

The GENI Test Automation Framework for New Protocol Development

Erik Golen^(✉), Sai Varun Prasanth, Shashank Rudroju, and Nirmala Shenoy

Rochester Institute of Technology, Rochester, NY 14623, USA
efgics@rit.edu

Abstract. The National Science Foundation's GENI testbed provides an open infrastructure environment for researchers to develop networking protocols from the ground up. During implementation and testing of a networking protocol, researchers typically begin with small scale topologies to show initial proof of concept before moving on to larger scale topologies. For small scale topologies, it is feasible to manually deploy, compile, and execute protocol code and collect performance metrics. However, when testing with topologies of tens of nodes or larger, the manually intensive task of code deployment, compilation, execution, and metrics collection becomes infeasible. To combat this issue, we present an automation framework for the GENI testbed that requires very little user interaction and utilizes existing GENI APIs and constructs, including the manifest RSPEC file, to perform automated testing of a networking protocol. In this article, we provide the details of the automation framework and its use in the development and evaluation of a new Layer 2.5 protocol, namely the Multi Node Label Routing protocol. The framework can also be used for collection of performance metrics in an automated fashion.

Keywords: Automation framework · Large networks · New protocol development and evaluation

1 Introduction

Recent research trends indicate the popularity of novel techniques to address continuing networking challenges. To evaluate novel techniques several methodologies are available, including analytical models, simulation, and testbeds. Analytical approaches limit testing the techniques in applied environments, while simulations can be very complex if all the environmental conditions are to be taken into account to reflect real life situations. The GENI testbed provides an ideal platform for the development and testing of new protocols. It also allows for evaluation and comparison of such new protocols with existing protocols.

GENI (Global Environment for Network Innovations) is a Lab as a Service (LaaS) that facilitates a distributed system for research and education. As a federated testbed, GENI provides a wide range of heterogeneous devices that abstracts complex and costly physical network creation and provides a virtual infrastructure to explore networks at scale, thus aiding in cutting-edge network research. GENI affords researchers a highly

flexible development environment through providing a bare metal image and allows users to install custom software on top of it.

However, to implement, test, and evaluate networking protocols, users are required to develop their own custom tools and scripts. For example, when users request computing resources from GENI, a generic RSPEC file with topology information is first created. There exists no method for pre-determining the host names and port numbers of the computing resources that will be granted to the user until a manifest RSPEC file is generated during topology creation time. This file must be parsed by a user generated script if the user would like to perform common protocol evaluation tasks that require iteration over a collection of network nodes, such as code deployment, code execution, or metrics collection.

To reduce the steepness of the GENI learning curve and to ease the transition of researchers from an analytical or simulation approach, we propose the GENI Test Automation Framework (GTAF) that is based upon a typical workflow for network protocol development and evaluation. In this article, we provide the details of GTAF and how it was used to evaluate a new non-IP based protocol that operates at Layer 2.5 on Ubuntu Linux 14.04.

In Sect. 2 we describe some of the challenges faced in IP based routing, which required the development of a new protocol. This section also provides some operational details on a new Layer 2.5 protocol that can provide routing within an Autonomous System using simple labels. Since it is deployed as a Layer 2.5 protocol, it bypasses IP and its routing impacts. In Sect. 3, we provide the details of the Python based GTAF to enable easy development and testing of the new protocol in large networks. Though we show only a 27-node topology in this work, the automation framework can easily be used in large networks of over 100 nodes. In Sect. 4, we describe how the automation framework was used for the testing and development of the Layer 2.5 protocol. Section 5 provides some initial results and Sect. 6 provides conclusions and future enhancements planned for the automation framework.

2 Development and Evaluation of New Protocols

One major challenge faced by current routing paradigms is scalability. Whether a routing protocol uses link state routing (OSPF) or path vector routing (BGP), as the number of networks increase, routing table sizes increase. In turn, operational complexity increases, resulting in potentially unstable routing. The forwarding information base (FIB) in Internet core routers have exceeded 600 k entries, while the routing Information base (RIB) of core routers has exceeded one million entries [1]. As a result, numerous looping packets and high churn rates are common in the Internet today.

