Designing A GENI Experimenter Tool To Support The ChoiceNet Internet Architecture

D. Brown, O. Ascigil, H. Nasir, C. Carpenter, J. Griffioen, and K. Calvert
Laboratory for Advanced Networking
University of Kentucky
Lexington, KY 40506
Email: [davidb,onur,nasir,charles,griff,calvert]@netlab.uky.edu

***Abstract*—Testbeds such as GENI provide an ideal environment for experimenting with future internet architectures such as *ChoiceNet*. Unlike the narrow waist of the current Internet (IP), ChoiceNet encourages alternatives and competition at the network layer via an economic plane that allows users to choose and purchase precisely the services they need.**

In this paper we describe our experiences implementing the ChoiceNet architecture on GENI. Some features of GENI, such as the ability to program the network layer, to leverage existing protocols and software, to run real applications generating realistic traffic, and the ability to perform long-running experiments made GENI an ideal platform for ChoiceNet experimentation. However, we found that GENI currently lacks the tools needed to make it easy to use these features.

To address this issue, we designed and implemented a *GENI Experimenter Tool* specifically designed and tailored to perform tasks commonly needed by experimenters such as dynamically configuring nodes, loading and compiling node-specific code, executing Click modules, running commands on sets of nodes, accessing the local file system on nodes, and dynamically logging into nodes.

I. Introduction

One of the original goals of the GENI network [1] was to create an at-scale playground for experimentation with next generation internet architectures [2], [3]. In conjuction with GENI, the National Science Foundation (NSF) supported several research efforts as part of the FIND [4], FIA [5], and now FIA-NP [6] programs focused on designing, implementing, and testing next generation internet architectures. Several of these project have used, or now are using, GENI as a platform for their experiments. For example, the NSF FIA projects XIA [7] and NDN [8] have both implemented a prototype of their network architecture on GENI [9], [10], which speaks to GENI's the ability to support a variety of different network architectures.

In contrast to XIA and NDN – which are focused on developing and demonstrating new network layer mechanisms such as addressing and forwarding – the ChoiceNet project [11], [12] (also an FIA project) is agnostic toward the design of these network layer mechanisms. Instead, ChoiceNet is focused on the *economics* of the network layer. Its goal is to support mechanisms that enable users to compensate (reward) Internet Service Providers (ISPs) for developing, deploying, and operating innovative new network layer services. In fact, ChoiceNet encourages competition between providers, allowing alternative network layer services to co-exist. For example, ChoiceNet allows multiple addressing and forwarding mechanisms to co-exist. One service might offer conventional IP-style packet lookup and forwarding, while another might offer NDN-style packet forwarding. As a result, users (or their applications) are able to select and pay for the network layer forwarding mechanism that best meets their needs.

To support these types of interactions, ChoiceNet introduces the concept of an *economy plane* and marketplace where customers and providers can rendezvous to buy and sell network layer services. Network layer services (e.g., forwarding) operate in the *use plane* and will not perform the specified service unless the request for service (e.g., the packet header) includes a non-forgeable, cryptographically-signed token obtained from the economy plane demonstrating "proof-of-purchase".

To evaluate the ChoiceNet architecture, we developed a prototype implementation that includes both *economy plane* and *use plane* services. Instead of designing and implementing a single forwarding mechanism, the ChoiceNet prototype provides the infrastructure for experimenters (acting as ISPs) to dynamically create and sell any type of new forwarding mechanisms they can imagine. To dynamically create new network layer services, ChoiceNet leverages the programmable routers available in GENI. Experimenters create code that can be loaded onto ChoiceNet routers to perform a new network layer task, and then advertise/sell the service in ChoiceNet's economy plane.

In this paper we describe our experiences implementing the ChoiceNet architecture on GENI. Some features of GENI, such as the ability to program the network layer, to leverage existing protocols and software, to run real applications generating realistic traffic, and the ability to perform long-running experiments made GENI an ideal platform for ChoiceNet experimentation. However, we found that GENI currently lacks the tools needed to make it easy to use these features. To address this issue, we designed and implmented a *GENI Experimenter Tool* specifically designed and tailord to perform tasks commonly needed by experimenters. Example tasks include dynamically configuring nodes, loading and compiling node-specific code, executing click modules, running commands on sets of nodes, accessing the local file system on nodes, and dynamically logging into node.

