX">
    <param name="platformendpoint" value="XXX:8101"/>
    <param name="transportendpoint" value="XXX:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-1" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>

  <!--
     * Set NEM 2 name, id and definition
     * Set platform endpoint address
```

```
     * Set transport endpoint address
    -->
  <nem name="NODE-XXX" id="XXX" definition="XXX">
    <param name="platformendpoint" value="XXX:8102"/>
    <param name="transportendpoint" value="XXX:8202"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-2" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>

  <!--
     * Set NEM 3 name, id and definition
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-XXX" id="XXX" definition="XXX">
    <param name="platformendpoint" value="XXX:8103"/>
    <param name="transportendpoint" value="XXX:8203"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-3" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>

  <!--
     * Set NEM N name, id and definition
     * Set platform endpoint address
     * Set transport endpoint address
     * Set transport parameter address
    -->
  <nem name="NODE-XXX" id="XXX" definition="XXX">
    <param name="platformendpoint" value="XXX:8104"/>
    <param name="transportendpoint" value="XXX:8204"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-XXX" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Review the **rfpipenem.xml** file.

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

Concept Review

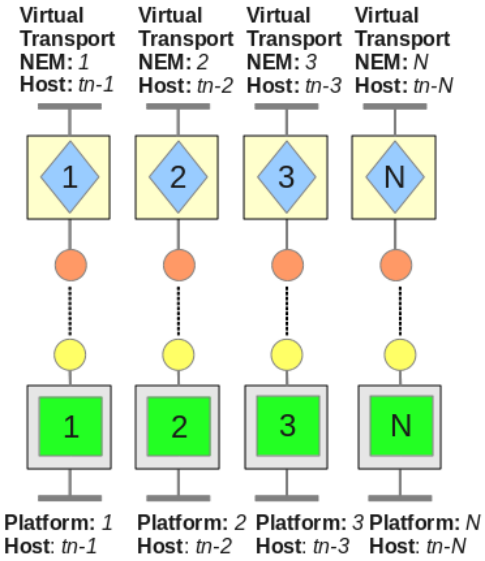
1. Why was the OTA Manager channel disabled in this workshop?
2. How do NEMs hosted by the same NEM Platform Server communicate?
3. What can be deduced from setting the Event Service device to *lo*?
4. What are the implications of not specifying an Event Service device?
5. What are the Platform *name* and NEM *name* XML attributes used for?

Workshop 02 - Distributed Deployment

The goal of this workshop is to become familiar with configuring a distributed EMANE deployment.

Deployment Description

This workshop contains a distributed RF Pipe Deployment using Virtual Transports. There are 4 NEM Platform Servers each hosting a single NEM.



Problem Statement

The 4 NEM Platform Server files `platform1.xml`, `platform2.xml`, `platform3.xml`, and `platformN.xml` contain incomplete configuration. Modify the NEM Platform Server configurations to achieve the following:

- Use Platform Ids 1, 2, 3, and N, where N is your Training Id.
- The OTA Manager mulitcast group is *224.1.2.8* and port *45702*.
- The OTA Manager device is *eth1*.
- The event service multicast group is *224.1.2.8* and port *45703*.
- The event service device is *eth1*.
- All NEMs should use the RF Pipe Model.
- Use NEM Ids 1, 2, 3, and N, where N is your Training Id.
- Use test node addresses *tn-1*, *tn-2*, *tn-3*, and *tn-N* or *10.99.0.1*, *10.99.0.2*, *10.99.0.3*, and *10.99.0.N*, where N is your Training Id.
- For NEM N use the virtual interface address *radio-N* or *10.100.0.N*, where N is your Training Id.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Platform Attributes

Name:	**name**
Description:	Descriptive name of the platform.
Type:	string

Name:	**id**
Description:	Platform identifier. **Must be unique for all platforms in the deployment.**
Type:	UINT16

Platform Parameters

Name:	**otamanagergroup**
Description:	The OTA Manager mulitcast endpoint that is used to communicate between all platforms in a test deployment. GROUP:PORT format. **Required only for multi-platform EMANE deployments.**
Note:	See *otamanagerdevice*.
Type:	string
XML Format:	<param name="otamanagergroup" value="224.1.2.8:45702"/>

Name:	**otamanagerdevice**
Description:	The network device to associate with OTA Manager mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *otamanagergroup*.
Type:	string
XML Format:	<param name="otamanagerdevice" value="eth1"/>

Name:	**otamanagerchannelenable**
Description:	The OTA Manager Channel Enable parameter enables or disables the OTA Manager channel. When disabled there is no platform to platform communication. Only locally managed NEMs will be able to communicate with each other.
Type:	bool
XML Format:	<param name="otamanagerchannelenable" value="on"/>

Name:	**eventservicegroup**
Description:	The Event Service multicast endpoint that is used to transmit events to and from the platform. GROUP:PORT format.
Note:	See *eventservicedevice*.
Type:	string
XML Format:	<param name="eventservicegroup" value="224.1.2.8:45703"/>

Name:	**eventservicedevice**
Description:	The network device to associate with Event Service mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *eventservicegroup*.
Type:	string
XML Format:	<param name="eventservicedevice" value="eth1"/>

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

XML Listing

Modify the **platform1.xml** file to achieve the workshop goals.

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

<!--
   * Set otamanagergroup value
   * Set otamanagerdevice value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform 1" id="1">
  <param name="otamanagergroup" value="XXX"/>
  <param name="otamanagerdevice" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>

  <!--
     * Set NEM 1 definition
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8101"/>
    <param name="transportendpoint" value="XXX:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-1" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Modify the **platform2.xml** file to achieve the workshop goals.

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

<!--
   * Set otamanagergroup value
   * Set otamanagerdevice value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform 2" id="2">
  <param name="otamanagergroup" value="XXX"/>
  <param name="otamanagerdevice" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>

  <!--
     * Set NEM 2 definition
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8102"/>
    <param name="transportendpoint" value="XXX:8202"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-2" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Modify the **platform3.xml** file to achieve the workshop goals.

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

<!--
   * Set otamanagergroup value
   * Set otamanagerdevice value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform 3" id="3">
  <param name="otamanagergroup" value="XXX"/>
  <param name="otamanagerdevice" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>

  <!--
     * Set NEM 3 definition
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8103"/>
    <param name="transportendpoint" value="XXX:8203"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-3" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>

</platform>
```

Modify the **platformN.xml** file to achieve the workshop goals.

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

```
<!--
   * Set otamanagergroup value
   * Set otamanagerdevice value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform XXX" id="XXX">
  <param name="otamanagergroup" value="XXX"/>
  <param name="otamanagerdevice" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>

  <!--
     * Set NEM N name, id, and  definition
     * Set platform endpoint address
     * Set transport endpoint address
     * Set transport parameter address
    -->
  <nem name="NODE-XXX" id="XXX" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8104"/>
    <param name="transportendpoint" value="XXX:8204"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-XXX" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Review the **rfpipenem.xml** file.

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

Concept Review

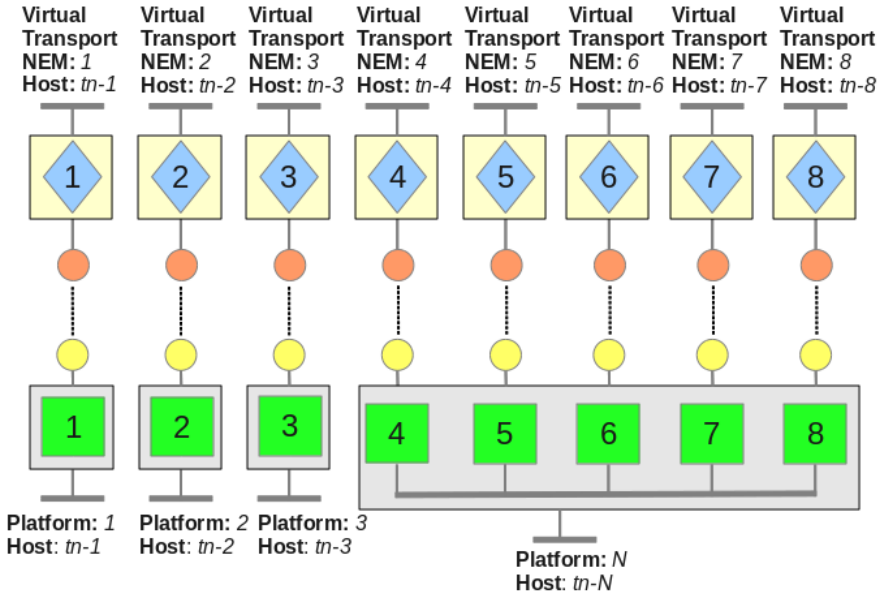
1. Why was the OTA Manager channel enabled in this workshop?
2. Why was the Event Service device set to *eth1*?
3. What are the implications of not specifying an OTA Manager device?
4. What are the EMANE components associated with the deployment type?
5. When is it possible to use **localhost** for the *platformendpoint* and *transportendpoint* parameters?

Workshop 03 - Hybrid Deployment

The goal of this workshop is to become familiar with configuring a hybrid EMANE deployment.

Deployment Description

This workshop contains a hybrid RF Pipe Deployment using Virtual Transports. There are 3 NEM Platform Servers each hosting a single NEM and 1 NEM Platform Server hosting 5 NEMs.



Problem Statement

There are 4 NEM Platform Server files `platform1.xml`, `platform2.xml`, `platform3.xml`, and `platformN.xml`. `platformN.xml` contains incomplete configuration. Modify the NEM Platform Server configuration to achieve the following:

- Use Platform Id N, where N is your Training Id.
- The OTA Manager mulitcast group is *224.1.2.8* and port *45702*.
- The OTA Manager device is *eth1*.
- The event service multicast group is *224.1.2.8* and port *45703*.
- The event service device is *eth1*.
- All NEMs should use the RF Pipe Model.
- Use NEM Ids 4 - 8.
- Platform 1 contains NEM 1 and resides on *tn-1*.
- Platform 2 contains NEM 2 and resides on *tn-2*.
- Platform 3 contains NEM 3 and resides on *tn-3*.
- Platform N contains NEM 4 - 8 and resides on *tn-N*, where N is your Training Id.

  - NEM 4's respective transport daemon resides on *tn-4*.
  - NEM 5's respective transport daemon resides on *tn-5*.
  - NEM 6's respective transport daemon resides on *tn-6*.
  - NEM 7's respective transport daemon resides on *tn-7*.
  - NEM 8's respective transport daemon resides on *tn-8*.
- Use test node addresses *tn-1* - *tn-8* and tn-N, or *10.99.0.1* - *10.99.0.8* and *10.99.0.N*, where N is your Training Id.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Platform Attributes

Name:	**name**
Description:	Descriptive name of the platform.
Type:	string

Name:	**id**
Description:	Platform identifier. **Must be unique for all platforms in the deployment.**
Type:	UINT16

Platform Parameters

Name:	**otamanagergroup**
Description:	The OTA Manager mulitcast endpoint that is used to communicate between all platforms in a test deployment. GROUP:PORT format. **Required only for multi-platform EMANE deployments.**
Note:	See *otamanagerdevice*.
Type:	string
XML Format:	<param name="otamanagergroup" value="224.1.2.8:45702"/>

Name:	**otamanagerdevice**
Description:	The network device to associate with OTA Manager mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *otamanagergroup*.
Type:	string
XML Format:	<param name="otamanagerdevice" value="eth1"/>

Name:	**otamanagerchannelenable**
Description:	The OTA Manager Channel Enable parameter enables or disables the OTA Manager channel. When disabled there is no platform to platform communication. Only locally managed NEMs will be able to communicate with each other.
Type:	bool
XML Format:	<param name="otamanagerchannelenable" value="on"/>

Name:	**eventservicegroup**
Description:	The Event Service multicast endpoint that is used to transmit events to and from the platform. GROUP:PORT format.
Note:	See *eventservicedevice*.
Type:	string
XML Format:	<param name="eventservicegroup" value="224.1.2.8:45703"/>

Name:	**eventservicedevice**
Description:	The network device to associate with Event Service mulitcast endpoint. If missing the kernel routing table is used to route multicast joins and packet transmissions.
Note:	See *eventservicegroup*.
Type:	string
XML Format:	<param name="eventservicedevice" value="eth1"/>

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

XML Listing

Review the **platform1.xml** file.

```
<?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="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="otamanagerdevice" value="eth1"/>
```

```
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth1"/>

  <nem name="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="tn-1:8101"/>
    <param name="transportendpoint" value="tn-1:8201"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-1" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Review the **platform2.xml** file.

```
<?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="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="otamanagerdevice" value="eth1"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth1"/>

  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="tn-2:8102"/>
    <param name="transportendpoint" value="tn-2:8202"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-2" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>
</platform>
```

Review the **platform3.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platform.dtd">
<platform name="Platform 3" id="3">
  <param name="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="otamanagerdevice" value="eth1"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth1"/>

  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="tn-3:8103"/>
    <param name="transportendpoint" value="tn-3:8203"/>
    <transport definition="transvirtual.xml">
      <param name="address" value="radio-3" />
      <param name="mask" value="255.255.255.0" />
    </transport>
  </nem>

</platform>
```

Modify the **platformN.xml** file to achieve the workshop goals.

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

<!--
   * Set Platform name and id
   * Set otamanagerchannelenable value
   * Set eventservicegroup value
   * Set eventservicedevice value
  -->
<platform name="Platform XXX" id="XXX">
  <param name="otamanagergroup" value="XXX"/>
  <param name="otamanagerdevice" value="XXX"/>
  <param name="eventservicegroup" value="XXX"/>
  <param name="eventservicedevice" value="XXX"/>
```

```
<!--
   * Set NEM 4 definition
   * Set platform endpoint address
   * Set transport endpoint address
  -->
<nem name="NODE-004" id="4" definition="rfpipenem.xml">
  <param name="platformendpoint" value="XXX:8104"/>
  <param name="transportendpoint" value="XXX:8204"/>
  <transport definition="transvirtual.xml">
    <param name="address" value="radio-4" />
    <param name="mask" value="255.255.255.0" />
  </transport>
</nem>

<!--
   * Set NEM 5 definition
   * Set platform endpoint address
   * Set transport endpoint address
  -->
<nem name="NODE-005" id="5" definition="rfpipenem.xml">
  <param name="platformendpoint" value="XXX:8105"/>
  <param name="transportendpoint" value="XXX:8205"/>
  <transport definition="transvirtual.xml">
    <param name="address" value="radio-5" />
    <param name="mask" value="255.255.255.0" />
  </transport>
</nem>

<!--
   * Set NEM 6 definition
   * Set platform endpoint address
   * Set transport endpoint address
  -->
<nem name="NODE-006" id="6" definition="rfpipenem.xml">
  <param name="platformendpoint" value="XXX:8106"/>
  <param name="transportendpoint" value="XXX:8206"/>
  <transport definition="transvirtual.xml">
    <param name="address" value="radio-6" />
    <param name="mask" value="255.255.255.0" />
  </transport>
</nem>

<!--
   * Set NEM 7 definition
   * Set platform endpoint address
   * Set transport endpoint address
  -->
<nem name="NODE-007" id="7" definition="rfpipenem.xml">
  <param name="platformendpoint" value="XXX:8107"/>
  <param name="transportendpoint" value="XXX:8207"/>
  <transport definition="transvirtual.xml">
    <param name="address" value="radio-7" />
    <param name="mask" value="255.255.255.0" />
  </transport>
</nem>


<!--
   * Set NEM 8 definition
   * Set platform endpoint address
   * Set transport endpoint address
  -->
<nem name="NODE-008" id="8" definition="rfpipenem.xml">
  <param name="platformendpoint" value="XXX:8108"/>
  <param name="transportendpoint" value="XXX:8208"/>
  <transport definition="transvirtual.xml">
    <param name="address" value="radio-8" />
    <param name="mask" value="255.255.255.0" />
  </transport>
</nem>
```

```
</platform>
```

Review the **rfpipenem.xml** file.

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

Concept Review

1. What is the `deployment.xml` file?
2. What is the communication path for messages sent from tn-1 to tn-8 via the OTA?
3. What is the communication path for messages sent from tn-5 to tn-8 via the OTA?

Part III. Transports

Contents

Transports 36
Virtual Transport 37
Raw Transport 38
Interactive Transport 39
Transport Interoperability 40
Workshop 04 - Virtual Transport 41
- Deployment Description 41
- Problem Statement 41
- NEM Attributes 41
- NEM Parameters 42
- Virtual Transport Parameters 42
- XML Listing 43
- Concept Review 45

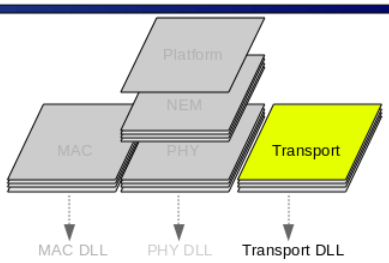
Workshop 05 - Raw Transport 46
- Deployment Description 46
- Problem Statement 46
- NEM Attributes 46
- NEM Parameters 47
- Raw Transport Parameters 47
- XML Listing 47
- Concept Review 49

Workshop 06 - Interactive Transport 50
- Deployment Description 50
- Problem Statement 50
- NEM Attributes 51
- NEM Parameters 51
- Interactive Transport Parameters 51
- Interactive Commands 51
- XML Listing 52
- Concept Review 53

Transports

Transport XML Explained

- library attribute specifies Transport DLL
  - libtransvirtual.so (Linux and OS X)
  - libtransvirtual.dll (win32)
- Specify configuration parameter defaults
  - Values can be overridden by those specified in the NEM, platform, and transport daemon XML files



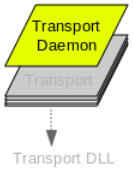

```
<transport name="Virtual Transport" library="transvirtual">
  <param name="bitrate" value="0"/>
  <param name="devicepath" value="/dev/net/tun"/>
  <param name="device" value="emane0"/>
  <param name="address" value="172.30.1.1"/>
  <param name="mask" value="255.255.0.0"/>
</transport>
```

0.6.2 30

Transport Daemon XML Sample

- Automatically generated from the platform XML using `emanegentransportxml` utility
- Specify the transport instances created by the transport daemon
  - Transport types
  - Associate NEMs
    - Ids
    - Endpoint connection information
  - Specify transport configuration parameter values

Transport Daemon

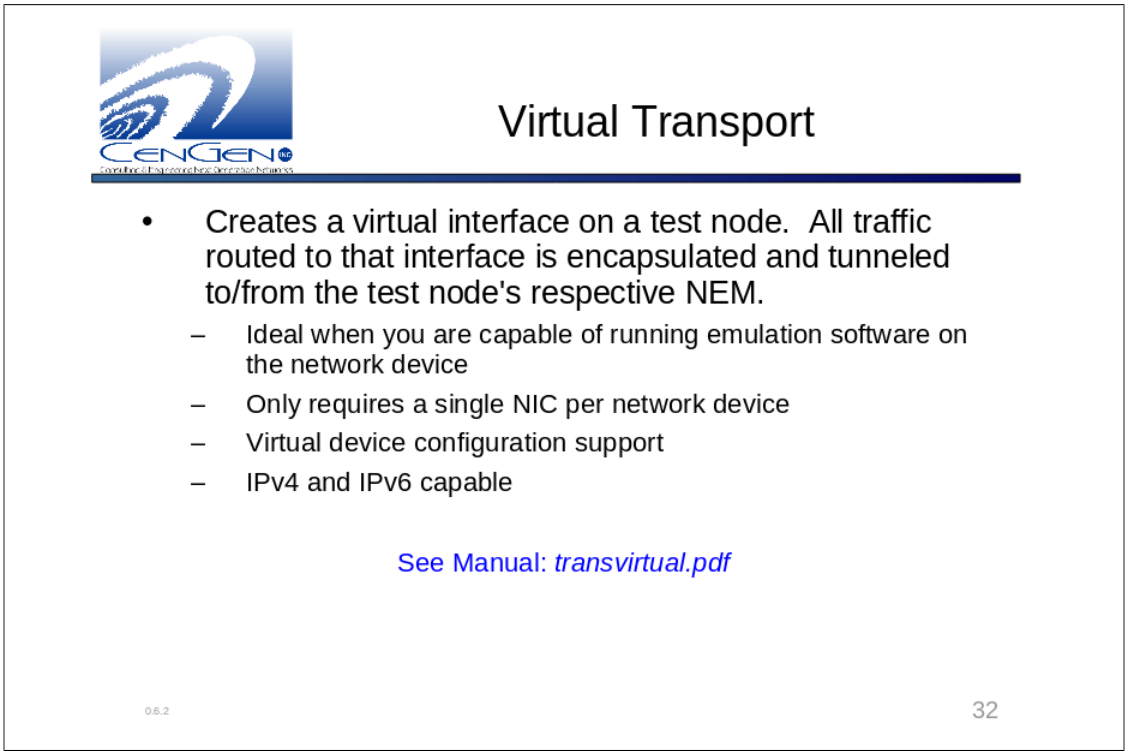



```
<transportdaemon>
  <instance nemid="1">
    <param name="transportendpoint" value="localhost:8171"/>
    <param name="platformendpoint" value="localhost:8181"/>
    <transport definition="transraw.xml">
      <param name="device" value="eth1"/>
    </transport>
  </instance>
  <instance nemid="2">
    <param name="transportendpoint" value="localhost:8172"/>
    <param name="platformendpoint" value="localhost:8182"/>
    <transport definition="transraw.xml">
      <param name="device" value="eth2"/>
    </transport>
  </instance>
  </transportdaemon>
```

0.6.2 31

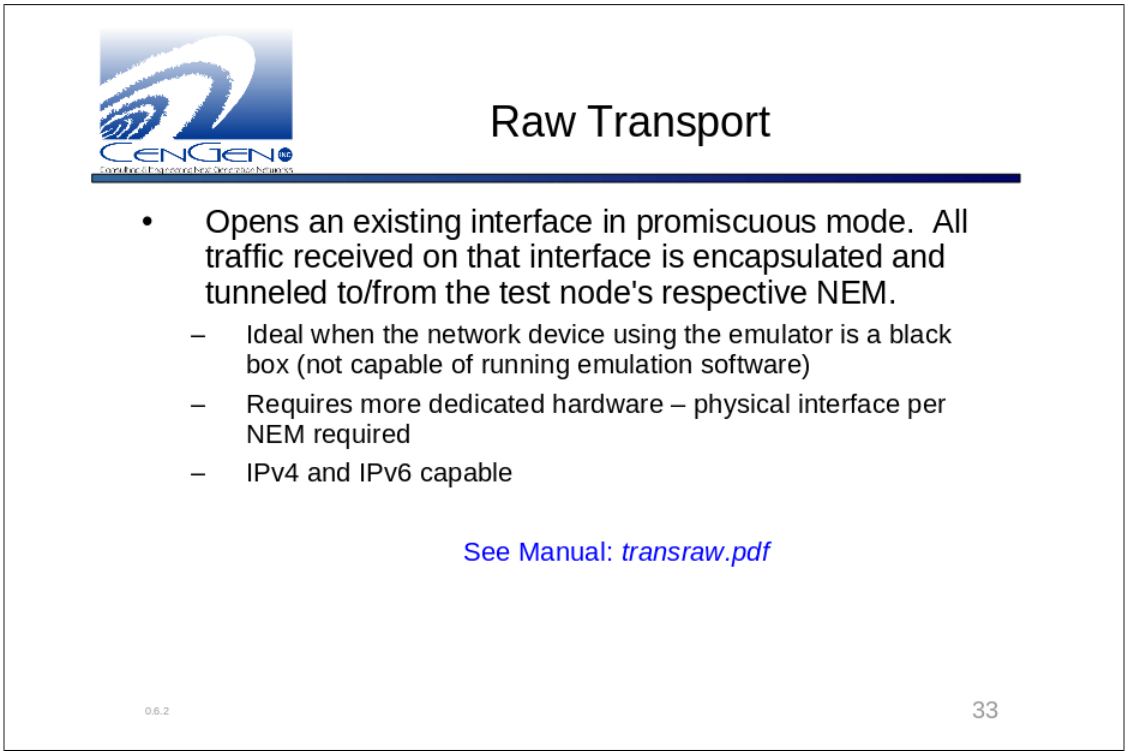
Virtual Transport

The virtual transport creates a virtual interface for use as the packet entry/exit point for a NEM network stack. IP packets routed to the virtual device are encapsulated and transmitted to their respective NEM. Packets received by the NEM's OTA Adapter are processed by the PHY and MAC layer and transmitted to the NEM's respective virtual transport for injection back into the kernel IP stack.



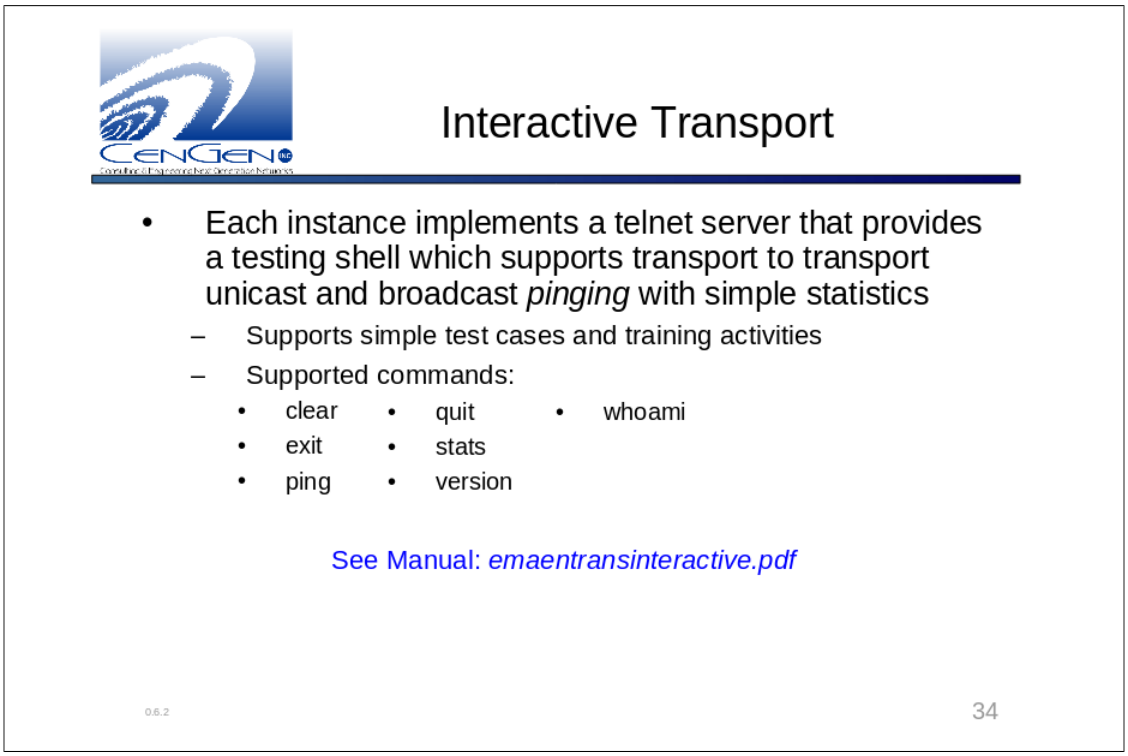
Raw Transport

The raw transport uses a specific network interface as the packet entry/exit point for a NEM network stack. IP packets read from the interface are encapsulated and transmitted to their respective NEM. Packets received by the NEM's OTA Adapter, are processed by the PHY and MAC layer and transmitted to the NEM's respective raw transport for transmission out the specified network interface.



Interactive Transport

The interactive transport implements a telnet server that provides a testing shell which supports transport to transport unicast and broadcast pinging with simple statistics.



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 tunnel opaque data to and from their respective NEM stacks. The format of the data is implementation dependent. It is up to individual transport developers to take the appropriate steps necessary to allow for compatibility with transports of similar types.

Both the Virtual Transport and Raw Transport tunnel 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 tunneled 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.

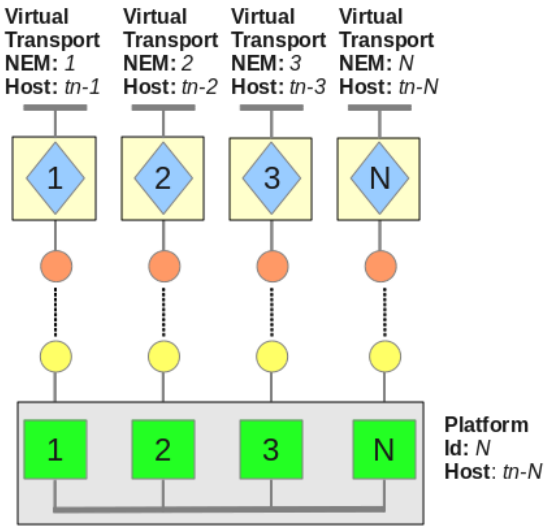
Parameter	Virtual Transport	Raw Transport
arpcacheenable	off	off
broadcastmode	on	N/A

Workshop 04 - Virtual Transport

The goal of this workshop is to become familiar with configuring a Virtual Transport.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Virtual Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 4 NEMs with incomplete configuration. Modify the XML configuration to achieve the following:

- Use the NEM Platform Server *tn-N* or *10.99.0.N*, where N is your Training Id.
- Use Platform Id N, where N is your Training Id.
- NEM 1 virtual device address use *radio-1* or *10.100.0.1* with a 24 bit mask.
- NEM 2 virtual device address use *radio-2* or *10.100.0.2* with a 24 bit mask.
- NEM 3 virtual device address use *radio-3* or *10.100.0.3* with a 24 bit mask.
- NEM N virtual device address use *radio-N* or *10.100.0.N* with a 24 bit mask, where N in your Training Id.
- NEM N virtual device name is *manet0*.
- All NEMs should be limited to *1Mpbs*.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

Virtual Transport Parameters

Name:	**bitrate**
Description:	Transport bitrate in Kbps. This is the total allowable throughput for the transport. A value of 0 disables the bitrate feature. The maximum value is only limited to the storage size of the type uint64 18446744073709551615 / 1024, or 18014398509481983.
Type:	Double
Default:	8096
XML Format:	<param name="bitrate" value="500"/>

Name:	**arpcacheenable**
Description:	Enable ARP request/reply monitoring to map ethernet address to NEM.
Type:	Boolean
Default:	on
XML Format:	<param name="arpcacheenable" value="on"/>

Name:	**devicepath**
Description:	Path to the tap device.
Type:	string
Default (linux):	/dev/net/tun
Default (OSX):	/dev/tap0
Default (Win32):	SYSTEM\\CurrentControlSet\\Control\\Network\\{4D36E972-E325-11CE-BFC1-08002BE10318}
XML Format:	<param name="devicepath" value="/dev/net/tun"/>

Name:	**device**
Description:	Virtual device name. Note: on OS X this *must* be tap0.
Type:	string
Default:	emane0
XML Format:	<param name="device" value="emane1"/>

Name:	**address**
Description:	Virtual device address. Supports IPv4 and IPv6.
Type:	string
Default:	none
XML Format:	<param name="address" value="172.30.1.1"/>

Name:	**mask**
Description:	Virtual device network mask. Supports IPv4 and IPv6.
Type:	string
Default:	none
XML Format:	<param name="mask" value="255.255.0.0"/>

Name:	**arpmode**
Description:	Enable arp on the virtual device.
Type:	boolean
Default:	on
XML Format:	<param name="arpmode" value="on"/>

Name:	**broadcastmode**
Description:	Broadcast all packets to all NEM's.
Type:	boolean
Default:	off
XML Format:	<param name="broadcastmode" value="on"/>

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

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

<!--
   * Set Platform name and id
  -->
<platform name="Platform XXX" id="XXX">
  <param name="otamanagerchannelenable" value="off"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="lo"/>

  <!--
     * Set NEM 1 platform endpoint address
    -->
  <nem name="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8101"/>
    <param name="transportendpoint" value="tn-1:8201"/>
    <transport definition="transvirtual.xml">
```

```
      <!--
         * Set transport parameter address
         * Set transport parameter mask
        -->
      <param name="XXX" value="XXX"/>
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>

  <!--
     * Set NEM 2 platform endpoint address
    -->
  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8102"/>
    <param name="transportendpoint" value="tn-2:8202"/>
    <transport definition="transvirtual.xml">
      <!--
         * Set transport parameter address
         * Set transport parameter mask
        -->
      <param name="XXX" value="XXX"/>
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>


  <!--
     * Set NEM 3 platform endpoint address
    -->
  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8103"/>
    <param name="transportendpoint" value="tn-3:8203"/>
    <transport definition="transvirtual.xml">
      <!--
         * Set transport parameter address
         * Set transport parameter mask
        -->
      <param name="XXX" value="XXX"/>
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>


  <!--
     * Set NEM N name and id
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-XXX" id="XXX" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8104"/>
    <param name="transportendpoint" value="XXX:8204"/>
    <transport definition="transvirtual.xml">
      <!--
         * Set transport parameter address
         * Set transport parameter mask
         * Set transport parameter device
        -->
      <param name="XXX" value="XXX"/>
      <param name="XXX" value="XXX"/>
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>
</platform>
```