Addressing these routing scalability challenges requires rethinking the fundamentals of current routing techniques [2]. When new approaches are discovered and designed, the next challenge is to craft the new approaches such that they have a deployment or transition path. In this context, it is very important that the solutions be demonstrated for their operation, preferably in testbeds that capture the attributes of a real network

and allow for comparison with current solutions. The GENI testbed provides all these unique capabilities.

In this section, we describe this new routing protocol briefly to provide context for how the GENI testbed provided the proper platform to code and demonstrate protocol operations; this is subsequently followed by the description and development of the GENI test automation framework (GTAF) to enable testing the new protocol in large network topologies with up to 100 nodes without the tedium of deploying the code and appropriately configuring each and every node. The GTAF may also be used for collecting performance metrics.

2.1 Multi Node Label Routing (MNLR) Protocol

The MNLR protocol [3] uses commonly existing structures in most networks to derive labels that capture the properties of the structure. These labels also capture the connectivity and relationships existing between sets of routers dedicated for certain operations such as Backbone Routers, Distribution Routers and Access Routers. The relational information in the labels is then used to route and forward packets. To provide an easy and smooth transition path, this protocol was developed as a Layer 2.5 protocol. When not required, MNLR can be turned off and routers would use IP and its routing protocol for routing and forwarding. When required, MNLR can be turned on, and will operate below the IP layer transparently without displacing IP.

MNLR can be deployed in a domain or Autonomous System (AS), as shown in Fig. 1. At the edge of the MNLR domain are IP networks that have not invoked MNLR. When an IP packet has to be delivered between End IP network 1 and End IP network 2 at the source edge MNLR node, the IP packet is encapsulated by MNLR headers (which carry the labels of the source MNLR node and destination MNLR node) and routing operations in the MNLR domain are carried out using only the MNLR label information.

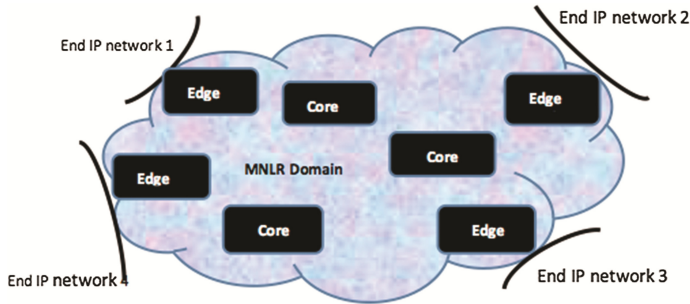


Fig. 1. Implementation of MNLR in IP networks

2.2 Multi Node Label Routing Protocol – Label Significance

Considering a typical AS network construct, MNLR uses a structural abstraction based on router functions, such as backbone (BB) routers, distribution (DB) routers and access

(AC) routers. The BB, DB, and AC router based 3-tier structure is common in many networks. A tiered structure can be noticed among ISPs to define their business relationships, thus in the event MNLR has to be extended for inter-AS routing, the labels can be defined to accordingly capture the tier structure and relationships existing among ISPs. Note, however, that this is out of scope for this article and not further elaborated upon here.

The label notation used in MNLR follows the format TV.UID i.e. Tier-Value.UniqueID. Each value in the label string that makes up a node's UID is separated using ':'. The labels for Tier 1 routers follow the format TV.UID and are denoted as 1.1, 1.2 and 1.3, as seen in Fig. 2.

