Requirements for Wireless GENI Experiment Control and Management

GDD-07-43

GENI: Global Environment for Network Innovations

February 20, 2007

Status: Draft Version 1.2

This document was prepared by the Wireless Working Group.

Editors:

Sanjoy Paul, *WINLAB, Rutgers University*

Sachin Ganu, *WINLAB, Rutgers University*

Pandurang Kamat, *WINLAB, Rutgers University*

Elizabeth Belding Royer, *University of California, Santa Barbara*

Contributing workgroup members:

Chip Elliot, *BBN*

Dipankar Raychaudhuri, *WINLAB, Rutgers University*

Gary Minden, *University of Kansas*

Mario Gerla, *University of California, Los Angeles*

Larry Peterson, *Princeton University*

We also acknowledge the very helpful comments and suggestions from Ivan Seskar and Kishore Ramachandran, WINLAB, Rutgers University.

The work is supported in part by NSF grants CNS-0540815 and CNS-0631422.

Table of Contents

List of figures

1 Introduction

This document describes the requirements for defining, running and controlling an experiment in the wireless networks. In addition, the requirements of a management system for wireless GENI are also described. The goal of this document is to define a control system that can be used for a variety of environments, such as, outdoor "field" deployments, indoor RF-controlled facilities, and even simulation in computing clusters, such that a GENI user can freely mix and match experiments across a number of embodiments, e.g., first perform experiments entirely in simulators, then in a mix of simulation and fielded systems.

However, it should be noted that as one moves from simulation to indoor RF-controlled environments to outdoor "field" deployment, the "control" of the experimenter on the environment becomes weaker and weaker. As a result, a slice that an experimenter may acquire for her experiments in simulation or in an indoor RF-controlled environment may not be available in the outdoor "field" deployment because of a variety of reasons outside the experimenter's control. This is not a limitation of GENI experiment control rather a constraint imposed by the environment. Throughout the description of various components, we will use a simple example to provide a clearer understanding of the system components and give the experimenters a feel for the kind of interface to expect

2 GENI Management and Control (GMC)

Figure-1 captures the architecture of GENI Management and Control (GMC). GMC is organized as a hierarchical architecture where each component is managed by a Component Manager (CM), a group of components is managed by an Aggregate Manager (AM), and entire GENI is managed by a root aggregate which is an “Aggregate of Aggregates”. It should be noted that the Component Manager (CM) need not be resident on the component itself, and similarly, the Aggregate Manager (AM) need not be resident on the programmable router in the Edge cluster as shown in Figure-1. A *Coordination Aggregate* provides a common interface for a related set of components, treating the set as a unit for the purpose of allocating resources, instantiating slices, and starting and stopping slices. We sometimes say that a coordination aggregate implements a *slice control* function, because it is responsible for a coordinated allocation of resources for slices in a subnet. There is also a concept of *Researcher Portal*. A *researcher portal* is an interface—graphical, programmatic, or both—that defines an “entry point” through which users access GENI components. A researcher portal for a wireless researcher might leverage a coordination aggregate for a wireless subnet for *resource discovery and resource allocation* across a wireless subnet. In the rest of the document it is assumed that a user who wants to run an experiment in a wireless subnet will contact the researcher portal for wireless subnet which in turn will use the services of a coordination aggregate for experiment configuration and management, as well as for other user-level services that contribute to the overall researcher experience. The process of creating a slice, configuring the slice for running an experiment, uploading the code into the configured slice (if necessary), executing the experiment, and collecting measurements is collectively referred to as “experiment” control. Experiment control is discussed in this document.

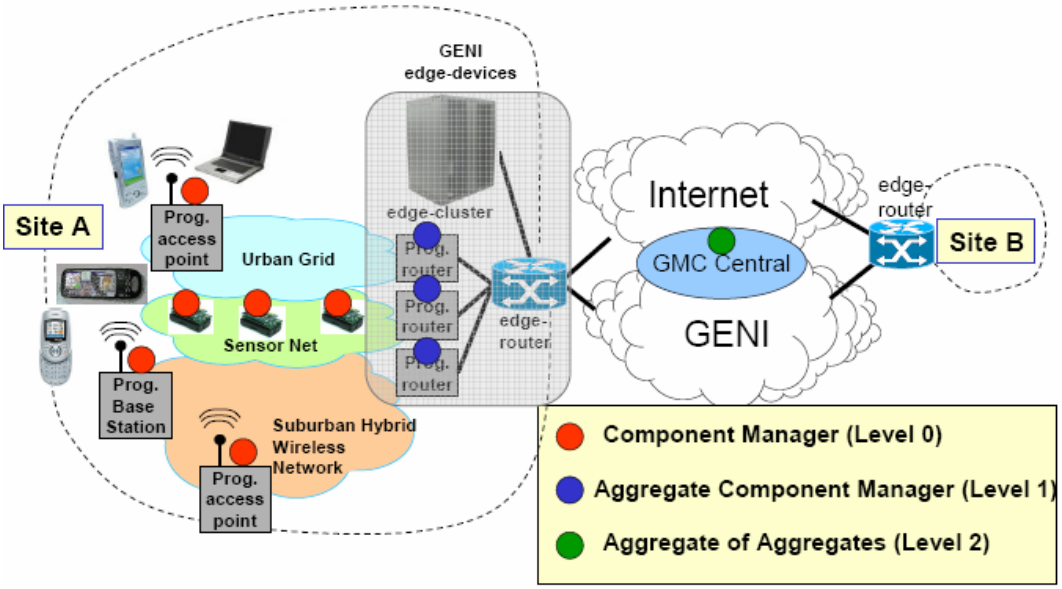

Figure 1 GENI Management and Control (GMC) Architecture

3 Experiment Control for Wireless GENI

This section defines the details of experiment control, especially how a GENI user who wants to run experiments on GENI would interface with GENI. The goal is to provide a uniform technology-agnostic API to the experimenter to create a slice, configure it, upload experiment code in the slice, run experiments and collect measurements to do an analysis.

