

Welcome to EMANE User Training

- Please select your VM of choice:

- Parallels:

`emanedemo/vm/parallels/emanedemo-0.7.x.20120223-1.pvm.tar.bz2`

- VMware or VirtualBox:

`emanedemo/vm/vbox_vmware/emanedemo-0.7.x.20120223-1.tar.bz2`

- To extract in a Linux or OS X terminal:

```
tar xjvf emanedemo/vm/parallels/emanedemo-0.7.x.20120223-1.pvm.tar.bz2
```

- To extract in Windows

- Double click on `emanedemo-0.7.x.20120223-1.tar.bz2`

- Double click on `emanedemo-0.7.x.20120223-1.tar`

- If you are using VirtualBox you will need to import the Virtual Machine

Feel free to ask for help





EMANE

User Training

0.7.3

Extendable Mobile Ad-hoc Emulator

The Extendable Mobile Ad-hoc Network Emulator (EMANE) is an open source framework which provides wireless network experimenters with a highly flexible modular environment for use during the design, development and testing of simple and complex network architectures.

EMANE provides a set of well-defined APIs to allow independent development of network emulation modules, emulation/application boundary interfaces and emulation environmental data distribution mechanisms.

Extendable Mobile Ad-hoc Emulator

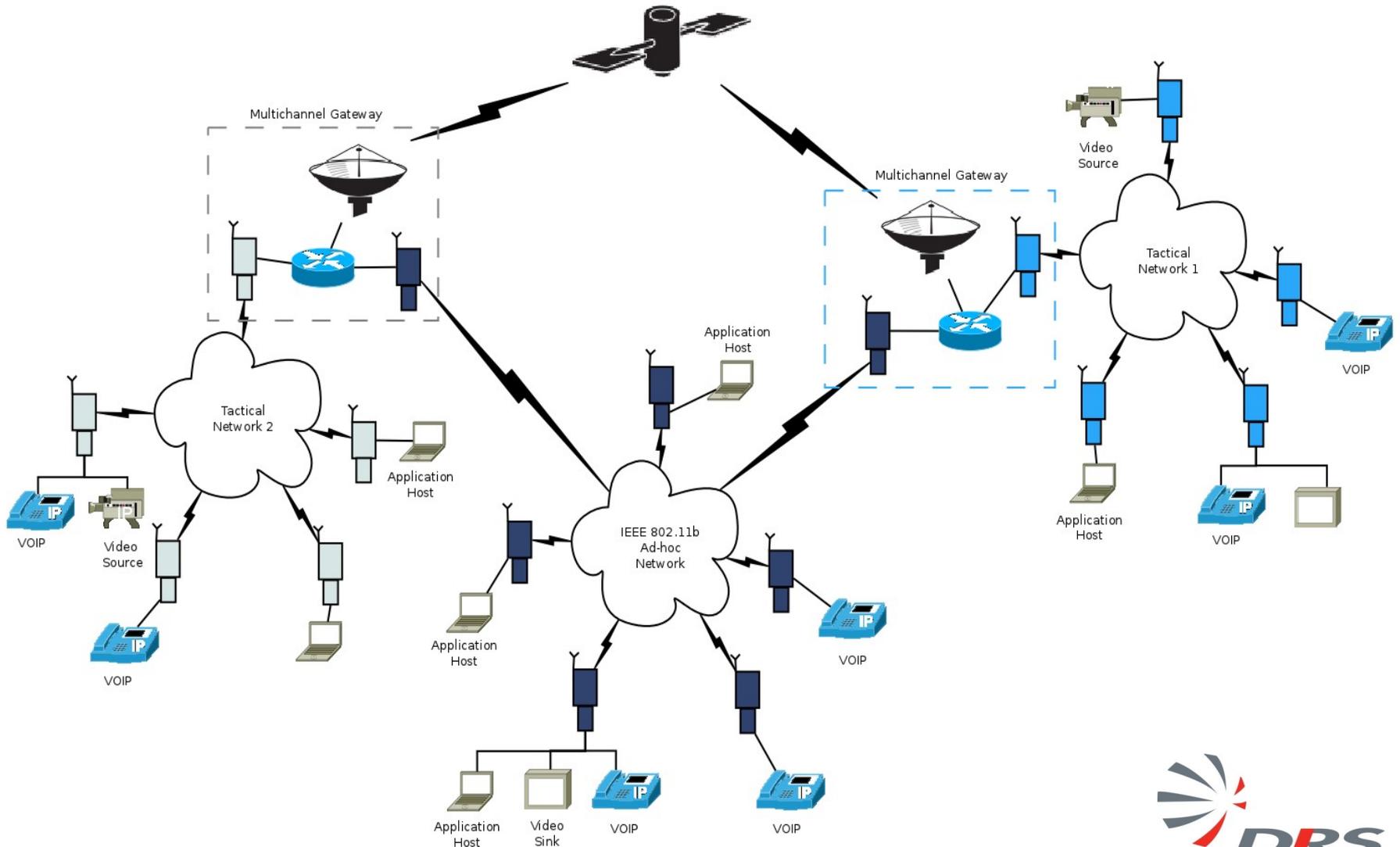
- Supports emulation of simple as well as complex network architectures
- Supports emulation of multichannel gateways
- Supports model specific configuration and control messaging
- Provides mechanisms to bridge emulation environment control information with non-emulation aware components
- Supports large scale testbeds with the same ease as small test networks
- Supports cross platform deployment (Unix, Linux, OS X, MS Windows)