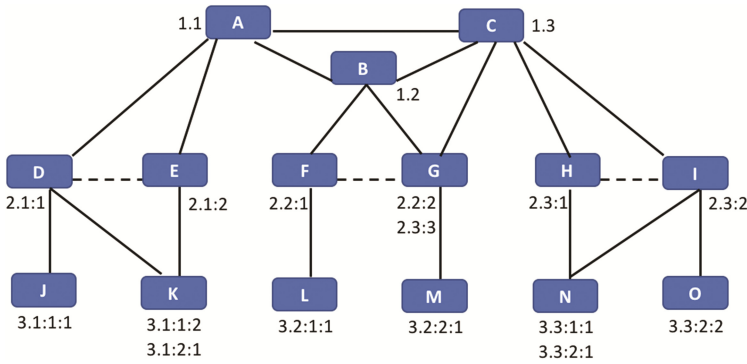


Fig. 2. Label assignment in MNLR protocol

Labels for Tier 2 routers can be allocated administratively, but they have to explicitly show the parent/child relationship to the Tier 1 routers to which they are connected. For example, Router G is a Tier 2 router connected to two Tier 1 routers, B and C, with labels 1.2 and 1.3, respectively. Router G's labels hence have to be 2.2:x and 2.3:y, where x and y have been appended to the UIDs of Routers B and C, respectively. UIDs of Router G (not considering the tier value) are thus 2:x and 3:y, where x and y can be any integer value. In a similar manner, Tier 3 routers are also allocated labels.

Multi Node Label Routing Protocol – Operations and Implementation

One of the configurations required for MNLR operations is the assignment of MNLR labels to all nodes in an MNLR domain. This can be done administratively. The MNLR label format allows for administratively assigning labels to only Tier 1 devices and then enabling auto-assignment from Tier 2 onwards. In this case, MNLR nodes still need to be assigned the tier value of the tier in which they reside.

End IP network addresses and ports may be learned dynamically or configured administratively. There can be multiple end IP networks connected to one edge node. The port information helps in directing the IP packets to the proper end IP network. Edge MNLR nodes populate an IP address to port mapping as shown in Table 1.

Table 1. Local node IP address to port mapping

IP network address	Port
10.10.3.0/24	2
10.10.2.0/24	3
10.10.1.0/24	4

Control Plane Operations

The first MNLR control operation requires the periodic broadcast of ‘hello’ messages by all MNLR nodes to advertise their MNLR labels to their directly connected neighbors. Receiving MNLR nodes populate a neighbor table. Once MNLR nodes have a label, they can start advertising their labels to their neighbors using ‘hello’ messages. The advertisements received by other MNLR nodes can be used for two purposes; (1) to check if a neighbor is still connected and (2) to join any other parent node. The reception and absence of the periodic ‘hello’ messages from neighbor MNLR nodes can inform a parent/child about a missing child/parent.

Table 2 shows the neighbor table at Router G. Port numbers are included in this table, but not shown in Fig. 2.

Table 2. Neighbor table at Router G

MNLR label	Port
1.2	1
1.3	2
3.2:2:1	3
2.2:1	4

End IP Network Address Dissemination

The second MNLR control operation is the end IP network address dissemination to all egress and ingress MNLR nodes. All edge nodes populate a table that maps end IP network addresses to MNLR labels of edge nodes at which they are accessible.

In Fig. 1, the edge MNLR nodes connected to end IP networks should have information about all the end network IP addresses that are connected by the MNLR domain. When an IP packet arrives at an ingress node, the ingress node has to find the label associated with the destination IP address, create the MNLR header with the source node MNLR label and destination node MNLR label, encapsulate the IP packet in the MNLR header, and forward.

Determining the destination node label associated with the destination IP network address requires all edge MNLR nodes to disseminate messages that carry a mapping of their labels to end IP network addresses. The dissemination messages are initiated only by edge nodes and received and stored also only by edge nodes. Core MNLR nodes will forward such advertisements. Let the edge node that has end IP networks as shown in Table 1 have two labels, 2.3:1 and 2.2:1. Another edge node that receives advertisements from this node would populate a table that has a MNLR label to end IP network address table as shown in Table 3.

Table 3. MNLR label to end IP network address mapping

MNLR label	IP network address
2.3:1	10.10.3.0/24
	10.10.2.0/24
	10.10.1.0/24
2.2:1	10.10.3.0/24
	10.10.2.0/24
	10.10.1.0/24

Data Plane Operations

The operations to forward IP packets between end IP networks include encapsulation of incoming IP packets in special MNLR headers at ingress MNLR nodes and de-encapsulating MNLR packets at the egress MNLR nodes to deliver IP packets to the destination IP network. The core nodes forward MNLR encapsulated packets towards egress MNLR nodes using the neighbor table and the forwarding algorithm. Details of the forwarding algorithm are available in [3].