This work is supported in part by the National Science Foundation under grants CNS-1111040 and CNS-134668 Subcontract 1928.

978-1-4799-6204-4/14$31.00 ©2014 IEEE

In the next section we provide a brief overview of the salient features of the ChoiceNet architecture. Section III then describes our implementation of ChoiceNet on GENI, followed by Section IV which highlights some of the challenges we encountered while developing our ChoiceNet prototype. Section V describes a new tool called the *GENI Experimenter Tool*, that we designed and implemented to address some of these issues and compares it with existing tools. Section VI provides some initial performance results using the GENI Experimenter Tool. Finally, Section VII offers some concluding thoughts.

II. CHOICENET

The ChoiceNet architecture views all functionality offered by the network layer as marketable *services*. Services are bought and sold in the ChoiceNet *economy plane* which defines the protocols that customers and providers use to discover services and exchange payment for services. Service providers advertise their services in ChoiceNet marketplaces where customers can come to discover the list of available services.

Services are used in the *use plane*. In order to invoke a service, the entity requesting the service must provide some proof that the service has been "paid for" in the economic plane. There are a variety of standard ways in which a customer can "pay for" a service, ranging from some monetary value such as a credit card, to proving knowledge of some (secret) credential, to presenting unforgeable proof of membership in a particular group (e.g., faculty, staff or students at a particular university), to a null payment (i.e., no payment required). Having "paid for" a service, the customer (user's application) receives a time-limited, cryptographically-generated token that can be presented to the service as "proof of purchase". Services verify these tokens before performing the specified service. As a result, use plane services can enforce the business relationships that have been established in the economy plane, ensuring that service is only rendered to paying customers.

Consider for example, the network layer task of forwarding a packet at a router. In ChoiceNet, the packet forwarding functionality provided by a router could be a service. In general a router would only perform its *packet forwarding service* on packets carrying tokens showing that the service had been purchased in the economy plane. Services can also be composed to form more complex services allowing, for example, a broker to purchase and combine the packet forwarding services of several routers to form a *path forwarding service* that carries packets end-to-end. The broker could then advertise and sell this composite service in the ChoiceNet marketplace.

Because ChoiceNet is intended to promote competition among providers and choice for customers, it is capable of supporting competing network layer services. For example, one provider may offer a conventional hop-by-hop packet routing/forwarding service on its routers (or routers it shares virtually with other providers). Another provider may decide to offer a source-routed packet forwarding service. A third provider may offer an NDN-style forwarding service based on content names, while a fourth offers and XIA-style forwarding service based on service identifiers. Ancillary services will in general also be offered. For example, the provider of source-routed forwarding services might also offer a path discovery service that senders can contact to obtain paths across that network—paths that traverse the provider's source-routed forwarding services.

To create an environment in which business relationships can develop and fluorish, ChoiceNet defines the protocols used to exchange value and it also provides the infrastructure needed to market, discover, compose, and use services. The next section describes a prototype implementation of the ChoiceNet infrastructure running on GENI.

III. THE CHOICENET PROTOTYPE

To evaluate the ChoiceNet architecture, we developed a ChoiceNet prototype running on the GENI testbed. The prototype runs in a long-lived GENI slice where experimenters–acting as providers–can create and market new network layer services, or–acting as customers–can develop and test applications that choose and purchase the services they need.

Although code to integrate PayPal into ChoiceNet has been implemented [13] and could have been used to obtain actual payments from customers, we did not use this feature in our testing since our goal was to evaluate the overall ChoiceNet architecture rather than to actually charge for services. Although we used "fake payments", applications did get back cryptographically signed "proof of purchase" tokens that services must verify before performing the service. In that sense, the prototype exhibited all the characteristics of a real ChoiceNet network.

Because the current Internet does not allow applications to select the paths their packets traverse, our initial goal was to develop a service (or set of services) that applications could invoke to identify and select the network path(s) that best meet their needs. We began by developing a *packet forwarding service* and deploying it at each router in the ChoiceNet network. Our packet forwarding service relays packets from a particular ingress port on the router to a particular egress port on the router, but only if the packet carries the necessary token (i.e., "proof of purchase").