The mechanism/messaging described in this document are expected to be very generic. That is, these techniques should work not only for “first class” GENI nodes (such as, Programmable Wireless Nodes or PWNs), but they should also work for "non-GENI nodes" (e.g. motes or personal cellphones). It would be the responsibility of the Aggregate Manager to map between the GENI Experimental Control and the existing control system used by non-GENI system for controlling the disadvantaged nodes, such as sensor nodes, cellphones, etc. In other words, the Aggregate Manager would act as the experiment control proxy for these devices, and communicate the needful to the non-GENI nodes. For example, at the request of an experimenter, the Aggregate Manager will select a set of non-GENI sensor nodes arranged in a given topology, configure them, define their functional roles and even take the necessary steps to collect the desired measurements using a non-GENI proprietary interface.

For the remainder of this document, we will assume that the following experiment needs to be conducted on the GENI platform.

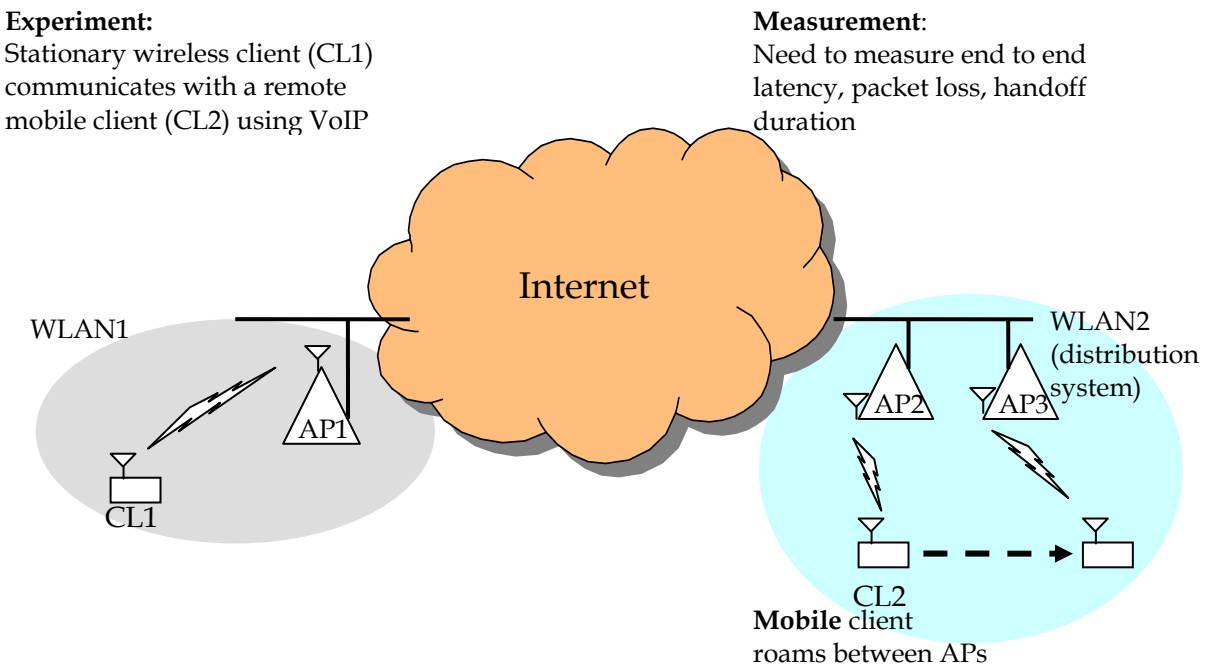

Figure 2 Sample experiment (VoIP session between wireless clients)

At a high level, the experiment involves a stationary WLAN client (CL1) engaged in a VoIP communication with a remote WLAN client (CL2) which is mobile. The desired outcome is a measurement of the end-to-end packet loss and latency as well as the handoff duration.

In order to conduct this experiment on the GENI infrastructure, one can envision the following steps

1. Creating new set of credentials for the experiment that are stored as a part of the slice.
2. Request for the nodes and desired topology by the experimenter to the GENI infrastructure **(TSpec)**
3. Definition of the functional roles of the various entities involved in the experiment (e.g VoIP sender, VoIP receiver) **(FSpec)**
4. Definition of the measurements to be made as a part of the experiment **(MSpec)**
5. Description of experiment dynamics
6. Resource allocation by the GENI infrastructure for this experiment in terms of nodes, links and experiment time
7. Actual experiment deployment (upload code, start services etc)

We cover each of these steps in detail from the experimenters perspective.

