Architectural Design and Specification of the INSTOOLS Measurement System*

James Griffioen, Zongming Fei, and Hussamuddin Nasir
Laboratory for Advanced Networking
University of Kentucky
Lexington, KY 40506

December 2009

*This material is based upon work supported by the National Science Foundation under Grant No. (CNS-0834243). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of GPO Technologies, Corp., the GENI Project Office, or the National Science Foundation.

Contents

1 Background

Spiral 1 of the GENI project has primarily been focused on the problem of building *control frameworks* [8]. Control frameworks form the basis for the GENI network, allocating and initializing the network resources (slivers) used by a GENI experiment (slice). However, allocating and initializing a slice's resources is only the first of several steps in an experiment.

An important, and perhaps equally difficult step, is the step of deploying an *instrumentation and measurement* system that assists a user in monitoring and capturing the behavior of his or her network (slice). Not only is the instrumentation and measurement system necessary for measuring the system (e.g., to numerically compare its performance to other alternatives), but it is needed to track and understand the network's behavior (e.g., to determine if the network is executing correctly or not).

The process of monitoring an experiment can roughly be divided into three stages: (1) *Instrumentation* refers to the stage in which data capture tools are deployed at particular points in the network to capture information of interest to the user, (2) *Measurement* refers to the stage in which data is collected from a running experiment and (possibly) combined with other data to form a more integrated data set, and (3) *Presentation* refers to the stage in which the data is processed into a semantically meaningful visual display of the data that is then presented/displayed to the experimenter.

It should be noted that a variety of hardware and software tools already exist that can be use to *instrument* various components of the network components of the network. Tools such as tcpdump [5] and port monitors on routers can be used to capture packets while SNMP-based software [10] can be used to capture network state information. Moreover, GENI projects such as the GENI Instrumentation and Measurement System (GIMS) at the University of Wisconsin [12, 9] are working on new high-performance link sensors that can capture and record packets at line-speed. Similarly, software exists to collect the captured data, process it and even present it in fancy graphical displays.

However, at present, relatively little attention has been given to the problem of helping experimenters (users) set up and operate the instrumentation and measurement infrastructure needed to monitor their experiments. Much like control frameworks make it easy to specify, configure, and deploy network infrastructure (i.e., slices), users need a *instrumentation and measurement framework* that make it easy for them to specify, configure, and deploy the instrumentation and measurement infrastructure needed to monitor and observe the behavior of their network.

The *Instrumentation Tools (INSTOOLS)* project at the University of Kentucky aims to fill this gap, by automatically setting up and initializing experiment-specific network measurement and monitoring capabilities on behalf of users. Much like the control frameworks try to hide the details of resource allocation and setup from users, the goal of the INSTOOLS project to is to hide the details of instrumentation setup from users so that users do not need to be experts in system administration or in network management in order to "see" what is going on with their experiment. Instead, our instrumentation tools are designed to provide users with an easy-to-use web-based user interface that use tables and graphs to visualize the runtime behavior and measurement data collected from the user's experiment.

The goal of the INSTOOLS project is *not* to reinvent the wheel (i.e., reinvent or improve on existing instrumentation tools and measurement software), but rather to develop ways to automatically deploy these tools, automatically collect data from the tools, and then process and display

the data to the users in visually useful ways. In other words, our goal is to leverage these tools on the user's behalf, automatically setting up a complete instrumentation and measurement infrastructure for the user without the need for the user to learn or understand the details of all these various tools.

It should be noted that a related document, namely the GIMS *Requirements and Specifications for the Instrumentation and Measurement Systems for GENI* [9] is complimentary to the system described in this report. While the GIMS document outlines the components that comprise an instrumentation and measurement system, the primary focus is on defining the sensors used to capture data (instrumentation) and on archiving the data collected by the instrumentation component. In that sense, the architectural design we present in this report is complimentary to the specification described in the GIMs report. As noted earlier, our goal is to leverage instrumentation infrastructure such as that described in the GIMs report and enhance it by adding the automation components that make it easy to specify and deploy an instrumention and measurement system.

2 Design Goals

The overall goal of the INSTOOLS project is to make it easy for users to set up and use instrumentation and measurement tools in their experiments. Users should not need to be experts in the use of the plethora of existing instrumentation tools in order to monitor the behavior of their experiments. Consequently, our design focuses on the problem of selecting, deploying, initializing, running, and displaying output from existing instrumentation tools on the user's behalf.

Our design is motivated, in part, by the desire to make the GENI infrastructure usable by students in networking and operating systems classes. Because students are not (in general) seasoned network or system administrators, they do not necessarily know what information is even available, let alone how to instrument the system to obtain that information. Consequently, a key assumption of our system is that the user will likely offer little help in the way of making instrumentation decisions. (In fact, even experienced network/system administrators usually prefer it if they can make fewer decisions about what to monitor, using defaults for most decisions). As a result, it is our task to make intelligent decisions–or at least reasonable guesses–about the instrumentation that will be most beneficial to the user. While some users will have unique instrumentation requirements, most users–particularly novice users like students–will benefit from a common set of instrumentation functionality. Although hooks need to be available to capture unique information in certain experiments, the primary focus of our INSTOOLS project is on the design of a generally-useful network instrumentation and measurement infrastructure that can be deployed as part of every experiment.

