# $S^3$ Monitor Version 1.0 Specifications and Integration Plan

Please direct comments regarding this documentation or the $S^3$ Monitor software to `fahmy@cs.purdue.edu` and `puneet.sharma@hp.com`.

# 1 Overview

The Scalable Sensing Service ($S^3$ Monitor) provides basic management services for users to take controlled measurements between GENI nodes. A web interface is provided to the user for scheduling and initiating measurements, managing ongoing measurements, and retrieving measurement results. The $S^3$ Monitor service manages the dissemination of schedules to nodes in the GENI slice and retrieval of measurement results on behalf of the user. Measurement results are stored by the system for later reference until purged by the user.

A GENI experimenter may deploy the $S^3$ Monitor service to a GENI experiment and use its facilities to collect network measurements within the experiment slices.

# 2 Terminology

The following definitions give the normal meaning of terms used in this documentation, including GENI terms as applied to $S^3$ Monitor. In places where specific usage differs from this section, the differences will be explained in the documentation.

**HRN** The Human-Readable Name for a GENI resource.

**Slice** A GENI slice created via a Slice Authority. The normal $S^3$ Monitor identification of a slice is by its HRN.

**Sliver** A GENI sliver create via a Component Manager or Slice Authority. The normal $S^3$ Monitor identification of a sliver is by its HRN.

**Interface** A network port (actual or emulated) on a GENI node which has been assigned an IP address at sliver creation. $S^3$ Monitor identifies interfaces by their IP address.

**Manifest** An XML document provided by the Component Manager or Aggregate Manager at the time of sliver creation, which describes the resources available in a sliver.

**Measurement** An invocation of a measurement tool such as ping or traceroute.

**Node, GENI Node** A physical or virtual machine allocated to a GENI Sliver.

**Sensor Pod** A bundle of $S^3$ Monitor software which is installed on each GENI node that will perform measurements on behalf of $S^3$ Monitor.

# 3 System Architecture

$S^3$ Monitor includes two primary components (depicted in Figure 1):

1. *Sensor pods*: A sensor pod is a light-weight web service-enabled framework which hosts measurement sensors (*e.g.*, ping, pathChirp, tulip). The sensor pod needs to run on each node which serves as an end point (source or sink) of a measurement. The sensor pods can be deployed on GENI nodes, where a GENI node is a reserved node used by a GENI experimenter.

2. *Web application*: This is the management application which triggers measurements on the sensor pods and collects data. The web application (portal) is written in Java. Users submit measurement requests through this portal. The web application can be installed on any host which can establish communication with the GENI slice nodes that will perform the actual measurements. This includes installation on a GENI node, or installation on a third-party machine. Since the expectation is that the web application is a long-lived service, installation on a GENI node is not recommended; rather, it should normally be installed on a machine which is used to manage the GENI nodes across slice instantiations.

# 4 Sensor Pod Architecture

The sensor pod is a web service-enabled framework to host sensors (*e.g.*, sensor for delay, loss, available bandwidth, bottleneck capacity). Figure 2 depicts the components of the sensor pod. The sensor pod runs CGI scripts on an http server (BOA). The CGI scripts are written in Python, and contain the framework code for plugging in the sensors. The sensors extend the interface provided by $S^3$ Monitorto include any new measurement tool. The sensor execution, data collection, and clean-up logic should be included in the code of each sensor. The sensor code needs to be developed using Python in a maximum of five files. These files can be uploaded to each node along with an xml schema defining the sensor. The schema for the xml is provided with $S^3$ Monitor.

The sensor pod includes the modules described in the following subsections.