To intercept packets passing through routers, we installed Click [14] on each of the routers in the ChoiceNet slice[1]. We then implemented our packet forwarding service as a Click forwarding element that decides whether to forward a packet or not by performing a cryptographic check of the token carried in the packet. If the verification succeeds, our forwarding element sends the packet to the next router specified in the packet.

The forwarding service at each router periodically sends service advertisements to a marketplace service we implemented called the *path service*. The path service acts as a marketplace in the sense that it receives advertisements from all the packet forwarding services, including information about their current performance characteristics (e.g., available bandwidth and current delay). Although marketplaces often only provide a listing of available services, our path service provides the value-added service of path computation. Knowing the topology and the performance characteristics of every packet forwarding service, the path service accepts requests from applications specifying the type of path desired (i.e., the

[1]For performance reasons we used the kernel module of Click.

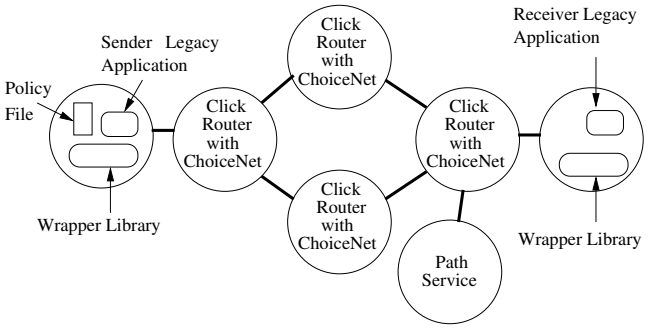

Fig. 1: Example GENI Topology from a ChoiceNet Experiment

source and destination and the path characteristics such as high bandwidth or low delay), and then identifies the best path for the application.

Applications interact with the path service by first purchasing the right to request paths from the path service. Once an application has purchased the ability to request paths from the path service, all paths returned by the path service are (seemingly) "free". That is, the path service computes the best path for the application, and then purchases (or pre-purchases) use of the forwarding services along the path on the application's behalf. As a result, the application gets back the set of tokens needed to use at forwarding services along the path. Because ChoiceNet supports the ability to *delegate tokens* to other services, the path service is able to "resell" the forwarding service tokens to applications. That is, the path service not only purchases the right to use a forwarding service, but also the right to delegate the tokens it receives to other parties. As a result, the forwarding services provide tokens to the path service that can be delegated to applications. In short, the paths returned to applications come with the tokens needed to use each router along the path so that applications do not need to purchase individual forwarding services themselves.

Having obtained a path, applications include the path (the list of packet forwarding services and the associated tokens required to use them) in the ChoiceNet packet header. The ChoiceNet packet header consists of a series of ChoiceNet service requests much like a source route specification in an IP packet. Each router examines its portion of the ChoiceNet packet header to verify that forwarding has been paid for before forwarding the packet.

In an attempt to leverage as much existing software as possible (i.e., existing IP-based programs and libraries), we implemented our ChoiceNet packet header in an IPv6 extension header, using IPv6 addressing as our addressing model. As a result, each packet forwarding service is defined by the ingress and egress IPv6 address it relays packets between. The Click-based packet forwarding element at each routers consults the IPv6 extension header to find the ingress/egress addresses and the proof-of-purchase token that must be verified.

To use legacy applications in ChoiceNet, we developed a "wrapper library" that we dynamically load when running a legacy application. The wrapper library intercepts standard network systems calls such as socket, bind, connect, send, and recv, and adds the ChoiceNet extension header to the IPv6 packet header. As a result the application is completely unaware of ChoiceNet. To enable choice, the wrapper library consults a local ChoiceNet policy file on the sender to know what type of paths to request for the application. The policy file might, for example, include an entry indicating that NetFlix applications should request high bandwidth paths, while another entry indicates that interactive first person shooter games should request low latency paths. Using the policy file the wrapper obtains the desired path from the path service. It then includes that path (along with the tokens) in the packets it sends out. Each Click router along the path verifies the token before forwarding the packet along the specified route. Figure 1 illustrates the various components of the ChoiceNet architecture running on an example GENI slice.