Modify the **transvirtual.xml** file to achieve the workshop goals.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE transport SYSTEM "file:///usr/share/emane/dtd/transport.dtd">
<transport name="Virtual Transport" library="transvirtual">
   <!--
      * Set transport parameter bitrate
```

```
     -->
   <param name="XXX" value="XXX"/>
</transport>
```

Concept Review

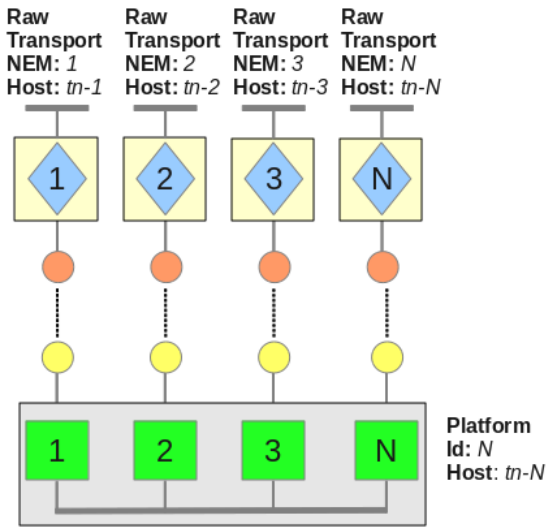
1. What are alternative ways the transport *bitrate* configuration for this workshop could have been specified?
2. What happens if you do not specify the address and mask for the Virtual Transport? When might this be desirable?
3. Are IPv6 virtual addresses supported?

Workshop 05 - Raw Transport

The goal of this workshop is to become familiar with configuring a Raw Transport.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Raw Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 4 NEMs with incomplete configuration. Modify the NEM Platform Server configuration to achieve the following:

- Use NEM Platform Server *tn-N* or *10.99.0.N*, where N is your Training Id.
- Use Platform Id N, where N is your Training Id.
- All NEMs should use the Raw Transport.
- For NEM N use test node address *tn-N* or *10.99.0.N*, where N is your Training Id.
- NEM 1 should use *eth2* as its raw device.
- NEM 2 should use *eth2* as its raw device.
- NEM 3 should use *eth2* as its raw device.
- NEM N should use *eth2* as its raw device.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

Raw Transport Parameters

Name:	**bitrate**
Description:	Transport bitrate in Kbps. This is the total allowable throughput for the transport. A value of 0 disables the bitrate feature. The maximum value is only limited to the storage size of the type uint64 18446744073709551615 / 1024, or 18014398509481983.
Type:	Double
Default:	8096
XML Format:	<param name="bitrate" value="500"/>

Name:	**arpcacheenable**
Description:	Enable ARP request/reply monitoring to map ethernet address to NEM.
Type:	Boolean
Default:	on
XML Format:	<param name="arpcacheenable" value="on"/>

Name:	**device**
Description:	Device to use as the raw packet entry point
Type:	string
Default:	none
XML Format:	<param name="device" value="eth1"/>

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

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

<!--
   * Set Platform name and id
  -->
```

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

  <!--
     * Set NEM 1 platform endpoint address
    -->
  <nem name="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8101"/>
    <param name="transportendpoint" value="tn-1:8201"/>
    <transport definition="XXX">
      <!--
         * Set transport definition
         * Set transport parameter device
        -->
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>

  <!--
     * Set NEM 2 platform endpoint address
    -->
  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8102"/>
    <param name="transportendpoint" value="tn-2:8202"/>
    <transport definition="XXX">
      <!--
         * Set transport definition
         * Set transport parameter device
        -->
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>

  <!--
     * Set NEM 3 platform endpoint address
    -->
  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8103"/>
    <param name="transportendpoint" value="tn-3:8203"/>
    <transport definition="XXX">
      <!--
         * Set transport definition
         * Set transport parameter device
        -->
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>

  <!--
     * Set NEM N name and id
     * Set platform endpoint address
     * Set transport endpoint address
    -->
  <nem name="NODE-XXX" id="XXX" definition="rfpipenem.xml">
    <param name="platformendpoint" value="XXX:8104"/>
    <param name="transportendpoint" value="XXX:8204"/>
    <transport definition="XXX">
      <!--
         * Set transport definition
         * Set transport parameter device
        -->
      <param name="XXX" value="XXX"/>
    </transport>
  </nem>
</platform>
```

Review the **transraw.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE transport SYSTEM "file:///usr/share/emane/dtd/transport.dtd">
<transport name="Raw transport" library="transraw"/>
```

Concept Review

1. What routes need to be added to client machines that connect to an interface dedicated for use by a Raw Transport?
2. Should applications that route traffic over an interface in use by a Raw Transport be used?
3. What does the *arpcacheenable* parameter do?
4. How does the Raw Transport address packets when *arpcacheenable* is off?

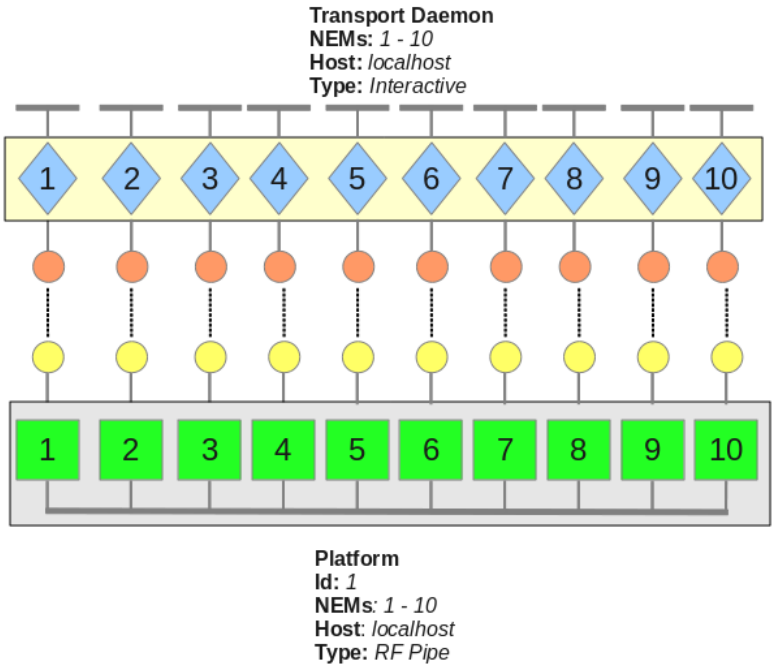
Workshop 06 - Interactive Transport

The goal of this workshop is to become familiar with configuring and interacting with the Interactive Transport.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 10 NEMs. The Interactive Transport implements a telnet server that provides a testing shell which supports transport to transport unicast and broadcast *pinging* with simple statistics. This transport is designed for use with simple model test cases and training activities.

You can log in to each of the 10 NEM transports created by this deployment using any telnet client. Each transport server is listening on a specific port based on the *portbase* transport configuration item and the transport NEM id. Each transport server (instance of transinteractive) only allows one active telnet connection at a time.



Problem Statement

The `platform.xml` file in this workshop contains 10 NEMs with incomplete transport configuration. Modify the NEM Platform Server configuration to achieve the following:

- Set all NEM transport group entries so all are managed by a single instance of the Transport Daemon.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Log in to a few of the interactive sessions and send unicast and broadcast pings between transports.

In this workshop *portbase* is 46000. To connect to an Interactive Transport associated with an NEM you need to connect to the telnet server located at the *portbase* + NEM Id.

For example, to connect to the Interactive Transport for NEM 1:

```
> telnet localhost 46001
```

NEM Attributes

Name:	**name**
Description:	Descriptive name of the NEM.
Type:	string

Name:	**id**
Description:	NEM identifier. **Must be unique for all NEMs in the deployment.**
Type:	UINT16

NEM Parameters

Name:	**platformendpoint**
Description:	The endpoint that the NEM should bind to in order to receive messages from it's respective transport.
Type:	string
XML Format:	<param name="platformendpoint" value="platformserver:8181"/>

Name:	**transportendpoint**
Description:	The transport endpoint that the NEM should use when transmitting messages to it's respective transport.
Type:	string
XML Format:	<param name="transportendpoint" value="tn-1:8181"/>

Interactive Transport Parameters

Name:	**portbase**
Description:	The telnet server will listen on port portbase + NEM id on all interfaces
Type:	UINT16
Default:	46000
XML Format:	<param name="portbase" value="46000"/>

Interactive Commands

The interactive transport command shell supports the following commands:

clear	Clear Screen
exit	Exit telnet session. Same as *quit*.
ping	Transport to Transport ping.
quit	Quit telnet session. Same as *exit*.
stats	Show or clear statistic counters.
version	Print version.
whoami	Print NEM id.

ping Usage

The ping commands sends an interactive transport PING Request to another interactive transport to elicit an interactive transport PING Response.

ping [-i interval] [-c count] [-s size] NEMID|'all'

interval	Wait interval seconds between sending each packet. The default is to wait for one second between each packet.
count	Stop after sending count ping requests packets.
size	Specifies the number of data bytes to be sent. The default is 4, which translates into 64 data bytes when combined with the 60 bytes of interactive ping header.

stats Usage

The stats command displays or clears stats related to inbound and outbound interactive transport PING requests and responses.

stats [clear]

XML Listing

Review the **platform.xml** file.

```
<?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="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-004" id="4" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>
```

```
  <nem name="NODE-005" id="5" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8105"/>
    <param name="transportendpoint" value="localhost:8205"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-006" id="6" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8106"/>
    <param name="transportendpoint" value="localhost:8206"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-007" id="7" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8107"/>
    <param name="transportendpoint" value="localhost:8207"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-008" id="8" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8108"/>
    <param name="transportendpoint" value="localhost:8208"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-009" id="9" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8109"/>
    <param name="transportendpoint" value="localhost:8209"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

  <nem name="NODE-010" id="10" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8110"/>
    <param name="transportendpoint" value="localhost:8210"/>
    <!--
       * Set transport group
      -->
    <transport definition="transinteractive.xml" />
  </nem>

</platform>
```

Review the **transinteractive.xml** file.

```
<!DOCTYPE transport SYSTEM "file:///usr/share/emane/dtd/transport.dtd">
<transport name="Interactive transport" library="transinteractive">
  <param name="portbase" value="46000"/>
</transport>
```

Concept Review

1. What data message format (or type of data message) does any NEM expect to receive from an attached Transport?

2. Can an Interactive Transport interoperate with a Raw or Virtual Transport?

Part IV. Network Emulation Models

Contents

Models 55
RF Pipe Model 56
- MAC Details 56
- PHY Details 56

IEEE 802.11 abg Model 58
- MAC Details 59
- PHY Details 59

Comm Effect Model 61
- Shim Details 61

Workshop 07 - RF Pipe Model 62
- Deployment Description 62
- Problem Statement 62
- RF Pipe MAC Parameters 63
- RF Pipe PHY Parameters 63
- Interactive Commands 66
- XML Listing 66
- Concept Review 67

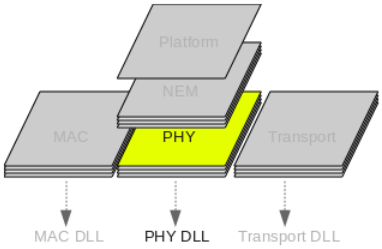
Workshop 08 - IEEE 802.11 abg Model Model 69
- Deployment Description 69
- Problem Statement 69
- IEEE 802.11 abg MAC Parameters 70
- IEEE 802.11 abg PHY Parameters 75
- Interactive Commands 77
- XML Listing 78
- Observations Explained 79
- Concept Review 80

Workshop 09 - Comm Effect Model Model 81
- Deployment Description 81
- Problem Statement 81
- Comm Effect Shim Parameters 82
- Interactive Commands 83
- XML Listing 84
- Concept Review 85

Models

PHY XML Explained

- library attribute specifies PHY DLL
  - librfpipephylayer.so (Linux and OS X)
  - librfpipephylayer.dll (win32)
- Specify configuration parameter defaults
  - Values can be overridden by those specified in the NEM and platform XML files

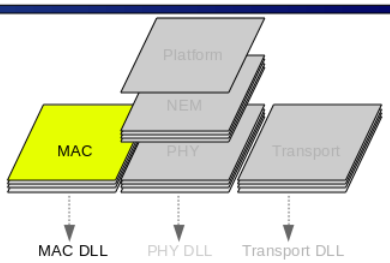



```
<phy name="RF-PIPE PHY" library="rfpipephylayer">
  <param name="txpower" value="0"/>
  <param name="antennagain" value="0"/>
  <param name="bitrate" value="0"/>
  <param name="jitter" value="0"/>
  <param name="frequency" value="2347"/>
  <param name="bandwidth" value="20"/>
  <param name="propagationdelay" value="0"/>
  <param name="pathlosssource" value="0"/>
  <param name="pathlossalgorithm" value="0"/>
  <param name="defaultconnectivity" value="on"/>
  <param name="rxpowernolossthreshold" value="90"/>
  <param name="rxpowerfulllossthreshold" value="110"/>
</phy>
```

0.6.2 37

MAC XML Explained

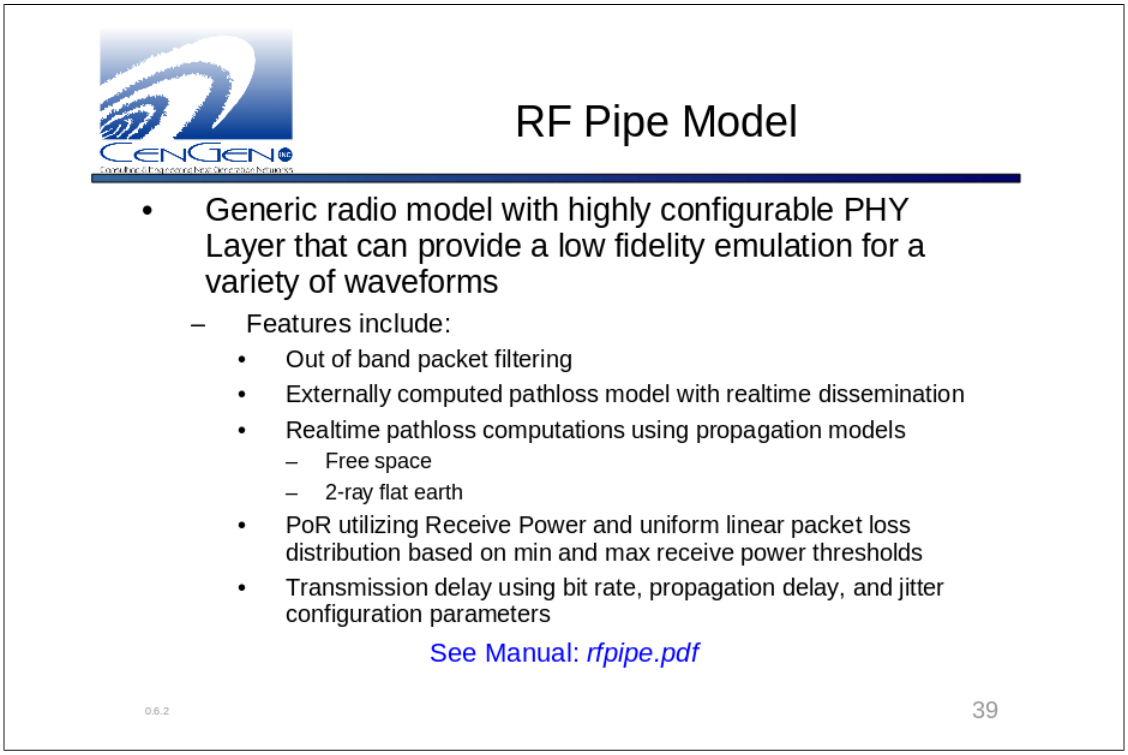
- library attribute specifies MAC DLL
  - libieee8011abgmaclayer.so (Linux and OS X)
  - libeee8011abgmaclayer.dll (win32)
- Specify configuration parameter defaults
  - Values can be overridden by those specified in the NEM and platform XML files




```
<mac name="IEEE802.11" library="ieee80211abgmaclayer">
  <param name="enablepromiscuousmode" value="off"/>
  <param name="distance" value="1000"/>
  <param name="rtsthreshold" value="0"/>
  <param name="mode" value="0"/>
  <param name="unicastrate" value="0"/>
  <param name="multicastrate" value="0"/>
  <param name="wmmenable" value="off"/>
  <param name="queuesize" value="0:255"/>
  <param name="retrylimit" value="0:3"/>
  <param name="cwmin" value="0:31"/>
</mac>
```

0.6.2 38

RF Pipe Model



MAC Details

The MAC is a simple bypass module with the ability to enable or disable promiscuous mode. When promiscuous mode is enabled, all over the air packets received from the PHY are sent up to the network layer. When promiscuous mode is disabled, only multicast/broadcast and unicast packets destined for the local node are sent up to the network layer.

PHY Details

The capabilities within the PHY component of the RF Pipe model are as follows:

1. Provides an out of band packet filter capability where all over the air packets received on a different frequency or with a different model ID are dropped.

2. Provides the ability to select from one of 2 sources for path loss. The 2 sources are as follows:

   a. External path loss events where the path loss is computed off line and provided to the PHY via the path loss event generator.

   b. Real time path loss computation based on node positions where the node positions are provided to the PHY via a position event generator. The path loss algorithm will utilize the transmitter and receiver positions and compute the path loss using one of two (free space or 2-ray flat earth) propagation models selected via configuration. The selected model is used for all received packets.

3. Determines probability of reception (PoR) by utilizing Received Signal Power and uniform linear packet loss distribution based on min and max receive power thresholds.

   a. The Rx Power is computed based on path loss between the transmitter and receiver, transmit power, Tx antenna gain and Rx antenna gain. The path loss is determined as defined above.

Transmit power and antenna gain are both fixed at initialization but can be configured differently for each node within the network to emulate platforms such as vehicles, dismounts, fixed stations, etc. The transmit power and Tx antenna gain are included as part of the PHY header in all over the air transmissions providing the ability for each receiver to compute the Rx Power. The receiver antenna gain is defined via configuration.

Rx Power (dBm) = Tx Power (dBm) + Tx Antenna Gain (dBi) + Rx Antenna Gain (dBi) - Pathloss (dB)

b. The min and max thresholds are configurable and PoR is determined as follows:

   i. If Rx Power >= -Rx Power No Loss Threshold, PoR = 1

   ii. If Rx Power <= -Rx Power Full Loss Threshold, PoR = 0

   iii. If -maxThreshold < Rx Power < - minThreshold, PoR is based on a linear curve between the min and max thresholds.

4. Provides the ability to emulate transmission delay. The PHY will compute a transmission delay for each packet prior to transmission and will wait the computed amount before transmitting the packet over the air. The transmission delay is computed as follows:

   txdelay = packetsize/bitrate + propagationdelay + jitter

   Bitrate, propagationdelay and jitter are all PHY configuration parameters.

The MAC is a simple bypass module with the ability to enable or disable promiscuous mode. When promiscuous mode is enabled, all over the air packets received from the PHY are sent up to the network layer. When promiscuous mode is disabled, only multicast/broadcast and unicast packets destined for the local node are sent up to the network layer.

IEEE 802.11 abg Model

The objective of the 802.11abg model is to emulate 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 following sections define the capabilities and configuration options of the emulated 802.11abg MAC and PHY components.

IEEE 802.11 abg Model

- Emulates 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)
  - MAC Layer features include:
    - 802.11b (DSS rates: 1, 2, 5.5 and 11 Mbps)
    - 802.11a/g (OFDM rates: 6, 9, 12, 18, 24, 36, 48 and 54 Mbps)
    - 802.11b/g (DSS and OFDM rates)
    - DCF channel access
    - Unicast and broadcast (RTS/CTS/ACK capable)

0.6.2 40

IEEE 802.11 abg Model

- PHY Layer features include:
  - Out of band packet filtering
  - Externally computed pathloss model with realtime dissemination
  - Half duplex operation
  - PoR utilizing Bit Error Rate (BER) curves based on SINR, packet size and data rate

See Manual: *ieee80211abg.pdf*

0.6.2 41

MAC Details

The IEEE 802.11 abg MAC Layer provides the following capabilities:

1. Supports the following 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)

2. Supports only the DCF channel access function. PCF and beacon transmissions are not supported within the MAC.

3. Supports both unicast and broadcast transmissions. Unicast transmissions include the ability to emulate control message (RTS/CTS/ACK) behavior as well as retries without actually transmitting the control messages or the retransmission of the data message. The emulation of unicast does not replicate exponential growth of the contention window as a result of detected failures.

4. Supports Wi-Fi multimedia (WMM) capabilities. The initial implementation supports the ability to classify 4 different traffic classes (Background, Best Effort, Video and Voice) where the higher priority classes (voice and video) are serviced first.

   WMM supports up to 4 Access Categories based on IP DSCP. Differentiated Service Code Points are defined using the 6 most significant bits of the IP TOS field. The mapping of the DSCP to the 4 Access Categories is as follows:

   a. Best Effort (DSCP: 0-7 & 24-31)

   b. Background (DSCP: 8-23)

   c. Video (DSCP: 32-47)

   d. Voice (DSCP: 48-63)

PHY Details

The IEEE 802.11 abg PHY Layer provides the following capabilities:

1. Filters all out of band packets. Out of band packets are those that are received from nodes on a different mode, frequency or a completely different emulation model. This feature provides the ability to support heterogeneous networks.