3.1 Slice Creation

Slice creation involves creation of a project related credentials and associating various existing users with them. A slice can be thought of as a long lasting set of credentials to which various resources such as nodes of interest, the desired resources (processing, memory, storage, etc.) and the desired connectivity (including bandwidth requirements) can be assigned on a per experiment basis.

3.2 Experiment Definition

3.2.1 Node and Topology specification (TSpec)

Expert users, instead of relying on GMC to provide a mapping of high-level specification to a set of nodes and the desired topology, may explicitly specify the nodes of choice and the topology using a low-level topology specification.

High-level topology specification

At the highest level of abstraction, topology may be specified using a GUI, where the users can drag, drop and connect various network components. This example shows a simple network consisting of three wireless Access points serving two wireless LANs and connected over a wide area wired network. This topology can be internally translated to an XML representation (Figure 3) that can be used by the Resource Manager for actual node allocations. This definition provides and efficient, flexible and unambiguous representation of such network topologies. Further, it facilitates the exchange of these topology representations between various programs such as co-ordination aggregates and resource managers

Detailed topology specification

Expert users may also be allowed to request specific resources (for example, nodes on the Chicago Urban wireless grid substrate). These detailed specifications may also consist of specific link level properties (example links with certain SNR, PER). Note that finding exact topologies may be more difficult than satisfying high level request.

```
<Topology>
 <Nodes>
  <Node id=AP1>
   <interface id=eth1
     type=wifi>
   </interface>
   <interface id=eth0
        type=ether>
     <properties>
       <property name=bandwidth>
        <value> 100M </value>
       </property>
     </properties>
   </interface>
  </Node>

  <Node id=CL1>
    <interface id=eth1
     type=wifi>
    </interface>
  </Node>

 <Node id=AP2>
  <interface id=eth1
    type=wifi>
  </interface>
  <interface id=eth0,
    type=ether>
     <properties>
       <property name=bandwidth>
        <value> 100M </value>
       </property>
     </properties>
  </interface>
 </Node>
..Other node definitions here
</Nodes>
```

```
 <Links>
   <Link id =link1 type=wifi-
    infrastructure>
     <interface>#AP1:eth1
     </interface>
     <interface>#CL1:eth1
     </interface>
     <properties>
       <property name=PER>
        <value>0.0001</value>
       </property>
     </properties>
    </Link>

   <Link id=link1 type=wired>
    <interface> #AP1:eth2
    </interface>
    <interface> #AP2:eth2
    </interface>
     <properties>
        <property name=PER>

          <value>0.000001
          </value>
        </property>
     </properties>
    </Link>
…other link definitions here…
 </Links>

</Topology>
```

Figure 3 High Level Topology Specification

```
<Topology>
 <Nodes>
  <Node id=Chicago.Outdoor.AP1>
   <interface id=eth1
     type=wifi>
```

Figure 4 Detailed Topology Specification

Note: In a simulation system or in an indoor RF-controlled emulation testbed, it would be easier to control link bandwidth, BER, SNR etc. parameters in a topology specification. However, for an outdoor "field" system, the experimenter may have to compromise on these requirements and accept what is available at the given instant of time in an operational network. Whenever the experimenter is willing to accept the ambient conditions with respect to some of the link attributes, the corresponding attributes (parameters) should be omitted from the tspec.

3.2.2 Configuring selected nodes/interfaces

Once a slice is created, the experimenter has to configure the slice for each experiment. The idea is to run multiple experiments on a given slice, and configure the slice separately for each experiment. Configuration of the set of selected nodes (slivers) can be done via the following function call:

Experiment_Id = CONFIGURE_NODES (List of (Node, Config_Parameters))

where each Node from the selected Node_Set participating in an experiment will be configured using Config_Parameters. Note that this allows for the case where all nodes in the selected Node_Set need not participate in an experiment. Config_Parameters is specified using (name, value) pairs of rspec. An experiment identified by the handle Experiment_id is instantiated at this point.

For example, in a selected sliver with an 802.11a interface and a CDMA 1x Ev-DO Rev.A interface, an experimenter may provide the following configuration using rspec. Note that the transmit power levels/carrier sense thresholds may only be selected from an allowable range.

- 802.11a interface:

  (SSID = HelloWorld), (rate = auto), (mode = ad-hoc), (channel_no = 36), (power = 10mw), (carrier_sense_threshold = -70dBm)

- CDMA Ev-DO Rev.A interface:

  (scheduling = prop_fair), (DL_datarate = 500 kbps), (UL_datarate= 100 kbps), (power = 50dBm)

3.2.3 Defining functional roles of selected nodes/interfaces

Once a slice is created and configured, functional role of each of the selected nodes may be defined. The idea is for GENI management system to provide a catalog of prototype functional roles that users can leverage for their experiments. Experimenters can either use these prototypes as is with the specified parameters or can derive specified roles from these base roles. Note that this step is optional because experimenter may upload the desired code at the desired slivers for her experiments and may not want to use the prototypes provided by GENI management system.