Using the wrapper library, we have run a variety of legacy applications such as remote login (ssh), file transfer programs, web clients and servers, and interactive online video games, each purchasing and using a path tailored to its needs.

IV. Challenges of using GENI

GENI is arguably the ideal platform for developing and evaluating our ChoiceNet prototype. It supports programmable routers needed by experimenters developing new ChoiceNet services. It supports experiments at-scale, allowing the creation of wide area network topologies with realistic bandwidths and delays. At the same time it allows for emulated topologies and link characteristics which are useful for targeted experiments. The ability to create long-running slices allows us to create and deploy a ChoiceNet network, and, over time, continue to enhance the network with new ChoiceNet services. Long-running experiments combined with the ability of real-world user to connect to and use services on the ChoiceNet network makes it great platform for attracting users (customers and providers), learning how they use the network, and experiencing realistic traffic loads. Moreover, the ability to quickly create new slices and tear them down, make it easy to test ChoiceNet with a wide range of topologies.

Although tools exist to create slices [15], [16], [17] and to instrumentize and measure slice behavior [18], [19], [20], we found that the tools needed to setup, configure, and control a running experiment were quite limited. Tools such as GUSH [21] and associated tools stork [22] and raven [23] provide programmatic ways to setup and configure nodes in a slice, but they have no inherent knowledge of the topology or how to configure it, placing the entire burden of deciding what to install where on the experimenter. Consequently, the configuration files for these tools are often complex and are not well documented or supported. Moreover, they provide very limited control over a running experiment/slice.

Post-boot scripts [24] provide another alternative to setup a slice, but require that the user write the scripts from scratch to deploy code and configure nodes. Moreover they offer no runtime control over running experiments.

Systems such as the GENI Desktop [20] are focused on providing instrumentation and measurement support, with only minimal support for setup and configuration or runtime control of an experiment.

Tools like JFED [25] incorporate a graphical time line to which you can attach bash commands to be run on specific nodes at specific times. These commands can be saved to the rspec, but are only run when using the JFED GUI.

Based on our experience implementing ChoiceNet in GENI, it became clear that there are several tasks that are likely to be common to almost all GENI experiments and could benefit from a new tool (or tools) to automate these tasks. For example, after a slice has been created, an experimenter will typically need to load software onto the nodes in the slice, and the software to be loaded will depend on the node's role in the topology (e.g., router software will differ from end system software). Because the software being loaded is often experimental software that is actively being developed and is frequently changing, the code may need to be re-downloaded and compiled on the node using node-specific information such as the node's header files and list of network interfaces. Moreover, this cycle of editing the code, reloading code onto nodes, (re)compiling the code at each node, and running or restarting the code on each node can repeat itself over and over again.

To control nodes in a running experiment, users often need to run commands on certain subsets of the nodes. Systems like the GENI Desktop provide some limited support for this, but do so through a graphical user interface. A similar interface that can be used in a scripting context would certainly benefit experimenters. Moreover, specifying nodes to be controlled via their (IP) addresses is cumbersome and error-prone, especially as the addresses of nodes in new slices are unpredictable, even if generated from an identical RSPEC. As such GENI slices would benefit from a simpler way to identify nodes to be controlled at runtime.

In the case of our ChoiceNet prototype, we needed the ability to add new router services (e.g., packet forwarding) which required significant effort to install and configure Click, followed by the challenges of enhancing Click with new forwarding elements. Because many experiments will require the ability to modify router behaviour, creating tools to assist with the installation and setup of Click routers and modules would greatly simplify the task of programming GENI routers.

V. The GENI Experimenter Tool

To assist with common tasks such as those described in the previous section, we designed and implemented a new *GENI Experimenter Tool (GET)* that makes it easy for users to dynamically perform—or to write scripts that perform—common tasks. We designed the tool to assist us in deploying and running ChoiceNet experiments on GENI, and thus some of the functionality is tailored to the needs of ChoiceNet experiments. While some of the tool's logic is devoted to ChoiceNet-specific configuration tasks, we believe most of the day-to-day functionality of GET is common to any experiment hosted on GENI, and thus would be of use to any researcher working within the GENI framework. The tool is designed to run in Unix environments, but can also be run on MS Windows using Unix-emulation packages like cygwin[2].