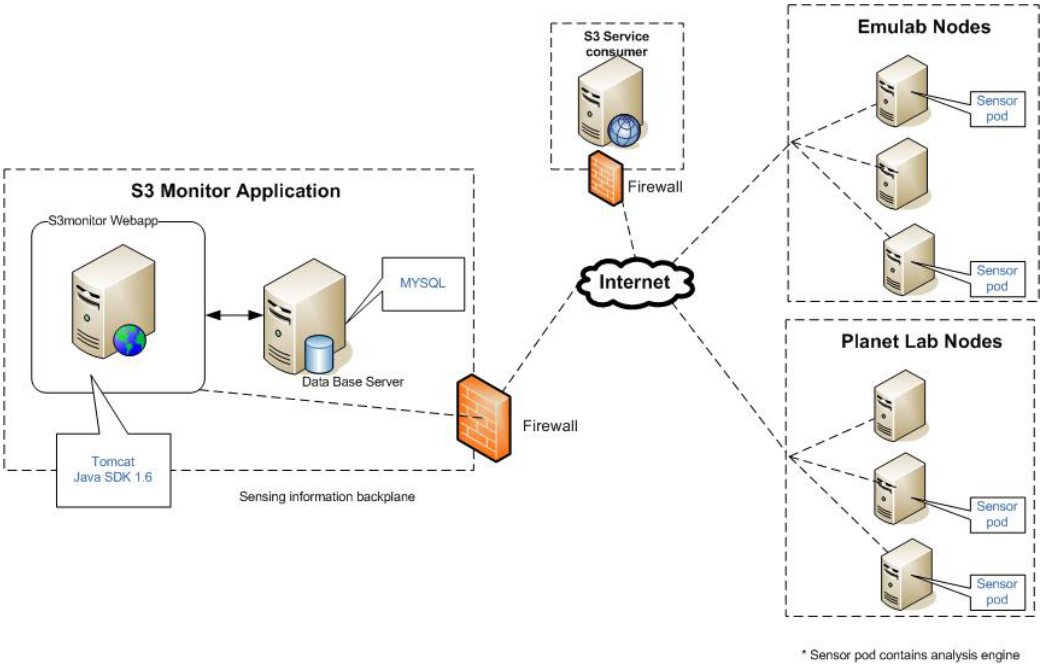


Figure 1: $S^3$ Monitor architecture

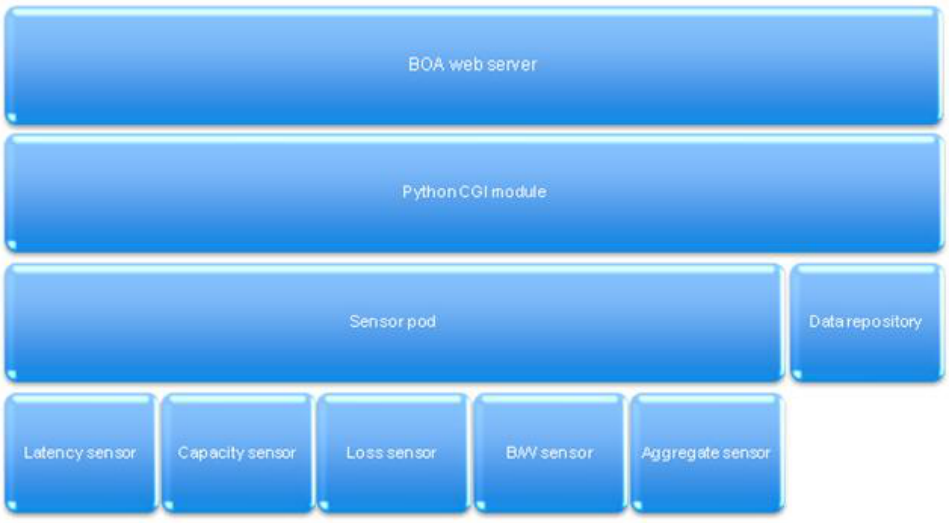


Figure 2: Sensor pod components

## 4.1 Boa Server

Boa is a single-threaded HTTP server. Boa does not fork a copy of itself or spawn a thread to handle each incoming connection, but rather internally multiplexes the connections. Boa only forks for CGI programs, automatic directory generation, and automatic file gunzipping, each of which must be a separate process. The primary design goals of Boa are speed and security.

## 4.2 Python CGI Module

A CGI script is invoked by an HTTP server, usually to process user input submitted through an HTTP request.

Most often, CGI scripts live in the server special cgi-bin directory. The HTTP server places all information about the request in the script shell environment, executes the script, and sends the script output back to the client.

This module handles a number of cases and provides a simpler interface to the Python script. It also provides a number of utilities that help in debugging scripts.

## 4.3 Sensor Pod Modules

This collection of modules includes the instant measurement module, configure periodic measurements module, remove periodic measurements module, and measurement results module.