Functional Specification (FSpec)

Functional role can be specified using a list of tuples (Type, Node_Set) where Type is one of the functional roles specified above, and Node_Set indicates the specific nodes that would be assigned the corresponding functional roles. Note that the functional roles can be derived from some underlying applications. As an example, if the underlying application is a traffic generator (TrafGen), a VoIP sender would be a derived application (specific function) of the general application, TrafGen. Applications can be contributed by experimenters to the GENI Tool Manager (section 4.3.1). Details of the application specification are described in Appendix

Functional spec can be assigned to the slivers in the experiment (handle: Experiment_id) using the function:

ASSIGN_ROLES (Experiment_id, FSpec)

An example FSpec might be:

Fspec = {
(Type=Sender; Node_Set = (node1, node2); Sending Rate = 50 kbps; Traffic = CBR)
(Type=Receiver; Node_Set = (node3))
(Type=Forwarder; Node_Set = REST)
}

This specifies two traffic sources at node1 and node2, one traffic sink at node3 and forwarding nodes at the remaining nodes. REST is a short-cut for all nodes in the user-created slice other than node1, node2 and node3.

Note that there may not be specific roles for slivers in GENI, as GENI experiments may not be neatly decomposable into these roles. The objective is not to force every GENI experiment to fit into this model, however, experimenters would benefit from creating an Fspec for their custom derived applications that can be used by GMC to configure various settings on groups of nodes supporting the same application. Sample Fspec templates can be made available by the GENI tool kit for customization.

Another point to note here is that there is, in general, no need to specify functionality of a node because the functionality of a node will be determined by the uploaded code. However, a user can leverage some prototype code available in GENI Tool Manager (section 4.3.1) if she so desires instead of uploading her own code. FSpec for our sample experiment is as follows

Fspec = {

(Type=VoIPClient; Node_Set = (CL1), Sending Rate = 64 kbps, Packet size: 160 bytes;Traffic = CBR)

(Type=VoIPClient; Node_Set = CL2)

Figure 5 Sample FSpec for a VoIP client

3.2.4 Measurement Specification (MSpec)

Measurement specification (MSpec) is a way for an experimenter to specify what parameters she is interested in measuring. MSpec has two types of components: (1) infrastructure-specific and (2) experiment-specific. Infrastructure-specific MSpec may consist of an extensible list of attributes corresponding to the infrastructure and would be available to ALL experimenters while experiment-specific MSpec would depend very much on the type of experiment being performed and would be made available to the requesting experimenter only. Measurement spec (MSpec) will be specified using 4-tuples: [Parameters; Frequency; Type; Instrument Location]

Measurement spec can be applied to an experiment (handle: Experiment_id) using the function:

ASSIGN_MEASUREMENT (Experiment_id, MSpec)

Here are some examples of **infrastructure-specific** MSpec:

Flow and Packet level measurements:

- Parameter=Transmission Power at Base Station; Frequency=Every 10 packet; Type=Mean; Instrument Location=Node_id

Mobility measurements:

- Parameter=Number of handoffs at Base Station; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Number of ESSIDs Heard; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

Hardware measurements:

- Parameter=Queue length at routers; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Buffer overflows at routers; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

Here are some examples of **experiment-specific** Mspec:

Flow and Packet level measurements:

- Parameter=Throughput; Frequency=Per Minute; Type= (Mean, Std. Dev), Instrument Location=Node_id]: Measure “Mean” and “Std. Dev” of Throughput at Node_id every minute.

- Parameter=End-to-End Latency; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=End-to-End Jitter; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Header Information (MAC, Network, Transport); Frequency=Per packet; Type=Sample, Instrument Location=Node_id

- Parameter=Packet Retransmissions; Frequency=Per packet; Type=Mean; Instrument Location=Node_id

Mobility measurements:

- Parameter=Handoff latency; Frequency=Each handoff; Type=Sample; Instrument Location=Node_id

- Parameter=Location co-ordinates; Frequency=Each handoff; Type=Sample; Instrument Location=Node_id

Wireless measurements:

- Parameter=Number of control packets; Frequency=Per data packet; Type=Mean; Instrument Location=Each link

Please see Appendix for a complete list of parameters.

A question that needs to be resolved is the architecture of instrumentation and measurement infrastructure. In other words, it is not yet clear how the collected RF measurements in fielded systems, RF controlled facilities, etc, are represented, stored, transported, archived and analyzed.

Another point to note is that collection of measurement is orthogonal to archival of measurements. Collection of measurement by experimenters can be either real-time or non real-time. Regardless, the experimenter has to specify for the measurement infra-structure what to collect and how often to collect. How long to archive the measurements is a policy decision, and there could be different policies for archiving infra-structure specific and experiment-specific measurements.