Neighbor Relationships and Packet Forwarding Options

A closer look at the UIDs of routers reveals the existence of several trees rooted at the backbone routers. The backbone routers themselves may be meshed. Each UID that an MNLR router is assigned provides a path to the root of a tree and vice versa. Due to the fact that a node is allowed multiple labels, a router can reside in multiple tree branches and have multiple paths to reach different backbone routers. The UIDs can be used to identify these multiple paths. A simple decision algorithm will allow a node to decide which path to use while forwarding a packet. Also, nodes in a tier, such as Tier 2 or Tier 3, may be mesh-connected or partial mesh connected, as seen in Fig. 2. In this case, the mesh information, if included in the neighbor table, can be used for forwarding packets.

In Fig. 2, Routers J and K have a common parent in Router D. To forward a packet between Routers J and K, Router D can be used. Looking at the labels of Routers J and K which are 3.1:1:1 and 3.1:1:2, gives the information that they have a common parent at Tier 2, which is 2.1:1. Similar string comparison algorithms and techniques can be used to identify a path to forward packets towards a destination node. Similarly, there is a peering relationship between Routers G and F. Based on the exchange of hello messages by its neighbors Router G records all its neighbors as shown in Table 1. (The ports noted in Table 1 are not shown in Fig. 2). This information can be used to forward packets between Routers L and M.

3 GENI Test Automation Framework

Though GENI provides an infrastructure to assist in new protocol development and testing, adopting the infrastructure to perform these tasks can be a complicated and time-consuming process. However, the new protocol development process can be simplified with the help of a test automation framework. Any protocol during its development phase requires frequent code changes and manually deploying code into a topology for testing,

which becomes a time infeasible task as topologies reach as few as six to eight nodes. The primary objective of this GENI Test Automation Framework (GTAF) is to ease the protocol development process in the GENI testbed. It not only helps in expediting this process, but also reduces the chances of hand operated or manual errors. The flowchart with each of the modules used in GTAF is shown in Fig. 3 and are described in the subsequent subsections.

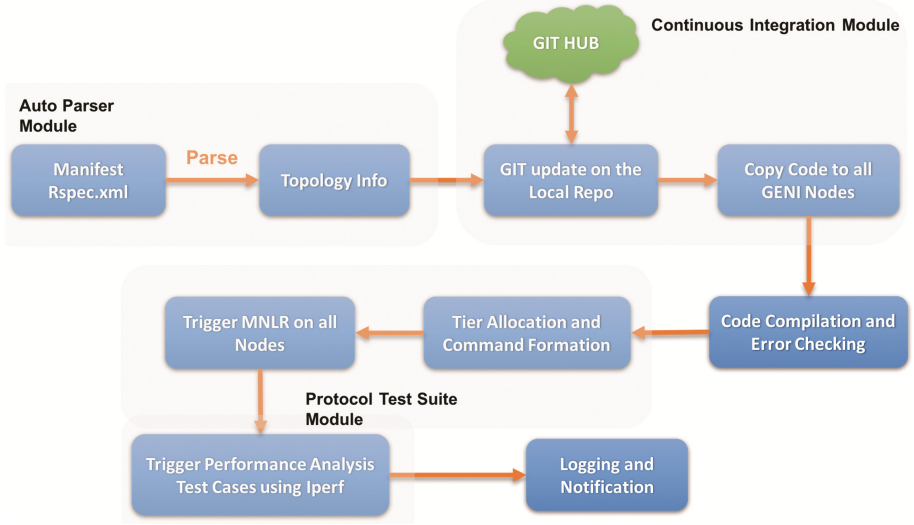


Fig. 3. Flowchart of the GENI test automation framework

3.1 Auto Parser

In order for GENI to allocate resources for a topology, a generic topology is constructed, resulting in an RSPEC file. Once the resources are allocated using one or more aggregates, a manifest RSPEC file is generated that contains all of the hostname and port information specific to the newly available topology. Since GENI does not provide a native parser that allows researchers to iterate over the resources to deploy code or perform other common development tasks, an Auto Parser module was developed with the manifest RSPEC file as input. The Auto Parser extracts all pertinent topology details from the manifest RSPEC, including port numbers, node hostnames, interfaces used, and node connectivity details. Once the parser has completed its operations, the topology information is written into memory and saved locally in the location where the test script was triggered.