### 4.3.1 Instant measurement module

This module triggers instant measurements, *e.g.*, for latency, loss, or bandwidth measurements between two nodes. The user can supply the parameters for the measurement tool. The module then processes the input parameters, invokes the requested command, triggers the measurement, and returns the result through the response object.

### 4.3.2 Configure measurements module

This module configures periodic measurements at the specified time intervals, and generates an identifier for each periodic measurements request. “crontab” is scheduled with the unique identifier for the requested time interval. The periodic measurements module runs the crontab, obtains the results, and stores them into the data repository.

### 4.3.3 Remove measurements module

This module removes the scheduled process using the identifier generated for every crontab entry as described above.

### 4.3.4 Measurement result module

The result stored in the data repository is retrieved by this module.

## 4.4 Data Repository

Measurement data is stored into the data repository.

## 4.5 Sensors

Sensors are invoked by $S^3$ Monitorto measure properties such as latency, loss, bottleneck capacity, and available bandwidth between two nodes. Sensors can be classified as basic sensors or aggregate sensors.

The sensor developer must provide the sensor functionality and an interface (wrapper) for each sensor. The sensor interface must include three functions: performing the measurement, obtaining the measurement results, and cleaning up the measurement-related files. The sensor interface is written in Python. The user needs to write the Python module following a template. This provides extensibility (plug-ability) to the sensor pod architecture.

### 4.5.1 Basic sensors

Basic sensors are independent sensors that can be triggered to obtain the requested measurement result directly, *e.g.*, the latency sensor ping.

### 4.5.2 Aggregate sensors

Aggregate sensors are a collection/group of sensors. Once a user invokes a sensor, the sensor may trigger another sensor and so on to obtain the result, *e.g.*, capacity sensor (pathrate).

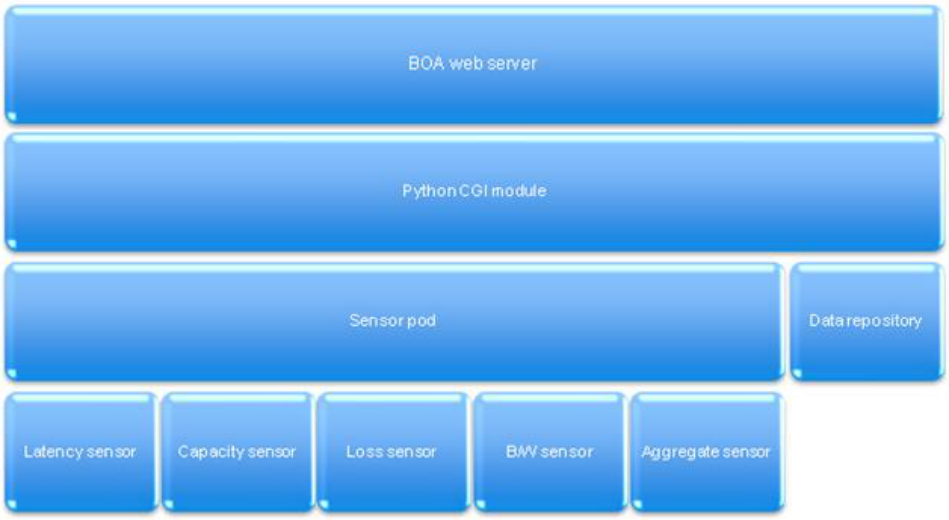


Figure 3: Web application components

# 5 Web Application Architecture

The web application (portal) provides management facilities to control the sensor pods and maintain measurement data. Figure 3 depicts the components of the web application. The application runs servlets on a Java web server (Tomcat). The servlets act as interfaces to the web pages to process user requests and respond accordingly. All responses from each servlet are in JSON format. The main processing work is performed in the business logic part. There are dedicated service modules to provide specific services (*e.g.*, DatabaseService, FileUploadService). Database access is through Hibernate (ORM). The web pages are written using JSP technology, and Javascript is used as the client-side scripting framework.

The web application includes the modules described in the following subsections.

## 5.1 Tomcat server

Apache Tomcat is an open-source servlet container that implements the Java Servlet and the Java Server Pages (JSP) specifications, and provides a "pure Java" HTTP web server environment for Java code to run. $S^3$ Monitor requires the Tomcat server to run the JSP/servlets.