Our design also leverages our experience with the Edulab [16, 6] system which was designed to help networking and operating system students monitor their running Emulab experiments. We observed that *network state* information is typically more important than *packet trace* information when it comes to identify the cause of problems. While there are times when it is important to capture and examine packet traces, many problems can be identified and corrected simply by looking at network state such as routing tables, packet counters (e.g., interface counters), cache entries (e.g., ARP, DNS, and NAT tables), processor load averages, memory statistics, host configurations, firewall rules, etc. Because network state is particularly useful and often easier to capture and record than packet traces, our focus is slanted more toward capturing network state

than it is toward capturing packet traces.

Given these assumptions, we defined several design goals for the system:

Ease-of-Use: We want the system and interface to be user-friendly – helping users with the experiment without burdening the users. The monitoring facilities should be set up on the user's behalf so that users do not need to deploy any software or services. We want a very simple interface without requiring users to select among many possible "options". The interface should show what most users want, with the potential to be extended to include information needed by unique experiments.

Slice-specific Monitoring: Most users are primarily interested in the behavior and performance of their experiment, with little (or at least less) desire to know about the performance of the GENI testbed as a whole. Consequently, our focus is on developing slice-specific monitoring infrastructure, that is tailored to capture, record, and display only information associated with that slice. Although users may opt to make available some or all of their information to globally-shared monitoring services and repositories (e.g., the GMOC [2]), our primary focus is on developing a slice-specific monitoring framework.

Leverage Existing Tools: We want to leverage off-the-shelf software as much as possible, instead of starting everything from scratch. Over the years, many monitoring tools have been developed. Most common monitoring tasks can be performed using standard tools that are well-documented and can be easily incorporated into the architecture.

Leverage Control Framework Functionality: Instead of designing a completely new system to oversee and allocate "measurement resources", our system should leverage and build on the resource allocation capabilities provided by the existing GENI control frameworks. In other words, the instrumentation and measurement infrastructure should itself be built from resources generally available in GENI.

Separate Measurement Plane: We want to minimize the load imposed on the data-plane by the measurement plane. Some monitoring operations, such as capturing detailed packet traces, can impose a heavy load on the network if the data must be centrally processed or centrally archived. We want measurement traffic to interfere as little as possible with the data plane of the running experiment. Moreover, even though some types of instrumentation (e.g., packet counters) do not produce large amounts of data individually, in aggregate the load could affect an experiment if the measurement information has to compete with the data plane traffic.

Virtualization Support: We want to be able to monitor both physical and virtual resources. As virtualization becomes an essential element of next generation network testbed environments, the measurement infrastructure should be able to get information about virtual interfaces offered by virtual routers. Unfortunately, much of the existing monitoring software is not yet "virtualization friendly". Also we want to make sure that users only have access to measurement data of their own experiment, even if it is running on a virtualized machine sharing a physical node with other users.

Scalability: We want to make the monitoring tools scalable. Multiple experiments may be running simultaneously and can potentially have measurement data to be collected. We want to have

a separate measurement node for each experiment so that the capacity of the system for processing the measurement data increases proportionally to the number of the experiments having measurement data to be collected.

Extensibility: We want to be able to incorporate new instrumentation as it becomes available.

3 Architecture Overview

The INSTOOLS system is designed to automatically create the instrumentation and measurement infrastructure needed to monitor experiments. The resulting instrumentation and measurement network is responsible for capturing measurement data from that slice, combining and processing the measurement data, and presenting the data to users.

Because users are primarily interested in measurement data associated with their experiment, the INSTOOLS system dynamically creates *experiment-specific* instrumentation and measurement infrastructure for each experiment. Providing each experiment with its own instrumentation and measurement infrastructure has several benefits. First, it can be tailored to the topology and needs of each experiment, rather than requiring all experiments to get their monitoring data from a single (generic) shared instrumentation and measurement infrastructure. Second, it means that an experiment's measurement data can be kept private to the experiment – never being stored or processed by resources outside of the experiment. Third, the monitoring load imposed by experiments is distributed to the nodes that comprise the experiments, rather than placing the monitoring burden on a single shared infrastructure. Moreover, monitoring traffic can be kept local to the experiment. Together, distribution and localization of the monitoring load improves the overall scalability of the system.

The INSTOOLS software is invoked by the control framework each time a new experiment (slice) is created so that slice-specific instrumentation and measurement infrastructure can be created along with the normal slice infrastructure. Each instrumentation and measurement network is constructed from available GENI resources that are allocated when the experiment starts and deallocated when the experiment ends. We utilized two types of GENI resources to create the instrumentation and measurement networks: (1) data capture nodes, and (2) general purpose processors.

We call the data capture nodes *Measurement Points (MPs)*. A Measurement Point is a GENI resource that is capable of capturing measurement information. For example, a network router in the user's experiment might act as an MP, capturing network state information such as routing table information, ARP cache information, or other SNMP MIB information. An MP could also capture packet trace information such as tcpdump-style traces[5] or netflow traces[11]. Alternatively, an MP could be an end system, a sensor-net node, a mobile (adhoc) node, or any other type of resource that is part of the user's slice. In addition, an MP could be an allocatable resource that is not part of the user's experiment such as a high-performance packet capture device located on the GENI backbone capable of snooping packets from the user's slice.