- Parameter=Jitter; Frequency=Per 10sec; Type=Mean,
- Parameter= Packet Loss, Frequency=Per 10sec; Type= Mean,
- Parameter=Handoff Latency; Frequency=Per minute; Type= Mean

Figure 6 Sample MSpec for measuring Overall Latency and Handoff time

3.3 Experiment Dynamics: Experiment Scripting Language (ESL)

ESL provides a scripting language for the experimenters to describe the dynamics of a given experiment. Note that ESL is an "optional" feature in that an experimenter has the option of controlling the dynamics of the experiment by changing parameters, such as, packet size, channel assignments, duration of the experiment etc. via ESL. For the sample experiment shown earlier,

the experiment dynamics consists of the remote wireless node roaming from one AP to the other. Providing mobility as a service to experimenters is a challenging problem. This can be done in several ways. For example, in a controlled grid-like setup, mobility may be emulated by switching packets through a discrete set of nodes in space at different times. Alternatively, one can use rovers to traverse preconfigured trajectories at specifiable speeds. The details of how mobility is provided as a service to the experimenters are beyond the scope of this document. The sample script written using ESL may look as follows.

```
#Start test traffic
whenAllInstalled()
  NodeSet['CL2'].startVoIPReceiver
  NodeSet['CL1'].startVoIPSender

#Wait for 5 minutes (300 secs)
  Wait 300

#Mobility emulation (We assume this to be an AP initiated roaming)
  NodeSet['AP2'].starthandoffInitiator
  NodeSet['AP3'].starthandoffAcceptor
```

Figure 7 Sample Experiment Script for the VoIP experiment

Note that handoffInitiator and handoffAcceptor are the applications or functions corresponding to handoff in the prototype library of the Resource Manager. Other option is simply to do whatever the experimenter wants to do by writing code that runs at the relevant nodes.

In future, ESL may also be augmented to dynamically configure experiment parameters based on the run-time measurements collected and fed back to ESL (for example, if the measured handoff latencies are higher than a threshold as determined from the run-time measurements, there may be feedback to AP2 to trigger a "smart roaming" mode)

The ESL in a sense captures the entire state of the experiment which can be used for future runs by the same experimenter as well as contributed to the GENI tool manager, so the results may be validated by other experimenters if desired. ESL thereby facilitates repeated experimentation thereby increasing the credibility of results in general.

3.4 Experiment cleanup

Upon the termination of the experiment (in case of short term experiments), the GMC can clear the state machine on the nodes as follows

CLEAR_EXPERIMENT (Experiment_id)

3.5 Putting it all together

Experiment control can thus be viewed as a multi-dimensional process:

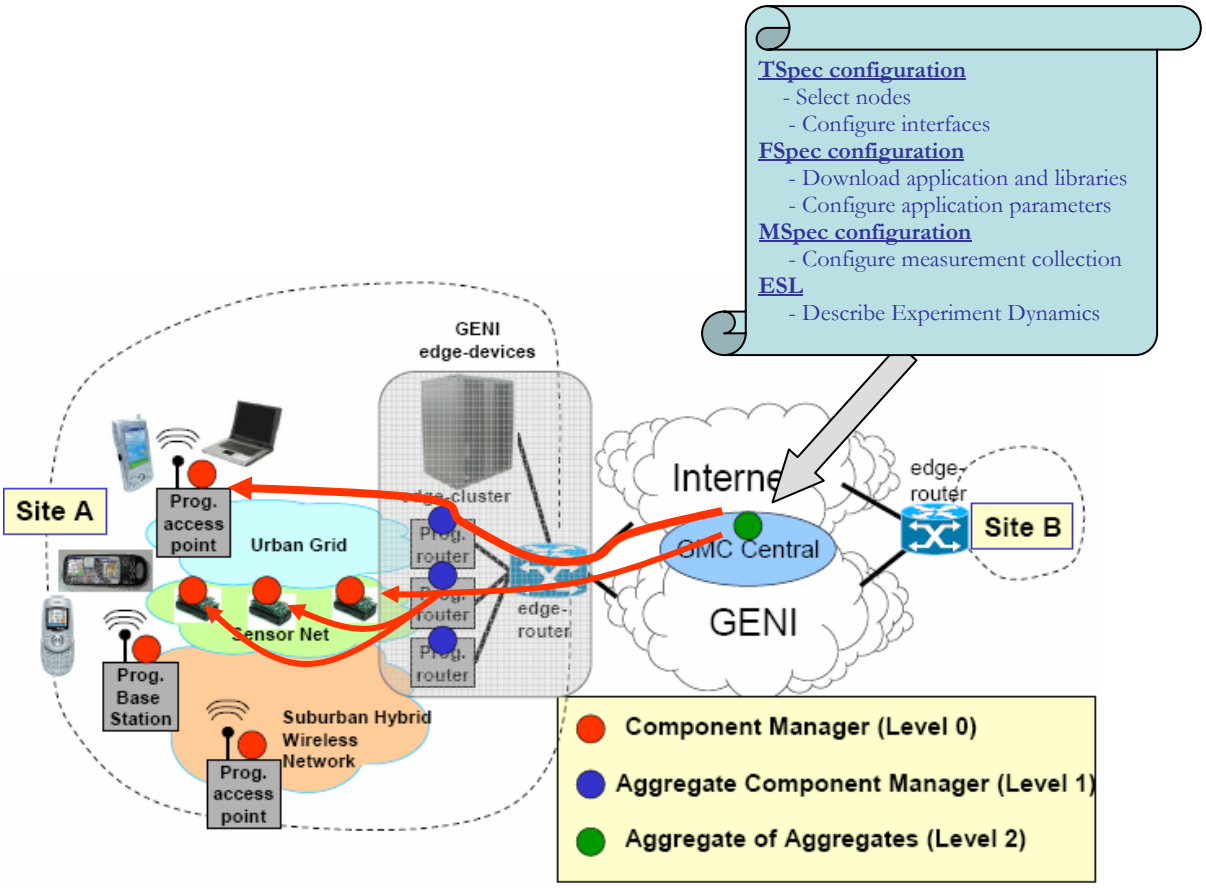

Figure 8 Overall experiment lifecycle

First, there is the "slice embedding" stage that gives the researcher a point of presence (a sliver) on a set of components. Over time, some of these components will crash and reboot; over time the researcher will add/remove components from this set.