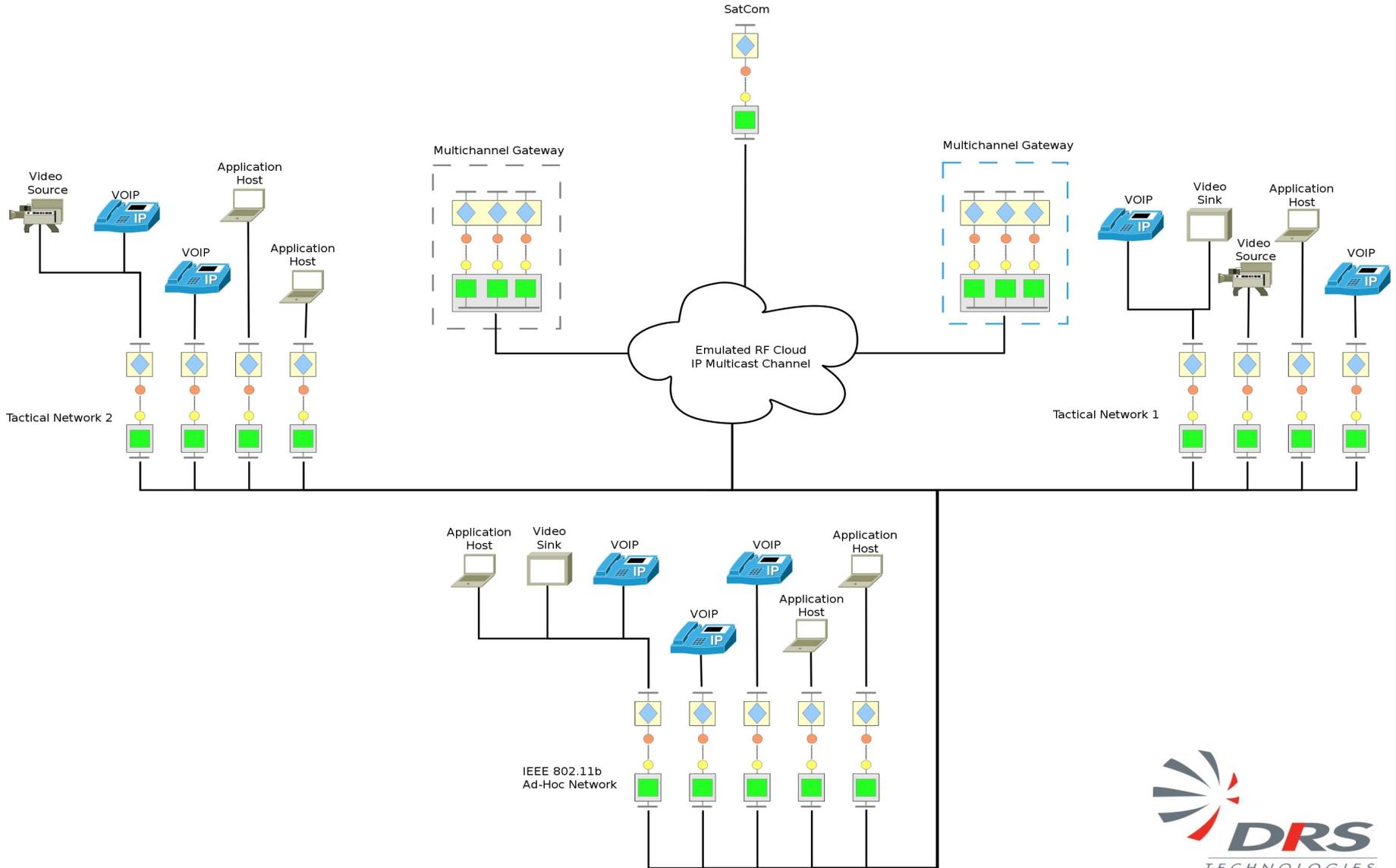
Mapping a Real World Deployment



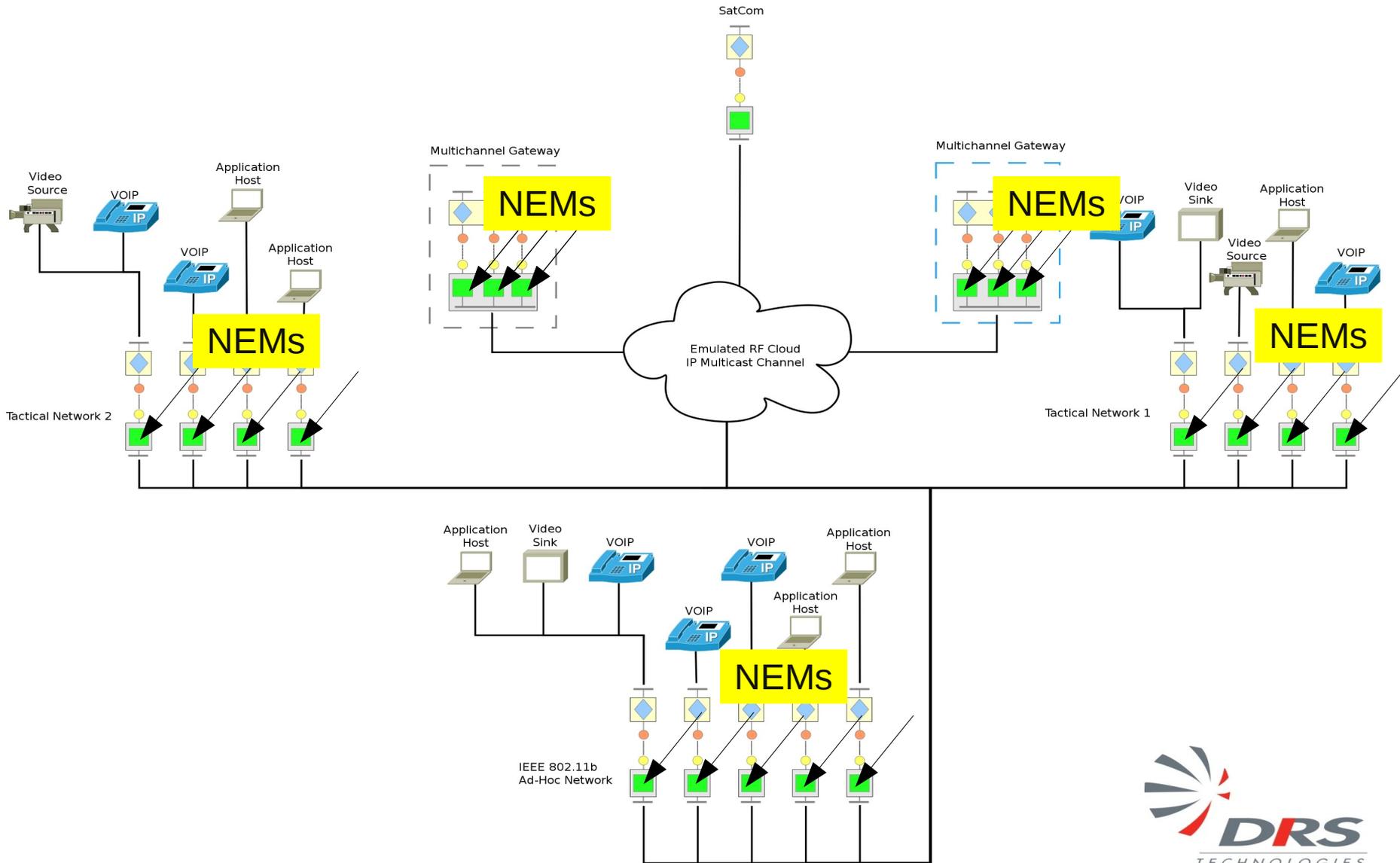
Heterogeneous Network



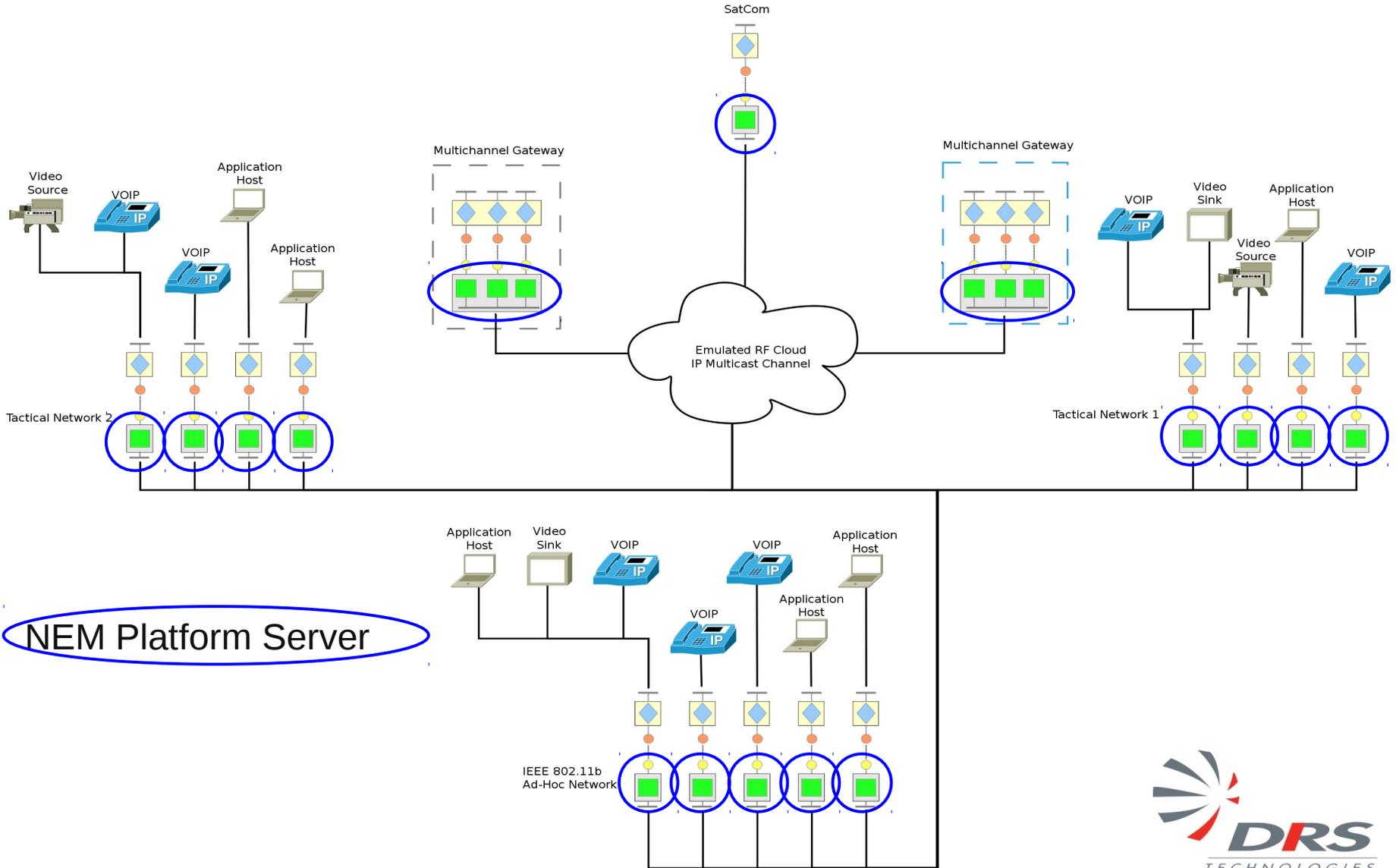
EMANE Heterogeneous Model Deployment



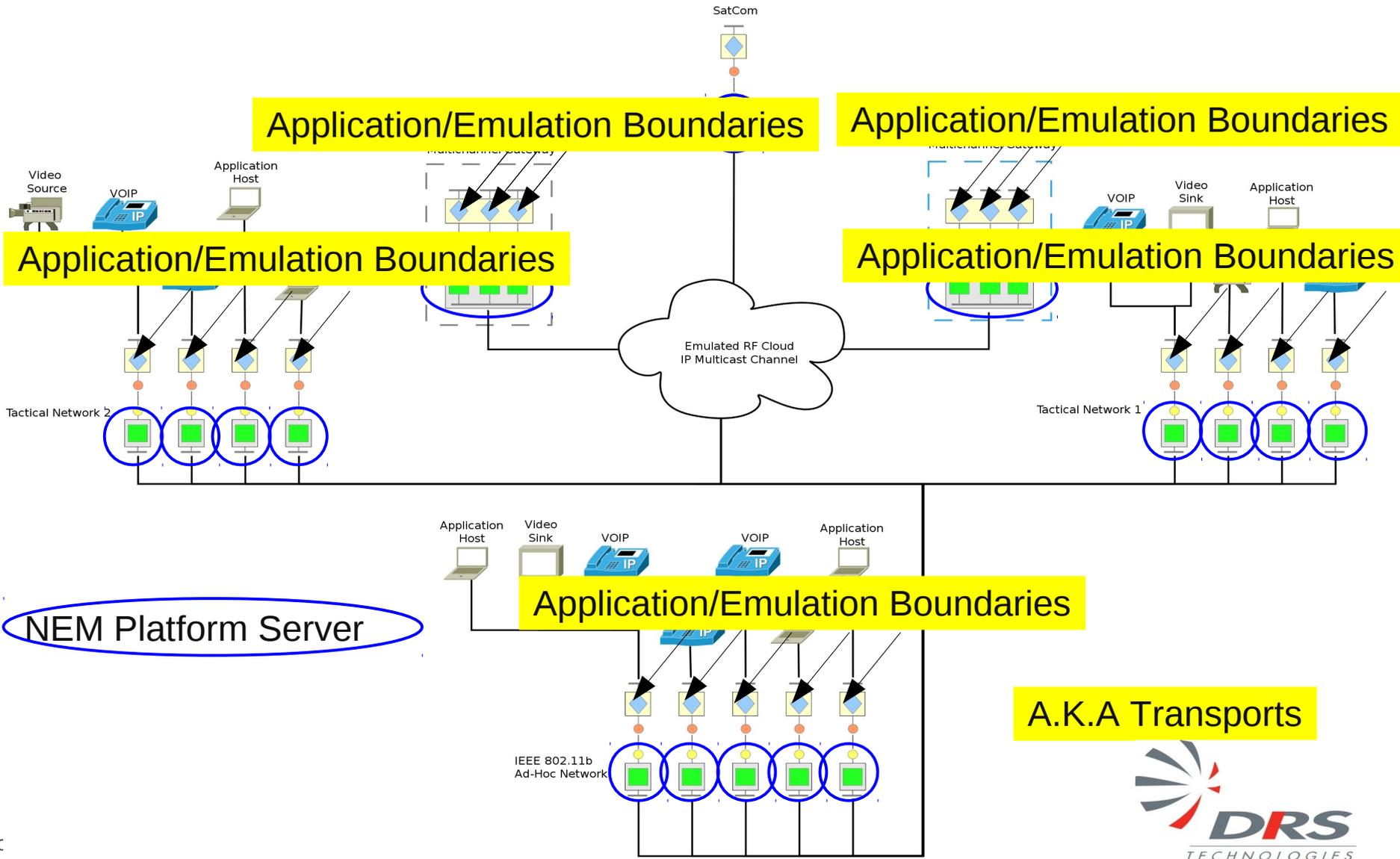
EMANE Heterogeneous Model Deployment



EMANE Heterogeneous Model Deployment



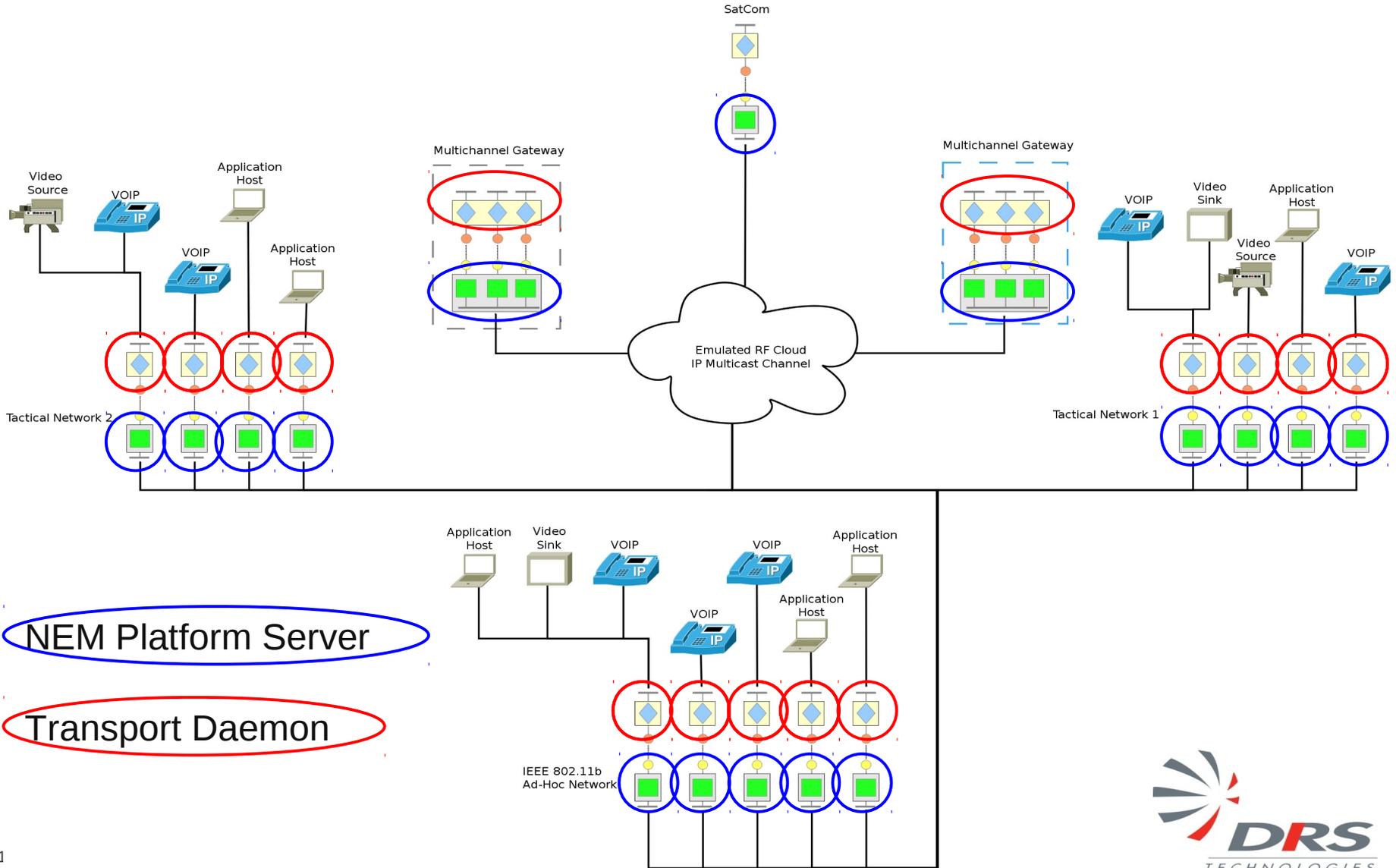
EMANE Heterogeneous Model Deployment



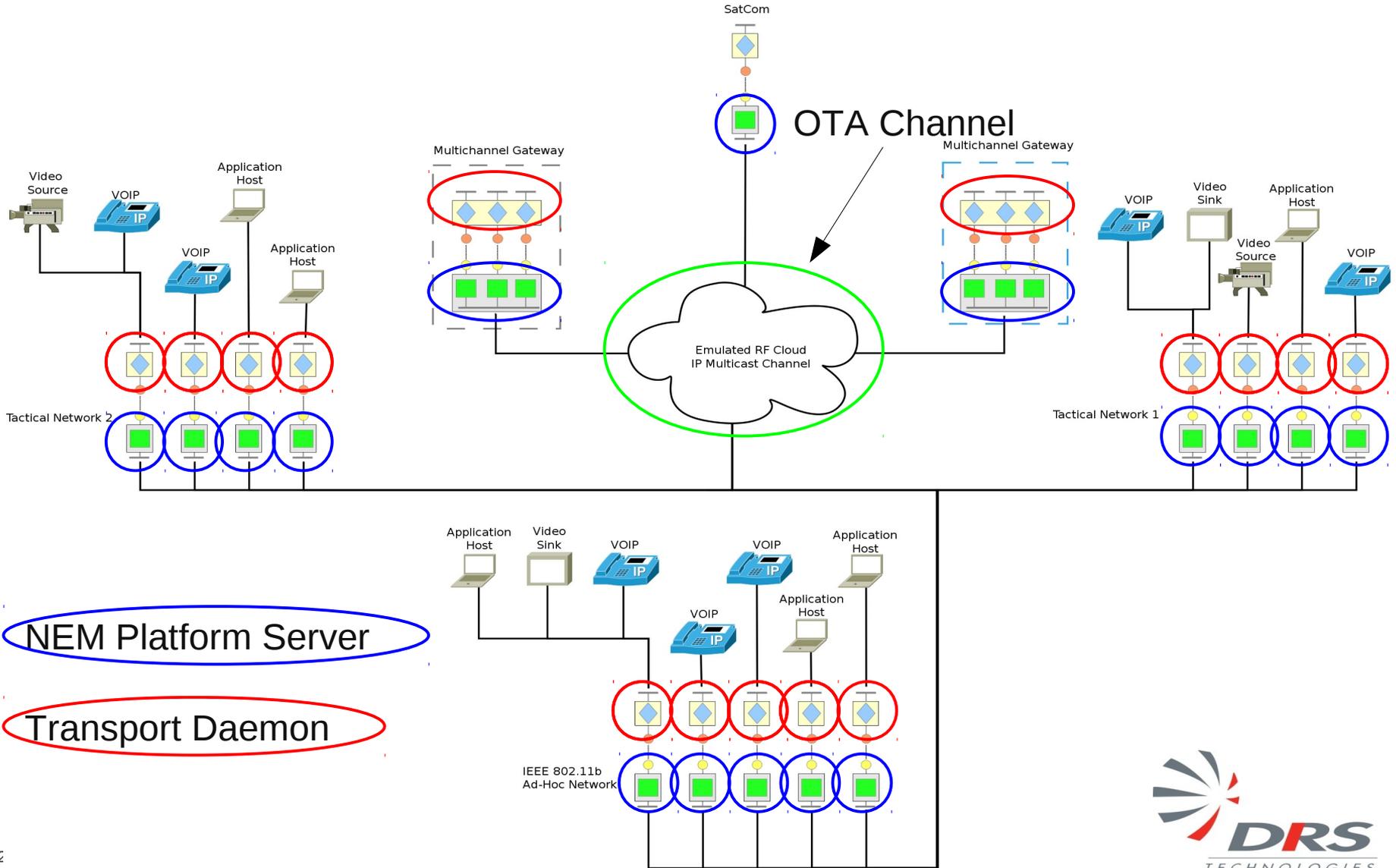
A.K.A Transports



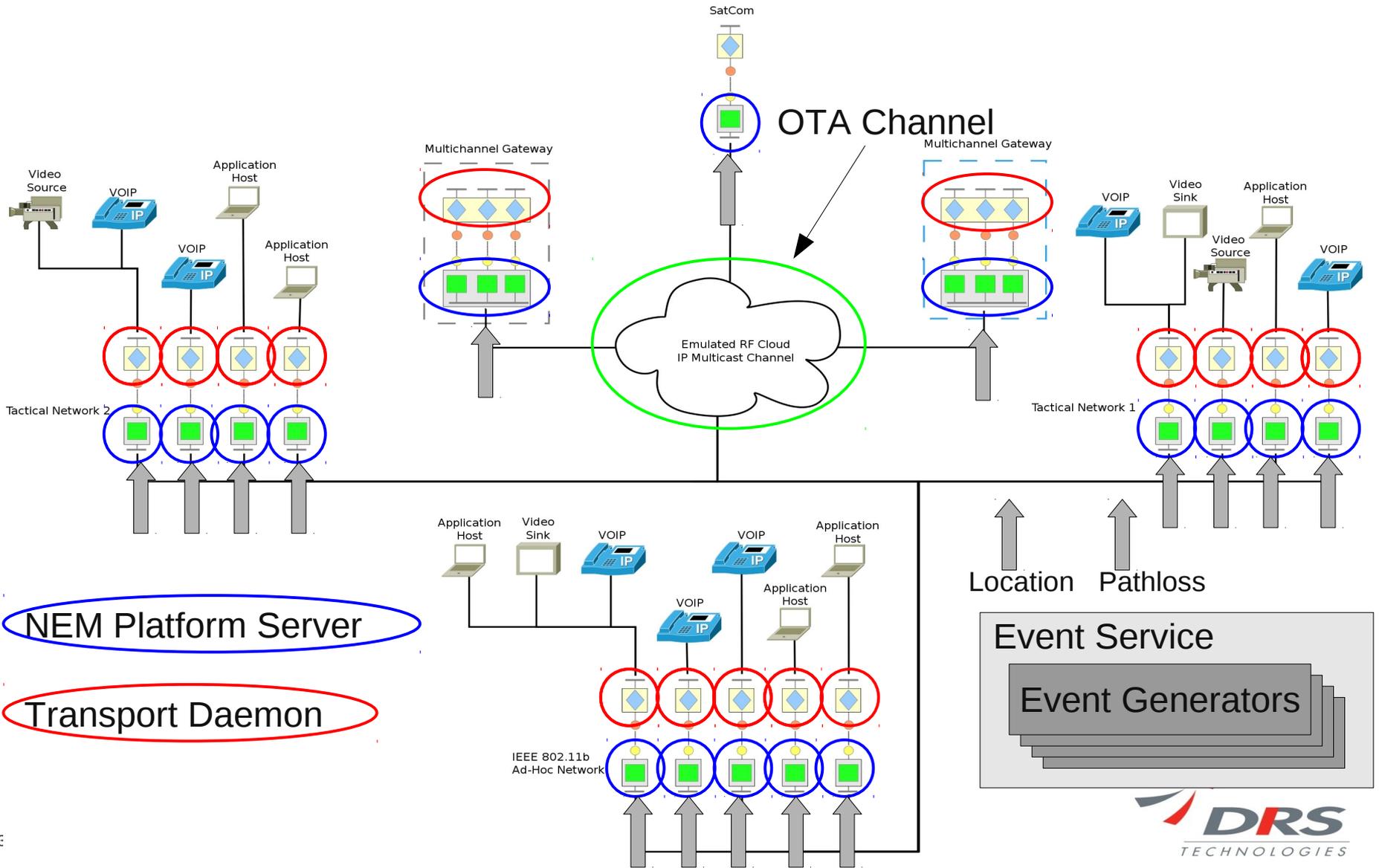
EMANE Heterogeneous Model Deployment



EMANE Heterogeneous Model Deployment



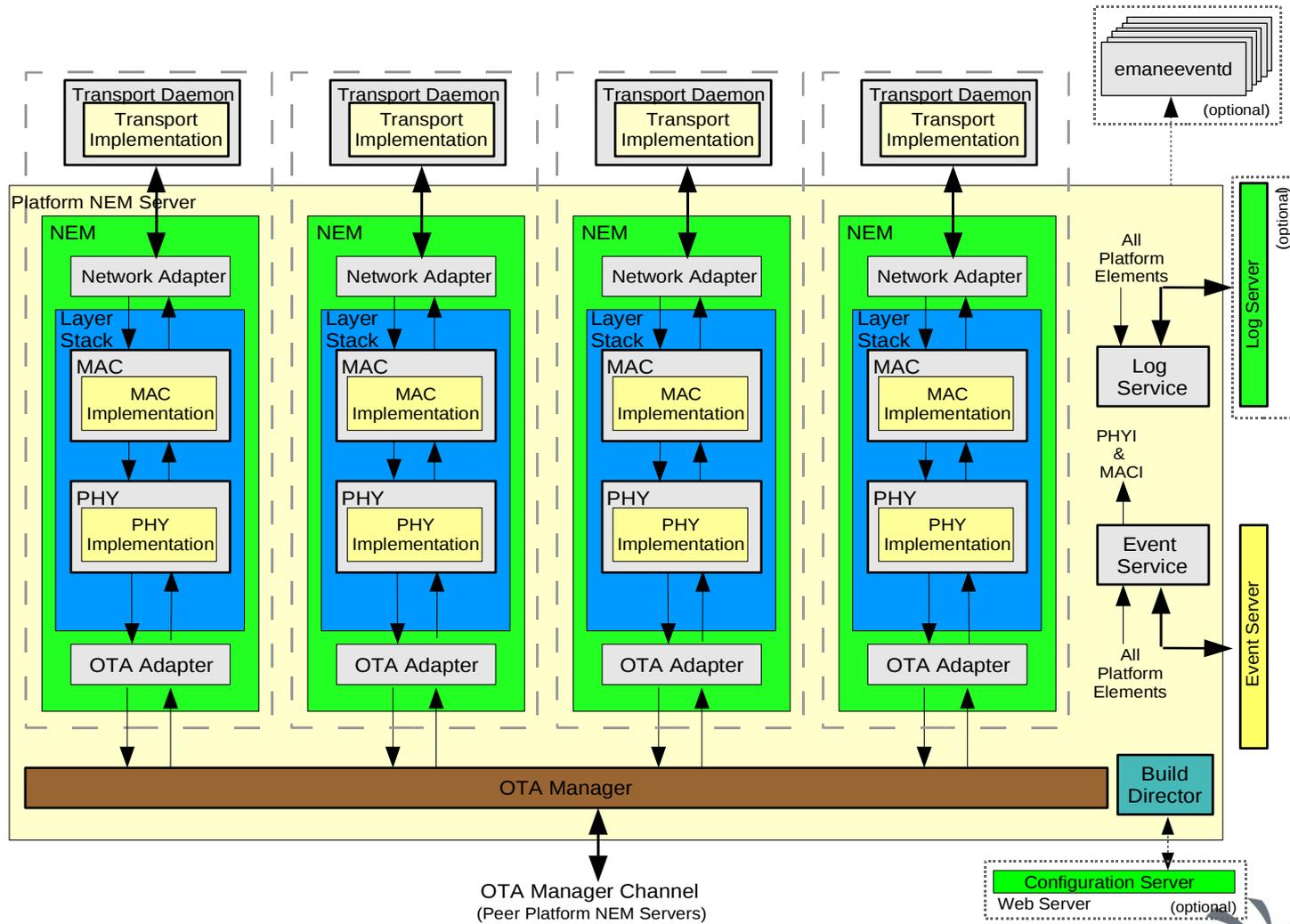
EMANE Heterogeneous Model Deployment



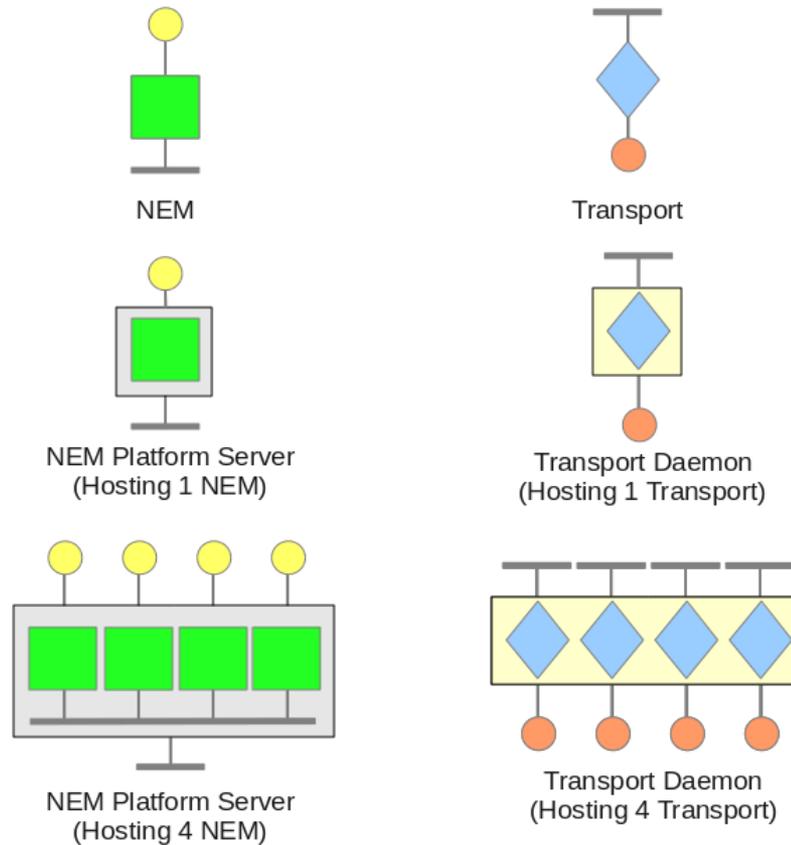
NEM Platform Server

Transport Daemon

EMANE Architecture



Deployment Diagram Key





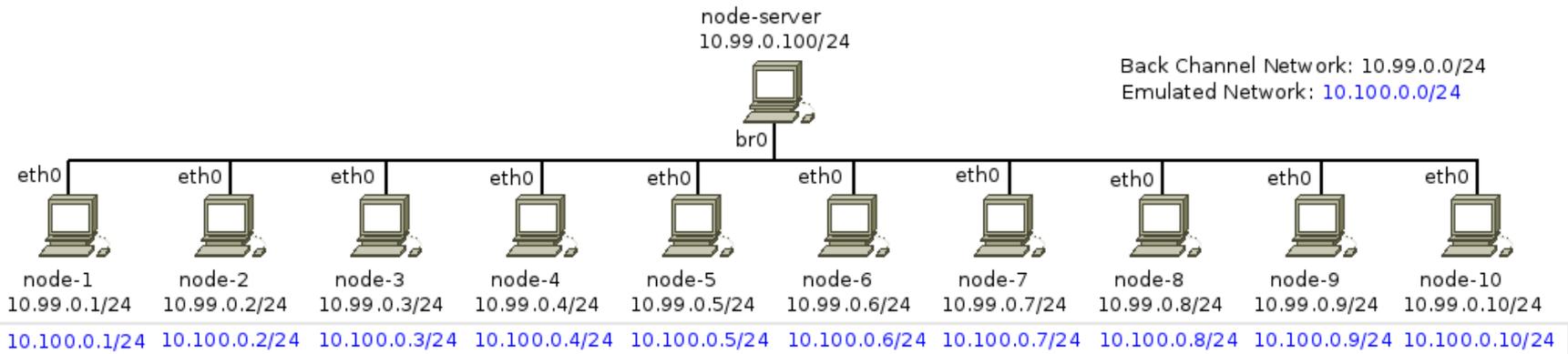
EMANE Demo VM



EMANE Demo VM

- The EMANE Demo Virtual Machine is a 32 bit RPM based Linux VM fully configured with the latest EMANE release and the latest EMANE User Manual Demonstrations.
- Any modern Linux installation with lxc Linux Container support running the latest EMANE release will be able to execute the EMANE User Manual demonstrations.
 - Each demonstration makes use of Linux Containers to create lightweight virtual nodes. Each container node is assigned their own Network and PID namespace to provide network stack and process isolation.
 - Each demonstration configures the containers and the applications running in each container node respective to the features and mechanisms being demonstrated.
 - Each container node is running an SSH server to allow hands on interaction and examination during the demonstration.

Demo Node Test Network



Virtual Transport demonstrations use emane0 for 10.100.0.0/24
Raw Transport demonstrations use eth1 for 10.100.0.0/24



Infrastructure Basics



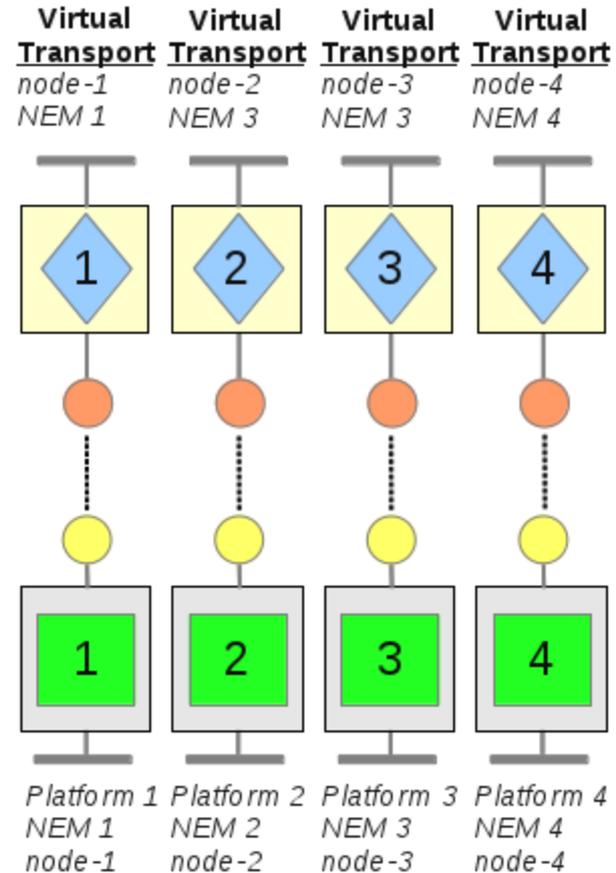
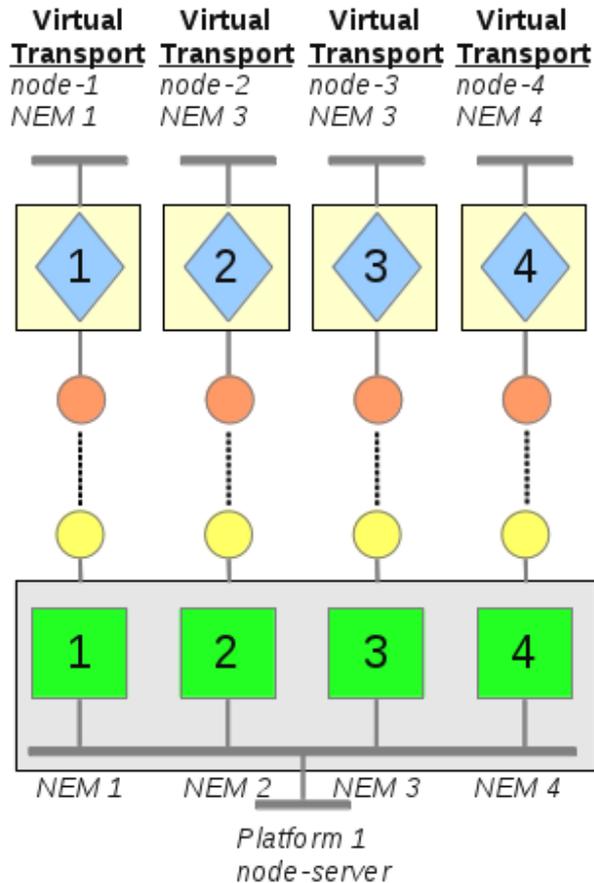
NEM Platform Server

- Creates and manages NEM instances
 - Processes an XML configuration file to determine the quantity and types of NEMs to instantiate.
- Manages NEM OTA Communication
 - NEMs belonging to the same platform use thread shared memory message passing.
 - NEMs belonging to different platforms communicate using a multicast channel referred to as ***Over-The-Air (OTA) Manager Channel***.
- Manages Event Distribution
 - NEMs belonging to the same platform use thread shared memory message passing.
 - NEMs belonging to different platforms communicate using a multicast channel referred to as the ***Event Service Channel***.

Deployment Types

- **Centralized Deployment** - Single NEM Platform Server that instantiates all of the NEMs contained in the deployment.
- **Distributed Deployment** - Multiple NEM Platform Servers each containing a single NEM instance. In a distributed deployment, the number of NEM Platform Servers equals the number of NEMs in the deployment.
- **Hybrid Deployment** - Multiple NEM Platform Servers, with at least one containing multiple NEM instances.

Centralized and Distributed Deployment Example



Centralized Deployment Example

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

Listing 2.2: NEM Platform Server configuration for Figure 2.1.

Distributed Deployment Example

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platfo
3 <platform name="Platform 1" id="1">
4   <param name="otamanagerchannelenable" value="on"/>
5   <param name="otamanagerdevice" value="eth0"/>
6   <param name="otamanagergroup" value="224.1.2.8:45702"/>
7   <param name="eventservicegroup" value="224.1.2.8:45703"/>
8   <param name="eventservicedevice" value="eth0"/>
9
10  <nem name="NEM-1" id="1" definition="bypassnem.xml">
11    <param name="platformendpoint" value="localhost:8201"/>
12    <param name="transportendpoint" value="localhost:8301"/>
13    <transport definition="transvirtual.xml">
14      <param name="address" value="10.100.0.1"/>
15      <param name="mask" value="255.255.255.0"/>
16    </transport>
17  </nem>
18 </platform>
```

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

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE platform SYSTEM "file:///usr/share/emane/dtd/platfo
3 <platform name="Platform 2" id="2">
4   <param name="otamanagerchannelenable" value="on"/>
5   <param name="otamanagerdevice" value="eth0"/>
6   <param name="otamanagergroup" value="224.1.2.8:45702"/>
7   <param name="eventservicegroup" value="224.1.2.8:45703"/>
8   <param name="eventservicedevice" value="eth0"/>
9
10  <nem name="NEM-2" id="2" definition="bypassnem.xml">
11    <param name="platformendpoint" value="localhost:8201"/>
12    <param name="transportendpoint" value="localhost:8301"/>
13    <transport definition="transvirtual.xml">
14      <param name="address" value="10.100.0.2"/>
15      <param name="mask" value="255.255.255.0"/>
16    </transport>
17  </nem>
18 </platform>
```

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

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

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

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

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

NEM Platform Server Configuration



- Over-The-Air Manager Channel Configuration
 - ***otamanagergroup*** - The Over-The-Air (OTA) Channel multicast endpoint used to communicate between multiple NEM Platform Servers in an EMANE deployment.
 - ***otamanagerdevice*** - The network device to associate with the OTA Manager Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.
 - ***otamanagerchannelenable*** - Enable or disable the OTA Manager Channel. When disabled, there is no inter-NEM Platform Server communication and only NEMs managed locally by a single NEM Platform Server will be able to communicate.
- Event Service Channel Configuration
 - ***eventservicegroup*** - The Event Service Channel multicast endpoint used to communicate events between EMANE components in an EMANE deployment.
 - ***eventservicedevice*** - The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.



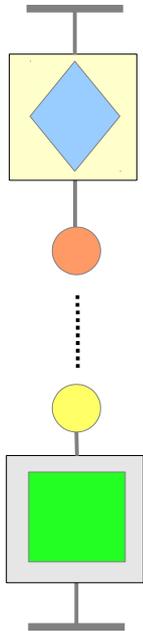
NEM Platform Server Configuration

- Debug Port
 - ***debugport*** - NEM Platform Server telnet debug UDP port.
 - ***debugportenable*** - Enable or disable the NEM Platform Server telnet debug port.

NEM Platform Server Shared Configuration



Transport Daemon
(Hosting 1 Transport)



NEM Platform Server
(Hosting 1 NEM)

NEM Platform Server and Transport Daemon Shared Configuration

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

Transport Daemon

- Creates and manages emulation/application boundary (Transport) instances
 - Processes an XML configuration file to determine the quantity and type of Transports to instantiate.
- Manages NEM/Transport communication
 - NEM Layer stacks communicate with their respective Transport instance using UDP messaging.
- Data received from the application domain is transmitted opaquely to the Transport's respective NEM for OTA message processing
 - Only EMANE component aware of the exact format of the application domain data.
 - Supplies the destination address, either a unicast or broadcast NEM identifier, to the NEM stack.
- Transports may or may not be designed to inter-operate with other dissimilar boundary components in the same experiment

Centralized Deployment Example

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

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

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

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

Centralized Deployment Example

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

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

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

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

Distributed Deployment Example

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

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

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

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

Distributed Deployment Example

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

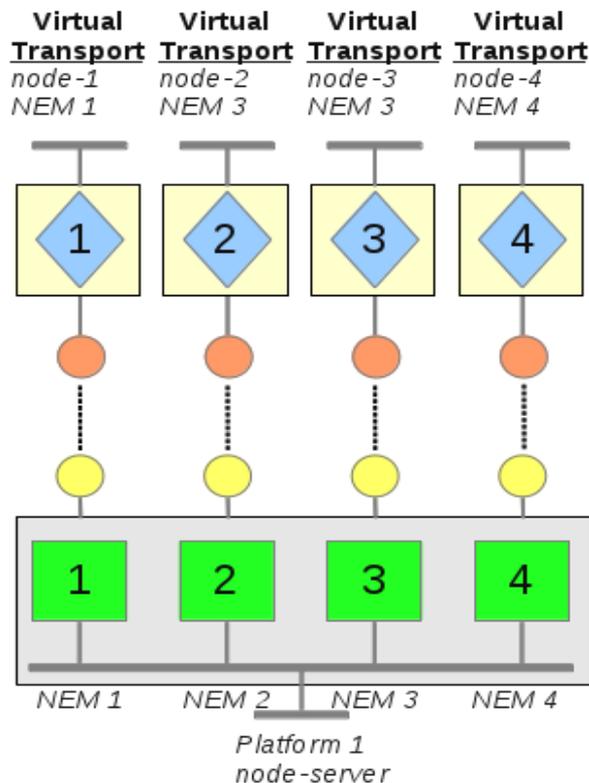
Listing 2.14: Transport Daemon 3 XML for NEM 3 from distributed NEM Platform Server XML Listing 2.5.