Data captured by the MPs is then sent to one or more general-purpose nodes where the data is stored, processed, and made available for presentation. We call these nodes *Measurement Controllers (MCs)* (see Figure 1). The MCs for an experiment are automatically created for the user by the INSTOOLS software when the experiment is created. Whenever possible, data transmission between the MPs and the MC occurs over an external channel, one that is *not* part of the user's

experimental network–i.e., the network that connects the resources (slivers) in the user's slice. For example, in Emulab/ProtoGENI-based aggregates communication between the MPs and the MC occurs through a separate "control interface" on each node. In Emulab-based systems the control interface connects to a distinct (physical) "control switch". In ProtoGENI systems, a node's normal IP interface acts as its "control interface" (with GRE tunnels providing the "experimental network interfaces").

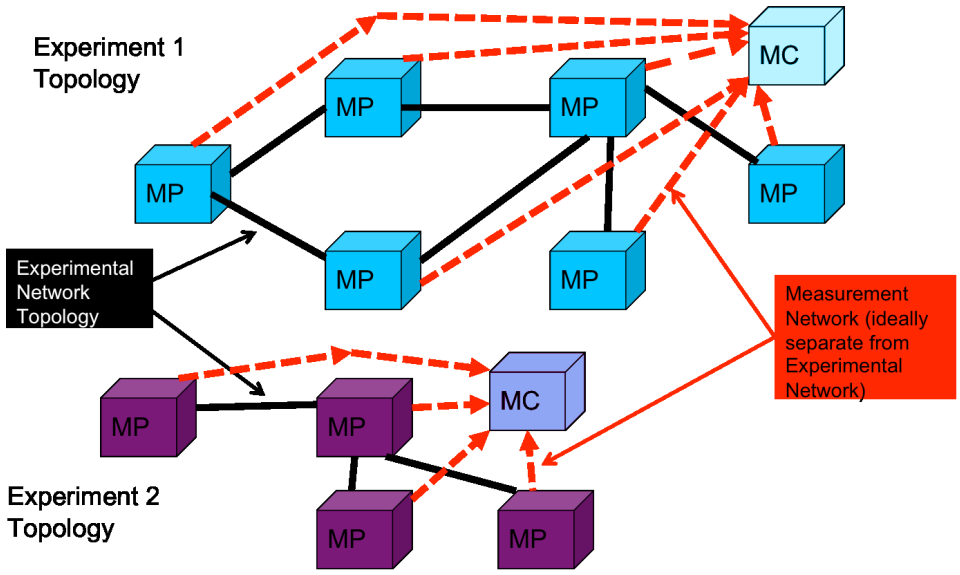

Figure 1: Each experiment/slice has its own MC and instrumentation and measurement network.

An experiment may have more than one MC. Multiple MCs may be required for scalability reasons and/or for management reasons. Because experiments can span multiple aggregates that are each under different authoritative control and (typically) offer different types of resources, our INSTOOLS architecture creates one MC for each aggregate in the experiment (slice). Creating one MC for each aggregate ensures that the measurement plane traffic stays within an aggregate, improving scalability. It also enables MCs to offer aggregate-specific monitoring features that are tailored to the type for resources offered by the aggregate. Additional MCs may be created within an aggregate to further enhance scalability.

Because an experiment may have multiple MCs, our INSTOOLS architecture includes an *MC Portal* through which users can interact with all the MCs that comprise their experiment. Similar in spirit to the ProtoGENI Clearinghouse which provides a single entry point into the system (but does not allocate any resources), the MC Portal does not perform any instrumentation or measurement task. Its sole purpose is to redirect requests for measurement information to the MC in the appropriate aggregate. In that sense the MC Portal provides users with a a single interface to all the MCs that comprise their experiment. Moreover the MC Portal can redirect users to the global archival server to locate archived (shared) measurement information. The architecture is illustrated in Figure 2.

Although MPs and MCs are at the heart of the INSTOOLS architecture, their existence is largely hidden from users by the control framework and MC Portals. MCs are automatically created