2. Emulates half duplex operations where all packets received during transmit are dropped.

3. Determines probability of reception (PoR) by utilizing Bit Error Rate (BER) curves based on receive power, noise/interference, and data rate.

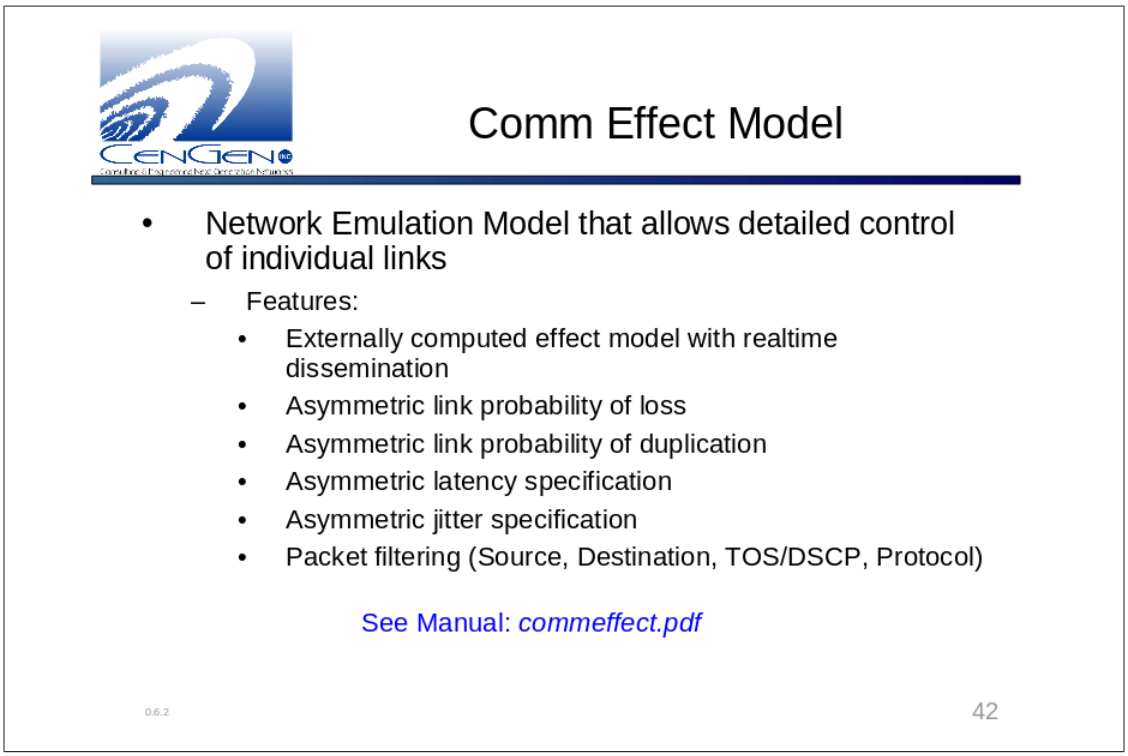
   a. The BER curves are a function of Signal to Interference Ratio (SINR) and the 802.11 data rate. The BER curves are generated off line for each data rate and are used at initialization to generate probability of reception tables (PoR) for various packet sizes. The PoR tables are then used at run time to determine if a packet would be received or not based on the actual size of the packet along with the SINR. Linear interpolation is performed when packet size does not align with the generated PoR tables.

   b. The receive power is computed based on path loss between transmitter and receiver, transmit power, Tx antenna gain and Rx antenna gain. The path loss is computed off line prior to the test and is provided via an external event during run time. Transmit power and antenna gain are

both fixed at initialization but can be configured differently for each node within the network to emulate platforms such as vehicles, dismounts, fixed stations, etc. The transmit power and antenna gain are included as part of the PHY header in all over the air transmissions providing the ability for each receiver to compute the receive power.

c. The noise/interference is currently based only on in-band packets. All over the air transmissions contain received packet duration information which is used to determine the time for the Rx Busy interval within the PHY. Once the PHY enters the Rx Busy state, all other packets received during the Rx Busy interval are considered as noise and will impact the PoR of the original received packet by lowering SINR accordingly.

Comm Effect Model



Shim Details

The CommEffect model provides the ability to control network impairments commonly provided via traditional COTS network emulators on a per link basis within the Extendable Mobile Ad-Hoc Network Emulation (EMANE) architecture. The CommEffect model is a **shim** only implementation (no MAC or PHY layer) utilizing *unstructured* EMANE NEMs

The CommEffects model provides the ability to define the following network impairments:

1. **Loss**: The percentage of packets that will be dropped utilizing the uniform loss distribution model.

2. **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.

3. **Duplicates**: The percentage of packets that will be duplicated at the receiver.

4. **Unicastbitrate**: The bitrate for packets destined for the NEM or handled in promiscous mode.

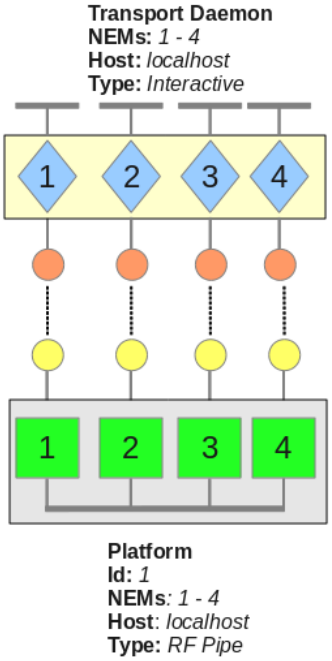
5. **Broadcastbitrate**: The bitrate for packets destined for the NEM broadcast address.

Workshop 07 - RF Pipe Model

The goal of this workshop is to become familiar with configuring the RF Pipe Model.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 4 NEMs with incomplete configuration. Modify the XML configuration to achieve the following:

- Add a 100 msec *jitter* to NEM 1.
- Add a 500 msec *jitter* to NEM 2.
- Add a 30 msec *jitter* to NEM 3.
- Change the *frequency* for all NEMs to 3000 MHz.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify the settings using the Interactive Transport to unicast and broadcast *ping* between NEMs.

RF Pipe MAC Parameters

Name:	**enablepromiscuousmode**
Description:	Determines if all packets received over the air will be sent up the stack
	on — Send all packets up
	off — Send only broadcast and locally addressed unicast up.
Type:	Boolean
Default:	off
XML Format:	<param name="enablepromiscuousmode" value="off"/>

Name:	**transmissioncontrolmap**
Description:	Sends the data rate, frequency, and power level to the PHY for all transmissions to a specified node. When a packet is transmitted to the destination NEM, the PHY will include the specified data rate, frequency, and transmission power in the OTA PHY header. A 0 in any value other than the destination will cause the XML configuration value for that respective RF Pipe PHY parameter to be used.
	Destination NEM — NEM Id of destination — UINT16
	Data Rate — Data Rate in Kbps — UINT16
	Frequency — Frequency in MHz — UINT16
	Tx Power — Power Level in dBm — UINT8
Type:	string
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"/>

RF Pipe PHY Parameters

Name:	**frequency**
Description:	Defines the frequency used in MHz for the emulated network. The frequency is included in the PHY header of all over the air messages and serves 2 functions:
	1. Filters out packets received from nodes that are on different frequencies.
	2. Is used to determine free space path loss when configured to use compute path loss based on node positions using the free space propagation algorithm. See configuration parameters *pathlosssource* and *pathlossalgorithm* below.
Type:	UINT16
Default:	2437
XML Format:	<param name="frequency" value="2437">

Name:	**bandwidth**
Description:	Defines the bandwidth in Khz. Bandwidth is included in the PHY header. The current version does not utilize this information.
Type:	UINT16
Default:	20
XML Format:	<param name="bandwidth" value="20">

Name:	**txpower**
Description:	Defines the transmit the power (dBm) and is used to compute the receive power in conjunction with path loss and antenna gains. The txpower is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="txpower" value="0">

Name:	**antennagain**
Description:	Defines the antenna gain (dBi) and is used to compute the receive power in conjunction with path loss and txpower. The antennagain is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="antennagain" value="0">

Name:	**bitrate**
Description:	Defines the bitrate/burstrate (kbps) of the waveform being emulated. It is used on the transmit side to compute transmission delay based on the packet size and bitrate.
Type:	UINT16
Default:	0
XML Format:	<param name="bitrate" value="0">

Name:	**propagationdelay**
Description:	Defines the propagation delay (usec) that is to be included in the transmission delay. The propagationdelay is added to the delay introduced by the bitrate defined above.
Type:	UINT32
Default:	0
XML Format:	<param name="propagationdelay" value="0">

Name:	**jitter**
Description:	Defines the jitter (usec) 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:	UINT32
Default:	0
XML Format:	<param name="jitter" value="0">

Name:	**pathlosssource**
Description:	Defines the method of use for determining path loss. The parameter accepts 2 options: 0 — Pathloss directly provided via an external event 1 — Pathloss computed real time based on external position events and selected *pathlossalgorithm*
Type:	UINT16
Default:	0
XML Format:	<param name="pathlosssource" value="0">

Name:	**pathlossalgorithm**
Description:	Defines the algorithm to be used to compute the path loss. This is valid only when *pathlosssource* is set to *1*. The algorithm selections are as follows: 0 — Free space path loss model 1 — 2 ray flat earth model
Type:	UINT16
Default:	0
XML Format:	<param name="pathlossalgorithm" value="0">

Name:	**rxpowernolossthreshold**
Description:	Defines the receive power threshold (dBm) for determining if packets will be received or dropped. This threshold is used in conjunction with the *rxpowerfulllossthreshold* to determine the probability of packet reception (PoR) based on received power as follows: Rx Power >= rxpowernolossthreshold ==> PoR = 1 rxpowerfulllossthreshold < Rx Power < rxpowernolossthreshold ==> linear packet loss Rx Power <= rxpowerfulllossthreshold ==> PoR = 0
Type:	UINT16
Default:	90
XML Format:	<param name="rxpowernolossthreshold" value="90">

Name:	**rxpowerfulllossthreshold**
Description:	See description of *rxpowernolossthreshold* above.
Type:	UINT16
Default:	110
XML Format:	<param name="rxpowerfulllossthreshold" value="110">

Name:	**defaultconnectivity**
Description:	Defines the default connectivity of all nodes at start up prior to receiving any mobility or path loss events. When set to *on* the reception of a single path loss or position event will disable the default connectivity setting. off — All nodes are assumed isolated on — All nodes are assumed to be fully informed
Type:	Boolean
Default:	on
XML Format:	<param name="defaultconnectivity" value="on">

Interactive Commands

The interactive transport command shell supports the following commands:

clear	Clear Screen
exit	Exit telnet session. Same as *quit*.
ping	Transport to Transport ping.
quit	Quit telnet session. Same as *exit*.
stats	Show or clear statistic counters.
version	Print version.
whoami	Print NEM id.

ping Usage

The ping commands sends an interactive transport PING Request to another interactive transport to elicit an interactive transport PING Response.

ping [-i interval] [-c count] [-s size] NEMID|'all'

interval	Wait interval seconds between sending each packet. The default is to wait for one second between each packet.
count	Stop after sending count ping requests packets.
size	Specifies the number of data bytes to be sent. The default is 4, which translates into 64 data bytes when combined with the 60 bytes of interactive ping header.

stats Usage

The stats command displays or clears stats related to inbound and outbound interactive transport PING requests and responses.

stats [clear]

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

```
<?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="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <phy definition="rfpipephy.xml">
      <!--
         Set PHY parameter jitter
        -->
      <param name="XXX" value="XXX"/>
    </phy>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <phy definition="rfpipephy.xml">
      <!--
         Set PHY parameter jitter
        -->
      <param name="XXX" value="XXX"/>
    </phy>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <phy definition="rfpipephy.xml">
      <!--
         Set PHY parameter jitter
        -->
      <param name="XXX" value="XXX"/>
    </phy>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-004" id="4" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>
</platform>
```

Modify the **rfpipephy.xml** file to achieve the workshop goals.

```
<!DOCTYPE phy SYSTEM "file:///usr/share/emane/dtd/phy.dtd">
<phy name="RF-PIPE PHY" library="rfpipephylayer">
  <param name="txpower" value="0"/>
  <param name="antennagain" value="0"/>
  <param name="bitrate" value="0"/>
  <param name="jitter" value="0"/>
  <!--
     Set parameter frequency
    -->
  <param name="frequency" value="XXX"/>
  <param name="bandwidth" value="20"/>
  <param name="propagationdelay" value="0"/>
  <param name="pathlosssource" value="1"/>
  <param name="pathlossalgorithm" value="0"/>
  <param name="defaultconnectivity" value="on"/>
  <param name="rxpowernolossthreshold" value="90"/>
  <param name="rxpowerfulllossthreshold" value="110"/>
</phy>
```

Concept Review

1. What does the *defaultconnectivity* parameter control?
2. What does the *rxpowernolossthreshold* parameter configure?

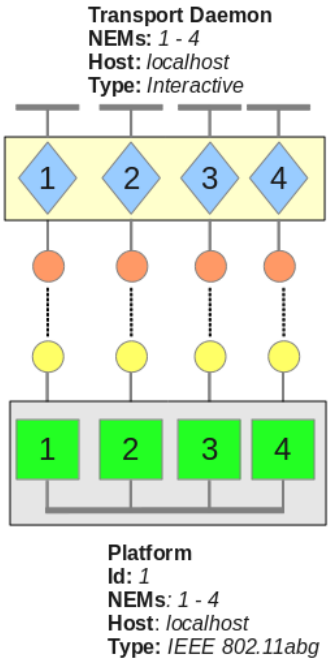
3. What does the *rxpowerfulllossthreshold* parameter configure?
4. Which events does the RF Pipe Model process?
5. What controls which event type is used as part of the receive power calculation?

Workshop 08 - IEEE 802.11 abg Model Model

The goal of this workshop is to become familiar with configuring the IEEE 802.11 abg Model.

Deployment Description

This workshop contains a centralized IEEE 802.11 abg Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 4 NEMs with incomplete configuration. Modify the XML configuration to achieve the following:

- Set NEM 1's unicast data rate to 1Mbps
- Set NEM 2's unicast data rate to 1Mbps
- Set NEM 3's unicast data rate to 11Mbps
- Set NEM 4's unicast data rate to 54Mbps

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

1. Verify the settings by using the Interactive Transport:

   a. From NEM 1 interactive transport session send 100 128 byte packets at a rate of 10 packets per second to NEM 2

```
transport-001 %% ping -i .1 -c 100 -s 128 2
```

Use the *stats* command to view the completion statistics. Note the results.

b. From NEM 1 interactive transport session send 100 128 byte packets at a rate of 10 packets per second to NEM 3

```
transport-001 %% ping -i .1 -c 100 -s 128 3
```

Use the *stats* command to view the completion statistics. Note the results.

c. From NEM 1 interactive transport session send 100 128 byte packets at a rate of 10 packets per second to NEM 4

```
transport-001 %% ping -i .1 -c 100 -s 128 4
```

Use the *stats* command to view the completion statistics. Note the results.

2. Stop the workskop components and set the antenna gain for NEM 1 to 13dBi.

Rerun the interactive commands and compare the results with what you observed previously.

IEEE 802.11 abg MAC Parameters

Name:	**mode**	
Description:	Defines the 802.11 mode. The mode is included in the PHY header of all over the air messages and is used to filter out packets received from nodes that are on different modes. Should be noted that mode setting does not care about frequency and is used only to identify the actual physical signal in space.	
	0	802.11b (DSSS only)
	1	802.11b (DSSS only)
	2	802.11 a or g (OFDM)
	3	802.11 b/g (DSSS and OFDM)
	4-255	NOT USED
Type:	UINT8	
Default:	0	
XML Format:	<param name="mode" value="0"/>	

Name:	**enablepromiscuousmode**	
Description:	Determines if all packets received over the air will be sent up the stack.	
	off	Disable or send only broadcast and locally addressed unicast up
	on	Enable or send all packets up
Type:	Boolean	
Default:	off	
XML Format:	<param name="enablepromiscousmode" value="off"/>	

Name:	**beaconinterval**
Description:	The model does not currently support beacon transmissions. The only mode currently supported is ad-hoc demo mode. 0 Disable beacon transmissions
Type:	UINT32
Default:	0
XML Format:	<param name="beaconinterval" value="0"/>

Name:	**distance**
Description:	Defines the maximum distance (meters) for point to point links within the network to be supported. This is used to adjust the slot timing to account for round trip propagation delays.
Type:	UINT32
Default:	1000
XML Format:	<param name="distance" value="1000"/>

Name:	**atpenable**
Description:	The model does not currently support adaptive data rate and power selections. off Disable on Enable **(Not Supported)**
Type:	Boolean
Default:	off
XML Format:	<param name="atpenable" value="off"/>

Name:	**unicastrate**
Description:	Defines the data rate to be used for all unicast transmissions. The rate selection must be valid for the mode selected as defined in the MAC capabilities overview. The rates are defined via an index between 1 through 12.
	0 — 11Mbps (Default)
	1 — 1 Mbps
	2 — 2 Mbps
	3 — 5.5 Mbps
	4 — 11 Mbps
	6 — 6 Mbps
	6 — 9 Mbps
	7 — 12 Mbps
	8 — 18 Mbps
	9 — 24 Mbps
	10 — 36 Mbps
	11 — 48 Mbps
	12 — 54 Mbps
	13-255 — Not Defined.
Type:	UINT8
Default:	0
XML Format:	<param name="unicastrate" value="0"/>

Name:	**multicastrate**
Description:	Defines the data rate to be used for all multicast/broadcast transmissions. The rate selection must be valid for the mode selected. The rates are defined via an index between 1 through 12.
	0 — 1 Mbps (Default)
	1-12 — Same as defined for *unicastrate*
Type:	UINT8
Default:	0
XML Format:	<param name="multicastrate" value="0"/>

Name:	**rtsthreshold**
Description:	Defines the minimum packet size in bytes required to trigger RTS/CTS for unicast transmissions.
	0 — Disable RTS/CTS
Type:	UINT16
Default:	0
XML Format:	<param name="rtsthreshold" value="0"/>

Name:	**enablertsctsack**	
Description:	Defines the mechanism for emulating point to point control messages such as RTS/CTS/ACK.	
	off	Disable
	on	Enable **(Not Supported)**
Type:	Boolean	
Default:	off	
XML Format:	<param name="enablertsctsack" value="off"/>	

Name:	**fragmentationthreshold**	
Description:	The model does not currently support packet fragmentation.	
	off	Disable fragmentation
	on	Enable fragmentation **(Not Supported)**
Type:	Boolean	
Default:	off	
XML Format:	<param name="fragmentation" value="off"/>	

Name:	**wmmenable**	
Description:	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.	
	off	Disable WMM
	on	Enable WMM
Type:	Boolean	
Default:	off	
XML Format:	<param name="wmmenable" value="off"/>	

Name:	**queuesize**
Description:	Defines the size of the queue for the given access category. The format for the value within XML is defined as follows: "A:B" , where A identifies the access category (0 to 4) and B defines the queue size (1 to 255) for access category A.
Type:	String
Default:	0:255 1:255 2:255 3:255
XML Format:	<param name="queuesize" value="0:255 1:255 2:255 3:55"/>

Name:	**cwmin**
Description:	Defines the minimum contention window in slots for the appropriate access category. The format for the value within XML is defined as follows: "A:C", where A identifies the access category (0 to 4) and C defines the minimum contention window for access category A.
Type:	String
Default:	0:31 1:31 2:15 3:7
XML Format:	<param name="cwmin" value="0:31 1:31 2:15 3:7"/>

Name:	**cwmax**
Description:	Defines the maximum contention window in slots for the appropriate access category. The format for the value within XML is defined as follows: "A:D", where A identifies the access category (0 to 4) and D defines the minimum contention window for access category A.
Type:	String
Default:	0:1023 1:1023 2:63 3:15
XML Format:	<param name="cwmax" value="0:1023 1:1023 2:63 3:15"/>

Name:	**aifs**
Description:	Defines the Arbitration Inter Frame Space (AIFS) time factor in slots for the appropriate access category. The inter frame space time (usec) is computed as follows: time = aifs*slotduration + sifs, where slot duration is a function of distance and sifs is a function of the 802.11 mode. The format for the value within XML is defined as follows: "A:E", where A identifies the access category (0 to 4) and E defines the AIFS factor value in slots for access category A.
Type:	String
Default:	0:2 1:2 2:2 3:1
XML Format:	<param name="aifs" value="0:2 1:2 2:2 3:1"/>

Name:	**txop**
Description:	Defines the maximum time (usec) 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. The format for the value within XML is defined as follows: "A:E", where A identifies the access category (0 to 4) and E defines the txop duration for access category A.
Type:	String
Default:	0:0 1:0 2:0 3:0
XML Format:	<param name="txop" value="0:0 1:0 2:0 3:0"/>

Name:	**retrylimit**
Description:	Defines the number of retries permitted for the unicast messages for the appropriate access category. The format for the value within XML is defined as follows: "A:F", where A identifies the access category (0 to 4) and F defines the number of retries for access category A.
Type:	String
Default:	0:3 1:3 2:3 3:3
XML Format:	<param name="retrylimit" value="0:3 1:3 2:3 3:3"/>

Name:	**maxmsdu**
Description:	Defines the maximum packet size (Bytes) permitted for the appropriate access category. The format for the value within XML is defined as follows: "A:G", where A identifies the access category (0 to 4) and G defines the packet size for access category A. Setting the value (G) to 0 disables the feature and allows all packet sizes
Type:	String
Default:	0:0 1:0 2:0 3:0
XML Format:	<param name="maxmsdu" value="0:0 1:0 2:0 3:0"/>

IEEE 802.11 abg PHY Parameters

Name:	**mode**
Description:	Defines the 802.11 mode. The mode is included in the PHY header of all over the air messages and is used to filter out packets received from nodes that are on different modes. Should be noted that mode setting does not care about frequency and is used only to identify the actual physical signal in space.
	0 — 802.11b (DSSS only)
	1 — 802.11b (DSSS only)
	2 — 802.11 a or g (OFDM)
	3 — 802.11 b/g (DSSS and OFDM)
	4-255 — Not Defined
Type:	UINT8
Default:	0
XML Format:	<param name="mode" value="0">

Name:	**frequency**
Description:	Defines the frequency in MHz for the 802.11 network. The frequency is included in the PHY header of all over the air messages and is used to filter out packets received from nodes that are on different frequencies. This will eventually be used to determine noise from out of band packets.
Type:	UINT16
Default:	2437
XML Format:	<param name="frequency" value="2437">

Name:	**bandwidth**
Description:	Defines the bandwidth in Khz. Bandwidth is included in the PHY header. The current version does not utilize this information. An upgrade to the 802.11abg model will use this information to determine the amount of band noise introduced by co-located networks operating within the same frequency space.
Type:	UINT16
Default:	20
XML Format:	<param name="bandwidth" value="20">

Name:	**systemnoisefloor**
Description:	Defines the system noise floor (dB) for a given radio which determines the radio's sensitivity. An entry of 95 represents sensitivity of -95dB for the radio. Upon the reception of a packet, the receive power of the received packet (based on path loss, txpower and tx/rx antenna gains) is compared to the noise floor. If the receive power is < noise floor, the packet is discarded.
Type:	UINT8
Default:	95
XML Format:	<param name="systemnoisefloor" value="95">

Name:	**carriersensethreshold**
Description:	Defines the System Interference Noise Ratio (SINR) value in dB where the PHY will notify the MAC of channel busy indication when the SINR from a given packet exceeds the threshold. The MAC uses this to hold off on local transmissions until the channel is free. Setting this to 0 (default) results in disabling this feature. **Current capability does not support carrier sense for out of band signaling.**
Type:	UINT8
Default:	0
XML Format:	<param name="carriersensethreshold" value="0">

Name:	**txpower**
Description:	Defines the transmit power (dBm) and is used to compute the receive power in conjunction with path loss and antenna gains. The txpower is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="txpower" value="0">

Name:	**antennatype**	
Description:	Defines if the antenna being utilized is omni or directional.	
	0	Omni
	1	Derectional **(Not currently supported)**
	2-255	Not Defined
Type:	UINT8	
Default:	0	
XML Format:	<param name="antennatype" value="0">	

Name:	**antennagain**
Description:	Defines the antenna gain (dBi) and is used to compute the receive power in conjunction with path loss and txpower. The antennagain is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="antennagain" value="0">

Name:	**antennabeamwidth**
Description:	Defines the antenna beam width (deg) when antennatype = Directional. Currently not used.
Type:	UINT8
Default:	0
XML Format:	<param name="antennabeamwidth" value="0">

Name:	**bercurves**
Description:	Identifies the XML file containing the BER curves for the 802.11 DSSS and OFDM rates. The bercurves XML file also contains the minimum packet size, maximum packet size and packet size interval parameters which are used at initialization to generate Probability of Reception (PoR) tables/curves. The PoR tables are a function of data rate, SINR and packet size and are generated using the following formula: PoR = (1 - BER)**L where L = number of bits in the packet and BER is the bit error rate as a fn(SINR, Data Rate) as provided via the bercurves XML file. **You must use a fully qualified URI (absolute file names).**
Type:	string
Default:	ieee80211bercurves.xml
XML Format:	<param name="bercurves" value="ieee80211bercurves.xml">"

Name:	**forcepass**
Description:	Provides the ability to bypass path loss and probability of reception checks. When forcepass is enabled, all received packets during idle (not busy Tx or Rx) that pass the mode and frequency checks are passed up to the MAC. off — Disable forcepass on — Enable forcepass
Type:	Boolean
Default:	off
XML Format:	<param name="forcepass" value="off">"

Name:	**defaultconnectivity**
Description:	Defines the default connectivity of all nodes at start up prior to receiving any path loss events. off — All nodes are assumed isolated on — All nodes are assumed to be fully informed
Type:	Boolean
Default:	on
XML Format:	<param name="defaultconnectivity" value="on">"

Interactive Commands

The interactive transport command shell supports the following commands:

clear	Clear Screen
exit	Exit telnet session. Same as *quit*.
ping	Transport to Transport ping.
quit	Quit telnet session. Same as *exit*.
stats	Show or clear statistic counters.
version	Print version.
whoami	Print NEM id.

ping Usage

The ping commands sends an interactive transport PING Request to another interactive transport to elicit an interactive transport PING Response.

ping [-i interval] [-c count] [-s size] NEMID|'all'

interval	Wait interval seconds between sending each packet. The default is to wait for one second between each packet.
count	Stop after sending count ping requests packets.
size	Specifies the number of data bytes to be sent. The default is 4, which translates into 64 data bytes when combined with the 60 bytes of interactive ping header.

stats Usage

The stats command displays or clears stats related to inbound and outbound interactive transport PING requests and responses.

stats [clear]

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

```
<?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="NODE-001" id="1" definition="ieee80211abgnem.xml">
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <mac definition="ieee80211abgmac.xml">
      <!--
         * Set unicast data rate to 1Mbps
        -->
      <param name="XXX" value="XXX"/>
    </mac>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-002" id="2" definition="ieee80211abgnem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <mac definition="ieee80211abgmac.xml">
      <!--
         * Set unicast data rate to 1Mbps
        -->
      <param name="XXX" value="XXX"/>
```

```
    </mac>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-003" id="3" definition="ieee80211abgnem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <mac definition="ieee80211abgmac.xml">
      <!--
         * Set unicast data rate to 11Mbps
        -->
      <param name="XXX" value="XXX"/>
    </mac>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-004" id="4" definition="ieee80211abgnem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <mac definition="ieee80211abgmac.xml">
      <!--
         * Set unicast data rate to 54Mbps
        -->
      <param name="XXX" value="XXX"/>
    </mac>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>
</platform>
```

Review the **ieee80211abgmac.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mac SYSTEM "file:///usr/share/emane/dtd/mac.dtd">
<mac name="IEEE 802.11 MAC" library="ieee80211abgmaclayer">
  <param name="enablepromiscuousmode" value="off"/>
  <param name="distance" value="1000"/>
  <param name="rtsthreshold" value="0"/>
  <param name="mode" value="3"/>
  <param name="unicastrate" value="0"/>
  <param name="multicastrate" value="0"/>
  <param name="wmmenable" value="off"/>
  <param name="queuesize" value="0:255"/>
  <param name="retrylimit" value="0:0"/>
  <param name="cwmin" value="0:31"/>
</mac>
```

Review the **ieee80211abgphy.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE phy SYSTEM "file:///usr/share/emane/dtd/phy.dtd">
<phy name="IEEE 802.11 PHY" library="ieee80211abgphylayer">
  <param name="mode" value="0"/>
  <param name="carriersensethreshold" value="0"/>
  <param name="frequency" value="2347"/>
  <param name="bandwidth" value="20"/>
  <param name="systemnoisefloor" value="95"/>
  <param name="txpower" value="0"/>
  <param name="antennagain" value="0"/>
  <param name="bercurves" value="file:///usr/share/emane/models/ieee80211abg/xml/
ieee80211bercurves.xml"/>
  <param name="forcepass" value="off"/>
  <param name="defaultconnectivity" value="off"/>
</phy>
```

Observations Explained

This workshop provides visibility into the physical layer component of the IEEE 802.11 abg model as it pertains to packet completion rate as a function of Signal to Noise (SINR), modulation type (data rate) and packet size. The workshop is configured such that all 4 nodes have the same SINR (8 dB) when

receiving a packet from any other node. SINR of 8 dB is achieved by using the default parameters for system noise floor (-95 dB), transmit power (0 dBm) and antenna gain (0 dBi) while setting the path loss to 87 dB (via events) between all nodes. In addition, each node is configured to transmit at the following rates:

- NEM 1: 1 Mbps
- NEM 2: 1 Mbps
- NEM 3: 11 Mbps
- NEM 4: 54 Mbps

As NEM 1 pings NEMs 2 through 4, the return ping received at NEM 1 will be processed based on Bit Error Rate curves associated with the respective modulation type (BPSK, QPSK and 64-QAM). The table below shows the expected completion rates from NEM 1 to all other nodes for packet size of 128Bytes and 1024Bytes.