The tool is aware of slices and begins by reading in the RSPEC (or manifest) associate with a slice. The location of the RSPEC is specified either through a command line argument to the program or through a configuration file. The tool then parses and extracts the topology from the RSPEC and then allows the user to reference nodes within the experiment using user-friendly node names (e.g., VM-0) specified in the RSPEC as opposed to the typically more arcane DNS or physical IP addresses (and possibly port number) of the individual nodes (e.g., pc6.lan.net.somewhere.edu port 34912).

Once the topology has been read and parsed, the tool is able to assist with a variety of tasks that experimenters commonly do including:

- **Listing information about nodes** — Lists information about one or more nodes within the slice including their network connectivity.
- **Remote file access** — Automatically creates an ssh-based file share to files on a node in the slice. While this has many uses, it is particularly useful for accessing log files on nodes in the topology that record information about a running experiment.
- **Remote command execution** — Allows the execution of a specific shell command on all (or any subset) of the nodes in a slice, with options for file or console output and multithreaded execution.
- **Remote login** — Establishes an ssh session with the node specified by the (hopefully friendlier) node name contained in the RSPEC[3].

In addition, the tool supports some ChoiceNet-specific features that could be re-programmed to the user's needs if desired:

- **IPv6 configuration** — Automates IPv6 address assignment and routing. By default, nodes are assigned link-local IPv6 addresses which are often not sufficient for the needs of the experimenter. Moreover, unlike IPv4 routing tables that are automatically configured so routing works across the slice, IPv6 routing tables are not setup by default. The GET tool assists by assigning addresses and setting up IPv6 routing on behalf of the user.
- **ChoiceNet Code installation** — Automatically installs the ChoiceNet code suite on all nodes in a slice. However, this feature is somewhat generalizable to software beyond ChoiceNet due to the fact that it pulls code from a configurable git repository and then builds the software based on configurable compilation commands. Moreover, the code that is installed and compiled will differ based on whether the node is acting as a router or an end system.
- **ChoiceNet Router activation** — Automatically starts/stops ChoiceNet functionality on routers within the slice. Like the code installation, this feature can be customized through configuration files.

Because GET is aware of the topology, it knows which nodes are acting as routers and which nodes are acting as end

[2]Remote file access may not be fully supported in non-Unix environments.

[3]GET's initial development was largely spurred by the frustration of having to constantly look up names and ports to ssh to – a very common task when working with virtual machines (VMs).

systems (hosts). Consequently, GET can automatically make decisions about what code needs to be installed on each node, how to configure each node, and which services need to be started on each node. This frees GET users from having to keep track of which nodes are which and then run the appropriate commands on them.

All of GET's features (described above) share the same method of identifying nodes to which the operation is to be applied. There are basically three ways to specify nodes. Users can request that operations be applied to "router" nodes (those having multiple network interfaces), hosts (end systems), or arbitrarily included or excluded nodes (by name with wildcard names also supported). For example, a user is able to easily run a command on all routers simply by selecting "routers" as the target node type. Similarly, a user can easily run a command on all "host" nodes. As another example, a user could run a command on all nodes whose RSPEC names begin with $VM-$.

To help deal with output from commands run on the specified nodes, GET provides the ability to collect all the output into a single file, or to split the output into separate files based on the node name and user-specified expression. For example, a user might give the name *%.run1-output.txt* as the output file name where the % would be replaced with the name of the node where the output file was generated (e.g., VM-0.run1-output.txt).

In some cases, it is helpful to know what nodes GET would apply an operation to. Consequently, GET has an option to print out the operations or commands it would have executed without actually taking any action. This can be helpful in determining which nodes would be affected by an operation, or to show how to perform an operation by hand.

For functions that target multiple nodes within a GENI slice, GET supports multithreaded execution, interacting with multiple nodes in the slice simultaneous. This is the default behavior as it generally reduces the amount of time required to perform the specified task. However, in cases where multiple VMs are hosted on the same physical nodes, the concurrent execution can actually increase the time required. In such cases, GET supports a single threading option that can improve performance.