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

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

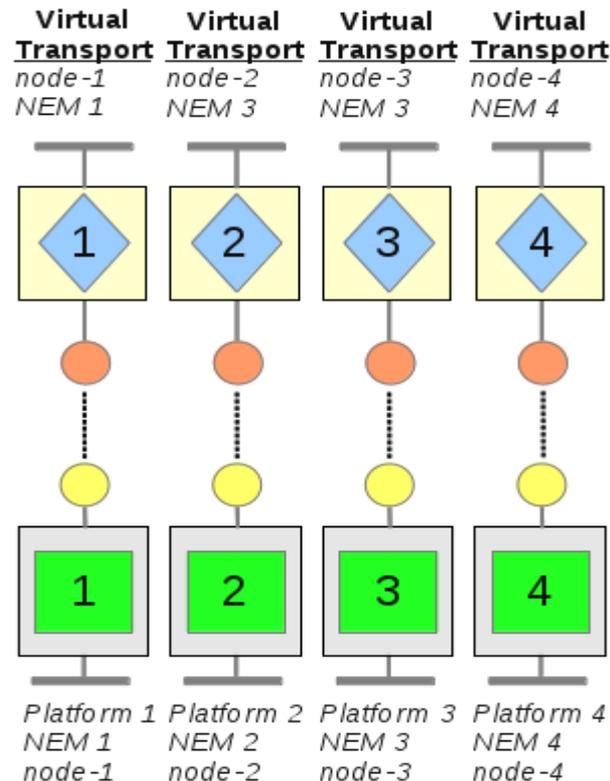
Demonstration 1

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



Demonstration 2

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



Network Emulation Modules

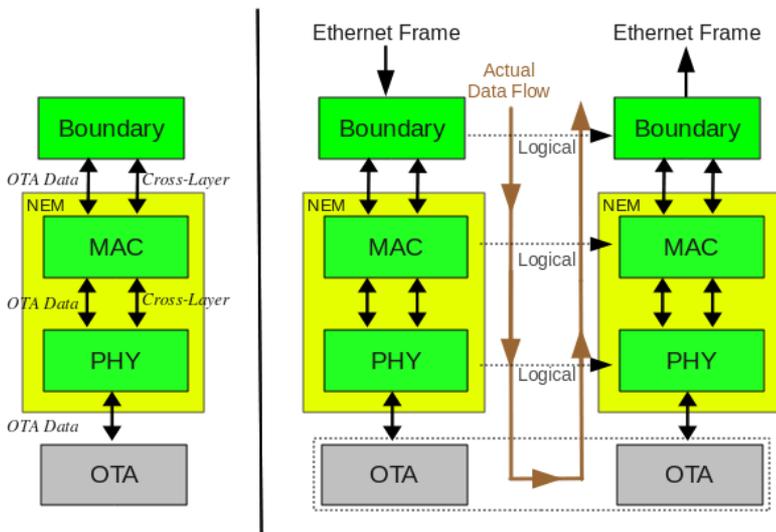


NEM Anatomy

- A Network Emulation Module (NEM) is a logical component that encapsulates all the functionality necessary to emulate a particular type of network technology.
 - **Structured NEM** - Component stack composed of a Physical (PHY) Layer implementation, a Medium Access Control (MAC) Layer implementation and zero or more Shim Layer implementations
 - **Unstructured NEM** - Component stack composed of zero or one PHY Layer implementation, zero or one MAC Layer implementation and zero or more Shim Layer implementations.

NEM Layer Communication

- NEM layers communicate with each other using a generic message passing interface.
- Each layer is capable of communicating cross-layer control messages and OTA messages with their neighboring layers
 - Examples of cross-layer messages include: per packet RSSI, carrier sense, and transmission control messages.



- Each layer has the capability to generate OTA messages for communication with their respective layer counterparts
- Layers may also append and strip layer specific headers to OTA messages.

Defining an NEM

- NEM is defined using an XML configuration file.
 - NEM Layer Stack defined **<mac>**, **<phy>**, **<shim>** and **<transport>** XML elements.
 - Each element has a mandatory definition attribute which references the XML configuration associated with the component.
- All NEM definitions are subject to the following rules which are enforced by the EMANE NEM DTD:
 - The order in which child elements are listed within the **<nem>** definition block corresponds to the order the plugin layers will be connected once instantiated, with the exception of the **<transport>** element.
 - The first child element in the **<nem>** definition block is the most upstream non-transport layer.
 - The **<transport>** element must be the last child element in the **<nem>** definition block.

Defining an NEM

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

Listing 3.3: Bypass NEM XML configuration.

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

Listing 3.4: Bypass MAC XML configuration.

Defining an NEM

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

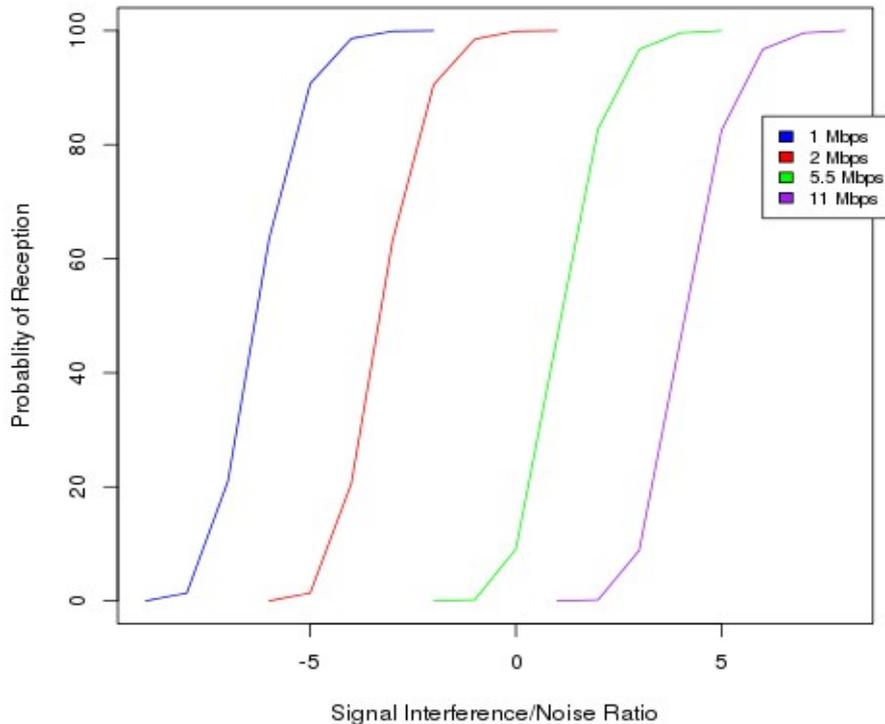
Listing 3.5: Bypass PHY XML configuration.

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

Listing 3.6: Virtual Transport XML configuration.

Physical Layer

- The primary function is to accurately account for the key set of factors that impact the reception of data.
- Data reception based on the Signal to Interference plus Noise Ratio (SINR) at the receiving node.
 - Slight variations in SINR can impact the receiver's ability to receive data



- Factors impacting the receive signal:
 - Signal Propagation Models
 - Antenna Modeling

Physical Layer

Factors impacting Interference and Noise:

- **Receiver Sensitivity** - Minimum input power at the receiver for (possible) successful data reception or the Noise Floor of the receiver when there is no other interference
 - Receiver sensitivity is based on Thermal Noise Power (dBm) and the Noise Figure (dB) associated with the receiver.
 - **Thermal Noise Power** = $-174 + 10\log(\text{bandWidth})$
 - **Noise Figure** – Any additional degradation of the signal caused by components within the RF signal chain of the receiver. Typically 4 to 6 dB.

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

Physical Layer

- Factors impacting Interference and Noise:
 - ***Interference from intentional and unintentional RF emitters*** - Any additional RF energy within the RF spectrum of the receiver and can raise the over all Noise Floor

Supporting Heterogeneous Waveforms

- Common PHY Header is a mandatory PHY Layer model header. Allows different physical layer models to process the potential spectrum impact of packets generated by other waveforms.
 - Supports Receive Power Control
 - Supports Noise/Interference Calculations
- Mandatory Header Contents:
 - Registration Id of the PHY Layer Model
 - Transmit power in dBm of the transmitter
 - Antenna gain in dBi of the transmitter
 - Timestamp of the transmitted packet
 - Duration of the transmitted packet
 - Center frequency in KHz
 - Bandwidth in KHz
 - Packet sequence number

Supporting Heterogeneous Waveforms

- Optional Header Contents:
 - Transmitter antenna type
 - Transmitter antenna azimuth beam width in degrees
 - Transmitter antenna elevation beam width in degrees
 - Transmitter antenna azimuth in degrees
 - Transmitter antenna elevation in degrees

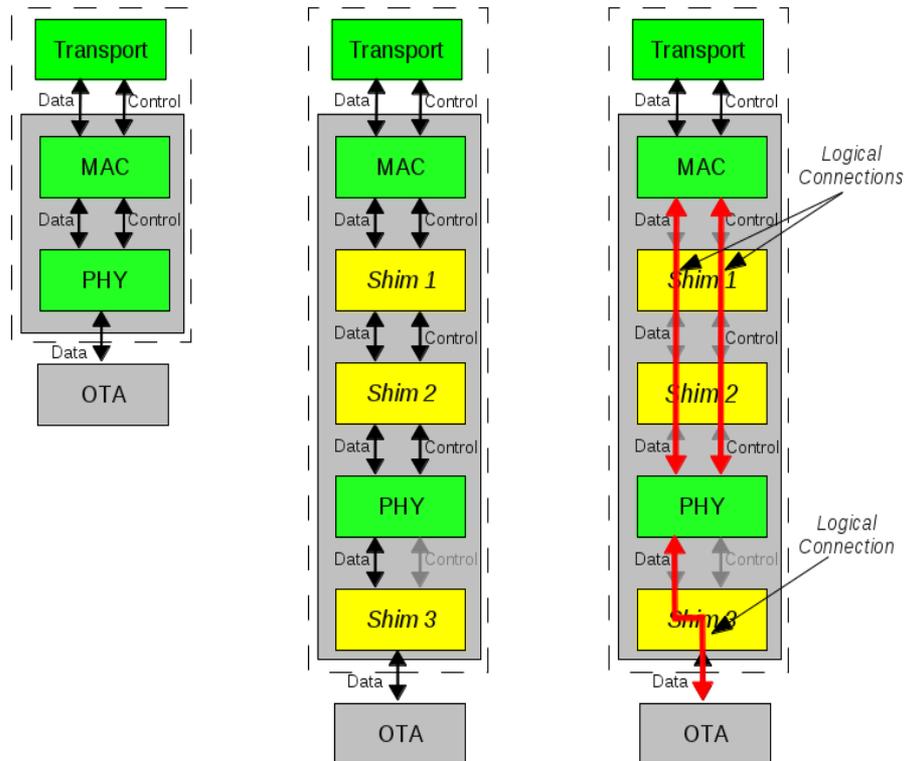
MAC Layer

- Defines the mechanisms used to control access to a wireless medium shared by multiple nodes
 - Channel access protocols
 - Time Division Multiple Access (TDMA)
 - Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA)
 - Frequency Division Multiple Access (FDMA)
 - Code Division Multiple Access (CDMA).
- May include support for QoS requirements:
 - Queuing
 - Acknowledgments
 - Retries
 - Fragmentation
 - Segmentation

Shim Layer

- Allows monitoring and modification of OTA and cross-layer messaging exchanged between contiguous layers.
- May generate OTA messages for communication with respective Shim Layers contained in different NEMs
- May append and strip layer specific headers.
- May be inserted between the layers of a NEM stack or at the top and bottom of a layer stack.
- Implements the same generic interface as a MAC and PHY which allows insertion between components without requiring those components to have knowledge of their presence.

NEM Layer Stack with Shim Layers





Events



Event Service

- **Events** - Opaquely distributed messages delivered in realtime to one or more targeted components.
- **Event Generators** - Components which create events based on scenarios.
- **Event Service** - Creates and manages Event Generator instances.
 - Processes an XML configuration file to determine the quantity and type of Event Generators to instantiate.

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

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

Event Service

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

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

Event Generators

- No restriction is placed on how Event Generators create events.
 - Read precomputed state information from input files and publish events on time boundaries.
 - Create event data algorithmically in realtime and publish the events based on update threshold logic.
- New event types require no modification to existing components provided those components are not required to process the new events.
- Only the Event Generator and the destination EMANE components need to know the more specialized form of the generically transmitted event.
- Events are addressable using a 3-tuple: NEM Platform Server identifier, NEM identifier and component identifier.

Event Generator XML Definition

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

Listing 4.4: Sample Event Generator XML configuration.

Event Service Configuration

- Event Service Channel Configuration
 - **eventservicegroup** - The Event Service Channel multicast endpoint used to communicate events between EMANE components in an EMANE deployment.
 - **eventservicedevice** - The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Event Daemon

- **Event Agent** - Component that translates events from their emulation domain representation to a format usable by application domain entities.
 - Facilitates reuse of any experiment scenario information propagated via an event that is of interest outside of the emulation domain.
- **Event Daemon** - Creates and manages Event Agent instances
 - Processes an XML configuration file to determine the quantity and type of Event Agent to instantiate.

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

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

Event Agent XML

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

Listing 4.7: Sample Event Agent XML configuration.

Event Daemon Configuration

- Event Service Channel Configuration
 - **eventservicegroup** - The Event Service Channel multicast endpoint used to communicate events between EMANE components in an EMANE deployment.
 - **eventservicedevice** - The network device to associate with the Event Service Channel multicast endpoint. If missing, the kernel routing table is used to route multicast joins and packet transmissions.

Event Types

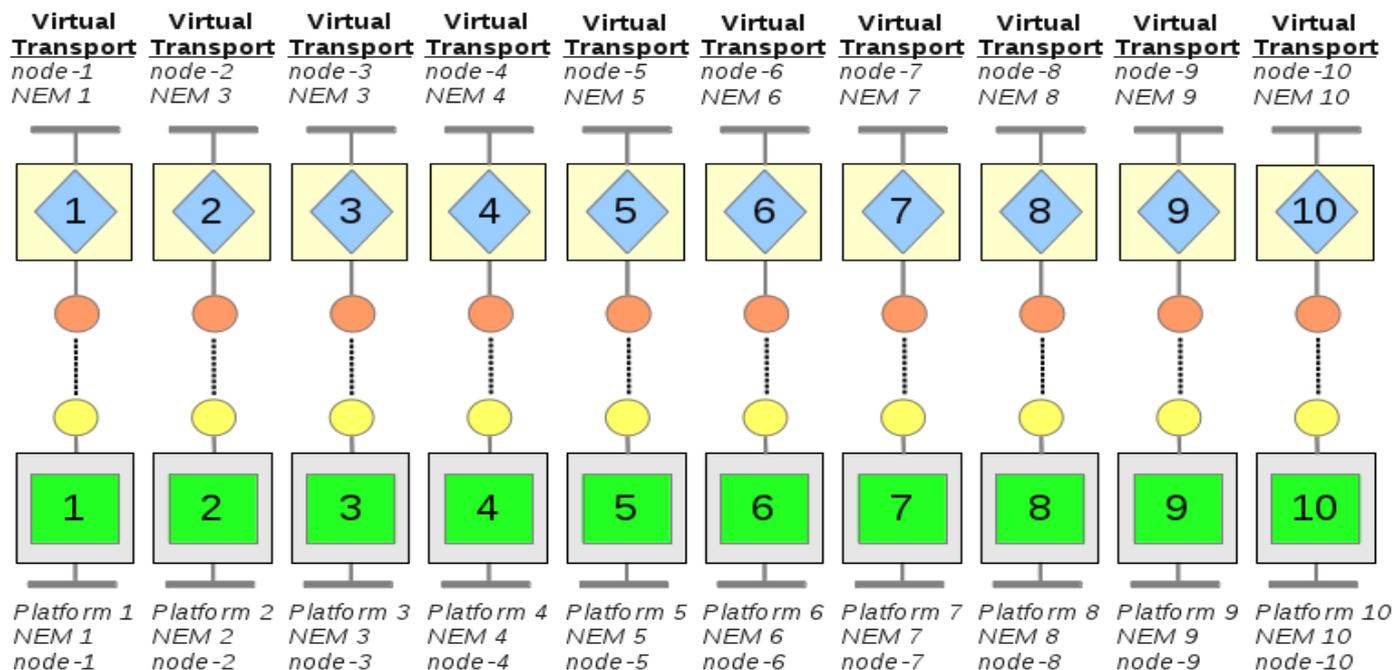
- Pathloss Event - Contains a variable length list of pathloss entries that are unique to each targeted NEM.
 - Entry consists of a transmitter NEM Id and the pathloss to/from the transmitter NEM with respect to the targeted NEM.
- Location Event - Contains a viable length list of location entries that update the GPS location of one or more NEMs.
 - Entry consists of an NEM Id and the latitude, longitude, and altitude associated with that NEM.
- Comm Effect Event - Contains a variable length list of communication effect entries that are unique to each targeted NEM.
 - Entry consists of a transmitter NEM Id and the latency, jitter, loss, duplication rate, unicast bitrate, and broadcast bitrate associated with packets received from the transmitter NEM to the target NEM.

Event Types

- Antenna direction Event - Contains a variable length list of antenna direction entries that update the antenna direction of one or more NEMs.
 - Entry consists of an NEM Id and the antenna elevation, azimuth, beam width elevation, and beam width azimuth associated with that NEM.

Demonstration 3

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





XML Configuration

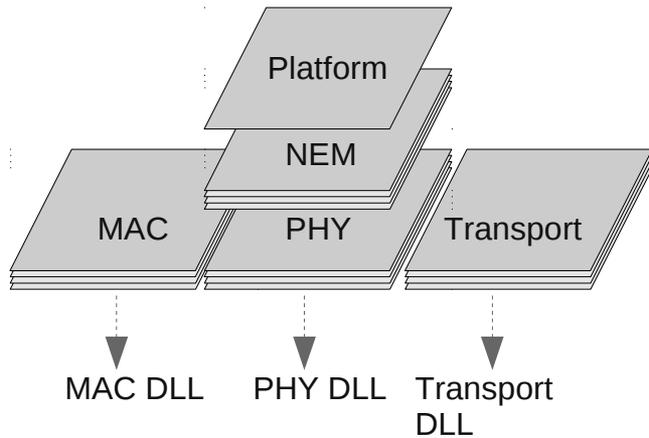


Layered XML Configuration

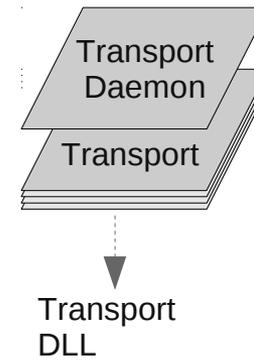
- EMANE uses a generic XML configuration design.
 - All components are capable of specifying any number of configuration parameters using a generic syntax.
 - Parameter/Value pairs are made accessible to their respective components via a configuration API.
- XML is layered to allow tailoring of lower levels of configuration to simplify deployment and promote reuse.
 - Complex components are created by combining XML definitions.
- Configuration parameters do not need to be present in the XML if they are not required.

EMANE XML Hierarchy

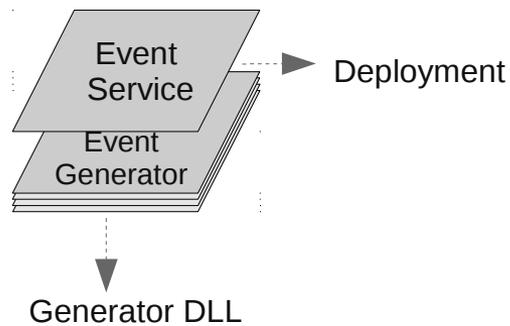
NEM Platform Server



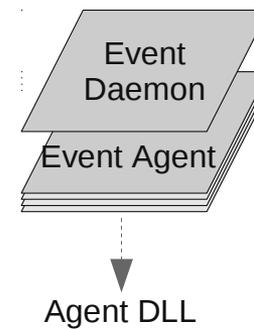
Transport Daemon



Event Service



Event Daemon



XML Layering Example

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

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

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

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

XML Layering Example

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

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

XML Layering Example

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

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

Automatic XML Generation

- The standard EMANE distribution provides two tools that can be used to automatically generate Transport Daemon XML and deployment XML:
 - *emanegentransportxml*
 - *emanegendeploymentxml*

Transport Grouping

- ***emanegentransportxml*** uses the Platform XML ***transport*** element's optional ***group*** attribute to group transports with matching values into a single Transport Daemon XML configuration.
 - Used when deploying multi-channel gateways
 - Used with Raw Transport instances when configuring EMANE as a black box emulator

Transport Grouping Example

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

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

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

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

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

Listing 5.6: Platform XML using Transport Grouping.

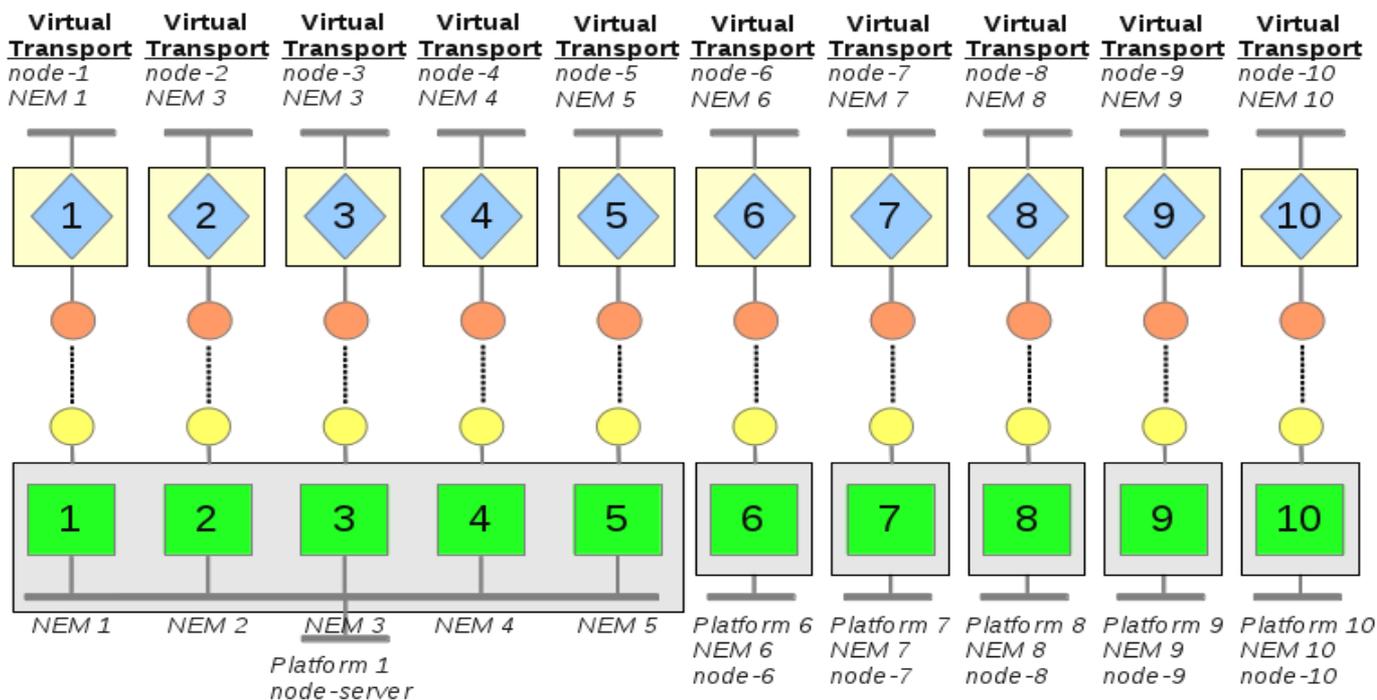
Transport Grouping Example

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

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

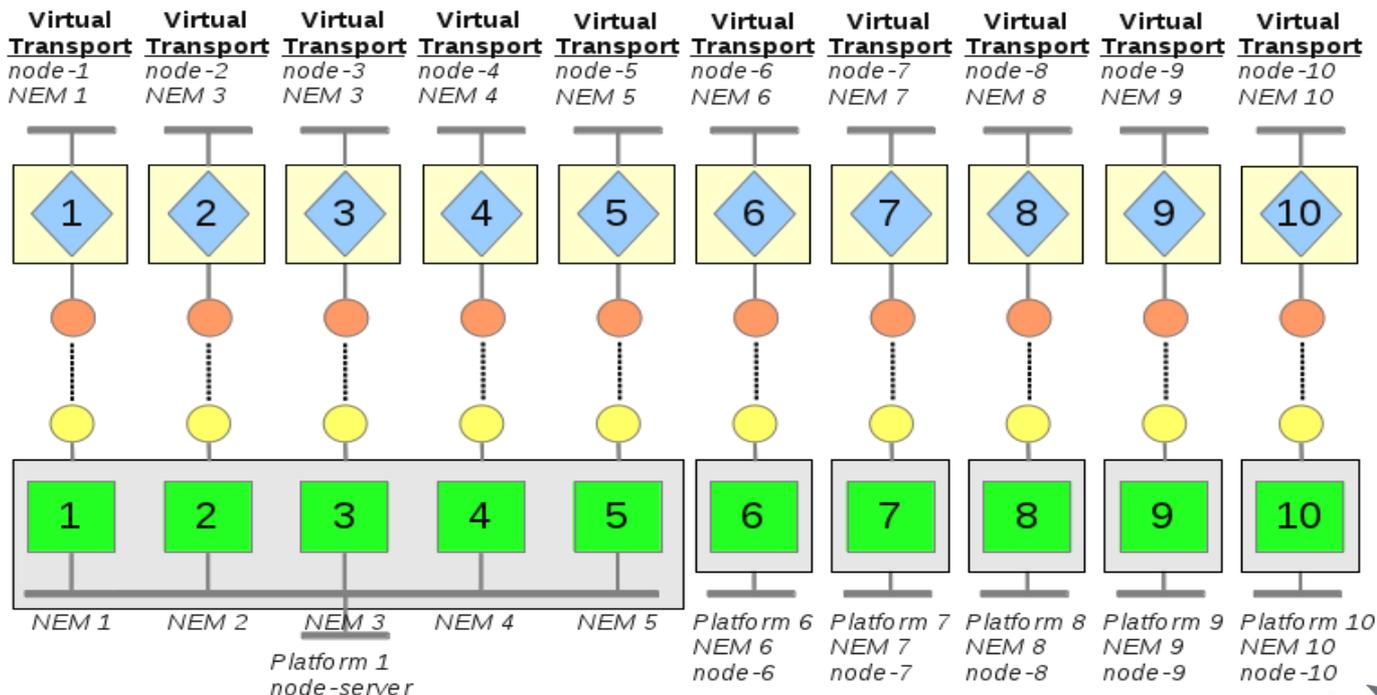
Demonstration 4

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



Demonstration 5

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



Deployment Debugging



NEM Platform Server Debug Port

- NEM Platform Server can be configured to enable an external debug port using the ***debugportenable*** parameter.
 - Debug Port is accessible via any telnet client.
 - Provides an interface to retrieve layer stack and statistic information for all NEMs contained in a platform.
 - The default debug port TCP port is 47000 and can be modified using the ***debugport*** parameter.