SINR = 8 dB			
	Completion Rate (%)		
Packet Size (Bytes)	NEM 2 (1 Mbps)	NEM 3 (11 Mbps)	NEM 4 (54 Mbps)
128	100	49	0
1024	100	0	0

The second part of the workshop requires the adjustment of the antenna gain to 13 dBi on NEM 1 which in essence raises the SINR from 8 dB to 20 dB for all packets received by NEM 1. Table below shows the expected completion rates with the higher SINR for the different rates.

SINR = 20 dB			
	Completion Rate (%)		
Packet Size (Bytes)	NEM 2 (1 Mbps)	NEM 3 (11 Mbps)	NEM 4 (54 Mbps)
128	100	100	70
1024	100	100	7

Concept Review

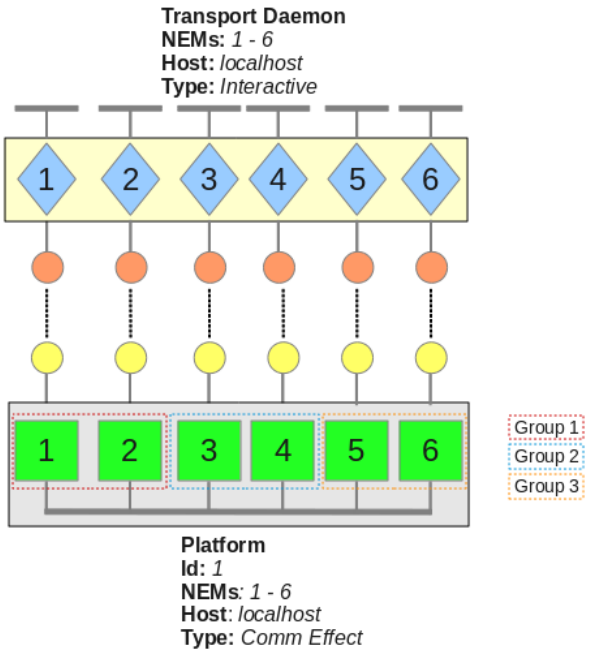
1. Why is timing so important for the IEEE 802.11 abg model?
2. How did the path loss between all the NEMs get set to 87 dB?

Workshop 09 - Comm Effect Model Model

The goal of this workshop is to become familiar with configuring the Comm Effect Model.

Deployment Description

This workshop contains a centralized Comm Effect Deployment using Interactive Transports. The single NEM Platform Server is hosting 6 NEMs.



Problem Statement

The `platform.xml` file in this workshop contains 6 NEMs with incomplete configuration. Modify the NEM Platform Server XML configuration to achieve the following:

- Place NEM 1 and NEM 2 into Comm Effect group 1
- Place NEM 3 and NEM 4 into Comm Effect group 2
- Place NEM 5 and NEM 6 into Comm Effect group 3

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify the settings using the Interactive Transport to unicast and broadcast *ping* between NEMs.

Comm Effect Shim Parameters

Name:	**defaultconnectivity:**	
Description:	Defines the default communication effects of all NEMs at start up prior to receiving any commeffect events. All filter rules, if any, are still processed regardless of whether *defaultconnectivity* is in use.	
	on	All NEMS are allowed to communication without first receiving commeffect event data.
	off	No NEMS are allowed to communication without first receiving comm effect event data that does not match any filter rules.
Type:	Boolean	
Default:	on	
XML Format:	<param name="defaultconnectivity" value="on"/>	

Name:	**filterfile:**	
Description:	The name of the CommEffect filter file to load. **You must use a fully qualified URI (absolute file names).**	
Type:	String	
Default:	None	
XML Format:	<param name="filterfile" value="file:///etc/emane/filterfile.xml"/>	

Name:	**groupid:**	
Description:	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.	
	Equal to 0	Not associated with a NEM Group
	Greater than 0	Associated with that NEM Group
Type:	UINT32	
Default:	none	
XML Format:	<param name="groupid" value="0"/>	

Name:	**enablepromiscuousmode:**	
Description:	Determines if all packets received will be sent upstream to the transport	
	on	Send only broadcast and locally addressed unicast up.
	off	Send all packets up
Type:	Boolean	
Default:	off	
XML Format:	<param name="enablepromiscuousmode" value="off"/>	

Name:	**enabletighttimingmode:**
Description:	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.**
	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
Type:	Boolean
Default:	off
XML Format:	<param name="enabletighttimingmode" value="off"/>

Name:	**receivebufferperiod:**
Description:	Speicify 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 src 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
Default:	1.0
XML Format:	<param name="receivebufferperiod" value="1.0"/>

Interactive Commands

The interactive transport command shell supports the following commands:

clear	Clear Screen
exit	Exit telnet session. Same as *quit*.
ping	Transport to Transport ping.
quit	Quit telnet session. Same as *exit*.
stats	Show or clear statistic counters.
version	Print version.
whoami	Print NEM id.

ping Usage

The ping commands sends an interactive transport PING Request to another interactive transport to elicit an interactive transport PING Response.

ping [-i interval] [-c count] [-s size] NEMID|'all'

interval	Wait interval seconds between sending each packet. The default is to wait for one second between each packet.
count	Stop after sending count ping requests packets.
size	Specifies the number of data bytes to be sent. The default is 4, which translates into 64 data bytes when combined with the 60 bytes of interactive ping header.

stats Usage

The stats command displays or clears stats related to inbound and outbound interactive transport PING requests and responses.

stats [clear]

XML Listing

Modify the **platform.xml** file to achieve the workshop goals.

```
<?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="NODE-001" id="1" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-002" id="2" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-003" id="3" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-004" id="4" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>
```

```
  <nem name="NODE-005" id="5" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8105"/>
    <param name="transportendpoint" value="localhost:8205"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-006" id="6" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8106"/>
    <param name="transportendpoint" value="localhost:8206"/>
    <shim definition="XXX">
      <!--
         Set the shim definition file
         Set the shim parameter groupid
        -->
      <param name="XXX" value="XXX" />
    </shim>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

</platform>
```

Review the **commeffectnem.xml** file.

```
<?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="transinteractive.xml"/>
</nem>
```

Concept Review

1. Is the Comm Effect Model a radio model?
2. What are filters?
3. What is an *unstructured* NEM?

Part V. Event Generators

Contents

Generators 87
Mitre Mobility Generator 88
- Generator Details 88
- Generator Data Format 88
- Data Format Sample 89

Emulation Script Generator 90
- Generator Details 90
- Data Format Sample 90

Emulation Event Log Generator 91
- Generator Details 91
- Generator Data Format 91
- Data Format Sample 92

Workshop 10 - Mitre Mobility Event Generator 93
- Deployment Description 93
- Problem Statement 93
- Mitre Mobility Generator Parameters 94
- XML Listing 95
- Concept Review 95

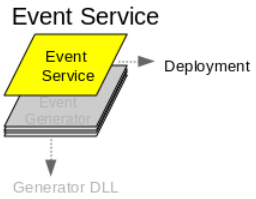
Workshop 11 - Emulation Script Generator 97
- Deployment Description 97
- Problem Statement 97
- Emulation Script Generator Parameters 98
- RF Pipe PHY Parameters 98
- XML Listing 101
- Concept Review 102

Workshop 12 - Emulation Event Log Generator 103
- Deployment Description 103
- Problem Statement 103
- Emulation Event Log Generator Parameters 104
- XML Listing 104
- Concept Review 104

Generators

Event Service XML Explained

- Specify the Event Generators created by the event service
  - List generator XML
    - *mitremobilitygenerator.xml*
- Specify deployment XML file containing emulation NEM platform relationships
  - *deployment.xml*
- Specify configuration parameter values
  - Event multicast channel (group, port, and device)



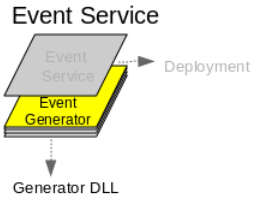

```
<eventservice name="Sample Event Service" deployment="deployment.xml">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>
  <generator name="Mobility Model Generator"
             definition="mitremobilitygenerator.xml"/>
</eventservice>
```

0.6.2

45

Event Generator XML Explained

- library attribute specifies Generator DLL
  - libmitremobilitymodelgenerator.so (Linux and OS X)
  - libmitremobilitymodelgenerator.dll (win32)
- Specify configuration parameter defaults
  - Values can be overridden by those specified in the event service XML file

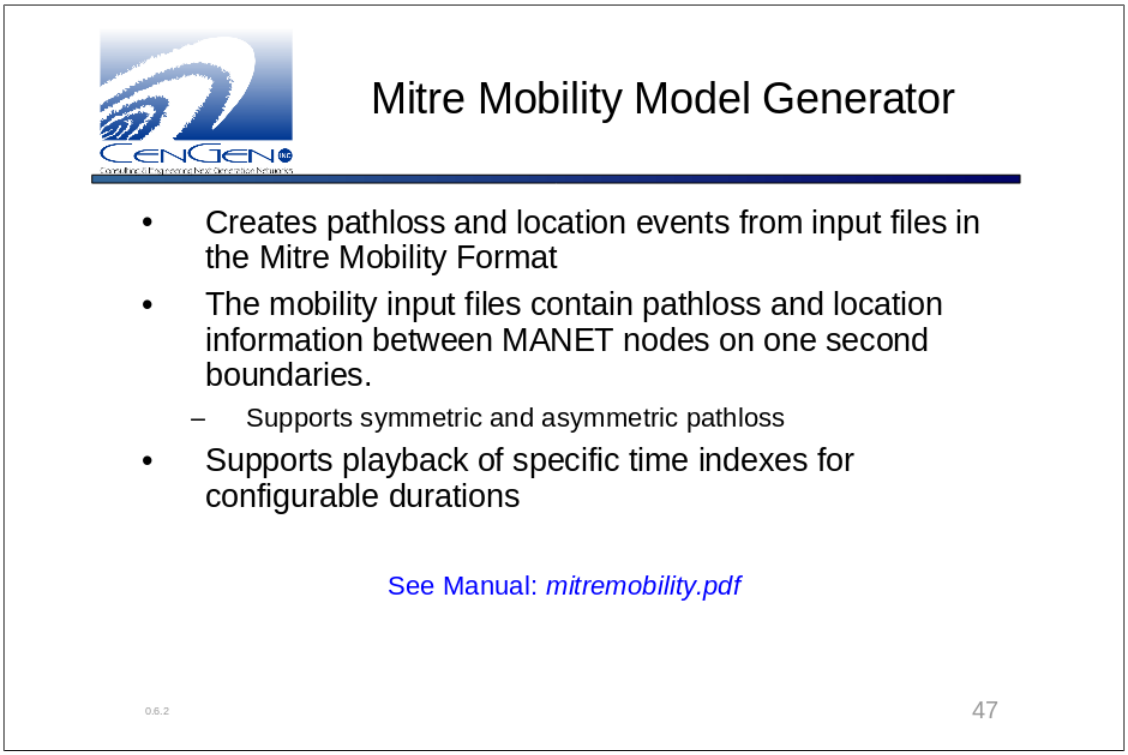



```
<eventgenerator name="mitremobilitymodel"
   library="mitremobilitygenerator">
  <param name="inputfileformat" value="matrix%02d.txt"/>
  <param name="inputfilecount" value="11"/>
  <param name="totalnodes" value="70"/>
  <param name="maxnemidpresent" value="12"/>
  <param name="repeatcount" value="1"/>
  <param name="utmzone" value="18T"/>
  <param name="entryreplay" value="0:3 1:10 2:100"/>
</eventgenerator>
```

0.6.2

46

Mitre Mobility Generator



Generator Details

The Mitre Mobility Generator creates pathloss and/or location events from input files in the Mitre Mobility Format. The file contains pathloss and location information between MANET nodes on one second boundaries. The one second connectivity matrix is sliced up and transmitted to PHY layers that have registered to receive the event.

Generator Data Format

Mitre Mobility Format is an ASCII text file containing a single entry for every pair of nodes in the mobility scenario on one second boundaries. For each second the number of entries required to completely describe the connectivity is the summation of N - i for all i from [1,N-1], where N is the number of nodes.

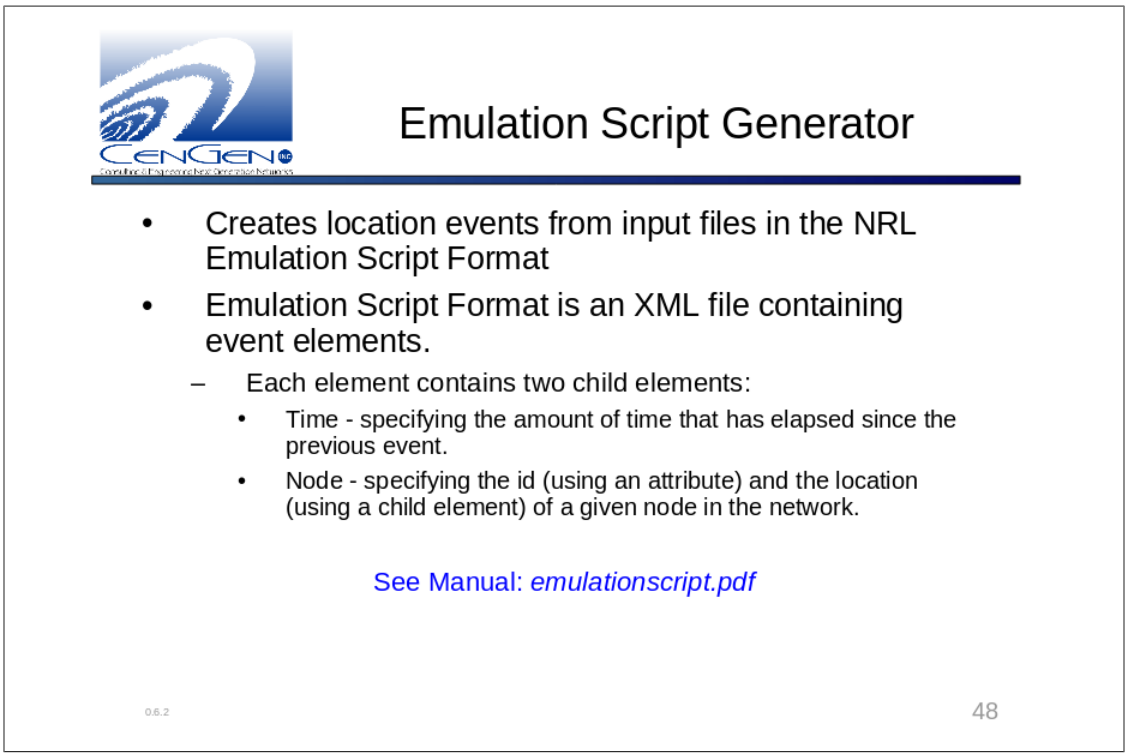
1. Time in Seconds
2. Node Target A
3. Node Target B
4. Pathloss between nodes. A single value denotes symmetric pathloss. Two values separated by a '/' denotes asymmetric pathloss. Where the first value is the pathloss from Node A to Node B and the second value is the pathloss from Node B to Node A.
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

Data Format Sample

```
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
```

Emulation Script Generator



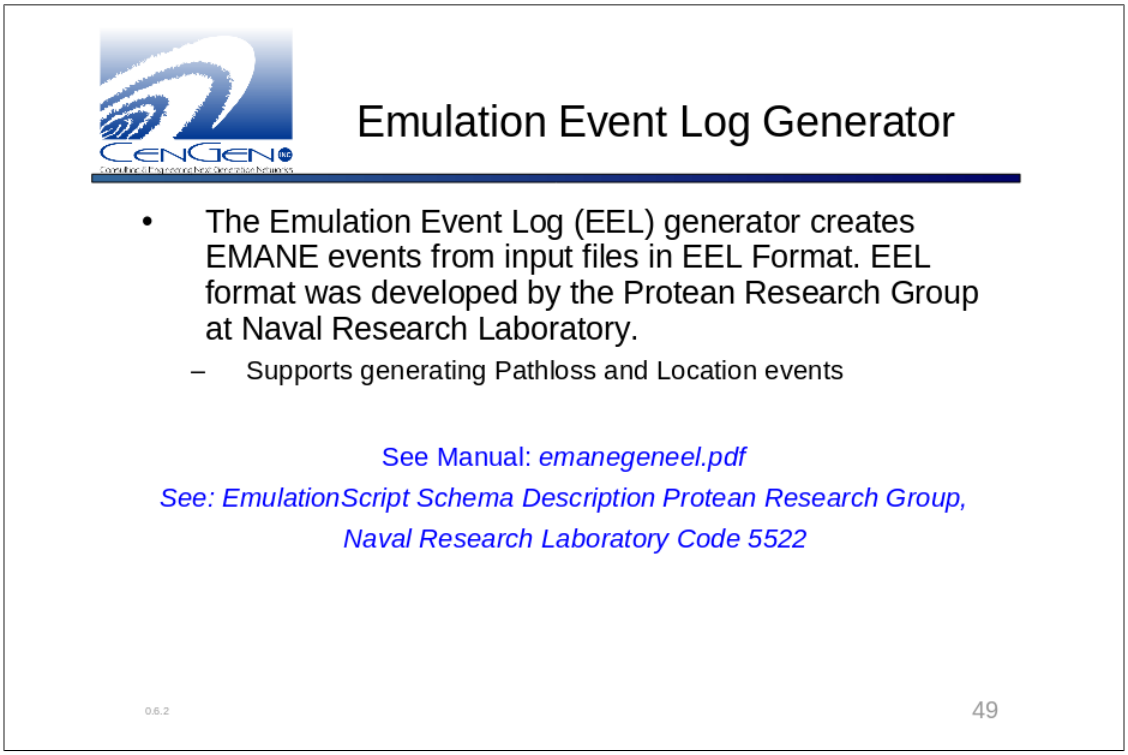
Generator Details

The emulation script event generator creates location events from input files in the Emulation Script Format. The file contains location information between MANET nodes on specific time boundaries. The location matrix is sliced up and transmitted to all EMANE component layers.

Data Format Sample

```
<?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>
```

Emulation Event Log Generator



Generator Details

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.

The EMANE 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.

Generator Data Format

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 rep- resent "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 ...>

— *EmulationScript Schema Description* Protean Research Group, Naval Research Laboratory Code 5522

Data Format Sample

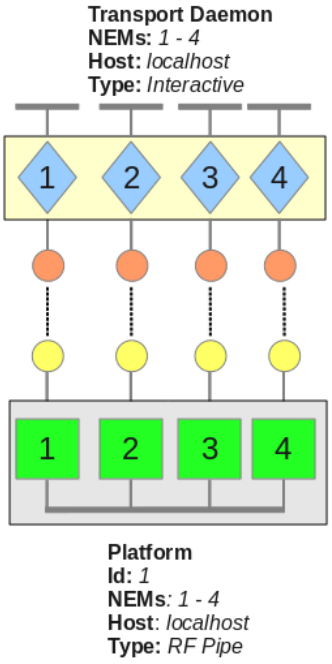
```
0.0  nem:70 pathLoss nem:22,96.3 nem:23,95.0 nem:24,95.1 nem:25,95.2 nem:26,95.3 nem:27,95.4 nem:28,95.5
0.0  nem:70 pathLoss nem:42,95.3 nem:43,95.4 nem:44,95.5 nem:45,95.0 nem:46,95.1 nem:47,95.2 nem:48,95.3
0.0  nem:70 pathLoss nem:62,95.2 nem:63,95.3 nem:64,95.4 nem:65,94.2 nem:66,94.2 nem:67,96.3 nem:68,96.3
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
```

Workshop 10 - Mitre Mobility Event Generator

The goal of this workshop is to become familiar with the Mitre Mobility Event Generator.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs. An Event Service hosting the Mitre Mobility Event Generator is used to publish pathloss and location events.



Problem Statement

The `mitremobilitygenerator.xml` file contained in this workshop is missing certain configuration. Modify the generator XML configuration to achieve the following:

- Use `/opt/mobilitymodel/path_loss_matrix_00.txt` as the single mobility input file
- Set the generator to loop continuously

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify that the EMANE Event Service is publishing events using the **locationviewer.py** and **pathlossviewer.pl** scripts.

Mitre Mobility Generator Parameters

Name:	**inputfileformat**
Description:	Full filename including path of mobility file. One or more files may be specified buy using a `printf()` style convention using the *inputfilecount* configuration item as an index.
Type:	Filename
XML Format:	<param name="inputfileformat" value="path_loss_matrix_%02d.txt"/>

Name:	**inputfilecount**
Description:	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:	UINT32
XML Format:	<param name="inputfilecount" value="1"/>

Name:	**totalnodes**
Description:	Total number of nodes whose data is contained in the mobility files.
Type:	UINT32
XML Format:	<param name="totalnodes" value="70"/>

Name:	**maxnemidpresent**
Description:	Maximum NEM Id present in test. This value is used to reduce the number of events generated when a subset of nodes from the larger mobility data are used.
Type:	UINT16
XML Format:	<param name="maxnemidpresent" value="12"/>

Name:	**repeatcount**
Description:	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 one allows you to process the data *repeatcount* times. A value of zero will process the data repeatedly, restarting indefinitely.
Type:	UINT32
XML Format:	<param name="repeatcount" value="1"/>

Name:	**utmzone**
Description:	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.
Limitation:	Position data cannot cross zones.
Type:	String
XML Format:	<param name="utmzone" value="18T"/>

Name:	**entryreplay**
Description:	Specify one or more space separated *time:count* pairs. Where time corresponds to the second of data contained in the mobility model and count represents the number of times (seconds) you want to replay that data. This effectively allows you 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 update 120 times and use the data at T3600 for 30 minutes.
Type:	String
XML Format:	<param name="entryreplay" value="0:120 3600:1800"/>

Name:	**publishlocationevents**
Description:	Create/Publish location events from mobility model input files.
Type:	Boolean
XML Format:	<param name="publishlocationevents" value="on"/>

Name:	**publishpathlossevents**
Description:	Create/Publish pathloss events from mobility model input files.
Type:	Boolean
XML Format:	<param name="publishpathlossevents" value="on"/>

XML Listing

Modify the **mitremobilitygenerator.xml** file to achieve the workshop goals.

```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE eventgenerator SYSTEM "file:///usr/share/emane/dtd/eventgenerator.dtd">
 <eventgenerator name="Mitre Mobility Generator" library="mitremobilitygenerator">
   <!--
      Set the mobility file input
     -->
   <param name="inputfileformat" value="XXX"/>
   <param name="inputfilecount" value="XXX"/>
   <param name="totalnodes" value="70"/>
   <param name="maxnemidpresent" value="40"/>

   <!--
       Set the repeat count to loop continuously
     -->
   <param name="repeatcount" value="XXX"/>
   <param name="utmzone" value="18T"/>

</eventgenerator>
```

Review the **eventservice.xml** file.

```
<?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="lo"/>
  <generator name="Mobility Model Generator" definition="mitremobilitygenerator.xml"/>
</eventservice>
```

Concept Review

1. How do you set the overall start time for the EMANE Event Service?

2. Why is the UTM Zone a configuration parameter?

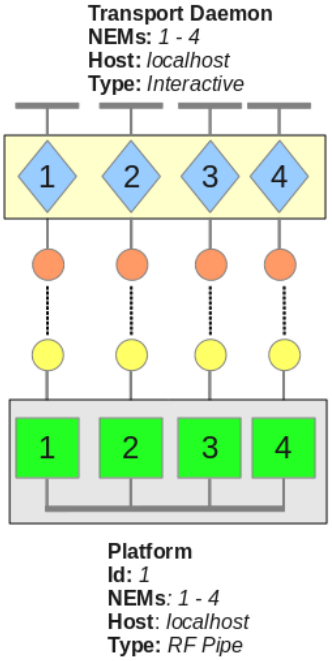
3. What is the purpose of the *entryreplay* configuration parameter?

Workshop 11 - Emulation Script Generator

The goal of this workshop is to become familiar with the Emulation Script Event Generator.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs. An Event Service hosting the Emulation Script Generator is used to publish location events.



Problem Statement

The `emulationscriptgenerator.xml` file contained in this workshop is missing certain configuration. Modify the XML configuration to achieve the following:

Use the following input files:

- `/opt/locationscript/location_00.xml`
- `/opt/locationscript/location_01.xml`
- `/opt/locationscript/location_02.xml`
- `/opt/locationscript/location_03.xml`
- Set the generator to loop continuously
- Configure the RF Pipe PHY layer to use location events to calculate path loss in realtime.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify that the EMANE Event Service is publishing events using the **locationviewer.py** script.