3.2 Continuous Integration

Since GENI nodes do not share a common memory space, protocol code has to be individually deployed to every newly spawned node. Software projects, especially large and complex ones, need an infrastructure for continuous integration to a configuration

management server to make sure the latest code is tested and verified. GTAF provides a means for continuous integration of code developed by the developers in their local machine to a central repository. The central repository used in this case is GitHub [4]. It should be noted that without loss of generality, other repositories such as Subversion or Perforce could also be substituted into GTAF. The automation framework ensures that the latest code is pulled from the GitHub and copied to all the GENI nodes for development purposes.

3.3 Code Compilation and Error Checking

Once the code is copied onto all GENI nodes in a topology, the code in the individual nodes is compiled and the framework checks for any errors and notifies the user via an email in case of any errors. The user can immediately take corrective actions and then rerun the framework.

3.4 Protocol Test Suite

There are two steps included in a protocol test suite. First, execution commands are formed dynamically by the framework, which is required for executing the protocol code. Execution commands include dynamically generating any input parameters required to run the protocol code. Of course, these execution commands may vary for each node based on its interfaces and IP addresses allocated by GENI. Once protocol code has been confirmed as executing, performance metric generation scripts are automatically executed on the respective nodes by the framework. These scripts may vary in scope from running iPerf [5] to custom metrics collection scripts.

3.5 Logging and Notification

GTAf ensures that all of these processes noted above are executed in their proper sequence. If any of the processes fails during framework runtime, the user is notified with an email including the reason for any failures.

4 Applying GTAF to MNLr

MNLr was proposed for operation in large scale networks. Due to the tedium involved in manually configuring such large testbed networks where different nodes may require different configuration needs, it was decided to develop GTAF. Below are some of the steps involved in MNLr testing and evaluation as applied to GTAF.

1. The test topology shown in Fig. 4, below, was created in the GENI testbed. From this, the manifest RSPEC file is generated once resources were requested.

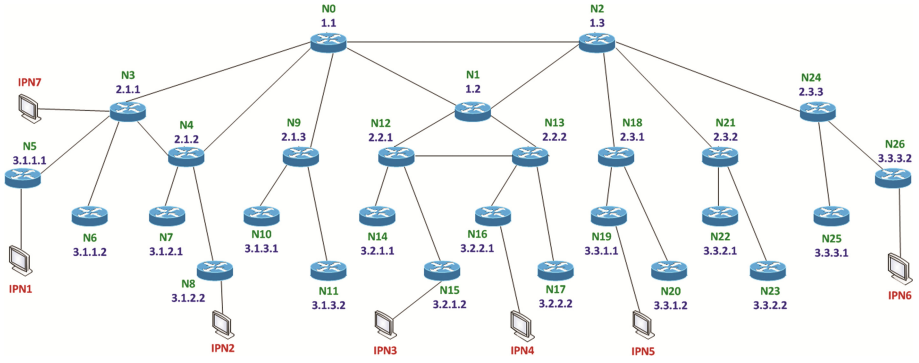


Fig. 4. 27 node topology configured using the automation framework to run MNLR

2. The manifest RSPEC (.xml) file containing the topology information is provided as input to GTAF.
3. The Auto Parser extracts all node information and saves it in a dictionary for later usage. The dictionary uses the node names as a key and their corresponding information as its values. An example is provided below:
u'node-5': ('pc4.instageni.ku.gpeni.net', '34368', u'interface-8', u'interface-23', u'10.10.5.1', u'10.10.12.2')
 In this example, node-5 is the name of the node, 'pc4.instageni.ku.gpeni.net' is the corresponding hostname, and 34368 is the port number. This node has two interfaces, 8 and 23, whose corresponding interface IP addresses were assigned as 10.10.5.1 and 10.10.12.2.
4. GTAF copies the latest updated MNLR code from its GitHub repository to all the nodes in the topology beginning with 'N' since those are MNLR nodes. 'IP' nodes are IP edge nodes and do not require MNLR code. The Continuous Integration module internally uses SCP (file transfer utility) to take care of the process.
5. The Compilation and Error Checking module compiles the MNLR code on all MNLR nodes and checks for any errors. Simple errors, including stray or duplicate files are handled by the framework. If the system is unable to fix the errors, it emails the user with the appropriate error information.
6. The MNLR protocol requires input command line parameters to execute the code. For example, `"/run -T 1.1 -N 1"` triggers the code in one of the nodes. In this case, the label is '1.1' and '-N' denotes the type of node, which indicates a core node. An example for an edge node command is `/run -T 2.1.1 -N 0 10.10.13.2 24 eth1`. In this example, **2.1.1** is the label, and the type of the node is **'0'**, which denotes that the node is an edge node, thus requiring two additional parameters - the IP address and the interface. All of these commands are generated and executed by the Protocol Test Suite module with the aid of a user input file containing all label assignments. Future work includes auto label assignment for Tier 2 and below.
7. Once the topology is stable, the Protocol Test Suite module triggers a series of tests executed in sequential order and consolidates the final status of each test that ran.