Second, there is the “experiment definition” stage which involves a separate set of mechanisms/ tools that will be used to move code and experiment configuration info to the current set of slivers, and download data from the current set of slivers. These tools don't just run once, but are used whenever the researcher indicates a need, and/or run periodically so as to account for the coming and going of components.

Third, there is a “resource allocation” stage which involves a separate service that will allocate resources to the slivers belonging to a slice. When this allocation lasts for, say, an hour, it's likely the researcher has an experiment running during that time. During the rest of the time, the slice is still available, and has default resources assigned to it, so the researcher can access the components to do maintenance, dry runs, etc. Long- running services will run with only default (best effort) resources, have hard allocations that last longer than an hour, or continually interact with the allocation service to acquire (and subsequently release) resources.

4 Management System for Wireless GENI

Management system of Wireless GENI will be responsible for managing the wireless GENI subnets and for providing tools and resources to the GENI users to facilitate running of their experiments. The main components of the management systems are: Resource Manager, Component Manager, and Tools Manager.

4.1 Resource Manager

The role of Resource Manager is to maintain the available inventory of resources in a network and provide controlled access to these shared resources. In order to achieve this goal, it should start with the multi-dimensional resource space where multiple dimensions refer to frequency, channels, time-slots, connectivity between nodes, etc. and keep track of the resources allocated to the currently active slices. Based on this information it can update the available inventory of resources for future allocations.

The main utility provided by resource manager to a GENI user will be a tool for discovering available resources in a wireless subnet at a given point of time. Resource Discovery goes hand in hand with the specification of nodes. Essentially, an experimenter, before specifying the needs of the experiment, would benefit largely by triggering a resource discovery phase, at the end of which she will know the constraints within which the nodes (slivers) must be chosen for her experiments. In the absence of this support, an experimenter may use trial and error to map their logical network (slice) on to the underlying physical network, and end up wasting a lot of time. Resource Manager would present a feasible set of nodes and the connectivity between them. This information may also be presented using a Graphical User Interface (GUI) that a GENI user might use to create her slice. The biggest advantage of the GUI would be to provide a visual representation of the available topologies in an operational network.

Sample commands for RM may be:

Node Query: List the available nodes.

Channel Query: List available non overlapping channels.

Nodes Request: The user requests Node allocation for experiments. This request can either be of the form "give me 5 nodes" in which case any 5 available nodes will be assigned by the RM; or it could be "give me the following 5 nodes..." in which case they will be allocated only if all those are available. This generalizes to “Node and Topology” specification (TSpec) as described in section 3.2.1 where one could either ask for a set of nodes connected in a given topology or could ask for a “specific” set of nodes connected in the desired topology.

Channel Request: The user will request channel and RM will allocate one from the available ones.

Further, we expect the RM to play an active role in enforcing the allocated configurations during experiments. While no details have been worked out yet, the aim is to have a software component on each node interacting with the driver to enforce the experiment configuration in concert with the RM.

The mapping algorithm to translate topology specification to actual resource allocation is beyond the scope of this document. Assuming such a mapping exists we discuss below some strategies for enforcing such a mapping and policing the usage of the virtual resources.

4.1.1 Enforcement strategies

Node-level enforcement

Node usage can be enforced using access control and VLANS. When a scheduled user gets access to a wireless substrate, the resource manager may create appropriate VLANS based on the resources allocated to that user; thereby restricting its ability to access other nodes.

Spectrum enforcement

- For spectrum monitoring, we need passive monitoring stations on all channels and then post-processing of logs. We will need to develop automated post-processing scripts that look at the logs and report users on every channel. This can then be monitored in real-time by the administrator using a web interface.

- The challenge is to create a system that can do this monitoring based on information gleaned from 802.11 frames and detect spurious activities without significant false alarms.

- How do we map from frame/packet-level information to user-level information?

- Alternatively, RM configures nodes before allowing access to the nodes and users are provided restricted access to the nodes.

- Modified drivers use dynamic configuration file from RM at the start of the experiment slot

- Modified drivers restrict the experimenter to using only assigned frequencies.

- Explore the use of virtual machine technology (such as Xen) in implementing this.