Emulation Script Generator Parameters

Name:	**inputfile**
Description:	Full filename including path of mobility file. One or more files may be specified by using this parameter multiple times modifying the value attribute accordingly.
Type:	Filename
XML Format:	<param name="inputfile" value="location_00.xml"/>

Name:	**repeatcount**
Description:	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 one allows you to process the data *repeatcount* times. A value of zero will process the data repeatedly, restarting indefinitely.
Type:	UINT32
XML Format:	<param name="repeatcount" value="1"/>

Name:	**schemalocation**
Description:	Specifies the location of the schema file used to validate files specified via *inputfile* parameters.
Type:	URI
XML Format:	<param name="schemalocation" value="http://configserver/schema/EmulationScriptSchema.xsd"/>

RF Pipe PHY Parameters

Name:	**frequency**
Description:	Defines the frequency used in MHz for the emulated network. The frequency is included in the PHY header of all over the air messages and serves 2 functions: 1. Filters out packets received from nodes that are on different frequencies. 2. Is used to determine free space path loss when configured to use compute path loss based on node positions using the free space propagation algorithm. See configuration parameters *pathlosssource* and *pathlossalgorithm* below.
Type:	UINT16
Default:	2437
XML Format:	<param name="frequency" value="2437">

Name:	**bandwidth**
Description:	Defines the bandwidth in Khz. Bandwidth is included in the PHY header. The current version does not utilize this information.
Type:	UINT16
Default:	20
XML Format:	<param name="bandwidth" value="20">

Name:	**txpower**
Description:	Defines the transmit the power (dBm) and is used to compute the receive power in conjunction with path loss and antenna gains. The txpower is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="txpower" value="0">

Name:	**antennagain**
Description:	Defines the antenna gain (dBi) and is used to compute the receive power in conjunction with path loss and txpower. The antennagain is included in the PHY header of all over the air messages.
Type:	UINT8
Default:	0
XML Format:	<param name="antennagain" value="0">

Name:	**bitrate**
Description:	Defines the bitrate/burstrate (kbps) of the waveform being emulated. It is used on the transmit side to compute transmission delay based on the packet size and bitrate.
Type:	UINT16
Default:	0
XML Format:	<param name="bitrate" value="0">

Name:	**propagationdelay**
Description:	Defines the propagation delay (usec) that is to be included in the transmission delay. The propagationdelay is added to the delay introduced by the bitrate defined above.
Type:	UINT32
Default:	0
XML Format:	<param name="propagationdelay" value="0">

Name:	**jitter**
Description:	Defines the jitter (usec) 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:	UINT32
Default:	0
XML Format:	<param name="jitter" value="0">

Name:	**pathlosssource**
Description:	Defines the method of use for determining path loss. The parameter accepts 2 options: 0 — Pathloss directly provided via an external event 1 — Pathloss computed real time based on external position events and selected *pathlossalgorithm*
Type:	UINT16
Default:	0
XML Format:	<param name="pathlosssource" value="0">

Name:	**pathlossalgorithm**
Description:	Defines the algorithm to be used to compute the path loss. This is valid only when *pathlosssource* is set to *1*. The algorithm selections are as follows: 0 — Free space path loss model 1 — 2 ray flat earth model
Type:	UINT16
Default:	0
XML Format:	<param name="pathlossalgorithm" value="0">

Name:	**rxpowernolossthreshold**
Description:	Defines the receive power threshold (dBm) for determining if packets will be received or dropped. This threshold is used in conjunction with the *rxpowerfulllossthreshold* to determine the probability of packet reception (PoR) based on received power as follows: Rx Power >= rxpowernolossthreshold ==> PoR = 1 rxpowerfulllossthreshold < Rx Power < rxpowernolossthreshold ==> linear packet loss Rx Power <= rxpowerfulllossthreshold ==> PoR = 0
Type:	UINT16
Default:	90
XML Format:	<param name="rxpowernolossthreshold" value="90">

Name:	**rxpowerfulllossthreshold**
Description:	See description of *rxpowernolossthreshold* above.
Type:	UINT16
Default:	110
XML Format:	<param name="rxpowerfulllossthreshold" value="110">

Name:	**defaultconnectivity**	
Description:	Defines the default connectivity of all nodes at start up prior to receiving any mobility or path loss events. When set to *on* the reception of a single path loss or position event will disable the default connectivity setting.	
	off	All nodes are assumed isolated
	on	All nodes are assumed to be fully informed
Type:	Boolean	
Default:	on	
XML Format:	<param name="defaultconnectivity" value="on">	

XML Listing

Modify the **emulationscriptgenerator.xml** file to achieve the workshop goals.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventgenerator SYSTEM "file:///usr/share/emane/dtd/eventgenerator.dtd">
<eventgenerator name="Emulation Script Generator" library="emulationscriptgenerator">

   <!--
       Add the mobility file input
     -->
   <param name="inputfile" value="XXX"/>
   <param name="inputfile" value="XXX"/>
   <param name="inputfile" value="XXX"/>
   <param name="inputfile" value="XXX"/>

   <!--
       Set the repeat count to play continuously
     -->
   <param name="repeatcount" value="XXX"/>

   <param name="schemalocation"
     value="file:///usr/share/doc/emane-gen-emulationscript-0.6.2/EmulationScriptSchema.xsd"/
>
</eventgenerator>
```

Modify the **rfpipephy.xml** file to achieve the workshop goals.

```
<!DOCTYPE phy SYSTEM "file:///usr/share/emane/dtd/phy.dtd">
<phy name="RF-PIPE PHY" library="rfpipephylayer">
  <param name="txpower" value="0"/>
  <param name="antennagain" value="0"/>
  <param name="bitrate" value="0"/>
  <param name="jitter" value="0"/>
  <param name="frequency" value="2347"/>
  <param name="bandwidth" value="20"/>
  <param name="propagationdelay" value="0"/>
  <!--
     Set the pathloss source
    -->
  <param name="pathlosssource" value="XXX"/>
  <param name="pathlossalgorithm" value="0"/>
  <param name="defaultconnectivity" value="on"/>
  <param name="rxpowernolossthreshold" value="90"/>
  <param name="rxpowerfulllossthreshold" value="110"/>
</phy>
```

Review the **eventservice.xml** file.

```
<?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="lo"/>
  <generator name="Emulation Script Generator" definition="emulationscriptgenerator.xml"/>
```

```
</eventservice>
```

Concept Review

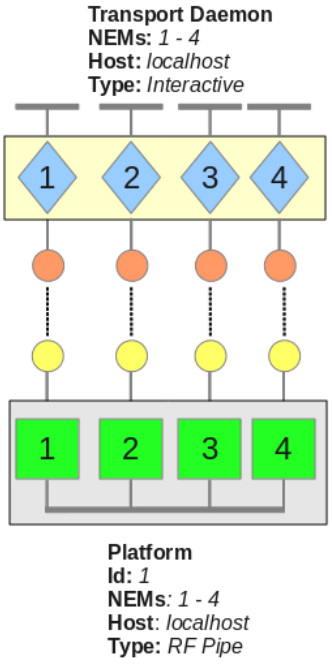
1. Why was it necessary to change the RF Pipe PHY Layer configuration?
2. Can the EMANE Event Service be configured to host multiple numbers of generators at the same time?

Workshop 12 - Emulation Event Log Generator

The goal of this workshop is to become familiar with the Emulation Event Log Event Generator.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs. An Event Service hosting the Emulation Event Log Event Generator is used to publish pathloss and location events.



Problem Statement

The `eelgenerator.xml` file contained in this workshop is missing certain configuration. Modify the generator XML configuration to achieve the following:

- Use `/opt/eelmodel/mobility.eel` as the single event input file.
- Configure a loader for *location* event type entries using the *eelloaderlocation* plugin. Use *full* updates.
- Configure a loader for *pathloss* event type entries using the *eelloaderpathloss* plugin. Use *full* updates.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify that the EMANE Event Service is publishing events using the **locationviewer.py** script.

Emulation Event Log Generator Parameters

Name:	**inputfile**
Description:	Full filename including path of EEL file. Additional EEL files may be specified using multiple *inputfile* parameters. Files are processed in the order they appear in the XML.
Multiplicity:	1:N
Type:	Filename
XML Format:	<param name="inputfile" value="/opt/mobility.eel"/>

Name:	**loader**
Description:	Map EEL event type keywords to EEL loader plugins. The optional **full** or **delta** specification 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**.
Syntax:	<eventType>:<plugin>[:full|delta]
Multiplicity:	0:N
Type:	string
XML Format:	<param name="loader" value="pathloss:eelloaderpathloss:full"/>

XML Listing

Modify the **eelgenerator.xml** file to achieve the workshop goals.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventgenerator SYSTEM "file:///usr/share/emane/dtd/eventgenerator.dtd">
<eventgenerator name="EEL Generator" library="eelgenerator">
  <!--
     * Add the mobility file input
    -->
  <param name="inputfile" value="XXX"/>
  <!--
     * Add loader parameter for location entries
    -->
  <param name="loader" value="XXX"/>
  <!--
     * Add loader parameter for pathloss entries
    -->
  <param name="loader" value="XXX"/>
</eventgenerator>
```

Review the **eventservice.xml** file.

```
<?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="lo"/>
  <generator name="EEL Generator" definition="eelgenerator.xml"/>
</eventservice>
```

Concept Review

1. What is the difference between the *full* and *delta* publish modes?
2. Give an example of when you might want to use the *full* publish mode?
3. How is an EEL loader associated with an event type keyword?

Part VI. Event Agents

Contents

Agents 106
GPSd Location Agents 107
Workshop 13 - GPSd Location Agent 108
Deployment Description 108
Problem Statement 108
GPSd Location Agent Parameters 108
XML Listing 108
Concept Review 109

Agents

Event Daemon XML Explained

- Specify the Event Agents created by the event daemon
  - List Agent XML
    - gpsdlocationagent.xml
- Specify the NEM id to associate with the event agents
  - Not used by all event agents (set to 0)
- Specify configuration parameter values
  - Event multicast channel (group, port, and device)

Event Daemon

```
<eventdaemon name="EMANE Event Daemon" nemid="1">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth0"/>
  <agent definition="gpsdlocationagent.xml"/>
</eventdaemon>
```

0.6.2 52

Event Agent XML Explained

- library attribute specifies event agent DLL
  - libgpslocationagent.so (Linux and OS X)
  - No win32 Support
- Specify configuration parameter defaults
  - Values can be overridden by those specified in the event daemon XML

Event Daemon

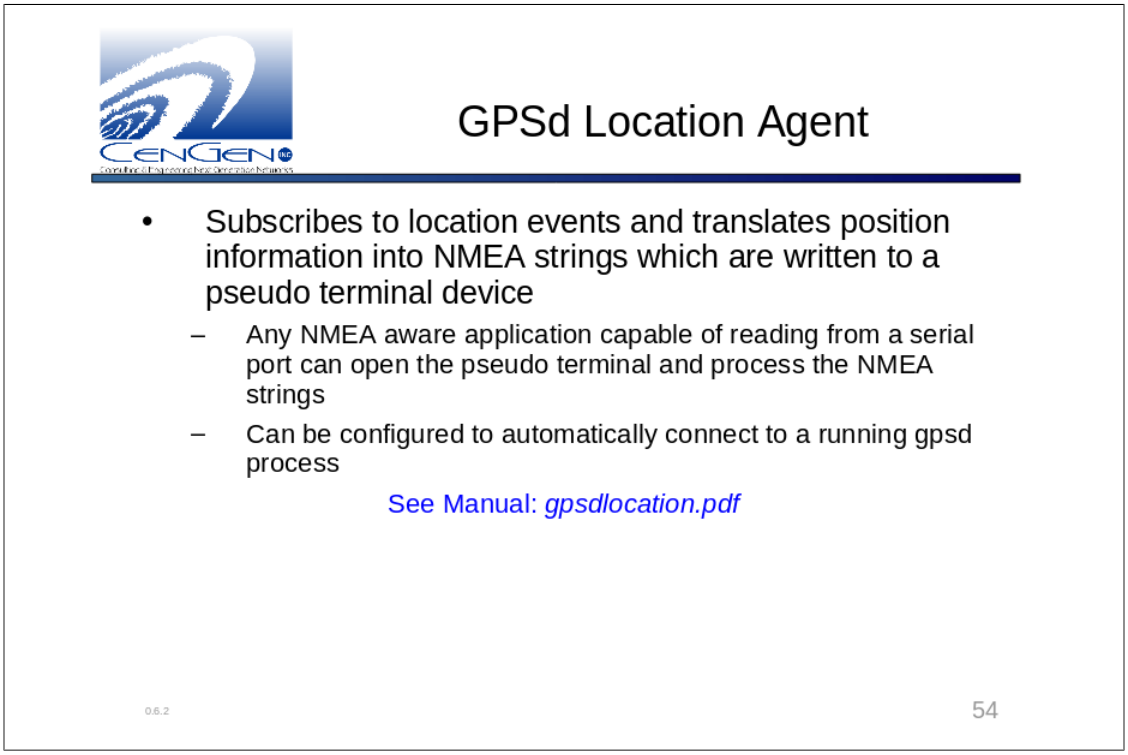
```
<eventagent name="gpsdlocationagent"
            library="gpsdlocationagent">
  <param name="gpsdconnectionenabled" value="off"/>
  <param name="pseudoterminalfile"
         value="/tmp/gpsdlocation.pty"/>
  <param name="gpsdcontrolsocket"
         value="/tmp/gpsd.control"/>
</eventagent>
```

0.6.2 53

GPSd Location Agents

The GPSd Location Agent creates a pseudo terminal and connects the terminal to a locally running GPSd process as a GPS device. NMEA strings are created from received location events and transmitted to GPSd making position information available to any GPSd aware applications using the standard libgpsd API.

Using the default configuration gpsdlocation agent will not actively attach to GPSd, instead it will create a file containing the name of the pseudo terminal currently in use which can then be used by any application expecting to receive NMEA strings from a terminal device.



Workshop 13 - GPSd Location Agent

The goal of this workshop is to become familiar with the GPSd Location Agent.

Deployment Description

This workshop contains an Event Service hosting the Mitre Mobility Event Generator publishing location events. The Event Daemon is hosting the GPSd Location Agent.

Problem Statement

The `gpsdlocationagent.xml` file contained in this workshop is missing certain configuration. Modify the agent XML configuration to achieve the following:

- Disable automatic connection to gpsd.
- Use `/tmp/gpsdlocation.pty` as the pty name file.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify location event translation using the GPSd client applications **gpsmon** and **cgps**.

GPSd Location Agent Parameters

Name:	**gpsdcontrolsocket**
Description:	The name of the GPSd control socket for adding the pseudo terminal to the device list.
Type:	Filename
Default:	/tmp/gpsd.control
XML Format:	<param name="gpsdcontrolsocket" value="/tmp/gpsd.control"/>

Name:	**pseudoterminalfile**
Description:	The name of the file to create which will contain the name of the pseudo terminal in use by the gpsdlocation agent. Only created when *gpsdconnectionenabled* set to *off*.
Type:	Filename
Default:	/tmp/gpsdlocation.pty
XML Format:	<param name="pseudoterminalfile" value="/tmp/gpsdlocation.pty"/>

Name:	**gpsdconnectionenabled**
Description:	Switch to set gpsdlocation 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
Default:	off
XML Format:	<param name="gpsdconnectionenabled" value="off"/>

XML Listing

Modify the **gpsdlocationagent.xml** file to achieve the workshop goals.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE eventagent SYSTEM "file:///usr/share/emane/dtd/eventagent.dtd">
<eventagent name="gpsdlocationagent" library="gpsdlocationagent">
  <!--
     Disable automatic connection to gpsd
    -->
  <param name="gpsdconnectionenabled" value="XXX"/>

  <!--
     Set the pseudo terminal file filename
    -->
  <param name="pseudoterminalfile" value="XXX"/>

  <param name="gpsdcontrolsocket" value="/tmp/gpsd.control"/>
</eventagent>
```

Review the **eventdaemon.xml.in** file. *Note the @ID@ variable is a replaced with your Training Id when the workshop is built.*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE eventdaemon SYSTEM "file:///usr/share/emane/dtd/eventdaemon.dtd">
<eventdaemon name="EMANE Event Daemon" nemid="@ID@">
  <param name="eventservicegroup" value="224.1.2.8:45703"/>
  <param name="eventservicedevice" value="eth1"/>
  <agent definition="gpsdlocationagent.xml"/>
</eventdaemon>
```

Concept Review

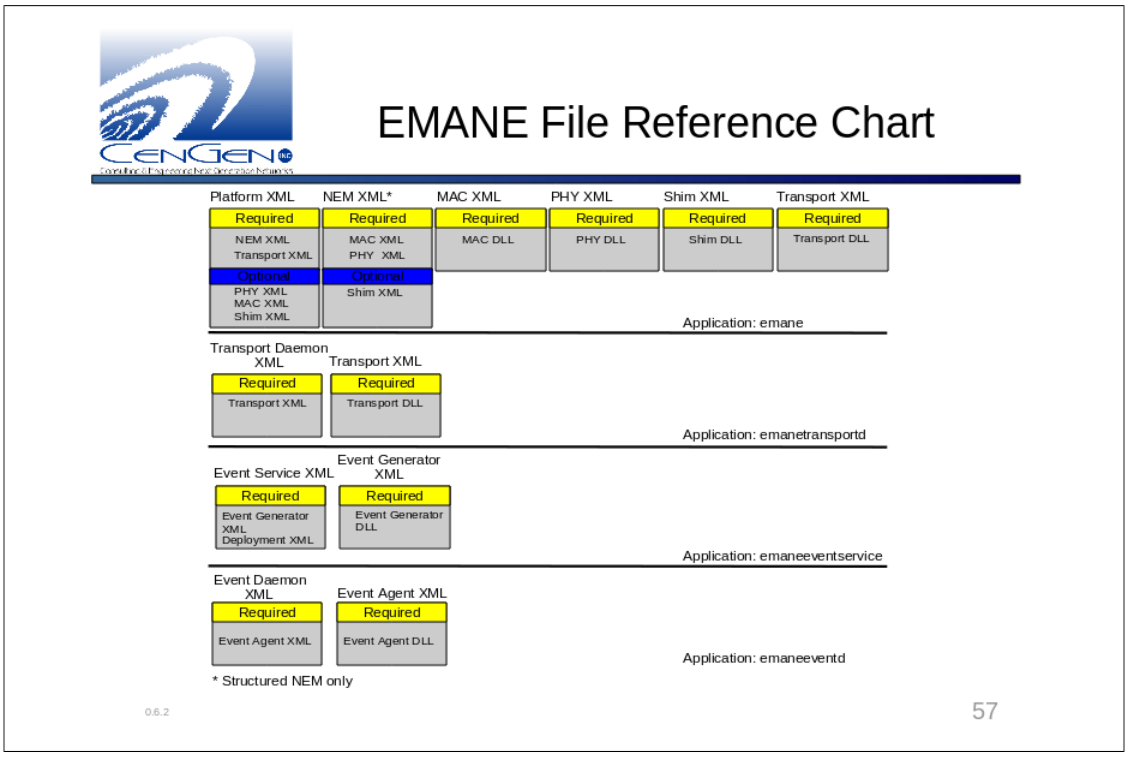
1. What is the purpose of the *pseudoterminalfile* configuration parameter?
2. Does this agent only work with gpsd?

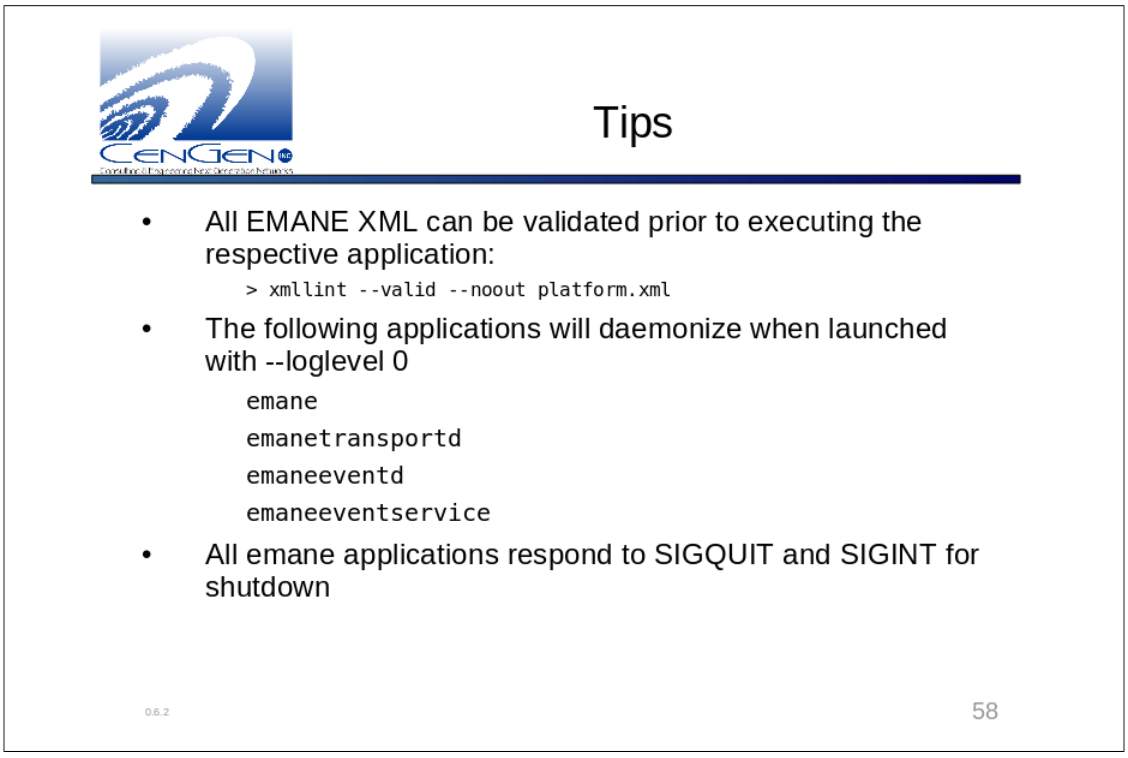
Part VII. Tips

Contents

Tips 111

Tips


EMANE File Reference Chart
Platform XML
Required
NEM XML
Transport XML
Optional
PHY XML
MAC XML
Shim XML
NEM XML*
Required
MAC XML
PHY XML
Optional
Shim XML
MAC XML
Required
MAC DLL
PHY XML
Required
PHY DLL
Shim XML
Required
Shim DLL
Transport XML
Required
Transport DLL
Application: emane
Transport Daemon XML
Required
Transport XML
Transport XML
Required
Transport DLL
Application: emanetransportd
Event Service XML
Required
Event Generator XML Deployment XML
Event Generator XML
Required
Event Generator DLL
Application: emaneeventservice
Event Daemon XML
Required
Event Agent XML
Event Agent XML
Required
Event Agent DLL
Application: emaneeventd
* Structured NEM only
0.6.2
57


Tips
All EMANE XML can be validated prior to executing the respective application:
> xmllint --valid --noout platform.xml
The following applications will daemonize when launched with --loglevel 0
emane
emanetransportd
emaneeventd
emaneeventservice
All emane applications respond to SIGQUIT and SIGINT for shutdown
0.6.2
58

Tips

- EMANE applications can use local or remote XML
  - Remote XML configuration can simplify large distributed depolyments

```
> emane http://configserver/test1/platform1.xml --logl 4
```

- Use the `emaneconvertdtdpath` utility to modify the dtd locations of your XML and DTDs

```
> emaneconvertdtdpath –dtdpath http://configserver/dtd *.xml *.dtd
```

- Launch applications with --loglevel 4 to see verbose developer logs for debugging purposes
- Configure devices dedicated to Raw Transports with atypical addresses and full netmasks to highlight they should not be used for direct communication

0.6.2 59

Tip - Multichannel Gateway

- A multichannel gateway is created using mutiple NEMs and transport instances and by configuring the appropriate routing
  - 1 NEM per channel
  - 1 Transport per channel
- Recommendations
  - NEMs should be part of the same platform NEM Server
  - Transport instances should be part of the same transport daemon
- Each gateway NEM (channel) should have PHY Layer separation
  - example: different frequency assignments or different NEM types
- Each NEM must have a unique id
  - Use the NEM name attribute to convey gateway ownership

```
<nem name="NODE-001 CHAN 2" id="10" definition="wipnnem.xml">
```

0.6.2 60

Tip - Multichannel Gateway

- Two Channel gateway
  - Chan 1 – 250MHz
  - Chan 2 – 350 MHz
- Name attribute used to provide gateway ownership
  - NODE-001 CHAN 1
  - NODE-001 CHAN 2
- NEM Ids are unique:
  - 1
  - 10

```
<platform name="Platform 1" id="1">
  <param name="otamanagergroup" value="224.1.2.8:45702"/>
  <param name="eventservicegroup" value="224.1.2.8:45703"/>

  <nem name="NODE-001 CHAN 1" id="1" definition="wipnnem.xml">
    <param name="platformendpoint" value="localhost:8181"/>
    <param name="transportendpoint" value="localhost:8171"/>
    <phy definition="rfpipephy.xml">
      <param name="frequency" value="250"/>
    </phy>
    <transport definition="transraw.xml" group="single1">
      <param name="device" value="eth5" />
    </transport>
  </nem>

  <nem name="NODE-001 CHAN 2" id="10" definition="wipnnem.xml">
    <param name="platformendpoint" value="localhost:8190"/>
    <param name="transportendpoint" value="localhost:8160"/>
    <phy definition="rfpipephy.xml">
      <param name="frequency" value="350"/>
    </phy>
    <transport definition="transraw.xml" group="single1">
      <param name="device" value="eth10" />
    </transport>
  </nem>
</platform>
```

0.6.2 61

Tip -XML Auto Generation

- Most EMANE XML can be autogenerated using one of two methods:
  - Define the platform XML files and use **`emanegentransportxml`** and **`emanegendeploymentxml`**

    See man page: *emanegentransportxml*
    See man page: *emanegendeploymentxml*
  - Use **`emanegenxml`** and create a text input file describing the deployment