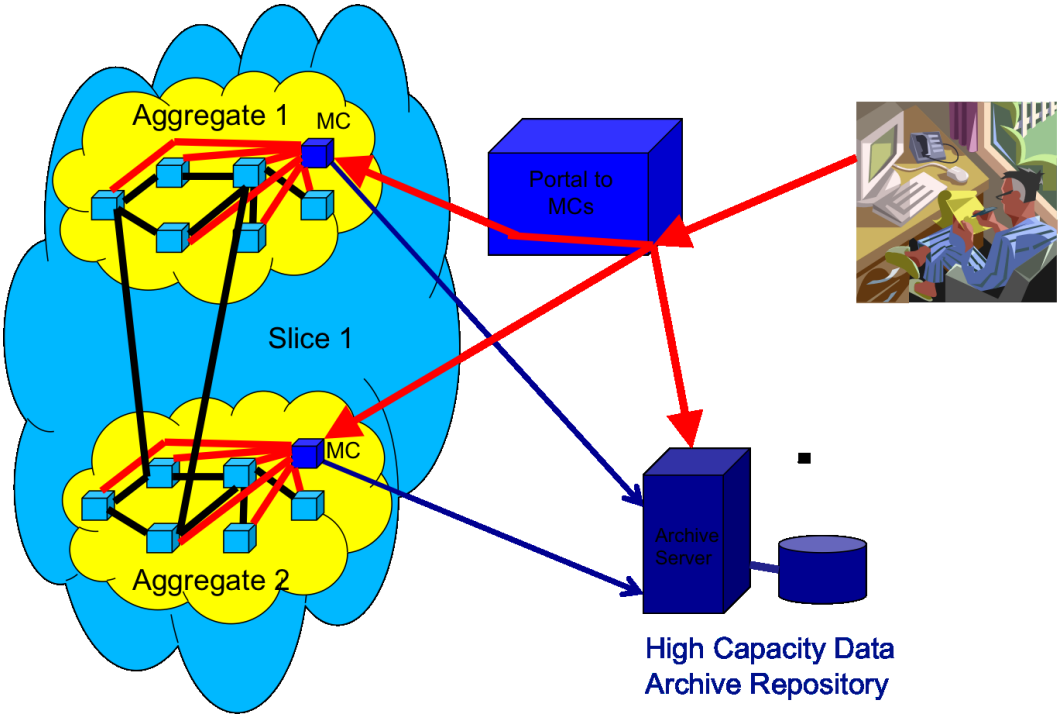

Figure 2: The MC Portal provides users with a single point of entry to their measurement data.

and initialized by the control framework (without user intervention) when the experiment is created. Because the MC Portal provides a single interface to all the MCs in an experiment (akin to a mashup of MCs), users only see the MC Portal not the individual MCs. When users use the MC Portal to enable and disable measurement points in the experiment, they need not know that the MCs are starting and stopping MPs to capture the specified data.

4 Architectural Components

Conceptually the INSTOOLS system is divided into seven components: (1) *Setup*, (2) *Capture*, (3) *Control*, (4) *Collection*, (5) *Storage*, (6) *Processing*, (7) *Access*, and (8) *Presentation*. These seven components are mapped onto the infrastructure described in the previous section (Section 3). Because they are logical components, they can be mapped to the infrastructure in many different ways. We currently map these functions to the infrastructure in the following way. *Setup* (and Teardown) is performed by the control framework when the experiment/slice is created–in our case, ProtoGENI [4] carries out this task. Data *capture* is the responsibility of the MPs. Control over what data is collected and when it is collected is determined by the MC which issues commands that the MPs carry out. Data *collection* is the responsibility of the MCs, which initiate the transfer of data from MPs to the MC. Data *storage* is the combined responsibility of the MPs, MCs, and shared archive repository. Data *processing* occurs at the MC. *Access* protection is also the responsibility of the MC, while *presentation* of the data is the result of work done by a web server and database on the MC creating web pages that are then rendered by the user's browser. Their

relationships and where they are implemented are illustrated in Figure 3. The following sections briefly describe the role and design of each of these components.

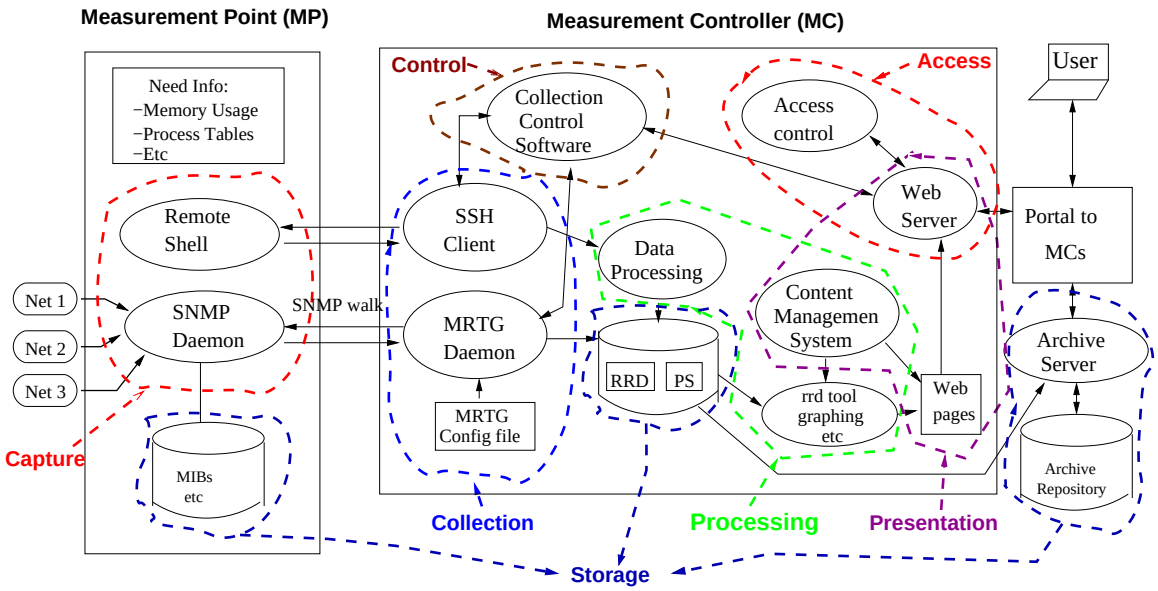

Figure 3: The Architectural Components of the INSTOOLS Toolset

4.1 Setup/Teardown

Users create new experiments by selecting GENI resources from the GENI clearinghouse [14] list. The desired resources are specified as an RSPEC [15] that is given to the component managers [1] executing in each of the aggregates [14]. It is the component managers' job to set up and initialize the resources specified in the RSPEC.