Sample RM output

A sample resource allocation policy generated and distributed by the RM may look like this:

```
Content-Type: text/xml;
<?xml version="1.0" encoding="utf-8"?>
<Policy File>
<info><Reservation Status for a particular reservation_id
/></info>
      <Resid><Resid=”12345“/></Resid>
      <User_name><User_name=”johndoe“/><User_name>
      <nodes>
        <node><id=”node1”/></node>
        <node><id=”node2”/></node>
        <node><id=”node3”/></node>
        <node><id=”node4”/></node>
      </nodes>
      <startdaytime>
        <stime><t1=”2007-01-22 22:00:00”/></stime>
      </startdaytime>
      <enddaytime>
        <etime><t2=”2007-01-22 23:30:00”/></etime>
      </enddaytime>
      <channelres>
        <channel><ch=”b1”/></channel>
      </channelres>
</Policy File>
```

Figure 9 Sample RM policy file

4.2 Component Manager

Role of Component Manager is to manage the components (nodes and interfaces) in a wireless subnet. Specific functions that may be performed by the component manager are:

1. POWER_ON (List of (Node, Interface))

Example: POWER_ON ((Node node1, Interface 1), (Node node2, Interface 1))

Special case: POWER_ON (Node node1) ➔ all interfaces of Node node1 are POWERed on

POWER_ON (Interface 1) ➔ interface 1 of all nodes are POWERed on

2. POWER_OFF (List of (Node, Interface))

Example: POWER_OFF ((Node node1, Interface 1), (Node node2, Interface 1))

Special case: POWER_OFF (Node node1) ➔ all interfaces of Node node1 are turned off

POWER_OFF (Interface 1) ➔ interface 1 of all nodes are turned off

3. RESET (List of (Node, Interface))

Example: RESET ((Node node1, Interface 1), (Node node2, Interface 1))

Special case: RESET (Node node1) ➔ all interfaces of Node node1 are reset

RESET (Interface 1) ➔ interface 1 of all nodes is reset

4. GET-STATUS (List of (Node, Interface))

Example: GET-STATUS ((Node node1, Interface 1), (Node node2, Interface 1))

Special case: GET-STATUS (Node node1) ➔ Get status report of all interfaces of Node node1

GET-STATUS (Interface 1) ➔ Get status report of interface 1 of all nodes

5. ACTIVATE (List of (Node))

Example: ACTIVATE ((Node node1), (Node node2))

Special case: ACTIVATE (Node node1) ➔ activates Node node1

ACTIVATE (ALL Nodes) ➔ activates all Nodes

6. DEACTIVATE (List of (Node))

Example: DECTIVATE ((Node node1), (Node node2))

Special case: DECTIVATE (Node node1) ➔ deactivates Node node1

DEACTIVATE (ALL Nodes) ➔ deactivates all Nodes

7. GET-NODE (Status, Type)

Example: GET-NODE (Active, 802.11a)

Special case: GET-NODE (Active) ➔ Get all "Active" nodes

GET-NODE (802.11a) ➔ Get all nodes with 802.11a interfaces

4.3 Tool Manager

Role of Tool Manager is to provide a library of prototypes, tools, and utilities that GENI users can leverage for specifying and running their experiments.

4.3.1 Prototypes

Tool manager may provide prototype code for a variety of common functional roles, such as:

1. VoIP Client using G711
2. FTP source for 10MB file transfer/FTP sink

3. Forwarding Node

4. CBR UDP traffic generator with 1024 byte payload @ 1 Mbps/Sink

This will be available as in the form of FSpecs for the available prototypes that GENI users may use or customize them to suit their needs.

4.3.2 Applications

In addition, the tool manager can also store the repository of contributed applications which are available in the form of ASpecs.

1. Traffic Generator
2. Packet sniffer
3. Low level statistics collection tool (application that collects driver level statistics)

5 Miscellaneous

Dynamic Control: provides additional flexibility to a GENI user for running experiments. It has two dimensions. First, a user may want to "dynamically" change the "slice" by adding additional nodes (slivers) that weren't part of the original slice or even deleting nodes (slivers) that were part of the original slice. Second, the user may require to "dynamically" change the "experimental configuration of slivers" in his slice where the configuration change may be in the form of experimental parameters, code, measurement variables etc. The dynamic changing of experimental configuration of slivers can be effected via ESL. We may need an equivalent of ESL for dynamically changing the “slice” itself.

In fact, the first part of experiment control (namely, slice embedding) together with the third part of experiment control (namely, resource allocation) partially answer this question. However, for a fully satisfactory answer, more detailed investigation is necessary.

Stability: tries to answer the question of what happens if some of the nodes chosen by the experimenter are broken during uploading of code, and the upload fails? For example, say I want to have 100 nodes, but when time comes to run the experiment, 3 or 4 are broken. Does the experiment get cancelled?

The answer to this question will depend on the type of experiment. Certain experiments can run with a subset of nodes and as long as the results have the recorded topology of available nodes, it’d be just fine. However, certain experiments can not be run because some of the key components are missing.
Hence a tentative proposal is to use a two-level specification for experiments. At the highest level, the nodes would be classified as mandatory or optional. Then the types and number of

nodes that are mandatory would have to be specified followed by the types and number of optional nodes. Thus, if one or more of the mandatory nodes are down, the experiment would not be initiated while if the same thing happens with optional nodes, the experiment would continue. Note that this can be communicated to the experimenter as a part of the Resource Manager response to a specific request.

Scheduling Experiments in Advance: Experiments on GENI can be scheduled ahead of time via a web-based interface. For example, many researchers would like to say "give me 50 nodes in this topology for 2 hours - anytime in the next 3 days". In order to accommodate future bookings of resources, the reservation information would have to be used by the Resource Manager (section 4.1) to paint a picture of the available resources at any point of time. For example, if a user wants to choose a specific set of 10 nodes for his slice on Monday, the resource manager, while presenting available resources for Monday onwards would exclude those 10 nodes.