Debug Port Commands

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

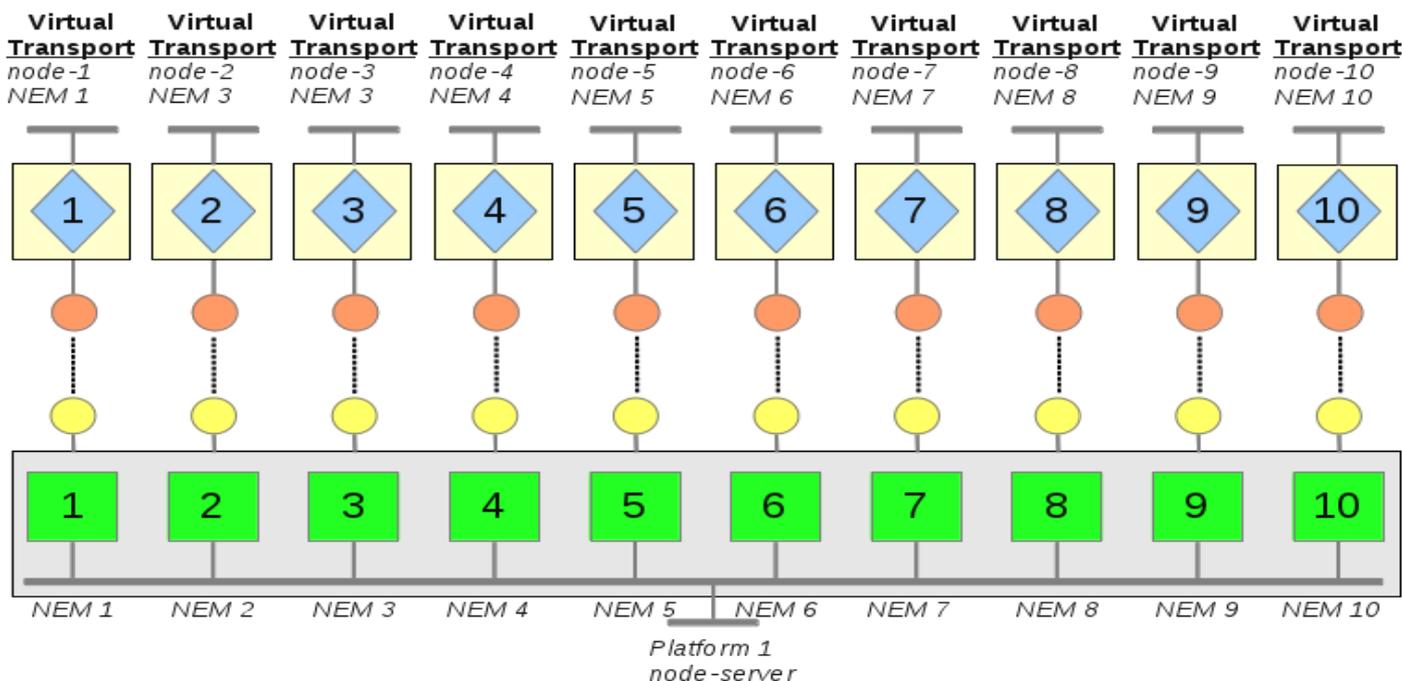
Logging

- All four EMANE container applications support logging: ***emane***, ***emanetransportd***, ***emaneservice*** and ***emaneeventd***.
- EMANE supports five log levels and logs can be directed to: ***stdout***, ***file***, ***syslog***, or ***ACE Log Server***.

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

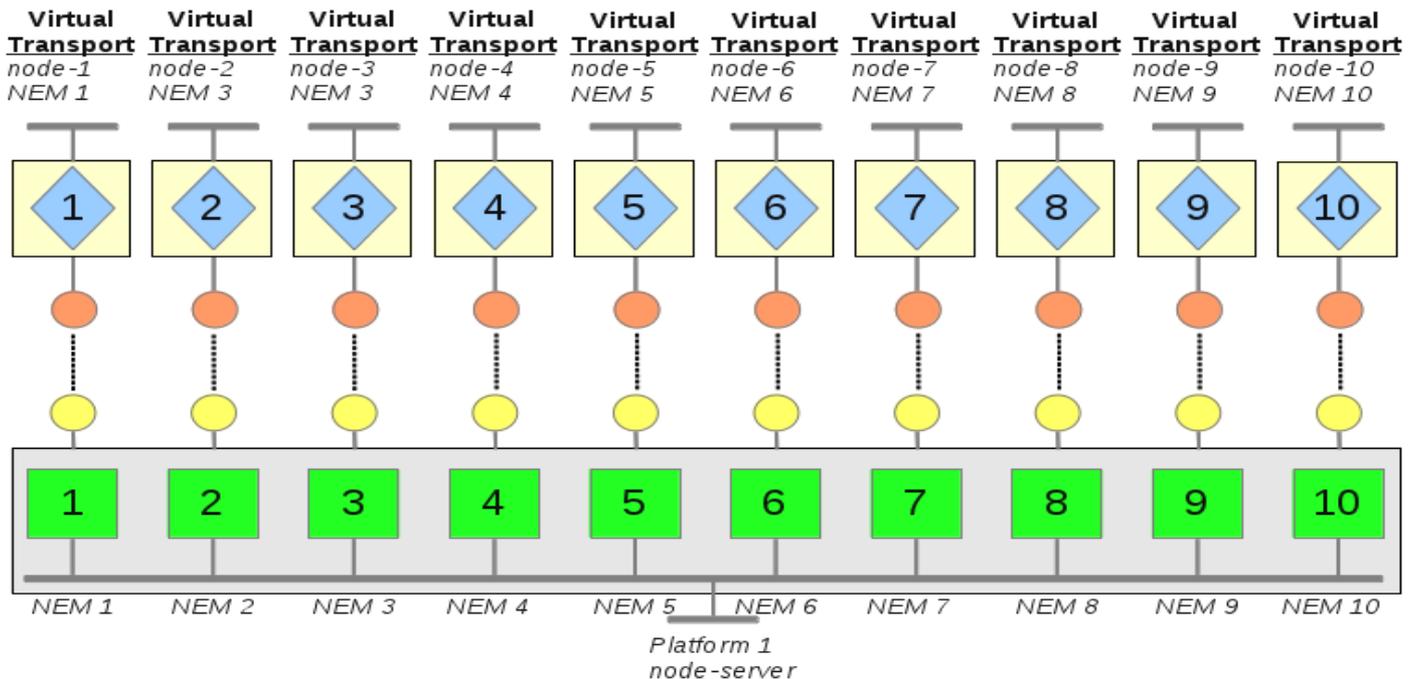
Demonstration 6

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



Demonstration 7

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



Universal PHY Layer

Universal PHY Layer

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

Pathloss Calculation

- Pathloss within the Universal PHY Layer is based on location or pathloss events.
- Pathloss is dynamically calculated based on location events when the ***pathlossmode*** configuration parameter is set to either ***2ray*** or ***freespace***
- Pathloss can be provided in realtime based on external propagation calculations using pathloss events. The ***pathlossmode*** configuration parameter should be set to ***pathloss*** to process pathloss events.

Receive Power Calculation

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

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

txPower - Packet Common PHY Header transmitter power

txAntennaGain - Packet Common PHY Header transmitter antenna gain

rxAntennaGain - Configuration parameter **antennagain**

pathloss - Pathloss between transmitter and receiver determined based on pathlossmode

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

$$rxSensitivity = -174 + noiseFigure + 10\log(\text{bandwidth})$$

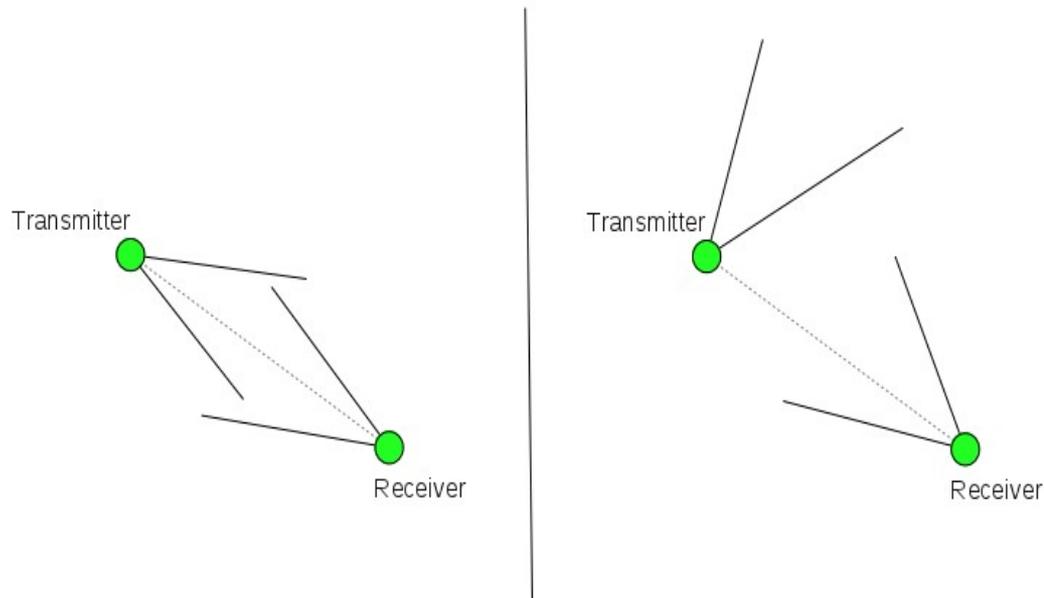
bandwidth - Configuration parameter **bandwidth**

noiseFigure - Configuration parameter **noisefigure**



Directional Antenna Support

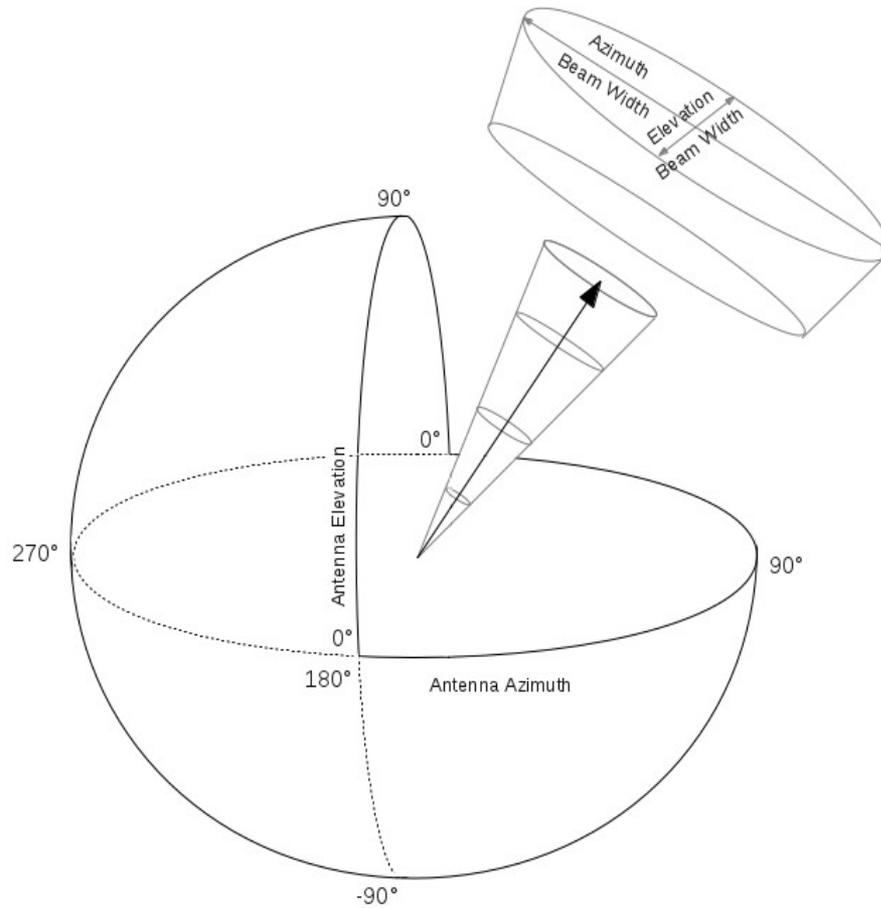
- Utilizes location events and Tx and Rx antenna information to determine if two nodes are visible
- Current directional antenna support is based on sector antennas, where a sector is defined by antenna azimuth and elevation beam width. Any intersection between the transmitting and receiving antenna will apply full gain.



Directional Antenna Support

- Provides ability to assign the directional antenna pointing and profile via three mechanisms.
 - Static from XML configuration
 - MAC Layer control message on a per packet basis
 - Antenna Direction events
- Four configuration items are used to define the pointing and profile characteristics of a directional antenna:
 - *antennaazimuth*
 - *antennaelevation*
 - *antennaazimuthbeamwidth*
 - *antennaelevationbeamwidth*

Directional Antenna Support



Noise Processing

- Assess the impact of intentional and unintentional noise sources within the emulation by adjusting the noise floor.
 - Achieved by summing the energy of interferers within the appropriate frequency of interest over a given time interval and adjusting the noise floor accordingly when a valid packet is received.
- Only computes interference for out-of-band packets
 - out-of-band packet is one which is not from the same emulated waveform.
 - Waveform type determined by comparing the PHY Registration Id, center frequency, and Universal PHY Layer subid of each packet
- MAC Layer implementation responsibility to account for in-band interference.

MAC-PHY Control Messaging

- Provides a control API on a per packet basis for every transmit (Tx) packet received from the MAC Layer for over-the-air transmission and every received (Rx) over-the-air packet sent to the MAC Layer for processing.
- The Tx Control API provides the MAC Layer with the ability to override default PHY Layer configuration for transmit power, message duration, transmit frequency and antenna pointing.
 - Utilizes the data from the Tx Control message to populate the Common PHY Header.
- The Rx Control API provides the MAC Layer with the appropriate receive information.
 - Receive power, noise floor, message duration, propagation delay and receive frequency.
 - Allows for performing SINR based packet completion calculation, in-band collision detection and channel access protocols.

Universal PHY Layer Configuration

- ***bandwidth*** - Defines the center frequency bandwidth in KHz. This is used to compute the receiver sensitivity and is also included in all over-the-air transmissions to support noise processing calculations.
- ***antennagain*** - Defines the antenna gain in dBi. This is used to compute the receive power associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets.
 - Can be overridden by a MAC Layer using the TX Control API.
- ***systemnoisefigure*** - Defines the system noise figure in dB. The system noise figure is used along with the ***bandwidth*** to compute the receiver sensitivity.

Universal PHY Layer Configuration

- ***frequencyofinterest*** - Defines a set of frequencies in KHz that the Universal PHY Layer will monitor.
 - Multiple frequencies can be monitored to support MAC Layer implementations with frequency agility or hopping capability.
 - Only packets received on a frequency of interest will be sent to the MAC Layer for processing provided waveform and receive power criteria are met.
 - Separate noise floor calculations are maintained for each frequency of interest.
- ***pathlossmode*** - Defines the pathloss mode of operation. The pathloss mode of operation determines whether pathloss or location events will be used as input to the Universal PHY Layer propagation functionality.
- ***noiseprocessingmode*** - Enables or disables noise processing. When ***on***, out of band packets (not of this waveform) within a frequency of interest will raise the noise floor accordingly.

Universal PHY Layer Configuration



- **defaultconnectivitymode** - Defines the default connectivity mode for pathloss.
 - When **on**, full connectivity will be engaged until a valid event (pathloss or location) is received based on the **pathlossmode** setting.
 - Any valid event of the appropriate type, regardless if it contains information for the receiving NEM, will disengage default connectivity mode.
 - Directional antenna functionality is bypassed when default connectivity mode is engaged.
 - When **off**, no connectivity is in effect until a valid event is received.
- **txpower** - Defines the transmit power in dBm. This is used to compute the receive power associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets.
 - Can be overridden by a MAC Layer using the TX Control API.

Universal PHY Layer Configuration

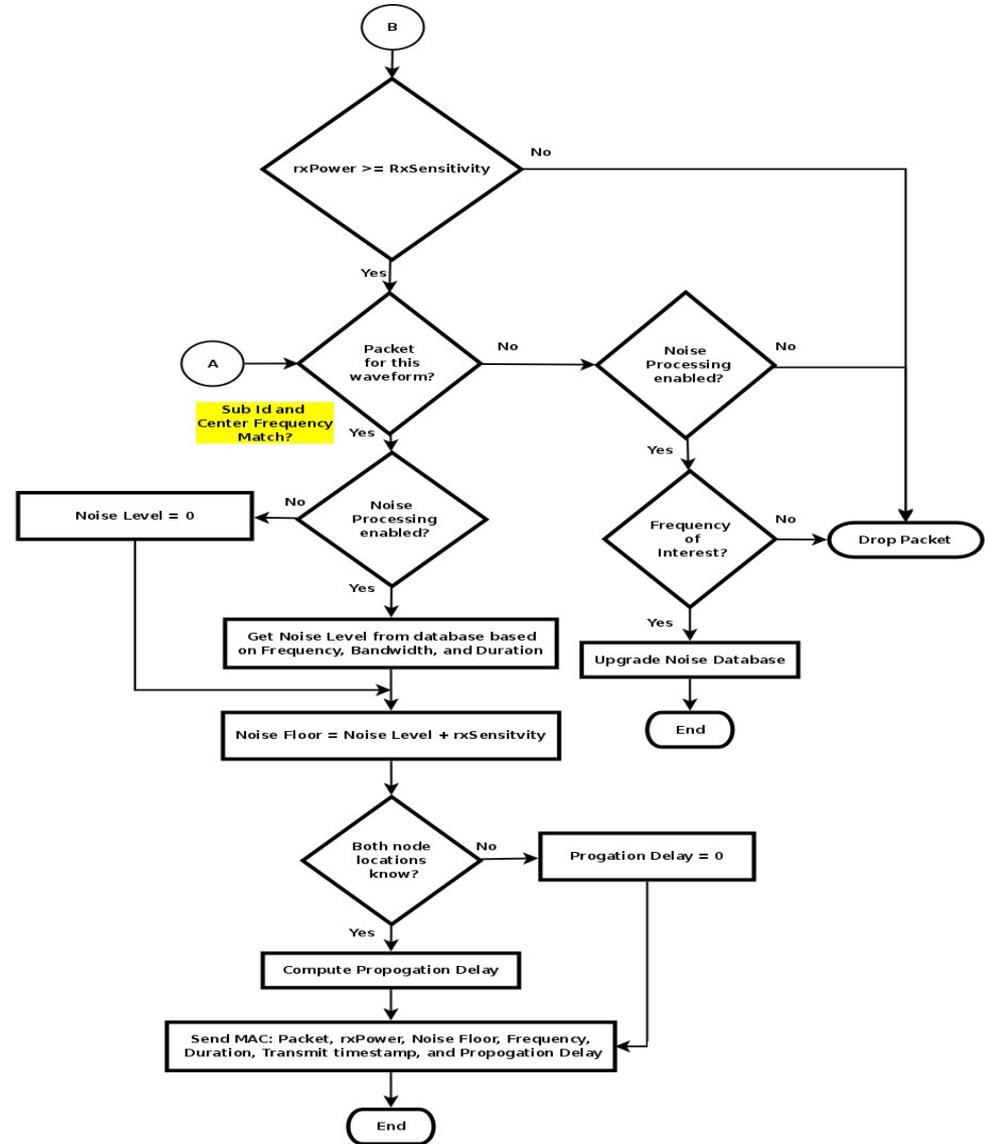
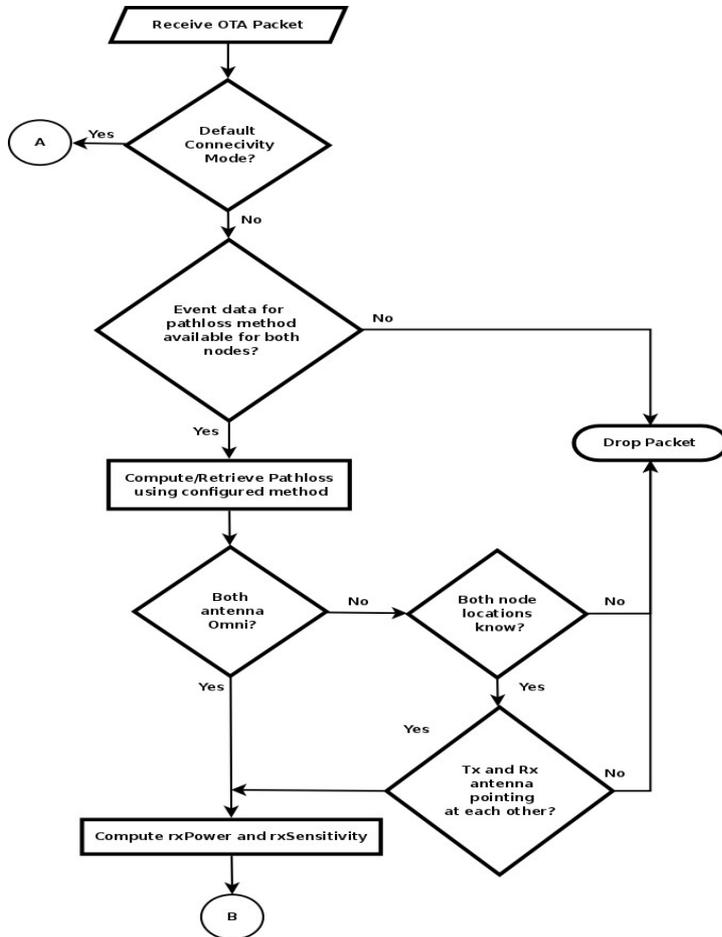
- **frequency** - Defines the transmit center frequency in KHz. This value is included in the Common PHY Header of all transmitted OTA packets.
 - Can be overridden by a MAC Layer using the TX Control API.
- Directional Antenna Configuration - Used in the directional antenna pointing computation associated with an OTA packet and included in the Common PHY Header of all transmitted OTA packets when the **antennatype** is **unidirectional**. All values can be overridden by a MAC Layer using the TX Control API and/or via antenna pointing events.
 - **antennaazimuthbeamwidth** - Defines the antenna azimuth beam width in degrees.
 - **antennaelevationbeamwidth** - Defines the antenna elevation beam width in degrees.
 - **antennaazimuth** - Defines the antenna azimuth degrees.
 - **antennaelevation** - Defines the elevation azimuth degrees.



Universal PHY Layer Configuration

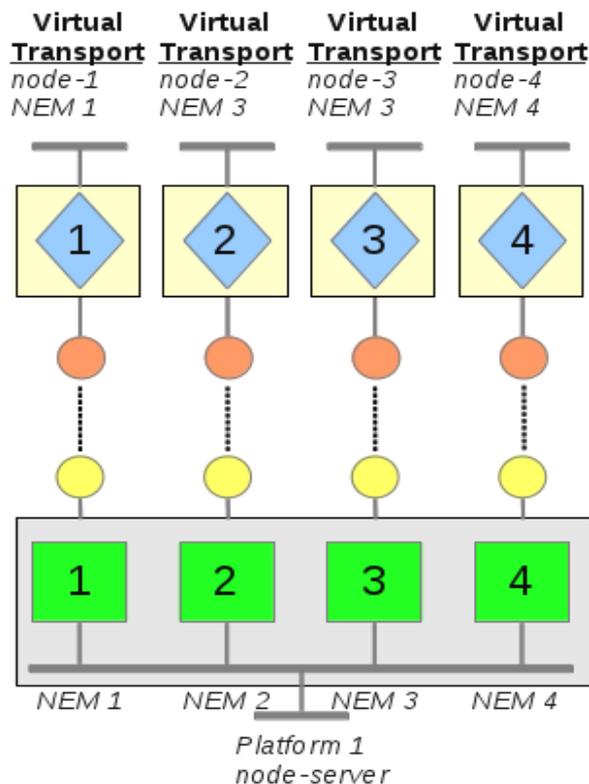
- **antennatype** - Defines the antenna type. If the antenna type is **unidirectional**, location event information for both the transmitter and receiver are required as inputs to the Universal PHY Layer for receive packet processing unless default connectivity mode is engaged.
- **subid** - Defines the Universal PHY Layer subid. The Universal PHY Layer is used by multiple NEM definitions. In order to differentiate between Universal PHY instances for different waveforms, the **subid** is used as part of the unique waveform identifying tuple: PHY Layer Registration Id, Universal PHY subid and packet center frequency.

Universal PHY Upstream Packet Flow

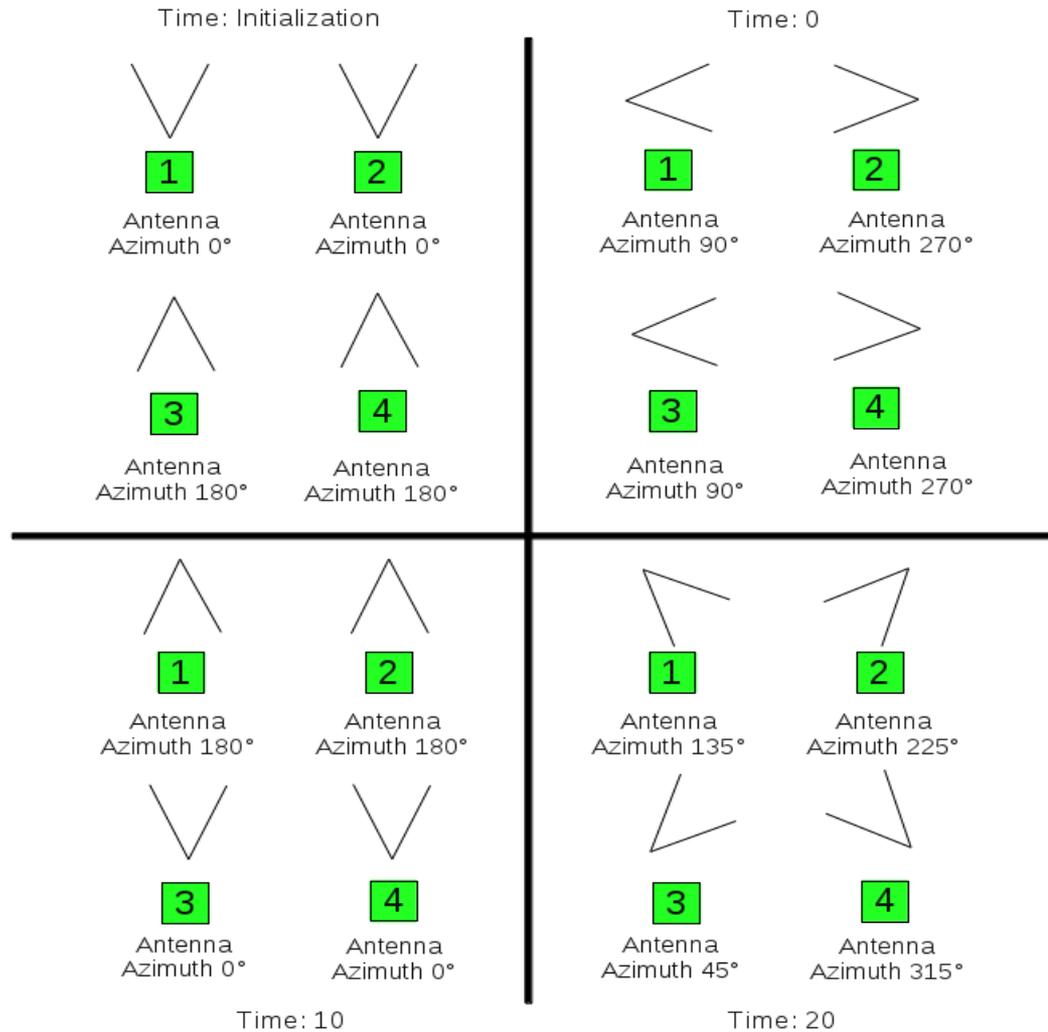


Demonstration 8

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



Demonstration 8



RF Pipe MAC Layer

RF Pipe MAC Layer

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

transmissionDelay = messageDuration + delay + jitter + propagationDelay

- *delay* - Configuration parameter **delay**
- *jitter* - configuration parameter **jitter**
- *messageDuration* - Provided by the transmitter via the Common PHY Header
- *propagationDelay* - Provided by the Universal PHY Layer when node positions are available via location events

RF Pipe MAC Layer

- Use of user defined Packet Completion Rate (PCR) curves as a function of SINR.
 - RF Pipe MAC Layer does not apply any additional interference effects and as such, the use of negative SINR values within the PCR Curve file is valid only when noise processing is enabled within the Universal PHY Layer to raise the noise floor above the inherent receiver sensitivity.