To automatically create the instrumentation and measurement infrastructure on the user's behalf, we added code to the component manager to invisibly change the RSPEC before it is processed by the component manager. In particular, we look for an available sliver in the aggregate that is capable of serving as the MC and we insert that node into the RSPEC. As a result, the component manager automatically creates and initializes an MC node as part of the slice. Because we need special software on the MC, we configure the RSPEC to use a custom OS Kernel on the MC node–one that has been loaded with the INSTOOLS software. An alternative would be to load the custom MC code after the default kernel boots up.

Because the MC is part of the slice, teardown of the measurement system is greatly simplified. When the user tears down a slice, the MC is automatically removed along with the other resources in the slice. If the user wants to retain measurement data from the experiment, it is the user's responsibility to copy the data off of the system before the slice is deleted. Our intent is that the user interface will offer methods to selectively copy data from the MC to the shared data archive repository.

Note that our current design does not allow the user to specify in the RSPEC which measurement data should be collected. Instead, it only allows the user to select whether the measurement

infrastructure is required or not. If not, the MC will not be automatically added. When the measurement infrastrucutre is created, users can use the GUI provided by the presentation component to specify the measurement data they want to collect.

4.2 Data Capture

Automatically setting up monitoring services on behalf of the user requires information about the topology and the types of resources (slivers) used in the experiment. Information about the types of slivers helps determine which (monitoring) software or services should be used to monitor the sliver. Topology information is needed to identify the links to be monitored. For example, in the ProtoGENI control framework, topology and sliver information can be obtained from the manifest returned by the aggregate manager.

To enable automatic data capture at slivers, we enhance the software installed on a sliver by adding a variety of off-the-shelf and custom monitoring software/services, and then modify the experiment creation code in the control framework to configure and launch the added monitoring software on each sliver. Because we expect that our monitoring services will be used by almost all of our users (i.e., students), we decided to make the monitoring software and services a standard part of every sliver. Users can specify that they do not want monitoring enabled, but the monitoring software is loaded by default.

The monitoring software that we add to each sliver includes an SNMP [10] daemon, various off-the-shelf system/network management programs (e.g., tcpdump [5], netflow [11], etc), our own custom monitoring code (based on the pcap library [3]) to collect certain packet statistics not captured by the SNMP daemon, and a remote access daemon to execute the capture software. For the most part, the software we add to each sliver is targeted toward IP-based network protocols. Because the assignments/projects that we assign in our network class are usually based on IP protocols, this software is sufficient. However, if GENI is to be used as a testbed for next generation Internet protocols, experiment-specific capture software may need to be installed during slice creation to capture packets that are not IP-based. Fortunately, the INSTOOLS infrastructure is independent of the capture software, allowing us to install new tools as they become available. Although we do not currently provide an interface or API that would allow users (or programs) to select the capture software to be installed on a measurement node, it is something that could be added.

In addition to capturing network information, we also capture operating system information such as CPU load, memory load, routing table configurations, ARP caches, loaded modules, etc. This data is captured via standard operating system tools like ps, vmstat, etc, and also via SNMP.

Having installed software to capture the desired measurement information, the next step is to add code that dynamically configures the capture software and services on a sliver before starting the sliver. To achieve this, we use information obtained from the manifest to identify the slivers to be monitored and then dynamically create configuration files for the SNMP daemon and other capture software on each monitored sliver–tailored to the network links used by the sliver.

It should be noted that ProtoGENI not only supports physical nodes, but also virtual nodes. Virtual network interfaces appear and behave differently than physical network interfaces. As a result, conventional SNMP implementations–that are unaware of virtualization–are not able to collect information related to virtual nodes and their interfaces. In such situations, we have had to develop our own code to capture the desired information–e.g., virtual interface stats–and make

the information available via a standard SNMP server.

4.3 Measurement Control

The capture software is installed and configured with a set of default settings that we believe are generally useful to a wide range of users. However, we also recognize that users will want to dynamically control the information being captured. To help users specify the data that should be captured, our INSTOOLS software includes a control GUI that allows users to select which information they would like to capture and display. The control settings entered by the user are used by the MC to enable, disable, or modify the configuration files of the associated capture software on the MPs. In our current design this is done by issuing ssh commands from the MC to the MPs. This allows the MC to change configurations and restart deamons, or to dynamically start and stop tasks (e.g., tcpdump).

4.4 Data Collection

The objective of the data collection component is to retrieve information from the MPs being monitored, transferring the data to the location(s) where it is needed or where it is to be stored/archived.