### 5.1.1 Web user interface

The user interface of $S^3$ Monitor is written in JSP along with HTML and Javascript, so as to separate the page logic from the visual elements. In Javascript, the jQuery library and other plug-ins have been used. The design of the front end is done using CSS.

### 5.1.2 Server backend

The server backend processes the requests from the user interface module and responds. This includes:

1. **Filter:**

   A filter is an object that performs filtering tasks on either the request to a resource (a servlet or static content), or on the response from a resource, or both. $S^3$ Monitor uses "Filter" to provide access control to the service.

2. **Servlets:**

   A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. $S^3$ Monitor has one servlet for each JSP page.

3. **Business Logic:**

   The business logic module is the actual processing unit of the $S^3$ Monitor web application. All $S^3$ Monitor functionality is controlled from this module. The business logic leverages the service module to process requests, perform tasks, and handle errors. It also converts the response to the appropriate format (JSON) before transmitting it to the user interface.

4. **Service Classes:**

   The service module implements all tasks that $S^3$ Monitor needs to perform. The tasks are grouped into classes as follows:

   - *Utilities:*
     All commonly-used functionality in $S^3$ Monitor is defined in this module.

- *Entities:*
  In this module, different entities of the service are defined and mapped to the respective database tables.
- *Response Object:*
  This module contains the Java objects to be mapped to JSON responses.
- *Exception:*
  In this module, all the exceptions in the service are defined.
- *Third-party libraries:*
  The $S^3$ Monitor web application uses several open source libraries for facilitating common application requirements, *e.g.*, log4j, dom4j.

# 6 Integration with Control Frameworks and other GENI Projects

$S^3$ Monitor integrates with the GENI control frameworks to identify experimenter resources within GENI slices. The $S^3$ Monitor Sensor Pod installs on GENI-allocated resources to perform active network measurements. Integration with other GENI projects revolves around management of $S^3$ Monitor sensor behavior and sharing of measurement data.

## 6.1 Control Frameworks

Communication between $S^3$ Monitor and the GENI control frameworks is through *RSpec*s:

- $S^3$ Monitor Sensor Pods can be deployed on ProtoGENI (`http://www.protogeni.net/trac/protogeni`) experiment nodes by including appropriate configuration in RSpec requests. Automated deployment on other aggregate resources is in progress.
- The $S^3$ Monitor web application uses RSpec manifests to identify possible Sensor Pod locations within an experiment, and to configure the network interfaces on Sensor Pod nodes.

Integration with GENI-allocated resources is in the form of the $S^3$ Monitor Sensor Pod. The Sensor Pod must be ported to the OS images that run on GENI resources. Sensor Pods for the default ProtoGENI image and the PlanetLab (`http://groups.geni.net/geni/wiki/PlanetLab`) image are provided already. More Sensor Pod ports (including ports to additional ProtoGENI images as well as ORCA (`https://geni-orca.renci.org/trac/wiki`) EC2 images) will be provided in the future, as well as an automated or semi-automated mechanism for porting the Sensor Pod to images which fulfill a basic set of requirements.

## 6.2 Other GENI Projects

$S^3$ Monitor can be integrated with several GENI measurement and management projects to provide measurement capabilities which those projects lack. Integration with other GENI projects may also provide measurement or management capabilities to $S^3$ Monitor which it lacks.

- There are plans underway to deploy $S^3$ Monitor sensor pods to INSTOOLS (`http://groups.geni.net/geni/wiki/InstrumentationTools`) instrumentized hosts in order to provide active measurement monitoring information to INSTOOLS users, as the current version of INSTOOLS provides only passive measurements.

- The same integration with INSTOOLS will provide a process for installing $S^3$ Monitor sensor pods to any host for which the INSTOOLS instrumentize mechanism has been implemented, including non-ProtoGENI aggregate hosts, via Flack (`http://protogeni.net/flack`).

- Data collected by $S^3$ Monitor deployments can be provided to GENI Measurement Data Archives. Integration with the existing Digital Object Registry (`http://groups.geni.net/geni/wiki/DigitalObjectRegistry`) Measurement Data Archive prototype is planned, as well as eventual implementation of the GENI Measurement Data Schema for data communication and storage once it is finalized.