The parser will parse through the logs and email the final status and results of the tests to the user.

5 Experiments and Results

As mentioned, Fig. 4 shows an example topology used for MNLR development. It includes 27 nodes running MNLR code and 7 IP nodes that emulate edge IP networks. GTAF deployed, compiled, and executed the MNLR code and ran performance metric collection scripts.

Edge MNLR nodes required special configuration to record the edge IP networks and ports in a table and also the edge MNLR label to IP address mapping, all of which was done using GTAF. A similar topology was created with BGP running in all of the nodes using the Quagga Routing Suite version 0.99.21 [6].

The performance metrics collected were churn rate and routing table sizes. Future GTAF performance metrics collection scripts are planned. Churn rate is the percentage of nodes that updated their routing tables in the event of link or node failure. In the topology in Fig. 4, the link between Node 0 and Node 1 was broken and the resultant churn rate was calculated for both protocols. The routing table sizes and churn rate upon the denoted link failure for BGP and MNLR are shown in Table 4.

Table 4. Performance Metrics BGP vs. MNLR

Metric	BGP	MNLR
Routing table size	29	4 (max)
Churn rate	18/27 (67.7%)	4/27 (14.8%)

6 Conclusions and Future Work

The MNLR protocol discussed in this article is to be demonstrated in large networks with more than 100 routers, which is a typical size in an Autonomous System (AS). When such large networks are created in GENI and demonstrated, the initial effort to setup and configure all the nodes can be very time consuming. The GENI Test Automation Framework presented in this article was written in Python and was used for setting up the MNLR topology for a 27 MNLR node topology and collecting performance metrics.

In all, GTAF took approximately 10 min to deploy, compile, and execute MNLR code in the 27 MNLR nodes in the topology and collect data for two performance metrics. If these tasks were done manually, several hours would have been spent and the potential for operator error would have been quite high.

Future work includes evaluating GTAF for larger network sizes as MNLR continues its development. This article provides insights into the use of such automation tools that can be used by GENI users to setup their test topologies faster. GTAF can eventually be used by all GENI users and will be made available upon project completion. In addition, a larger collection of performance metrics collection scripts will be added,

including end to end latency in delivering IP packets and calculating the number of lost packets upon link or node failure, among others.

References

1. BGP Analysis Report. <http://bgp.potaroo.net/as6447/>. Accessed 10 Jan 2015
2. Nozaki, Y., Tuncer, H., Shenoy, N.: A tiered addressing scheme based on floating cloud internetworking model. In: 12th International Conference on Distributed Computing and Networking, IEEE Sponsored ICDCN Conference Bangalore, January 2011
3. Kethe, P., Golen, E.F., Ragila, S., Shenoy, N.: Modularity in routing – a multi-node label routing protocol. In: IEEE International Conference on High Performance Switching and Routing, 14–17 June 2016
4. GitHub. <http://www.github.com>
5. iPerf – The Ultimate Speed Test Tool for UDP, BGP, and SCTP. <http://www.iperf.fr/>
6. Quagga Routing Suite. <http://www.nongnu.org/quagga>

Future Network Systems and Security

Third International Conference, FNSS 2017, Gainesville,

FL, USA, August 31 - September 2, 2017, Proceedings

Doss, R.; Piramuthu, S.; Zhou, W. (Eds.)

2017, X, 199 p. 60 illus., Softcover

ISBN: 978-3-319-65547-5