A list of example GET commands is shown in Figure 2. Our experience using GET has significantly improved the productivity of our team. Beyond making it trivial to ssh to nodes in a slice using user-friendly names or running commands remotely, GET has brought down the time requirement for configuring a newly create slice from hours of repetitive (and unfortunately error-prone) routing configuration and software building to a pair of shell commands and tens of minutes minutes (or less) of waiting until ChoiceNet is fully operational in the slice.

While originally designed as a ChoiceNet-specific tool, GET has been developed as generally as possible. Many of its features can be immediately used by any other experimenter (e.g., remote login, remote file access, running commands on certain nodes, etc.). Other ChoiceNet-specific features would require some modifications. However,the ChoiceNet-specific functions of the tool are developed as a "module" on top of the general-purpose GENI interaction functionality, implying that it could be easily customized for use in other experiments as well.

`get list` List all nodes
`get list -R` List all nodes identified as routers
`get ssh VM-0` Initiate ssh connection with node named "VM-0"
`get list -R -x rt*` List all nodes identified as routers, excluding those with names that match "rt*"
`get cmd "ls /etc" -h host*` Execute "ls /etc" on all nodes with names that match "host*"
`get configure -h VM-0,VM-1` Run ChoiceNet configuration (including IPv6 neighbor and routing tables) on nodes VM-0 and VM-1
`get install -H` Install ChoiceNet suite on all non-router nodes
`get install -s -R` Install ChoiceNet suite on all routers and force single-threaded execution
`get cmd "hostname" -t` Execute "hostname" on all nodes and force multithreaded execution.
`get cmd "uname -a" -o '%.txt'` Execute "uname -a" on all nodes and save output for each command to a unique file (e.g, a node named VM-1 will generate VM-1.txt)
`get sshfs VM-1 ~/vm1` Use sshfs to map a node named "VM-1"'s file system to local directory ~/vm1

Fig. 2: Example GET commands

VI. Experimental Results

To evaluate the GET tool, we created several GENI slices (topologies) of varying size, and then used GET to install the ChoiceNet system on the slices. We also used GET to interact with and control the slices once they were set up.

Figure 3 shows example output from GET for a 7 node topology. In particular it shows output from the list operation, remote command execution, and remote config file access. In this example, the list command is first used to obtain the names of the nodes identified as hosts (using -H option). Next, the *route* command is remotely executed on a particular host – namely VM-5. The example also demonstrates running a *netstat* command on all nodes identified as routers (using the -R option), which prints the routing tables from each each router following the node's name. Finally, GET is used to access a configuration file ("config") on the router node VM-0 by mounting VM-0's file system to the user's machine (via the sshfs option of GET).

Setting up a ChoiceNet slice involves several steps that are greatly simplified by GET, including installing a wrapper

```
File  Edit  Tabs  Help
$ get -H list
VM - pc12.utahddc.geniracks.net:31034
  VM:if0 (10.10.1.1/02:1d:25:c4:67:22) -> VM-0:if0 (10.10.1.2/02:91:e1:97:8f:e7)

VM-5 - pc12.utahddc.geniracks.net:31040
  VM-5:if0 (10.10.8.2/02:38:aa:81:e6:67) -> VM-4:if3 (10.10.8.1/02:fb:87:ce:3d:ce)

$ get -h VM-5 cmd "route"
VM-5:
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         172.16.0.1      0.0.0.0         UG    0      0        0 eth0
10.0.0.0        VM-4-lan7       255.0.0.0       UG    0      0        0 eth1
10.10.8.0       *               255.255.255.0   U     0      0        0 eth1
172.16.0.0      *               255.240.0.0     U     0      0        0 eth0
$ get -R cmd "netstat -rn | head -n 4"
VM-0:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG        0 0          0 eth0
10.0.0.0        10.10.2.2       255.0.0.0       UG        0 0          0 eth2
VM-1:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG        0 0          0 eth0
10.0.0.0        10.10.2.1       255.0.0.0       UG        0 0          0 eth1
VM-2:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG        0 0          0 eth0
10.0.0.0        10.10.4.1       255.0.0.0       UG        0 0          0 eth1
VM-3:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG        0 0          0 eth0
10.0.0.0        10.10.7.2       255.0.0.0       UG        0 0          0 eth2
VM-4:
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         172.16.0.1      0.0.0.0         UG        0 0          0 eth0
10.0.0.0        10.10.3.1       255.0.0.0       UG        0 0          0 eth1
$ get sshfs VM-0 ./router0
$ cat ./router0/click/config
FromDevice@1 -> ChoiceHost@2;
FromDevice@3 -> ChoiceHost@4;
FromDevice@5 -> ChoiceHost@6;
FromDevice@7 -> ChoiceHost@8;
$
```