    See man page: *emanegenxml*

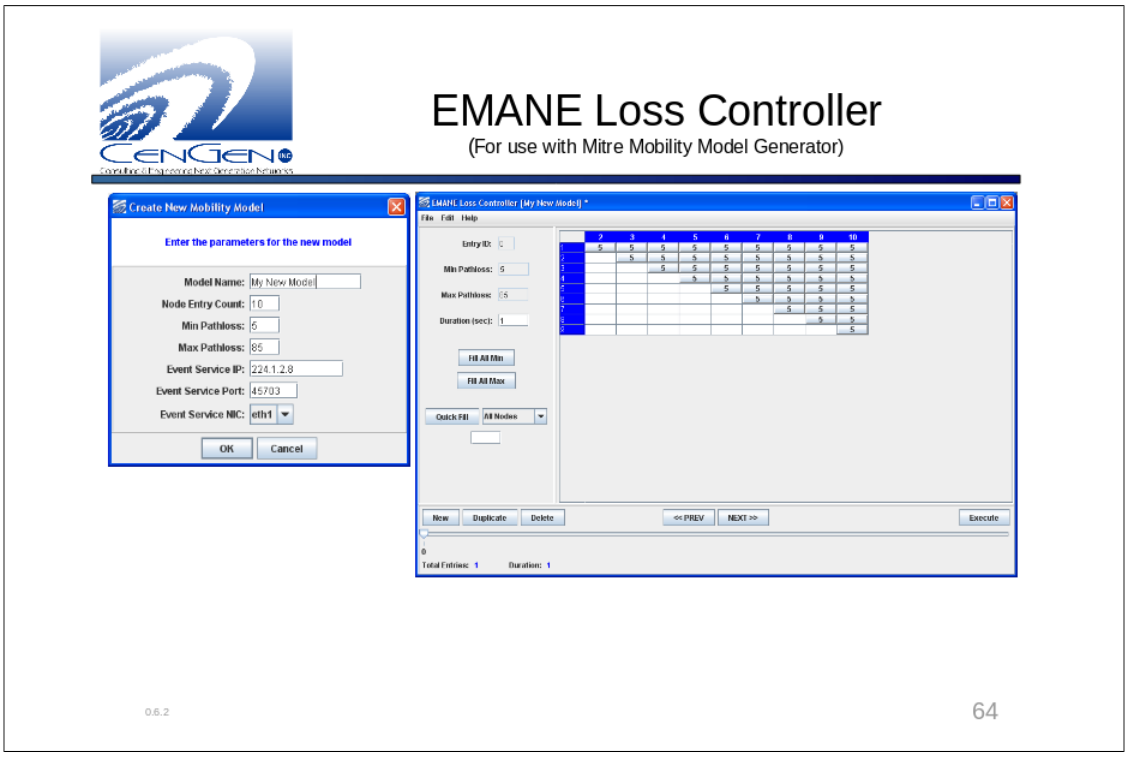
0.6.2 62

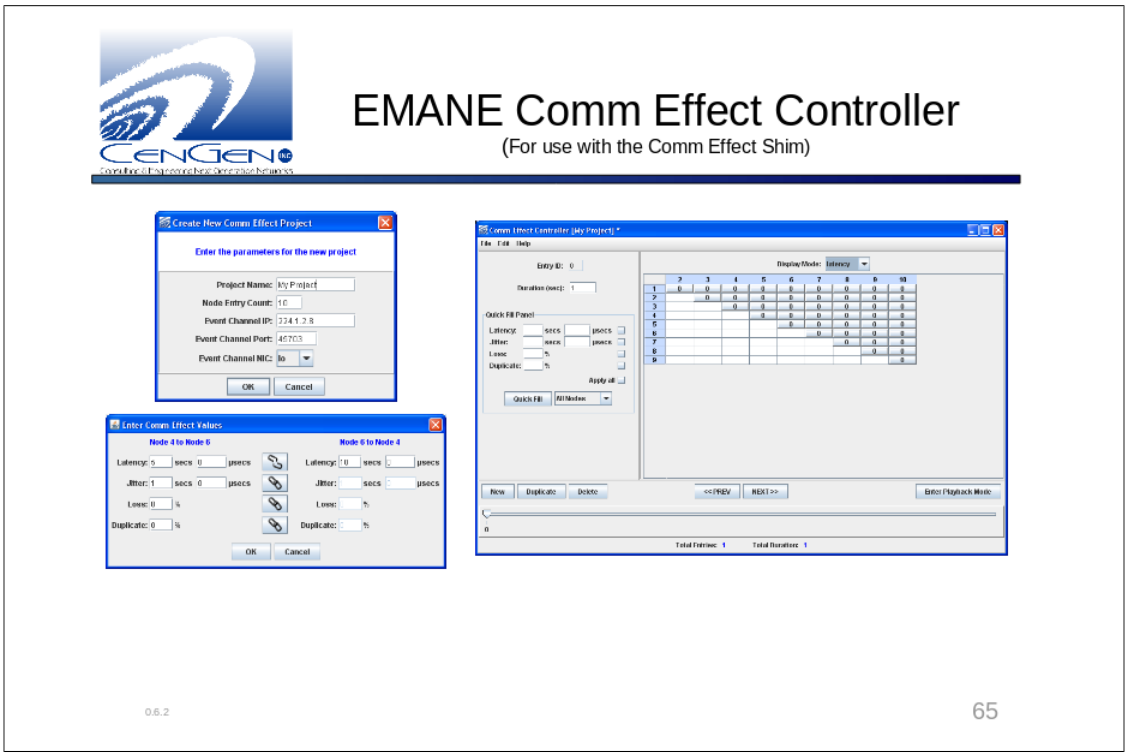
Part VIII. Controllers

Contents

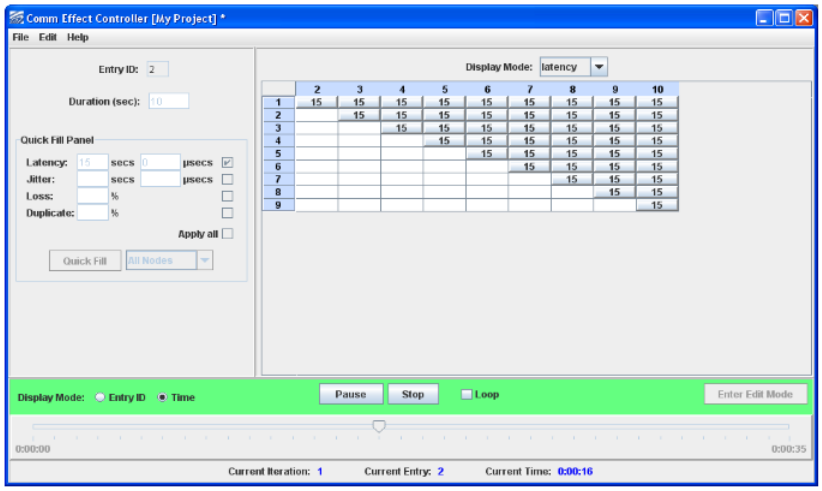
Controllers 115
Workshop 14 - EMANE Loss Controller 117
Deployment Description 117
Problem Statement 117
XML Listing 117
Concept Review 118
Workshop 15 - EMANE Comm Effect Controller 119
Deployment Description 119
Problem Statement 119
XML Listing 119
Concept Review 120

Controllers





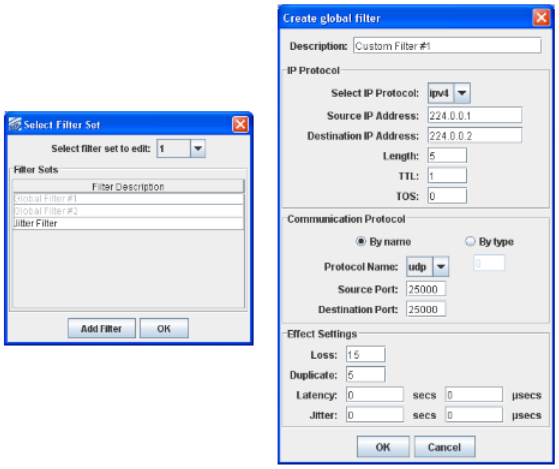
Playback Mode



0.6.2

66

Filters



0.6.2

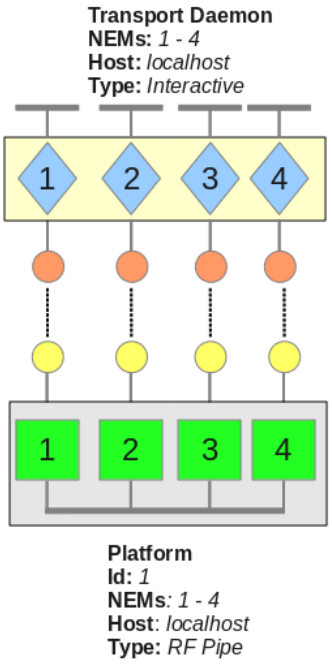
67

Workshop 14 - EMANE Loss Controller

The goal of this workshop is to become familiar with the EMANE Loss Controller application.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 4 NEMs.



Problem Statement

Using the **emanelosscontroller** to create and execute a 4 node loss scenario. Create a scenario to achieve the following:

- Use *lo* as the event device.
- Set the no loss receive threshold to *90*.
- Set the full loss receive threshold to *110*.
- Use the controller to publish pathloss events to modify link conditions.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify the executing controller scenario by using the Interactive Transport to unicast and broadcast *ping* between NEMs.

XML Listing

Review the **platform.xml** file.

```
<?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="NODE-001" id="1" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-002" id="2" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-003" id="3" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-004" id="4" definition="rfpipenem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>
</platform>
```

Concept Review

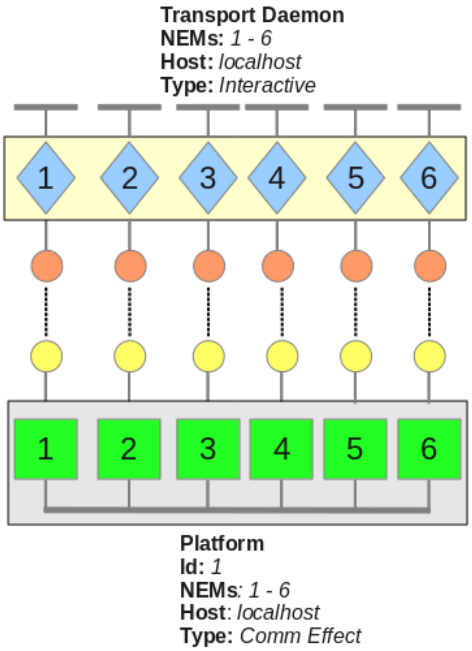
1. What are the file components of a loss controller project?

Workshop 15 - EMANE Comm Effect Controller

The goal of this workshop is to become familiar with the EMANE Comm Effect Controller application.

Deployment Description

This workshop contains a centralized Comm Effect Deployment using Interactive Transports. The single NEM Platform Server is hosting 6 NEMs.



Problem Statement

Using the **emanecommeffectcontroller** to create and execute a 6 node loss scenario. Create a scenario to achieve the following:

- Use *lo* as the event device.
- Use the controller to publish comm effect events to modify link conditions.

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify the executing controller scenario by using the Interactive Transport to unicast and broadcast *ping* between NEMs.

XML Listing

Review the **platform.xml** file.

```
<?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="NODE-001" id="1" definition="commeffectnem.xml">
```

```
    <param name="platformendpoint" value="localhost:8101"/>
    <param name="transportendpoint" value="localhost:8201"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-002" id="2" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8102"/>
    <param name="transportendpoint" value="localhost:8202"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-003" id="3" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8103"/>
    <param name="transportendpoint" value="localhost:8203"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-004" id="4" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8104"/>
    <param name="transportendpoint" value="localhost:8204"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-005" id="5" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8105"/>
    <param name="transportendpoint" value="localhost:8205"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

  <nem name="NODE-006" id="6" definition="commeffectnem.xml">
    <param name="platformendpoint" value="localhost:8106"/>
    <param name="transportendpoint" value="localhost:8206"/>
    <transport definition="transinteractive.xml" group="1"/>
  </nem>

</platform>
```

Concept Review

1. What are the file components of a Comm Effect Controller project?

Part IX. More On Deployments

Contents

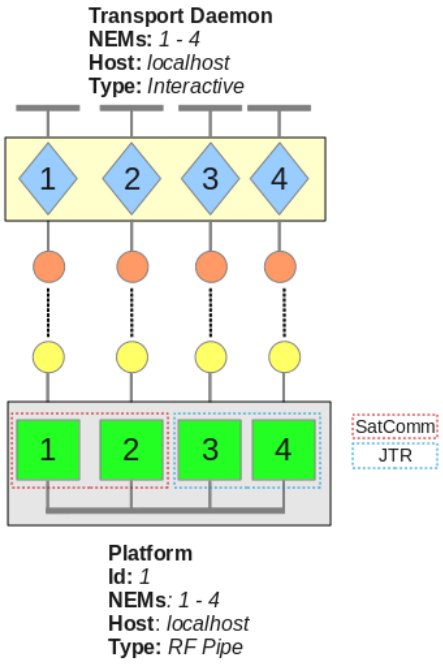
Workshop 16 - Surrogate Radio Models using RF Pipe 122
Deployment Description 122
Problem Statement 122
XML Listing 123

Workshop 16 - Surrogate Radio Models using RF Pipe

The goal of this workshop is to become familiar with using the RF Pipe Model to create surrogate radio definitions.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Interactive Transports. The single NEM Platform Server is hosting 2 NEMs. Two NEMs are surrogate SatComm radios and two NEMs are surrogate JTR radios.



Problem Statement

The `satcommnem.xml` file and the `jtrnem.xml` file are surrogate radio definitions. Use interactive sessions to verify the surrogate model performance.

- NEM 1 and NEM 2 are SatComms
- NEM 3 and NEM 4 are JTR radios

Use the **start_workshop.sh** and **stop_workshop.sh** scripts to *start* and *stop* the workshop EMANE components.

Verify the following using the Interactive Transport sessions:

1. Node 1 ping to Node 2 with 1000 byte packets results in ~360 msec delay (80 msec for bitrate and 100 msec for propagation delay).

2. Node 1 can only send ~ 12 1000 byte packets per second and maintain the 360 msec latency. Once you exceed 12 packets per second the delay will start to increase.

3. Node 3 ping to Node 4 with 1000 byte packets results in ~56 msec delay (8 msec for the bitrate and 20 msec for propagation delay).

4. Node 3 can send about 125 packets per second before the latency starts to grow.

XML Listing

Review the **satcommnem.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="SatComm NEM">
  <mac definition="rfpipemac.xml"/>
  <phy definition="rfpipephy.xml">
    <param name="frequency" value="2100"/>
    <param name="bitrate" value="100"/>
    <param name="propagationdelay" value="100000"/>
    <param name="jitter" value="0"/>
  </phy>
  <transport definition="transinteractive.xml"/>
</nem>
```

Review the **jtrnem.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nem SYSTEM "file:///usr/share/emane/dtd/nem.dtd">
<nem name="JTR NEM">
  <mac definition="rfpipemac.xml"/>
  <phy definition="rfpipephy.xml">
    <param name="frequency" value="2500"/>
    <param name="bitrate" value="1000"/>
    <param name="propagationdelay" value="20000"/>
    <param name="jitter" value="0"/>
  </phy>
  <transport definition="transinteractive.xml"/>
</nem>
```

Part X. Advanced Event Capabilities

Contents

Event Service Language Bindings 125
Workshop 17 - libemaneeventservice 126
Description 126
Publisher Sample Application 126
Subscriber Sample Application 126
libemaneeventservice Configuration 126
Configuration Format 127
Workshop 18 - libemaneeventservice Perl language bindings 128
Description 128
Sample Scripts 128
Workshop 19 - libemaneeventservice Python language bindings 129
Description 129
Sample Scripts 129

Event Service Language Bindings

Event Service Language Bindings

- libemaneeventservice – C Language API
  - Perl: EMANE::EventService
  - Python: emaneeventservice.EventService
- libemaneeventpathloss– C Language API
  - Perl: EMANE::Event::Pathloss
  - Python: emaneeventpathloss.EventPathloss
- libemaneeventlocation – C Language API
  - Perl: EMANE::Event::Location
  - Python: emaneeventlocation.EventLocation
- libemaneeventcommeffect – C Language API
  - Perl: EMANE::Event::CommEffect
  - Python: emaneeventcommeffect.EventCommEffect

0.6.2 72

Workshop 17 - libemaneeventservice

The goal of this workshop is to become familiar with libemaneeventservice.

Description

libemaneeventservice is a C library that exposes a publisher/subscriber API for the EMANE event service.

The following man pages contain detailed API information:

1. emaneEventService (3)
2. emaneEventPathloss (3)
3. emaneEventLocation (3)
4. emaneEventCommEffect (3)

Publisher Sample Application

Review the **publisher** sample application which creates and publishes pathloss, location, and comm effect events.

usage: publisher [-p] [-c] [-l] [-n msgcount] [-x nodecount]

-p	Generate Pathloss events
-c	Generate CommEffect events
-l	Generate Location events
-n msgcount	Send *msgcount* events
-x nodecount	Set the number of nodes in the deployment

Subscriber Sample Application

Review the **subscriber** sample application which subscribes to pathloss, location, and comm effect events and prints a summary of received events.

usage: subscriber [-q]

-q	Quiet mode. Do not print

libemaneeventservice Configuration

Configuration for the event service is searched for in three locations prior to using default configuration

1. Environment Variable LIBEMANEEVENTSERVICECONFIG

   If LIBEMANEEVENTSERVICECONFIG exists it will be used. This variable may be set to a configuration file name.

2. $HOME/.libemaneeventservice.xml

   If .libemaneeventservice.xml exists it will be used.

3. /etc/libemaneeventservice.xml

If /etc/libemaneeventservice.xml exists it will be used.

4. Library default values

   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.**

Configuration Format

```
<?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>
```

Workshop 18 - libemaneeventservice Perl language bindings

The goal of this workshop is to become familiar with libemaneeventservice Perl language bindings.

Description

libemaneeventservice is a C library that exposes a publisher/subscriber API for the EMANE event service.

The following man pages contain detailed API information:

1. EMANE::EventService (3)
2. EMANE::Event::Pathloss (3)
3. EMANE::Event::Location (3)
4. EMANE::Event::CommEffect (3)

Sample Scripts

Review the following sample scripts:

ex01_publisher.pl	Simple publisher example
ex02_subscriber.pl	Simple subscriber example
ex03_subscriber_default.pl	Simple subscriber example using default callback
ex04_publisher_pathloss.pl	Simple pathloss publisher
ex05_subscriber_pathloss.pl	Simple pathloss subscriber that prints out event info
ex06_publisher_location.pl	Simple location publisher
ex07_subscriber_location.pl	Simple location subscriber that prints out event info
ex08_publisher_commeffect.pl	Simple comm effect publisher
ex09_subscriber_commeffect.pl	Simple comm effect subscriber that prints out event info

Workshop 19 - libemaneeventservice Python language bindings

The goal of this workshop is to become familiar with libemaneeventservice Python language bindings.

Description

libemaneeventservice is a C library that exposes a publisher/subscriber API for the EMANE event service.

The following pydoc contain detailed API information:

1. emaneeventservice
2. emaneeventservice.EventService
3. emaneeventpathloss
4. emaneeventpathloss.EventPathloss
5. emaneeventlocation
6. emaneeventlocation.EventLocation
7. emaneeventcommeffect
8. emaneeventcommeffect.EventCommEffect

Sample Scripts

Review the following sample scripts:

ex01_publisher.py	Simple publisher example
ex02_subscriber.py	Simple subscriber example
ex03_publisher_pathloss.py	Simple pathloss publisher
ex04_subscriber_pathloss.py	Simple pathloss subscriber that prints out event info
ex05_publisher_location.py	Simple location publisher
ex06_subscriber_location.py	Simple location subscriber that prints out event info
ex07_publisher_commeffect.py	Simple comm effect publisher
ex08_subscriber_commeffect.py	Simple comm effect subscriber that prints out event info

Part XI. Putting It All Together

Contents

Workshop 20 - Putting it All together: Distributed RF Pipe MANET 131
Deployment Description 131
Workshop 21 - Putting it All together: Centralized RF Pipe MANET 132
Deployment Description 132
Workshop 22 - Putting it All together: Distributed Comm Effect MANET 133
Deployment Description 133
Workshop 23 - Putting it All together: Centralized Comm Effect MANET 134
Deployment Description 134
Workshop 24 - Putting it All together: Centralized IEEE 802.11 abg MANET 135
Deployment Description 135

Workshop 20 - Putting it All together: Distributed RF Pipe MANET

The goal of this workshop is to have the entire class interact using a distributed RF Pipe deployment.

Deployment Description

This workshop contains a distributed RF Pipe Deployment using Virtual Transports. Each student will run their own NEM Platform Server hosting a single NEM.

Each student will run the following applications:

1. emane
2. emanetransportd
3. emaneeventd
4. olsrd
5. gpsd

The instructor will run the following application:

1. emaneeventservice

Use the **olsrviewer.sh** command to visually monitor olsrd.

Workshop 21 - Putting it All together: Centralized RF Pipe MANET

The goal of this workshop is to have the entire class interact using a centralized RF Pipe deployment.

Deployment Description

This workshop contains a centralized RF Pipe Deployment using Virtual Transports. Each student will run their own NEM transport.

Each student will run the following applications:

1. emanetransportd
2. emaneeventd
3. olsrd
4. gpsd

The instructor will run the following application:

1. emane
2. emaneeventservice

Use the **olsrviewer.sh** command to visually monitor olsrd.

Workshop 22 - Putting it All together: Distributed Comm Effect MANET

The goal of this workshop is to have the entire class interact using a distributed Comm Effect deployment.

Deployment Description

This workshop contains a distributed Comm Effect Deployment using Virtual Transports. Each student will run their own NEM Platform Server hosting a single NEM.

Each student will run the following applications:

1. emane
2. emanetransportd
3. emaneeventd
4. olsrd
5. gpsd

The instructor will run the following application:

1. emaneeventservice

Use the **olsrviewer.sh** command to visually monitor olsrd.

Workshop 23 - Putting it All together: Centralized Comm Effect MANET

The goal of this workshop is to have the entire class interact using a centralized Comm Effect deployment.

Deployment Description

This workshop contains a centralized Comm Effect Deployment using Virtual Transports. Each student will run their own NEM transport.

Each student will run the following applications:

1. emanetransportd
2. emaneeventd
3. olsrd
4. gpsd

The instructor will run the following application:

1. emane
2. emaneeventservice

Use the **olsrviewer.sh** command to visually monitor olsrd.

Workshop 24 - Putting it All together: Centralized IEEE 802.11 abg MANET

The goal of this workshop is to have the entire class interact using a centralized IEEE 802.11 abg deployment.

Deployment Description

This workshop contains a centralized IEEE 802.11 abg Deployment using Virtual Transports. Each student will run their own NEM transport.

Each student will run the following applications:

1. emanetransportd
2. emaneeventd
3. olsrd
4. gpsd

The instructor will run the following application:

1. emane
2. emaneeventservice

Use the **olsrviewer.sh** command to visually monitor olsrd.

Part XII. Manual Pages

Contents

C emaneEventService 137
C emaneEventLocation 141
C emaneEventPathloss 145
C emaneEventCommEffect 149
Perl EMANE::EventService 153
Perl EMANE::Event::Location 156
Perl EMANE::Event::Pathloss 158
Perl EMANE::Event::CommEffect 161
Python emaneeventservice 164
Python emaneeventlocation 166
Python emaneeventservice 168
Python emaneeventservice 170

C emaneEventService

```
NAME
       emaneEventService  -  Allows  publication  and  subscription  of  EMANE
       events.  EMANE events are used to transmit emulation data  in  realtime
       to EMANE NEM components.

SYNOPSIS
       #include <libemaneeventservice/emaneeventservice.h>

       int emaneEventServiceInitialize(emaneEventService * pService);

       int emaneEventServiceDestroy(emaneEventService * pService);

       int emaneEventServiceSubscribe(emaneEventService * pService,
              emaneEventId id,
              emaneEventServiceHandler handler,
              void * arg);

       int emaneEventServiceUnsubscribe(emaneEventService * pService,
              emaneEventId id,
              void ** p);

       int emaneEventServicePublish(emaneEventService * pService,
             emaneEventId id,
             emanePlatformId platform,
             emaneNEMId nem,
             emaneComponentId component,
             const void * buf,
             size_t count);

       int emaneEventServiceDestroy(emaneEventService * pService);

       int emaneEventServiceLoop(emaneEventService * pService);

       int emaneEventServiceLoopWithDefault(emaneEventService * pService,
             emaneEventServiceHandler handler,
             void * arg);

       int emaneEventServiceNextEvent(emaneEventService * pService,
             emaneEventId * pEvent,
             emanePlatformId * pPlatform,
             emaneNEMId * pNEM,
             emaneComponentId * pComponent,
             void * pBuf,
             size_t length);

       int emaneEventServiceBreakloop(emaneEventService * pService);

       const char * emaneEventServiceError(emaneEventService * pService);


DESCRIPTION
       The   emaneEventServiceInitialize()   function   initializes   an  ema-
       neEventService instance.

       The emaneEventServiceDestroy() function destroys  an  emaneEventService
       instance.

       The  emaneEventServiceSubscribe()  function  subscribes  to EMANE event
       emaneEventId.  The callback function handler will be invoked  with  the
       arg  parameter.   If  no parameter is required use NULL. Callbacks that
       return non zero values will cause the emaneEventServiceLoop() and  ema-
       neEventServiceLoopWithDefault() loops to exit.

       The emaneEventServiceUnsubscribe() function unsubscribes the emaneEven-
       tId event from callback processing.  If p is not NULL is will be set to
       arg from the emaneEventServiceSubscribe call.

       The emaneEventServicePublish() function publishes event emaneEventId to
       a specified platform nem component.  Use EMANE_PLATFORMID_ANY to target
```

```
       all  platforms  in  an EMANE deployment.  Use EMANE_NEMID_ANY to target
       all NEMs in the specified platform.  Use EMANE_COMPONENTID_ANY to  tar-
       get  all  components  of  the  specified  NEM.  EMANE_COMPONENTID_MACI,
       EMANE_COMPONENTID_PHYI, and EMANE_COMPONENTID_SHIMI  target  MAC,  PHY,
       and SHIM components, respectively.

       The   emaneEventServiceLoop()   and  emaneEventServiceLoopWithDefault()
       functions process incoming events,  calling  the  appropriate  handlers
       when  subscribed  events are received. The emaneEventServiceLoopWithDe-
       fault() function will invoke a default callback handler  with  argument
       arg when an unsubscribe event is received.  The emaneEventServiceBreak-
       loop() function can be used to exit from processing. A default callback
       that  returns  a  non zero values will cause the emaneEventServiceLoop-
       WithDefault() loop to exit.

       The emaneEventServiceNextEvent() function blocks waiting to return  the
       next  received  event.  pEvent  will  be  set to the received event id.
       pPlatform will be set to the destination platform id.  pNEM will set to
       the destination NEM id.  pComponent will be set to the destination com-
       ponent id. The event data will be stored in pBuf.  A maximum of  length
       bytes  will be stored. The emaneEventServiceBreakloop() function can be
       used to exit from processing.

       The emaneEventServiceBreakloop() function will cause the emaneEventSer-
       viceLoop(),   emaneEventServiceLoopWithDefault(),   and  emaneEventSer-
       viceNextEvent() functions to exit.

       The emaneEventServiceError function returns a string description of the
       current emaneEventService error.

RETURN VALUE
       On error, all functions return -1 except emaneEventServiceError() which
       will always return a valid string representing the error.

EXAMPLES
         #include <libemaneeventservice/emaneeventservice.h>
         #include <libemaneeventservice/emaneeventpathloss.h>

         int handlePathlossEvent(emaneEventId event,
                                 emanePlatformId platform,
                                 emaneNEMId nem,
                                 emaneComponentId component,
                                 const void * buf,
                                 size_t length,
                                 void * p);

         ...

         emaneEventService message_service;


         if(emaneEventServiceInitialize(&message_service) < 0)
           {
             fprintf(stderr,
                     "emaneEventServiceInitialize: %s\n",
                     emaneEventServiceError(&message_service));

             return EXIT_FAILURE;
           }

         if(emaneEventServiceSubscribe(&message_service,
                                       EMANE_EVENT_PATHLOSS,
                                       &handlePathlossEvent,
                                       &pathlossStat) < 0)
           {
             fprintf(stderr,
                     "emaneEventServiceSubscribe: %s\n",
                     emaneEventServiceError(&message_service));

             return EXIT_FAILURE;
           }