A variety of collection methods are possible. In keeping with our goal of leveraging existing, tested, robust software, our current collection component uses two methods to retrieve data from the MPs: (1) SNMP and (2) ssh/scp.

First, we decided to use the SNMP protocol to retrieve MIB tables from the various MPs that comprise the system. We selected the SNMP protocol as our primary retrieval method because of its generality. Even when used in its standard configuration, the SNMP protocol can be used to retrieve a variety of different types of information from routers, ranging from routing tables, to packet counts, to CPU load. Moreover, by adding new MIBs, SNMP can be used to retrieve almost any type of information a user would like to collect.

Second, we use secure copy (scp) or secure shell (ssh) to retrieve data that cannot be obtained via SNMP. Like the SNMP protocol, ssh/scp services are typically available on most routers and end hosts. Moreover, the ability to execute arbitrary commands on an MP or to copy arbitrary data from an MP to the MC gives us complete flexibility to retrieve any type of data from the MP.

Ideally, measurement data will be collected from the MPs over a dedicated measurement network as shown in Figure 1. Using a dedicated measurement plane ensures that the monitoring system does not interfere with an experiment's data plane traffic. However, it is difficult to guarantee that the measurement plane will be completely independent of the data plane. For ProtoGENI experiments that utilize Emulab resources, separation is possible. For example, our current implementation utilizes Emulab's control network as the measurement plane, separating measurement data from the experiment's data plane (which is constructed using VLANs on the experimental switches). Because each slice has one MC per aggregate, measurement data is always collected locally within the aggregate. We chose this design because it limits the scope of the measurement network. Creating a dedicated global measurement network with sufficient bandwidth to support an instrumentation system–particularly one that includes high-speed packet capture devices–would be difficult. By limiting the scope of the measurement network to the local aggregate, we reduce the potential for competition between the data plane and the measurement

plane. Moreover, additional MCs can be created within an aggregate to further improve the separation between the measurement and data planes.

4.5 Data Storage

The data storage task is broken into three parts: (1) captured data is initially stored (temporarily) at the MPs before being collected and moved to the MC, (2) the primary storage location is the MC for each aggregate, and (3) important data is archived for long-term storage at the archive repository (see Figure 2).

Each MP captures data and temporarily stores it until it is transferred to the the MC. Consequently, we assume that each MP has sufficient storage space to hold the data until it can be sent to the MC. In some cases, the data may be processed or filtered before being sent to the MC. Data stored on the MPs is not assumed to be on stable storage.

The primary data storage location is the MC in the aggregate where the data was captured. Consequently, each MC must have enough storage space to hold all the data collected from the slivers in the aggregate for the slice. In some cases, the data may be stored in its original (raw) format (e.g., tcpdump [5] format). However, to make it readily viewable using a variety of existing graphing programs, the preferred storage format is the round robin database (RRD) [13] format. The RRD format also allows us to bound the space consumed by the captured data. Our current implementation leverages the MRTG [7] software which interfaces with software like snmpwalk to retrieve and store the data in RRD format so that it can be easily converted to charts and graphs for viewing. In addition to basic file storage, the MC must provide a conventional SQL database service that can be used to store data and references to data files that can be searched to select only the data of interest to the user.

Finally, a shared global storage repository is used to store data for long-term use–use that could extend beyond the lifetime of the experiment. The archival repository is designed with the assumption that users will infrequently access data from the archival repository. Users that want to visualize the data from their experiment will typically contact the MC to see their data, not the archival repository. Data from the archival service will primarily be downloaded to the user's machine for postmortem processing. Consequently, the interface to the archival service is expected to be a simple file transfer service. Only data that will be needed beyond the lifetime of the experiment, or data that cannot be lost, or data that is to be "published" for shared use, is expected to be stored with the archival service. By default, no data is sent to the archival service. Users must explicitly specify data to be sent to the archival service. We have not yet defined the method by which data is saved to the archival service, nor has the interface to retrieve data from the repository been defined. These are both areas of future research.

4.6 Data Processing

Data processing is occasionally needed in order to transform or summarize the captured data. Examples include filtering the data to select items of interests (e.g., packets of a certain type or packets belonging to a certain flow), computing averages, maximum, minimum, and total values, removing or eliding fields from the data, analyzing netflow data, etc. In the case of high-performance packet capture devices, processing and/or filtering will likely occur on the device itself (i.e., on the MP). Some processing may occur on other MPs as well. However, we assume

that most of the processing will occur on the MC because the MC holds (stores) all the data and is the best suited to perform processing across data from multiple MPs.

4.7 Data Presentation

Data presentation occurs in two steps. First the data must be converted into graphs and charts that will help the user visualize the data. The second step involves creating web pages to display the graphs and charts in an intuitive way.

4.7.1 Creating Graphs and Tables

A variety of data visualization techniques are possible, and the INSTOOLS architecture is designed to be agnostic about the way in which data is displayed. Instead, it provides mechanisms by which users can write scripts to visualize their data. By default, the INSTOOLS software comes with a set of scripts based on the rrdtools [13] to create graphs from the RRD data files. It also includes custom software to create tables from the collected information (e.g., routing tables, ARP tables, etc.). However, users are free to write their own scripts to process the data into a form that can be displayed as part of a web page.