RF Pipe MAC Layer Configuration

- ***enablepromiscuousmode*** - Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.
- ***enabletighttiming*** - Determines if the over-the-air time *rxTime* - *txTime* should be included in the overall packet delay time. Implies that the source and destination are in tight time sync.
- ***transmissioncontrolmap*** - Defines the data rate, frequency, and power level to be used by the PHY Layer for all transmissions to a specified node. When a packet is transmitted to the destination NEM, the MAC Layer will send an accompanying control message to the PHY Layer that will cause the specified data rate, frequency and transmit power to be included in the Common PHY header.
- ***datarate*** - Defines the data/burst rate in Kbps of the waveform being emulated. It is used on the transmit side (downstream) to compute transmission delay based on the packet size and data rate. The RF Pipe MAC Layer will wait for the message delay to expire before transmitting another packet.

RF Pipe MAC Layer Configuration

- **delay** - Defines the delay in microseconds that is to be included in the transmission delay. The delay is added to the delay introduced by the `datarate` parameter.
- **jitter** - Defines the jitter in microseconds to be included to the transmission delay. The jitter will be computed for each packet transmission based on uniform random distribution between +/- the configured jitter value.
- **pcrcurveuri** - Defines the absolute file name that contains the SINR/PCR curve values. A minimum of one SINR/PCR pair is required, $POR=0.0$ and $POR=100.0$. Entries shall be in unique ascending order with up to two decimal places of precision for SINR. The PCR values shall represent the percentage with up to two decimal places of precision.

RF Pipe MAC Layer Configuration



- **flowcontrolenable** - Enables downstream traffic flow control with Virtual Transport.
 - Only valid when using the Virtual Transport
 - Setting to either **on** or **off** must match the setting of **flowcontrolenable** within the Virtual Transport configuration.
- **flowcontroltokens** - Defines the number of flow control tokens. This is an optional parameter used to override the default token setting when **flowcontrolenable** is **on**.

Packet Completion Rate

- The RF Pipe Packet Completion Rate is specified as a curve defined via XML.
 - Curve definition comprised of a series of SINR values along with their corresponding probability of reception.
 - Linear interpolation is preformed when an exact SINR match is not found.
- Specifying a packet size in the curve file will adjust the POR based on received packet size. Specifying a **pktsize** of 0 disregards received packet size when computing the POR.

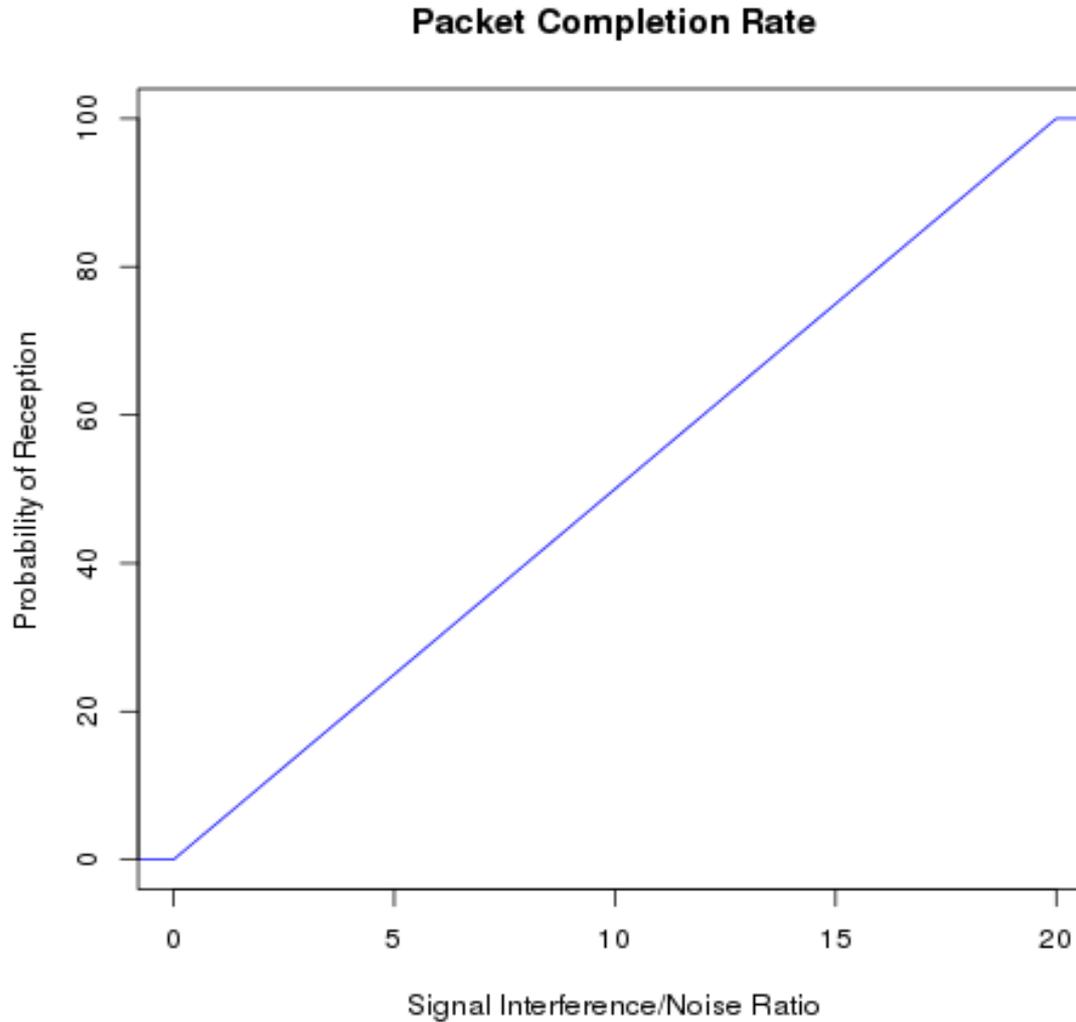
$$POR = POR_0^{S_1/S_0}$$

POR_0 - POR value determined from the PCR curve for the given SINR value

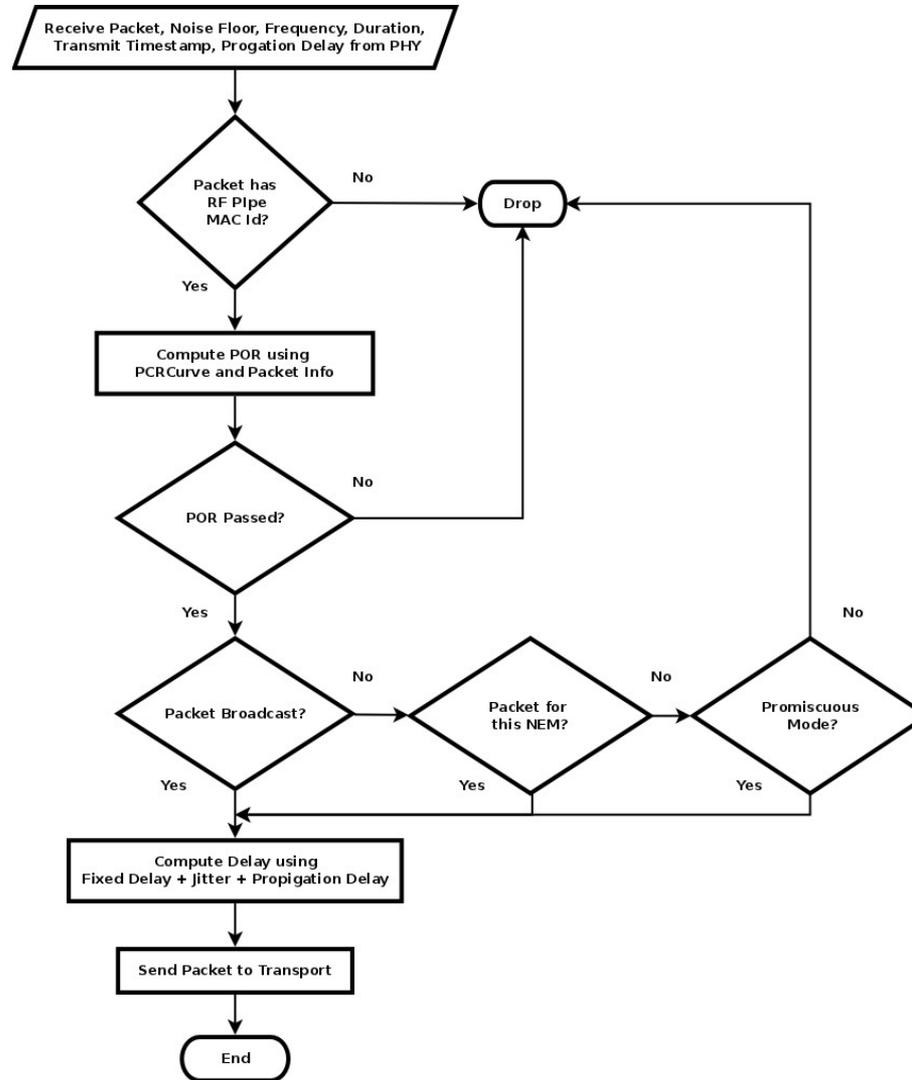
S_0 - Packet size specified in the curve file

S_1 - Received packet size

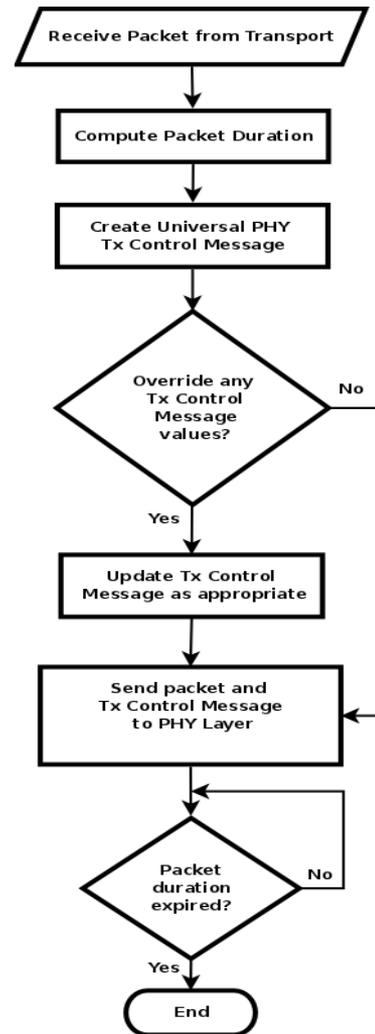
Packet Completion Rate



RF Pipe Upstream Packet Flow

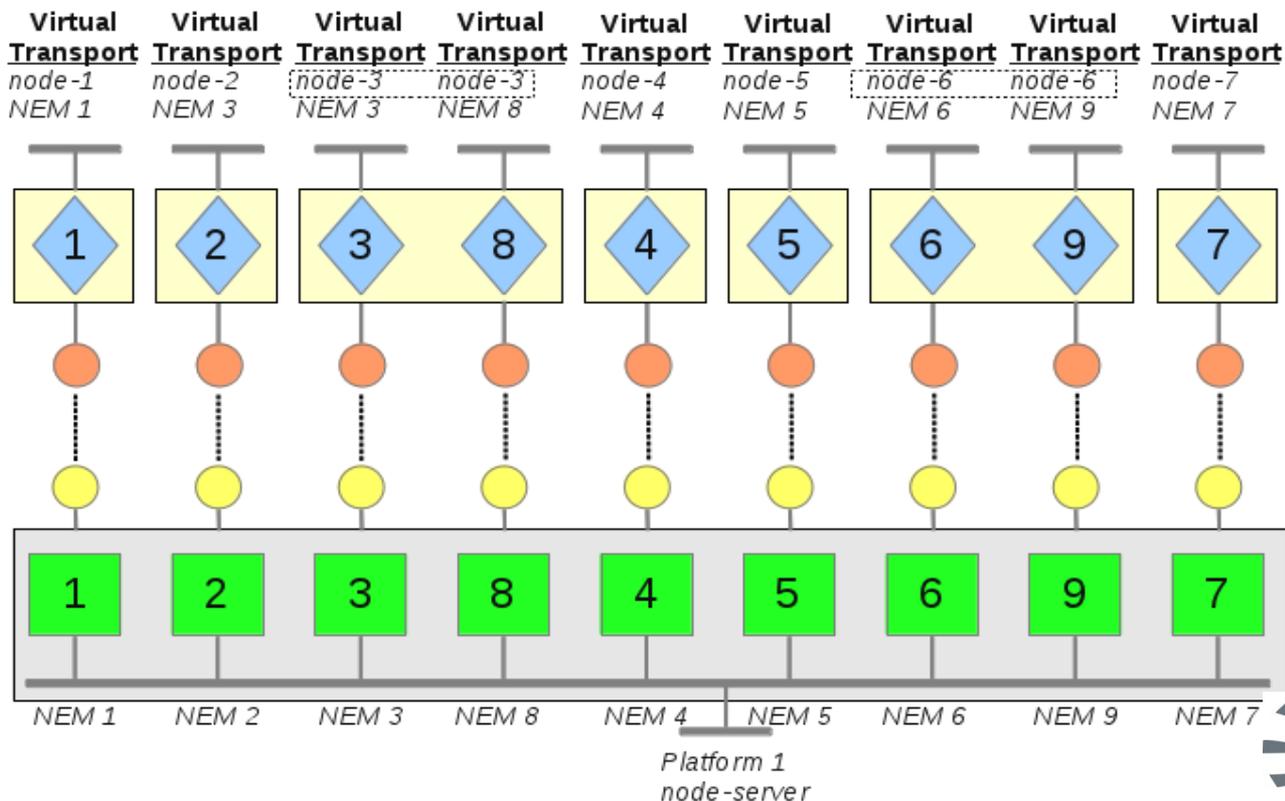


RF Pipe Downstream Packet Flow

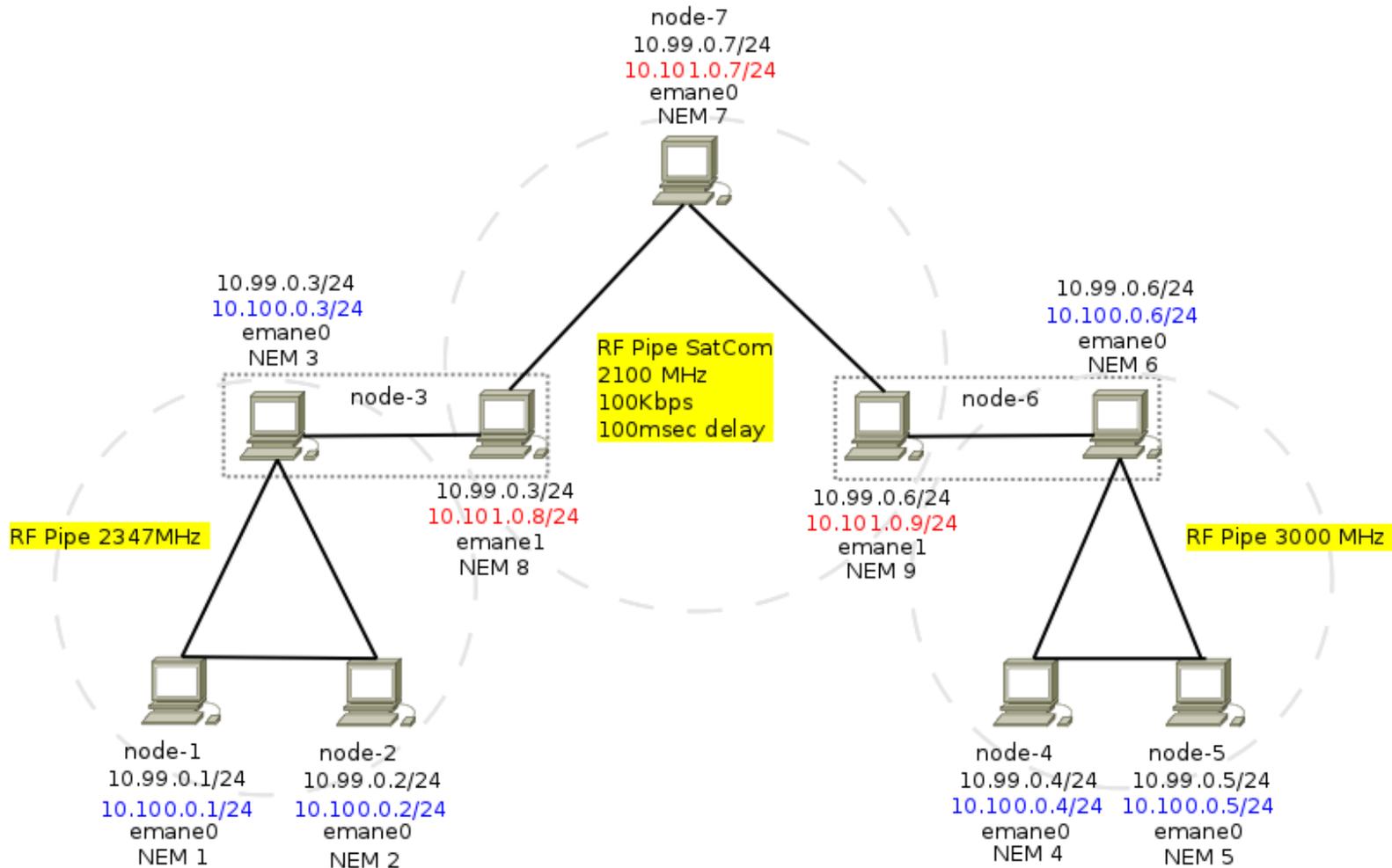


Demonstration 9

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



Demonstration 9



IEEE 802.11abg MAC Layer



IEEE 802.11abg MAC Layer

- Supports flow control with the Virtual Transport
- Supports the following waveform modes and data rates with the appropriate timing:
 - 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)
- Supports only the DCF channel access function. PCF and beacon transmissions are not supported.
- Supports both unicast and broadcast transmissions.
 - Unicast transmissions include the ability to emulate control message (RTS/CTS) behavior as well as retries without actually transmitting the control messages or the re-transmission of the data message.
 - The emulation of unicast does not replicate exponential growth of the contention window as a result of detected failures.



IEEE 802.11abg MAC Layer

- Supports Wi-Fi multimedia (WMM) capabilities.
 - Initial implementation supports the ability to classify four different traffic classes (Background, Best Effort, Video and Voice) where the higher priority classes (voice and video) are serviced first.
- Supports user defined Packet Completion Rate (PCR) curves as a function of SINR.
- Default curves are provided for each of the supported 802.11 modulation and data rate combinations.
 - Default curves are based on theoretical equations for determining Bit Error Rate (BER) in an Additive White Gaussian Noise (AWGN) channel.
- Adjusts the interference on a per packet basis based on detected collisions and as such supports negative SINR values as can be seen in the default curves.

IEEE 802.11abg MAC Layer Configuration

- **mode** - Defines the 802.11 mode.
- **enablepromiscuousmode** - Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.
- **distance** - Defines the maximum distance in meters for supported point to point links within the network. This is used to adjust the slot timing to account for round trip propagation delays.

$$\text{slotTime} = \text{fixedSlotTime} + \text{propagationTime}$$

$$\text{fixedSlotTime} = 9$$

$$\text{propagationTime} = \text{distance}/300$$

- **unicastrate** - Defines the data rate index to be used for all unicast transmissions. The rate selection must be valid for the **mode** selected.
- **multicastrate** - Defines the data rate index to be used for all multicast transmissions. The rate selection must be valid for the **mode** selected.

IEEE 802.11abg MAC Layer Configuration

- ***rtsthreshold*** - Defines the minimum packet size in bytes required to trigger RTS/CTS for unicast transmissions.
 - A value of **0** disables RTS/CTS.
 - The effect of RTS/CTS for unicast transmissions is emulated using a statistical model
- ***wmmenable*** - Provides the ability to enable the WiFi Multimedia (WMM) type feature.
 - Current capability supports the ability to service packets from a higher priority queue first and does not yet support the internal contention based logic as defined by 802.11e.
- ***pcrcurveuri*** - Defines the absolute file name that contains the SINR/PCR curve values. A minimum of two SINR/PCR row entries per data rate are required, *POR=0.0* and *POR=100.0*. Entries shall be in unique ascending order with up to two decimal places of precision for SINR. The PCR values shall represent the percentage with up to two decimal places of precision.

IEEE 802.11abg MAC Layer Configuration



- **flowcontrolenable** - Enables downstream traffic flow control with Virtual Transport.
 - Only valid when using the Virtual Transport
 - Setting to either **on** or **off** must match the setting of **flowcontrolenable** within the Virtual Transport configuration.
- **flowcontroltokens** - Defines the number of flow control tokens. This is an optional parameter used to override the default token setting when **flowcontrolenable** is **on**.
- **queuesize** - Defines the size of the queue for the given access category. When **wmmenable** is off only access category **0** is used.
- **cwmin** - Defines the minimum contention window in slots for the appropriate access category. This value is used when calculating the overall packet duration. When **wmmenable** is off only access category **0** is used.



IEEE 802.11abg MAC Layer Configuration

- ***cwmax*** - Defines the maximum contention window in slots for the appropriate access category. Not currently used for point-to-point failure exponential growth. When ***wmmenable*** is ***off*** only access category **0** is used.
- ***aifs*** - Defines the Arbitration Inter Frame Space (AIFS) time factor in slots for the appropriate access category. When ***wmmenable*** is ***off*** only access category **0** is used.
 - The inter frame space time in microseconds is computed as follows:
$$time = aifs * slotduration + sifs$$

slotduration is a function of distance.

- *sifs* is a function of the 802.11 mode.

IEEE 802.11abg MAC Layer Configuration

- ***txop*** - Defines the maximum time in microseconds a packet can reside within the queue for a given access category. When ***wmmenable*** is ***off*** only access category **0** is used.
 - 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.
- ***retrylimit*** - Defines the number of retries permitted for the unicast messages for the appropriate access category. When ***wmmenable*** is ***off*** only access category **0** is used.

Packet Completion Rate

- Packet Completion Rate is specified as curves defined via XML.
 - The curve definitions are comprised of a series SINR values along with their corresponding probability of reception.
 - A curve definition must contain a minimum of two points with one SINR representing POR=0 and one SINR representing POR=100.
 - Linear interpolation is preformed when an exact SINR match is not found.
- Specifying a packet size in the curve file will adjust the POR based on received packet size. Specifying a pktsize of 0 disregards received packet size when computing the POR.