Appendix

Application Spec: Application Specifications can be used by application developers to describe the properties and capabilities of their application contributed to the GENI Tool Manager. As an example, we consider a widely used traffic generation tool, Iperf as our sample application. The ASpec for Iperf is described as follows

```
# Create an application representation from scratch
#

a = AppDefinition.create(‘iperf')
a.name = "iperfs"
a.version(0, 0, 1)
a.shortDescription = "Iperf traffic generator"
a.description = <<TEXT
Iperf is a traffic generator for TCP and UDP traffic. It contains
generators
producing various forms of packet streams and port for sending
these packets via various transports, such as TCP and UDP.
TEXT

a.addProperty('udp', 'Use UDP, otherwise TCP by default', nil, String,
false)
a.addProperty('client', 'Run as client', nil, String, false)
a.addProperty('port', 'Sender port number', nil, Integer, false)
a.addProperty('window', 'TCP Send Window Size', nil, Integer, false)
a.addProperty('len', "Payload length (bytes)", nil, Integer, false)
a.addProperty('bandwidth', "Offered load for UDP", nil, Integer, false)
a.addProperty('time', "Duration of traffic generation(secs)", nil, Integer,
false)

a.addMeasurement(‘Offered Load’, 'Float’)

a.path = "/usr/bin/iperf"
```

Examples of derived applications from Iperf could be VoIP G.711 sender (based on the underlying Iperf configured to use CBR traffic with UDP payload of 160 bytes at 50 packets per second)

```
# Create a derived VoIP G.711 sender based on generic Iperf application

p = Prototype.create(“voipsender")
p.name = "Iperf Based VoIP Sender"
p.defProperty('udp', 'Protocol to use')
p.defProperty('client', 'Host to send packets to')
p.defProperty(‘rate', 'Number of bits per second', 64000)
p.defProperty('len', 'Size of packets in bytes', 160)

# Here, we bind the derived application to the actual application
(Iperf)
a = addApplication(“iperf")
a.bindProperty('udp')
iperfs.bindProperty('client')
iperfs.bindProperty('bandwidth','rate')
iperfs.bindProperty('len')
```

The FSpec (described in 3.2.2) can use this derived application to select the functionality of a set of nodes as follows

FSpec = {

(Type=VoIPSender; Node_Set = (node1, node2);

}

This will configure nodes node1 and node2 as VoIP senders

Infrastructure-specific Mspec:

Flow and Packet level measurements:

- Parameter=Packet Loss Rate; Frequency=Per 15 Minutes; Type=Mean; Instrument Location=Each Link: Measure “Packet Loss Rate” at Each Link every 15 Minutes.

- Parameter=Packet Transmission Power; Frequency=Per packet; Type=Mean; Instrument Location=Node_id

Mobility measurements:

- Parameter=Handoff Frequency; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Number of ESSIDs Heard; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Association Message; Frequency=Per Minute; Type=Sample; Instrument Location=Each link

- Parameter=Latency between Association Requests; Frequency=Per Minute; Type=Sample; Instrument Location=Each link

Wireless measurements:

- Parameter=Bit Error Rate; Frequency=Per Minute; Type=Mean, Instrument Location=Each Link

- Parameter=SNR; Frequency=Per second; Type=Sample; Instrument Location=Each Node

- Parameter=Node Pair Delay; Frequency=Per Minute; Type=Sample; Instrument Location=Each Node

Experiment-specific MSpec:

Flow and Packet level measurements:

- Parameter=Throughput; Frequency=Per Minute; Type= (Mean, Std. Dev), Instrument Location=Node_id]: Measure "Mean" and "Std. Dev" of Throughput at Node_id every minute.

- Parameter=End-to-End Latency; Frequency=Per minute; Type=Sample; Instrument Location=Node_id

- Parameter=End-to-End Jitter; Frequency=Per minute; Type=Mean; Instrument Location=Node_id

- Parameter=Header Information (MAC, Network, Transport); Frequency=Per packet; Type=Sample, Instrument Location=Node_id

- Parameter=Flow Length; Frequency=Per flow; Type=Sample; Instrument Location=Node_id

- Parameter=Packet Retransmissions; Frequency=Per packet; Type=Mean; Instrument Location=Node_id

- Parameter=Packet Size; Frequency=Per packet; Type = Mean; Instrument Location=Node_id

- Parameter=Packet Transmission Power; Frequency=Per Packet; Type=Mean; Instrument Location=Node_id

- Parameter=Data Rate; Frequency=Per Packet; Type=Sample; Instrument Location=Node_id

- Parameter=Flow Inter-Arrival Frequency; Frequency=Per Minute; Type=Mean; Instrument Location=Node_id

- Parameter=Backoff Period per Data Packet; Frequency=Per Packet; Type=Mean; Instrument Location=Node_id

Mobility measurements:

- Parameter=Handoff latency; Frequency=Each handoff; Type=Sample; Instrument Location=Node_id

- Parameter=Location co-ordinates; Frequency=Each handoff; Type=Sample; Instrument Location=Node_id

Wireless measurements:

- Parameter=Number of control packets; Frequency=Per data packet; Type=Mean; Instrument Location=Each link

- Parameter=Bit Error Rate; Frequency=Per packet; Type=Mean; Instrument Location=Node_id