4.7.2 The Web Interface

A key objective of our web interface is to provide users with one-stop access to all the measurement data associated with an experiment/slice. We want an interface that is easy to understood and simple to use, not cluttered with a myriad of buttons selecting options that are incomprehensible to everyone except experts. Common information such as packet counts should be readily available for viewing, while more detailed information such as packet traces should be requested on demand. To make the data easily accessible from any machine, we decided to make the data available via web pages that can be accessed from any browser (without the need for special pluggins or other browser features). Because users think about and view the topology as a set of slivers connected by links, the web interface should display the data to users in a layout that matches their conceptual view – i.e., a layout organized around slivers and links, clearly associating the data being displayed with the sliver or link it was derived from.

The job of the web interface system is to help users see the information of interest to them. In particular, it should dynamically create web pages that display the graphs/charts selected by the user for the slivers and links selected by the user. For example, selecting an MP from the list of links and MPs should take the user to a web page that contains network state information and traffic performance graphs for all the interfaces attached to the selected MP. Similarly, selecting a link should bring the user to a page with traffic information specific to the link. Moreover, users should be able to select which information they want to display for a sliver or link.

To reduce the load on the measurement system, the web interface system only generates a web page's contents on demand, both when it is first displayed and periodically thereafter for "live" traffic graphs. In that way graphs are not generated until they are requested by the user.

4.8 Access Protection

Up to this point our discussion has been focused on the functionality needed to create and display measurement information to the user. However, the system must also include security mechanisms to protect the data, making it available only to authorized users. Moreover, because the INSTOOLS system is performing many operations on behalf of the user–including software installation, data capture, data collection, and data processing–INSTOOLS requires the access rights needed to perform these task in the user's slice.

Because our user interface is designed around web pages, conventional web authorization techniques and services can be used to protect access to the data. We have not yet selected a web authorization mechanism but hope to leverage the web page authorization mechanisms used in the Emulab web interface (or its ProtoGENI successor). Sharing data with other users will require additional (group) authorization mechanisms that have not yet been defined.

The more pressing authorization issue is the problem of installing and controlling the the INSTOOLS software on the various slivers that make up the instrumentation and measurement infrastructure. In particular, the INSTOOLS software needs to run under a user account on the MC with the ability to access data on the MPs. Our current implementation relies on an old Emulab (ssh) key distribution mechanism to login or copy data from other machines, but this is not an accepted part of the GENI API and it is not clear how to do this if the Emulab features are no longer accessible in a GENI environment. This is a point of continued debate.

5 Implementation Status

We implemented and deployed an initial version of the INSTOOLS system in the context of the ProtoGENI [4] control framework. We modified the ProtoGENI control framework software to automatically add the MC to the RSPEC so that it will then be created by the ProtoGENI component manager. In order to load the needed software on the MC and the MPs we created a new MC kernel image and changed the ProtoGENI code to load our MC kernel onto the MC node and our MP kernel onto the MPs.

Each MC includes an apache web server through which users are able to view the measurement data for their slice. In addition to a web server, the MC contains MRTG software to retrieve data from the MPs. The data is stored in RRD files on the MCs local hard disk using a naming convention and metadata files that record information about the time, place, and type of information collected.

Each MP is loaded with an SNMP daemon and custom MIBs to collect information about the routing tables, packet counts, and some OS configuration information. The MC retrieves information from the SNMP daemons on the MPs using snmpwalk (called from MRTG). Our current implementation uses a shared community string to protect access to the SNMP daemon which is less than ideal. An ssh daemon provides the MC with the ability to execute arbitrary Unix commands to see OS configuration information, and to copy data files off of the MPs. We are in the early stages of adding support for Netflow data, and expect to copy netflow data from the MPs to the MC using secure copy.

5.1 Instrumentation List

Table 1 lists the instrumentation information currently available via our web interface.

Software	Information	Display Type
SNMPd	Routing Table	Table
	IP Traffic	Graph
	ICMP Traffic	Graph
	TCP Traffic	Graph
	UDP Traffic	Graph
	CPU Utilization	Graph
	Memory Utilization	Graph
	Total Network Traffic	Graph
	Link-specific Traffic	Graph
	Link-specific Unicast Traffic	Graph
ssh/arp	ARP cache	Table
ssh/netstat	TCP streams	Table
ssh/netstat	UDP listeners	Table
ssh/ps	Process list	Table
ssh/lsmod	Installed Kernel Modules	Table

Table 1: Measurement information collected and displayed

5.2 Web Interface

The current web interface is designed to be functional in order to demonstrate the ability to capture data and display it via a single (web) interface. When a user's experiment is created, the slice creation scripts output the IP address/URL of the measurement system web pages. After the slice has been configured and started, the measurement pages become available for viewing. A user simply needs to type in the specified URL to view their measurement data.

The measurement web page includes a pull-down menu of the resources (slivers) that are being monitored. Users select the sliver or link they want to see data for and then specify which information they want to see for that sliver/link (see Figure 4).