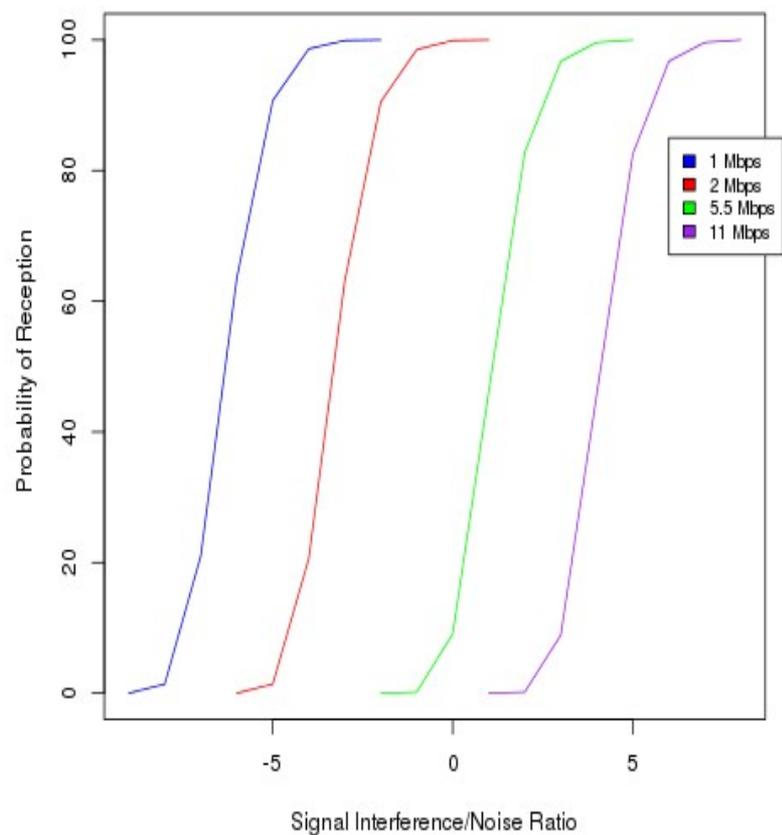
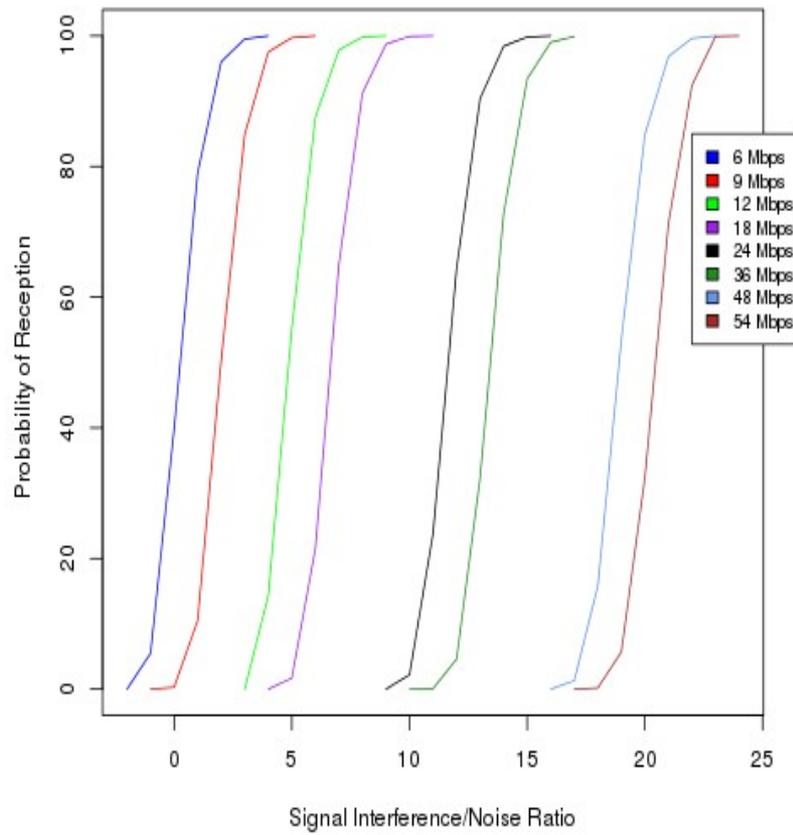
$$POR = POR_0^{S_1/S_0}$$

POR_0 - POR value determined from the PCR curve for the given SINR value\

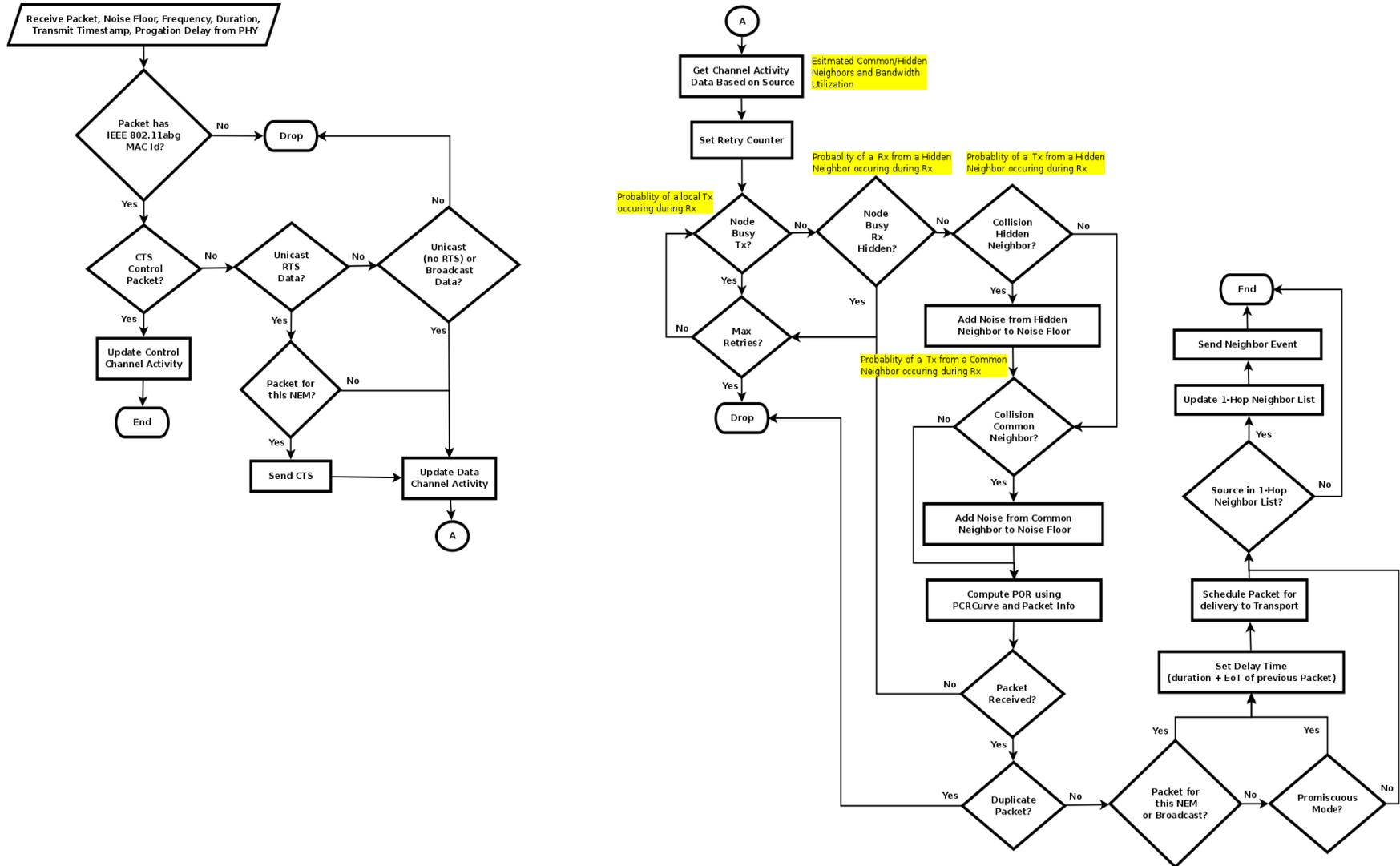
S_0 - Packet size specified in the curve file

S_1 - & Received packet size

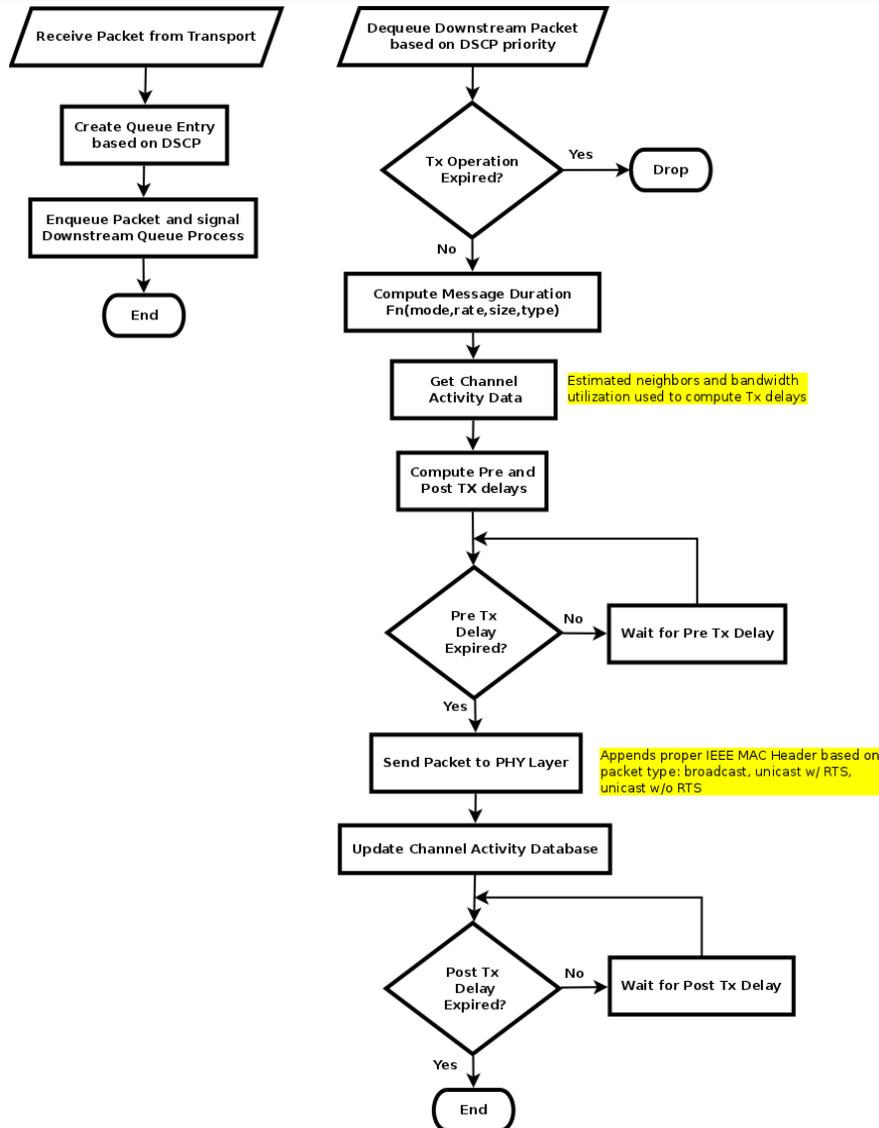




IEEE 802.11abg Upstream Packet Flow

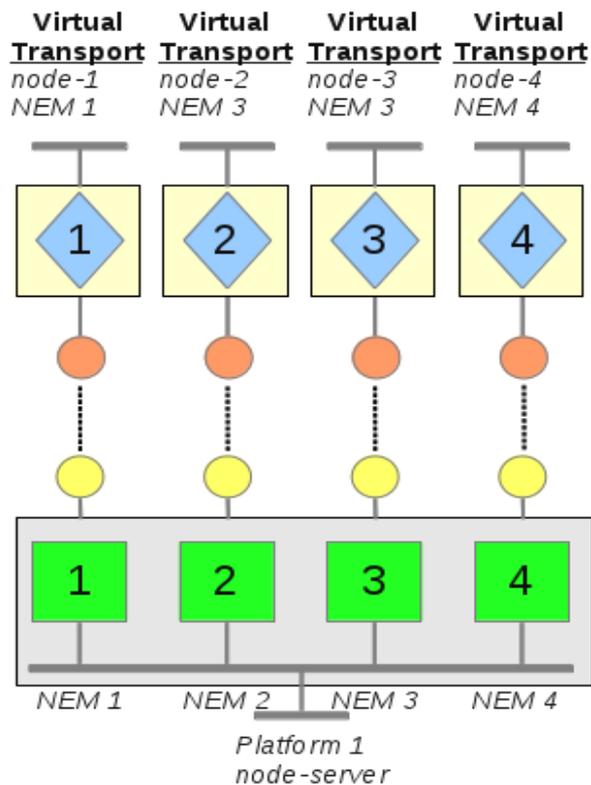


IEEE 802.11abg Downstream Packet Flow



Demonstration 10

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



Comm Effect Shim Layer



Comm Effect Shim Layer

- The Comm Effect Shim Layer provides the ability to define the following network impairments:
 - Loss - Percentage of packets that will be dropped utilizing a uniform loss distribution model.
 - Latency - 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.
 - Duplicates - Percentage of packets that will be duplicated at the receiver.
 - Unicast Bitrate: Bitrate for packets destined for the NEM or handled in promiscuous mode.
 - Broadcast Bitrate: Bitrate for packets destined for the NEM broadcast address.
- The network impairments can be controlled via two mechanisms:
 - Comm Effect Events
 - Static filter based impairments



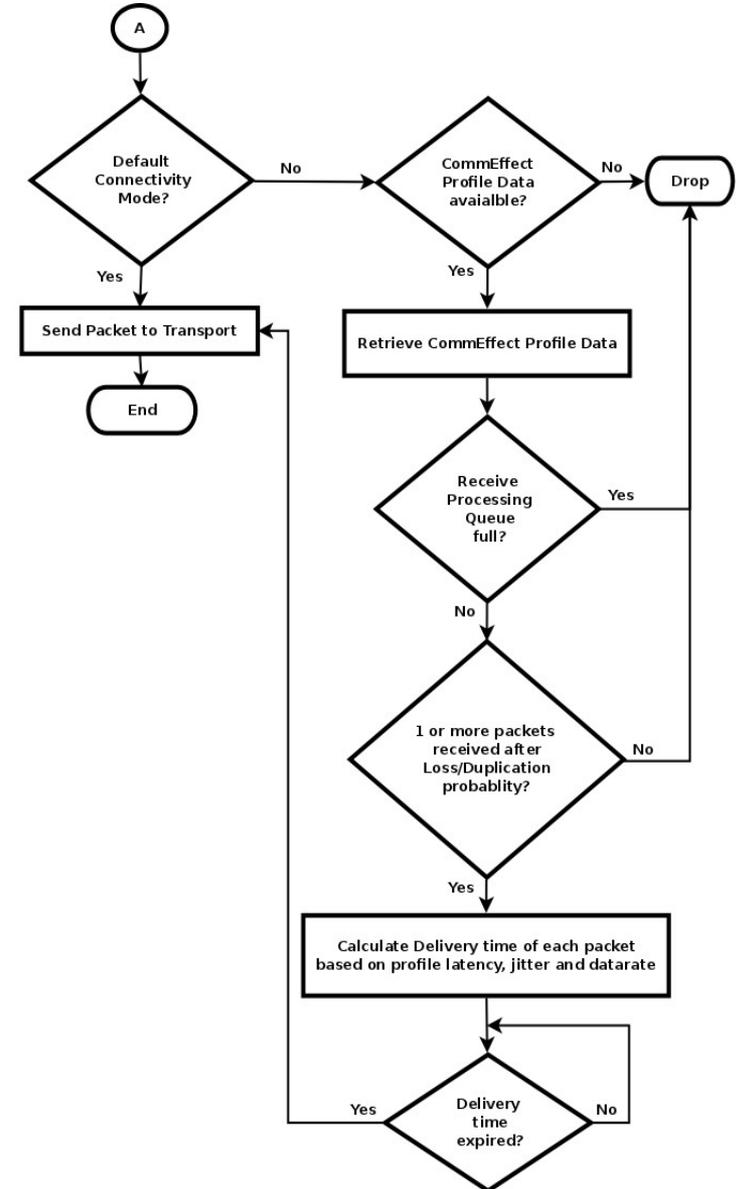
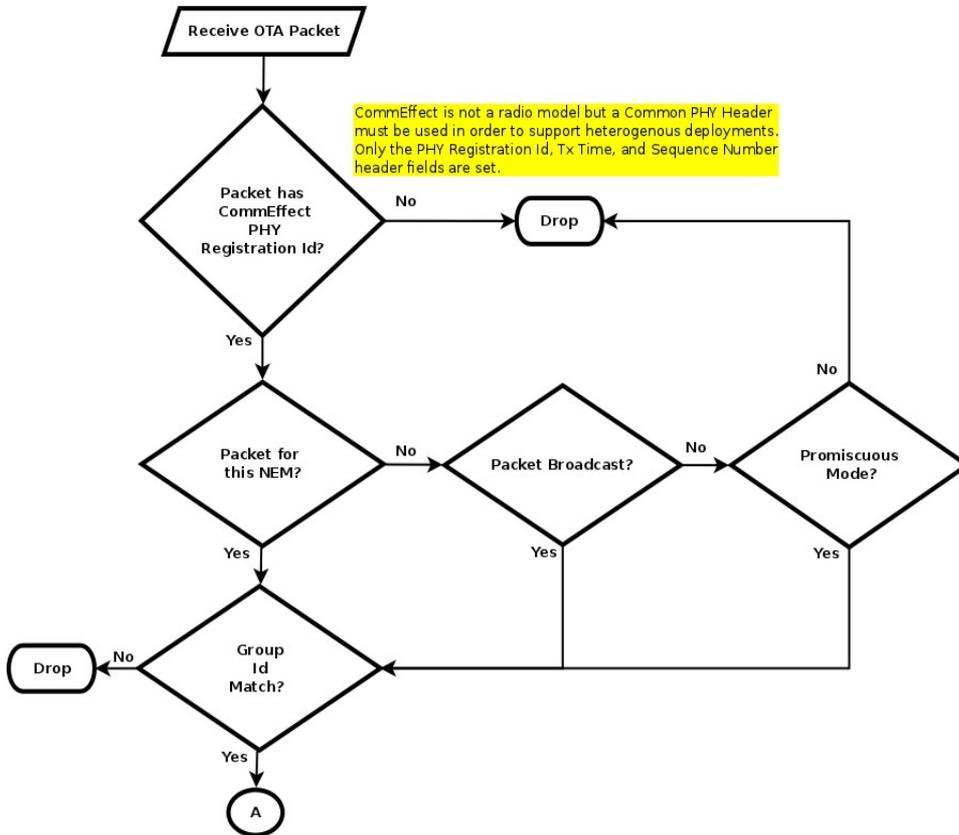
Comm Effect Shim Layer Configuration

- ***defaultconnectivity*** - Defines the default communication effects of all NEMs at start up prior to receiving any Comm Effect events. All filter rules, if any, are still processed regardless of whether ***defaultconnectivity*** is in use.
- ***filterfile*** - The absolute file name of the Comm Effect filter file to load.
- ***groupid*** - Defines the NEM Group Id which will be used to group NEMs by the assigned Id value. When an NEM is assigned to a group it can only receive traffic from other members of the same group regardless of communication effects.
 - If set to **0**, the NEM is not associated with an NEM Group.
 - If set greater than **0**, the NEM is associated with the NEM Group of the same value.
- ***enablepromiscuousmode*** - Determines if all packets received over-the-air will be sent up the stack regardless of the destination NEM Id.

Comm Effect Shim Layer Configuration

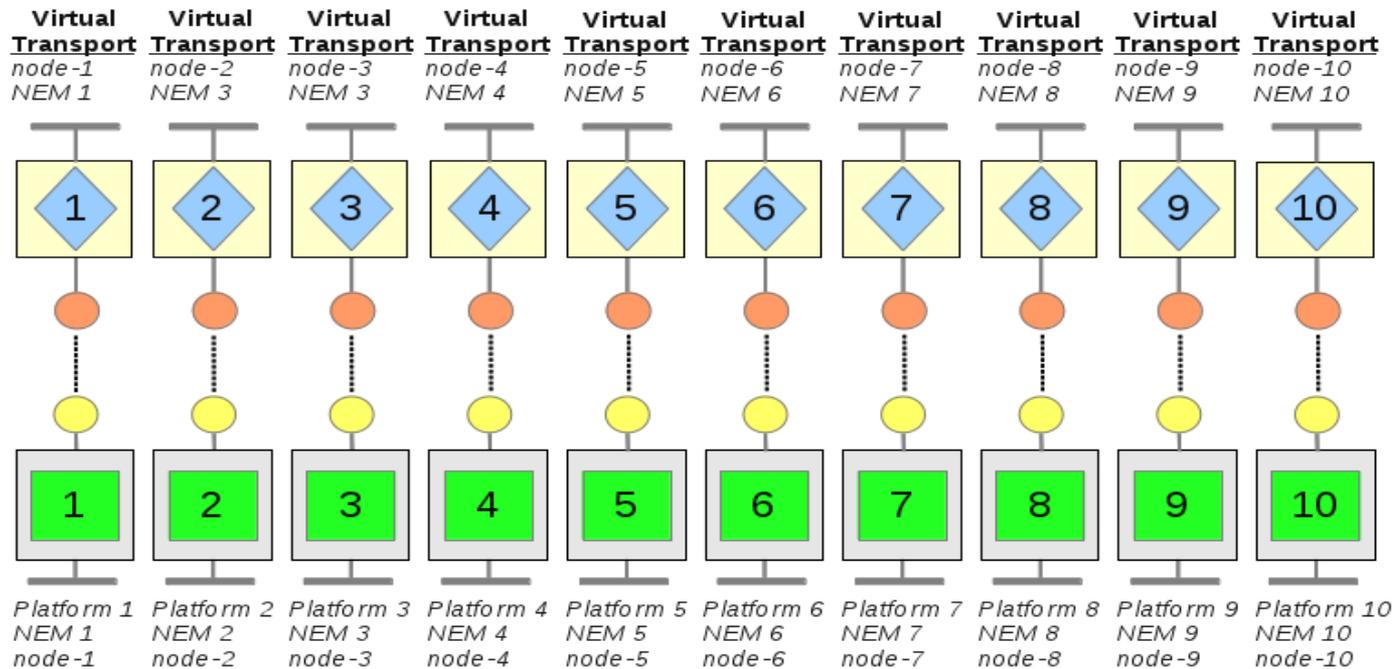
- ***enabletighttimingmode*** - Determines whether transmission time of the packet will be factored in when calculating delivery scheduling time.
- ***receivebufferperiod*** - Specify the max sum of buffering time in seconds for packets received from an NEM.
 - The buffering interval for each packet is determined by the bitrate for the source NEM and packet size.
 - Packets are placed in a timed queue based on this interval.
 - 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.

Comm Effect Shim Upstream Packet Flow



Demonstration 11

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





Virtual Transport



Virtual Transport

- Creates a virtual interface for use as the emulation/application domain boundary.
- IPv4 and IPv6 Capable - Supports IPv4 and IPv6 virtual interface address assignments and packet processing.
- Flow Control - Supports flow control with a corresponding flow control capable NEM layer in order to provide feedback between the emulation stack and application domain socket queues.
- Virtual Interface Management - Supports configuring virtual interface addresses or can be configured to allow virtual interfaces to be managed externally, for example via DHCP.
- Raw Transport Interoperability - Supports interoperability with Raw Transport emulation/application domain boundaries using ARP caching to learn network/NEM Id associations.



Virtual Transport

- Bitrate Enforcement - Supports bitrate enforcement for use with models that do not limit bitrate based on emulation implementation.
- Broadcast Only Mode - Supports forced NEM broadcasting of all IP packet types: unicast, broadcast and multicast.

Virtual Transport Configuration

- ***address*** - Virtual device address. Supports IPv4 and IPv6.
- ***arpcheenable*** - Enable ARP request/reply monitoring to map ethernet address to NEM.
- ***arpmode*** - Enable ARP on the virtual device.
- ***bitrate*** - Transport bitrate in Kbps. This is the total allowable throughput for the transport combined in both directions (upstream and downstream). A value of **0** disables the bitrate feature.
- ***broadcastmode*** - Broadcast all packets to all NEMs.
- ***device*** - Virtual device name.
- ***devicepath*** - Path to the tap device.

Virtual Transport Configuration

- ***flowcontrolenable*** - Enables downstream traffic flow control with a corresponding flow control capable NEM layer. The ***flowcontrolenable*** parameter value must match the setting of the corresponding NEM layer's ***flowcontrolenable*** parameter.
- ***mask*** - Virtual device network mask. Supports IPv4 and IPv6.

Flow Control

- Supports flow control using a token based exchange mechanism performed in coordination with a corresponding NEM layer.
 - flow control token - Packet transmission unit where a single token represents permission for the transport to transmit a single packet downstream to a coordinating NEM layer.
 - Flow control must be enabled on both the transport and the coordinating flow control capable NEM layer
- Number of tokens available is specified by the coordinating flow control capable NEM layer
- Flow control enabled layer sends a control message to the Virtual Transport specifying the number of tokens available and then waits for the transport to acknowledge receipt of the token count.
 - Any downstream packets received from the transport in the period between when the token count control message is sent and the acknowledgment is received are discarded.

Flow Control

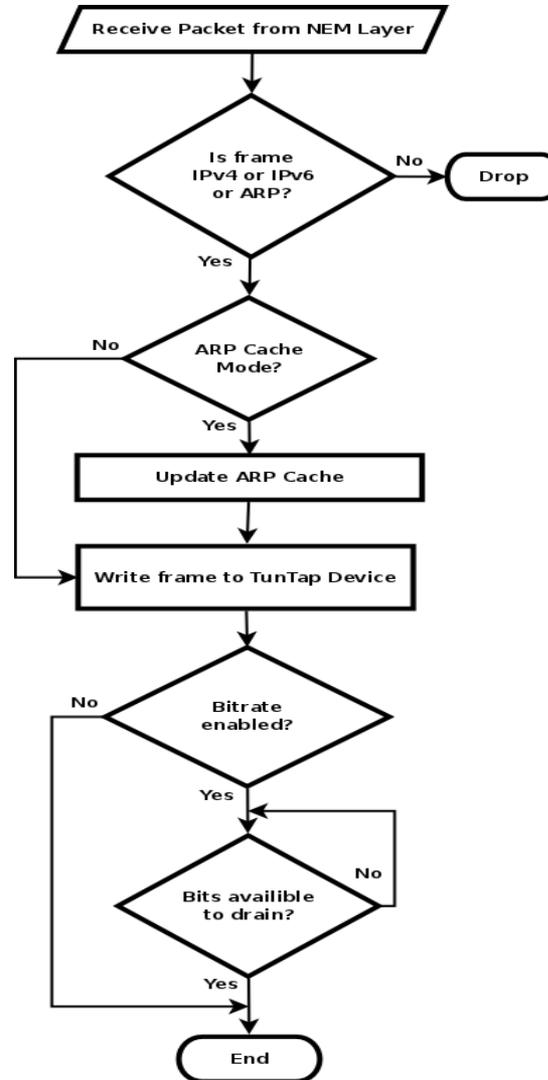
- When the Virtual Transport starts, it sends a control message to the flow control enabled layer requesting the current token count.
 - No downstream packets are transmitted to the flow control enabled layer until the flow control token count control message is received.
- Once received, the transport will send an acknowledgment control message.
 - This acknowledgment will satisfy the flow control enabled component in the situation where it was started prior to the transport and was blocked waiting for a previous acknowledgment.
- The Virtual Transport decrements its token count each time it sends a downstream packet.
 - When the token count reaches zero no further packets are transmitted causing application socket queues to backup.
 - The flow control enabled layer shadows the token count of the transport in order to detect when the transport has run out of tokens.