Fig. 3: Example Output from GET

library and ChoiceNet applications, adding IPv6 addresses to the interfaces, inserting routing table entries to host nodes and installing the Click router that is programmed to perform ChoiceNet forwarding services to the router nodes in the GENI experiment topology.

Issuing commands from GET to a single node, as one would expect, takes relatively little time. The main advantage in this case is the simplified user interface. For example, configuring a single host (i.e. adding IPv6 addresses and configuring routing tables) using GET takes around 2 seconds[4], while installing and compiling the necessary applications and wrapper library on a single host node takes as much as 11 seconds. Installing and compiling the Click router on a single router node takes 8 minutes and 40 seconds which is dominated by the file copy and compilation times.

An important feature of the GET tool is the ability to use either multi-threaded execution or single-threaded execution to carry out tasks on multiple nodes. In the multi-threaded case, installation and compilation are carried out in parallel, while the single-threaded option causes installation and compilation to occur on one node at at time.

Figures 4 and 5 shows the time needed to set up ChoiceNet on topologies of sizes ranging from 3 nodes to 15 nodes. Each topology consisted of two end system (host) machines with the remaining machines acting as routers. Figure 4

[4]All performance numbers were obtained using Xen nodes running on InstaGENI racks.

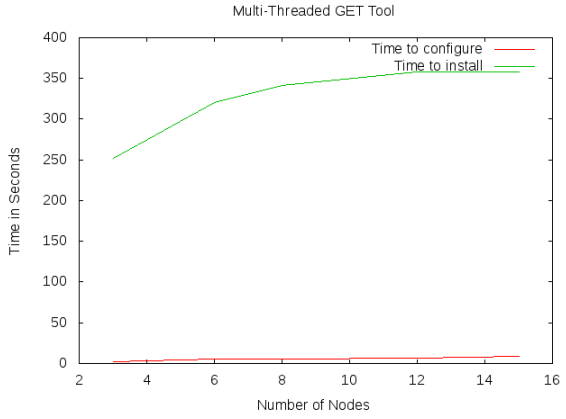

Fig. 4: Multi-threaded time neeed to install/compile and configure ChoiceNet over various size GENI topologies using GET

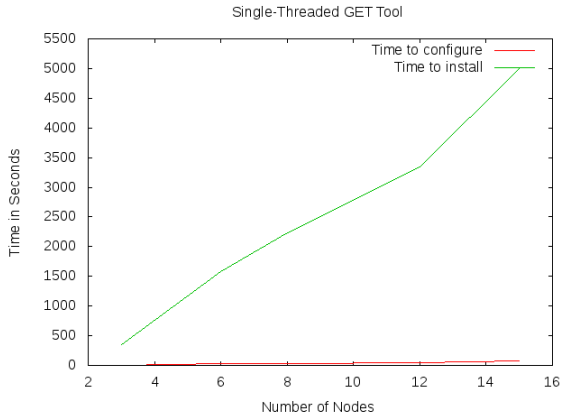

Fig. 5: Single-threaded time neeed to install/compile and configure ChoiceNet over various size GENI topologies using GET

shows the time required when multi-threaded installation is used while Figure 5 show the time required when single-threaded installation is used. In both cases the overall setup time is dominated by the software installation and compilation phase, with the configuration phase taking very little time (i.e., installation and compilation takes tens of minutes, while configuration takes only a few seconds). Not surprisingly, the time required for the single-threaded case grows linearly as the number of nodes increases, while the time required for the multi-threaded case only grows a little since the operations are carried out in parallel. As noted earlier, the main reason for single-threaded execution occurs when all nodes are allocated from the same rack causing the nodes to compete for resources on the rack when run in multi-threaded mode. In that case, single-threading removes the competition and associated multi-threading overhead.