```

```

        if(emaneEventServiceLoop(&message_service) < 0)
          {
            fprintf(stderr,
                    "emaneEventServiceLoop: %s\n",
                    emaneEventServiceError(&message_service));

            return EXIT_FAILURE;
          }

        if(emaneEventServiceUnsubscribe(&message_service,
                                        EMANE_EVENT_PATHLOSS,0) < 0)
          {
            fprintf(stderr,
                    "emaneEventServiceUnsubscribe: %s [%hu]\n",
                    emaneEventServiceError(&message_service),
                    EMANE_EVENT_PATHLOSS);

            return EXIT_FAILURE;
          }


        if(emaneEventServiceDestroy(&message_service) < 0)
          {
            fprintf(stderr,
                    "emaneEventServiceDestroy: %s\n",
                    emaneEventServiceError(&message_service));

            return EXIT_FAILURE;
          }

        ...

        if(emaneEventServicePublish(&eventService,
                                    EMANE_EVENT_PATHLOSS,
                                    EMANE_PLATFORMID_ANY,
                                    1,
                                    EMANE_COMPONENTID_PHYI,
                                    emaneEventPathlossExport(&ePathloss),
                                    emaneEventPathlossExportSizeBytes(&ePathloss)) <
0)
          {
            fprintf(stderr,
                    "emaneEventServicePublish: %s\n",
                    emaneEventServiceError(&eventService));

            return EXIT_FAILURE;
          }

        ...


NOTES
       Configuration for the Event Service is is searched for in  three  loca-
       tions prior to using default configuration.  In the following order:

        * If  the  environment  variable  LIBEMANEEVENTSERVICECONFIG exists it
          will be used. This variable may be set to a configuration file name.


        * If $HOME/.libemaneeventservice.xml  exists it will be used.


        * If /etc/libemaneeventservice.xml exists it will be used.


        * 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.


CONFIGURATION FILE FORMAT
```

```
<?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>
```

C emaneEventLocation

```
NAME
       emaneEventLocation  -  Creates and parses EMANE Location events.  Loca-
       tion events contain the GPS lcoation of all NEMs contained in the EMANE
       deployment.

SYNOPSIS
       #include <libemaneeventservice/emaneeventlocation.h>

       int emaneEventLocationInitialize(emaneEventLocation * pEvent,
              int iNodeCount);

       int emaneEventLocationInitializeFromImport(emaneEventLocation * pEvent,
              const void * buf,
              size_t length);

       int emaneEventLocationDestroy(emaneEventLocation * pEvent);

       int emaneEventLocationSetLocation(emaneEventLocation * pEvent,
              int iIndex,
              u_int16_t u16NEMId,
              double dLatitude,
              double Longitude,
              int32_t i32AltitudeMeters);

       int emaneEventLocationGetLocation(emaneEventLocation * pEvent,
              int iIndex,
              u_int16_t * pu16NEMId,
              double * pdLatitude,
              double * dLongitude,
              int32_t * pi32AltitudeMeters);

       int emaneEventLocationNodeCount(emaneEventLocation * pEvent,
              int * piNodeCount);

       int emaneEventLocationExportSizeBytes(emaneEventLocation * pEvent);

       const void * emaneEventLocationExport(emaneEventLocation * pEvent);

       const char * emaneEventLocationError(emaneEventLocation * pEvent);


DESCRIPTION
       The  emaneEventLocationInitialize()  and  emaneEventLocationInitialize-
       FromImport()  functions initialize an emaneEventLocation instance.  The
       emaneEventLocationInitialize() function creates a  new  location  event
       with iEntryCount entries.

       The emaneEventLocationDestroy() function destroys an emaneEventLocation
       instance.

       The emaneEventLocationSetLocation() function sets the location informa-
       tion  for  a  NEM  located at the iIndex position of the location entry
       data.  The location information contains the NEM id u16NEMId, the lati-
       tude in degrees dLatitude, the longitude dLongitude in degrees, and the
       altitude in meters i32AltitudeMeters.

       The emaneEventLocationGetLocation() function gets the location informa-
       tion  for  a  NEM  located at the iIndex position of the location entry
       data.   The  location  information  retreived  contains  the   NEM   id
       pu16NEMId,  the  latitude in degrees pdLatitude, the longitude pdLongi-
       tude in degrees, and the altitude in meters pi32AltitudeMeters.

       The emaneEventLocationNodeCount() function gets the total number of NEM
       entries contained in the location message and stores that value in piN-
       odeCount.

       The emaneEventLocationExport() function returns the location event data
       in a format suitable for publication.

```

```
       The  emaneEventLocationExportSizeBytes  returns  the length in bytes of
       the data returned by emaneEventLocationExport().

       The emaneEventLocationError function returns a  string  description  of
       the current emaneEventLocation error.

RETURN VALUE
       On error, all functions return -1 except emaneEventServiceError() which
       will always return a valid string and emaneEventLocationExport()  which
       will return NULL.

EXAMPLES
         #include <libemaneeventservice/emaneeventservice.h>
         #include <libemaneeventservice/emaneeventlocation.h>


         ...

         if(emaneEventServiceSubscribe(&message_service,
                                       EMANE_EVENT_LOCATION,
                                       &handleLocationEvent,
                                       &locationStat) < 0)
           {
             fprintf(stderr,
                     "emaneEventServiceSubscribe: %s\n",
                     emaneEventServiceError(&message_service));

             return EXIT_FAILURE;
           }

          ...

         for(i = 0; i < iNodeCount; ++i)
            {
              int nem = i+1;

              if(emaneEventLocationSetLocation(&eLocation,
                                               i,
                                               nem,
                                               40.76503,
                                               -73.97499,
                                               i) < 0)
                   {
                     fprintf(stderr,
                             "emaneEventLocationSetLocation: %s\n",
                             emaneEventLocationError(&eLocation));

                     return EXIT_FAILURE;
                   }
             }


         if(emaneEventServicePublish(&eventService,
                                     EMANE_EVENT_LOCATION,
                                     EMANE_PLATFORMID_ANY,
                                     EMANE_NEMID_ANY,
                                     EMANE_COMPONENTID_ANY,
                                     emaneEventLocationExport(&eLocation),
                                     emaneEventLocationExportSizeBytes(&eLocation)) <
0)
           {
              fprintf(stderr,
                     "emaneEventServicePublish: %s\n",
                      emaneEventServiceError(&eventService));
                      return EXIT_FAILURE;
           }

         ...

         int handleLocationEvent(emaneEventId event,
                                 emanePlatformId platform,
                                 emaneNEMId nem,
```

```
                                emaneComponentId component,
                                const void * buf,
                                size_t length,
                                void * p)
        {
          emaneEventLocation eLocation;
          int iNodeCount = 0;
          emaneNEMId nemId = 0;
          double dLatitudeDegrees   = 0;
          double dLongitudeDegrees  = 0;
          int32_t i32AltitudeMeters = 0;
          int i = 0;

          EventStatistic * pStat = (EventStatistic *) p;

          if(emaneEventLocationInitializeFromImport(&eLocation,
                                                    buf,
                                                    length) < 0)
            {
              fprintf(stderr,
                      "emaneEventLocationInitializeFromImport: %s\n",
                      emaneEventLocationError(&eLocation));

              return 1;
            }

          if(emaneEventLocationNodeCount(&eLocation,&iNodeCount) < 0)
            {
              fprintf(stderr,
                      "emaneEventLocationNodeCount: %s\n",
                      emaneEventLocationError(&eLocation));

              return 1;
            }

          for(i = 0; i < iNodeCount; ++i)
            {
              if(emaneEventLocationGetLocation(&eLocation,
                                               i,
                                               &nemId,
                                               &dLatitudeDegrees,
                                               &dLongitudeDegrees,
                                               &i32AltitudeMeters) < 0)
                {
                  fprintf(stderr,
                          "emaneEventLocationNodeCount: %s\n",
                          emaneEventLocationError(&eLocation));

                  return 1;
                }

              if(!iQuiet)
                {
                  fprintf(stdout,"node: %d [%hu] lon: %lf lat: %lf alt: %d\n",
                          i,
                          nemId,
                          dLongitudeDegrees,
                          dLatitudeDegrees,
                          i32AltitudeMeters);
                }
            }

          if(emaneEventLocationDestroy(&eLocation) < 0)
            {
              fprintf(stderr,
                      "emaneEventLocationDestroy: %s\n",
                      emaneEventLocationError(&eLocation));

              return 1;
            }

          ++pStat->ulCount;
```

```

            return 0;
          }

          ...

```

C emaneEventPathloss

```
NAME
       emaneEventPathloss   -   Creates  and  parses  EMANE  Pathloss  events.
       Pathloss events contain RF Pathloss in dB between a single NEM and  all
       the other NEMs contained in the EMANE deployment.

SYNOPSIS
       #include <libemaneeventservice/emaneeventpathloss.h>

       int emaneEventPathlossInitialize(emaneEventPathloss * pEvent,
              int iNodeCount);

       int emaneEventPathlossInitializeFromImport(emaneEventPathloss * pEvent,
              const void * buf,
              size_t length);

       int emaneEventPathlossDestroy(emaneEventPathloss * pEvent);

       int emaneEventPathlossSetPathloss(emaneEventPathloss * pEvent,
              int iIndex,
              u_int16_t u16TxNEMId,
              float fPathloss,
              float fRevPathloss);

       int emaneEventPathlossGetPathloss(emaneEventPathloss * pEvent,
              int iIndex,
              u_int16_t * pu16TxNEMId,
              float * pfPathloss,
              float * pfRevPathloss);

       int emaneEventPathlossNodeCount(emaneEventPathloss * pEvent,
              int * piNodeCount);

       int emaneEventPathlossExportSizeBytes(emaneEventPathloss * pEvent);

       const void * emaneEventPathlossExport(emaneEventPathloss * pEvent);

       const char * emaneEventPathlossError(emaneEventPathloss * pEvent);


DESCRIPTION
       The  emaneEventPathlossInitialize()  and  emaneEventPathlossInitialize-
       FromImport()  functions initialize an emaneEventPathloss instance.  The
       emaneEventPathlossInitialize() function creates a  new  pathloss  event
       with iEntryCount entries.  The emaneEventPathlossInitializeFromImport()
       creates a pathloss event from a pathloss  data  buffer  buf  of  length
       bytes.

       The emaneEventPathlossDestroy() function destroys an emaneEventPathloss
       instance.

       The emaneEventPathlossSetPathloss() function sets the pathloss informa-
       tion  for  the  transmitting  NEM located at the iIndex position of the
       pathloss entry data.  The pathloss information contains  the  transmit-
       ting  NEM  id u16TxNEMId, the pathloss from Tx NEM to Rx NEM fPathloss,
       and the pathloss from Rx NEM to Tx NEM fRevPathloss.

       The emaneEventPathlossGetPathloss() function gets the pathloss informa-
       tion  for  the  transmitting  NEM located at the iIndex position of the
       pathloss entry data.  The pathloss information retrieved  contains  the
       transmitting  NEM  id  pu16TxNEMId,  the pathloss from Tx NEM to Rx NEM
       pfPathloss, and the pathloss from Rx NEM to Tx NEM pfRevPathloss.

       The emaneEventPathlossNodeCount() function gets the total number of NEM
       entries contained in the pathloss message and stores that value in piN-
       odeCount.

       The emaneEventPathlossExport() function returns the pathloss event data
       in a format suitable for publication.

```

```
       The  emaneEventPathlossExportSizeBytes  returns  the length in bytes of
       the data returned by emaneEventPathlossExport().

       The emaneEventPathlossError function returns a  string  description  of
       the current emaneEventPathloss error.

RETURN VALUE
       On error, all functions return -1 except emaneEventServiceError() which
       will always return a valid string and emaneEventPathlossExport()  which
       will return NULL.

EXAMPLES
         #include <libemaneeventservice/emaneeventservice.h>
         #include <libemaneeventservice/emaneeventpathloss.h>


         ...

         if(emaneEventServiceSubscribe(&message_service,
                                       EMANE_EVENT_PATHLOSS,
                                       &handlePathlossEvent,
                                       &pathlossStat) < 0)
           {
             fprintf(stderr,
                     "emaneEventServiceSubscribe: %s\n",
                     emaneEventServiceError(&message_service));

             return EXIT_FAILURE;
           }

          ...

          for(i = 0; i < iNodeCount; ++i)
            {
              int rx = i+1;

              int index = 0;

              for(j = 0; j< iNodeCount; ++j)
                {
                  int tx = j+1;

                  if(emaneEventPathlossSetPathloss(&ePathloss,
                                                   index,
                                                   tx,
                                                   10 + j + .2,
                                                   10 + j + .6) < 0)
                    {
                      fprintf(stderr,
                              "emaneEventPathlossSetPathloss: %s\n",
                               emaneEventPathlossError(&ePathloss));

                      return EXIT_FAILURE;
                    }

                  ++index;
                 }

              if(emaneEventServicePublish(&eventService,
                                          EMANE_EVENT_PATHLOSS,
                                          EMANE_PLATFORMID_ANY,
                                          rx,
                                          EMANE_COMPONENTID_PHYI,
                                          emaneEventPathlossExport(&ePathloss),

emaneEventPathlossExportSizeBytes(&ePathloss)) < 0)
                 {
                   fprintf(stderr,
                           "emaneEventServicePublish: %s\n",
                           emaneEventServiceError(&eventService));

                   return EXIT_FAILURE;
```

```
141                 }
142
143             }
144
145         ...
146
147         int handlePathlossEvent(emaneEventId event,
148                                 emanePlatformId platform,
149                                 emaneNEMId nem,
150                                 emaneComponentId component,
151                                 const void * buf,
152                                 size_t length,
153                                 void * p)
154         {
155           emaneEventPathloss ePathloss;
156           int iNodeCount = 0;
157           int i = 0;
158           u_int16_t u16TxNEMId = 0;
159           float  fPathloss     = 0;
160           float  fRevPathloss  = 0;
161           EventStatistic * pStat = (EventStatistic *) p;
162
163
164           if(emaneEventPathlossInitializeFromImport(&ePathloss,
165                                                     buf,
166                                                     length) < 0)
167             {
168               fprintf(stderr,
169                       "emaneEventPathlossInitializeFromImport: %s\n",
170                       emaneEventPathlossError(&ePathloss));
171
172               return 1;
173             }
174
175           if(emaneEventPathlossNodeCount(&ePathloss,&iNodeCount) < 0)
176            {
177               fprintf(stderr,
178                       "emaneEventPathlossNodeCount: %s\n",
179                       emaneEventPathlossError(&ePathloss));
180
181               return 1;
182            }
183
184           for(i = 0; i < iNodeCount; ++i)
185            {
186               if(emaneEventPathlossGetPathloss(&ePathloss,
187                                                i,
188                                                &u16TxNEMId,
189                                                &fPathloss,
190                                                &fRevPathloss) < 0)
191                 {
192                   fprintf(stderr,
193                           "emaneEventPathlossGetPathloss: %s\n",
194                           emaneEventPathlossError(&ePathloss));
195
196                   return 1;
197                 }
198
199               if(!iQuiet)
200                 {
201                   printf("%d %f %f\n",
202                           u16TxNEMId,
203                           fPathloss,
204                           fRevPathloss);
205                 }
206             }
207
208
209           if(emaneEventPathlossDestroy(&ePathloss) < 0)
210             {
211               fprintf(stderr,
212                       "emaneEventPathlossDestroy: %s\n",
```

```
                        emaneEventPathlossError(&ePathloss));

                return 1;
              }

            return 0;
          }

          ...

```

C emaneEventCommEffect

```
NAME
       emaneEventCommEffect  -  Creates  and  parses  EMANE CommEffect events.
       CommEffect events contain communication effects between  a  single  NEM
       and all the other NEMs contained in the EMANE deployment.

SYNOPSIS
       #include <libemaneeventservice/emaneeventcommeffect.h>

       int emaneEventCommEffectInitialize(emaneEventCommEffect * pEvent,
              int iNodeCount);

       int   emaneEventCommEffectInitializeFromImport(emaneEventCommEffect   *
       pEvent,
              const void * buf,
              size_t length);

       int emaneEventCommEffectDestroy(emaneEventCommEffect * pEvent);

       int emaneEventCommEffectSetCommEffect(emaneEventCommEffect * pEvent,
              int iIndex,
              u_int16_t u16TxNEMId,
              u_int32_t u32LatencySeconds,
              u_int32_t u32LatencyMicroSeconds,
              u_int32_t u32JitterSeconds,
              u_int32_t u32JitterMicroSeconds,
              u_int16_t u16ProbabilityLoss,
              u_int16_t u16ProbabilityDuplicate,
              u_int64_t u64UnicastBitRate,
              u_int64_t u64BroadcastBitRate);

       int emaneEventCommEffectGetCommEffect(emaneEventCommEffect * pEvent,
              int iIndex,
              u_int16_t * pu16TxNEMId,
              u_int32_t * pu32LatencySeconds,
              u_int32_t * pu32LatencyMicroSeconds,
              u_int32_t * pu32JitterSeconds,
              u_int32_t * pu32JitterMicroSeconds,
              u_int16_t * pu16ProbabilityLoss,
              u_int16_t * pu16ProbabilityDuplicate,
              u_int64_t * pu64UnicastBitRate,
              u_int64_t * pu64BroadcastBitRate);

       int emaneEventCommEffectNodeCount(emaneEventCommEffect * pEvent,
              int * piNodeCount);

       int emaneEventCommEffectExportSizeBytes(emaneEventCommEffect * pEvent);

       const void * emaneEventCommEffectExport(emaneEventCommEffect * pEvent);

       const char * emaneEventCommEffectError(emaneEventCommEffect * pEvent);


DESCRIPTION
       The emaneEventCommEffectInitialize()  and  emaneEventCommEffectInitial-
       izeFromImport()  functions initialize an emaneEventCommEffect instance.
       The emaneEventCommEffectInitialize() function creates a new  commeffect
       event  with  iEntryCount  entries.  The emaneEventCommEffectInitialize-
       FromImport() creates a commeffect event from a commeffect  data  buffer
       buf of length bytes.

       The  emaneEventCommEffectDestroy() function destroys an emaneEventComm-
       Effect instance.

       The emaneEventCommEffectSetCommEffect() function  sets  the  commeffect
       information  for the transmitting NEM located at the iIndex position of
       the commeffect entry data.  The  commeffect  information  contains  the
       transmitting  NEM  id  u16TxNEMId,  the  whole  seconds  portion of the
       latency  u32LatencySeconds,  the  whole  microseconds  portion  of  the
       latency u32LatencyMicroSeconds, the whole seconds portion of the jitter
```

```
       u32JitterSeconds, the whole microseconds portion of the jitter  u32Jit-
       terMicroSeconds,  the probablity of loss u16ProbabilityLoss, the proba-
       bility of duplication u16ProbabilityDuplicate,  the  unicast  bit  rate
       (bps) u64UnicastBitRate, and the broadcast bit rate (bps) u64Broadcast-
       BitRate.

       The emaneEventCommEffectGetCommEffect() function  gets  the  commeffect
       information  for the transmitting NEM located at the iIndex position of
       the commeffect entry data.  The commeffect information  retrieved  con-
       tains the transmitting NEM id pu16TxNEMId, the whole seconds portion of
       the latency pu32LatencySeconds, the whole microseconds portion  of  the
       latency  pu32LatencyMicroSeconds, the whole seconds portion of the jit-
       ter pu32JitterSeconds, the the whole microseconds portion of the jitter
       pu32JitterMicroSeconds, the probablity of loss pu16ProbabilityLoss, the
       probability of duplication pu16ProbabilityDuplicate,  the  unicast  bit
       rate  (bps)  pu64UnicastBitRate,  and  the  broadcast  bit  rate  (bps)
       pu64BroadcastBitRate.

       The emaneEventCommEffectNodeCount() function gets the total  number  of
       NEM  entries  contained in the commeffect message and stores that value
       in piNodeCount.

       The emaneEventCommEffectExport() function returns the commeffect  event
       data in a format suitable for publication.

       The  emaneEventCommEffectExportSizeBytes returns the length in bytes of
       the data returned by emaneEventCommEffectExport().

       The emaneEventCommEffectError function returns a string description  of
       the current emaneEventCommEffect error.

RETURN VALUE
       On error, all functions return -1 except emaneEventServiceError() which
       will always return  a  valid  string  and  emaneEventCommEffectExport()
       which will return NULL.

EXAMPLES
         #include <libemaneeventservice/emaneeventservice.h>
         #include <libemaneeventservice/emaneeventcommeffect.h>


         ...

         if(emaneEventServiceSubscribe(&message_service,
                                       EMANE_EVENT_COMMEFFECT,
                                       &handleCommEffectEvent,
                                       &commeffectStat) < 0)
           {
             fprintf(stderr,
                     "emaneEventServiceSubscribe: %s\n",
                     emaneEventServiceError(&message_service));

             return EXIT_FAILURE;
           }

          ...

          for(i = 0; i < iNodeCount; ++i)
            {
              int rx = i+1;

              int index = 0;

              for(j = 0; j< iNodeCount; ++j)
                {
                  int tx = j+1;

                  if(emaneEventCommEffectSetCommEffect(&eCommEffect,
                                                       index,
                                                       tx,
                                                       0,
                                                       50000,
```

```
                                                        0,
                                                        0,
                                                        0,
                                                        1,
                                                        100000,
                                                        100000) < 0)
                        {
                          fprintf(stderr,
                                  "emaneEventCommEffectSetCommEffect: %s\n",
                                  emaneEventCommEffectError(&eCommEffect));

                          return EXIT_FAILURE;
                        }

                    ++index;
                    }

                if(emaneEventServicePublish(&eventService,
                                            EMANE_EVENT_COMMEFFECT,
                                            EMANE_PLATFORMID_ANY,
                                            rx,
                                            EMANE_COMPONENTID_PHYI,
                                            emaneEventCommEffectExport(&eCommEffect),

emaneEventCommEffectExportSizeBytes(&eCommEffect)) < 0)
                  {
                    fprintf(stderr,
                            "emaneEventServicePublish: %s\n",
                            emaneEventServiceError(&eventService));

                    return EXIT_FAILURE;
                  }

              }

          ...

          int handleCommEffectEvent(emaneEventId event,
                                    emanePlatformId platform,
                                    emaneNEMId nem,
                                    emaneComponentId component,
                                    const void * buf,
                                    size_t length,
                                    void * p)
          {
            emaneEventCommEffect eCommEffect;
            int iNodeCount = 0;
            u_int16_t u16TxNEMId = 0;
            u_int32_t u32LatencySeconds = 0;
            u_int32_t u32LatencyMicroSeconds = 0;
            u_int32_t u32JitterSeconds = 0;
            u_int32_t u32JitterMicroSeconds = 0;
            u_int16_t u16ProbabilityLoss = 0;
            u_int16_t u16ProbabilityDuplicate = 0;
            u_int64_t u64UnicastBitRate = 0;
            u_int64_t u64BroadcastBitRate = 0;
            int i = 0;

            EventStatistic * pStat = (EventStatistic *) p;

            if(emaneEventCommEffectInitializeFromImport(&eCommEffect,
                                                        buf,
                                                        length) < 0)
              {
                fprintf(stderr,
                        "emaneEventCommEffectInitializeFromImport: %s\n",
                        emaneEventCommEffectError(&eCommEffect));

                return 1;
              }


```

```
213             if(emaneEventCommEffectNodeCount(&eCommEffect,&iNodeCount) < 0)
214               {
215                 fprintf(stderr,
216                         "emaneEventCommEffectNodeCount: %s\n",
217                         emaneEventCommEffectError(&eCommEffect));
218
219                 return 1;
220               }
221
222             for(i = 0; i < iNodeCount; ++i)
223               {
224                 if(emaneEventCommEffectGetCommEffect(&eCommEffect,
225                                                      i,
226                                                      &u16TxNEMId,
227                                                      &u32LatencySeconds,
228                                                      &u32LatencyMicroSeconds,
229                                                      &u32JitterSeconds,
230                                                      &u32JitterMicroSeconds,
231                                                      &u16ProbabilityLoss,
232                                                      &u16ProbabilityDuplicate,
233                                                      &u64UnicastBitRate,
234                                                      &u64BroadcastBitRate) < 0)
235                   {
236                     fprintf(stderr,
237                             "emaneEventCommEffectGetCommeffect: %s\n",
238                             emaneEventCommEffectError(&eCommEffect));
239
240                     return 1;
241                   }
242
243                 if(u16TxNEMId != 0 && u16RxNEMId != 0)
244                   {
245                     if(!iQuiet)
246                       {
247                         printf("%d %d.%06d %d.%06d %-3hu %-3hu\n",
248                                u16TxNEMId,
249                                u32LatencySeconds,
250                                u32LatencyMicroSeconds,
251                                u32JitterSeconds,
252                                u32JitterMicroSeconds,
253                                u16ProbabilityLoss,
254                                u16ProbabilityDuplicate);
255                       }
256                   }
257
258               }
259
260             if(emaneEventCommEffectDestroy(&eCommEffect) < 0)
261               {
262                 fprintf(stderr,
263                         "emaneEventCommEffectDestroy: %s\n",
264                         emaneEventCommEffectError(&eCommEffect));
265
266                 return 1;
267               }
268
269             ++pStat->ulCount;
270             return 0;
271           }
272
273           ...
274
```

Perl EMANE::EventService

```
NAME
    EMANE::EventService - Perl extension for libemaneeventservice

SYNOPSIS
      use EMANE::EventService;
      use EMANE::Event::Location;

      my $event = new EMANE::Event::Location(entryCount => 70);

      my $eventservice = new EMANE::EventService();

      ...

      $eventservice->publish(EMANE_EVENT_LOCATION,
                             EMANE_PLATFORMID_ANY,
                             EMANE_NEMID_ANY,
                             EMANE_COMPONENTID_ANY,
                             $event->export());

      ...

      $eventservice->subscribe(EMANE_EVENT_LOCATION,\&handleLocation);

      $eventservice->loop();

      $eventservice->unsubscribe(EMANE_EVENT_LOCATION);

      ...

DESCRIPTION
    EMANE::EventService - EMANE EventService allows publication and
    subscription of EMANE events. EMANE events are used to transmit
    emulation data in realtime to EMANE NEM components.

  new()
    Creates an Event Service.

  publish()
    Publish event data.

      $eventservice->publish($eventid,
                             $platformid,
                             $nemid,
                             $componentid,
                             $data);

    eventid
        Event id of event data being published.

    platformid
        Platform id of the target platform. Use EMANE_PLATFORMID_ANY for all
        platforms in the deployment.

    nemid
        NEM id of target module. Use EMANE_NEMID_ANY for all NEMs contained
        in the target platform.

    componentid
        Component id of the target NEM component. Use EMANE_COMPONENTID_ANY
        for all components contained in the target NEM.

    data
        Event data to publish.

  subscribe()
    Subscribe to receive event data for a specific event id. Callbacks are
    called only while using loop().

      $eventservice->subscribe(eventid,\&handleEventCallback);
```

```

    eventid
        Event id of the event to subscribe.

    handleEventCallback
        Reference to a callback subroutine which is invoke when an event of
        type eventid is received.

  unsubscribe()
    Unsubscribe from receiving event data for a specific event id.

      $eventservice->unsubscribe(eventid);

    eventid
        Event id of the event to unsubscribe.

  loop()
    Receive incoming EMANE events and invoke respective callbacks when a
    subscribed event is received or a default callback if specified. Use
    breakloop() to end processing.

      $eventservice->loop();

      $eventservice->loop(default=>\&default);

    default
        Reference to a callback subroutine which is invoke when an
        unsubscribed event is received.

  nextEvent()
    Block waiting for the next event to be received. Returns a list
    containing the event id, platform id, NEM id, component id, and data for
    the received event. Use breakloop() to end processing.

      ($event,$platform,$nem,$component,$data) = $eventService->nextEvent();

   breakloop()
    End the event processing loop() or cause nextEvent() to return.

  Exportable constants
      EMANE_COMPONENTID_ANY
      EMANE_COMPONENTID_MACI
      EMANE_COMPONENTID_PHYI
      EMANE_COMPONENTID_SHIMI

      EMANE_NEMID_ANY

      EMANE_PLATFORMID_ANY

CONFIGURATION
    EMANE::EventService is a perl binding of libemaneeventservice.
    Configuration for the event service is is searched for in three
    locations prior to using default configuration. In the following order:

    1. Environment variable LIBEMANEEVENTSERVICECONFIG
        If LIBEMANEEVENTSERVICECONFIG exists it will be used. This variable
        may be set to a configuration file name.

    2. $HOME/.libemaneeventservice.xml
        If .libemaneeventservice.xml exists it will be used.

    3. /etc/libemaneeventservice.xml
        If /etc/libemaneeventservice.xml exists it will be used.

    4. Library default values
        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.

CONFIGURATION FORMAT
      <?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>

```

Perl EMANE::Event::Location

```
NAME
    EMANE::Event::Location - Perl extension for libemaneeventlocation

SYNOPSIS
      use EMANE::EventService;
      use EMANE::Event::Location;

      $eventservice = new EMANE::EventService();

      $eventservice->subscribe(EMANE_EVENT_LOCATION,sub
                               {
                                 ($event,$platform,$nem,$comp,$data) = @_;

                                  $location =
                                    new EMANE::Event::Location(import=>$data);

                                  %entries = $location->entries();

                                 for $entry (sort {$a <=> $b} keys %entries)
                                 {
                                    $ref = \%{$entries{$entry}};

                                    printf("%-3d %-13.7f %-13.7f %6d\n",
                                           $ref->{nemId},
                                           $ref->{latitude},
                                           $ref->{longitude},
                                           $ref->{altitude});
                                 }
                               }
                              );

      ...


      $event = new EMANE::Event::Location(entryCount => 70);

      for($nem = 0; $nem < 70; ++$nem)
      {
        $event->set(0,$nem+1,39.236581,-76.827025,0);
      }

      $eventservice->publish(EMANE_EVENT_LOCATION,
                             EMANE_PLATFORMID_ANY,
                             EMANE_NEMID_ANY,
                             EMANE_COMPONENTID_ANY,
                             $event->export());
      ...

DESCRIPTION
    EMANE::Event::Location creates and parses EMANE Location events.
    Location events contain the GPS lcoation of NEMs entries contained in
    the event.

  new()
    Creates a location event message data object.

      $event = new EMANE::Event::Location(entryCount => 70);

      $event = new EMANE::Event::Location(import=>$data);

    entryCount
        Number of NEMs entries contained in the event. Use to construct a
        new location event message. Not valid when used with import.

    import
        Binary event data received from the event service. Use to construct
        a location event message from received data. Not valid when used
        with entryCount.