Our current web pages only display information for one sliver or one link, but our next generation interface is expected to be based on a content management system where the user will have much more control over the look and feel of the display. Users currently view multiple slivers and links via multiple tabs or multiple instances of a web browser.

After selecting the information to be viewed, the web software (PHP scripts) generates the selected tables and graphs. The graphs are currently updated and redrawn every five seconds to give the appearance of a "live" view of the network. Example table output is shown in Figure 5, and example graph output is shown in Figure 6.

Figure 4: Initial prototype interface to select measurement nodes and information for display.

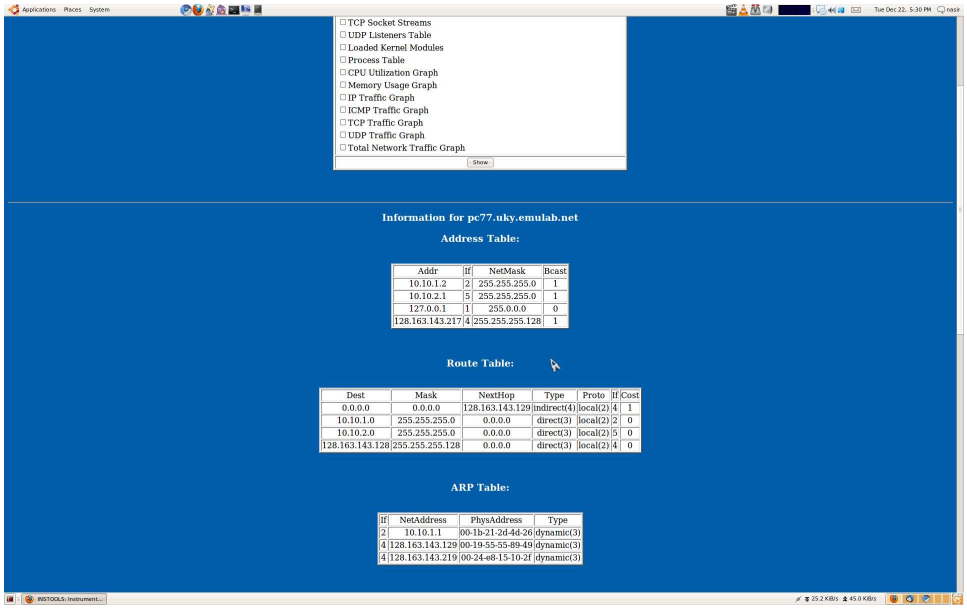

Figure 5: Example table output for a sliver.

Figure 6: Example traffic graphs.

5.3 Security Considerations

Our current implementation utilizes the Emulab ssh-key distribution mechanisms to ensure that only the MC sliver has access to the MPs. Security and authentication are an area of further research and will ultimately depend to a large extent on the security mechanisms that end up being supported by the underlying control framework. We are currently exploring a variety of possible methods to secure access to the web interface.

References

[1] GENI glossary. http://groups.geni.net/geni/wiki/GeniGlossary.

[2] GENI Meta Operations Center. http://gmoc.grnoc.iu.edu/gmoc/index.html.

[3] Libpcap. http://www.tcpdump.org/.

[4] ProtoGENI. http://www.protogeni.net.

[5] TCPdump. http://www.tcpdump.org/.

[6] The Edulab Project. http://www.netlab.uky.edu/edulab.net.

[7] The Multi Router Traffic Grapher. http://oss.oetiker.ch/mrtg/.

[8] GENI SpiralOne, 2008. http://groups.geni.net/geni/wiki/SpiralOne.

[9] P. Barford, M. Blodgett, M. Crovella, and J. Sommers. Requirements and Specifications for the Instrumentation and Measurement Systems for GENI: Version 0.2, May 2009. http://groups.geni.net/geni/attachment/wiki/MeasurementSystem/MeasurementSysSpec.pdf.

[10] J. Case, R. Mundy, D. Partain, and B. Stewart. Introduction and Applicability Statements for Internet Standard Management Framework. RFC 3410, Dec. 2002.

[11] Cisco. Introduction to Cisco IOS netflow - a technical overview, 2007. http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6555/ps6601/prod_white_paper0900aecd80406232.html.

[12] Paul Barford (ed). GENI Instrumentation and Measurement Systems (GIMS) Specification, 2006. GENI Design Document 06-12, Facility Architecture Working Group, http://groups.geni.net/geni/attachment/wiki/OldGPGDesignDocuments/GDD-06-12.pdf.

[13] Tobias Oetiker. RRDtool. http://oss.oetiker.ch/rrdtool/doc/rrdtool.en.html.

[14] GENI Project Office. GENI: System Requirements Document, GENI-SE-SY-RQ-02.0.pdf, July 2009. http://groups.geni.net/geni/attachment/wiki/SysReqDoc/GENI-SE-SY-RQ-02.0.pdf.

[15] Robert Ricci and Ted Faber. GeniRspec. http://groups.geni.net/geni/attachment/wiki/GeniRspec/rspec-draft-v0.5.doc.

[16] James N. Griffioen W. David Laverell, Zongming Fei. Isn't It Time You Had An Emulab? In *Proceedings of the ACM SIGSCE 2008 Conference*, March 2008.