VII. Conclusion

In this paper, we introduced the GENI Experimenter Tool, which is specifically designed and tailored to perform tasks

commonly needed by experimenters. GET is useful for experimenters to configure nodes, place and install node-specific code and modules as well as executing custom commands on a set of nodes and collecting their outputs. In addition, GET has the capability of carrying out tasks on multiple nodes in parallel and thus completes time-consuming tasks such as installing a large module (e.g. Click router) on all experiment nodes rapidly. Overall, GET has proven to be very useful for our future Internet architecture and protocols experiments on GENI and it is customizable to be used by other experimenters from different areas.

REFERENCES

[1] G. P. Office, "Global Environment for Network Innovations - System Overview," 2008. [Online]. Available: http://www.cra.org/ccc/files/docs/GENISysOvrvw092908.pdf

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet Impasse through Virtualization," vol. 38, no. 4, 2005, pp. 33–41, also appears as GENI Design Document GDD-05-01.

[3] D. Clark, S. Shenker, and A. Falk, "GENI Research Plan," April 2007, GDD-06-28.

[4] "National Science Foundation NeTS FIND (Future INternet Design) initiative." [Online]. Available: http://www.nets-find.net/

[5] "National Science Foundation Future Internet Architecture (FIA) project." [Online]. Available: http://www.nets-fia.net/

[6] "National Science Foundation Future Internet Architectures - Next Phase (FIA-NP)." [Online]. Available: http://www.nsf.gov/pubs/2013/nsf13538/nsf13538.htm

[7] D. Han, A. An, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "XIA: Efficient Support for Evolvable Internetworking," in *in Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2012.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking Named Content," in *In CoNEXT 09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, 2009, pp. 1–12.

[9] "XIA Demo at the 15th Geni Engineering Conference (GEC)." [Online]. Available: http://www.cs.cmu.edu/~./xia/Events-News/Events-News/gec15.html

[10] "NDN Demo at the 13th Geni Engineering Conference (GEC)." [Online]. Available: http://groups.geni.net/geni/wiki/GEC13Agenda/GENIUpdates

[11] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldine, and A. Nagurney, "Choice as a Principle in Network Architecture," in *Proc. of ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, August 2012, pp. 105–106, (Poster).

[12] T. Wolf, J. Griffioen, K. Calvert, R. Dutta, G. Rouskas, I. Baldine, and A. Nagurney, "ChoiceNet: Toward an Economy Plane for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 44, 2014.

[13] X. Chen, T. Wolf, J. Griffioen, O. Ascigil, R. Dutta, G. Rouskas, S. Bhat, and K. Calvert, "Design and Implementation of an Economy Plane for the Internet," under review.

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and F. Kaashoek, "The Click Modular Router," *TOCS*, vol. 18, no. 3, pp. 263–297, 2000.

[15] "Flack Tutorial." [Online]. Available: http://www.protogeni.net/ProtoGeni/wiki/FlackTutorial

[16] "GENI Portal." [Online]. Available: http://groups.geni.net/geni/wiki/GEC15Agenda/PortalClearinghouse

[17] "GENI OMNI Tutorial." [Online]. Available: http://trac.gpolab.bbn.com/gcf/wiki/Omni

[18] "GEMINI: A GENI Measurement and Instrumentation Infrastructure." [Online]. Available: http://groups.geni.net/geni/wiki/GEMINI

[19] "GIMI: Large-scale GENI Instrumentation and Measurement Infrastructure." [Online]. Available: http://groups.geni.net/geni/wiki/GIMI

[20] "GENI Desktop." [Online]. Available: http://genidesktop.netlab.uky.edu

[21] "Gush Project." [Online]. Available: http://gush.cs.williams.edu

[22] "PlanetLab Stork Project." [Online]. Available: http://www.cs.arizona.edu/stork/

[23] "Raven Project." [Online]. Available: http://groups.geni.net/geni/wiki/ProvisioningService/

[24] "ExoGENI." [Online]. Available: https://wiki.exogeni.net/

[25] "jFed: A Java-based Framework to support SFA Testbed Federation Client Tools." [Online]. Available: http://jfed.iminds.be/