```

```
  set()
    Set location information for a specific NEM in the location event data.

      $event->set($index,
                  $NEMId,
                  $latitude,
                  $longitude,
                  $altitude);

    index
        Entry index.

    NEMId
        NEM id.

    latitude
        Latitude in degrees.

    longitude
        Longitude in degrees.

    altitude
        Altitude in meters.

  export()
    Returns a scalar containing the location event data in a format suitable
    for publication.

      $eventservice->publish(EMANE_EVENT_LOCATION,
                             EMANE_PLATFORMID_ANY,
                             EMANE_NEMID_ANY,
                             EMANE_COMPOMENTID_ANY,
                             $event->export());

  entries()
    Returns a hash of hash references. The returned hash key values
    correspond to NEM ids and the hash reference values contain the location
    information.

    The following example shows a dump of the resulting enties hash for a
    location event containg entries for 3 NEMs:

      %entries = $location->entries();
      print Dumper(\%entries);

    Produces:
     $VAR1 = {
              '1' => { 'longitude' => '-73.97499',
                       'latitude'  => '40.76503',
                       'nemId' => 1,
                       'altitude' => 0},
              '3' => { 'longitude' => '-73.97499',
                       'latitude' => '40.76503',
                       'nemId' => 3,
                       'altitude' => 2},
              '2' => { 'longitude' => '-73.97499',
                       'latitude' => '40.76503',
                       'nemId' => 2, 'altitude' => 1}
             };

  Exportable constants
      EMANE_EVENT_LOCATION

SEE ALSO
    EMANE::EventService

```

Perl EMANE::Event::Pathloss

```
NAME
    EMANE::Event::Pathloss - Perl extension for libemaneeventpathloss

SYNOPSIS
      use EMANE::EventService;
      use EMANE::Event::Pathloss;

      $eventservice = new EMANE::EventService();

      $eventservice->subscribe(EMANE_EVENT_PATHLOSS,sub
                               {
                                 ($event,$platform,$nem,$comp,$data) = @_;

                                 $pathloss =
                                   new EMANE::Event::Pathloss(import=>$data);

                                 %entries = $pathloss->entries();

                                 for $entry (sort {$a <=> $b} keys %entries)
                                 {
                                    $ref = \%{$entries{$entry}};

                                   printf("%d %f %f\n",
                                          $ref->{txNEMId},
                                          $ref->{pathloss},
                                          $ref->{revPathloss});
                                 }
                               }
                              );

      ...


      $nemcount = 70;

      $event = new EMANE::Event::Pathloss(entryCount => $nemcount - 1);


      for($rx = 0; $rx < $nemcount - 1; ++$rx)
      {
        my $index = 0;

        for($tx = 0; $tx < $nemcount - 1; ++$tx)
        {
          next if $rx == $tx;

          $event->set($index++,$tx+1,80.1,90.5);
        }

        $eventservice->publish(EMANE_EVENT_PATHLOSS,
                               EMANE_PLATFORMID_ANY,
                               $rx + 1,
                               EMANE_COMPONENTID_ANY,
                               $event->export());
      }
      ...

DESCRIPTION
    EMANE::Event::Pathloss creates and parses EMANE Pathloss events.
    Pathloss events contain RF Pathloss in dB between a the receiving NEM
    and the Tx NEM entries contained in the event.

  new()
    Creates a pathloss event message data object.

      $event = new EMANE::Event::Pathloss(entryCount => 70);

      $event = new EMANE::Event::Pathloss(import=>$data);

```

```
    entryCount
        Number of NEMs pathloss entries contained in the event. Use to
        construct a new pathloss event message. Not valid when used with
        import.

    import
        Binary event data received from the event service. Use to construct
        a pathloss event message from received data. Not valid when used
        with entryCount.

  set()
    Set pathloss information for a specific NEM in the pathloss event data.

      $event->set($index,$txNEM,$pathloss,$reversePathloss);

    index
        Entry index.

    txNEM
        NEM id of the transmitting NEM.

    pathloss
        Pathloss from the txNEM to the rxNEM.

    reversePathloss
        Pathloss from rxNEM to the txNEM.

  export()
    Returns a scalar containing the pathloss event data in a format suitable
    for publication.

      $eventservice->publish(EMANE_EVENT_PATHLOSS,
                             EMANE_PLATFORMID_ANY,
                             EMANE_NEMID_ANY,
                             EMANE_COMPOMENTID_ANY,
                             $event->export());

  entries()
    Returns a hash of hash references. The returned hash key values
    correspond to transmit NEM ids and the hash reference values contain the
    pathloss information between the transmitter NEM and the receiver NEM.

    The following example shows a dump of the resulting enties hash for a 3
    NEM deployment:

      %entries = $pathloss->entries();
      print Dumper(\%entries);

    Produces:

      $VAR1 = {
                '2' => {
                          'txNEMId' => 2,
                          'pathloss' => '11.19921875',
                          'revPathloss' => '11.59765625'
                        },
                '1' => {
                          'txNEMId' => 1,
                          'pathloss' => '10.19921875',
                          'revPathloss' => '10.59765625'
                        }
              };

    Note:

    The entries() method returns N entries, where N is the number of
    pathloss entries contained in the event.

  Exportable constants
      EMANE_EVENT_PATHLOSS

SEE ALSO
```

```
    EMANE::EventService

```

Perl EMANE::Event::CommEffect

```
NAME
    EMANE::Event::CommEffect - Perl extension for libemaneeventcommeffect

SYNOPSIS
      use EMANE::EventService;
      use EMANE::Event::CommEffect;

      $eventservice = new EMANE::EventService();

      $eventservice->subscribe(EMANE_EVENT_COMMEFFECT,sub
                               {
                                 ($event,$platform,$nem,$comp,$data) = @_;

                                  $commeffect =
                                    new EMANE::Event::CommEffect(import=>$data);

                                  %entries = $commeffect->entries();

                                 for $entry (sort {$a <=> $b} keys %entries)
                                 {
                                    $ref = \%{$entries{$entry}};

                                    printf("%d %d.%06d %d.%06d %d %d %s %s\n",
                                           $ref->{txNEMId},
                                           $ref->{latencySeconds},
                                           $ref->{latencyMicroSeconds},
                                           $ref->{jitterSeconds},
                                           $ref->{jitterMicroSeconds},
                                           $ref->{probabilityLoss},
                                           $ref->{probabilityDuplicate},
                                           $ref->{unicastBitRate},
                                           $ref->{broadcastBitRate},
                                          );
                                 }
                               }
                              );

      ...

      $nemcount = 70;

      $event = new EMANE::Event::CommEffect(entryCount => $nemcount - 1);

      for($rx = 0; $rx < $nemcount - 1; ++$rx)
      {
        $index = 0;

        for($tx = 0; $tx < $nemcount - 1; ++$tx)
        {
          next if $rx == $tx;

          $event->set($index++,
                      $tx+1,
                      100,
                      300,
                      200,
                      400,
                      55,
                      99,
                      18446744073709551615,
                      65535);
        }

        $eventservice->publish(EMANE_EVENT_COMMEFFECT,
                               EMANE_PLATFORMID_ANY,
                               $rx + 1,
                               EMANE_COMPONENTID_ANY,
                               $event->export());
      }
```

```

      ...

DESCRIPTION
    EMANE::Event::CommEffect creates and parses EMANE CommEffect events.

  new()
    Creates a commeffect event message data object.

      $event = new EMANE::Event::CommEffect(entryCount => 70);

      $event = new EMANE::Event::CommEffect(import=>$data);

    entryCount
        Number of NEMs entries contained in the event. Use to construct a
        new commeffect event message. Not valid when used with import.

    import
        Binary event data received from the event service. Use to construct
        a commeffect event message from received data. Not valid when used
        with entryCount.

  set()
    Set commeffect information for a specific NEM in the commeffect event
    data.

      $event->set($index,
                  $txNEM,
                  $latencySeconds,
                  $latecyMicroSeconds,
                  $jitterSeconds,
                  $jitterMicroSeconds,
                  $probabilityLoss,
                  $probabilityDuplicate,
                  $unicastBitRate,
                  $broadcastBitRate);

    index
        Entry index.

    txNEM
        NEM id of the transmitting NEM.

    latencySeconds
        Whole seconds portion of the latency from txNEM to rxNEM.

    latecyMicroSeconds
        Whole microseconds portion of the latency from txNEM to rxNEM.

    jitterSeconds
        Whole seconds portion of the jitter from txNEM to rxNEM.

    jitterMicroSeconds
        Whole microseconds portion of the jitter from txNEM to rxNEM.

    probabilityLoss
        Probablity of loss. Integer value [0,100].

    probabilityDuplicate
        Probablity of duplication. Integer value [0,65535].

    unicastBitRate
        Unicast bit rate from txNEM to rxNEM.

    broadcastBitRate
        Broadcast bit rate from txNEM to rxNEM.

  export()
    Returns a scalar containing the commeffect event data in a format
    suitable for publication.

      $eventservice->publish(EMANE_EVENT_COMMEFFECT,
```

```
                              EMANE_PLATFORMID_ANY,
                              EMANE_NEMID_ANY,
                              EMANE_COMPOMENTID_ANY,
                              $event->export());

  entries()
    Returns a hash of hash references. The returned hash key values
    correspond to transmit NEM ids and the hash reference values contain the
    commeffect information between the transmitter NEM and the receiver NEM.

    The following example shows a dump of the resulting enties hash for an
    event containing 3 NEM entries:

      %entries = $commeffect->entries();
      print Dumper(\%entries);

    Produces:
      $VAR1 = {
                '1' => { 'latencySeconds' => 5,
                         'jitterMicroSeconds' => 300,
                         'probabilityLoss' => 13,
                         'jitterSeconds' => 6,
                         'txNEMId' => 1,
                         'probabilityDuplicate' => 0,
                         'latencyMicroSeconds' => 0,
                         'unicastBitRate' => '18446744073709551615',
                         'broadcastBitRate' => '65535', },
                '2' => { 'latencySeconds' => 5,
                         'jitterMicroSeconds' => 300,
                         'probabilityLoss' => 13,
                         'jitterSeconds' => 6,
                         'txNEMId' => 2,
                         'probabilityDuplicate' => 0,
                         'latencyMicroSeconds' => 0,
                         'unicastBitRate' => '18446744073709551615',
                         'broadcastBitRate' => '65535', }
              };

  Exportable constants
      EMANE_EVENT_COMMEFFECT

SEE ALSO
    EMANE::EventService

```

Python emaneeventservice

```
NAME
    emaneeventservice - EMANE Event Service allows publication and subscription of
EMANE events

DESCRIPTION
    Configuration for the event service is is searched for in three locations
    prior to using default configuration.  In the following order:

     1. If LIBEMANEEVENTSERVICECONFIG evnironment exists it will be used.
        This variable may 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

    Configuration format:
     <?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>

    Classes
     EventService

DATA
    COMPONENTID_ANY = 0
    COMPONENTID_MACI = 8
    COMPONENTID_PHYI = 2
    COMPONENTID_SHIMI = 32
    NEMID_ANY = 0
    PLATFORMID_ANY = 0
    error = 'emaneeventservice.error'
```

```
emaneeventservice.EventService = class EventService(object)
 |  EMANE EventService object
 |
 |  SYNOPSIS
 |
 |  import emaneeventservice
 |
 |  service = emaneeventservice.EventService()
 |
 |  def handleLocationEvent(event,platform,nem,component,data):
 |      print "event ", event, " length ",len(data), " bytes"
 |
 |  service.subscribe(204,handleLocationEvent)
 |
 |  (event,data) = service.nextEvent()
 |
 |  service.loop()
 |
 |  service.unsubscribe(204)
 |
 |  ...
 |
 |  service.publish(204,
 |                  emaneeventservice.PLATFORMID_ANY,
 |                  emaneeventservice.NEMID_ANY,
 |                  emaneeventservice.COMPONENTID_ANY,
 |                  data);
```

```
 28  |
 29  |  Methods defined here:
 30  |
 31  |  breakloop(...)
 32  |      breakloop()
 33  |
 34  |      End the event processing loop() or cause nextEvent() to return.
 35  |
 36  |  loop(...)
 37  |      loop()
 38  |
 39  |      loop(callback)
 40  |
 41  |      Receive incoming EMANE events and invoke respective callbacks
 42  |      when a subscribed event is received or a default callback if
 43  |      is specified.  Use breakloop() to end processing.
 44  |
 45  |      callback  - Subroutine to invoke when unsubscribed event is received.
 46  |
 47  |  nextEvent(...)
 48  |      nextEvent() -> (event,platform,nem,component,data)
 49  |
 50  |      Block waiting for the next event to be received.  Returns a
 51  |      tuple containing the event id, platform id, NEM id, component
 52  |      id, and data for the received event.
 53  |      Use breakloop() to end processing.
 54  |
 55  |  publish(...)
 56  |      publish(event,platform,nem,component,data)
 57  |
 58  |      Publish event data.
 59  |
 60  |      event     - Event id of event data being published.
 61  |
 62  |      platform  - Platform id of the target platform. Use
 63  |                  EMANE_PLATFORMID_ANY for all platforms in the deployment.
 64  |
 65  |      nem       - NEM id of target module.  Use EMANE_NEMID_ANY for all
 66  |                  NEMs contained in the target platform.
 67  |
 68  |      component - Component id of the target NEM component. Use
 69  |                  EMANE_COMPONENTID_ANY for all components contained in the
 70  |                  target NEM.
 71  |
 72  |      data      - Event data to publish.
 73  |
 74  |  subscribe(...)
 75  |      subscribe(event,callback)
 76  |
 77  |      Subscribe to receive event data for a specific event id.
 78  |      Callbacks are called only while using loop().
 79  |
 80  |      event     - Event id of event to subscribe
 81  |
 82  |      callback  - Subroutine to invoke when the event is received.
 83  |
 84  |  unsubscribe(...)
 85  |      unsubscribe(event)
 86  |
 87  |      Unsubscribe from receiving event data for a specific event.
 88  |
 89  |      event     - Event id of event to unsubscribe
 90  |
 91  |  ----------------------------------------------------------------------
 92  |  Data and other attributes defined here:
 93  |
 94  |  __new__ = <built-in method __new__ of type object at 0x7f6bde8e57a0>
 95  |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```

Python emaneeventlocation

```
NAME
    emaneeventlocation

DESCRIPTION
    EMANE Event Location creates and parses EMANE Location events.
    Location events contain the GPS lcoation of NEMs.

    CLASSES
     EventLocation

DATA
    EVENT_ID = 204
    error = 'emaneeventlocation.error'
```

```
emaneeventlocation.EventLocation = class EventLocation(object)
 |  EventLocation(entryCount)
 |  EventLocation(data)
 |
 |  EMANE Event Location objects can be created by specifying the
 |  number of event entries to create or from event data received
 |  from the EventService
 |
 |  entryCount - Number of entries to create.  Use to construct a new
 |               location event message. Not valid when used with
 |               data.
 |
 |  data       - Binary event data received from the event service.
 |               Use to construct a location event message from
 |               received data. Not valid when used with entryCount
 |
 |  SYNOPSIS
 |
 |  import emaneeventservice
 |
 |  service = emaneeventservice.EventService()
 |
 |  def handleLocationEvent(event, platform, nem, component, data):
 |      print "event ", event, " length ",len(data), " bytes"
 |      event = emaneeventlocation.EventLocation(data)
 |      entries  = event.entries()
 |      for e in entries.values():
 |          print e
 |
 |  service.subscribe(emaneeventlocation.EVENT_ID,
 |                    handleLocationEvent)
 |
 |  service.loop()
 |
 |  service.unsubscribe(emaneeventlocation.EVENT_ID)
 |
 |  ...
 |
 |  service.publish(emaneeventlocation.EVENT_ID,
 |                  emaneeventservice.PLATFORMID_ANY,
 |                  emaneeventservice.NEMID_ANY,
 |                  emaneeventservice.COMPONENTID_ANY,
 |                  data);
 |
 |  Methods defined here:
 |
 |  entries(...)
 |      entries() -> {NEMId,(NEMId,latitude,longitude,altitude)}
 |
 |      Returns a dictionary with NEMId keys and tuple values containing the
 |      NEM id, latitude (degrees), longitude (degrees), and altitude (meters)
 |
 |  export(...)
 |      export() -> string
 |
```

```
 |      Returns a string containing the binary location event data in a format
 |      suitable for publication
 |
 |  set(...)
 |      set(index,NEMId,latitude,longitude,altitude)
 |
 |      Set location information for a specific NEM in the location event data.
 |
 |      index     - Entry to set.
 |
 |      NEMId     - NEM Id
 |
 |      latitude  - Latitude in degrees
 |
 |      longitude - Longitude in degrees
 |
 |      altitude  - Altitude in meters
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __new__ = <built-in method __new__ of type object at 0x7f33746d3ae0>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T

```

Python emaneeventservice

```
NAME
    emaneeventpathloss

DESCRIPTION
    EMANE Event Pathloss creates and parses EMANE Pathloss events.  Pathloss
    events contain RF Pathloss in dB between transmitting NEMs and the receiving
    NEM.

    CLASSES
     EventPathloss

DATA
    EVENT_ID = 203
    error = 'emaneeventpathloss.error'
```

```
emaneeventpathloss.EventPathloss = class EventPathloss(object)
 |  EventPathloss(entryCount)
 |  EventPathloss(data)
 |
 |  EMANE Event Pathloss objects can be created by specifying the
 |  number of event entries to create or from received event data.
 |
 |  entryCount - Number of pathloss event entries.
 |               Use to construct a new pathloss event message.
 |               Not valid when used with data
 |
 |  data       - Binary event data received from the event service.
 |               Use to construct a pathloss event message from
 |               received data. Not valid when used with entryCount
 |
 |  SYNOPSIS
 |
 |  import emaneeventservice
 |
 |  service = emaneeventservice.EventService()
 |
 |  def handlePathlossEvent(event,platform,nem,component,data):
 |      print "event ", event, " length ",len(data), " bytes"
 |      event = emaneeventpathloss.EventPathloss(data)
 |      entries  = event.entries()
 |      for e in entries.values():
 |          print e
 |
 |  service.subscribe(emaneeventpathloss.EVENT_ID,
 |                    handlePathlossEvent)
 |
 |  service.loop()
 |
 |  service.unsubscribe(emaneeventpathloss.EVENT_ID)
 |
 |  ...
 |
 |  service.publish(emaneeventpathloss.EVENT_ID,
 |                  emaneeventservice.PLATFORMID_ANY,
 |                  emaneeventservice.NEMID_ANY,
 |                  emaneeventservice.COMPONENTID_ANY,
 |                  data);
 |
 |  Methods defined here:
 |
 |  entries(...)
 |      entries() -> {txNEM,(txNEM,pathloss,reversePathloss)}
 |
 |      Returns a dictionary with txNEM keys and tuple values containing the
 |      transmitting NEM id, receiving NEM id, pathloss, and
 |      reverse pathloss
 |
 |  export(...)
 |      export() -> string
```

```
 |
 |      Returns a string containing the binary pathloss event data in a format
 |      suitable for publication
 |
 |  set(...)
 |      set(index,txNNEM,pathloss,reversePathloss)
 |
 |      Set pathloss information for a specific NEM in the pathloss
 |      event data
 |
 |      index           - Entry index to set.
 |
 |      txNEM           - NEM id of the transmitting NEM.
 |
 |      pathloss        - Pathloss from the txNEM to the rxNEM
 |
 |      reversePathloss - Pathloss from rxNEM to the txNEM
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __new__ = <built-in method __new__ of type object at 0x7fe45c131b00>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```

Python emaneeventservice

```
NAME
    emaneeventcommeffect

DESCRIPTION
    EMANE Event CommEffect creates and parses EMANE CommEffect events.
    CommEffect events contain communication effects from transmitting NEMs
    to a recieving NEM.

    CLASSES
     EventCommEffect

DATA
    EVENT_ID = 207
    error = 'emaneeventcommeffect.error'
```

```
emaneeventcommeffect.EventCommEffect = class EventCommEffect(object)
 |  EventCommEffect(entryCount)
 |  EventCommEffect(data)
 |
 |  EMANE Event CommEffect objects can be created by specifying the
 |  number of event entries to create or from received event data.
 |
 |  entryCount - Number of event entries.  Use to construct a new
 |               commeffect event message. Not valid when used with
 |               data.
 |
 |  data       - Binary event data received from the event service.
 |               Use to construct a commeffect event message from
 |               received data. Not valid when used with entryCount
 |
 |  SYNOPSIS
 |
 |  import emaneeventservice
 |
 |  service = emaneeventservice.EventService()
 |
 |  def handleCommEffectEvent(event, platform, nem, component, data):
 |      print "event ", event, " length ",len(data), " bytes"
 |      event = emaneeventcommeffect.EventCommEffect(data)
 |      entries  = event.entries()
 |      for e in entries.values():
 |          print e
 |
 |  service.subscribe(emaneeventcommeffect.EVENT_ID,
 |                    handleCommEffectEvent)
 |
 |  service.loop()
 |
 |  service.unsubscribe(emaneeventcommeffect.EVENT_ID)
 |
 |  ...
 |
 |  service.publish(emaneeventcommeffect.EVENT_ID,
 |                  emaneeventservice.PLATFORMID_ANY,
 |                  emaneeventservice.NEMID_ANY,
 |                  emaneeventservice.COMPONENTID_ANY,
 |                  data);
 |
 |  Methods defined here:
 |
 |  entries(...)
 |      entries() -> {txNEM,(txNNEM,latencySeconds,latencyMicroSeconds,
 |                           jitterSeconds, jitterMicroSeconds,
 |                           probablityLoss,probablityDuplicate,
 |                           unicastBitRate,broadcastBitRate)}
 |
 |      Returns a dictionary with txNEM keys and tuple values containing the
 |      transmitting NEM id, latency, jitter, loss, duplicate, and bit rate comm
```

```
 |      effects
 |
 |  export(...)
 |      export() -> string
 |
 |      Returns a string containing the binary commeffect event data in a format
 |      suitable for publication
 |
 |  set(...)
 |      set(index,txNNEM,latencySeconds,latencyMicroSeconds,
 |          jitterSeconds,jitterMicroSeconds,probablityLoss,
 |          probablityDuplicate,unicastBitRate,broadcastBitRate)
 |
 |      Set commeffect information for a specific NEM in the commeffect
 |      event data.
 |
 |      index               - Index of entry to set.
 |
 |      txNEM               - NEM id of the transmitting NEM.
 |
 |      latencySeconds      - Whole seconds portion of the latency from txNEM to
 |                            rxNEM
 |
 |      latencyMicroSeconds - Whole microseconds portion of the latency from
 |                            txNEM to rxNEM
 |
 |      jitterSeconds       - Whole seconds portion of the jitter from txNEM
 |                            to rxNEM
 |
 |      jitterMicroSeconds  - Whole microseconds portion of the jitter from
 |                            txNEM to rxNEM
 |
 |      probabilityLoss     - Probablity of loss.  Integer value [0,100]
 |
 |      probablityDuplicate - Probablity of duplication. Integer value [0,65535]
 |
 |      unicastBitRate      - Unicast bit rate from txNEM to rxNEM
 |
 |      broadcastBitRate    - Broadcast bit rate from txNEM to rxNEM
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __new__ = <built-in method __new__ of type object at 0x7f4fb846a6e0>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```