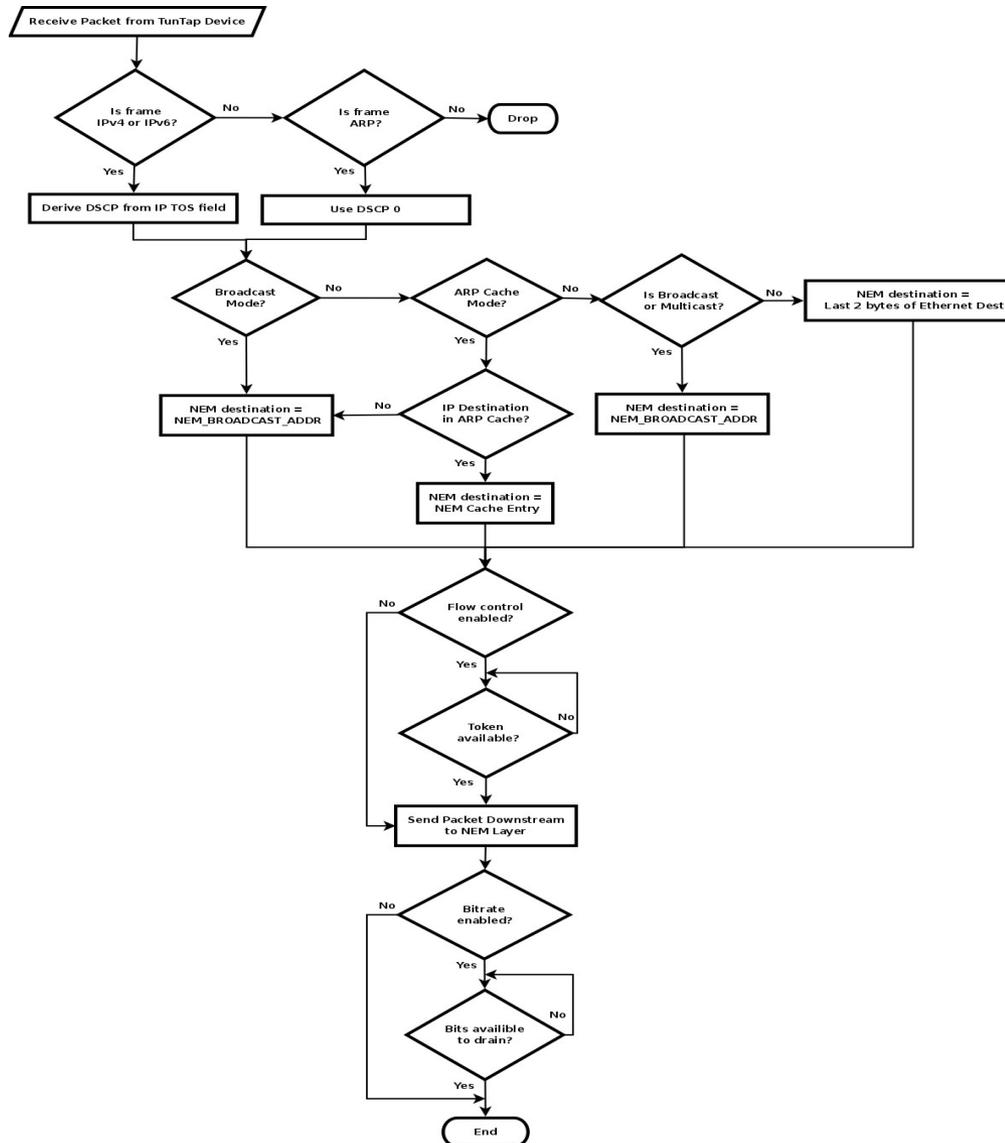
Flow Control

- Once available, the flow control enabled layer will send a flow control token count message restarting the process.
- If either flow control component, the Virtual Transport or the coordinating layer restarts, the token count will resync automatically.

Virtual Transport Upstream Packet Flow

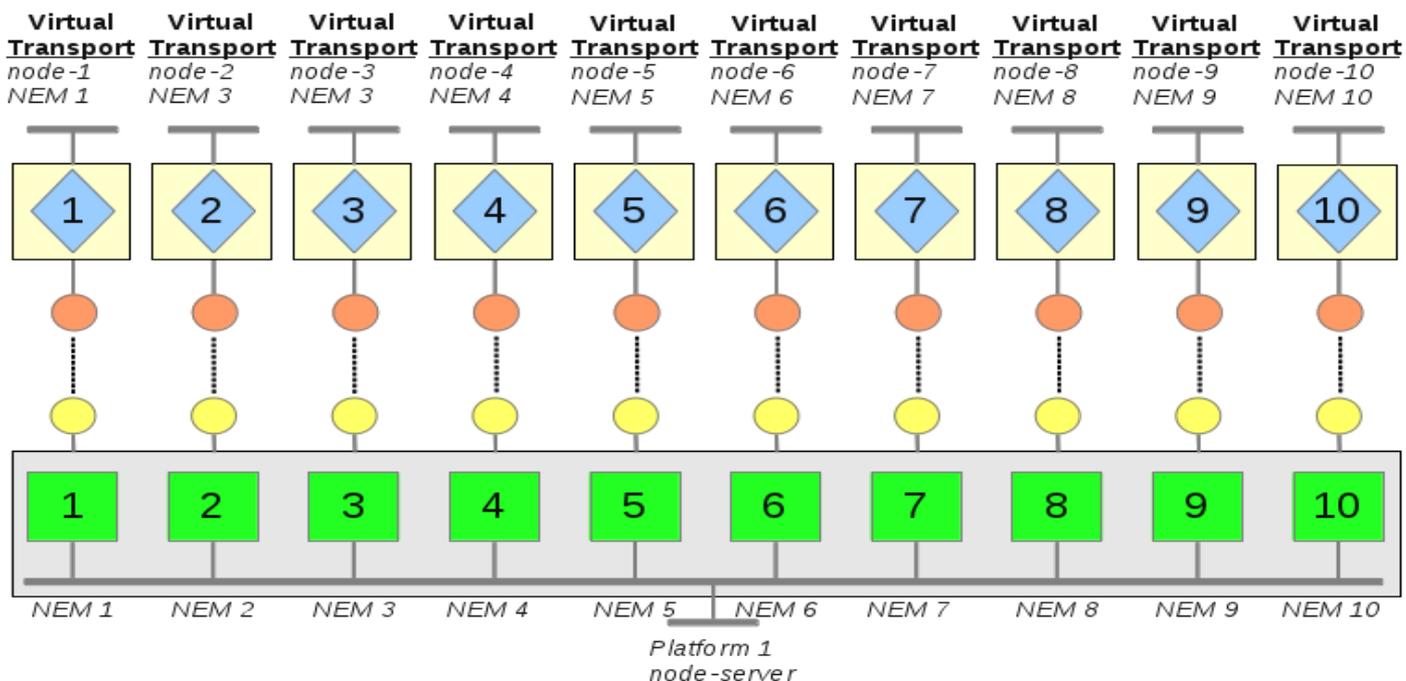


Virtual Transport Downstream Packet Flow



Demonstration 12

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





Raw Transport

Raw Transport

- The Raw Transport uses a specific network interface as the application/emulation boundary.
- IP packets read from the interface are encapsulated and transmitted to the Transport's respective NEM for downstream processing.
- Packets received upstream from the Transport's associated NEM are processed and transmitted out the specified network interface.

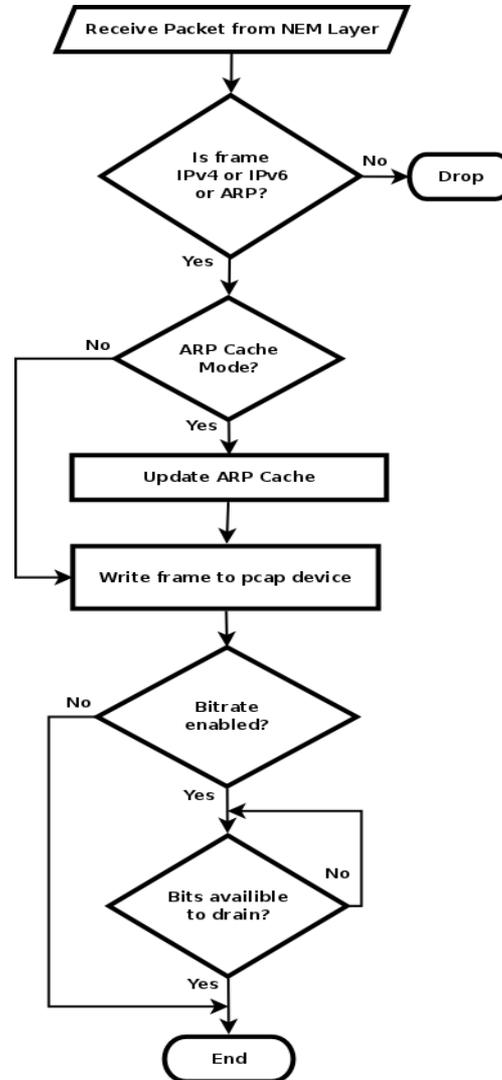
Raw Transport Configuration

- ***bitrate*** - Transport bitrate in Kbps. This is the total allowable throughput for the transport combined in both directions (upstream and downstream). A value of **0** disables the bitrate feature.
- ***broadcastmode*** - Broadcast all packets to all NEMs
- ***arpcheenable*** - Enable ARP request/reply monitoring to map ethernet address to NEM.
- ***device*** - Device to use as the raw packet entry point. Once a network device has been dedicated to a Raw Transport it should not be used for any other communication other than what should be routed into the emulation domain.

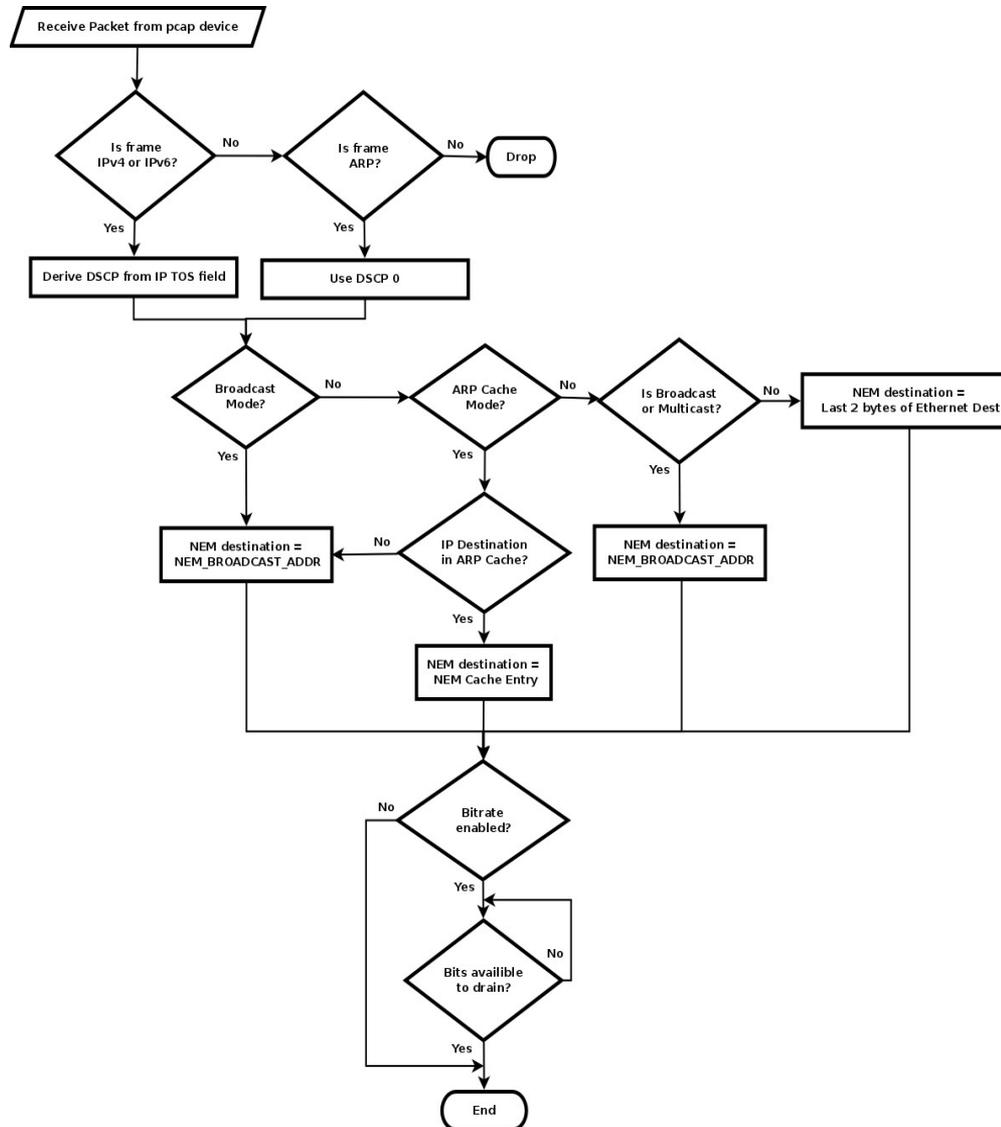
Transport Interoperability

- There is no guarantee that heterogeneous transports can be used in a given deployment.
 - The Transport API is designed to allow transports to transmit opaque data to and from their respective NEM stacks. The format of the data is implementation dependent.
- Both the Virtual Transport and Raw Transport route Ethernet frames. There are two configuration options that will allow both transports to communicate with each other: ARP Cache and Broadcast mode.
- When ***arpcacheenable*** is enabled, each transport will peek at ARP response packets that are sent upstream from their respective NEM stacks in order to build a table of destination addresses and associated NEM Ids.
- When ***broadcastmode*** is enabled, all packets are delivered using the NEM broadcast address. Each inbound transport will attempt to deliver the unicast Ethernet frames and rely on the kernel to drop all packets that do not match the host.

Raw Transport Upstream Packet Flow

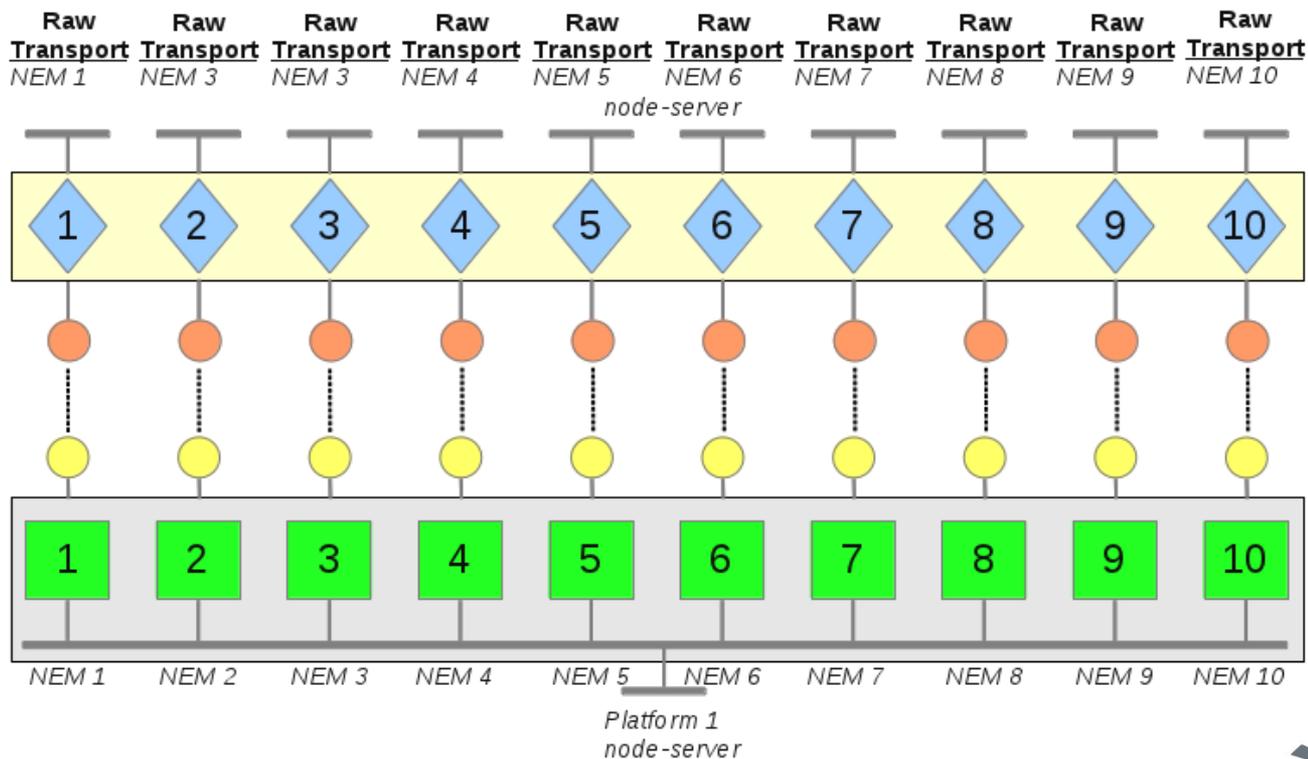


Raw Transport Downstream Packet Flow



Demonstration 13

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



Mitre Mobility Model Event Generator



Mitre Mobility Model Event Generator

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

Mitre Mobility Model Event Generator Configuration



- ***inputfileformat*** - Absolute file name of mobility file. One or more files may be specified by using a ***printf()*** style convention using the ***inputfilecount*** configuration item as an index.
- ***inputfilecount*** - Total number of input files. If the ***inputfilename*** contains a ***printf()*** style expression the count will be used as an index when creating the file names.
- ***totalnodes*** - Total number of nodes whose data is contained in the mobility files.
- ***maxnemidpresent*** - Maximum NEM Id present in the emulation experiment. This value is used to reduce the number of events generated when a subset of nodes from the larger mobility data are used.

Mitre Mobility Model Event Generator Configuration



- **repeatcount** - The number of times the mobility data should be parsed and events generated. A **repeatcount** of **1** simply means to process the file once, generating events, and stop when the file is complete. A value greater than **1** allows you to process the data **repeatcount** times. A value of **0** will process the data repeatedly, restarting indefinitely.
- **utmzone** - The UTM zone that corresponds to the UTM position information contained in the mobility data. This is required to convert the data when generating location events. A limitation of the Mitre Mobility Model format is that position data cannot cross UTM zones.
- **entryreplay** - Specify one or more space separated *time:count* pairs. Effectively allows you to hold at certain mobility entries for a specified amount of time.
 - A value of "0:120 3600:1800" would use the data at mobility entry T_0 for 2 minutes, sending out the T_0 events 120 times and use the data at T_{3600} for 30 minutes.



Mitre Mobility Model Event Generator Configuration



- ***publishpathlossevents*** - Create/Publish pathloss events from mobility model input files.
- ***publishlocationevents*** - Create/Publish location events from mobility model input files

Mitre Mobility Model Format

- Mitre Mobility Model Format is an ASCII text file containing a single entry for every pair of nodes in the mobility scenario on one second boundaries.
- For each second, the number of entries required to completely describe the connectivity for N nodes can be represented by the following equation: $\sum_{i=1}^{N-1} N-i$
- The Mitre Mobility Model text file contains eleven columns as defined below:
 1. Time in Seconds
 2. Node Target A
 3. Node Target B
 4. Pathloss between nodes. A single value denotes symmetric pathloss. Two values separated by a '/' denotes asymmetric pathloss. Where the first value is the pathloss from Node A to Node B and the second value is the pathloss from Node B to Node A.

Mitre Mobility Model Format



5. Distance between nodes in meters
6. Node A UTM X position
7. Node A UTM Y position
8. Node A Antenna Height (altitude) in meters
9. Node B UTM X position
10. Node B UTM Y position
11. Node B Antenna Height (altitude) in meters

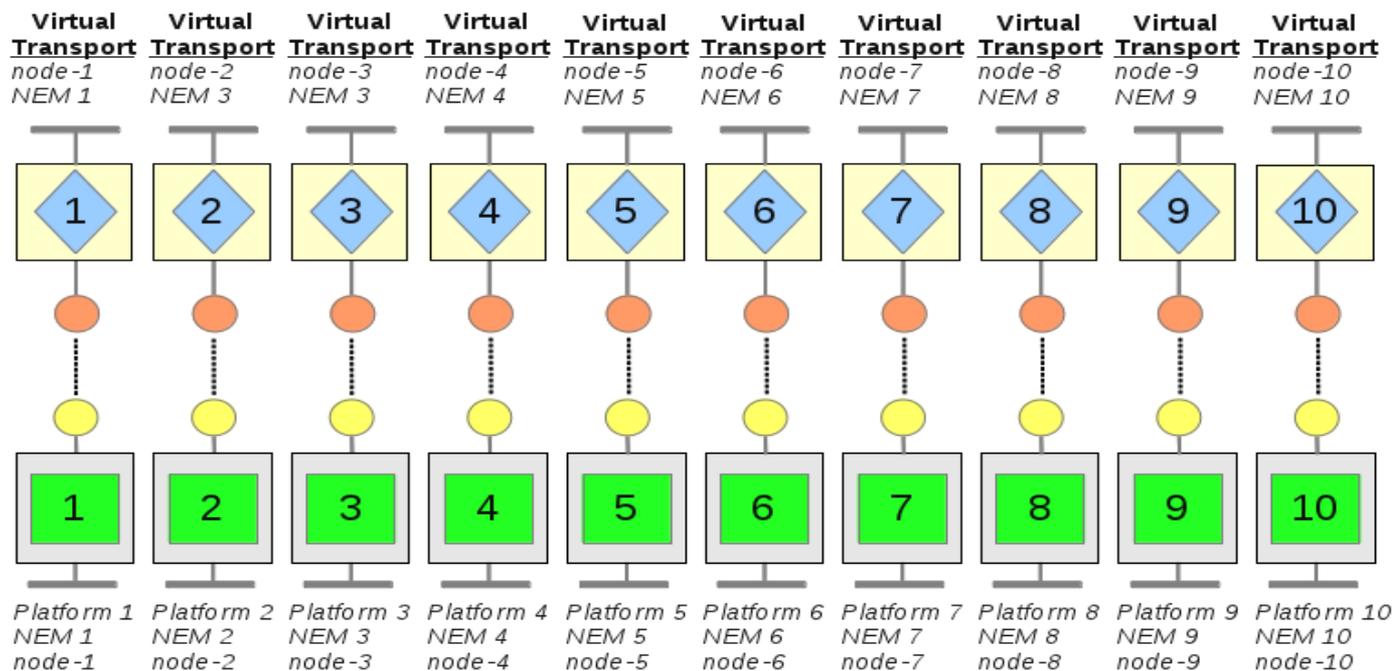
0	1	2	102	13	540654	4431315	3	540663	4431325	3
0	1	3	103/301	13	540654	4431315	3	540663	4431325	3
0	1	4	104/401	13	540654	4431315	3	540663	4431325	3
0	1	5	105/501	13	540654	4431315	3	540663	4431325	3
0	2	3	203/302	13	540654	4431315	3	540663	4431325	3
0	2	4	204/402	13	540654	4431315	3	540663	4431325	3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

Listing 13.1: Mitre Mobility file sample.



Demonstration 14

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



Emulation Script Event Generator

Emulation Script Event Generator

- The Emulation Script Event Generator creates location events from input files in the Emulation Script Format. Emulation Script Format was developed by the Protean Research Group at Naval Research Laboratory [Protean Research Group, 2010]. The file contains location information between nodes on specific time boundaries.

Emulation Script Event Generator Configuration



- ***inputfile*** - Absolute file name of the emulation script mobility input file.
 - One or more files may be specified by using this parameter multiple times and modifying the value attribute accordingly.
 - Files are processed in the order they appear in the XML.
- ***totalnodes*** - Total number of nodes whose data is contained in the files.
- ***repeatcount*** - The number of times the data should be parsed and events generated. A ***repeatcount*** of **1** simply means to process the file once, generating events, and stop when the file is complete. A value greater than **1** allows you to process the data ***repeatcount*** times. A value of **0** will process the data repeatedly, restarting indefinitely.
- ***schemalocation*** - Specifies the location of the schema file used to validate files specified via ***inputfile*** parameters.

Emulation Script Data Format

- Emulation Script Format is an XML file containing **Event** elements. Each element contains two child elements:
 - **time** - Specifying the amount of time in seconds that has elapsed since the start (initial event).
 - **Node** - Specifying the Id (using an attribute) and the location (using a child element) of a given node in the network.
- See [Protean Research Group, 2010] for a more detailed description of Emulation Script Format.

Emulation Script Data Format

- Emulation Script Format is an XML file containing **Event** elements. Each element contains two child elements:
 - **time** - Specifying the amount of time in seconds that has elapsed since the start (initial event).
 - **Node** - Specifying the Id (using an attribute) and the location (using a child element) of a given node in the network.
- See [Protean Research Group, 2010] for a more detailed description of Emulation Script Format.

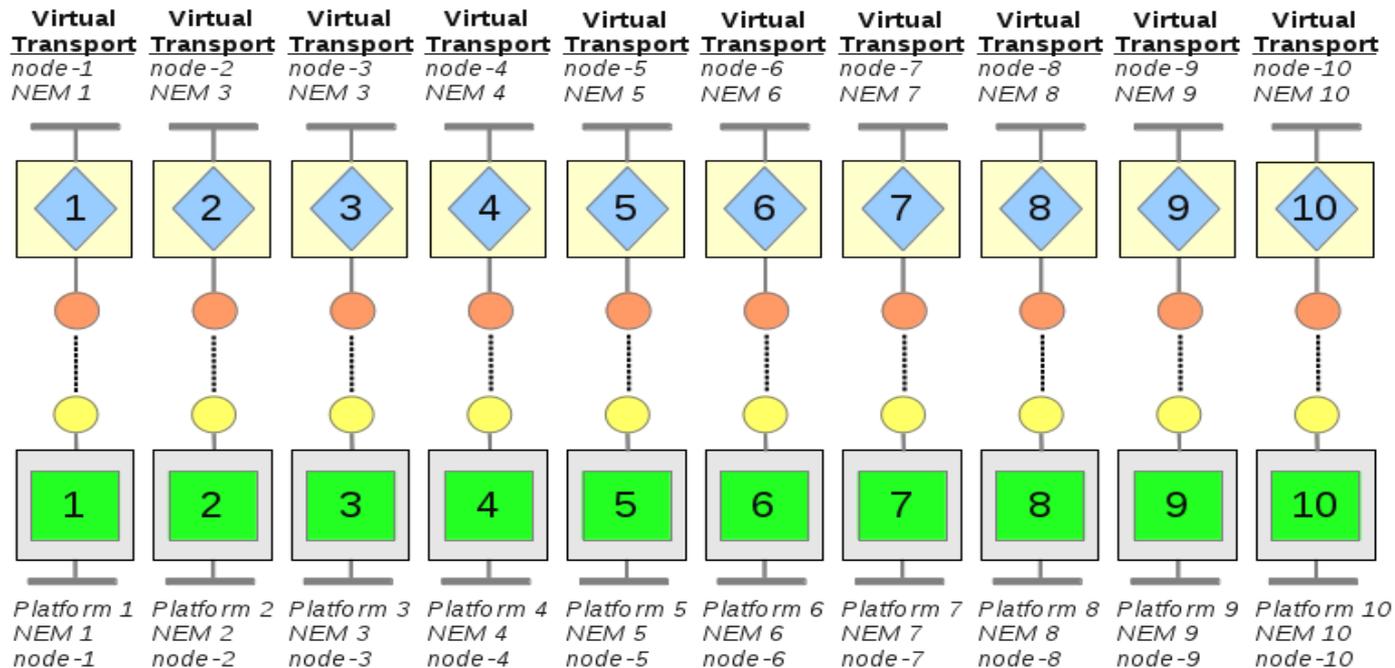
Emulation Script Data Format

```
<?xml version="1.0" encoding="UTF-8"?>
<EmulationScript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="EmulationScriptSchema.xsd">
  <Event>
    <time>0</time>
    <Node id="1">
      <location>40.0310751857906,-74.5235179912516,3</location>
    </Node>
    <Node id="2">
      <location>40.0311648464297,-74.5234118838455,3</location>
    </Node>
  </Event>
  <Event>
    <time>1</time>
    <Node id="1">
      <location>40.0311648464297,-74.5234118838455,3</location>
    </Node>
    <Node id="2">
      <location>40.031227237558,-74.5232473639998,3</location>
    </Node>
  </Event>
</EmulationScript>
```

Listing 14.1: Emulation Script Data file sample.

Demonstration 15

This demonstration deploys a ten node distributed RP Pipe NEM emulation experiment. The goal of this demonstration is to become familiar with the Emulation Script Generator.





Emulation Event Log Generator



Emulation Event Log Generator

- The Emulation Event Log (EEL) Generator creates EMANE events from input files in EEL Format. EEL format was developed by the Protean Research Group at Naval Research Laboratory [Protean Research Group, 2010].
- The EEL Event Generator loads EEL sentence parsing plugins to parse and build EMANE events.
- Plugins are associated with event type keywords and are capable of producing either full or delta event updates.
 - A delta event update contains 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 information loaded by the respective plugin.
- Any EEL entries encountered that are not handled by a loaded parser are ignored.

Sentence Parsing Plugins

- Pathloss Parser - Parses pathloss sentences and builds the resulting event.

***<time> nem:<Id> pathloss nem:<Id>,<pathloss>[,<reversePathloss>]
[nem:<Id>,<pathloss>[,<reversePathloss>]]...***

pathloss - Pathloss in dB.

reversePathloss - Reverse Pathloss in dB

- Location Parser - Parses location sentences and builds the resulting event.

<time> nem:<Id> location <latitude>,<longitude>,<altitude>[,<msl|agl>]

latitude - Latitude in degrees.

longitude - Longitude in degrees.

altitude - Altitude in meters.

Sentence Parsing Plugins

- Antenna Direction Parser - Parses antenna direction sentences and builds the resulting event.

***<time> nem:<Id> antennadirection
<elevation>,<azimuth>,<elevationBeamWidth>,<azimuthBeamWidth>***

- *elevation* - Antenna elevation in degrees.
- *azimuth* - Antenna azimuth in degrees.
- *elevationBeamWidth* - Antenna elevation beam width in degrees.
- *azimuthBeamWidth* - Antenna azimuth beam width in degrees.

Emulation Event Log Generator Configuration



- ***inputfile*** - Absolute file name of the EEL input file.
 - Additional EEL files may be specified using multiple ***inputfile*** parameters.
 - Files are processed in the order they appear in the XML.
- ***loader*** - Map EEL event type keywords to EEL loader plugins.

<eventType>:<Plugin Name>:[full|delta]

- The optional ***full*** or ***delta*** determines whether events produced from the plugins represent only the new EEL entries processed since the last request for events or the complete current cached state.
- The default specification is ***delta***.

Emulation Event Log Format

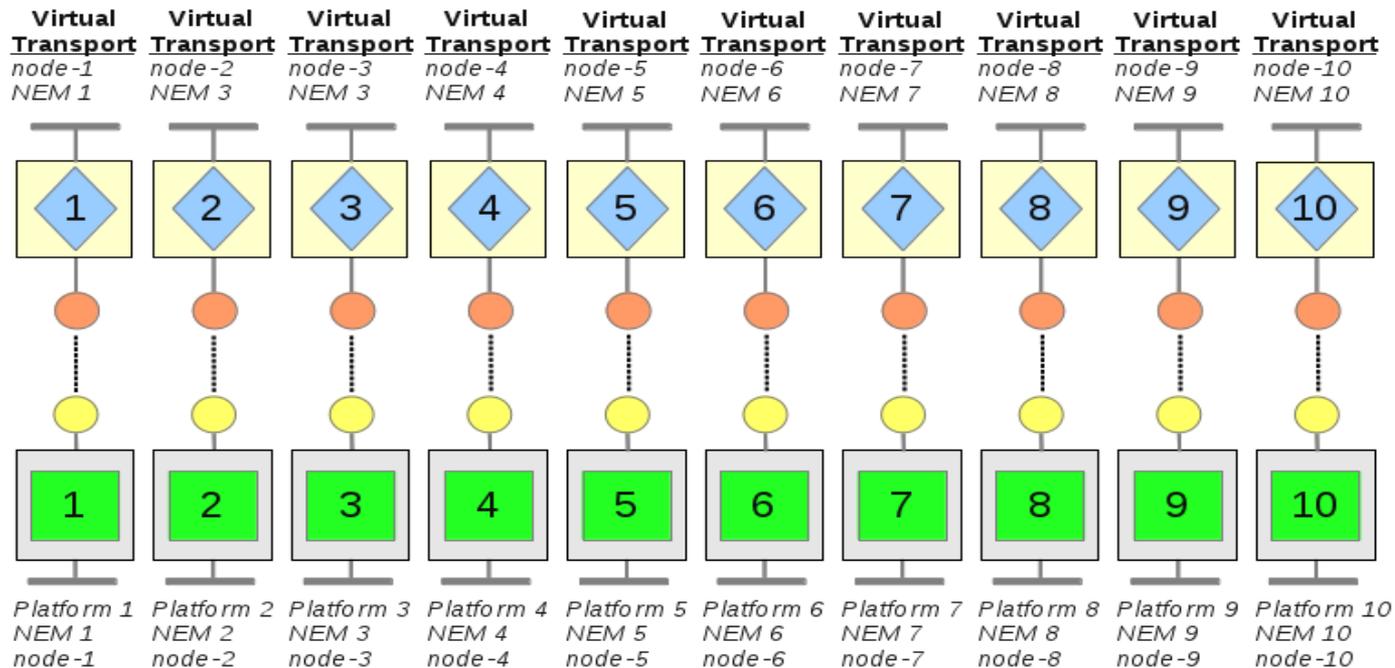
- For more information on Emulation Event Log Format see [Protean Research Group, 2010].

```
0.0 nem:70 pathLoss nem:22,96.3 nem:23,95.0 nem:24,95.1 nem:25,95.2 nem:26,95.3 nem:27,95.4 nem:28,95.5
nem:29,95.0 nem:30,95.1 nem:31,95.2 nem:32,95
0.0 nem:70 pathLoss nem:42,95.3 nem:43,95.4 nem:44,95.5 nem:45,95.0 nem:46,95.1 nem:47,95.2 nem:48,95.3
nem:49,95.4 nem:50,95.5 nem:51,95.5 nem:52,95.6
0.0 nem:70 pathLoss nem:62,95.2 nem:63,95.3 nem:64,95.4 nem:65,94.2 nem:66,94.2 nem:67,96.3 nem:68,96.3
nem:69,123.3
0.0 nem:1 location gps 40.031075,-74.523518,3.000000
0.0 nem:2 location gps 40.031165,-74.523412,3.000000
0.0 nem:3 location gps 40.031227,-74.523247,3.000000
0.0 nem:4 location gps 40.031290,-74.523095,3.000000
0.0 nem:1 antennadirection 0,90,180,10
0.0 nem:2 antennadirection 0,270,180,10
0.0 nem:3 antennadirection 0,90,180,10
0.0 nem:4 antennadirection 0,270,180,10
```

Listing 15.1: Emulation Event Log sample.

Demonstration 16

This demonstration deploys a ten node distributed RP Pipe NEM emulation experiment. The goal of this demonstration is to become familiar with the Emulation Event Log Generator.



Comm Effect Event Generator



Comm Effect Event Generator

- The Comm Effect Event Generator creates Comm Effect events from input files in the Comm Effect Impairment Format. The file contains impairment information between nodes on one second boundaries.

Comm Effect Event Generator Configuration



- ***inputfile*** - Absolute file name of the impairment file.
 - Additional impairment files may be specified using multiple ***inputfile*** parameters.
 - Files are processed in the order they appear in the XML.
- ***totalnodes*** - Total number of nodes whose data is contained in the impairment file(s).
- ***maxnemidpresent*** - Maximum NEM Id present in the emulation experiment. This value is used to reduce the number of events generated when a subset of nodes from the larger Comm Effect data are used.
- ***repeatcount*** - The number of times the impairment data should be parsed and events generated. A ***repeatcount*** of **1** simply means to process the file once, generating events, and stop when the file is complete. A value greater than **1** allows you to process the data ***repeatcount*** times. A value of **0** will process the data repeatedly, restarting indefinitely.

Comm Effect Event Generator Configuration



- ***entryreplay*** - Specify one or more space separated *time:count* pairs.
 - Effectively allows you to hold at certain impairment entries for a specified amount of time.
 - A value of "0:120 3600:1800" would use the data at impairment entry T_0 for 2 minutes, sending out the T_0 events 120 times and use the data at T_{3600} for 30 minutes.

Comm Effect Impairment Format

- Comm Effect Impairment Format is an ASCII text file containing a single entry for every pair of nodes in the scenario on one second boundaries. For each second, the number of entries required to completely describe the impairments for N nodes can be represented by the following equation: $\sum_{i=1}^{N-1} N-i$
- The Comm Effects Impairment text file contains eleven columns as defined below:
 1. Time in seconds
 2. NEM A Target
 3. NEM B Target
 4. Latency (seconds) - Second component of the average delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric latency between the NEM pairs.
 5. Latency (microseconds) - Microsecond component of the average delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric latency between the NEM pairs.



Comm Effect Impairment Format

6. Jitter (seconds) - Second component of the jitter on the delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric jitter between the NEM pairs.
7. Jitter (microseconds) - Microsecond component of the jitter on the delay to be introduced for packets between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric jitter between the NEM pairs.
8. Loss (percentage) - Loss percentage to be introduced between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric loss between the NEM pairs.
9. Duplicates (percentage) - The duplicate percentage to be introduced between NEM A and NEM B. A single value denotes symmetry and two values separated by a '/' denotes asymmetric duplication between the NEM pairs.
10. Unicastbitrate (bps) - The bitrate to be introduced between NEM A and NEM B for packets addressed to the NEM or handled in promiscuous mode. A single value denotes symmetry and two values separated by a '/' denotes asymmetric bitrate between the NEM pairs.
11. Broadcastbitrate (bps) - The bitrate to be introduced between NEM A and NEM B for packets sent the NEM Broadcast Address. A single value denotes symmetry and two values separated by a '/' denotes asymmetric bitrate between the NEM pairs.



Comm Effect Impairment Format



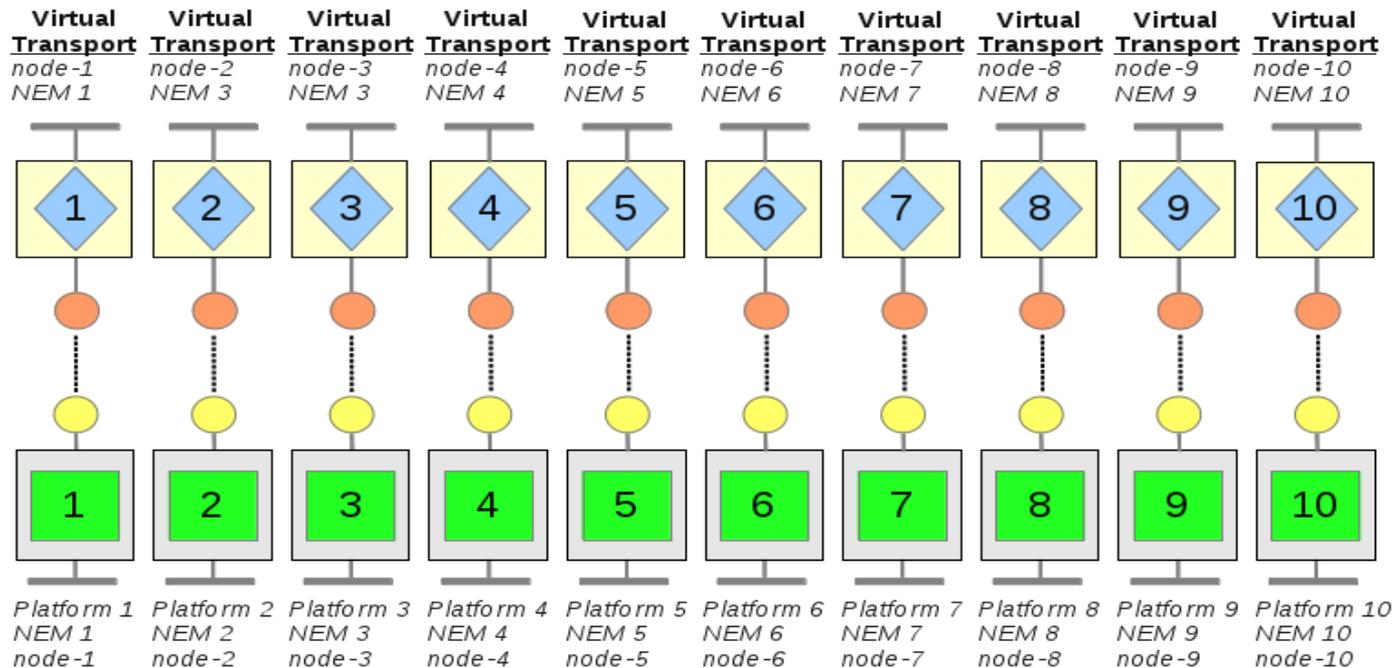
0	1	2	0	0	0	0	0	0	0	10000	1000
0	1	3	0	0	0	0	0	0	0	10000	1000
0	1	4	0	0	0	0	0	0	0	10000	1000
0	2	3	0	0	0	0	0	0	0	10000	1000
0	2	4	0	0	0	0	0	0	0	10000	1000
0	3	4	0	0	0	0	0	0	0	10000	1000
1	1	2	0	500000	0	100000	50	0	0	10000	1000
1	1	3	0	500000	0	100000	50	0	0	10000	1000
1	1	4	0	500000	0	100000	50	0	0	10000	1000
1	2	3	0	500000	0	100000	50	0	0	10000	1000
1	2	4	0	500000	0	100000	50	0	0	10000	1000
1	3	4	0	500000	0	100000	50	0	0	10000	1000
2	1	2	0	0	0	0	100	0	0	10000	1000
2	1	3	0	0	0	0	100	0	0	10000	1000
2	1	4	0	0	0	0	100	0	0	10000	1000
2	2	3	0	0	0	0	100	0	0	10000	1000
2	2	4	0	0	0	0	100	0	0	10000	1000
2	3	4	0	0	0	0	100	0	0	10000	1000
3	1	2	0	300000/500000	0	100000/200000	25/40	0/30	0	10000	1000
3	1	3	0	300000/500000	0	100000/200000	25/35	0/100	0	10000	1000
3	1	4	0	300000/500000	0	100000/200000	25/50	0/50	0	10000	1000
3	2	3	0	500000	0	100000	25	0	0	10000	1000
3	2	4	0	500000	0	100000	25	0	0	10000	1000
3	3	4	0	500000	0	100000	25	0	0	10000	1000

Listing 16.1: Comm Effect Effect file sample.



Demonstration 17

This demonstration deploys a ten node distributed Comm Effect NEM emulation experiment. The goal of this demonstration is to become familiar with the Comm Effect Event Generator.



Antenna Direction Event Generator



Antenna Direction Event Generator

- The Antenna Direction Event Generator creates antenna direction events from input files in the Antenna Direction format. The file contains node based antenna profile and pointing information on one second boundaries.

Antenna Direction Event Generator Configuration



- ***inputfileformat*** - Absolute file name of the antenna direction file. One or more files may be specified by using a ***printf()*** style convention using the ***inputfilecount*** configuration item as an index.
- ***inputfilecount*** - Total number of input files. If the ***inputfilename*** contains a ***printf()*** style expression the count will be used as an index when creating the file names.
- ***totalnodes*** - Total number of nodes whose data is contained in the antenna direction file.
- ***repeatcount*** - The number of times the antenna direction data should be parsed and events generated. A ***repeatcount*** of **1** simply means to process the file once, generating events, and stop when the file is complete. A value greater than **1** allows you to process the data ***repeatcount*** times. A value of **0** will process the data repeatedly, restarting indefinitely.



Antenna Direction Format

- Antenna Direction Format is an ASCII text file containing a single entry for every node using directional antenna on one second boundaries.
- The Antenna Direction text file contains eleven columns as defined below:
 1. Time in Seconds
 2. Node Target
 3. Antenna Elevation in degrees
 4. Antenna Azimuth in degrees
 5. Antenna Elevation Beam Width in degrees
 6. Antenna Azimuth Beam Width in degrees

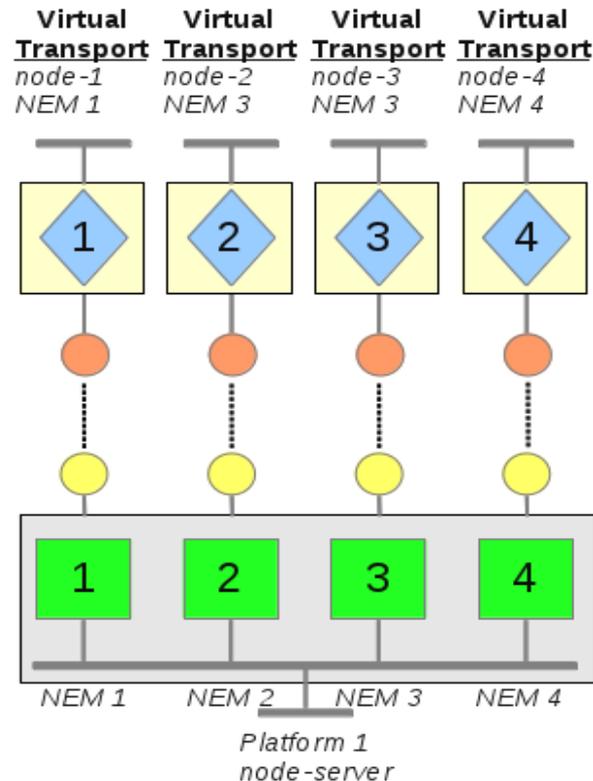
Antenna Direction Format

```
0 1 0 90 180 10
0 2 0 270 180 10
0 3 0 90 180 10
0 4 0 270 180 10
1 1 0 180 180 10
1 2 0 180 180 10
1 3 0 0 180 10
1 4 0 0 180 10
2 1 0 135 180 10
2 2 0 225 180 10
2 3 0 45 180 10
2 4 0 315 180 10
```

Listing 17.1: Antenna Direction file sample.

Demonstration 18

This demonstration deploys a four node centralized IEEE 802.11abg NEM emulation experiment. The goal of this demonstration is to become familiar with the Antenna Direction Event Generator.





GPSd Location Agent



GPSd Location Agent

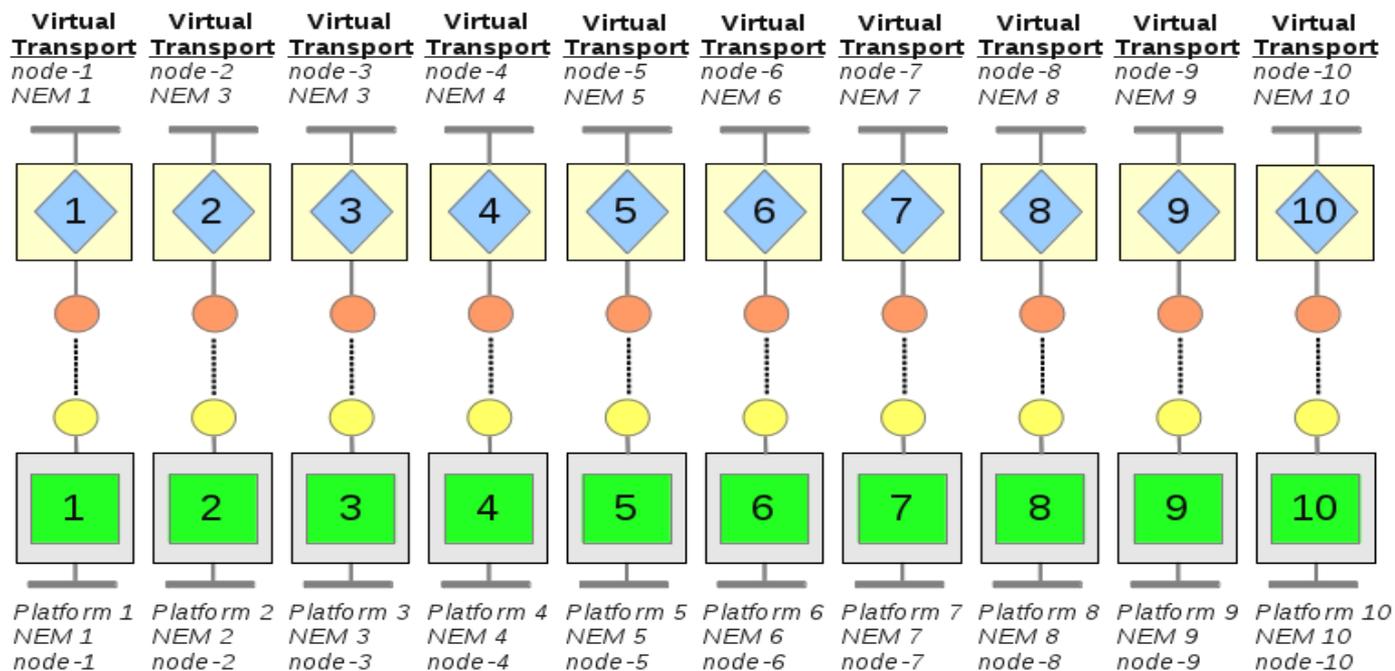
- The GPSd Location Agent uses a pseudo terminal to emulate a GPS Receiver.
- Received location events are used to generate NMEA strings which are written to a pseudo terminal in order to make position information available to any application capable of parsing NMEA strings.

GPSd Location Agent Configuration

- ***gpsdcontrolsocket*** - The name of the GPSd control socket for adding the pseudo terminal to the device list. The control socket is used when the GPSd Location Agent instance should attempt to connect to GPSd. Only used when ***gpsdconnectionenabled*** is set to ***on***.
- ***pseudoterminalfile*** - The name of the file to create containing the name of the pseudo terminal in use by the GPSd Location Agent. Only created when ***gpsdconnectionenabled*** set to ***off***.
- ***gpsdconnectionenabled*** - Switch to set GPSd Location Agent to either actively connect to GPSd (***on***) or instead create a file containing the name of the pseudo terminal currently in use (***off***).

Demonstration 19

This demonstration deploys a ten node distributed IEEE 802.11abg NEM emulation experiment. The goal of this demonstration is to become familiar with the GPSd Location Agent.



Python Event Service Bindings



Python Event Service Bindings

- The Python Event Service bindings allow for the creation of custom Python scripts which can interact with the EMANE Event Service using libemaneventservice, a C language library for developing embedded event processing applications.
- The Event Service Python bindings are comprised of bindings for the Event Service and the four standard EMANE events: Location, Pathloss, Comm Effect and Antenna Direction.

libmaneeventservice Configuration

libmaneeventservice will search for its configuration in three locations before falling back and using application defaults. The search order is as follows:

1. If the environment variable **LIBEMANEEVENTSERVICECONFIG** exists, it will be used. This variable must be set to a configuration file name.
2. If **\$HOME/libmaneeventservice.xml** exists, it will be used.
3. If **/etc/libmaneeventservice.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.

```
1 <?xml version='1.0' standalone='yes'?'>
2 <emaneeventmsgsvc>
3   <group>224.1.2.8</group>
4   <port>45703</port>
5   <device>lo</device>
6   <mcloop>1</mcloop>
7   <ttl>32</ttl>
8 </emaneeventmsgsvc>
```

Listing 19.1: libmaneeventservice configuration file.

Demonstration 20

This demonstration uses two sample Python scripts to illustrate how to use the Python Event Service bindings.

Python EMANE Bindings

Python EMANE Bindings

- The EMANE Library Python bindings are contained in the Python ***emane*** module.
- By importing the ***emane*** module, a Python script can configure and run the equivalent of any of the EMANE applications: **emane**, **emanetransportd**, **emaneeventservice** and **emaneeventd**.
- Configuration is performed via Python tuples and does not require XML configuration.
- Access to the EMANE logger allows log level and log destination configuration, similar to the application logger command line arguments.

Demonstration 21

This demonstration deploys a ten node distributed RF Pipe NEM emulation experiment. Each node executes a Python script similar to the one developed above.

